UTILIZAÇÃO DE ARTE CONCEITUAL PARA GERAÇÃO DE MODELOS 3D

RENATO DIAS VIANA

UTILIZAÇÃO DE ARTE CONCEITUAL PARA GERAÇÃO DE MODELOS 3D

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Orientador: Renato Antonio Celso Ferreira

Belo Horizonte

Março de 2014

Viana, Renato Dias

V614u Utilização de arte conceitual para geração de modelos 3D / Renato Dias Viana. — Belo Horizonte, 2014

xxiv, 54 f. : il. ; 29cm

Dissertação (mestrado) — Universidade Federal de Minas Gerais

Orientador: Renato Antonio Celso Ferreira

1. Computação — Dissertação. 2. Computação Gráfica — Dissertação. I. Orientador. II. Título.

CDU 519.6*83.043

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

Utilização de Arte Conceitual Para Geração de Modelos 3D

RENATO DIAS VIANA

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

PROF. RENATO ANTÔNIO CELSO FERREIRA - Orientador
Departamento de Ciência da Computação - UFMG

PROF. ERICKSON RANGEL DO NASCIMENTO - Coorientador
Departamento de Ciência da Computação - UFMG

PROF. LUIZ CHAIMOWICZ
Departamento de Ciência da Computação - UFMG

PROF. FRANCISCO CARLOS DE CARVALHO MARINHO
Departamento de Fotografia, Cinema e Teatro - UFMG

Belo Horizonte, 14 de março de 2014.

Dedicuum cest laborae a quelquis personatum que ajudorat a facirelo.

Agradecimentos

Ao terminar este projeto que me custou muitos dias de trabalho e pesquisa, lembro quando decidi fazer o Mestrado em Ciência da Computação. Não tinha ideia de como seria voltar a estudar, depois de algum tempo somente trabalhando, tão pouco imaginava o que seria meu trabalho de pesquisa. Pensei que eu conseguiria terminar o Mestrado trabalhando quarenta horas semanais, apesar de todas advertências de amigos que passaram por experiência.

Os dias foram passando, algumas dificuldades vieram, então percebi que estava errado. Não conseguiria me formar trabalhando tantas horas diárias. E então, tive que abandonar meu trabalho, a fim de me dedicar exclusivamente ao estudo. Sem o apoio das pessoas ao meu redor, isso não seria possível. Por isso, tenho profunda gradidão por todos que de alguma forma colaboraram com isso.

Meu maior agradecimento é para o Deus Pai e para o Senhor Jesus, que com seu amor, me alcançou e me deu forças quando me sentia fraco, me dando a paz que todas as pessoas precisam. Sou muito grato também à minha esposa, Fernanda, que eu amo tanto. Fê, sem você ao meu lado nada disso seria possível, você é minha inspiração. Amo você! Obrigado por estar cada dia ao meu lado e acreditar em mim mais do que eu mesmo.

Agradeço, também, a toda minha família em especial a minha mãe, Elmi, meu irmão, Bell, meu sobrinho, meu sogro Renato, minha sogra Cecília, meus cunhados Marcela e Pedro, minha vó Tereza, tios, tias e primos, pelas orações, carinho e apoio. Amo vocês! Também agradeço a todos os irmãos da igreja, porque sem a oração e o cuidado da igreja não teria chegado até aqui.

Agradeço também, ao meu orientador o Prof. Renato Ferreira e ao Prof. Erickson, que me ajudaram e direcionaram meus estudos. Sem ajuda de vocês eu teria tido muito mais dificuldades e teria seguido caminhos que, provavelmente, seriam mais difíceis em algumas situações.

A todos vocês, o meu muito obrigado!

"Escondi a tua palavra em meu coração, para não pecar contra ti."
(Salmo 19:11)

Resumo

Este trabalho apresenta uma ferramenta que automatiza o processo de criação de modelos 3D a partir de desenhos conceituais. Na abordagem tradicional para a criação de modelos 3D, um artista 2D desenvolve a arte conceitual dos objetos e, posteriormente, um modelador 3D cria o modelo manualmente utilizando esses esboços. Apesar da qualidade dos objetos gerados nesse processo essa é uma técnica lenta e onerosa.

Neste projeto foi desenvolvida uma aplicação que recebe como entrada as imagens da arte conceitual com visualizações de um objeto em diferentes pontos de vistas, gerando como saída o modelo 3D. O aplicativo utiliza um processo que através de técnicas derivadas dos algoritmos *Visual Hull* e *Marching Intersects* é capaz de gerar o modelo 3D automaticamente.

Visual Hull e *Marching Intersects* são algoritmos que processam as diferente visões de um mesmo objeto e projetam em uma estrutura de voxels, a aproximação tridimensional do objeto. A partir dessas informações é possível extrair o modelo 3D utilizando o algoritmo *Marching Cubes*.

A solução desenvolvida possui aprimoramentos para se extrair modelos 3D mais precisos, já que, no processo de geração de modelos 3D foi incluída uma etapa de suavização da malha e outra de redução poligonal. Os resultados alcançados mostram que a aplicacação é capaz de gerar modelos precisos a partir de esboços bastante simplificados.

Palavras-chave: Sketching, Arte Conceitual, Visual Hull, Voxel, Marching Intersects, Marching Cubes.

Abstract

This document presents a set of tools that automates the creation of 3D models from conceptual sketches. The traditional approach to 3D model creation consists of a 2D artist developing conceptual art of the objects, and later, a 3D modeller creates the model manually based upon these sketches. Despite the quality of the model generated by this process, it's slow and costly.

We have developed a solution in this project that receives the conceptual art of an object in different visualizations as input, and generate the model 3D as output. The application uses a process derived from algorithms like Visual Hull and Marching Intersects to generate the model automatically.

Visual Hull and Marching Intersects are algorithms that analyze the different cameras of the same object and generate an approximation of the object in a voxel structure. Processing this information it's possible to generate the model 3D using an algorithm called Marching Cubes.

The solution developed has improvements to extract more accurate 3D models, since in the generation of 3D models process was included a step of smoothing the mesh and other of polygon reduction. The result shows that the application is capable to generate accurate from very simplified sketches.

Keywords: Graphic Computer, Sketching, Visual Hull, Voxel, Marching Intersects.

Lista de Figuras

Lista de Tabelas

Sumário

Capítulo 1

Introdução

Tarefas com alto grau de complexidade, tais como a criação de um cenário virtual ou objetos tridimensionais ricos em detalhes geométricos e visuais, normalmente são executadas criando um rascunho da ideia geral daquele cenário ou objeto. Esse rascunho provê um modelo simplificado, por exemplo, uma versão em duas dimensões de um objeto, e será utilizado para a criação de uma versão final mais detalhada, como a geração de um modelo tridimensional do mesmo objeto.

A técnica de criação de um modelo simplificado contendo os traços mais significantes de um objeto é chamada de *Sketching*. Essa técnica é largamente utilizada em diversas aplicações, como por exemplo na criação de desenhos conceituais, esboços de paisagens e na criação de cenários e objetos tridimensionais.

Normalmente, no processo de criação de modelos trimidimensionais utilizados na indústria de entretenimento digital, um artista 2D esboça o desenho em vários ângulos de visualização de um personagem ou objeto em um programa gráfico de desenho como o Photoshop Adobe Systems Inc. [1989]. Esse tipo de desenho é o chamado *Concept Art* (arte conceitual) de um determinado objeto. Após o artista finalizar a criação do desenho conceitual, um segundo artista, o modelador 3D, utilizando ferramentas tais como: Maya Alias Systems Inc. [2001], 3D Studio Max Autodesk Inc. [2001] ou Blender Foundation [2006], gera um modelo em três dimensões baseado no desenho conceitual. Assim como na geração do desenho conceitual, o modelo 3D é criado manualmente. A Figura 1.1 ilustra as 4 visualizações diferentes 2D de um pequeno avião e o modelo 3D produzido a partir delas.

Essa metodologia além de demandar longo tempo de trabalho, possui custos elevados para a indústria de entretenimento, pois requer ao menos dois artistas altamente especializados (um artista para criar a arte conceitual e o modelador 3D). Por meio de uma ferramenta de sketching que seja capaz de analisar os esboços de desenhos criados

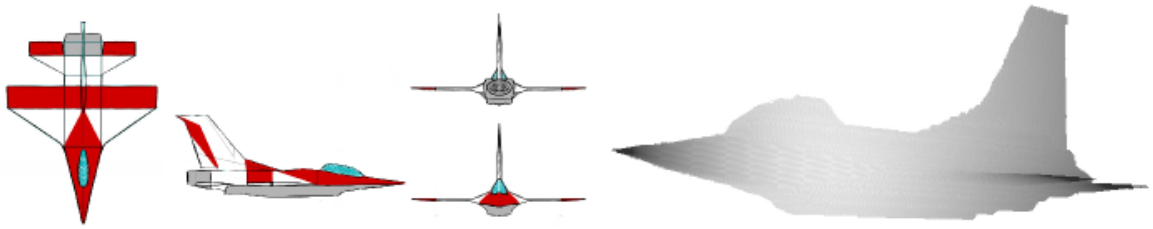


Figura 1.1. Desenhos conceituais de um avião à esquerda e seu modelo 3D corresponde à direita.

pelo artista 2D, pode-se simplificar drasticamente a segunda etapa da modelagem do objeto em 3D, tornando possível, assim, a criação automática pelo computador. Isso além de reduzir o custo torna o processo mais rápido. O modelador poderia importar a malha 3D gerada a partir dessa ferramenta de sketching e fazer apenas ajustes manuais, retirando ou colocando polígonos em locais pontuais, para alcançar o resultado desejado. As empresas poderiam, dessa forma, alocar na maior parte do tempo os modeladores 3D em atividades como animação ou montagem de cenários, gerando assim maior produtividade.

Este projeto de pesquisa foi motivado pelo convívio diário com artistas no desenvolvimento de jogos digitais, onde foi observada a dificuldade para se obter um modelo 3D. Isso ocorre devido o longo processo necessário para obter o resultado da malha de triângulos. Esse processo possui custo elevado no orçamento de projetos nas empresas, pois consome longo tempo de desenvolvimento. Sendo assim, acredita-se que, por meio da técnica de processamento da arte conceitual, pode-se produzir modelos 3D automaticamente por algoritmos, reduzindo tanto o custo quanto o prazo referente ao desenvolvimento de projetos de entretenimento digital.

Neste trabalho, é proposta uma nova técnica para gerar modelos 3D a partir de esboços simplificados em duas dimensões. A metodologia recebe como entrada as diferentes visualizações do objeto (esquerda, direita, frente), executa alguns algoritmos (detalhados nos capítulos posteriores) e gera como saída o modelo em 3D do objeto. Esse modelo pode ser exportado em algum formato de arquivo como o *Collada* de propriedade da G. Khronos [2004], e posteriormente pode ser ajustado manualmente pelo modelador 3D, para se obter o resultado final esperado.

1.1 Contribuições

Este projeto de pesquisa apresenta uma solução capaz para gerar modelos 3D a partir de artes conceituais de objetos. Para gerar um modelo 3D a partir das visualizações do objeto, é apresentado como contribuição, a definição de um processo que utiliza algoritmos comuns na literatura.

Primeiramente, é necessário extrair informações volumétricas referente ao objeto. Este projeto apresenta o trabalho Cignoni et al. [2001] cujo algoritmo é chamado de *Marching Intersects*, que é uma melhoria da solução clássica *Visual Hull* apresentado em Matusik et al. [2000] e é capaz de gerar informações volumétricas mais precisas dos objetos. Como contribuição é apresentada uma modificação no algoritmo *Marching Intersects* que simplifica a implementação.

Após obtidas as informações volumétricas, é apresentado o algoritmo clássico de extração da borda (malha 3D de polígonos) como mostra o trabalho Lorensen & E. [1987] chamado de *Marching Cubes* .

Como contribuição, este trabalho apresenta uma solução de suavização da malha gerada com o objetivo de remover áreas pontiagudas e serrilhadas. Posteriormente, é apresentada outra contribuição na qual é modificado um algoritmo de *Level of Details* para se obter uma solução de redução do número de polígonos da malha sem perder a qualidade da mesma.

1.2 Resumo dos capítulos

O Capítulo 2 apresenta o conhecimento de background para dissertar sobre o tema, além disso, são apresentadas diferentes soluções para o problema mostrando as diferenças dessas soluções em relação a este projeto pesquisa.

O Capítulo 3 aborda a solução desenvolvida nesse trabalho de pesquisa, apresentando todas as etapas necessárias para geração do modelo 3D através da arte conceitual do objeto.

O Capítulo 4 apresenta a solução proposta para melhorar o resultado da malha gerada, através de um processo de suavização da malha e outro de redução poligonal.

O Capítulo 5 apresenta os experimentos realizados, assim como, os resultados alcançados.

Finalmente, o Capítulo 6, apresenta o fechamento do trabalho, apresentando também as limitações da abordagem utilizada. Também são apresentadas nesse capítulo propostas de trabalhos futuros.

Capítulo 2

Revisão bibliográfica

Este capítulo apresenta a revisão bibliográfica da literatura utilizada como base para este trabalho de pesquisa. Primeiramente, é mostrado o conhecimento básico necessário para dar prosseguimento à leitura do restante do texto. Posteriormente são apresentados trabalhos relacionados, de geração de modelos 3D, contrastando com a metodologia proposta neste trabalho.

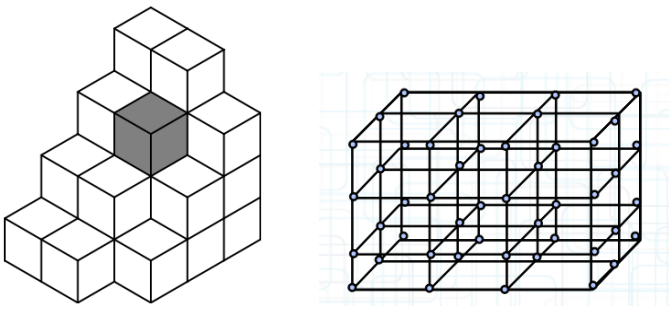
2.1 Background

A dissertação de pesquisa apresenta uma nova metodologia capaz de simplificar o processo de geração de modelos 3D. Utilizando algoritmos comuns na literatura, é prosposto um processo que recebe como entrada as diferentes visualizações de um objeto e gera os modelos 3D automaticamente, sem necessidade de iteração humana.

Os objetos podem ser representados através de duas formas. A primeira forma de representação é através de representação por células no qual o espaço ao redor do objeto é divido em celulas e marcado cada uma dessas células como fazendo parte do objeto ou não. A segunda forma de representação é a representação por borda no qual descrito os contornos da superfície. É possíve definir os contornos por funções implícitas, ou funções paramétricas, ou por tabelas de funções.

Existem algumas formas de representação por célula. As mais comuns são através de pixel e unidades de volume. Nessas formas de representação amazenamos informações de visível ou não visível (verdadeiro ou falso), ou também informações mais detalhadas como cores. A unidade de volume é chamada de *Voxel* como mostra o trabalho Milne et al. [2004].

Voxel é um cubo que representa o volume de uma região no espaço tridimensional. Sendo que verdadeiro representa algo visível, e falso quando não existe nenhum objeto



(a) (b)

Figura 2.1. A Figura (a) ilustra a representação de um voxel no espaço. A Figura (b) apresenta um grid 3D de voxels.

visível dentro do volume. É possível imaginar uma sala; se dividirmos a sala em vários cubos podemos preencher todo o espaço relativo a ela. Nesse exemplo temos que cada cubo é um voxel. A Figura 2.1 (a) mostra a visualização de um voxel marcado no espaço. Quanto menor for o tamanho do voxel melhor será a aproximação do objeto, já que é possível capturar informações com detalhes mais finos.

Sendo assim, é possível perceber que voxel e pixel em sua essência são exatamente a mesma coisa. Porém, o pixel é a representação da informação do objeto em um espaço 2D e o voxel é representação da informação do objeto em um espaço tridimensional. Esse tipo de representação é chamada de representação por amostragem espacial, ou enumeração de células.

Para armazenar as informações de algum objeto complexo em uma imagem é necessário uma matriz de pixels. Em objetos tridimensionais, sejam eles armazenados por meio de uma malha de polígonos ou por voxels é necessário utilizar alguma estrutura capaz armazenar todo o volume em três dimensões. Existem dois tipos básicos de estruturas capazes de armazer informações de voxels, a estrutura uniforme e não uniforme.

Estrutura uniforme é uma estrutura na qual o mundo é dividido em partes iguais; esse tipo de estrutura recebe o nome de *grid*. Um *grid* subdivide o espaço de maneira homogênea em células (em nosso caso voxels) de volumes iguais. A Figura 2.1 (b) apresenta uma estrutura uniforme de voxels.

Estruturas não uniformes são estruturas pela qual o espaço não é divido em tamanhos iguais. A idéia por trás desse tipo de estrutura é subdividir o espaço baseado na necessidade de detalhes para representar uma determinada região. Como exemplo é possível pensar em um campo de futebol, de um gol até o outro. Basicamente existe o piso, não há nada no volume além do piso, mas quando se aproxima das traves e

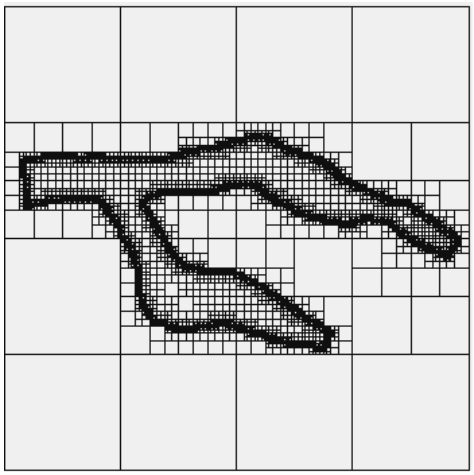


Figura 2.2. Figura ilustrativa que representa uma estrutura de Quadtree.

redes o volume de informações no eixo Y é grande. Se pensarmos em uma estrutura não uniforme, o espaço seria divido em muitas células pequenas próximos aos gols e ao longo do campo uma grande célula seria suficiente para preencher todos os detalhes do volume.

Na literatura existem duas estruturas não uniformes. A primeira estrutura é chamada de *Quadtree*, apresentada por Finkel & Bentley [1974]. Nessa estrutura o espaço é divido em quatro partes somente levando em consideração os eixos X e Z. Sempre que for necessário apresentar informações mais detalhadas em algum quadrante, ele será subdivido novamente em mais quatro partes iguais até que todos os detalhes sejam representados. A Figura 2.2 apresenta um exemplo de uso de estrutura de *Quadtree*.

A outra estrutura comum não uniforme encontrada na literatura é chamada de *Octree*; essa estrutura foi inicialmente proposta por Meagher [1980]. Octree é uma estrutura que divide o espaço em oito cubos, sempre que se precisa detalhar mais o espaço, o cubo é divido novamente por mais oito cubos, até que não seja necessário fazer mais detalhamento do espaço. A maior diferença do *Octree* em relação ao *Quadtree* é que o *Octree* trabalha com o espaço nos três eixos X, Y e Z. Estruturas não uniformes levam como vantagem em relação a estrutura uniforme o consumo mais baixo de memória; porém, elas apresentam maior grau de complexidade na implementação em relação a estrutura uniforme.

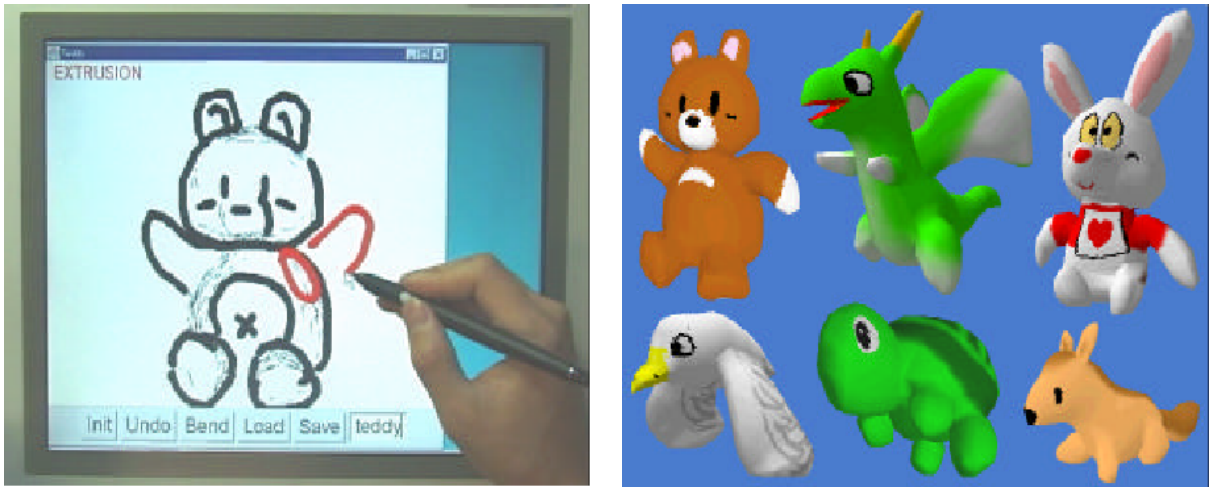
Na representação por bordas armazenamos o contorno do objeto e não informações do objeto em si. Isso pode ser feito de várias formas: por função implícita, por exemplo, onde é definida uma função $F(p) > 0$ se o ponto estiver fora do objeto, $F(p) = 0$ se o ponto está na borda e $F(p) < 0$ se o ponto está dentro do objeto. Mas

definir funções tem uma série de complicações, dado que não é fácil achar funções para formas complexas.

Para resolver esse problema, utiliza-se linearização por partes, nesse caso simplesmente é possível guardar a função dentro de uma espécie de tabela verdade. É a partir dessa idéia que surge a outra forma de representação dada por vértices, arestas e faces. Que nada mais é do que uma linearização por partes, onde as funções matemáticas que representam o objeto são substituídas pelo conjuntos de vértices, arestas e faces.

Como todas as formas de representação tratam informações de um mesmo objeto mas em formas diferentes, então teoricamente seria possível transitar de uma informação para a outra fazendo transformações entre as representações. Este trabalho propõe encontrar um solução por algoritmos capaz de transitar por essas representações, começando por pixels, transformando em volumes e posteriormente em bordas (conjuntos de vértices, arestas e faces).

2.2 Trabalhos relacionados



(a) (b)

Figura 2.3. Utilização da interface do sistema Teddy (a) e alguns exemplos de modelos gerados (b).

Extrair características tridimensionais a partir de imagens 2D para a criação de modelos 3D tem sido uma estratégia amplamente explorada nos últimos anos. Trabalhos como Grimm & Joshi [2012], Olsen et al. [2011], Barequet & Sharir [1994] e Markosian et al. [1999] utilizam imagens geradas por meio de projeção perspectiva, para estimar, por exemplo, o ponto 3D correspondente a um determinado pixel. Todos

esses trabalhos utilizam apenas uma única imagem em perspectiva de um objeto, bem como características como sombras Grimm & Joshi [2012] para a geração do modelo final em três dimensões.

Essa abordagens podem simplificar o trabalho de um artista 2D, pois é necessário fazer somente um desenho 2D para que o modelo 3D seja gerado. Porém, em vários tipos de objetos, tal abordagem se revela imprecisa, visto que vários objetos 3D possuem características que não são perceptíveis por apenas uma visualização. É fácil perceber esse problema utilizando como exemplo alguma máquina que possua de um lado determinados tipos de botões e do outro lado outros tipos de botões. Utilizando essa metologia, o modelo 3D gerado seria impreciso, já que não seria possível diferenciar um lado do outro. Outro problema dessa abordagem é a complexidade matemática; nesses trabalhos, processos de alto custo computacional complexos são necessários para aproximar a morfologia dos objetos. Isso ocorre porque utilizando somente uma única visualização de um objeto não é suficiente para se extrair informações volumétricas desse objeto.

Em Igarashi et al. [1999] é apresentado o sistema Teddy, desenvolvido para gerar modelos 3D baseados em formas 2D criadas pelo usuário. As formas são providas pelo usuário por meio de uma interface de desenho. Essa interface fornece ao usuário ferramentas para criar, utilizando traços 2D, a silhueta do objeto e um conjunto de operações, como criação, extrusão, suavização e corte, para fazer a modelagem do objeto 2D em três dimensões. A aplicação das operações é realizada de maneira iterativa, facilitando o aprendizado de uso do usuário e a correção do modelo 3D. A malha produzida pelo sistema é criada em tempo de execução sendo possível também corrigí-la. A Figura 2.3 (a) demonstra a forma de se utilizar o desenho 2D para a geração dos modelos 3D no Teddy. Na Figura 2.3 (b) são mostrados alguns modelos gerados por meio da interface do Teddy. Esse trabalho também apresenta os mesmo problemas descritos acima por utilizar metologia similar.

Da mesma forma que a softwares de modelagem CAD como Dassualt Systemes Inc. [2009b], Dassualt Systemes Inc. [2009a] e Autodesk Inc. [1982], a metodologia apresentada nesta dissertação propõe trabalhar com até seis visões de um objeto providas por uma câmera ortográfica: frontal, traseira, esquerda, direita, inferior e superior. Contudo, diferente desses softwares de modelagem, as visões não são imagens de um objeto 3D, mas visões diferentes de um mesmo desenho 2D representado em diversos ângulos e desenhados por um artista 2D. A princípio em nenhum momento é necessário fazer qualquer edição no modelo em 3D. A câmera em visão perspectiva tem, tão somente, o objetivo de visualizar o modelo 3D gerado a partir dos desenhos em 2D.

Uma abordagem similar com a proposta deste trabalho foi utilizada no projeto

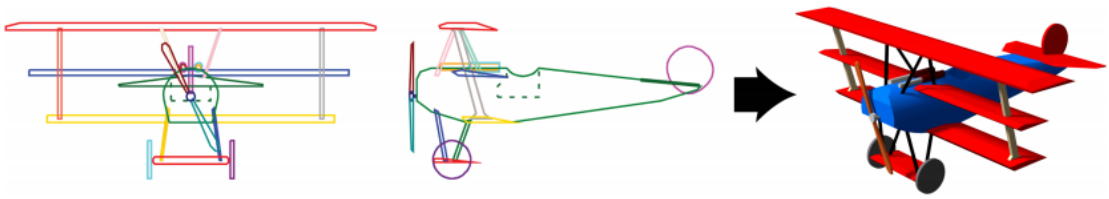


Figura 2.4. Exemplos de objetos 3D gerados por meio da utilização da técnica chamadada Visual Hull e baseada em silhuetas (Imagem extraída do trabalho [Rivers et al. 2010]).

Rivers et al. [2010]. Tal abordagem se sobressai às outras pelo baixo custo computacional e pela capacidade de gerar modelos 3D mais precisos e próximos dos esboços em 2D originais. Por outro lado, uma desvantagem em relação às abordagens que utilizam somente uma visualização perspectiva é a necessidade do artista prover o desenho do objeto em diversos ângulos. No entanto, tendo em vista que o projeto de pesquisa visa a atender a indústria do entretenimento, esse fato não é desvantagem tão significativa, uma vez que os desenhos são usualmente feitos em diversos ângulos de visão antes de serem modelados em 3D. A Figura 2.4 mostra um modelo gerado pela metodologia apresentada em Rivers et al. [2010].

A grande diferença desse trabalho em relação ao trabalho proposto neste projeto é que essa metologia necessita que o artista 2D utilize uma ferramenta própria para edição de desenhos que possuem uma série de limitações de usabilidade. Isso é necessário porque o desenho precisa seguir um padrão definido por essa metodologia para que o modelo 3D seja gerado. Na abordagem proposta neste trabalho de pesquisa o artista 2D pode fazer seus desenhos utilizando qualquer software de edição de imagem disponíveis, uma vez que essas imagens é que vão servir como entrada para o algoritmo.

A Figura 2.5 mostra o software de sketching desenvolvido pelo Google Inc. [2009]. Essa aplicação utiliza técnicas de sketching para fazer alterações em modelo 3D. A abordagem utiliza uma câmera perspectiva que possibilita a visualização da cena e também possui ferramentas capazes de alterar a malha do modelo por meio de desenhos. Essa solução é diferente da solução proposta neste trabalho, pois ela não tem como objetivo gerar modelos em 3D e sim, modificar modelos em 3D utilizando rabiscos como base para edição.

Outro software para geração de modelos 3D por sketching foi desenvolvido pela Mirye Inc. [2010]. Essa abordagem utiliza uma câmera perspectiva, pelo qual o usuário deve marcar os bordas e curvaturas dentro do desenho. Posteriormente, a aplicação utiliza essas informações para gerar o modelo 3D em um processo que envolve etapas

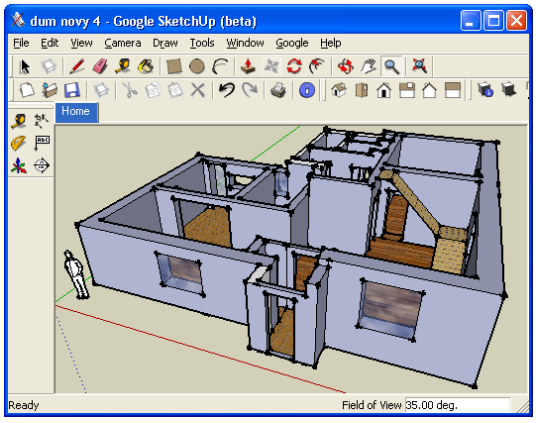
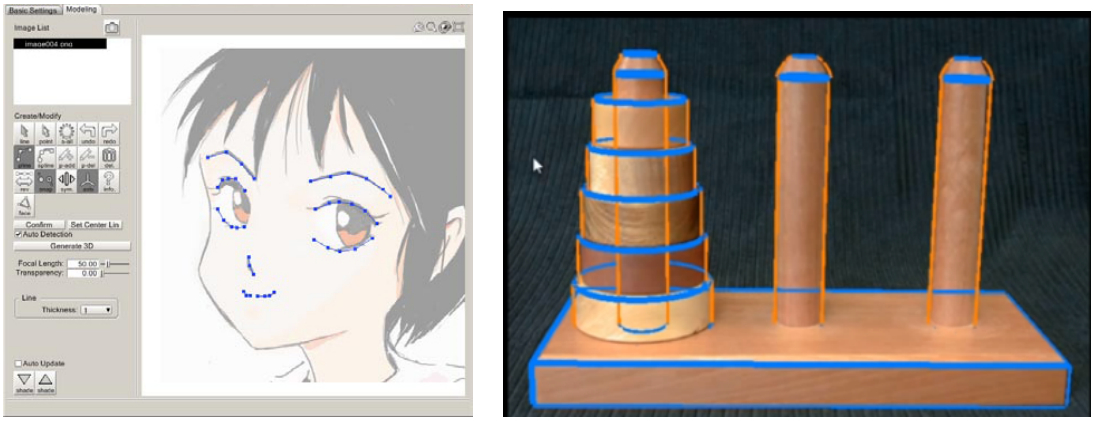


Figura 2.5. Tela do Sketch Up, software para geração de objetos 3D desenvolvido pelo Google.

manuais. A Figura 2.6 (a) mostra a interface utilizada pelo usuário par gerar o modelo 3D.



(a) (b)

Figura 2.6. A imagem (a) mostra tela do Shape 3D, aplicativo responsável por gerar modelos 3D a partir de imagens e a figura (b) apresenta solução semi-automática apresentada no SIGGRAPH Asia 2013.

Similar à solução acima, com aperfeiçoamentos, foi a abordagem Chen et al. [2013] apresentada no SIGGRAPH Asia 2013. Essa solução utiliza uma visualização do objeto para gerar o modelo 3D em um processo semi-automático. O usuário é responsável por definir as curvaturas e tipo da composição do objeto. Após feito isso, o algoritmo detecta a borda e utiliza as informações topológicas do objeto para gerar o modelo 3D. Essa abordagem possui como vantagem a utilização de somente uma visualização do objeto; porém, existe um trabalho manual por parte do modelador 3D. O modelador 3D define a topologia do objeto manualmente como demonstra a imagem 2.6 (b), após

feito isso é executado um algoritmo que gera o modelo em 3D. A grande desvantagem dessa abordagem é que detalhes dos objetos podem em um determidado ponto de vista não ser gerados corretamente; já que somente é utilizada uma visualização do objeto.

Essas duas abordagens semi-automáticas possuem duas desvantagens em relação à proposta deste trabalho de pesquisa. A primeira é a utilização de somente uma visualização perpectiva, isso pode gerar modelos com falta de detalhes devido à diferença de topologia dos objetos em lados diferentes. A segunda desvantagem é que essas abordagens necessitam de um trabalho manual por parte de um modelador 3D para que modelo 3D seja gerado. Porém, esse processo semi-automático tem uma vantagem em relação à nossa abordagem. Como o processo não é totalmente automático, objetos que possuem superfícies côncavas podem ser gerados corretamente apenas ajustando manualmente a morfologia do objeto. A solução proposta neste trabalho de pesquisa possui problemas com superfícies côncavas que serão abordados nos Capítulos 3 e 6.

Apesar de todos os trabalhos apresentados acima apresentam soluções para geração de modelos 3D utilizando visualizações de objetos. Essas abordagens apresentaram soluções satisfatórias, porém, nenhuma delas utilizou o processo atual utilizado na indústria como base para suas soluções. A indústria de entretenimento digital é resistente em relação a grandes mudanças no processo de desenvolvimento, sendo assim, soluções que alterem muito o processo são complicadas de serem implantadas e possuem pouca aceitação. A grande vantagem da metodologia apresentada nesta dissertação está na aplicação prática, já que ela não altera o processo utilizado pela indústria, apenas simplifica e melhora, gerando redução no custo e no prazo dos projetos.

2.3 Sumário

Neste capítulo foi apresentada uma revisão bibliográfica, no qual foram mostradas as formas de representação por célula (pixels e voxels) e reprensentação por borda utilizando vértices, arestas e polígonos. Também foram apresentadas as estruturas principais para representar objetos utilizando voxels. Além disso, foram apresentadas outras ferramentas desenvolvidas em trabalhos relacionados, sendo comparado as soluções propostas nesses trabalho com a metodologia abordada neste trabalho de pesquisa.

No Capítulo 3 será apresentada a metologia proposta neste trabalho de pesquisa para transitar entre as formas de representação. Inicialmente, sendo extraídos os pixels e transformando essa informação em uma estrutura de voxels. Posteriormente, sendo extraída a borda que representa a geometria do objeto em 3D.

Capítulo 3

Metodologia

Neste capítulo é descrito todo o processo de construção do modelo 3D proposto. São apresentados todos os passos necessários para se extrair as informações volumétricas de um objeto através de três ou mais pontos de vistas distintos, e posteriormente exportar a malha de polígonos que representam a borda do objeto na estrutura de voxels.

O aplicativo recebe como entrada as diversas visualizações de um objeto em pontos de vista diferentes, de 3 a 6 visualizações. Utiliza o algoritmo *Visual Hull* apresentado por Laurentine [1994] para fazer a aproximação em 3D da arte conceitual em um grid de voxel. O algoritmo analisa todas a imagens e verifica o que está visível volumetricamente na cena. Depois de executada essa etapa, o aplicativo utiliza uma abordagem para gerar uma aproximação mais precisa do objeto chamada de *Marching Intersects* desenvolvido por Cignoni et al. [2001]; esse algoritmo analisa as bordas da imagem e atualiza os vértices dos voxels para melhorar o resultado da construção das informações de volume.

Após finalizadas as etapas de aproximação 3D do objeto, o aplicativo utiliza uma técnica para gerar a malha triangular chamada de *Marching Cubes* apresentada por Lorensen & E. [1987]. Baseado no estado do voxel esse algoritmo utiliza uma série de opções para criação de triângulos. Ele verifica quais vértices do voxel que estão visíveis na cena e utiliza essa informação para extrair a borda de polígonos do grid de voxel.

Para finalizar o processo, no Capítulo 4 são apresentadas mais duas etapas que entraram como contribuição para melhorar o resultado da malha 3D. A partir do momento em que o aplicativo possui malha 3D, é utilizada uma solução para fazer suavização da malha gerada a fim de obter um modelo mais preciso. Para finalizar, o algoritmo faz redução poligonal através de um algoritmo baseado em uma técnica de LOD (Level of Details), gerando como saída um modelo em 3D com menor quantidade de polígonos.

Cada uma das etapas da metologia utilizada é abordada em uma seção específica,

nas quais são detalhadas o funcionamento de cada um dos algoritmos utilizados, apresentando também suas limitações, vantagens e desvantagens. No final da apresentação de cada uma dessas etapas é incluída a análise de complexidade da solução proposta. A Figura 3.1 apresenta todo o processo descrito acima, mostrando a evolução até o resultado final da solução.

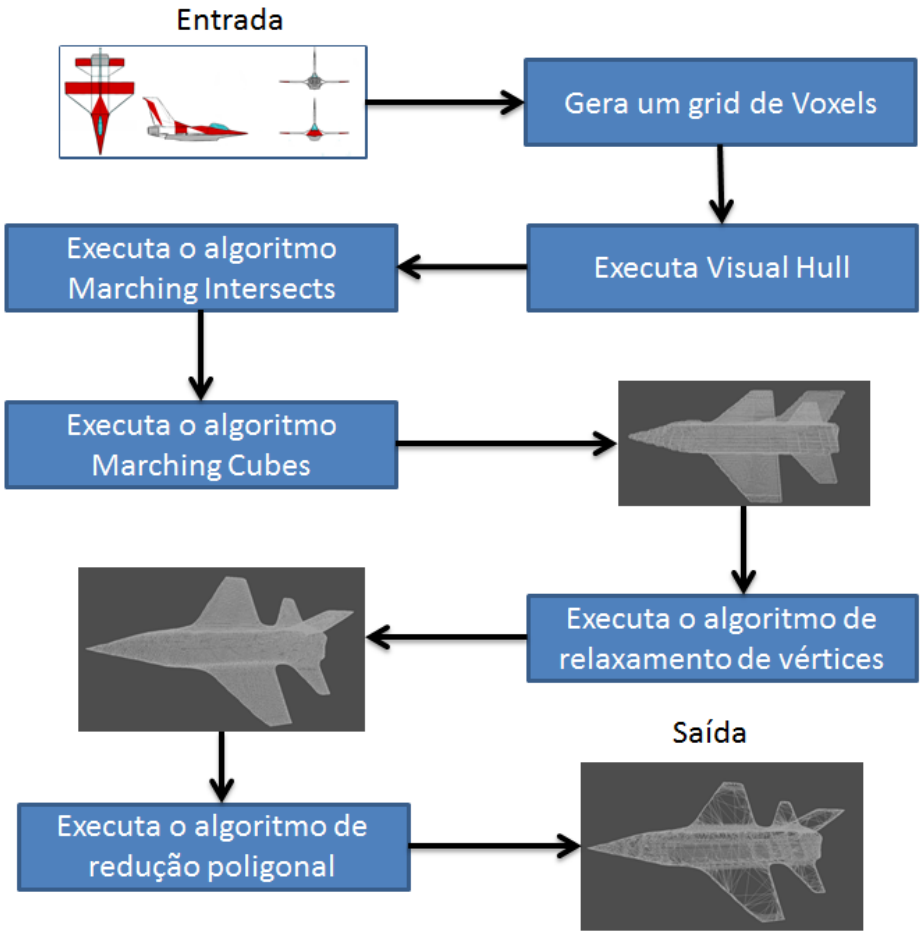


Figura 3.1. Fluxograma com as principais etapas da metodologia. Após determinar a visibilidade e estimar a posição de cada voxel, nossa abordagem cria uma malha de triângulos baseada nos centróides dos voxels, e sua vizinhança suaviza a malha gerada e reduz a quantidade de polígonos necessário para representar o modelo 3D.

3.1 Visual Hull

Para fazer a geração de modelos em 3D de forma automatizada, utilizando arte conceitual, este trabalho usa uma abordagem baseada na estratégia de Visual Hull mostrada por Laurentine [1994] e aperfeiçoada nos trabalhos Matusik et al. [2000] e Lazebnik et al. [2007].

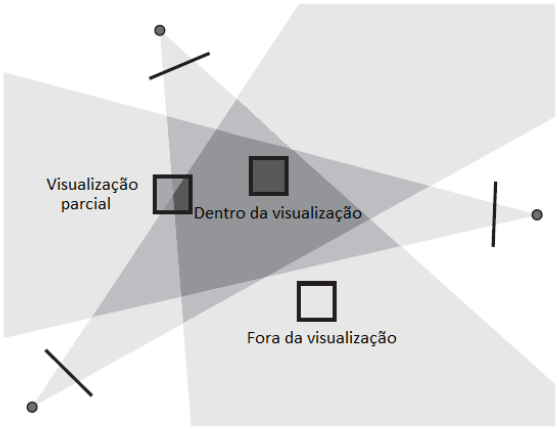


Figura 3.2. Imagem ilustrativa da abordagem de Visual Hull. Utilizando a interseção de cones de visualização, essa abordagem estima a estrutura geométrica de um objeto.

Nessa estratégia são utilizadas duas ou mais visualizações diferentes de um mesmo objeto para reconstruir o objeto em outra visualização. A geração dessa nova visualização é baseada na interseção dos cones de visualização de cada uma das câmeras. A Figura 3.2 ilustra a interseção dos cones de visualização de três câmeras diferentes.

Um dos algoritmos mais tradicionais para recontrução de modelos 3D utilizando imagens de pontos de vista distintos utiliza *Voxels* para implementar uma solução de Visual Hull. Após posicionar as câmeras em locais diferentes, a metodologia constrói o objeto em 3D utilizando as interseções dos cones das câmeras. A Figura 3.3 ilustra essa idéia.

O algoritmo proposto neste trabalho se baseia na utilização de um *grid* de voxels. Outras abordagens como o trabalho Milne et al. [2004] utilizam *Octrees* para representar o espaço. Essa abordagem utiliza menos memória que a abordagem de grids utilizadas nesta dissertação, porém possui maior complexidade de implementação.

A Figura 3.3 apresenta um plano de um grid tridimensional de voxels. Todos os voxels coloridos de roxo representam o objeto. Os voxels coloridos de vermelho representam uma aproximação do modelo real. Para construção do modelo em 3D,

utilizando as imagens de diversas câmeras, o algoritmo processa a imagem de cada câmera separadamente e molda o objeto retirando os voxels que não estão contidos na visualização. Quanto menor as dimensões dos voxels forem utilizados para fazer a aproximação do desenho 2D em um modelo 3D, melhor é a aproximação; porém, com maior custo computacional e maior uso de memória.

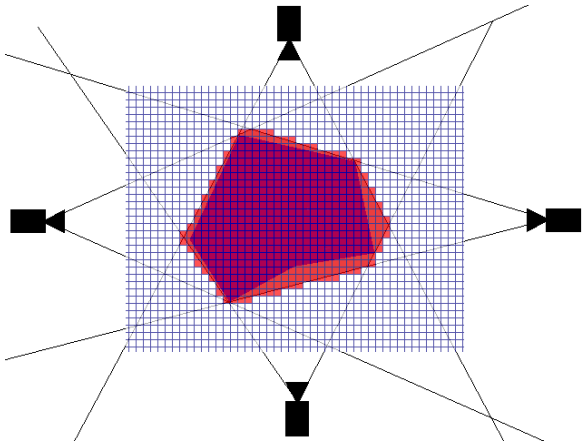


Figura 3.3. Visualização de um plano em 2D da interseção no *grid* de voxels da diversas visualizações de câmeras

Abaixo é apresentado o pseudo-código do algoritmo Visual Hull:

```
cria a grid de voxel com dimensões iguais;
for cada arte conceitual do
    for cada voxel do
        calcula a posição do voxel em relação a visualização;
        if voxel está dentro da visualização then
            marca o voxel como visível;
        else
            marca o voxel como não visível;
        end
    end
end
```

Algoritmo 1: Pseudo-código do algoritmo Visual Hull

Uma das principais vantagens dessa abordagem é o tempo de execução do algoritmo controlado pelo tamanho da grid de voxels. Além disso, ela possui baixa complexidade de implementação, já que não necessita de calculos matemáticos complexos.

Apesar de possuir essas vantagens, a geração de modelos em 3D a partir de desenho em 2D possui alguns problemas como por exemplo a grande quantidade de

polígonos gerados para se conseguir boa definição do modelo. Além disso, não é possível detectar cavidades côncavas nas imagens. Outro problema é que para ser feita a construção das informações de volume do objeto, são necessários desenhos conceituais com resoluções e escalas exatamente iguais.

3.2 Marching Intersects

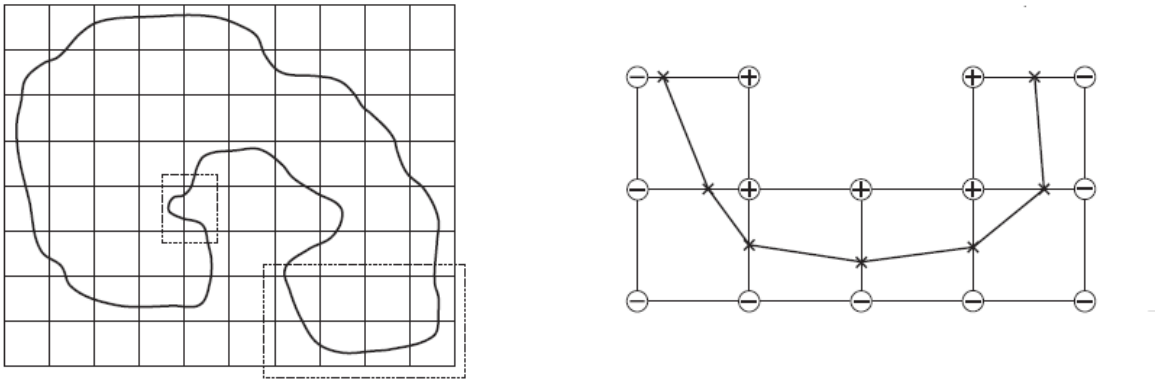


Figura 3.4. A figura acima mostra a silhueta de um desenho projetado sobre um grid de voxels; além disso, mostra os pontos de contatos da borda da imagem com os voxels.

Nossa abordagem utiliza uma variação do algoritmo tradicional de *Visual Hull* chamada de Marching Intersects apresentado por Cignoni et al. [2001] e aperfeiçoado em Rocchini et al. [2004]. Esse algoritmo possui uma solução que melhora a aproximação da reconstrução do modelo em 3D, utilizando a idéia de fazer interseções entre as bordas dos objetos nas imagens, com os voxels.

A ideia principal do algoritmo é percorrer as bordas da imagem e procurar encontrar o local exato onde a borda da imagem colide com a uma superfície do voxel. Esse procedimento é ilustrado na Figura 3.4. Dessa maneira, o algoritmo cria um vértice no local exato da borda e utiliza o mesmo ao invés de utilizar a quina do voxel. Na abordagem utilizada neste trabalho foi feita uma alteração na ideia original do algoritmo. Ao invés de criar novos vértices, são atualizadas as quinas do voxel para posicioná-lo em relação às bordas da imagem, gerando assim uma deformação na estrutura uniforme. Essa alteração simplifica a implementação da solução.

O algoritmo percorre todos os vértices incluídos no grid de voxel, para cada vértice, é verificado se o ele pertence à superfície da malha a ser criada. Para fazer essa

verificação, o algoritmo utiliza a tabela de consulta do algoritmo *Marching Cubes* que está detalhado na próxima seção. Atráves dessa tabela é possível saber se um vértice pertence à superfície e precisa corrigir sua posição em relação a borda das visualizações do objeto. Vértices fora das visualizações ou dentro do modelo são ignorados, pois não são utilizados para criação da malha de polígonos.

Ao encontrar um vértice que compõe a superfície da malha a ser criada, o algoritmo utiliza uma estratégia de força bruta para atualizar a posição exata da colisão da borda das silhuetas em relação ao voxel. O algoritmo direciona o vértice em direção ao primeiro vértice fora da superfície e detecta o local exato da colisão da borda da imagem em relação à borda do voxel. Isso é feito para todos os vértices que compõem a superfície da aproximação. Apesar de produzir resultados melhores que o algoritmo tradicional, essa abordagem ainda mantém alguns dos problemas enumerados anteriormente.

Abaixo é apresentado o pseudo-código do algoritmo Marching Intersects:

```
for para cada vértice X, Y, Z do
    verificar se o vértice pertence a superfície da malha utilizando a tabela do
    Marching Cubes;
    if o vértice está na superfície then
        for cada posição entre o voxel corrente e o primeiro fora da
        visualização do
            for cada visualização do objeto do
                if a posição atual está dentro da visualização then
                    atualiza a posição do voxel;
                else
                    condição de erro, finaliza a execução do laço;
                end
            end
        end
    end
end
```

Algoritmo 2: Pseudo-código do algoritmo Marching Intersects

Para determinar a complexidade do algoritmo é analisada cada etapa do processo. Inicialmente o algoritmo cria o grid de voxel. Se o grid possuir o mesmo tamanho em X, Y e Z é possível dizer que esse algoritmo possui $O(G^3)$, onde o G representa o tamanho do grid.

Posteriormente, é executado o algoritmo de *Visual Hull*, utilizando a abordagem do *Marching Intersects*. Esse algoritmo percorre para cada imagem todos os voxels verificando quais são visíveis na cena. Sendo assim, esse passo possui $O(CG^3)$, onde C

é número de câmeras e G é o tamanho do grid. Para a execução do *Marching Intersects* o algoritmo percorre cada voxel e verifica quais os vértices pertencem à superfície da malha e atualiza as posições desses vértices, para esse algoritmo é executado $O(G^3) + O(v)$ para atualização da posição de cada vértice que compõe a malha.

3.3 Marching Cubes

Após definidos os elementos de volumes (Voxels) que representam o modelo 3D é necessário extrair a malha de polígonos correspondentes. Existem várias abordagens para se extrair a malha de polígonos de uma estrutura de Voxel. A abordagem chamada de *Surface Nets*, apresentada no trabalho Gibson [1998], cria uma rede de ligações entre os voxels que fazem parte da superfície da malha e gera a malha de triângulos já suavizada. Basicamente, o algoritmo adiciona um vértice no centro de cada voxel e faz uma ligação através de uma aresta entre esse vértice e os vértices inseridos nos voxels vizinhos. Após criada essa estrutura são relaxadas as posições dos vértices para suavizar suas posições de forma parecida ao processo de suavização proposto neste trabalho. Essa abordagem apresenta resultados similares a metologia escolhida neste trabalho, mas devido a necessidade da criação da estrutura que liga os voxels vizinhos entre si, ela exige grande quantidade de memória e complexidade de implementação.

Outra abordagem é a Schaefer & Warren [2004]; essa metodologia é uma variação moderna do algoritmo escolhido Lorensen & E. [1987]. Essa abordagem utiliza uma estrutura chamada de Dual Grid e Quadric Error Function para detectar a curvatura da malha em relação ao voxel. Apesar de produzir resultados mais precisos que o algoritmo Marching Cubes tradicional, essa abordagem possui grande complexidade de implementação, sendo assim, não foi escolhida como base para este projeto. Outra abordagem que também utilizado *Quadric Error Function* é o trabalho Ju et al. [2002].

Para criação da malha de polígonos extraída de alguma estrutura de voxels foi utilizado o algoritmo tradicional *Marching Cubes* Lorensen & E. [1987]. Esse algoritmo utiliza uma tabela de opções diferentes para criação de triângulos dentro de um voxel. Para simplicar a explicação é utilizada uma variação 2D do algoritmo chamado de *Marching Square*. O algoritmo *Marching Square* é a implementação do algoritmo *Marching Cubes* porém para duas dimensões. A Figura 3.5 (a) mostra uma silhueta em azul projetada sobre um grid de voxels em duas dimensões. Todos os vértices que compõem o contorno e a parte interna do desenho são marcados de vermelho e rosa. O algoritmo *Marching Square* utliza esses vértices marcados para criar os polígonos em 2D.

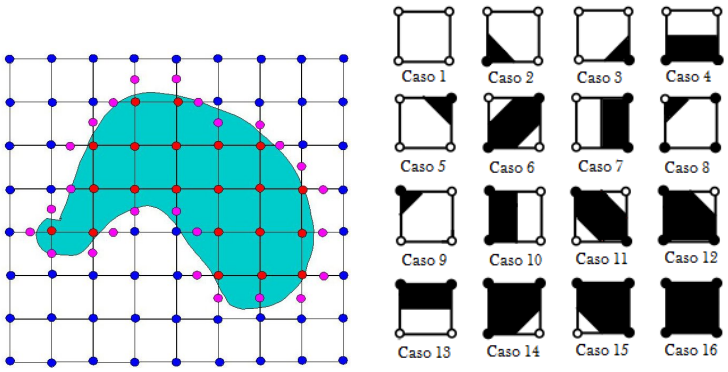


Figura 3.5. A figura (a) mostra a interseção da borda de uma silhueta com o grid de voxels em 2D. Os vértices marcados em vermelhos e rosas fazem parte da aproximação do desenho projetado sobre o voxel e a figura (b) mostra a tabela de opções de polígnos utilizada pelo algoritmo Marching Square.

A Figura 3.5 (b) apresenta as 12 opções diferentes de criação de polígonos para um voxel em 2D. No primeiro caso nenhum vértice do voxel está presente na silheta do desenho; sendo assim, nenhum polígono é criado para esse caso. No segundo caso é visto que somente o vértice baixo à esquerda está presente na silhueta do desenho; sendo assim, o algoritmo cria um polígono entre o vértice esquerdo superior e o vértice direito inferior. Os demais casos seguem o mesmo princípio.

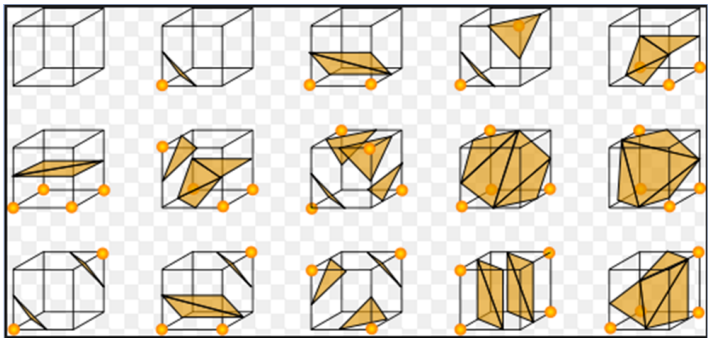


Figura 3.6. As 15 opções bases para criação do polígnos no algoritmo Marching Cubes

O algoritmo *Marching Cubes* utiliza o mesmo princípio da abordagem 2D; porém, faz isso em três dimenssões. São 256 opções diferente de criação de polígonos baseados na configuração do voxel. A Figura 3.6 apresenta as 16 opções bases para criação de polígonos através de um Grid de Voxels. Apenas fazendo rotações com essas variações pode-se contemplar todas as 256 configurações diferentes.

O algoritmo também utiliza essa tabela para verificar quais os vértices que compõem a superfície da malha que será exportada. Para fazer essa verificação, o algoritmo verifica se o vértice é usado para criar algum triângulo dentre as opções disponíveis pela tabela. Caso o vértice seja usado, então esse vértice compõe a superfície da malha a ser exportada. Para cada voxel (X, Y, Z) é verificada na tabela em qual opção esse voxel se encaixa. Caso se encaixe em alguma das opções que necessitam da criação de polígonos, então são adicionados os vértices necessários na lista de triângulos a serem criados. Posteriormente, esse triângulos são criados na ordem de inclusão dos vértices.

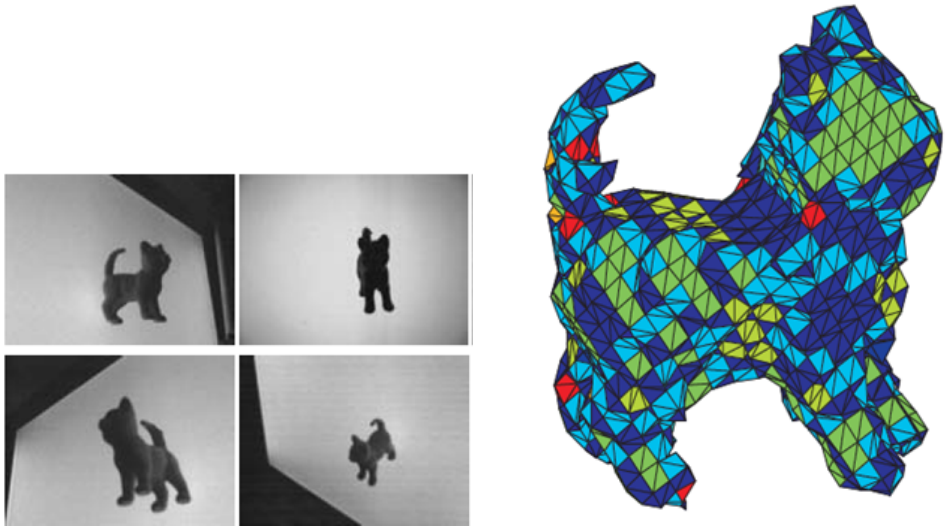
Segue abaixo o pseudo-código do algoritmo de Marching Cubes:

```
for x = 1 até N do
    for y = 1 até N do
        for z = 1 até N do
            verifica na tabela qual o opção o voxel (X, Y, Z) se encaixa;
            if o voxel se encaixe em alguma opção de criação de polígono then
                verifica na tabela quais vértices são necessários para criar os
                polígonos;
                adiciona na lista de triângulos a serem criados;
                for cada grupo de três vértices do
                    cria o triângulo correspondente;
                end
            end
        end
    end
end
```

Algoritmo 3: Pseudo-código do algoritmo extração de malha 3D através de estrutura de voxels.

O algoritmo *Marching Cubes* é utilizado para geração da malha de triângulos. Ele percorre cada voxel e verifica os vértices visíveis, consulta uma tabela e gera os triângulos. Esse algorimto possui ordem de complexidade de $O(N^3)$, já que a consulta na tabela verdade pode ser considerada como $O(1)$.

Apesar de gerar a malha triangular utilizando a estrutura de voxels, essa abordagem gera modelos com bordas pontiagudas e com formação de *alias*. A Figura 3.7 demonstra essas limitações do algoritmo Marching Cubes. Sendo assim, para melhorar o resultado é apresentada neste trabalho uma solução de suavização da malha descrita no próximo capítulo.

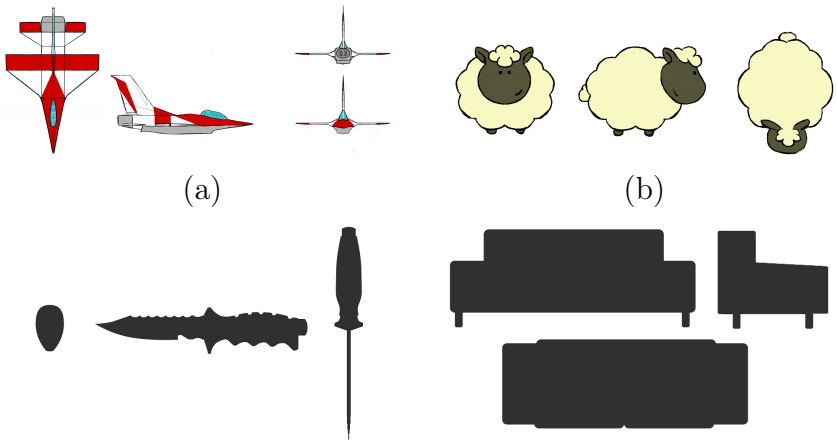


(a) (b)

Figura 3.7. Modelo de um gato gerado a partir de uma solução de Marching Cubes.

3.4 Testes

Para avaliar a metodologia proposta neste trabalho, foram realizados experimentos utilizando artes conceituais de quatro objetos. As artes conceituais das diferentes visualizações dos objetos foram recebidas como entrada, o algoritmo então é executado para serem gerados os modelos 3D. É apresentado também o resultado ao longo de todo o processo proposto, posteriormente é apresentado o tempo de execução dos algoritmos.



(a) (b)

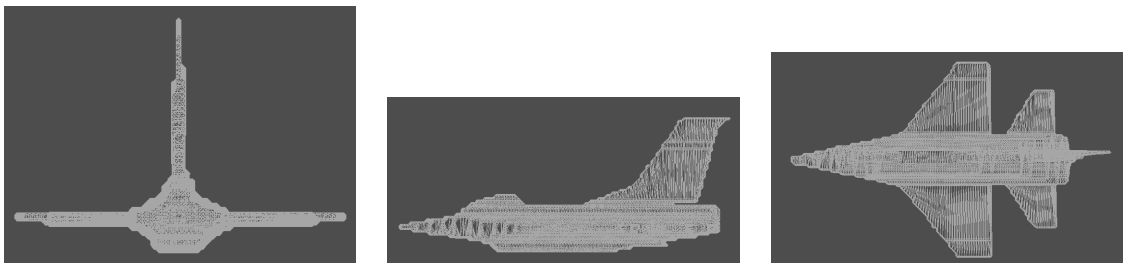
(c) (d)

Figura 3.8. Arte conceitual de um avião, ovelha, faca e sofá utilizados para geração de modelos em 3D.

Foram realizados experimentos utilizando artes conceituais de alguns objetos: um avião com quatro pontos de vistas diferentes, uma ovelha com três pontos de vistas, uma faca com três visualizações e um sofá com 4 visualizações.

Para o sofá é incluída uma arte conceitual com uma máscara especial para gerar o modelo 3D com áreas côncavas. Isso é necessário para gerar esse modelo devido às limitações da solução escolhida que serão detalhadas na próxima seção. As artes conceituais para cada um dos objetos são mostradas nas figuras 3.8 (a), (b), (c) e (d).

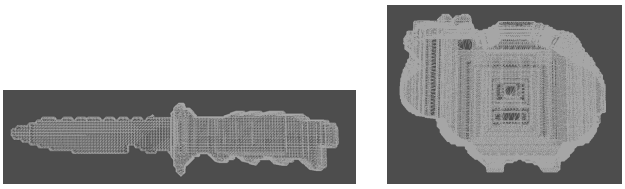
As figuras 3.9 apresentam as diferentes visualizações da aproximação 3D em *Wireframe* gerados a partir da arte conceitual 3.8 (a) do avião, utilizando a metodologia desenvolvida.



(a) (b) (c)

Figura 3.9. Modelo 3D do avião gerado após a execução do algoritmo proposto.

A figuras 3.10 (a) e (b) apresentam outros dois modelos gerados a partir da solução proposta. É possível verificar que algoritmo foi capaz de aproximar o modelos em relação à morfologia do objeto visualizado nas artes conceituais, mesmo para um objeto complexo como um avião; porém, os modelos gerados apresentaram pontas e serrilhados em áreas curvas e diagonais. Para resolver esse problema é proposta uma solução de suavização da malha 3D, que será apresentada no capítulo posterior.



(a) (b)

Figura 3.10. A figura (a) apresenta a faca gerada utilizando a metologia proposta e a figura (b) apresenta a ovelha gerada.

3.5 Limitações

Durante a execução dos testes percebemos que a solução não seria capaz de gerar modelos com superfícies côncavas como um sofá, já que, a área côncava não era possível ser detectada pelas visualizações do objeto em duas dimensões. Para resolver o problema foi aplicada uma segunda máscara no objeto para retirar a área côncava do objeto. A Figura 3.11 (a) mostra que o algoritmo não foi capaz de remover a área côncava da superfície do objeto. A Figura 3.11 (b) apresenta a máscara utilizada para resolver o problema e a Figura 3.11 (c) apresenta o resultado.

(a) (b) (c)

Figura 3.11. A Figura (a) apresenta o sofá gerado pelo metodologia, a Figura (b) apresenta uma máscara aplicada como solução paleativa para resolver o problema das superfícies côncavas e a Figura (c) apresenta o resultado após a aplicação da máscara sobre o objeto.

3.6 Tempo de execução dos algoritmos

Em pesquisa informal utilizando o site de relacionamento Linkedin foi feito um questionamento para modeladores 3D, sobre qual o tempo médio para se modelar objetos similares aos usados neste projeto de pesquisa. Em média, os modeladores responderam que é gasto entre 2 horas a 1 dia de trabalho para se modelar personagens, objetos e aeronaves com arte conceitual como as mostradas nas figuras 3.8. O tempo de modelagem varia, dependendo da complexidade do objeto a ser modelado e da experiência do modelador com as ferramentas atualmentes utilizadas.

Objeto	**Tempo médio de processamento (em segundos)**
Avião	39 segundos
Faca	12 segundos
Sofá	196 segundos
Ovelha	243 segundos

Tabela 3.1. A tabela acima apresenta o tempo gasto de processamento para gerar cada modelo 3D pelo algoritmo proposto.

A Tabela 3.1 apresenta os tempos gastos para geração dos modelos 3D utilizando os algoritmo apresentado. Os tempos gastos para gerar o modelo 3D demonstram a capacidade da solução em reduzir o tempo necessário para gerar os modelos 3D.

3.7 Sumário

Neste capítulo foi apresentada a metodologia utilizada nesta dissertação, detalhando o processo para extrair a estrutura tridimensional a partir das diversas visualizações de um objeto utilizando os algoritmos *Visual Hull* e *Marching Intersects*. Além disso, foi abordado o algoritmo clássico *Marching Cubes* responsável por extrair a malha de polígonos do grid de voxels.

No Capítulo 4 são apresentadas soluções para melhorar o resultado da malha 3D. Apesar dos esforços para remover problemas de áreas pontiagudas e serrilhadas utilizando o algoritmo *Marching Intersections*, esse problema é inerente à utilização de voxels. Como tentativa de resolver o problema é apresentada uma solução de suavização da malha. Outro problema é a quantidade de polígonos necessários para representar a borda devido ao alto número de vértices da estrutura de voxels. Como em objetos existem várias áreas planas ou semi-planas, muitos vértices são desnecessários. Sendo assim, também é apresentada uma solução que tem por objetivo reduzir o número de polígonos, sem comprometer a qualidade visual.

Capítulo 4

Melhorias do resultado da malha

Este capítulo apresenta duas etapas responsáveis por melhorar o resultado da malha 3D gerada a partir do grid de voxel. Primeiramente, é executada uma etapa de suavização da malha, que procura retirar as quinas pontiagudas e áreas serrilhadas arredondando a malha. Porteriormente, é apresentada uma solução que tenta fazer redução de polígonos da malha, para transformar o modelo 3D em um modelo low poly, sem comprometer a qualidade da morfologia do objeto.

4.1 Suavização da malha

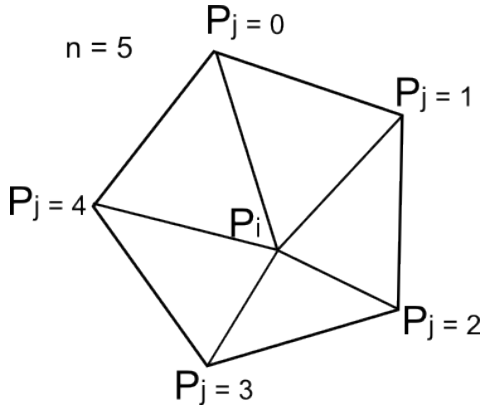


Figura 4.1. A figura acima representa um exemplo de situação para relaxamento de arestas, reposicionando os vértices.

Suavização da malha é um processo pelo qual os vértices de uma malha 3D, gerada a partir das visualizações 2D, são posicionados de forma a eliminar quinas que não deveriam ocorrer. O algoritmo funciona de forma iterativa, ou seja, cada vez que é feita a suavização da malha, o algoritmo reposiciona os vértices em relação ao seus

vizinhos, isso pode ser feito N vezes até que seja alcançado o resultado esperado. O algoritmo utilizado é baseado no operador laplaciano proposto no trabalho Bardyn et al. [2010]. Abordagem similar é apresentada no trabalho Chica et al. [2008]. O operator laplaciano é definido por:

$$\sum_{j=1}^{N}(P_j - P_i) \tag{4.1}$$

Onde P_i é vértice pelo qual deve ser suavizado, P_j são os vértices adjacentes a P_i e (P_j - P_i) é o vetor que mostra a direção de P_i para P_j. A figura 4.1 mostra um exemplo de polígonos pelo qual pode-se aplicar o operador laplaciano para fazer relaxamento de arestas.

O algoritmo utiliza o conceito de relaxamento, similar aos algoritmos tradicionais de menor caminho; porém, é relaxado a posição do vértice. Ao invés de selecionar a menor aresta, ele reposiciona o vértice para que as arestas relativas aos vértices vizinhos fiquem relaxadas ou suavizadas. Para se fazer ao suavização da malha é utilizada a média de todos os vetores adjacentes ao vértice escolhido. Faz-se o somatório de todos os vetores adjacentes e divide pelo número de vizinhos, definido pela fórmula:

$$Pi' = (Pi + \sum_{j=1}^{N}(P_j - Pi) \times \frac{1}{N}) \tag{4.2}$$

.

Abaixo é apresentado o pseudo-código do algoritmo de suavização da malha 3D:

```
for cada vértice X, Y, Z que compõe a malha do
    for cada vértice adjacente do
        calcular o vetor direção para o vértice adjacente;
        fazer somatório entre vetores adjacentes;
    end
    calcular a média suavizada do somatório dos vetores;
    atualizar a posição do vértice corrente;
end
```

Algoritmo 4: Pseudo-código do algoritmo de suavização

A abordagem utilizada para este trabalho produz resultados satisfatórios com malha regular estilo grid. Isso ocorre porque o relaxamento tende a aproximar os vértices uns dos outros. Quando o modelo possui polígonos com arestas grandes, esse algoritmo não produz bons resultados deformando a malha, pois vértices que compõem bordas dos objetos possuem tendência a sair de suas posições originais.

A etapa da suaviação da malha percorre todos os vértices da malha triângular e suaviza a malha quantas vezes o usuário desejar. Para fazer essa operação é calculada a posição do vértice corrente em relação ao seus vizinhos. Sendo assim, é possível afirmar que a suavização possui ordem de complexidade de $O(V + E^2)$, onde V é número de vértices e E é o número e arestas. É desconsiderado o número de iterações pois o algoritmo continuará sendo $O(V)$.

4.2 Redução poligonal

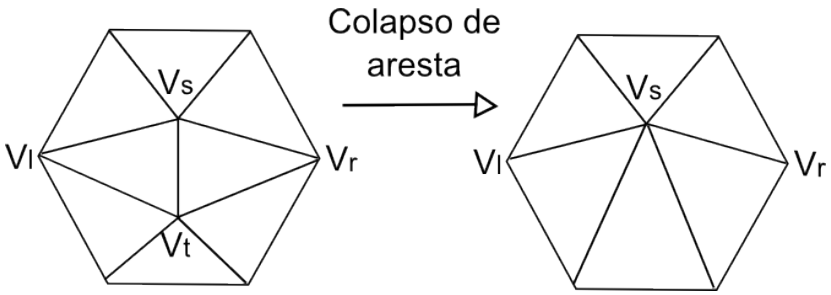


Figura 4.2. A figura acima demostra a forma como é feita a redução poligonal, fazendo colapso de arestas e destruindo vértices desnecessários.

Um dos maiores problemas de se gerar modelos 3D a partir de estrutura de voxels é a quantidade de polígonos gerados, já que, malhas extraídas dessa estrutura possuem auto nível de detalhes, como consequência são gerados muitos polígonos. Sendo assim, para atender à indústria do entretenimento é necessário fazer uma redução poligonal, pois em aplicações em tempo real como jogos, modelos com grande quantidade de polígonos podem gerar custo na performance da aplicação.

Esse algoritmo é baseado em uma técnica muito usada por algoritmos de LOD (Level of Details). Os algoritmos de LOD fazem redução poligonal sem se preocuparem com a qualidade máxima do modelo, pois o objetivo é realmente simplificar a malha para que a mesma seja visualizada em distâncias diferentes. Por exemplo, quando um motorista está dirigindo em uma estrada, ao ver um carro de perto ele percebe todos os detalhes do carro; porém, quanto mais ele se distância do carro, menos detalhes são visíveis do carro. Sendo assim, quanto maior for a distância menor a necessidade do modelo 3D possuir grande quantidade de informações. Essa técnica é muito utilizada em jogos digitais.

Para resolver esse problema, foi incluído no processo de geração de modelos 3D a etapa de redução poligonal. O algoritmo que foi proposto se baseia no trabalho Zhu et al. [2005], que utiliza técnicas comuns em algoritmos de LOD como no trabalho

Melax [1998]. Na abordagem utilizada, foi incluído um parâmetro para determinar qual a curvatura máxima entre os triângulos adjacentes que podem ser removidos, isso foi feito já que, diferente dos algoritmos de LOD que não se preocupam com a definição precisa do modelo simplicado, este projeto de pesquisa procura extrair de imagens 2D modelos 3D precisos. Sendo assim, com essa extensão, o algoritmo permite simplificar a malha onde não é necessário muitos polígonos e mantém os detalhes quando o modelo necessita de maiores detalhes, como locais com muitas curvas ou mudanças de direção da superfície do objeto.

O algoritmo é iterativo e executa enquanto existirem triângulos que podem ser excluídos da malha. Ao iniciar o algoritmo é passado o ângulo máximo entre os triângulos adjacentes que podem ser apagados. Por Exemplo: Ao passar 10 graus como ângulo máximo, o algoritmo exclui todos os vértices, cujos triângulos adjacentes possuem ângulos como no máximo 10 graus. Para fazer a exclusão do vértices, o algoritmo calcula a importância de cada vértice da malha 3D.

A importância é calculada utilizando a seguinte fórmula:

$$I_{vi} = Curvatura(V_i) \times \sum_{j=1}^{N} Lenght(V_j - V_i) \tag{4.3}$$

Onde I_{vi} é a importância do vértice i, a curvatura de V_i é calculada como sendo a maior curvatura entre os triângulos adjacentes ao V_i.

A fórmula que representa o calculo da curvatura é dada por:

$$CurvaturaV_i = max(arcoseno(normal(T_a).normal(T_b))) \tag{4.4}$$

Onde T_a e T_b são triângulos adjacentes ao vértice V_i.

Durante o cálculo da importância dos vértices, caso os triângulos adjacentes ao vértice possuam ângulos entre si superior ao limite estabelicido, o vértice é marcado como indisponível para exclusão.

Após calculada a importância de cada vértice, o algoritmo seleciona o vértice que possuir menor importância para a malha 3D. O vértice escolhido e todos os triângulos que compõem esse vértice são excluídos. Ao fazer isso, a malha fica desconexa, o algoritmo então conecta os vértice adjacentes ao vértice adjacente mais próximo refazendo a conexão. Além disso, para cada vértice adjacente ao vértice excluído é recalculado sua importância e todo o processo é repetido até que não exista nenhum vértice que possa ser removido.

Abaixo é apresentado o pseudo-código dessa solução:

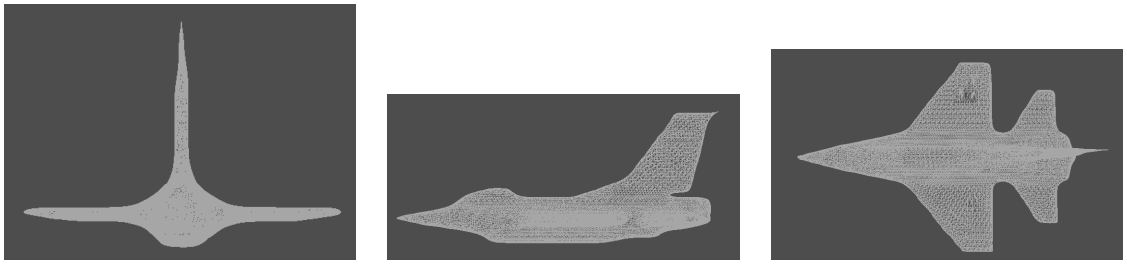
```
for cada vértice incluído na malha do
    if existir algum ângulo entre os triângulos adjacente maior que ângulo α
    then
        marcar o vértice como vértice indisponível para exclusão;
    else
        calcula a importância do vértice;
        escolher o vértice adjacente cuja a aresta possui a menor angulação
        entre triângulos adjacentes;
    end
end
while Existe vértice que pode ser excluído do
    Escolher o vértice de menor importância;
    remover o vértice;
    for cada triângulo adjacente ao vértice excluído do
        conectar os triângulos restantes ao vértice adjacente escolhido;
    end
    for cada vértice adjacente ao excluído do
        if existir algum ângulo entre os triângulos adjacente maior que ângulo
        α then
            marcar o vértice como vértice indisponível para exclusão;
        else
            recalcular a importância do vértice;
        end
    end
end
```

Algoritmo 5: Pseudo-código da solução de redução poligonal.

Por fim, executando o algoritmo de redução poligonal é possível perceber que ele calcula para cada vértice, a sua importância na malha, e posteriormente, remove o vértice de menor importância, reconectando os vértices adjacentes com o vizinho de maior importância. Para calcular a importância de cada vértice, calcula-se a curvatura dos triângulos adjacentes aos triângulos formados pelo vértice corrente, para tal, é necessário visitar todos os vértices adjacentes. Sendo assim, o algoritmo percorre as arestas 2 vezes. Pois o passo descrito acima é feito para todos os vértices. Se for levado em consideração que é possível remover toda a malha então é possível prever que esse passo do algoritmo terá $O(VA^2 + V(VA^2))$. Onde V são os vértices e A as arestas que levam aos vértice adjacentes.

4.3 Testes

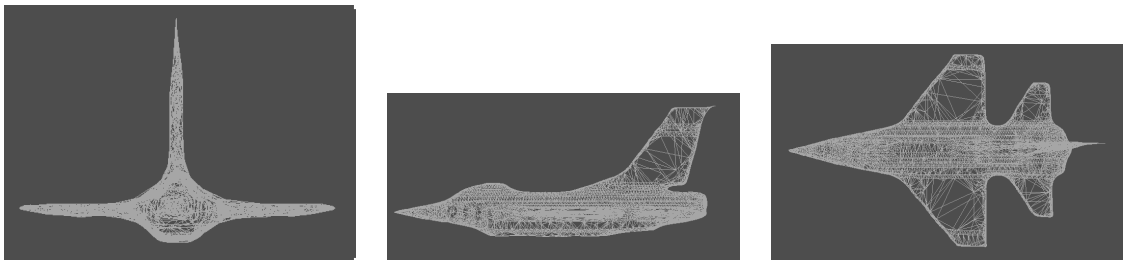
No capítulo anterior foram apresentados os testes utilizados para verificar a capacidade da metologia de gerar modelos 3D a partir de desenhos conceituais de objetos. Dando continuidade a esse processo são apresentados os testes feitos nos modelos após a execução das etapas de suavização e redução poligonal. As figuras 4.3 (a), (b) e (c) apresentam o avião após a etapa de suavização. É possível verificar que, os contornos do avião ficaram mais próximos da arte conceitual.



(a) (b) (c)

Figura 4.3. Modelo 3D do avião após etapa de suavização da malha 3D.

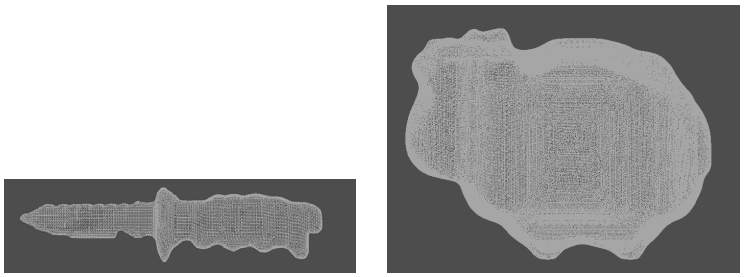
Após executada a etapa de suavização da malha temos um modelo cuja superfície foi estimada corretamente. As figuras 4.4 (a), (b) e (c) apresentam diferentes visualizações do avião após a execução da etapa de redução poligonal. Percebe-se que o algoritmo removeu polígonos em áreas mais planas, onde não existia necessidade de tantos polígonos e manteve mais polígonos em áreas curvas, evitando assim deformações da malha, mesmo para um objeto com uma geometria mais complexa, como é o caso de um avião.



(a) (b) (c)

Figura 4.4. Modelo 3D do avião após etapa de redução poligonal.

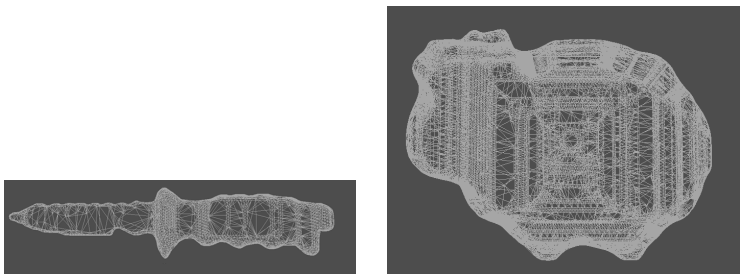
As figuras 4.5 (a) e (b) apresentam a ovelha e a faca após executado o processo de suavização da malha 3D.



(a) (b)

Figura 4.5. A figura (a) apresenta a faca e figura (b) apresenta a ovelha após a etapa suavização da malha 3D.

As figuras 4.6 (a) e (b) apresentam a ovelha e a faca após executado o processo de redução poligonal. É possível verificar em todos os modelos que, após executada essa etapa foi alcançado o objetivo proposto. As malhas apresentam resultado suave aproximados às artes conceituais; além disso, os modelos em low poly não comprometeram fortemente o resultado da morfologia dos objetos utilizados.



(a) (b)

Figura 4.6. A figura (a) apresenta a faca gerada utilizando a metologia proposta e a figura (b) apresenta a ovelha gerada.

4.4 Sumário

Neste Capítulo foram apresentadas duas etapas incluídas no processo de geração da malha, com o fim de melhorar a qualidade da malha gerada e atender a indústria do entretenimento digital. Primeiramente, é apresentado um algoritmo que recebe a malha 3D como entrada e faz um processo de suavização, retirando locais pontiagudos e serrilhados. Posteriormente, foi apresentado um algoritmo capaz de fazer a redução do números de polígonos, transformando a malha 3D em um modelo low poly, sem comprometer a qualidade da malha gerada.

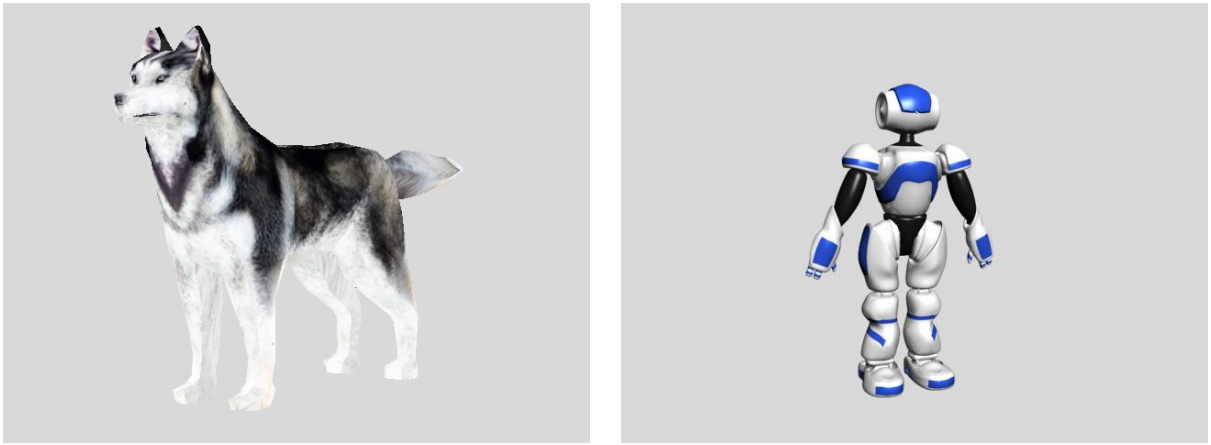
No Capítulo 5 apresentamos experimentos que foram realizados para verificar qualidade da malha gerada. Primeiramente, são utilizados dois modelos 3D modelados pelo processo manual atualmente utilizado na indústria. São extraidas as visualizações desses objetos, e os modelos 3D são reconstruídos utilizando a metologia proposta para comparar os resultados. A partir disso, são feitos testes quantitativos para validar a qualidade da malha gerada nesse modelos e também nas artes conceituais utilizadas nos testes iniciais.

Capítulo 5

Experimentos

Para fazer experimentos foram utilizados modelos 3D modelados utilizando a abordagem manual. Foram extraídas as visualizações desses modelos 3D em 3 pontos de vistas diferentes e, após feito isso, os modelos foram reconstruídos utilizando o algoritmo proposto neste trabalho de pesquisa. Ainda é apresentada uma seção que faz uma análise quantitativa dos resultados apresentados para verificar a qualidade da malha gerada.

5.1 Validação com modelos 3D



(a) (b)

Figura 5.1. Modelos 3D utilizados para comparar os resultados com o algoritmo automático, (a) o cachorro e (b) o robô.

Além dos testes feitos com as artes conceituais, para se comprovar a capacidade do algoritmo de gerar modelos 3D, foram selecionados dois modelos 3D modelados

utilizando o processo manual; posteriormente, foi tirada um foto do modelo nos ângulos frontal, lateral e superior. Essas visualizações foram utilizadas para gerar o modelo pela abordagem apresentada e comparar o resultado com o modelo original modelado manualmente. Os modelos escolhidos foram um cachorro e um robô humanoide como mostram a Figuras 5.1 (a) e (b).

Objeto	**Polígonos**
Cachorro	1544
Robô	17906

Tabela 5.1. A tabela acima apresenta a quantidade de polígonos de cada um dos modelos utilizados.

A Tabela 5.1 mostra a quantidade de polígonos de cada um dos modelos utilizando a abordagem manual. Para verificar em qual estágio o algoritmo desenvolvido se encontra, foi escolhido um modelo low polly (cachorro) e outro high polly (rôbo). O objetivo é comparar os modelos gerados pelo algoritmo com o modelo original, e verificar se o algoritmo é capaz de gerar modelos low polly de qualidade ou modelos high polly.

As Figuras 5.2 (a) até (f) mostram o modelo 3D do cachorro com suas quantidades de polígonos respectivos. Para se fazer esse teste, foi alternada a variável que define o ângulo limite entre polígonos adjacentes que podem ser removidos da malha. Inicialmente, é permitido no máximo 5 graus, e no último teste é permitido fazer redução poligonal em polígonos adjacentes com no máximo 40 graus. Quanto maior o ângulo permitido maior é redução poligonal executada pelo algoritmo.

É possível verificar pelos testes que o modelo gerado com polígonos similares ao original, apesar de apresentar a morfologia desejada e resultado satisfatório, apresentou algumas deformações na malha. Porém, o modelo high polly apresentou ótimo resultado. Esse resultado comprova que, apesar do algoritmo de redução poligonal reduzir consideravelmente a malha gerada, ainda não é capaz de gerar a malha em low polly sem possuir erros na morfologia do objeto, sendo necessário se fazer ajustes na abordagem de redução poligonal para evitar que essas deformações possam aparecer na malha gerada. É apresentada também a visualização do cachorro em visão perspectiva nas Figuras 5.3 (a) e (b).

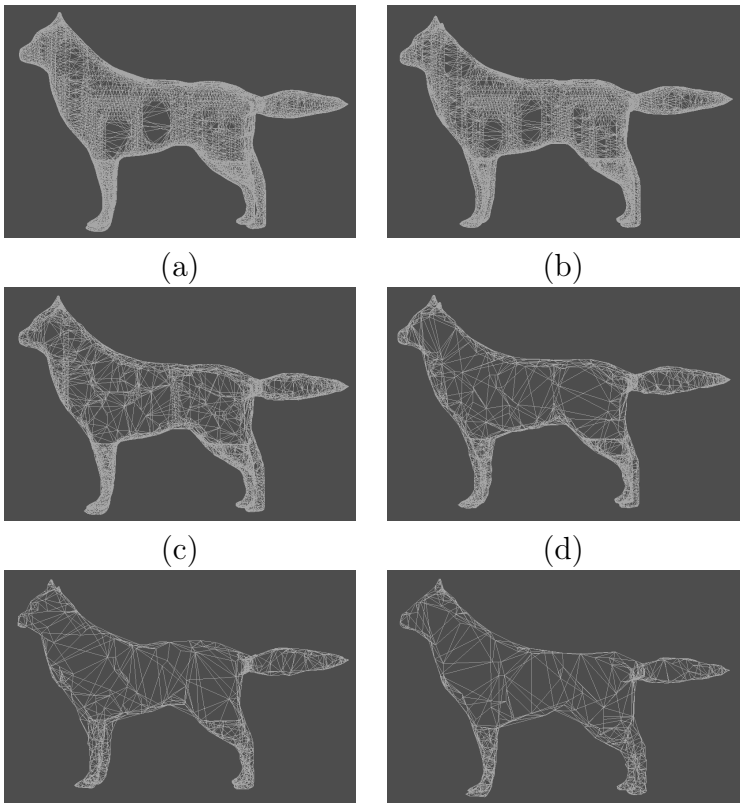


Figura 5.2. As Figuras apresentam os modelos do cachorro gerados utilizando a visualizações. A Figura (a) possui 23266 polígonos, (b) possui 19120, (c) possui 10052, (d) possui 5864, (e) possui 2554 e (f) possui 1644 polígonos.

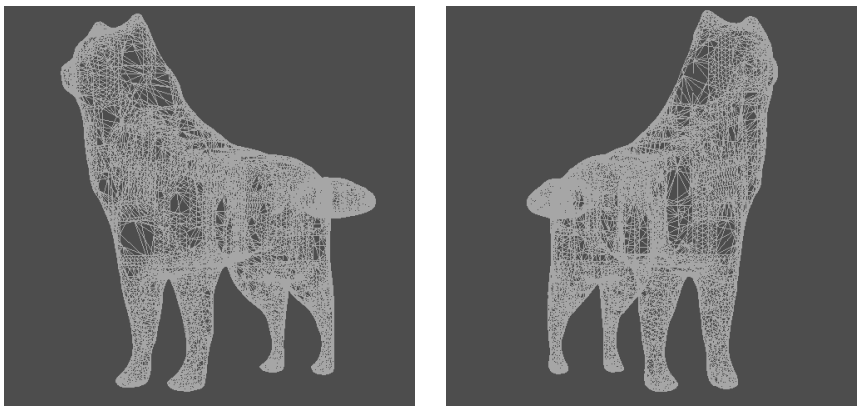
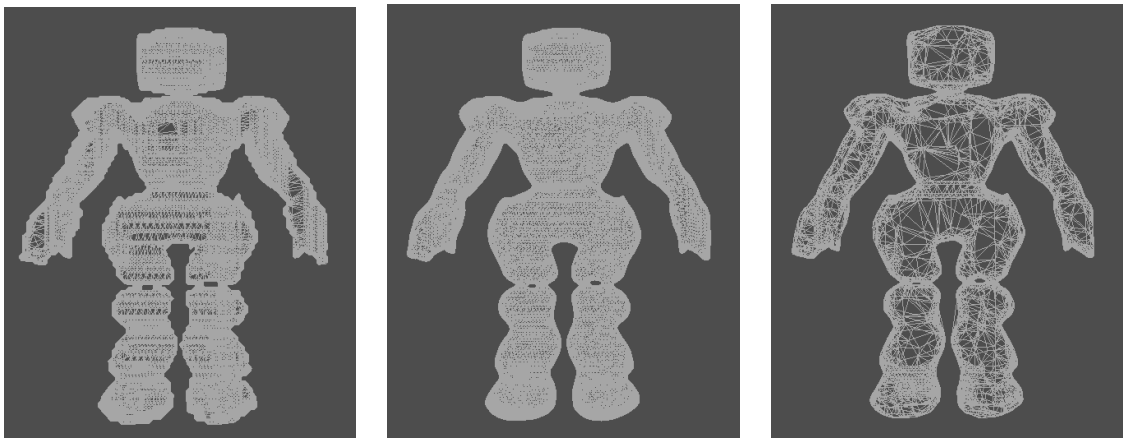


Figura 5.3. As Figuras (a) e (b) mostram o cachorro em visão perspectiva.

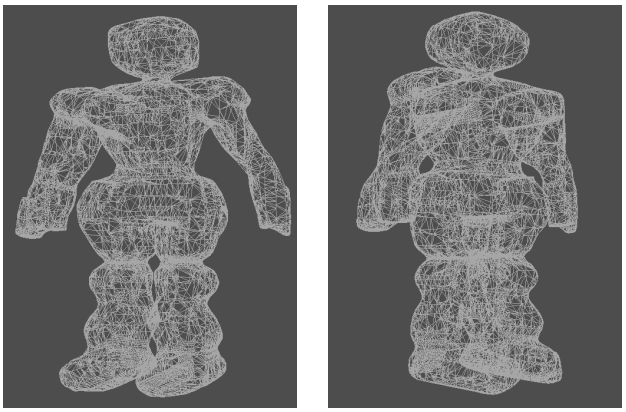
As Figuras 5.4 (a), (b) e (c) apresentam o robô gerado utilizando a abordagem apresentada. A Figuras 5.5 (a) e (b) em visão perspectiva mostram o rôbo com quan-

tidade de polígonos similares ao modelo original sem grande perda na qualidade da malha gerada.



(a) (b) (c)

Figura 5.4. A Figura (a) mostra a faca após a execução do algoritmo Visual Hull, a imagem (b) apresenta o resultado da suavização da malha 3D e a Figura (c) é a faca com redução poligonal.



(a) (b)

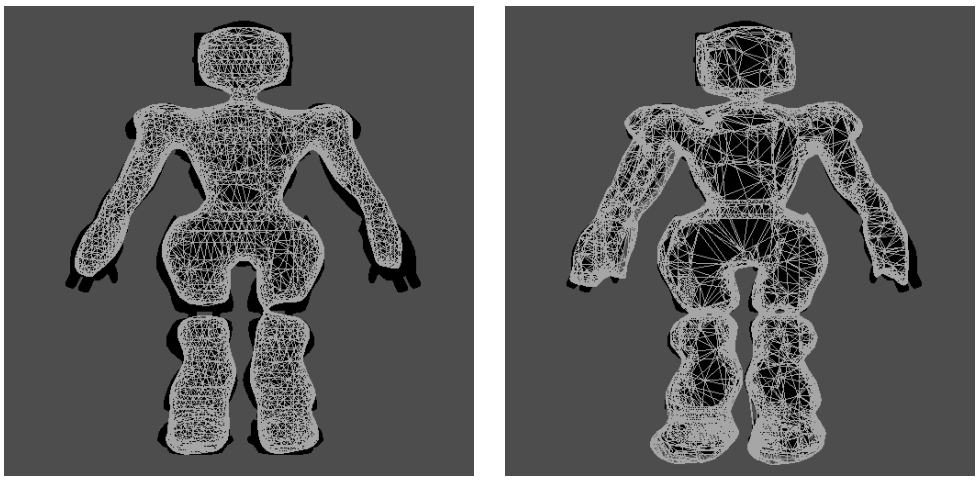
Figura 5.5. As figuras (a) e (b) mostram o robô gerado com 18210 polígonos.

A Tabela 5.2 apresenta a comparação da quantidade de polígonos dos modelos reconstruídos na metodologia em relação aos modelos originais. Foi possível perceber que o algoritmo de redução poligonal reduziu consideravelmente a quantidade polígonos sem gerar grandes deformações, porém, ele não foi capaz de atingir o número de polígonos dos modelos originais construídos pelo processo manual. O que leva a conclusão que o algoritmo de redução poligonal ainda precisa de ajustes para alcançar o resultado esperado.

Modelo	Cachorro	Robô
Original	1544	17906
Reconstruído sem otimização	22300	56200
Otimizado sem deformações	2500	22000
Otimizado com deformações	1700	18300

Tabela 5.2. A tabela acima a comparação da quantidade de polígonos dos modelos em situação.

5.2 Análise quantitativa dos resultados



(a) (b)

Figura 5.6. Malha 3D sobreposta sobre a arte conceitual do rôbo na visualização frontal para validação da aproximação. A Figura (a) foi gerada a partir de grid com resolução 8 vezes menor e a Figura (b) com resolução 4 vezes menor.

Para se fazer a análise quantitativa dos resultados, foram tiradas fotos das imagens geradas pelos modelos 3D na mesma escala que os desenhos conceituais. Posteriormente, foram sobrepostas as imagens umas sobre as outras e verificadas quão distantes ficaram umas imagens das outras através de contagem de pixels. Foram feitos testes com grid de voxels com tamanho 8 e 4 vezes menor que a resolução da imagem. Quanto maior for a resolução do grid, mais próximo fica da resolução da imagem, sendo capaz de extrair mais detalhes das visualizações. As Figuras 5.6 (a) e (b) mostram a forma como a malha 3D foi sobreposta sobre a arte conceitual para analisar a qualidade de aproximação gerada. A parte escura da imagem é a arte conceitual do modelo, a coloração clara é malha 3D gerada pelo algoritmo proposto.

Testes como resolução de grid maiores não foram feitos devido à limitação de capacidade de memória da máquina utilizada para testes.

Objeto	**Erro em relação as imagens**	**Áreas em falta**	**Áreas em excesso**
Avião	14,6%	12,2%	2,4%
Faca	48,8%	48,1%	0,6%
Ovelha	6,5%	5,7%	0,8%
Cachorro	20,6%	20,4%	0,2%
Robô	13,5%	10,8%	2,7%

Tabela 5.3. A tabela acima apresenta a porcentagem de erros dos modelos 3D para para resolução do grid 8 vezes menor que a resolução das imagens.

A Tabela 5.3 apresenta os resultados dos testes realizados com grid de voxels com resolução 8 vezes menor que das imagens das visualizações dos objetos e a Tabela 5.4 apresenta os resultados com grid de voxel com resolução 4 vezes menor. É apresentado a porcentagem de erro da malha gerada em relação a arte conceitual das diferentes visualizações dos objetos. O resultado confirma que quanto maior a resolução do grid, melhor é o resultado de aproximação. Além disso, é possível verificar que objetos com áreas finas como a faca apresentam resultados muito ruins quando o grid de voxel possui baixa resolução, e objetos com superfícies mais cheia como a ovelha apresentam aproximação aceitável para resoluções do grid mais baixas.

Objeto	**Erro em relação as imagens**	**Áreas em falta**	**Áreas em excesso**
Avião	4,6%	3,8%	0,8%
Faca	4,2%	3,8%	0,4%
Ovelha	2,1%	1,7%	0,4%
Cachorro	4,9%	4,6%	0,3%
Robô	3,1%	2,3%	0,8%

Tabela 5.4. A tabela acima apresenta a porcentagem de erros dos modelos 3D para resolução do grid 4 vezes menor que a resolução das imagens.

O gráfico mostrado na Figura 5.7 apresenta a variação do erro em relação ao grid de voxel. Ao analisar o gráfico fica claro a melhora na qualidade do resultado à medida que a resolução do grid se aproxima da resolução das imagens. Isso é fácil de entender, também levando em conta que o voxel é a represetação do pixel, porém em três dimenssões.

O algoritmo apresentou média de 20,78% de erros para grid de voxel com 8 vezes menos de resolução em relação as imagens da visualização e média de 3,78% de erros

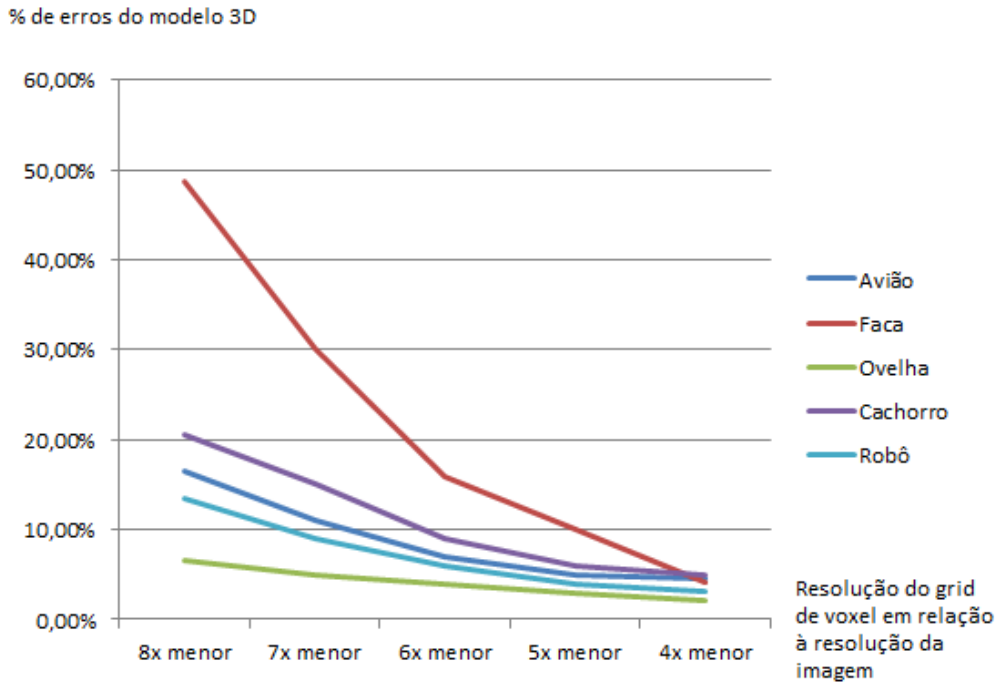


Figura 5.7. A figura acima apresenta o gráfico que mostra o erros dos modelos 3D em relação ao tamanho do grid de voxels.

para grid de voxel com resolução 4 vezes menor que resolução das imagens originais.

Quando verificado a capacidade do algoritmo de gerar modelos 3D por algoritmo é preciso levar em consideração também o tempo de processamento. O gráfico mostrado na Figura 5.8 mostra o tempo gasto pelo algoritmo para gerar os modelos 3D, variando o tamanho do grid de voxel. Pelo gráfico é possível perceber que mesmo para um grid com resolução 4 vezes menor o tempo de processamento é aceitável; porém, o gráfico deixa claro que se aumentarmos ainda mais a resolução do grid a tendência é que o tempo aumente bastante.

Para finalizar, foi feito uma pesquisa informal com alguns modeladores e animadores 3D, no qual foi questionado sobre a qualidade da malha 3D gerada. Em geral, os artistas mostraram preocupação com a malha após executada a redução poligonal, pois a malha não respeitas traços importantes para animação como por exemplo o esqueleto do objeto. Sendo assim, foi possível concluir que pode ser necessário incluir mais dados na solução para se obter a malha com a qualidade necessária. Um opção seria a importação da pelo sistema do esquelo do modelo e o esqueleto ser utilizado para se fazer o calculo da importância do vértice para malha no algoritmo de redução

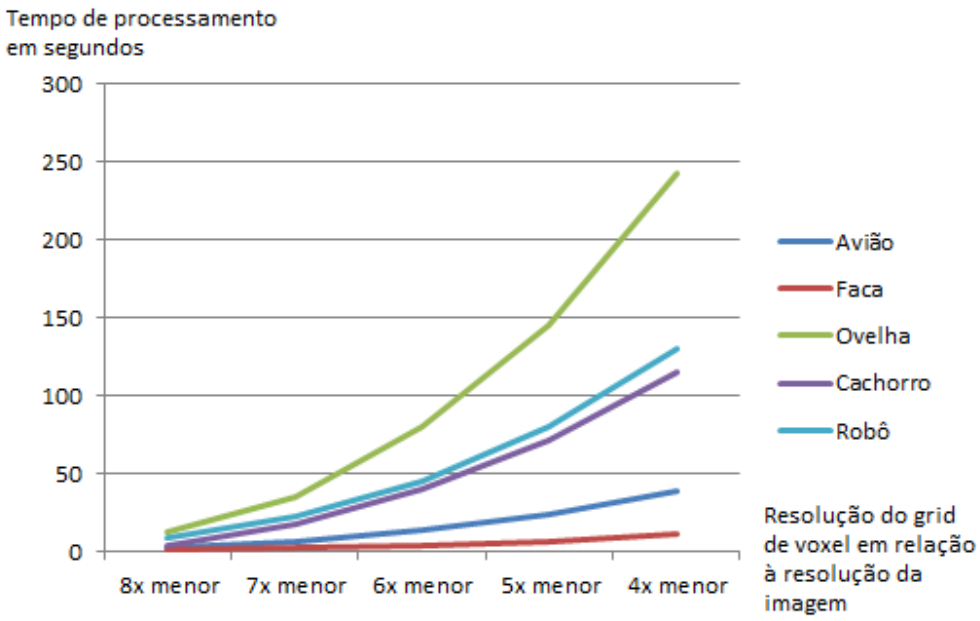


Figura 5.8. A figura acima mostra o gráfico do tempo de processamento em relação ao tamanho do grid de voxels.

poligonal.

5.3 Sumário

Neste Capítulo foram apresentados os experimentos realizados para validar a capacidade do algoritmo de automatizar o processo de geração de modelos 3D. Foram feitos experimentos quantitativos com modelos 3D modelados utilizando o processo manual, além de algumas artes conceituais previamentes escolhidas; foi mostrada a porcentagem de erro do modelos 3D gerados em relação as visualizações, utilizando duas configurações de tamanho do grid de voxels. Na primeira configuração, o grid de voxel possuia resolução 8 vezes menor que a resolução das imagens, mostrando resultados ruins para essa configuração. Posteriormente, foi utilizado um grid voxel com resolução 4 vezes menor que a resolução das imagens, apresentando resultados satisfatórios.

No Capítulo 6 é apresentada a conclusão do trabalho, sendo apresentadas também, as limitações da solução abordada e propostas de trabalhos futuros.

Capítulo 6

Conclusão

O metodologia apresentada neste projeto é uma solução para automatizar o processo de geração de modelos 3D através das visualizações 2D de algum objeto. O tempo de processamento dos algoritmo apresentados na tabela 3.1 mostra a capacidade dessa metodologia de gerar modelos 3D a partir das visualizações 2D de um objeto em um tempo capaz de melhorar o processo manual atualmente utilizado na indústria.

A solução prosposta se mostrou de fácil implementação, pois não utiliza cálculos matemáticos complexos e utiliza algoritmos já conhecidos. Sendo adicionado como contribuição a apresentação do algoritmo *Marching Intersections* que melhora a aproximação das informações de volume referentes as imagens 2D, a inclusão da etapa de suavização da malha 3D que remove as áreas pontiagudas e serrilhadas da malha 3D gerada utilizando o algoritmo *Marching Cubes* e a inclusão de uma etapa de redução poligonal com o intuito de remover polígonos desnecessários, tornando, assim, o modelo em low poly.

Os experimentos realizados comprovaram que esta metodologia é capaz de gerar modelos que apresentam pequena quantidade de erros utilizando apenas desenhos simplificados. As tabelas de erros apresentadas no capítulo 5 mostram que para um grid de voxel com resolução 4 vezes menor que as resoluções das imagens, é possível gerar modelos com qualidade aceitável. Em média esses modelos apresentaram erros da faixa de 2% a 5%, sendo que, se a resolução do grid fosse maior ainda, pelo gráfico 5.7 fica evidente que os erros possuem tendência de queda.

Sendo assim, concluímos que apesar de existirem algumas limitações que estão descritas na seção abaixo, a metologia proposta neste trabalho de pesquisa se mostrou eficiente e capaz de gerar modelos 3D a partir de visualizações simplificadas; porém, cabem melhorias para que a solução atendenda integralmente à necessidade da indústria de entretenimento digital.

6.1 Limitações da abordagem escolhida

A abordagem apresentada neste trabalho é capaz de gerar modelos 3D precisos a partir de desenhos simplificados; porém, ela possui algumas limitações:

- Não consegue gerar modelos precisos em superfícies côncavas ou aberturas na silhueta, como sofás, cones, etc;
- Não mantém a precisão dos modelos gerados quando a arte conceitual possui silhuetas com proporções diferentes;
- Apesar de simplificar a malha triângular, quando comparada a modelos low poly, é possível verificar pequenos erros de deformação na malha;

Das limitações descritas acima a mais impactante é a primeira, pois existe grande quantidade de objetos com as características descritas. Para entender melhor o problema, é possível imaginar um sofá. Tanto as visualizações laterais, superiores ou frontais são incapazes de detectar a abertura central onde as pessoas se sentam. Isso ocorre porque a análise morfológica de objetos possui grande complexidade. Os algoritmos até hoje estudados não conseguem ter a precisão de entendimento da capacidade humana ainda.

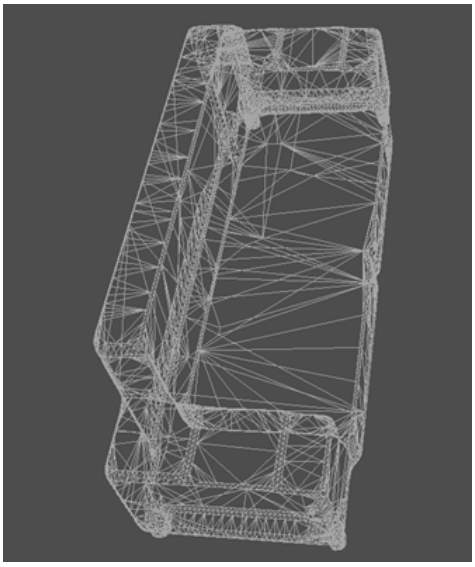


Figura 6.1. A figura acima mostra um sofá gerado através da técnica descrita acima.

Sendo assim, todas as soluções para o problema encontradas até o momento são soluções técnicas e paleativas. O trabalho Rivers et al. [2010] propõe a utililização de

cores diferentes nas silhuetas para que a análise morfológica seja feita. Outra abordagem seria trabalhar com as variações de cores nos próprios desenhos 2D, o que pode não ser trivial, já que textura de modelos possuem variações de cores muito fortes.

Nesse trabalho foi utilizada uma solução para o problema que, diferentemente das soluções abordadas até aqui não trabalha com as cores para detecção da morfologia do objeto. A solução utiliza um processo semi-automático, no qual o artista define mais de uma máscara para a mesma visualização e, posteriormente, define até que momento do espaço aquela visualização deve ser processada. A Figura 6.1 mostra um sofá gerado utilizando essa solução.

6.2 Trabalhos futuros

Para trabalhos futuros é planejado aplicar textura extraída das imagens no modelo gerado, trabalhar na solução côncavas, de análise de cores diferentes. Além disso, é possível substituir o algoritmo Marching Cubes tradicional Lorensen & E. [1987] pela solução morderna Dual Marching Cubes Schaefer & Warren [2004], que produz resultados mais precisos na extração de polígonos da estrutura de Voxels. Pode-se fazer também análise morfológica das imagens para funcionar mesmo com silhuetas de tamanhos aproximados e não somente exatos como desenvolvido nesta pesquisa. Por fim, é proposto também a criação de um editor no qual o usuário desenha as visualizações e o modelo 3D é gerado. Essa aplicação poderia ser desenvolvida para PC, mesa de desenho e tablets.

Além disso, para atender a demanda da indústria do entretenimento digital é necessária a melhoria no resultado produzido pelo algoritmo de redução poligonal, pois a solução desenvolvida, apesar de produzir resultados satisfatórios ainda não é capaz de reduz a malha em proporções semelhantes à abordagem manual, sem gerar deformações na malha.

Referências Bibliográficas

Adobe Systems Inc. (1989). Adobe photoshop. `http://adobe.com/photoshop`.

Alias Systems Inc. (2001). Maya. `http://www.autodesk.com/maya`. In property of Autodesk Media and Entertainment Inc.

Autodesk Inc. (1982). Auto cad. `http://usa.autodesk.com/autocad/`.

Autodesk Inc. (2001). 3d studio max. `http://www.autodesk.com/3dsmax`.

Bardyn, T.; Reyes, M.; Larrea, X. & Büchler, P. (2010). Influence of smoothing on voxel-based mesh accuracy in micro finite-element. *Computational Biomechanics for Medicine*, pp. 85–96.

Barequet, G. & Sharir, M. (1994). Piecewise-linear interpolation between polygonal slices. *ACM 10th Computational Geometry Proceedings*, 1:93–102.

Blender Foundation (2006). Blender. `http://www.blender.org/`.

Chen, T.; Zhu1, Z.; Shamir, A.; Hu1, S.-M. & Cohen-Or, D. (2013). Sweep: Extracting editable objects from a single photo. *SIGGRAPH ASIA*.

Chica, A.; Williams, J.; Andujar, C.; Brunet, P.; Navazo, I.; Rossignac, J. & Vinacua, A. (2008). Pressing: Smooth isosurfaces with flats from binary grids. *Computer Graphics Forum*, 27:36–46.

Cignoni, P.; Ganovelli, F.; Montani, C.; Pingi, P. & Scopigno, R. (2001). Marching intersections: an eficient resampling algorithm for surface management. *Shape Modeling and Applications, SMI 2001 International Conference on.*, pp. 296–305.

Dassualt Systemes Inc. (2009a). Catia. `http://www.3ds.com/products/catia/`.

Dassualt Systemes Inc. (2009b). Solid works. `http://www.solidworks.com/`.

Finkel, R. & Bentley, J. L. (1974). Quad trees: A data structure for retrieval on composite keys. *Acta Informatica*, 4:1–9.

G. Khronos (2004). Collada. `https://collada.org/`. COLLADA is an interchange file format for interactive 3D applications.

Gibson, S. F. F. (1998). Constrained elastic surface nets: Generating smooth surfaces from binary segmented data. *First International Conference Cambridge*, 21:888–898.

Google Inc. (2009). Google sketchup. `http://www.sketchup.com/`.

Grimm, C. & Joshi, P. (2012). An interface for sketching 3d curves. *EUROGRAPHICS Proceedings of the International Symposium on Sketch-Based Interfaces and Modeling*, 1:121–130.

Igarashi, T.; Matsuoka, S. & Tanaka, H. (1999). Teddy: A sketching interface for 3D freeform design. *ACM SIGGRAPH*, 1:1–8.

Ju, T.; Losasso, F.; Schaefer, S. & Warren, J. (2002). Dual contouring of hermite data. *ACM Transactions on Graphics (TOG) - Proceedings of ACM SIGGRAPH 2002*, 3:339–346.

Laurentine, A. (1994). The visual hull concept for silhouette-based image understanding. *Transaction on Pattern Analysis and Machine Intelligence*, 16(2).

Lazebnik, S.; Furukawa, Y. & Ponce, J. (2007). Projective visual hulls. *International Computing Vision Conference*, 74:137–165.

Lorensen, W. E. & E., H. (1987). Marching cubes: A high resolution 3d surface construction algorithm. *In: Computer Graphics*, 21(4).

Markosian, L.; Cohen, J.; Crulli, T. & Hughes, J. (1999). An interface for sketching 3d curves. *Symposium on Interactive 3D Graphics*, 1:17–21.

Matusik, W.; Buehler, C.; Raskar, R. & McMillan, L. (2000). Image based visual hulls. *ACM Transactions on Graphics*, 27:369–374.

Meagher, D. (1980). Octree encoding: A new technique for the representation, manipulation and display of arbitrary 3-d objects by computer. *Technical Report IPL-TR*, pp. 80–111.

Melax, S. (1998). A simple, fast, and effective polygon reduction algorithm. *Game Developer Magazine*.

Milne, P.; Nicollas, F. & Jager, G. (2004). Visual hull surface estimation. *Fifteenth Annual Symposium of the Pattern Recognition Association of South Africa*.

Mirye Inc. (2010). Shape 3d. `http://www.mirye.net/`.

Olsen, L.; Samavati, F. & Jorge, J. (2011). Naturasketch: Modeling from images and natural sketches. *IEEE Computing Graphics Appl. 31*, 1:24–34.

Rivers, A.; Durand, F. & Igarashi, T. (2010). 3d modeling with silhouettes. *ACM Transactions on Graphics*, 29(109).

Rocchini, C.; Cignoni, P.; Ganovelli, F.; Montani, C.; Pingi, P. & Scopigno, R. (2004). The marching intersections algorithm for merging range images. *The Visual Computer*, pp. 149–164.

Schaefer, S. & Warren, J. (2004). Dual marching cubes: Primal contouring of dual grids. *Proceedings of the Computer Graphics and Applications, 12th Pacific*, pp. 70–76.

Zhu, J.; Tanaka, T. & Saito, Y. (2005). A new mesh simplification algorithm for the application of surface sculpture.

Apêndice A

Tutorial de utilização dos algoritmos

Este tutorial apresenta instruções sobre como utilizar os algoritmos, classes e seus métodos desenvolvidos neste trabalho de pesquisa. Para o problema, foi desenvolvida uma classe que contém todos métodos necessários para adicionar as imagens da visualizações, processar as visualizações fazer suavizações e reduções sucessivas.

Foi desenvolvida uma classe chamada de GridVoxel, o código foi escrito em C++, utilizando a biblioteca de janelas multiplataforma Qt. Sendo que é necessário baixar o SDK do Qt para utlização do código. A classe possui um método pelo qual é possível se extrair os vértices da malha para salvar em algum modelo 3D ou renderizar utilizando alguma biblioteca gráfica como OpenGL ou DirectX.

Primeiramente, é necessário criar o grid de voxel utilizando o método *createGrid(int size)* é responsável por criar o grid de voxel, nesse método é passado o tamanho do voxel. O algoritmo suporta tamanhos menores que a resolução das visualizações. Porém, quanto maior o tamanho, maior o consumo de memória e tempo de processamento; portanto é recomedado utilizar grid com tamanho entre 128 e 256.

Após a criação do grid de voxels, é necessária a inclusão das visualizações dos objetos. O algoritmo exige, no mínimo, duas visualizações. São seis métodos disponíveis para adicionar as imagens, um para cada visualização. São suportados formatos padrões de arquivos (bmp, png, jpg e tga). As visualizações necessariamente devem possuir a mesma resolução. Resoluções diferentes podem ocasionar quebras do código. Os métodos disponíveis são *addFrontFile, addBackFile, addLeftFile, addRightFile, addTopFile, addBottomFile*. Além disso, é necessário configurar a cor responsável para ser usada como máscara no processamento das visualizações. A máscara é a cor que não representa o objeto; o método utilizado para esse fim possui o nome *setColorMask*.

Após configurados todos os parâmetros para o funcionamento do algoritmo, é o momento de rodar o algoritmo utilizando o método *processVisualHull(bool wireFrame)*. O parâmetro wireFrame é para informar ao algoritmo se é para exportar a malha em wireFrame. Quando se utiliza a malha em wire frame é necessário adicionar o vértice inicial ao final de cada triângulo para desenhá-lo utilizando OpenGL ou DirectX.

Após processado a algoritmo é preciso gerar a malha 3D utilizando o método *generateMesh(float pivotInterpolation = 0.5f, float limitAngleBetweenNormals = 5.0f, int smoothSteps = 5)*. O primeiro parâmetro representa um slide sobre a forma de interpolação que o usuário quer para gerar a malha; o valor varia de 0 até 1, sendo 0 para malhas pontiagudas e 1 para malhas mais arredondadas. O parâmetro limitAngleBetweenNormals é usado para configurar o ângulo máximo utilizado para redução poligonal e o parâmetro smoothStep é usado para definir a quantidade de passos que serão utilizados para fazer a suavização da malha 3D.

Por fim para renderizar a malha a classe GriVoxel possui três métodos para extrair as informações. Um é utilizado para retornar os vértices dos triângulos, o outro é utilizado para exportar os vetores *normais* e por último existe um método que retorna as cores associadas a cada vértices. Os métodos são *getVisibleVertices, getVisibleVerticeNormals, getVisibleVerticesColors*.

O código abaixo é um exemplo de como utilizar a classe GridVoxel.

```
void run()
{
    mGrid.addFrontFile(QString("c:/model/dog_front.png"));
    mGrid.addTopFile(QString("c:/model/dog_top.png"));
    mGrid.addRightFile(QString("c:/model/dog_right.png"));
    mGrid.setColorMask(QColor(255, 0, 0, 255));

    mGrid.createGrid(64);

    mGrid.processVisualHull(true);

    mGrid.generateMesh(0.5f, 0.0f, 7);
}

void drawModel()
{
    makeCurrent();
```

```
glEnable(GL_DEPTH_TEST);
glEnable(GL_CULL_FACE);
glEnable(GL_DOUBLE);

glClearColor(0.3f, 0.3f, 0.3f, 1.0f);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

QMatrix4x4 modelMatrix;
modelMatrix.setToIdentity();
modelMatrix.translate(0.0f, 1.0f, 5.0f);

modelMatrix.rotate(90.0f, 0.0f, 1.0f, 0.0f);
modelMatrix.scale(1.5f);

QMatrix4x4 viewMatrix;
viewMatrix.setToIdentity();
viewMatrix.lookAt(QVector3D(0.0f, 0.0f, -10.0f),
                  QVector3D(0.0f, 1.0f, 0.0f),
                  QVector3D(0.0f, 1.0f, 0.0f));

QMatrix4x4 modelViewProjectionMatrix =
             mCamera.getProjectionMatrix() *
             viewMatrix * modelMatrix;

program->bind();

program->setUniformValue("modelMatrix",
                         viewMatrix * modelMatrix);

program->setUniformValue("normalMatrix",
                         modelMatrix.inverted());

program->setUniformValue("matrix",
                         modelViewProjectionMatrix);

program->enableAttributeArray(PROGRAM_VERTEX_ATTRIBUTE);

if (mMustDrawLine == false)
```

```
    {
        program->setUniformValue("lightColor",
                                 QVector4D(1.0f, 1.0f, 1.0f, 1.0f));

        program->setUniformValue("specPow", 0.9f);
        program->setUniformValue("specShininess", 0.5f);
        program->setUniformValue("lightIntensity", 1.0f);
        program->setUniformValue("ambientIntensity", 0.1f);

        program->setUniformValue("lightSource",
                                  QVector3D(100.0f, 100.0f, 100.0f));

        program->enableAttributeArray(PROGRAM_TEXCOORD_ATTRIBUTE);
        program->setAttributeArray(PROGRAM_TEXCOORD_ATTRIBUTE,
                              mGrid.getVisibleVerticesColors().data());
        program->enableAttributeArray(PROGRAM_VERTEX_NORMAL_ATTRIBUTE);
        program->setAttributeArray(PROGRAM_VERTEX_NORMAL_ATTRIBUTE,
                               mGrid.getVisibleVerticeNormals().data());
        program->setAttributeArray(PROGRAM_VERTEX_ATTRIBUTE,
                               mGrid.getVisibleVertices().data());
        glDrawArrays(GL_TRIANGLES, 0, mGrid.getVisibleVertices().size());
    }
    else
    {
        program->setAttributeArray(PROGRAM_VERTEX_ATTRIBUTE,
                                    mGrid.getVisibleVertices().data());
        glDrawArrays(GL_LINES, 0, mGrid.getVisibleVertices().size());
    }

    glDisable(GL_DEPTH_TEST);
    glDisable(GL_CULL_FACE);
    glDisable(GL_DOUBLE);
}
```