JURANDIR DE OLIVEIRA SANTOS JUNIOR

RECONSTRUÇÃO PRECISA DE MODELOS 3D COM APLICAÇÃO NA PRESERVAÇÃO DIGITAL DE PATRIMÔNIOS NATURAIS E CULTURAIS

CURITIBA

2012

JURANDIR DE OLIVEIRA SANTOS JUNIOR

RECONSTRUÇÃO PRECISA DE MODELOS 3D COM APLICAÇÃO NA PRESERVAÇÃO DIGITAL DE PATRIMÔNIOS NATURAIS E CULTURAIS

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientadora: Profa. Dra. Olga Regina Pereira Bellon
Orientador: Prof. Dr. Luciano Silva

CURITIBA

2012

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

RESUMO

A preservação digital de patrimônios naturais e culturais é uma das aplicações mais desafiadoras para a reconstrução 3D, já que seus resultados exigem alta fidelidade tanto da geometria quanto da textura do modelo reconstruído. Dentro do processo de reconstrução, após as etapas iniciais de aquisição e alinhamento, os dados de profundidade precisam ser integrados formando um único modelo, onde eventuais buracos devem ser preenchidos e uma malha 3D então é gerada. Uma vez reconstruído o modelo, seguem ainda as etapas de geração de textura, simplificação e visualização do modelo 3D.

Nesse processo de reconstrução, esta dissertação se concentra nas etapas que compreendem a reconstrução propriamente dita da superfície do modelo, ou seja, as etapas de integração de vistas, preenchimento de buracos e geração da malha 3D dentro do *pipeline* desenvolvido pelo grupo IMAGO. Buscando agilizar o processo de reconstrução e torná-lo mais funcional, esta dissertação apresenta uma nova ferramenta para realizar essas etapas, aumentando a interação com o modelo 3D reconstruído e facilitando a realização de análises dos modelos gerados.

Esta dissertação também revisa as abordagens do estado-da-arte para a reconstrução 3D e avalia os métodos implementados na nova ferramenta de reconstrução proposta. Além disso, propõe-se também um conjunto de soluções para as limitações apontadas nessa dissertação para o conhecido método de reconstrução *Poisson Surface Reconstruction*.

ABSTRACT

Digital preservation of natural and cultural heritage is one of the most challenging applications for 3D reconstruction, since its results require high-fidelity of geometry and texture of the reconstructed model. Within the reconstruction process after the initial stages of acquisition and alignment, range data need to be integrated into a single model, where some holes must to be filled then a 3D mesh is generated. Further, there are steps for generating texture, followed by simplification and visualization of the 3D model.

In this reconstruction process, this dissertation focuses on steps that comprise the actual reconstruction of the surface of the model, i. e., the steps of integration, hole filling and 3D mesh generation within the pipeline developed by the group IMAGO. Seeking to accelerate the reconstruction process and make it more functional, this dissertation presents a new tool to perform these steps, to increase interaction with the 3D model and make the analysis of the generated models easy.

This dissertation also reviews the approaches of the state-of-the-art for 3D reconstruction and evaluates the methods implemented in the new tool. Moreover, it is also proposed a set of solutions to the limitations described in this dissertation to the known reconstruction method *Poisson Surface Reconstruction*.

CAPÍTULO 1

INTRODUÇÃO

A reconstrução 3D é um destacado campo de pesquisas na área de visão computacional, devido a diversidade de aplicações em que modelos 3D podem ser empregados, tais como identificação biométrica [36], filmes, jogos e preservação digital.

Algumas dessas aplicações, como a preservação digital de patrimônios naturais e culturais [33] [26], exigem que o modelo 3D reconstruído seja fidedigno ao objeto real, preservando suas características singulares. Esses modelos podem ser utilizados no desenvolvimento de museus virtuais [30] e/ou no acompanhamento da evolução do estado das obras, e ainda contribuir para restauração das mesmas [3]. Isso permite às gerações futuras compreender e contextualizar a história e a cultura dos seus povos [25].

Várias são as etapas que compreendem a reconstrução 3D de objetos. Entre os trabalhos que apresentam uma sequência pré-definida de etapas para reconstrução, ou seja, um *pipeline*, destacam-se os trabalhos de Curless *et al.* [13], Bernardini *et al.* [7] Ikeuchi *et al.* [22] e Vrubel *et al.* [47]. O último será descrito brevemente abaixo, pois apresenta o *pipeline* desenvolvido pelo grupo IMAGO que é a base para esse trabalho.

1.1 Pipeline de reconstrução 3D do IMAGO

Visando a preservação de patrimônios naturais e culturais, um *pipeline* de reconstrução 3D foi desenvolvido no grupo IMAGO[1], e foi apresentado por Vrubel *et al.* [47] [48]. Ele compreende várias etapas, como ilustrado na Figura 1.1.

O *pipeline* inicia com a aquisição de dados que é feita utilizando um *scanner* a *laser*.

[1] http://imago.ufpr.br

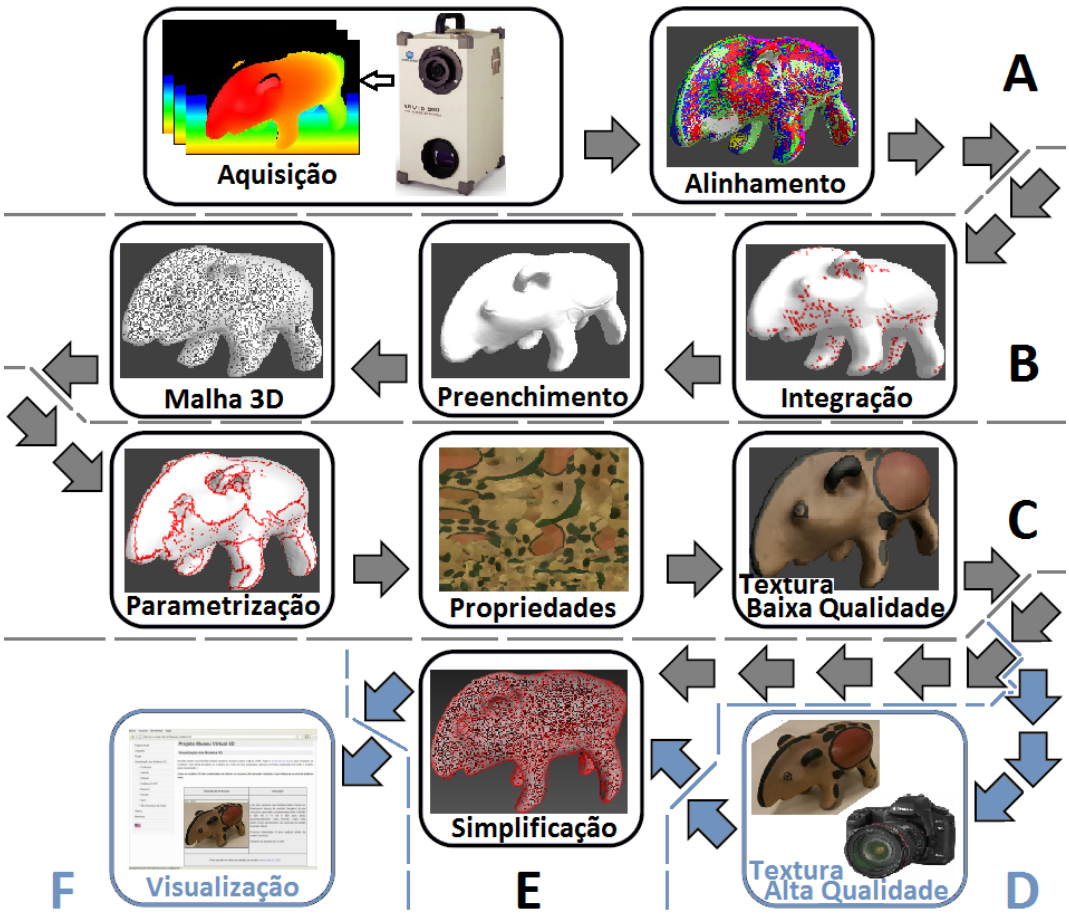


Figura 1.1: Pipeline de reconstrução do IMAGO. As etapas contidas em (a), (b), (c) e (e) representam o *pipeline* inicialmente concebido. As etapas (d) e (f) (em azul) foram incluídas posteriormente.

O *scanner* adquire imagens de profundidade de alta precisão, essas imagens são representadas por uma matriz bidimensional, com amostras regulares da imagem, onde cada elemento dessa matriz tem um valor real associado a distância naquele ponto, entre o objeto e o sensor [4]. O *scanner* adquire uma imagem de profundidade (vista do objeto) por vez, portanto várias vistas são capturadas para cobrir toda a superfície do objeto a ser reconstruído. Na segunda etapa do *pipeline*, as vistas capturadas são alinhadas em um mesmo sistema de coordenadas global. Inicialmente, elas são sobrepostas manualmente e depois pré-alinhadas usando o método proposto por Besl *et al.* [9], depois um alinhamento global automático é realizado usando o método proposto Pulli [35].

Uma vez alinhadas, as vistas precisam ser integradas para formar um único modelo. No *pipeline* do IMAGO, a integração pode ser realizada por três diferentes métodos volumétricos: VRIP [8] [13], *Consensus Surfaces* (CS) [31] [49] e IVIA, método híbrido

desenvolvido para combinar aspectos positivos dos outros dois métodos [48] [43]. Os métodos volumétricos foram implementados no *pipeline*, pois impõem menos restrições aos objetos reconstruídos; oferecem uma maneira fácil de alterar a precisão dos resultados (*voxels*); podem se adaptar facilmente a técnica de *space carving* [13], e podem trabalhar na presença de ruídos. Nesta etapa de integração, o método híbrido IVIA é o mais utilizado por alcançar os melhores resultados [43]. Ele consegue lidar com diferentes tipos de artefatos presentes nos dados capturados, gerando um modelo com geometria mais confiável.

Após a integração das vistas, por um dos métodos volumétricos, eventuais regiões não capturadas pelo sensor (por exemplo, regiões inacessíveis ao *scanner*) surgirão em forma de buracos no modelo integrado. Tais regiões não podem ser admitidas quando se busca gerar um modelo de alta fidelidade, por isso no *pipeline* proposto elas são preenchidas com o método de difusão volumétrica de Davis *et al.* [14]. Depois da etapa de preenchimento de buracos, uma malha 3D é extraída usando o algoritmo *Marching-Cubes* [28], que gera uma malha de triângulos interpolando a superfície em uma grade de cubos.

Juntamente com as imagens de profundidade, na etapa de aquisição de dados o *scanner* utilizado (*Vivid* 910 da Konica da Minolta) captura também imagens coloridas do objeto a ser reconstruído. Então, finalizada a geração da geometria, o *pipeline* segue com a geração de textura realizando a parametrização e calculando as propriedades da superfície do objeto, tais como cor e reflectância, a partir das imagens coloridas. Após a extração, a malha 3D pode ainda ser simplificada posteriormente.

Para tornar o processo de reconstrução mais ágil e funcional, o *pipeline* originalmente proposto por Vrubel *et al.* [48] está sendo desmembrado em módulos independentes, mas conectados, como mostra a Figura 1.1, e várias melhorias já foram propostas. Gomes [18] propôs melhorias para as etapas iniciais do *pipeline* (Fig. 1.1(a)), agilizando a etapa de aquisição através do uso de uma ferramenta que facilita a identificação de regiões não digitalizadas, além de tornar o pré-alinhamento automático, o que antes era realizado manualmente. A qualidade da textura gerada (Fig. 1.1(d)) também foi melhorada por

Andrade [2], que propôs a utilização de câmeras digitais para capturar as imagens coloridas, as quais eram antes capturadas em uma qualidade menor pelo *scanner*. Uma siguinificante contribuição proposta por Mendes [30], foi a criação de uma ferramenta na *Web* para visualização dos modelos 3D gerados (Fig. 1.1(f)), que funciona como um museu virtual.

1.2 Motivação e objetivos

Seguindo o contexto de continuação do *pipeline* do IMAGO, este trabalho propõe melhorias para as etapas indicadas pela Figura 1.1(b). A necessidade de melhorias nessas etapas é justificada, pois os objetos inicialmente preservados continham um volume menor de dados (pequenos e médios), no entanto, quanto maiores os objetos reais a serem preservados maior é o desafio em reconstruí-los. Com o projeto do grupo IMAGO em parceria com a UNESCO para a digitalização dos Profetas de Aleijadinho [3], o processo no *pipeline* antigo se torna menos praticável.

Para esta limitação, é proposto neste trabalho uma nova ferramenta 3D para realizar separadamente as etapas de integração de vistas, preenchimento de buracos, extração da malha 3D e coloração de vértices e geração de textura de baixa qualidade.

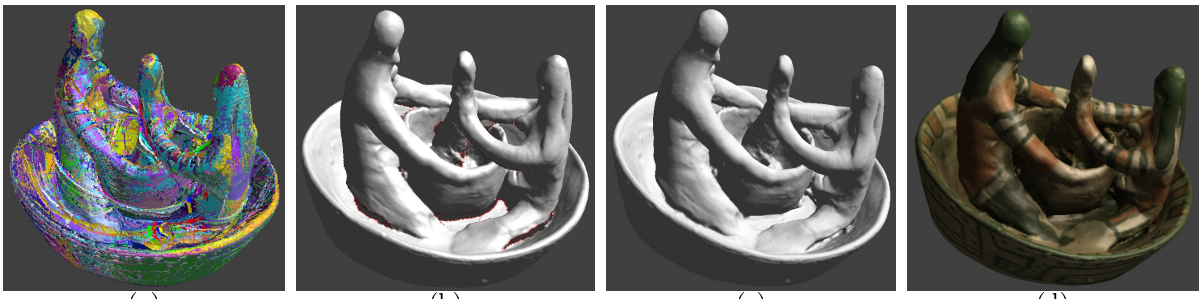
Dentre as etapas do *pipeline* do IMAGO [48], a mais afetada pelo aumento no volume de dados (pela reconstrução de objetos maiores) é a etapa de integração das vistas. Isso acontece porque os métodos volumétricos utilizados nessa etapa têm seus desempenhos quanto ao tempo e a memória relacionados ao volume dos dados, ou seja, são afetados por esse aumento.

Para superar esta limitação, foram estudados na literatura outros métodos de integração de vistas, como descrito no capítulo 2, e dentre eles o *Poisson Surface Reconstruction* (PSR) foi o escolhido para ser implementado na ferramenta proposta, por apresentar resultados satisfatórios quanto ao tempo de execução e a precisão dos modelos gerados. Como este método também possui suas limitações, este trabalho também propõe um con-

junto de soluções para superar os problemas do PSR, além de adaptá-lo as demais etapas do *pipeline* implementadas na ferramenta.

Visando tornar o *pipeline* mais funcional, agilizar o processo e garantir a fidelidade nos modelos reconstruídos, este trabalho tem por objetivos:

- Realizar um estudo sobre os métodos de integração de vistas;
- Adicionar na etapa de integração os métodos que possam contribuir para melhorar o desempenho e/ou a fidelidade dos modelos gerados, adaptando-os as etapas subsequêntes;
- Implementar uma ferramenta que realize as etapas (b) e (c) do *pipeline* (Figura 1.1) desmembrando-o. A nova ferramenta deve conter as modificações realizadas nas etapas (b) do *pipeline*, e adaptá-las para serem acopladas com as etapas da parte (c). Assim, a ferramenta deve funcionar de acordo com a Figura 1.2.



(a) (b) (c) (d)

Figura 1.2: Ferramenta para a reconstrução de superfícies: (a) entrada: vistas alinhadas; (b) modelo integrado contendo buracos (regiões em vermelho); (c) modelo com buracos preenchidos; (d) saída: malha 3D gerada do modelo integrado com buracos preenchidos e com textura em baixa resolução.

1.3 Organização do trabalho

Este trabalho está organizado como segue: no capítulo 2 é feita uma revisão sobre os métodos para reconstrução de superfícies[2] do estado-da-arte. O capítulo 3 apresenta os

[2]o termo reconstrução de superfícies se refere as etapas de integração, preenchimento de buracos e geração da malha 3D

experimentos, avaliações e comparações realizadas entre os métodos de integração VRIP, IVIA e PSR. O capítulo 4 descreve o conjunto de soluções propostas para lidar com as limitações do PSR e adaptá-lo ao *pipeline*. O capítulo 5 apresenta e avalia os resultados obtidos após a aplicação das soluções propostas no capítulo 4. A nova ferramenta de reconstrução 3D é apresentada no capítulo 6. Finalmente, as conclusões são dadas no capítulo 7.

CAPÍTULO 2

ABORDAGENS PARA RECONSTRUÇÃO DE SUPERFÍCIES

Este capítulo discute as principais abordagens utilizadas na reconstrução propriamente dita da superfície do objeto a ser preservado. A etapa mais custosa quanto ao tempo e memória é a de integração das vistas, ela também é fundamental para garantir a fidelidade dos modelos gerados. Na área de preservação digital, o modelo reconstruído deve conter até mesmo os pequenos detalhes do modelo real, por isso é importante que as vistas sejam integradas formando um modelo de alta resolução.

Devido a importância da etapa de integração, a seção 2.1 faz uma revisão dos métodos do estado-da-arte, na seção 2.2 os métodos de integração usados no *pipeline* de [48] são discutidos, assim como os utilizados nas etapas de preenchimento de buracos e extração da malha 3D. A seção 2.3 descreve um novo método para reconstrução estudado, o qual propõe-se utilizar na nova ferramenta de reconstrução.

2.1 Métodos do estado da arte para a integração

Quando o conjunto de vistas do objeto a ser reconstruído já se encontra alinhado em um único sistema global de coordenadas, segue então a etapa de integração de vistas. Diversas abordagens têm sido propostas para resolver esta etapa, cada qual possui suas vantagens e suas desvantagens. Então, seguindo a necessidade de atualizar o *pipeline* utilizado no grupo IMAGO para lidar com os desafios de preservar objetos maiores, esta seção revisa os métodos do estado da arte para a integração de vistas, visando encontrar um que se adapte a esse novo contexto.

De acordo com a taxonomía proposta por Bernardini [7] os métodos de integração podem ser: (1) baseados em Delaunay; (2) baseados em superfíces; (3) superfícies para-

métricas e; (4) métodos volumétricos. Os primeiros extraem um subcomplexo a partir do complexo de Dalaunay. Edelsbrunner [16] apresenta uma revisão deles, destacando os que se baseam em *alpha-shapes* [5]; *power crusts* [1]; *cocones* [15]; e em *eigencrusts* [24]. Têm como principal limitação a sensibilidade a ruídos e *outliers* já que interpolam os dados de entrada, além de baixo desempenho.

Seguindo, existem os métodos que criam ou manipulam as superfícies diretamente, onde cada vista define uma superfície parcial do objeto. Entre eles se destacam os algoritmos *Zippered Meshes* [27] e *Ball-pivoting* [6], além dos propostos por Soucy *et al.* [45] e Gopi *et al.* [19]. Esses métodos podem falhar em regiões de alta curvatura, gerar soluções topologicamente incorretas ou ainda superfícies ruidosas ao combinar ruídos provenientes de cada vista.

Métodos baseados em superfícies paramétricas usam forças externas e reações internas para deformar uma aproximação inicial do objeto [46] [34]. Há também abordagens que representam o modelo integrado usando uma ou mais superfícies geradas analiticamente [32] [17]. Outros métodos utilizam *level sets* [50] [52]. A maioria dessas abordagens usam conceitos de métodos volumétricos, tais como funções implícitas. É o caso do método proposto por Kazhdan *et al.* [23], que resolvem um sistema linear que representa uma equação de Poisson calculando funções base suportadas localmente. Duas novas implementações desse método, uma paralela e outra *out-of-core*, são propostas também por Kazhdan *et al.* [11] [10].

Como a maioria desses métodos assumem uma função diferencial contínua para representar a superfície, eles têm dificuldade em representar cantos ponteagudos. Outra limitação é dificuldade em encontrar um equilíbrio entre um efeito de super suavização e a não eliminação de ruídos. Eles também podem preencher buracos incorretamente, por não usar toda a informação existente nos dados de profundidade.

Métodos volumétricos criam uma representação volumétrica implícita do modelo final [20] [29] [38] [37] [21] [51]. A superfície do objeto é definida como sendo a isosuperfície cujo valor de distância é igual a 0, esta superfície é geralmente extraída usando o algo-

ritmo MC (*Marching Cubes*) [28] [12]. O cálculo da função de distância é diferente em diversos algoritmos. Curless e Levoy [13] calculam a função de distância de acordo com a linha de visão do *scanner* e utilizam pesos para avaliar a confiabilidade de cada medida. Wheeler *et al.* [49] descartam *outliers* através do cálculo de uma superfície consensual. Sagawa *et al.* proporam várias melhoras a esse método em [41] [39] [42] [40]. Vrubel [48] propôs um método híbrido que combina as vantagens do método de Curless e Levoy [13] com o método de Wheeler *et al.* [49], para detectar e eliminar ou suavizar diferentes tipos de artefatos presentes nos dados de entrada, gerando um modelo final mais confiável.

Métodos volumétricos têm a vantagem de usar todas as informações disponíveis e garantir a geração de topologias *manifold*[1] [13]. Mas têm em sua performance sua principal limitação, em termos de uso de memória e de processamento.

2.2 Abordagens presentes no *pipeline* do IMAGO

Esta seção discute as abordagens implementadas no *pipeline* de reconstrução ilustrado na Figura 1.1(c). Os métodos usados para integração de vistas são tratados na seção 2.2.1. A seção 2.2.2 descreve o método de difusão volumétrica [14] usado na etapa de preenchimento de buracos, e o algoritmo *Marching-Cubes* [28] usado para extrair a malha 3D é discutido na seção 2.2.3

2.2.1 Integração de vistas

No *pipeline* do IMAGO foram implementados três algoritmos de integração volumétrica, o algoritmo de Wheeler *et al.* [49] (*Consensus Surface*), o algoritmo de Curless e Levoy [13] (VRIP), e o algoritmo híbrido desenvolvido no grupo IMAGO [43] (IVIA). A seguir, esses métodos são descritos.

[1] Espaço topológico que localmente é similar a um espaço euclidiano.

2.2.1.1 *Consensus Surfaces* (CS)

Wheeler *et al.* [49] propuseram o uso da média de distâncias consensuais para calcular a distância com sinal para a superfície a partir do *voxel* corrente. Em regiões consensuais de várias vistas, o CS promete eliminar *outliers* e fornecer uma superfície integrada relativamente suave. Ele utiliza uma representação via *octrees*, de forma a reduzir o custo computacional e de memória. Em sua formulação original todos os *voxels* próximos da superfície estão presentes, de forma que o algoritmo *Marching Cubes* pode ser utilizado sem nenhuma modificação. Os artigos de Sagawa *et al.* [41] [39] [42] [40] aperfeiçoam este algoritmo, eliminando a restrição de que a *octree* seja dividida até o nível máximo próximo a superfície, mas requer uma adaptação do algoritmo *Marching Cubes*.

O CS busca gerar um ponto candidato para cada vista, classificando-o como em consenso ou não. Se este ponto em consenso não existir, o ponto candidato sem consenso de maior peso (ou confiabilidade) é utilizado. Essa busca pelo candidato torna o CS vagaroso, pois cada candidato deve encontrar seu ponto mais próximo em todas as outras vistas e checar se o ponto encontrado é similar a ele, o que torna a busca quadrática no número de vistas (O (n^2)). Depois de descobrir os candidatos, a distância com sinal do *voxel* é calculada, o limiar de consenso é testado com o peso acumulado, ou seja, a quantidade de medidas similares em todas as vistas para o candidato em questão.

Em seus experimentos Vrubel *et al.* [43] notaram algumas limitações do CS, tais como: execução lenta; cálculo incorreto da magnitude e da distância com sinal em regiões próximas das bordas das vistas; e quando ele preenche automaticamente os buracos, os resultados podem apresentar inconsistências.

Então, para tentar resolver as principais limitações do CS, sua implementação no *pipeline* do IMAGO continha algumas melhorias propostas:

- utilização de *voxels* de tamanho igual para alcançar a maior precisão possível, diferente da implementação original de Wheeler *et al.*;
- descarte das medidas de distância quando o ponto mais próximo em uma vista está

na borda da malha, já que o CS calcula essas medidas incorretamente;

- validação adicional nos dados pós-integração. Isso descarta *voxels* cujas distâncias não são compatíveis com a dos seus vizinhos, eliminando dados incorretos remanescentes das etapas anteriores.

Mesmo com as modificações propostas em sua implementação no *pipeline* do IMAGO, o CS ainda ainda carrega algumas limitações do algoritmo original, tais como:

- dificuldade em escolher os limiares;
- necessidade de ter todas as vistas carregadas na memória simultaneamente e;
- *outliers* podem ainda aparecer nos resultados integrados.

2.2.1.2 *Volumetric Range Image Processing* (VRIP)

O objetivo do VRIP (Curless e Levoy [13]) é calcular a distância com sinal de cada *voxel* do volume para superfície integrada, baseando-se na soma ponderada das distâncias referentes a cada vista, cujo cálculo é restrito a proximidade das superfícies, ou seja, o peso das medidas tende a zero quando a distância se aproxima das distâncias-limite. Estas distâncias são calculadas na direção da linha-de-visão do *scanner* para cada imagem de profundidade. O peso da ponderação refere-se ao grau de confiabilidade de cada ponto, então uma superfície média que representa a integração das vistas é gerada pela soma das diversas medidas

Para auxiliar a etapa de preenchimento de buracos, pode ser utilizada ainda a técnica conhecida como *space carving*. Ela consiste em usar a informação dos *voxels* que são vazios, sabe-se que eles são os que se encontram entre a superfície da vista e o sensor do *scanner*.

O VRIP executa rápido (O(n), no número de vistas) e permite a adição incremental de vistas. Além disso, ele gera uma superfície integrada mais suave, reduz o ruído e

não necessita de ter todas as vista carregadas na memória simultaneamente. Contudo, a principal limitação do VRIP é não possuir nenhum procedimento específico para descartar *outliers*, sendo difícil até mesmo reduzir o peso deles, já que cada vista é processada individualmente. Ele também integra métricas de diferentes pontos de vista, o que faz com que a representação volumétrica não seja uniforme. Além disso, uma falha faz com que artefatos sejam criados em cantos e superfícies finas, como apontado pelos autores [13]. Como o algoritmo combina valores positivos e negativos para encontrar a superfície média, isso acaba criando "declives" em superfícies finas onde um lado deve interferir no outro, aumentando assim a expessura dessas regiões.

A implementação do VRIP no *pipeline* do IMAGO, busca reduzir tanto a influência dos *outliers* quanto a falha perto de cantos e superfícies finas. Para isso, é utilizado uma nova curva assimétrica de atenuação do peso de acordo com o valor da distância com sinal.

2.2.1.3 *IMAGO Volumetric Integration Algorithm* (IVIA)

Buscando superar as limitações do VRIP e do CS o Algoritmo de Integração Volumétrica do IMAGO (IVIA) [43] foi desenvolvido combinando as forças dos dois métodos anteriores com novas ideias, para automaticamente detectar e eliminar a maioria dos artefatos existentes nos dados de profundidade. O IVIA realiza a integração em duas etapas. A primeira cria uma representação volumétrica do modelo integrado, utilizando um VRIP modificado, e a segunda usa a representação criada pela primeira para detectar e eliminar *outliers* durante a geração da integração volumétrica final.

Na primeira etapa um volume binário "vazio" também é criado para representar a operação de *space carving*. Para cada *voxel* há um *bit* correspondente em "vazio" se o *bit* é igual a 1 o *voxel* correspondente é considerado vazio (ou seja, fora do objeto). Como a distância é calculada sobre a linha de visão do *scanner*, os valores negativos são considerados "vazios" e o *bit* correspondente em "vazio" recebe o valor 1. A nova

curva de peso de distância utilizada na implementação do VRIP no *pipeline* do IMAGO é usada para ajudar a eliminar os *outliers* da superfície. Uma suavização ainda é feita para completar os *voxels* individuais com valores plausíveis e diminuir o ruído e os *outliers* da superfície. Essa primeira etapa tem complexidade linear no número de vistas, assim como o VRIP (O(n)).

A segunda etapa do algoritmo faz a integração definitiva, descartando os *outliers*. Esta etapa possui elementos do CS, como o cálculo de distância Euclidiana. Essa distância é usada para evitar que superfícies parciais sejam geradas quando as distâncias do VRIP são usadas. Além disso, as distâncias euclidianas contribuem com o algoritmo de de difusão volumétrica de Davis [14] usado no *pipeline* para o preenchimento de buracos.

O algoritmo IVIA lida com ruído e *outliers* de várias maneiras. Com a técnica de *space carving* os *outliers* distantes da superfície são eliminados. Na primeira etapa são eliminados os *outliers* próximos da superfície e as medidas fora de consenso. As medidas ainda têm sua confiabilidade avaliada através de vários parâmetros para ajudar a diminuir a influência de qualquer *outlier* que não tenha sido removido na primeira etapa. Ao contrário do CS que tem complexidade quadrática no número de vistas, o IVIA agiliza essa etapa avaliando apenas *voxels* próximos a superfície e que não setados como vazios. Então, no IVIA essa etapa tem complexidade linear (O(n)).

A principal limitação do IVIA pode ser sua performance relacionada ao tempo, sobretudo quando o volume de dados do objeto a ser integrado é grande. Isso é verificado nos experimentos apresentados no capítulo 3.

2.2.2 Preenchimento de buracos

Após a finalização da etapa de integração, o modelo integrado ainda pode conter eventuais buracos, gerados por falta de informação em algumas regiões. Isso acontece pois durante o processo de aquisição podem haver oclusões e concavidades profundas nos objetos, o que impede que a informação de profundidade seja capturada. Como o *pipeline* do IMAGO

foi desenvolvido para o escopo de preservação cultural, esses buracos não são aceitáveis, pois diminuem a fidelidade do modelo gerado ao objeto original. Então, para gerar um modelo completo uma etapa de preenchimento de buracos é realizada após a integração das vistas.

O método usado para preencher buracos é o algoritmo de difusão volumétrica de Davis [14], que apresenta resultados satisfatórios até mesmo em casos topologicamente difíceis. Abaixo, este método é brevemente descrito, pois deverá ser utilizado na nova ferramenta de reconstrução de superfícies (ver capítulo 6). O algoritmo de Difusão Volumétrica (VD) de Davis consiste em alternar etapas de difusão e de combinação com os dados originais. A difusão é obtida através de um filtro de suavização. Já que os dados acabam sendo suavizados várias vezes, o tipo de filtro acaba não interferindo no resultado. Na implementação do *pipeline* do IMAGO é utilizado o filtro proposto por Davis [14]. O VD opera sobre a representação volumétrica utilizada na etapa de integração, ele então difundi as informações dos *voxels* conhecidos para os sem informação. Para não ter que operar sobre todo o volume, apenas as regiões próximas a buracos são processadas.

O algoritmo VD consegue tratar de casos topologicamente difíceis, e tem a vantagem de garantir a geração de um resultado *manifold*, sem interpenetrações e com uma superfície suave. Outra vantagem do VD é conseguir utilizar as informações de espaços vazios ao redor do objeto, obtidas através do *space carving*.

2.2.3 Geração da malha 3D

Depois de gerar uma representação volumétrica integrada e com eventuais buracos preenchidos, o *pipeline* do IMAGO segue com a geração de uma malha poligonal. O método adotado no *pipeline* é o *Marching Cubes* MC [28].

O MC percorre ordenadamente os *voxels* da representação volumétrica gerada (os *voxels* funcionam como pontos no espaço), onde para cada cubo do volume existem 8 valores de *voxels* nos vértices do cubo, e triângulos são gerados de forma a separar os

voxels de valores positivos dos de valores negativos no mesmo cubo. Isto acaba extraindo a isosuperfície equivalente ao valor 0, que representa a superfície do objeto.

A implementação do *Marching Cubes* no *pipeline* foi otimizada de forma que apenas 4 fatias do volume precisassem estar simultaneamente carregadas na memória. Diferentemente, a implementação original requeria que todo o volume estivesse carregado na RAM, o que é impraticável para objetos muito grandes. A grande vantagem do MC é seu rápido desempenho, pois apesar da complexidade dos diversos casos, a maioria das decisões ficam codificadas em tabelas (*look-up tables*).

2.3 Poisson Surface Reconstruction (PSR)

Diante do novo escopo de objetos a serem preservados pelo grupo IMAGO e das limitações existentes, sobretudo nos métodos de integração do *pipeline* de reconstrução, buscou-se então um outro método para agilizar essa etapa do *pipeline* e manter a fidelidade nos modelos gerados. Existem diversas abordagens para realizar a integração de vistas, como mostrado na seção 2.1. Dentre elas, destaca-se o método de integração baseado em superfícies paramétricas *Poisson Surface Reconstruction* (Reconstrução de Superfície Poisson PSR) [23]. O PSR pode ser visto como uma aternativa a mais no *pipeline*, além dos métodos volumétricos já existentes.

A motivação para estudar esse método é que ele é considerado o método do estado-da-arte para a reconstrução 3D. Além disso, ele apresenta resultados superiores quando comparado com outras abordagens e bom desempenho tanto em termos de uso de memória e tempo de processamento [43]. Por isso, esse método é descrito a seguir para melhor compreender a abordagem proposta pelos autores.

Kazhdan *et al.* [23] expressam a reconstrução da superfície como a solução para uma equação de Poisson. Para isso, eles abordam o problema usando uma estrutura de função implícita, onde a ideia é calcular uma *função indicadora* χ, cujo valor é igual a um dentro da superfície e zero fora. A superfície do objeto é reconstruída através da extração de

uma isosuperfície apropriada.

A ideia chave do PSR é que existe uma relação integral entre uma amostra de pontos orientados $\overrightarrow{V}$ a partir da superfície de um modelo e a função de indicadora desse modelo χ_M, que é a possibilidade dessas amostras serem vistas como amostras do gradiente da função indicadora do modelo ∇_{χ_M}. Essa relação se dá pois o gradiente da função indicadora é um campo vetorial, cujo valor é zero em quase todos os lugares menos em pontos próximos à superfície, onde o valor é igual a normal da superfície interna. A Figura 2.1 retirada de [23], traz uma ilustração bidimenssional do PSR.

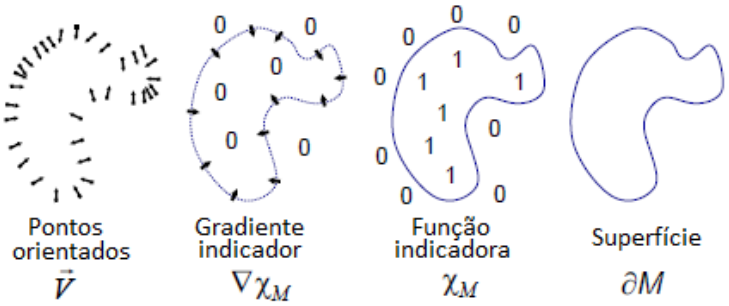


Figura 2.1: Ilustração intuitiva do PSR em 2D

Os autores resolvem a função indicadora invertendo o operador de gradiente, encontrando a função escalar cujo gradiente melhor corresponde ao campo vetorial $\overrightarrow{V}$, um problema que representa uma variação de uma equação de Poisson padrão (equação 2.1). Então, eles definem um espaço de funções com alta resolução perto da superfície do modelo e baixa longe dela, expressando o campo vetorial $\overrightarrow{V}$ como uma soma linear de funções neste espaço.

$$\Delta\chi \equiv \nabla.\nabla\chi = \nabla.\vec{V}. \tag{2.1}$$

Como uma representação precisa da função implícita é necessária apenas perto da superfície reconstruída, o PSR usa uma *octree* adaptável tanto para representar a função implícita, quanto para resolver a equação de Poisson. Para extrair a isosuperfície da função indicadora, o usa-se um método similar ao *Marching Cubes*, no entanto adaptado

para *octrees*.

O PSR tem bom desempenho em termos de uso de memória e tempo de processamento, e tem como benefício a facilidade em aumentar a resolução, aumentando a profundidade da *octree*. Contudo, com o aumento de cada nível da *octree* o tempo de execução e o uso de memória crescem aproximadamente em um fator de 4, juntamente com o número de triângulos do modelo final gerado. No entanto, apesar de sua natureza quadrática ($O(n^2)$), o PSR é escalável o que o faz ser mais ágil em reconstruções de alta resolução, principalmente comparado aos métodos volumétricos [23].

Assim como os outros métodos de integração, o PSR também carrega algumas limitações, tais como: a dificuldade em encontrar um equilíbrio entre um efeito de suavisação excessiva e a não eliminação de ruídos, a não garantia da geração de topologias *manifold*. O PSR também não utiliza todas as informações disponíveis (tais como a linha de visão do *scanner*), o que pode provocar conexões incorretas de algumas regiões, um problema apontado pelos autores [23]. No capitulo 3 uma análise do PSR é feita, onde essas vantagens e desvantagens são abordadas.

CAPÍTULO 3

EXPERIMENTOS

Este capítulo apresenta os experimentos realizados afim de comparar a performance dos métodos discutidos no capítulo 2. Como esse trabalho se insere no contexto da preservação natural e cultural de objetos, nestes experimentos são avaliados não apenas os desempenhos em relação ao tempo e memória, mas sobretudo a fidelidade dos resultados gerados. Ressalta-se que as comparações aqui realizadas são qualitativas, já que não são conhecidas métricas para comparar quantitativamente resultados de reconstrução. Essa avaliação compõe um artigo vencedor do *Best Paper* da Conferencia Internacional em Análise e Processamento de Imagens (ICIAP) [43] e outro publicado no Workshop de Teses e Dissertações do SIBGRAPI [44].

Os experimentos buscam avaliar como as vantagens/desvantagens teóricas de cada método (vistos no capítulo 2), afetam positiva ou negativamente os objetos em casos reais. Como já demonstrado em alguns trabalhos [48] [43] [44], o algoritmo *Consensus Surfaces* de Wheeler *et al.* [49] não consegue atingir resultados precisos comparado a outros métodos do estado-da-arte. Portanto, nesse capítulo ele não é avaliado, visto que ele também não consta na ferramenta de reconstrução desenvolvida.

Para os experimentos realizados, foram utilizados objetos do patrimônio cultural e natural, afim de avaliar os métodos em casos reais que fazem parte dos projetos de pesquisa desenvolvidos pelo grupo IMAGO. A base de dados é composta de objetos de arte indígena (do Museu Metropolitano de Curitiba), fósseis (do Museu de Ciências Naturais da UFPR), obras-primas barrocas (Profetas de Aleijadinho), além de objetos pessoais que representam situações desafiadoras.

Inicialmente, os métodos são comparados quanto à eficiência temporal, na seção 3.1.

Depois, os métodos já implementados no *pipeline* do IMAGO são avaliados individualmente nas seções 3.2 e 3.3, respectivamente. A seção 3.4 avalia as vantagens de desvantagens do método estudado PSR.

3.1 Eficiência temporal

A Tabela 3.1 apresenta a comparação da eficiência temporal de cada um dos métodos de integração. Partindo da esquerda: nome do modelo; material que é composto; volume de dados em MB; número de faces triangulares do modelo final; tempo da etapa de integração em segundos para cada um dos métodos.

Tabela 3.1: Comparação da performance temporal.

Modelo	Material	Dados	#Triang.	VRIP	IVIA	PSR
Anta	cerâmica	86.5	349.985	312.5	494	241
Bote	cerâmica	114	409.984	301.1	394	277
Indias	cerâmica	183.3	619.990	512	1025	452
Pato	cerâmica	136.6	318.314	227	900	239
Zoolito	cerâmica	99.5	105.871	121	498	77
Protocyon	fóssil	188	562.221	860	1200	780
Cypreacassis	concha	135	685.719	227	985	498
Padre	cerâmica	48.8	99.840	43.1	77.6	43
Shrek	plástico	66.1	199.565	127	893	98
Profeta Joel	pedra-sabão	482.4	1.216.825	1097	6372	1072
Profeta Daniel	pedra-sabão	773.5	2.9320.06	2068	8527	1610

A eficiência temporal do VRIP, apontada como vantagem (seção 2.2.1.2) é confirmada pela Tabela 3.1, principalmente quando comparado ao desempenho do IVIA. No entanto, percebe-se que o tempo começa a aumentar consideravelmente a medida que a precisão aumenta. Já o tempo do IVIA não se distancia muito do VRIP para objetos menores, contudo com o aumento no volume de dados, esse tempo de execução se torna expressivamente maior. Para modelos maiores como os profetas de Aleijadinho, o tempo de execução cresce demais tornando o método vagaroso.

A eficiência temporal do PSR também foi comprovada. Ele conseguiu os melhores resultados em quase todos os modelos se aproximando do VRIP em modelos menores,

mas com tempos bastante superior ao IVIA. O bom desempenho temporal do PSR fica mais evidente ao reconstruir grandes volumes de dados em alta resolução, isso pode ser notado já para o modelo do *profeta daniel*.

Afim de evidenciar ainda mais os bons resultados do PSR, a Tabela 3.2 traz o tempo de execução para os cinco profetas até então reconstruídos em alta resolução (partindo da esquerda: nome do modelo; volume de dados em MB; número de faces triangulares do modelo final; tempo da etapa de integração em segundos). A combinação tempo de execução e uso de memória é consideravelmente melhor do que qualquer método volumétrico, porque além dos baixos tempos verificados na Tabela 3.1 o pico de memória nesses casos não ultrapassou 3.1GB.

Tabela 3.2: Performance temporal do PSR na geração de modelos de alta resolução.

Modelo	Dados	#Triang.	Tempo
Joel	482.4	3.650.478	2317
Oseias	423	5.701.350	3324
Jonas	584.3	7.482.166	3972
Ezequiel	532	7.811.600	4269
Daniel	773.5	8.796.020	4832

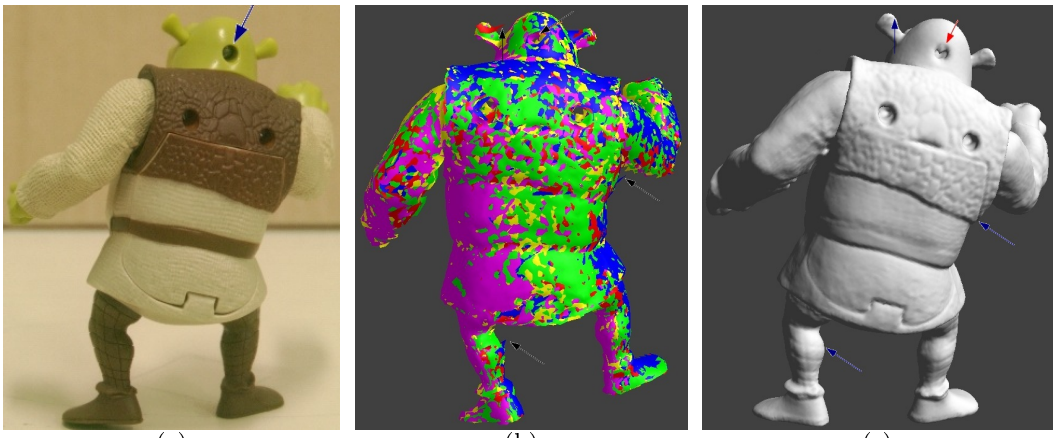
Esta seção comprovou boa eficiência temporal do VRIP e o baixo desempenho do IVIA. Também notou-se que o PSR é capaz de gerar modelos de alta resolução com um desempenho em relação a tempo e memória consideravelmente superiores aos métodos volumétricos.

3.2 Avaliação do VRIP

O primeiro método a ser avaliado é o VRIP de Curless e Levoy [13], discutido na seção 2.2.1.2. Assim, objetos com volume de dados (imagens de profundidade e coloridas capturadas pelo *scanner*) variados foram usados nesse primeiro teste.

A principal vantagem apontada é quanto a seu desempenho em relação ao tempo, que já foi avaliado na seção 3.1. Então, essa seção se dedica a avaliar as desvantagens do método. A primeira delas, apontada na seção 2.2.1.2, é a não existência de um procedimento

para descartar *outliers*. Na implementação do método no *pipeline* do IMAGO, buscou-se reduzir a influência deles, porém os testes realizados mostram que mesmo assim *outliers* ainda podem "sobreviver"(como mostra a Figura 3.1). Isso afeta negativamente não apenas a etapa de integração, mas também outras etapas do processo de reconstrução, como discutido abaixo.

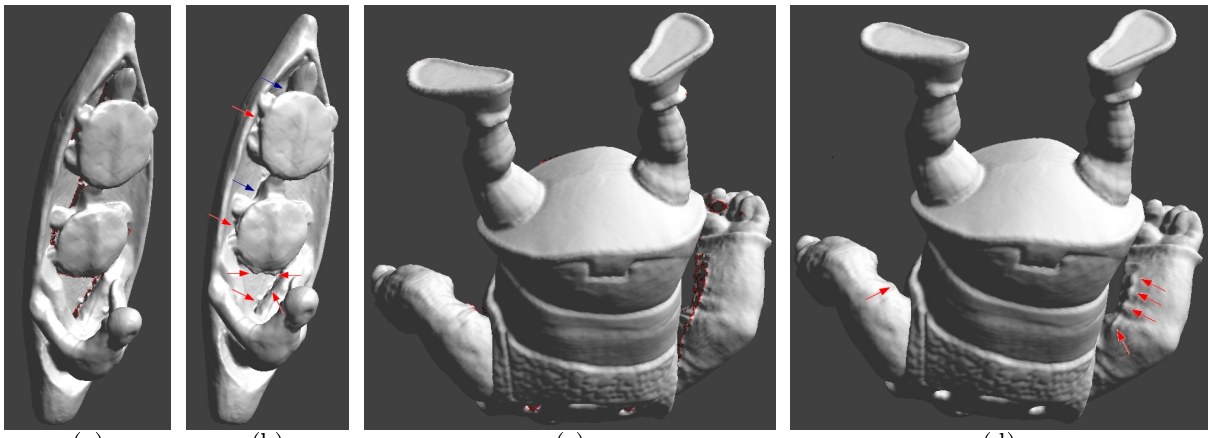


(a) (b) (c)

Figura 3.1: Eliminação de *outliers* do VRIP: (a) vista por trás do objeto *shrek* capturado; (b) Imagens de profundidade do objeto alinhadas, onde é possível ver alguns dados incorretos retornados pelo *scanner* (flechas pretas); (c) objeto reconstruído, onde nota-se que nem todos os dados incorretos foram eliminados pelo VRIP (flecha vermelha).

Exemplos do efeito danoso da não eliminação de *outliers* do VRIP, são ilustrados na Figura 3.2. Nela, após a integração dos modelos os buracos existentes são preenchidos usando o método de difusão volumétrica (VD) [14]. O VRIP busca contribuir com essa etapa efetuando a técnica *space carving*, e essa contribuição foi verificada durante os experimentos. Mas, como nota-se (Figuras 3.2b e 3.2c setas vermelhas) a não eliminação de *outliers* nas regiões das bordas dos buracos compromete o desempenho do VD. É perceptivel que a maior parte dos buracos deu lugar a regiões que preenchidas se apresentam como "protuberâncias" no modelo final, que não se aproximam do objeto real. Outro exemplo de preenchimento incorreto pode ser visto na Figura 3.4g.

Além do efeito danoso da não eliminação de *outliers* pelo VRIP, outra limitação preocupante é quanto a "falha" mencionada na seção 2.2.1.2. Durante os experimentos notou-se que realmente, em alguns casos, o VRIP gera "declives" em cantos finos, onde eles não deveriam existir. Um desses casos é ilustrado na Figura 3.3, onde é possível ver claramente



(a) (b) (c) (d)

Figura 3.2: Preenchimento de buracos pós-VRIP: (a) objeto *bote* integrado pelo VRIP, onde os buracos estão circulados em vermelho; (b) objeto *shrek* após o preenchimento de buracos. Alguns buracos (flechas azuis) foram bem preenchidos, no entanto protuberâncias (flechas vermelhas) surgiram no modelo após o preenchimento de alguns buracos; (c) outro modelo integrado pelo VRIP, onde os buracos estão circulados em vermelho; (d) o mesmo problema em (b) se repete neste modelo (flechas vermelhas).

no canto do pergaminho do *profeta joel* a discordância entre o objeto real e o modelo gerado pelo VRIP.

(a) (b)

Figura 3.3: Problema gerado pela falha do VRIP: (a) foto que ilustra a base e o pergaminho do *profeta joel* com detalhe para os cantos (flechas azuis); (b) modelo gerado pelo VRIP. As flechas vermelhas destacam a "dobra" gerada incorretamente pelo VRIP.

A Figura 3.4 ilustra outros objetos listados na Tabela 3.1. Bons resultados podem ser vistos nas Figuras: 3.4a *padre*; 3.4b *protocyon*; 3.4c e 3.4d *cypreacassis*; 3.4e *zoolito*. Destaque para os problemas gerados por *outliers* nas Figuras: 3.4f *anta*; e 3.4g *indias*.

Resultados do VRIP para o *profeta daniel* são ilustrados na Figura 3.5. O VRIP

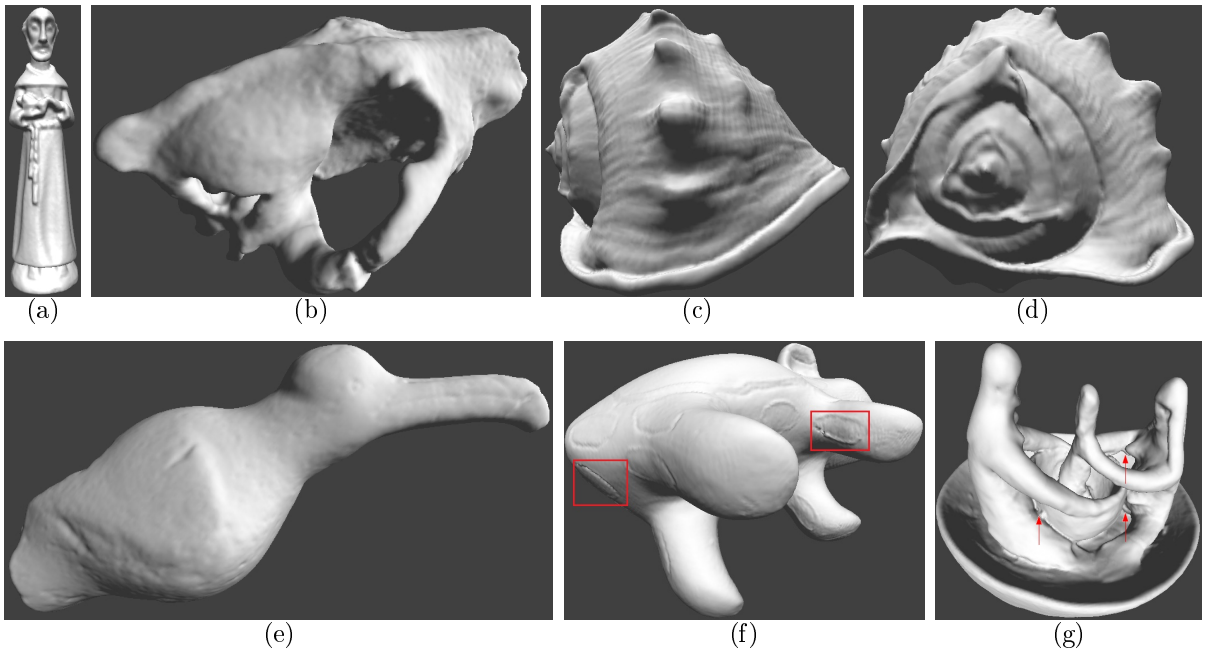


Figura 3.4: Outros resultados para o VRIP.

alcança bons resultados quando não há *outliers* (flechas azuis). As flechas vermelhas ilustram os resultados incorretos, provocados pelos *outliers*.

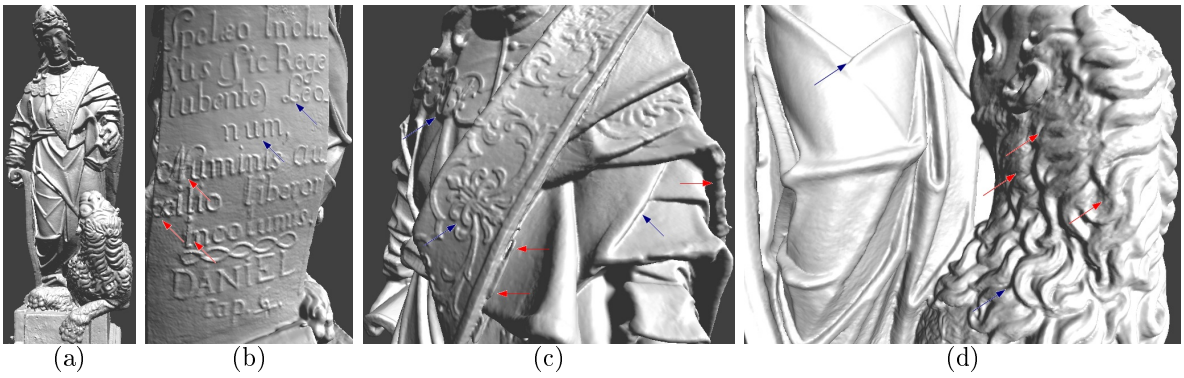


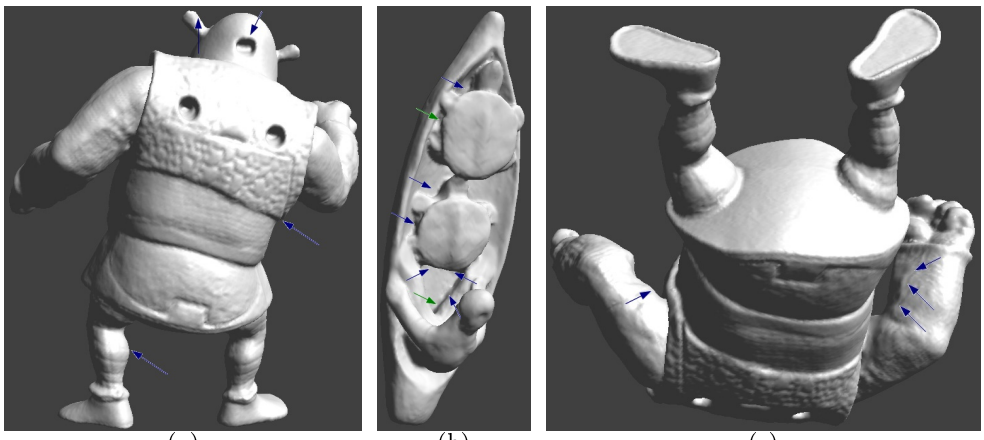
Figura 3.5: Resultados do VRIP: *profeta daniel*.

Os experimentos mostrados nessa seção confirmam as desvantagens descritas na seção 2.2.1.2. No contexto de preservação digital é certo que a precisão e fidelidade dos resultados tem relevância maior do que a eficiência temporal. Assim, como demostrado nessa seção, o VRIP gera resultados que podem-se dizer não muito "aceitáveis", principalmente quando o objeto é preservado com o intuito de acompanhar a sua condição ao passar dos anos, como é o caso dos profetas.

3.3 Avaliação do IVIA

Essa seção avalia o método IVIA desenvolvido no grupo IMAGO [48], discutido na seção 2.2.1.3. A principal vantagem do IVIA apontada é a sua capacidade de lidar com diferentes artefatos presentes nos dados de profundidade, combinando estratégias para a eliminação de *outliers*. Como o IVIA é um método híbrido que utiliza o VRIP em seu primeiro passo, a principal preocupação foi verificar se os problemas do VRIP ainda eram identificados no IVIA.

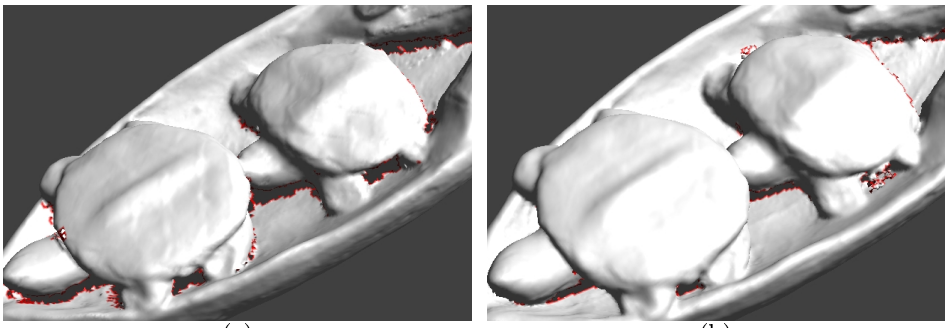
Na Figura 3.6a nota-se que o modelo reconstruído não contem a imperfeição existente no modelo gerado pelo VRIP. O IVIA parece também contribuir com o algoritmo de preenchimento de buracos VD, já que as "protuberâncias" vistas no modelo *bote* não existem, como ilustrado na Figura 3.6b. Na Figura 3.6c bons resultados também são vistos, sobretudo no braço direito do *shrek* onde o preenchimento parece bem mais aceitável que o realizado pelo VRIP.



(a) (b) (c)

Figura 3.6: Análise do IVIA: (a) modelo correspondente a Figura 3.1a, gerado sem a imperfeição vista em Figura 3.1c ; (b) modelo equivalente a Figura 3.2a, com os buracos preenchidos. Note-se que a maioria dos buracos foram bem preenchidos (flechas azuis) ou os erros encontrados na Figura 3.2b foram atenuados (flechas verdes); (c) modelo equivalente a Figura 3.2c com buracos preenchidos. Nota-se que o resultado após o preenchimento de buracos é mais aceitável do que o visto na Figura 3.2d.

Os bons resultados alcançados pelo IVIA são explicados pela sua detecção e eliminação de *outliers*. Além disso, como apontado em [48] o IVIA atribui pesos menores a regiões de bordas, já que os dados ai encontrados podem não ser confiáveis. Isso leva a eliminação de

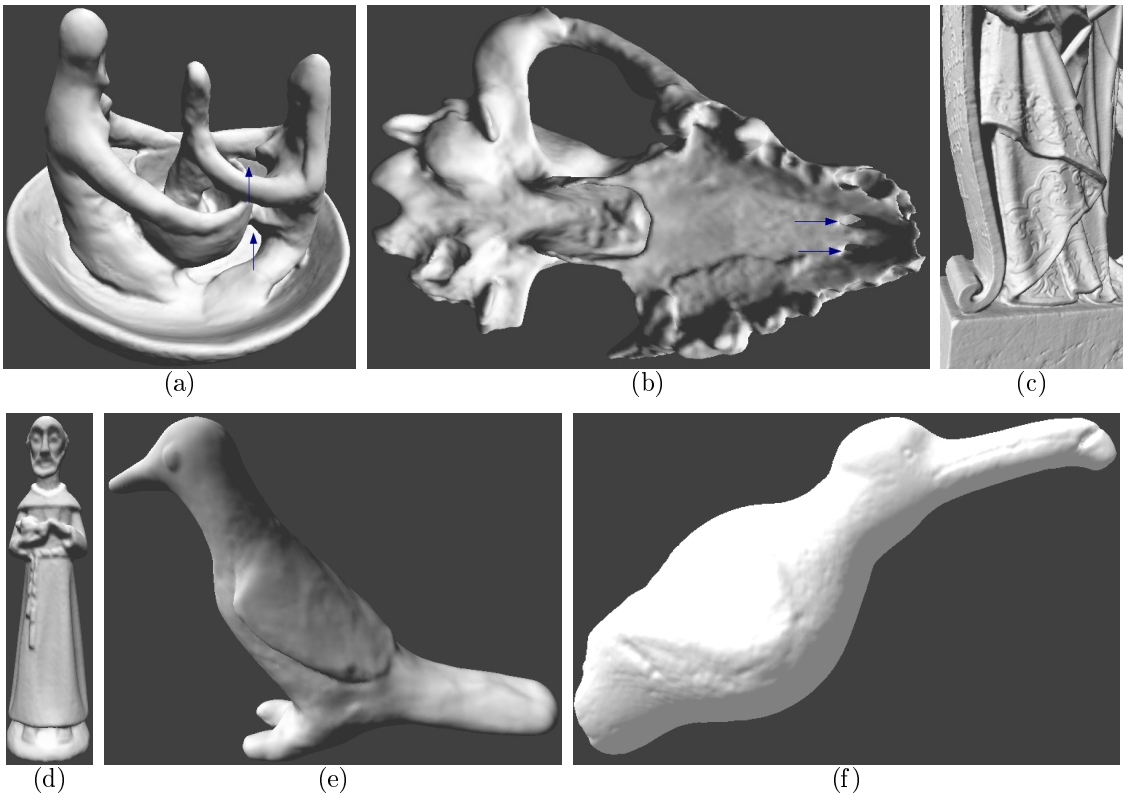


(a) (b)

Figura 3.7: Eliminação de bordas: (a) buracos gerados quando o IVIA foi usado. Como ele elimina dados em regiões não confiáveis (como bordas), os buracos gerados são um pouco maiores; (b) buracos gerados quando o VRIP foi usado.

pedaços das bordas, gerando buracos maiores no modelo, como ilustra a Figura 3.7. No entando, esse procedimento ajuda o algoritmo VD, como verificado na Figura 3.6 uma vez que as regiões que serão difundidas durante o preenchimento não contém dados incorretos.

Outros resultados para o IVIA são ilustrados na Figura 3.8: 3.8a *indias* e 3.8b *protocyon*, com destaque para o bom preenchimento de buracos; 3.8c detalhe da veste do *Profeta Joel*; 3.8d *padre*; 3.8e *pato*; 3.8f *zoolito*.



(a) (b) (c)

(d) (e) (f)

Figura 3.8: Outros resultados para o IVIA.

As vantagens do IVIA descritas na seção 2.2.1.2 são confirmadas por esse experimentos. No entanto, verificou-se também que ele possui um conjunto extenso de parâmetros interrelacionados, dos quais sua precisão na eliminação de *outliers* depende.

3.4 Avaliação do PSR

Durante os testes com os dois métodos presentes no *pipeline* do IMAGO, algumas limitações se tornaram evidentes. O VRIP executa rápido, porém pode criar modelos com certas imperfeições que prejudicam a fidelidade ao objeto real. O IVIA gera modelos mais fiéis, mas sua eficácia está ligada a escolha correta de parâmetros e seu tempo de execução se aproxima do improdutivo quando o volume de dados do objeto capturado é muito grande.

Diante das limitações observadas, buscou-se então um método capaz de combinar a eficácia do IVIA na eliminação de *outliers* e a rapidez na execução. Assim, na seção 2.3 foi discutido o método de reconstrução de superfícies Poisson (PSR), e esta seção traz os resultados de alguns experimentos com o PSR.

Durante os experimentos, observou-se que ele realmente consegue eliminar os diferentes artefatos que podem estar presentes nos dados de profundidade. Isso pode ser notado nos modelos ilustrados na Figura 3.9. As flechas azuis destacam regiões com alto nível de detalhes. Figura 3.9a *padre*; 3.9b *bote*; 3.9c detalhe para região das costas das tartarugas do *bote*; 3.9d e 3.9e *indias*; 3.9f e 3.9g *anta*; 3.9h, 3.9i e 3.9j *cypreacassis*.

Também foi notado que o PSR é bastante sensível a alinhamentos incorretos, ou seja, é preciso garantir que as vistas do objeto estejam bem alinhadas antes do conjunto de pontos ser passado a ele. Isso é ilustrado nas Figuras 3.10d e 3.10f, onde o mal alinhamento provocou resultados incorretos (flechas vermelhas). As Figuras 3.10a, 3.10b, 3.10c e 3.10d, ilustram o *profeta joel*. As Figuras 3.10e e 3.10f ilustram o *profeta oseias*.

Como já dito, o PSR é eficaz na eliminação de artefatos presentes nos dados de profundidade, e também executa rápido comparado a outros métodos, principalmente diante

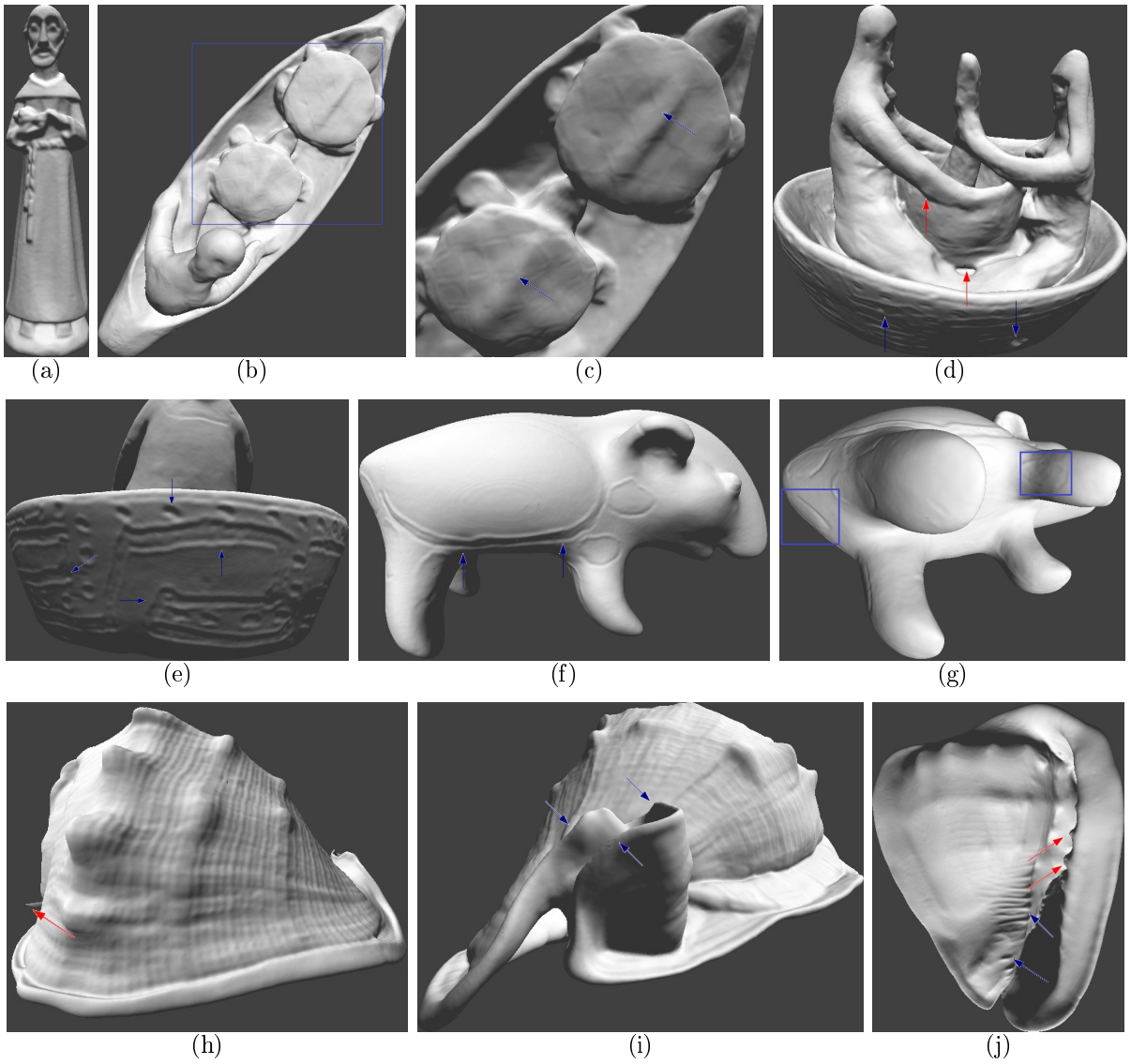


Figura 3.9: Diferentes resultados para o PSR.

de grandes volumes de dados. Contudo, os testes também confirmaram algumas limitações. A primeira observada foi quanto ao preenchimento de buracos, como ilustra a Figura 3.12. Como o PSR não utiliza a informação da linha de visão do *scanner* ele acaba preenchendo incorretamente alguns buracos. Esse preenchimento incorreto não acontece quando o algoritmo VD é usado na etapa de preenchimento, porque ele utiliza toda a informação disponível que possa ser extraída dos dados de profundidade para auxiliá-lo durante o preenchimento.

Sabe-se também que outra limitação do PSR, assim como todos os métodos baseados em superfícies paramétricas, é a não garantia de geração de uma malha *manifold*. Isso é

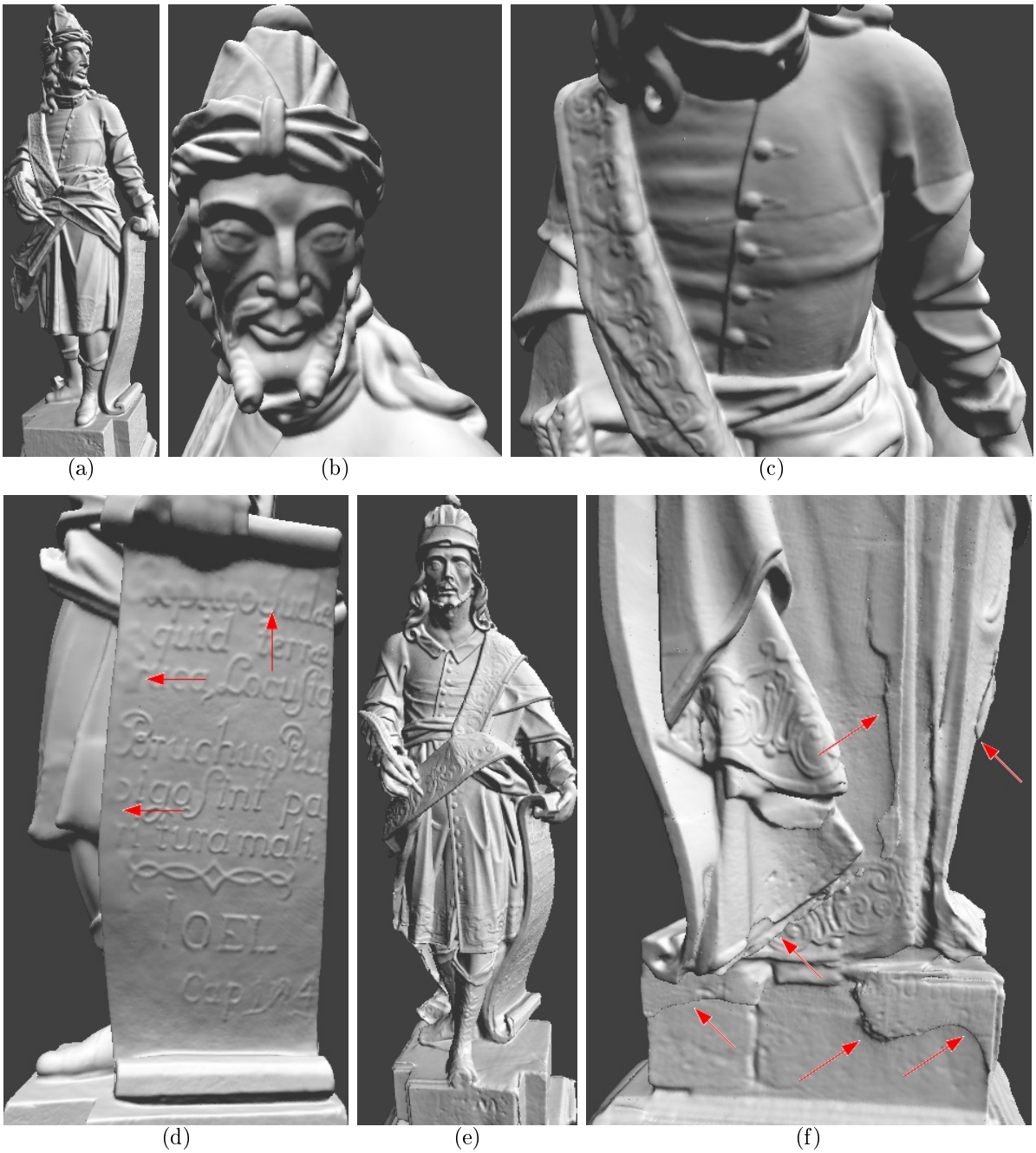
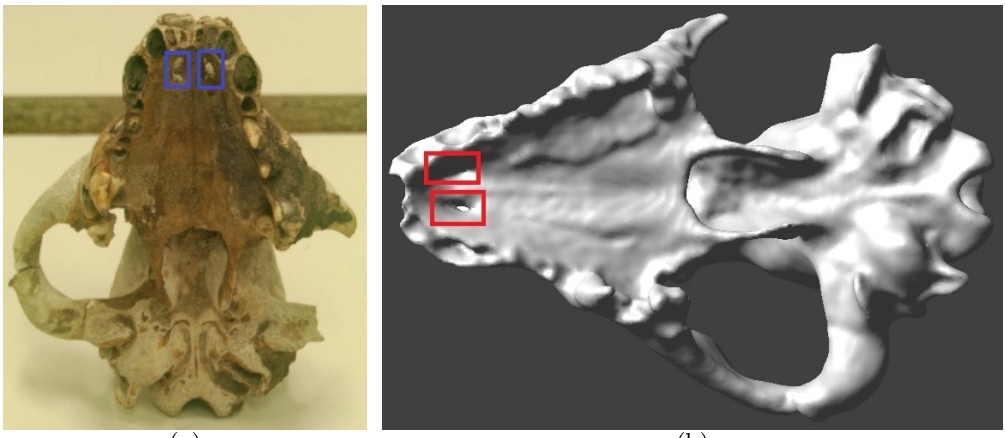


Figura 3.10: Outros resultados para o PSR.

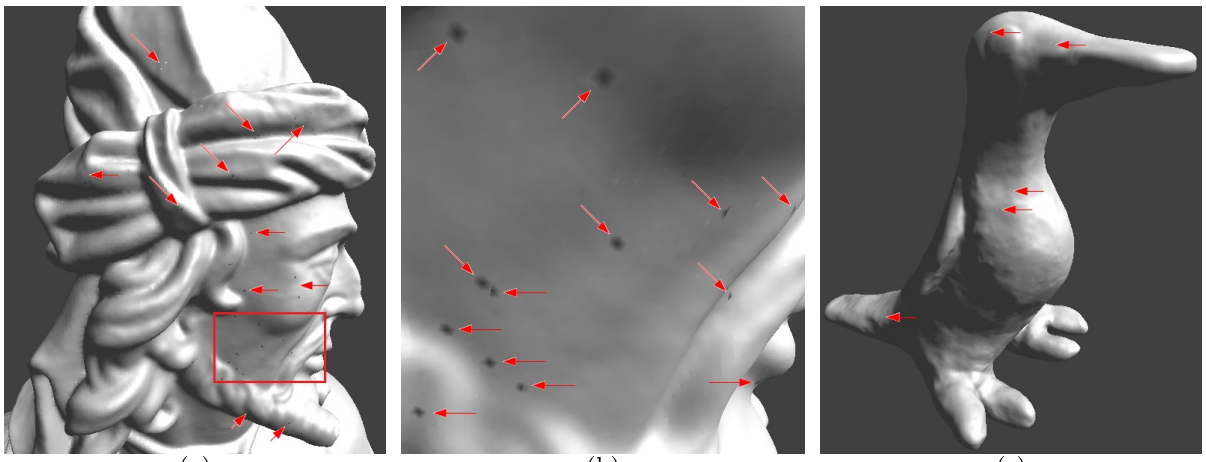
uma dificuldade, pois as etapas que seguem a geração de malhas no *pipeline* do IMAGO (geração de textura, por exemplo) trabalham com a garantia de que uma malha *manifold* foi gerada previamente, se isso não pode ser garantido, essas etapas devem falhar. Durante os experimentos notou-se que para conseguir gerar malhas *manifold* era preciso aumentar o valor de um parâmetro de suavização. No entanto, essa alteração provoca um efeito de super-suavização, como pode ser notado na Figura 3.12a. Notou-se também que mesmo



(a) (b)

Figura 3.11: Preenchimento de buracos do PSR: (a) vista capturada do objeto real, com destaque (retângulos azuis) para os buracos existentes; (b) objeto gerado pelo PSR, com destaque para os buracos preenchidos incorretamente (retângulos vermelhos).

quando o PSR gera malhas *manifold* alguns *outliers* em forma de vértices podem surgir no modelo final. Em objetos menores esses *outliers* são menos evidentes e aparecem em menor número espalhados por todo o modelo. Contudo em objetos maiores como o *profeta joel*, eles se tornam muito mais evidentes e em maior número por todo o modelo.



(a) (b) (c)

Figura 3.12: *Outliers* gerados pelo PSR: (a) face do *profeta joel* com diversas imperfeições (flechas vermelhas); (b) detalhe da região destacada (retângulo) de (a), onde os vértices incorretos podem ser notados mais nitidamente; (c) mesmo em menor proporção *outliers* em forma de vértices também aparecem em modelos menores, como o pato ilustrado.

Desconsiderando os *outliers* gerados, o PSR gera modelos bastante precisos e com um nível de detalhes consideravelmente melhor comparado aos outros métodos. Esses resultados podem ser vistos na Figura 3.13, onde as flechas azuis os evidenciam os detalhes no modelo do *profeta daniel*.

Figura 3.13: Resultados para o PSR: *profeta daniel*.

Os experimentos relatados nessa seção ressaltam a capacidade do PSR de eliminar artefatos existentes nos dados de entrada. Também são identificadas suas principais limitações. Portanto, algumas melhoras precisam ser feitas sobretudo para garantir a geração de malhas *manifold*, evitar o efeito de super-suavização e os eventuais *outliers* gerados e principalmente auxiliar o PSR no processo de preenchimento de buracos. O conjunto de estratégias empregados para superar essas limitações são descritos no capítulo 4.

CAPÍTULO 4

IMPLEMENTAÇÃO DE SOLUÇÕES PARA O PSR

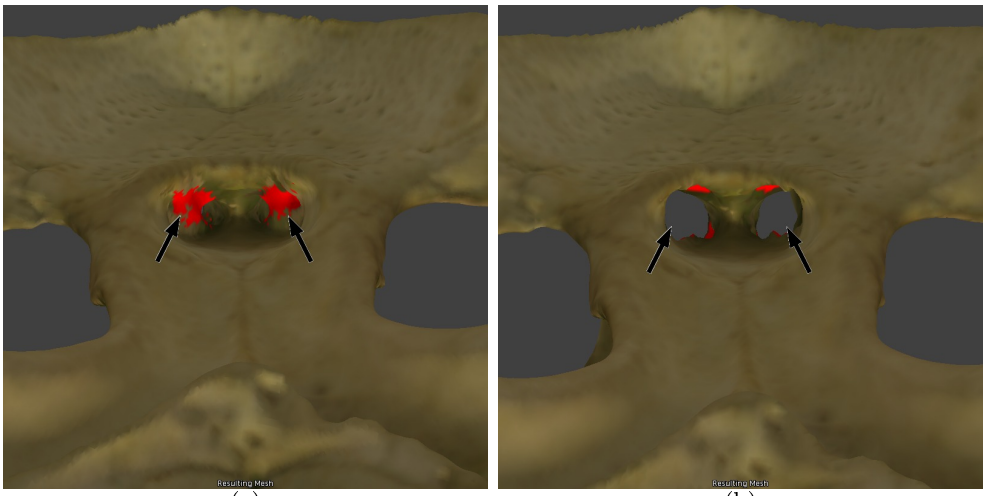
Seguindo o objetivo principal deste trabalho, que é o desenvolvimento de uma ferramenta para realizar as etapas (b) e (c) do *pipeline* de reconstrução ilustrado na Figura 1.1, os capítulos anteriores revisaram as técnicas para integração de vistas e as abordagens já implementadas no *pipeline* do IMAGO (ver capítulo 2), e apresentaram os experimentos comparativos entre as técnicas já implementadas no *pipeline* e o PSR (ver capítulo 3), o qual busca-se incorporar na nova ferramenta. A partir disso, identificadas as limitações do PSR, neste capítulo apresentam-se as técnicas utilizadas na tentativa de solucionar essas limitações, e adaptar o PSR à sequência do *pipeline* na nova ferramenta de reconstrução.

Neste capítulo primeiramente a seção 4.1 descreve o problema principal do PSR. A solução inicial proposta é descrita na seção 4.2. As seções 4.3, 4.4 e 4.5 descrevem as técnicas utilizadas afim de superar os problemas do PSR e gerar um modelo final com maior fidelidade ao objeto real reconstruído. Por fim, a seção 4.6 resume em um algoritmo o conjunto de soluções adotadas.

4.1 Descrição do problema

Inicialmente, o preenchimento incorreto de alguns buracos foi identificado como principal problema do PSR (seção 3.4). Isso ocorre pois, ele não incorpora nenhuma informação associada ao processo de aquisição (como a linha de visão do *scanner*), assim acaba interpolando através de regiões sem dados, pois tais regiões não acrescentam amostras para o cálculo da função indicadora (seção 2.3). Essas regiões desconsideradas podem então aparecer no modelo final erroneamente conectadas, já que é a partir da função indicadora do modelo que o PSR gera uma malha 3D, através da extração de um isovalor.

Esse problema é apontado pelos próprios autores [23] e a Figura 4.1 ilustra um caso típico de preenchimento incorreto.



(a) (b)

Figura 4.1: Preenchimento incorreto: (a) PSR; (b) modelo corretamente preenchido. Os pontos em vermelho correspondem a região preenchida.

4.2 A ideia inicial

Considerando o problema descrito, buscou-se então uma forma de acrescentar ao PSR as informações provenientes do processo de aquisição, de forma a orientá-lo em regiões onde dados não foram capturados. A ideia inicial consistia basicamente em passar ao PSR a informação de espaços vazios, extraída a partir da linha de visão do *scanner*. Essa informação faria parte do arquivo de entrada do PSR (arquivo com pontos alinhados), que então além do conjunto de pontos e normais teria também um conjunto de pontos "vazios", ou seja, pontos que estão no espaço vazio entre a superfície do modelo e o ponto de vista do *scanner*. A Figura 4.2 ilustra essa ideia.

Estes pontos vazios foram criados da seguinte maneira:

1. Primeiramente, para cada vista i do modelo capturado, pontos "vazios" p_{vazio} foram gerados para cada ponto "real" p existente na superfície. Assim, dado um ponto p pertencente a uma vista i, calcula-se o vetor unitário que aponta de p para v, onde v corresponde ao ponto de vista do *scanner* para aquela determinada vista (imagem);

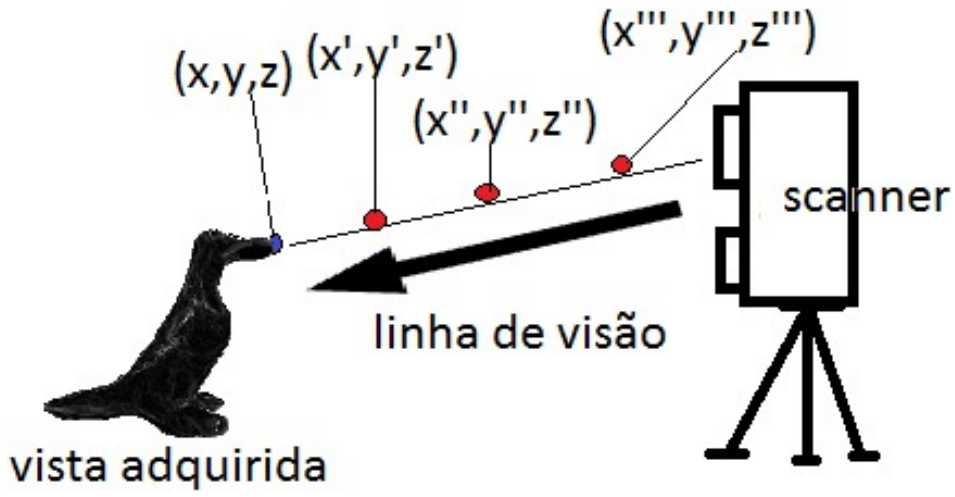


Figura 4.2: Geração de pontos "vazios". Para cada ponto real (em azul), um conjunto de pontos (em vermelho) seriam criados para representar o espaço vazio entre a superfície do modelo e o *scanner*.

2. Em seguida, foram gerados de 1 a n pontos "vazios", seguindo alguma regra de espaçamento. Durante experimentos, os p_{vazios} foram gerados a partir do valor de erro do *scanner*. Como as coordenadas dos pontos p estão em milímetros, o *scannerError* é em milímetros também, e o vetor direção que aponta de p para v (chamemos de d) é unitário, para gerar os pontos "vazios" bastava fazer: $p_{vazio} = p + d * fator$, onde *fator* seria 2**scannerError*, 4**scannerError*, e assim por diante;

3. Depois de calculados, cada p_{vazio} é multiplicado pela matriz que o converte do sistema de coordenadas local da vista para o sistema de coordenadas global (resultante do alinhamento das vistas).

Portanto, o arquivo de entrada para o PSR passa a conter para cada ponto "real" alguns pontos "vazios", gerados a partir da linha de visão do *scanner*, e que representam o espaço vazio entre a superfície do modelo e o *scanner*. A partir dessa interferência nos pontos de entrada, buscava-se colocar amostras de nodos *octree* vazios, mas com pesos indicando regiões a serem "evitadas" nos cálculos da função indicadora, ou seja, para evitar que as regiões vazias fossem interpoladas.

Contudo, para que a informação dos pontos "vazios" fossem consideradas nos cálculos, algumas alterações precisavam ser feitas. No PSR os dados de entrada são lidos para

construir a *octree* e depois relidos para inicializar os valores de pesos e normais na *octree*, a partir de médias dos pontos de entrada. O problema é que se passou a ter dois tipos de entradas: pontos "reais" e pontos "vazios". Então, o procedimento de construção da *octree* precisava ser alterado, pois existiriam folhas "reais", possuindo um valor de normal e construídas com os dados "reais"; e folhas da *octree* "vazias", resultantes da fusão de várias amostras vazias em determinadas regiões do espaço, recebendo normal zero.

Como o PSR calcula posições e pesos médios, baseados na distribuição no espaço das amostras, a ideia consistia em efetuar os mesmos cálculos, só que em dois conjuntos de pontos: os "reais" e os "vazios". Então, as *octrees* seriam mescladas e quando ocorresse de duas folhas, uma "real" e outra "vazia", ocuparem o mesmo lugar do espaço, a folha real seria considerada só que com peso reduzido, já que amostras vazias no mesmo lugar de reais indicariam que as reais não estavam muito consistentes. Com isso, buscava-se introduzir esses pontos "vazios" contendo direções nulas mas pesos consistentes, no cálculo da função indicadora, que antes simplesmente ignorava essas informações, não gerando dados de entrada (coeficientes).

No entanto, nossa ideia original foi prejudicada por diversos fatores. Inicialmente, a função base usada pelo PSR para realizar os cálculos ignorou completamente as informações passadas a partir dos pontos "vazios" criados. Isso ocorre porque o PSR utiliza uma função base F onde a posição de cada ponto da amostra é substituida pela a do centro do nodo da folha que o contém. Assim, o vetor que corresponde as amostras de pontos é expressado como a soma linear de F onde cada amostra contribui com um peso, que é exatamente o vetor normal para o coeficiente correspondente a função do nó da sua folha. Como as folhas vazias possuem normal zero, essa informação é desconsiderada, independente das alterações feitas no processo de construção das *octrees*. Como os pesos usados não dependem em nada da metodologia usada para geração da *octree*, não seria possível gerar pesos "consistentes" aos pesos reais usados, e nem se deveria atribuir pesos arbitrariamente. Além disso, verificou-se também, que mesmo que pesos consistentes fossem atribuidos, durante o cálculo dos operadores laplaciano e de divergência, qualquer infor-

mação contento normal zero seria novamente desconsiderada previamente e não interferia em nada nos cálculos.

Portanto, depois de constatar que a ideia inicial não funcionaria, pois os cálculos matemáticos do PSR desconsideraria qualquer tentativa de interferência partiu-se para outras ideias, como descrevem as seções seguintes.

4.3 Nova representação volumétrica

Já que não foi possível passar ao PSR as informações da linha de visão do *scanner*, decidiu-se então desconsiderar os dados que representavam regiões que preenchiam buracos. Essa decisão foi motivada também por outros fatores, sobretudo a identificação de outras limitações do PSR. Depois de testes em diferentes modelos, verificou-se que os problemas não ocorrem apenas em algumas regiões que podem ser erroneamente conectadas, mas sim se mostra de diferentes maneiras em diversas regiões onde não existe informação suficiente, podendo aparecer em forma de:

- regiões que não correspondem a buracos mas são incorretamente preenchidas;
- variação muito grande entre os vértices de uma mesma região preenchida;
- conexão incorreta de regiões que deveriam representar diferentes buracos;
- pedaços desconexos (que deveriam estar conectados);
- redução da área original do buraco e;
- baixa resolução nas regiões preenchidas, sobretudo em grandes buracos.

Esses problemas ainda se somam a baixa qualidade da malha gerada pelo PSR, que contém diversos tipos de inconsistências, tais como o aparecimento de *outliers* como vértices no modelo final, assim como apontado no capítulo 3. Depois de mais testes com diferentes modelos verificou-se que os *outliers* que aparecem no modelo final correspondem

a três tipos de problemas: (1) vértices e faces duplicados; (2) vértices não referenciados e; (3) vértices e faces não *manifold* espalhados por todo o modelo. Decidiu-se então, ao invés de interferir no processo de reconstrução do PSR (como havia sido a proposta inicial (seção 4.2)), utilizar o modelo gerado por ele como um "primeiro passo" para a geração de um modelo final que representasse com maior fidelidade o objeto a ser reconstruído.

Primeiramente, buscou-se solucionar três problemas: (1) qualidade da malha gerada pelo PSR; (2) diferença de resolução existente em regiões de buracos e; (3) efeito de suavização excessiva. A origem dos dois primeiros problemas está em uma limitação do algoritmo *Marching Cubes* (MC) [28] [12] usado pelo PSR para extrair a malha 3D. O MC requer uma distribuição homogênea de *voxels*, ou seja, todos devem ser do mesmo tamanho. Em métodos de reconstrução como o PSR que utilizam *octrees* para armazenar o volume, o MC pode não conseguir tratar de "cubos deformados" que ocorrem quando a isosuperfície passa por nodos da *octree* de tamanhos diferentes. Por isso, a malha do PSR contém os diversos tipos de problemas já descritos.

Para evitar que vértices e faces não *manifold* sejam criados é possível aumentar o número mínimo de pontos que caem em um mesmo nodo da *octree*. Isso é controlado pelo parâmetro *samplesPerNode*, que os autores utilizam principalmente para reduzir ruídos e gerar uma malha mais suave. No entanto, ao usar esse parâmetro para evitar que uma malha não *manifold* seja criada, o que se obtem é o efeito de suavização excessiva comentado no capítulo 3.

Buscando resolver os problemas de malha (e da suavização) e de diferença de resolução, optou-se por gerar uma nova representação volumétrica a partir do modelo reconstruído inicialmente pelo PSR. Esse novo volume é criado usando *voxels*, ao invés de *octrees* como faz o PSR, evitando os problemas de malha. Essa nova representação também é compatível com aquela usada na sequência do *pipeline*, adaptando assim o modelo do PSR o que torna possível realizar as etapas que seguem o processo de reconstrução na nova ferramenta proposta (ver capítulo 6). A nova malha 3D é extraída usando a implementação do MC contida no *pipeline* do IMAGO, descrita na seção 2.2.3.

A nova representação é gerada calculando para cada *voxel* próximo à superfície a distância com sinal do centro do *voxel* para a malha do modelo gerado pelo PSR. Assim, o resultado inicialmente é o mesmo, apenas retriangulado. Isso resolve os problemas da diferença de resolução (baixa resolução) nas regiões de buracos, e também elimina os diversos problemas existentes na malha inicialmente gerada pelo PSR garantindo a geração de uma malha *manifold* sem gerar o efeito de suavização excessiva, pois não é necessário interferir nos parâmetros do PSR.

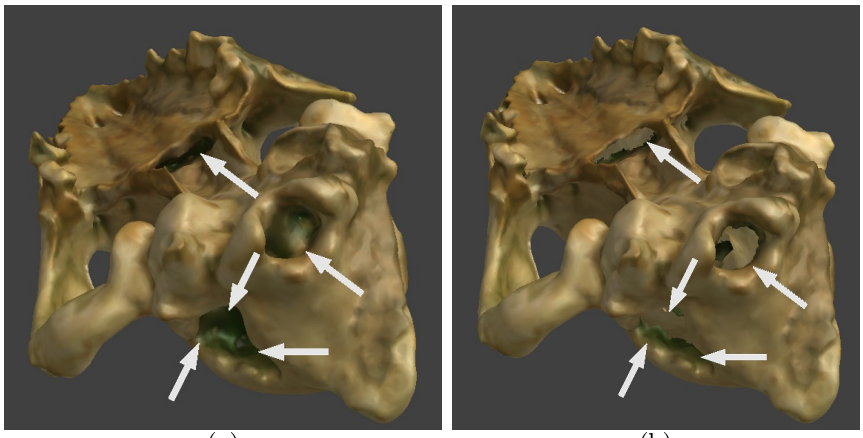
Após a criação da nova representação volumétrica seguem as alterações quanto ao preenchimento de buracos, descritas na seção 4.4.

4.4 Eliminação do preenchimento de buracos

Solucionados os problemas em relação a malha do PSR, o próximo passo é lidar com os diversos problemas relacionados ao preenchimento de buracos. Como não foi possível interferir nos cálculos do PSR para fazê-lo considerar as informações de espaço vazio (ver seção 4.2), optou-se por desconsiderar o preenchimento.

A ideia consiste em detectar no modelo gerado pelo PSR aqueles vértices e faces que correspondem a regiões preenchidas, ou seja, regiões das quais não existem informações. Para identificar um vértice como pertencente a um buraco ou não, realiza-se uma busca (para agilizar a busca, *kd-trees* são usadas) nas vistas capturadas do objeto reconstruído. Se a distância entre um dado vértice do modelo gerado e um vértice de uma determinada vista for menor que um limiar, então esse vértice representa uma região do modelo que foi capturada por alguma vista, senão esse vértice é resultado do preenchimento buracos do PSR. Durante os experimentos o limiar usado foi a informação de erro do *scanner* (*scannerError*), apresentando bons resultados, como ilustra a Figura 4.3. A ideia para a eliminação do preenchimento feito pelo PSR é resumida no Algoritmo 4.1.

Depois de desconsiderar o preenchimento de buracos do PSR, é preciso repreencher os buracos gerados, de forma que melhor se aproximem de como deveriam ser no objeto real,



(a) (b)

Figura 4.3: Eliminação do preenchimento de buracos usando *scannerError*: (a) modelo gerado pelo PSR com buracos preenchidos; (b) modelo após eliminação do preenchimento.

Algoritmo 4.1 Eliminação do Preenchimento de Buracos

Requer: $Mesh$ = Modelo 3D (PSR) sem buracos
Garante: $Mesh$ = Modelo 3D com buracos

1: $arrayFlag[Mesh.num_{vertices}] = false$;
2: **para** $i = 0$ até $Mesh.num_{views}$ **faça**
3: carrega $view[i].Mesh$;
4: carrega $view[i].Kdtree$;
5: **para** $j = 0$ até $num_{vertices}$**faça**
6: **se** $arrayFlag[j] == false$ **então**
7: $point = vertice[j]$*$view[i].MeshLocal$; //projeta o vértice j na vista i
8: $view[i].Kdtree.findClosest(point,scannerError)$;
9: **se** $view[i].Kdtree.closest$!= $false$ **então** //então achou correspondente
10: $arrayFlag[num_{vertices}] = true$;
11: **senão**
12: **continue**;
13: **fim se**
14: **fim se**
15: **fim para**
16: **fim para**
17: **para** $j = 0$ até $Mesh.num_{vertices}$ **faça**
18: **se** $arrayFlag[numero_{vertices}] == false$ **então** //então é buraco
19: **delete** vertice[j];
20: **delete** faces que o referenciam;
21: **fim se**
22: **fim para**
23: **fim**

evitando os diversos problemas observados no resultado do PSR. A seção 4.5 descreve a solução empregada.

4.5 Novo preenchimento de buracos

Depois que o modelo 3D já não possui mais os buracos preenchidos, e está representado de maneira compatível com a representação usada no *pipeline*, a próxima etapa consiste em repreencher os buracos usando o algoritmo de difusão volumétrica (VD) de Davis *et al.* [14], descrito no capítulo 2.

Diversas vantagens para o uso desse algoritmo já foram citadas na seção 2.2.3, tais como o tratamento de casos topologicamente difíceis, a geração de uma superfície suave, a garantia de um resultado *manifold*, e a eficiência temporal. No entanto, é preciso atentar para o fato de que em sua implementação básica (sem considerar a informação de espaços vazios) o VD também pode gerar resultados errados, ao propagar a informação por regiões não conhecidas como vazias. Portanto, a solução para o problema do preenchimento de buracos depende dessa informação, e por saber disso os autores do VD incluiram no algoritmo a possibilidade de acrescentar essa informação no processo de difusão.

No *pipeline* do IMAGO, os algoritmos volumétricos VRIP e IVIA realizam o *carving* nas vistas do modelo durante o processo de integração (no IVIA é realizado na primeira etapa), marcando como vazio os *voxels* que estão fora do modelo. Esta é uma técnica utilizada pelos próprios autores do VD e será explicada na seção 4.5.1.

A Figura 4.4 ilustra alguns "cortes" que mostram a região que corresponde a um buraco, onde o marrom corresponde aos *voxels* sem informação e o azul os *voxels* conhecidos como vazios. O resultado após a aplicação do VD pode ser visto também, onde nota-se a influência da informação de espaço vazio (azul). As Figuras 4.4a e 4.4c ilustram o preenchimento para o modelo gerado pelo VRIP e com informação de espaço vazio auxiliando o VD. As Figuras 4.4b e 4.4d ilustram o preenchimento para um modelo gerado pelo PSR (com preenchimento eliminado) que depois foi repreenchido pelo VD, mas sem utilizar informação de espaço vazio. Nota-se que os buracos internos do *protocyon* foram melhor preenchidos, já que nenhuma informação foi difundida na região conhecida como vazia. Diversas outras comparações são também apresentadas no capítulo 5.

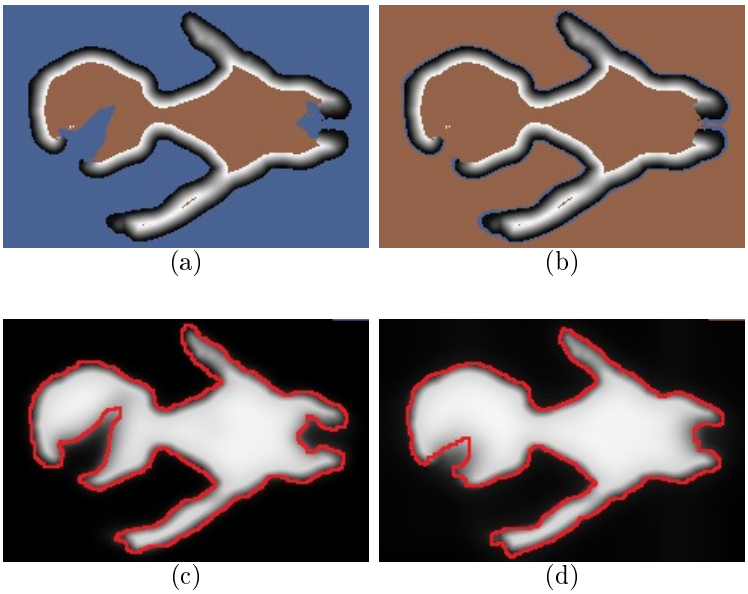


Figura 4.4: Preenchimento de buracos.

4.5.1 Aplicação do *Space Carving* (SC)

Em sua implementação no IVIA o SC é utilizado inicialmente para eliminar *outliers* (seção 2.2.1.3). Como a integração do PSR gera bons resultados, como apresentado no capítulo 3, não é preciso usar essa informação para eliminar *outliers*, ela será usada apenas para guiar o VD.

O importante aqui, é o espaço vazio, do ponto de vista de cada vista (imagem), ou seja, é justamente calcular os *voxels* vazios que cada vista determina, para que eles sejam propagados pelo VD. Para isso, usa-se a mesma ideia do IVIA de ter um bit para cada *voxel* que recebe valor 1 (ou seja, vazio), sempre que o cálculo da distância com sinal dele para a vista que ele pertence for negativo. Como a distância é calculada sobre a linha de visão do *scanner*, sabe-se que os voxels fora da superfície terão distância negativa. É importante ressaltar também que não interessa o valor da distância, pois esse já foi calculado uma única vez direto para a malha do PSR (seção 4.3), o importante aqui é apenas marcar os *voxels* vazios.

Outro ponto importante, é que os *voxels* considerados no SC são aqueles cuja distância não foi calculada anteriormente (durante a geração do novo volume). Pois, durante a geração do novo volume os *voxels* que não foram marcados como próximos da superfície

são considerados como "desconhecidos" (assim como funciona no IVIA), e agora no SC são "avaliados" para saber se são ou não "vazios". Quando um *voxel* for marcado como vazio por alguma vista, em outras ele não precisa ser calculado de novo, o que ajuda a diminuir o tempo gasto para o cálculo. Tomando a Figura 4.4b como exemplo, os *voxels* calculados correspondem a região marrom, para que se tornem eventualmente azuis (vazios).

Para que a informação dos *voxels* vazios seja considerada pelo VD, utiliza-se um parâmetro α (entre 0.001 e 0.01 [14]) que representa o peso atribuído a esses *voxels*. Isso é necessário pois, os *voxels* identificados como vazios não têm peso atribuído (peso = 0, pois foram desconsiderados na geração do novo volume), no entanto, este peso precisa ser maior que zero para que eles interfiram no processo de preenchimento de buracos.

A Figura 4.5 (retirada de Davis *et al.* [14]) ilustra o processo de difusão volumétrica em 2D, utilizando a informação dos *voxels* vazios. Incorporar essa informação adicional (em azul) faz com que a superfície combinada fique em sua maioria fora da região que se sabe ser "vazia", permanecendo suave.

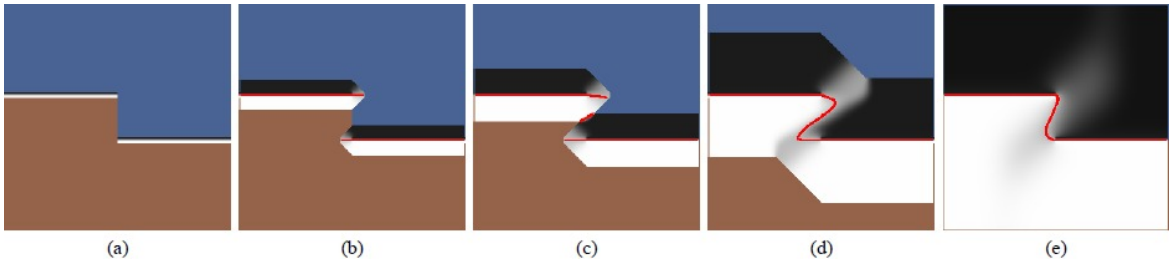


Figura 4.5: Ilustração do processo de difusão em 2D, usando a informação de espaço vazio (em azul): (a) distância com sinal (em escala de cinza), onde preto está fora da superfície e branco dentro, *voxels* invalidos estão em marrom, azul representa o espaço vazio; (b) começa o processo de difusão; (c) as superfíces começam a interagir; (d) o buraco se fecha; (e) a superfície converge.

Pode-se dizer que a única dificuldade em realizar o *space carving* é o aumento no desempenho temporal total. Isso acontece pois, o SC não depende exclusivamente do número de vistas, mas também é afetado pelo número de *voxels*, ou seja, quanto maior for a resolução da malha gerada, maior será o número de *voxels* e consequentemente o tempo gasto pelo SC. No entanto, apesar de ser a usada pelos autores do VD, esta não é a implementação mais eficiente do SC, e pode ser futuramente melhorada. Uma ideia seria efetuar o SC através da rasterização 3D de tetraedros, sendo que cada tetraedro teria

uma base triangular em uma face da malha 3D da vista, e o outro vértice no centro da lente do scanner. Isso torna desnecessário o cálculo de distância (sobre a linha de visão do *scanner*) para todos os *voxels* de cada vista, acelerando consideravelmente a execução do SC.

4.6 Resumindo o algoritmo

Finalizando a descrição das soluções empregadas para a geração de um modelo mais preciso, a partir da integração realizada pelo PSR, o Algoritmo 4.2 apresenta as soluções ordenadamente. Como entrada, recebem-se as vistas devidamente alinhadas cujos pontos são a entrada para a execução do PSR. Uma vez obtido o modelo integrado pelo PSR, aplica-se a eliminação de buracos baseando-se no *scannerError* (linha 2). Essa busca utiliza as *kd-trees* herdadas do processo de alinhamento. Uma nova representação volumétrica é criada (compatível com a utilizada no *pipeline* e na nova ferramenta), a partir do cálculo da distância com sinal dos *voxels* próximos à superfície para a malha do PSR (linhas 3-7). Depois, aplica-se a técnica de *space carving* marcando os *voxels* "vazios" (que se encontram entre a superfície e a linha de visão do *scanner*) (linhas 8-13). A seguir, os buracos são preenchidos usando o algoritmo de Difusão Volumétrica (VD) combinado com as informações de espaço vazio captadas do *space carving* (linha 14). Finalmente, uma nova malha é extraída usando o algoritmo *Marching Cubes* (MC) em sua implementação existente no *pipeline* do IMAGO (linha 16). Pode ainda se optar pela coloração dos vértices do novo modelo gerado, e pela sequência normal do *pipeline*.

A etapa mais lenta do Algoritmo 4.2 é o *space carving*, como já comentado na seção anterior. A etapa de eliminação do preenchimento apesar de estar relacionada ao número de vistas e do número de vértices do modelo do PSR, é acelerada pela utilização das *kd-trees*. A etapa da geração do novo volume depende da resolução desejada, que determina número de *voxels*, mas o processo é acelerado pois são considerados apenas aqueles próximos da superfícies.

Algoritmo 4.2 Geração do Novo modelo 3D

Requer: Vistas do modelo alinhadas
Garante: $Mesh$ = Novo modelo 3D gerado

1: Poisson Surface Reconstruction // Gera modelo do PSR
2: Algoritmo 4.1($Mesh$); //Elimina preenchimento de buracos
3: Marcam-se os *voxels* próximos da superfície;
4: **para** $k = 0$ até *voxels* **faça**
5: SDF = Calcula a distância com sinal de $v[k]$ para $Mesh$; //Gera novo volume
6: $Mesh = SDF$; //Salva o novo volume criado, excluindo o antigo
7: **fim para**
8: **para** $k = 0$ até *voxels* **faça**
9: **para** $i = 0$ até *views* **faça** //Aplica-se o *space carving*
10: SDF = Calcula a distância com sinal de $v[k]$ para $view[i].Mesh$;
11: **se** SDF == -1 **então** //Se $v[k]$ não pertence a superfície
12: $v[k].vazio = true$; //Marca o *voxel* $v[k]$ como vazio
13: **fim se**
14: **fim para**
15: **fim para**
16: Aplica Difusão Volumétrica //Preenche os buracos (VD+SC ($v[k].vazio$));
17: Extrai a malha 3D //usando o *Marching Cubes*;
18: **fim**

CAPÍTULO 5

ANÁLISE DOS RESULTADOS

Recaptulando os problemas encontrados nos modelos reconstruídos pelo PSR:

- baixa qualidade da malha 3D gerada;
- baixa resolução nas regiões preenchidas, sobretudo em grandes buracos;
- regiões que não correspondem a buracos mas que são incorretamente preenchidas;
- conexão incorreta de regiões que deveriam representar diferentes buracos;
- pedaços desconexos (que deveriam estar conectados);
- redução da área do buraco e;
- variação muito grande entre os vértices de uma mesma região preenchida.

Partindo destes problemas, o capítulo 4 apresentou as soluções propostas. Neste capítulo são apresentados os resultados dos experimentos realizados com diversos modelos. Busca-se aqui analisar, através de comparações qualitativas e quantitativas, a viabilidade das soluções propostas. Para isso, foram selecionados para os experimentos, modelos com diferentes geometrias e com buracos de diferentes topologias, tamanhos e situações que representassem desafios. A Figura 5.1 traz uma ilustração geral dos principais modelos utilizados nos experimentos.

Para os experimentos analisados aqui inicialmente foram utilizados modelos com buracos reais, ou seja, para os quais nenhuma informação foi capturada pelo *scanner* (Figs. 5.1a, 5.1b, 5.1c e 5.1d). No entanto, afim de analisar de uma forma mais quantitativa a viabilidade da solução porposta para o preenchimento de buracos, foram realizados

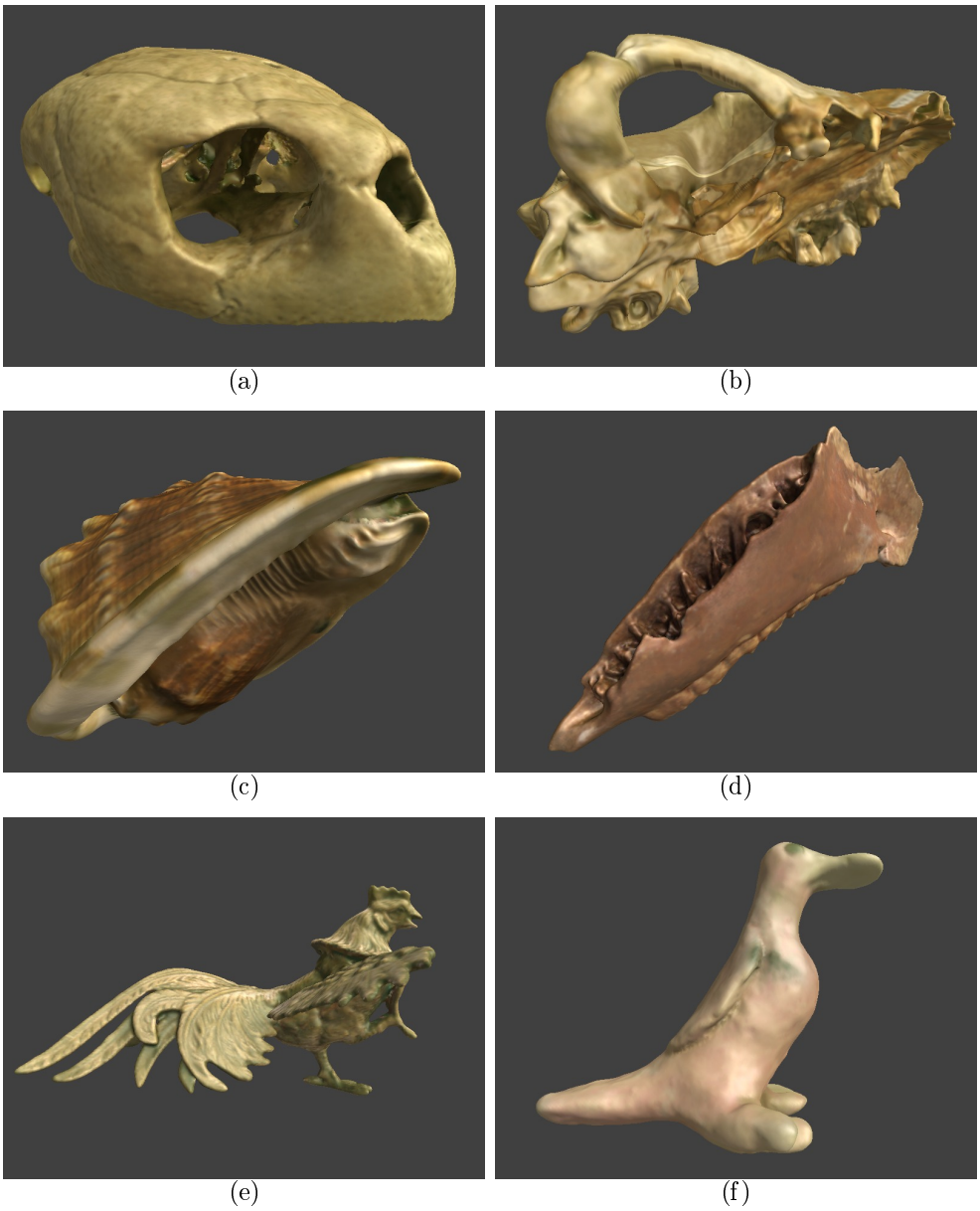


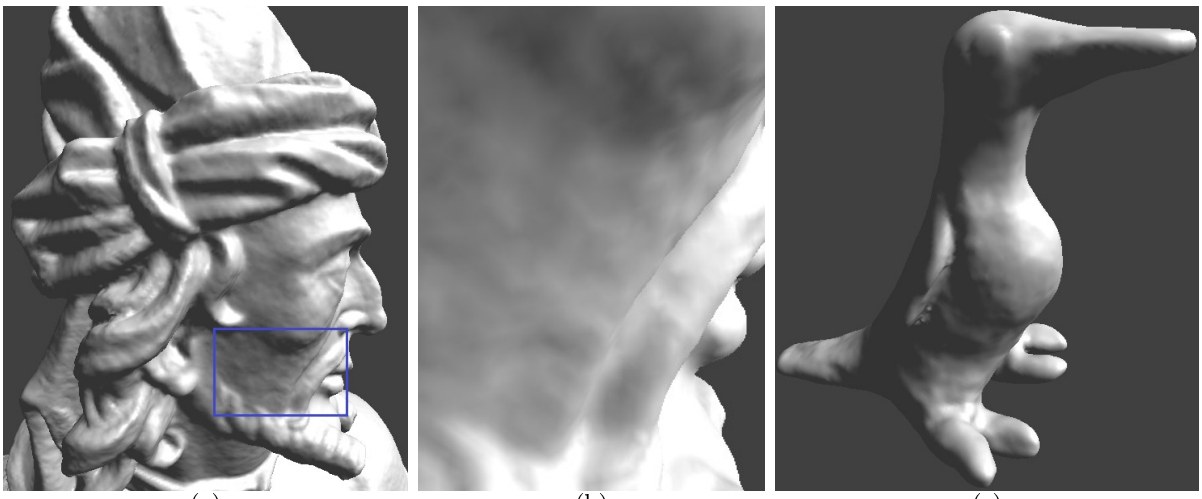
Figura 5.1: Modelos usados nos experimentos: (a) *tartaruga*; (b) *protocyon*; (c) *cypreacassis*; (d) *cavalo* (madibula); (e) *galo*; (f) *pato*.

também experimentos com buracos gerados em modelos onde já existia informação capturada pelo *scanner*. Esses experimentos serão descritos detalhadamente na seção 5.4, e utilizam, entre outros modelos, o *galo* e o *pato* ilustrados na Figura 5.1e e 5.1f.

As seções 5.1 e 5.2 analisam os resultados quanto à qualidade e resolução da malha 3D gerada. A seção 5.3 analisa qualitativamente os experimentos com o preenchimento de buracos e a seção 5.4 apresenta alguns experimentos realizados afim de analizar quantitativamente os resultados.

5.1 Qualidade da malha gerada

Como já comentado nos capítulos 3 e 4, *outliers* em forma de vértice aparecem na malha final gerada pelo PSR. A solução adotada foi gerar uma nova representação volumétrica e extrair uma nova malha 3D, usando a versão do *Marching Cubes* [28] [12] implementada no *pipeline* do IMAGO (ver seção 2.2.3). A Figura 5.2 ilustra o modelo do *pato* gerado pelo PSR e depois da nova representação gerada, nela os problemas existentes anteriormente (setas) não são mais verificados.



(a) (b) (c)

Figura 5.2: Nova representação volumétrica: Modelos equivalentes aos criados pelo PSR, ilustrados na Fig. 3.12

5.2 Resolução da malha em buracos preenchidos

Verificou-se também que a resolução da malha em regiões de buracos preenchidos pelo PSR era inferior ao restante da malha, sobretudo em regiões com buracos muito grandes. A retriangulação do modelo através da geração de uma nova representação volumétrica (ver seção 4.3) resolve esse problema, deixando a malha com resolução uniforme, como mostra a Figura 5.3. Para facilitar a visualização, os modelos são ilustrados em modo *wireframe*.

Nas Figuras 5.3a, 5.3b e 5.3c é possível observar a diferença na resolução (tamanho dos

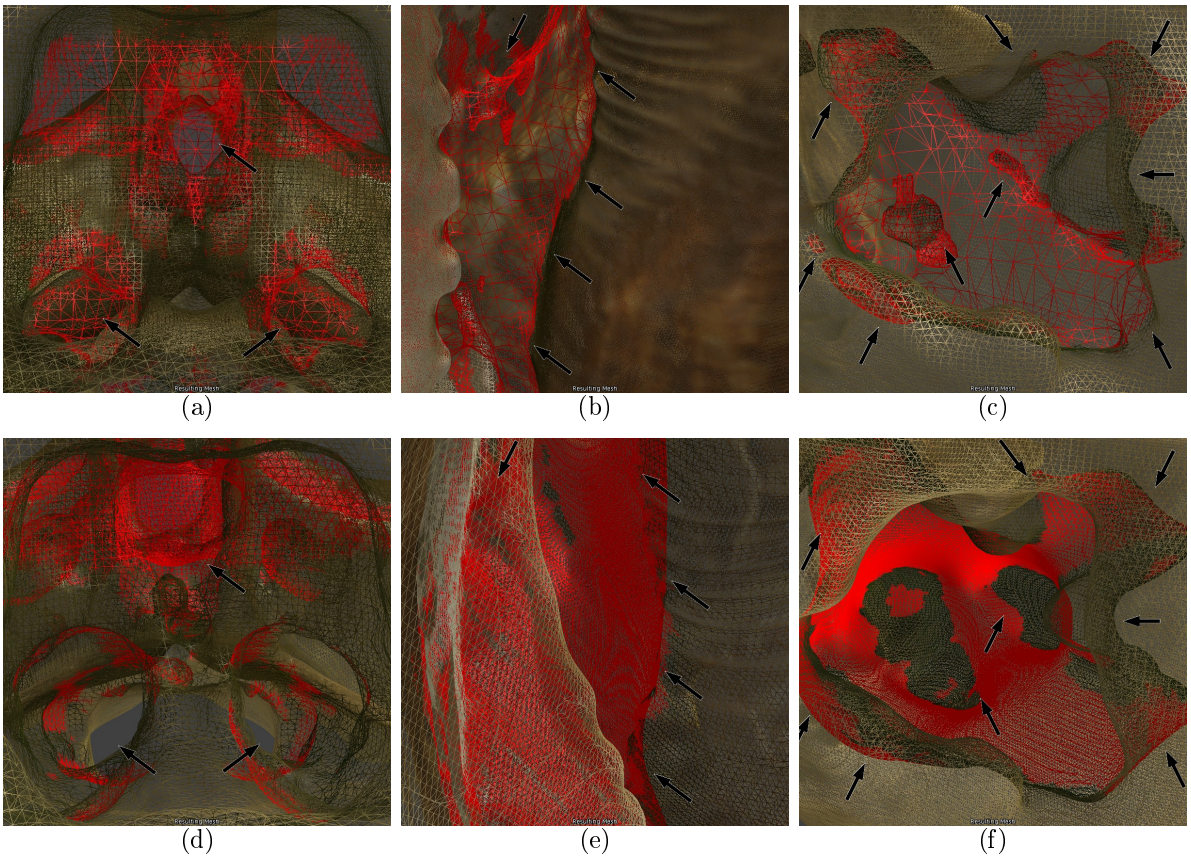


Figura 5.3: Resolução da malha: (a, d) *tartaruga*; (b, e) *cypreacassis*; (c, f) *protocyon*. Acima modelos gerados pelo PSR, abaixo modelos com a nova malha gerada.

triângulos) entre as regiões que correspondem a buracos preenchidos (regiões em vermelho) e o restante do modelo (textura normal). Já nos mesmos modelos após a geração da nova malha (Figuras 5.3d, 5.3e 5.3f) a resolução é uniforme, não se diferenciando nas regiões preenchidas (vermelho).

5.3 Preenchimento incorreto de buracos

Diversas são as situações em que o PSR preenche incorretamente os buracos. A Figura 5.4 ilustra dois modelos que "resumem" essas situações. Posteriormente, cada um dos problemas será discutido individualmente. Como grande parte dos problemas do PSR ocorrem em buracos localizados no interior dos objetos (ou que possuem a maior parte de sua área no interior), para facilitar a visualização os modelos são ilustrados em *wireframe* e com vértices em vermelho nas regiões preenchidas.

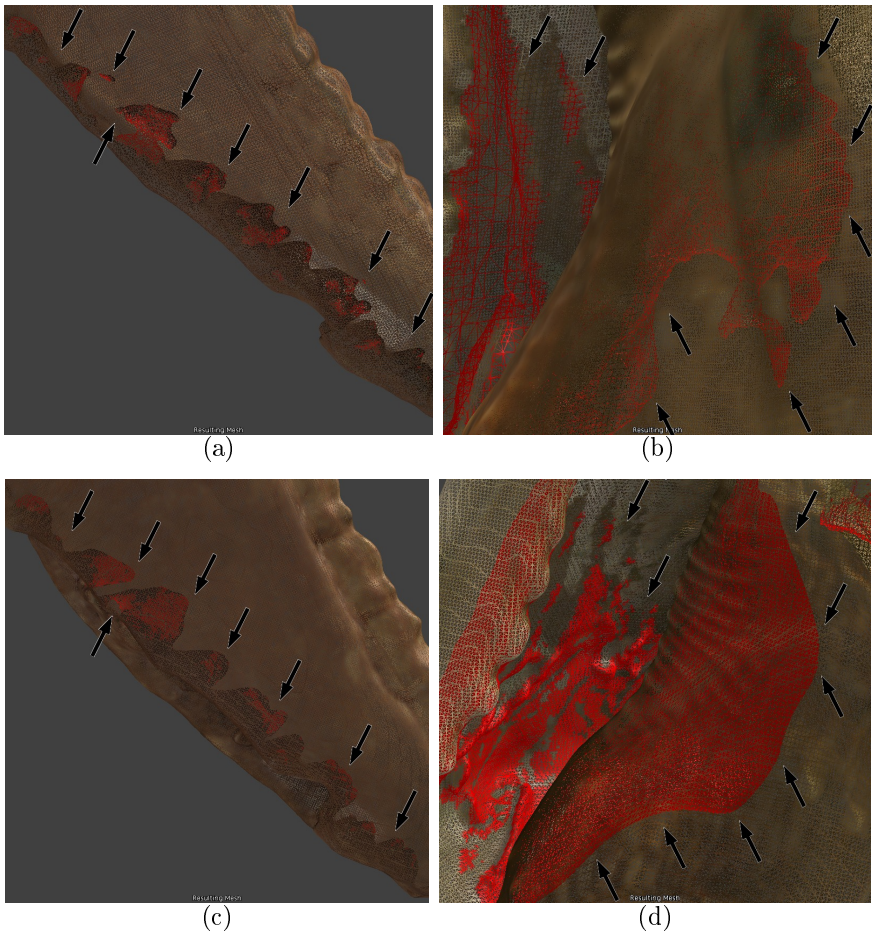


Figura 5.4: Preenchimento de buracos: (a, c) *cavalo*; (b,d) *cypreacassis*. Acima modelos gerados pelo PSR, abaixo modelos preenchidos pelo VDSC.

As Figuras 5.4a e 5.4c comparam o preenchimento de buracos em um fóssil de uma mandíbula de um cavalo pré-histórico. No modelo do PSR (Fig. 5.4a) é possível notar buracos mais curtos e regiões desconectadas que deveriam pertencer ao mesmo buraco, como se nota no modelo repreenchido (Fig. 5.4c). Uma forma mais "acidentada" ou descontinuada é um problema comum do PSR ao preencher buracos maiores, como na Figura 5.4b. O modelo ilustrado trata-se de uma concha de um *cypreacassis*, que devido ao seu formato torna impossível a captura de dados no interior da sua concha. Como o PSR não considera informações da linha de visão do *scanner*, a região preenchida por ele nem mesmo consegue se aproximar de uma forma mais "natural", que seria seguir o formato da concha, como se pode notar no modelo com preenchido usando VDSC (Fig. 5.4d).

5.3.1 Redução da área do buraco

Outro problema que pode ocorrer, devido a falta de informações para guiar o preenchimento de buracos, é que os buracos preenchidos (principalmente no interior dos modelos) tenham uma área inferior àquela que possívelmente teriam. Isso pode ser visualizado nos modelos da Figura 5.5, que estão em *wireframe* e com vértices em vermelho nas regiões preenchidas.

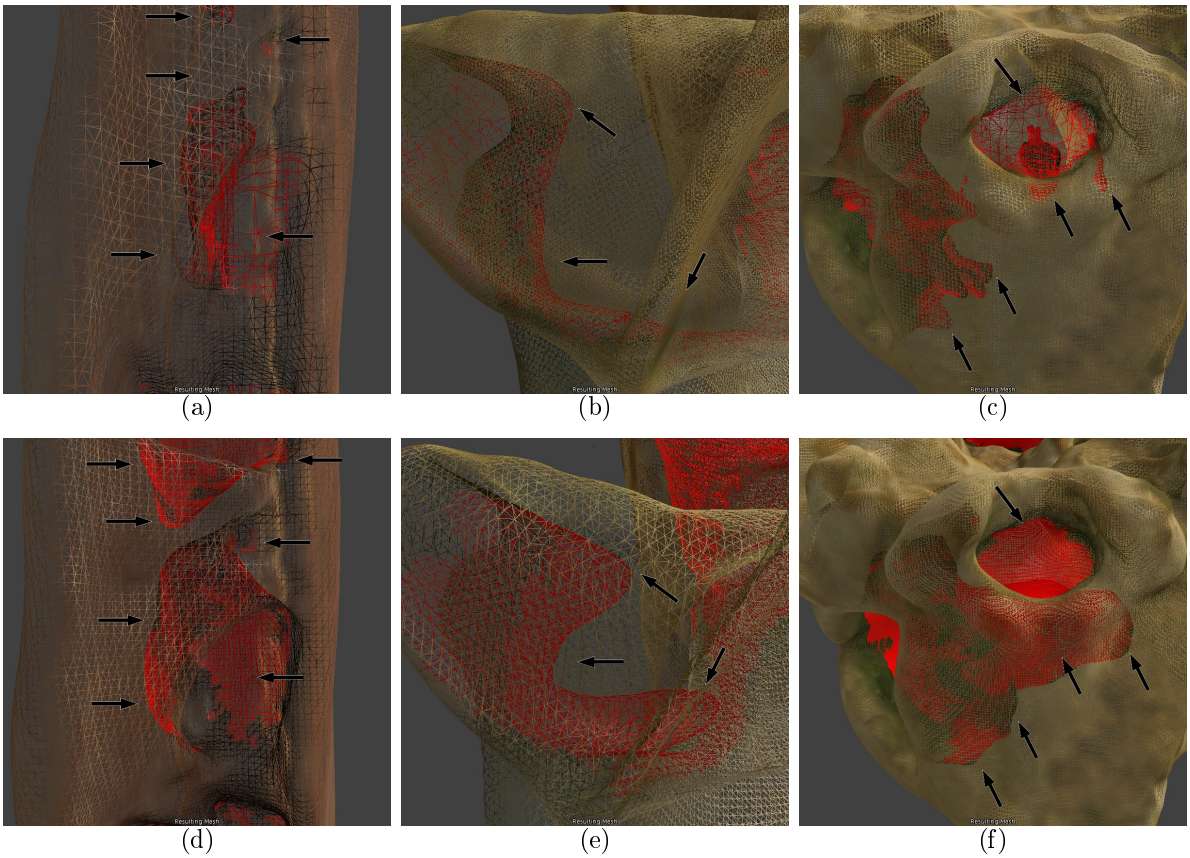


Figura 5.5: Área referente ao buraco: (a, d) *cavalo*; (b, e) *cypreacassis*; (c, f) *protocyon*. Acima modelos gerados pelo PSR, abaixo modelos gerados pelo VDSC.

Nos modelos do *cavalo* e *cypreacassis* da Figura 5.5 nota-se uma maior profundidade nos buracos preenchidos com o VDSC (Figs. 5.5d e 5.5e) em relação aos resultados do PSR (Figs. 5.5a e 5.5b). Nas Figuras 5.5c e 5.5f observa-se uma das causas desse problema, onde o resultado do PSR (Fig. 5.5f) desconsidera "pedaços" preenchendo apenas uma parte do buraco. Por outro lado, o resultado do VDSC (Fig. 5.5c) considera essa informações guiando melhor o seu preenchimento.

5.3.2 Pedaços desconexos

O problema de desconsiderar pedaços de informações comentado na seção 5.3.1 pode melhor ser visto na Figura 5.6. Esse problema acontece sobretudo em buracos internos do modelo onde, ao invés de conectar esses pedaços através do preenchimento das regiões vazias entre eles para formar uma única região que represente todo o buraco, o PSR os considera como regiões que possuem buracos individuais (e não como pertencentes ao mesmo buraco). Assim, ele os preenche individualmente, o que gera resultados bastante distantes da realidade (como pedaços de informações que flutuam no interior de objetos, desconectados do restante do modelo).

A Figura 5.6 traz 3 modelos em *wireframe* e com vértices em vermelho nas regiões preenchidas. Afim de possibilitar uma noção maior da proporção das regiões preenchidas no modelo da *tartaruga*, as Figuras 5.6a e 5.6e ilustram uma "visão por baixo" de toda região preenchida no interior do modelo, e as Figuras 5.6b e 5.6f uma "visão por cima". Como se pode notar são grandes regiões oriundas de diversos buracos localizados em regiões do interior do crânio da *tartaruga* (Fig. 5.1a), que se interconectam seguindo a mesma forma da extrutura externa (região com a textura original). No entanto, isso acontece apenas nos modelos preenchidos pelo VDSC (Figs. 5.6e e 5.6f), já nos resultados do PSR (Figs. 5.6a e 5.6b) os buracos preenchidos formam grandes regiões desconexas, que visivelmente não seguem a geometria do objeto. Resultados semelhantes podem ser vistos para o modelo do *protocyon* ((PSR) Fig. 5.6j e (VDSC) Fig. 5.6n).

Como dito anteriormente, a grande região preenchida na *tartaruga* é formada por diversos buracos, alguns deles podem ser vistos nas Figuras 5.6c, 5.6d (PSR) e nas Figuras 5.6g, 5.6h (VDSC). Nelas é possível ver pedaços que "flutuam" desconectados do restante do buraco, ao qual deveriam pertencer. Além, de não possuir pedaços desconexos os resultados do VDSC ainda não preenchem erradamente algumas regiões como faz o PSR. Outros exemplos de pedaços desconexos podem ser vistos no interior da mandíbula do *cavalo* Fig. 5.6i (PSR) e Fig. 5.6m (VDSC) e no interior do crânio do *protocyon* Figs. 5.6k, 5.6l (PSR) e Figs. 5.6o, 5.6p (VDSC). Esse é um dos problemas mais comuns

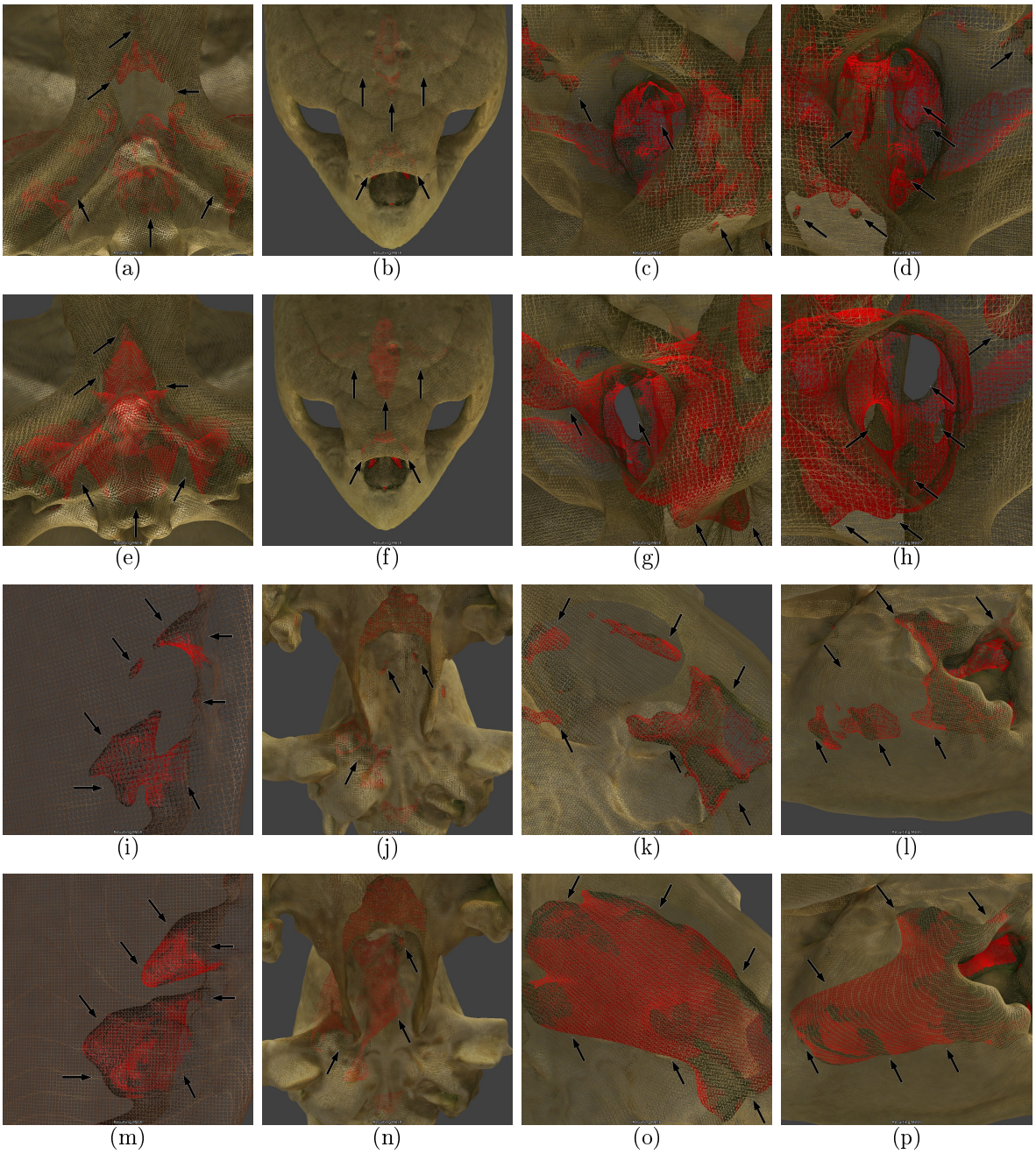


Figura 5.6: Pedaços desconexos: (a - h) *tartaruga* - acima modelos gerados pelo PSR, abaixo modelos gerados pelo VDSC; (i, m) *cavalo*; (j - l; n - p) *protocyon* - acima modelos gerados pelo PSR, abaixo modelos gerados pelo VDSC.

do preenchimento do PSR, verificado em todos os modelos onde foi preciso preencher regiões internas, e que sem a visualização em modo *wireframe* (e diferenciando os vértices da região preenchida) é bastante difícil de ser detectado.

5.3.3 Conexão incorreta de regiões

Assim como em alguns casos, sobretudo no interior dos modelos, o PSR gera regiões desconexas, em outros casos (esses sobretudo nas regiões externas do modelo), o PSR preenche buracos de forma a conectar incorretamente regiões que deveriam estar desconectadas. Isso ocorre quando regiões próximas que contém buracos distintos, aparentam (a partir de determinados pontos de vista) estarem sobrepostas. Então, sem a informação de outros pontos de vista seus buracos parecem pertencer a uma mesma região conexa (ligando as duas ou mais regiões), e é isso que o PSR considera na hora de preenchê-los, como ilustra a Figura 5.7.

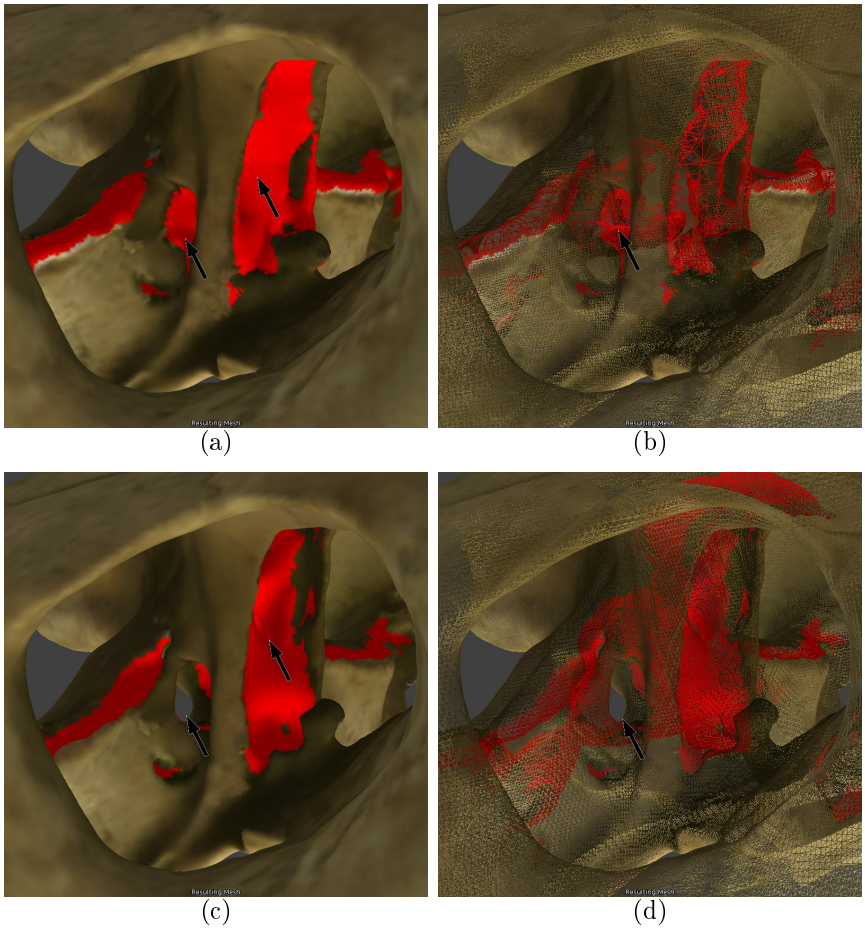


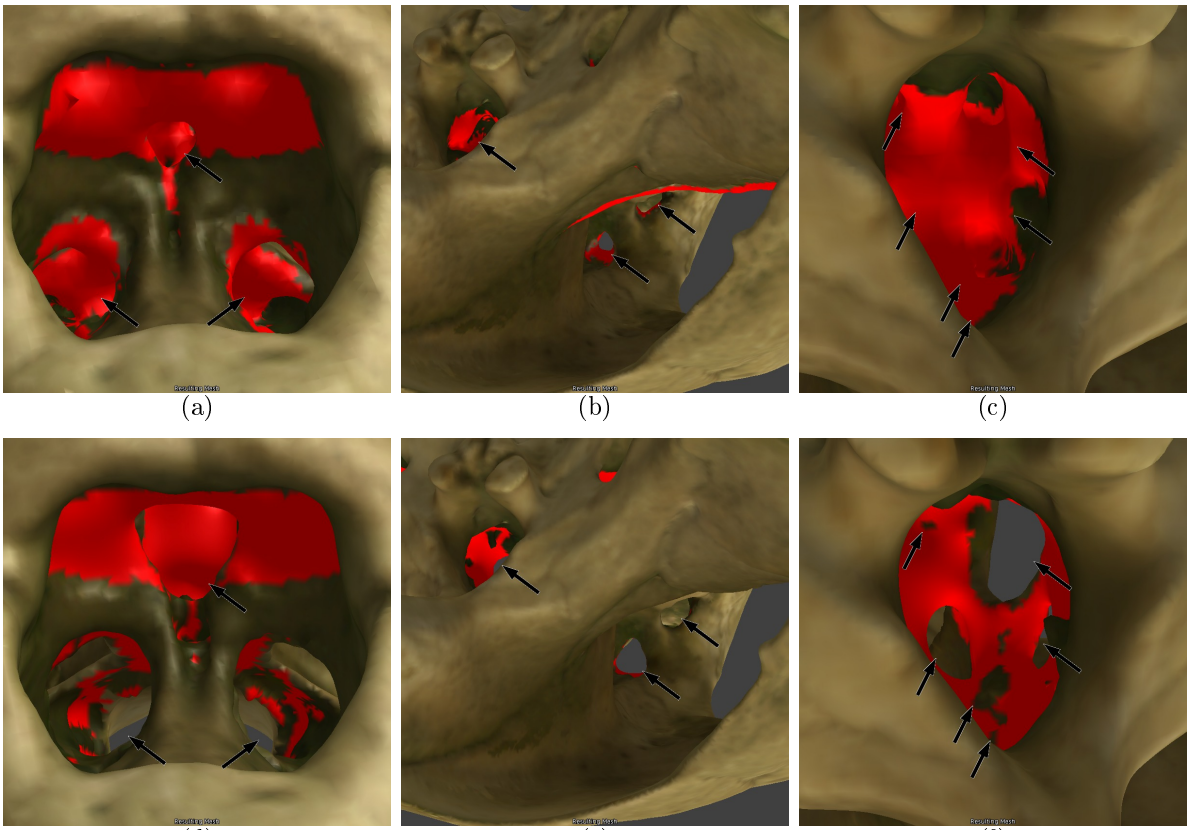
Figura 5.7: Conexão incorreta de regiões no interior da cabeça da *tartaruga*. Acima modelos gerados pelo PSR, abaixo modelos gerados pelo VDSC.

A Figura 5.7 ilustra uma mesma região solida e em *wireframe* que representa um desafio para o preenchimento de buracos. A figura ilustra duas regiões paralelas com

buracos diferentes que se conectam no fundo, mas podem ser facilmente confundidos como um único buraco conectando as duas regiões, que é o que ocorre com o modelo gerado pelo PSR (Figs. 5.7a, 5.7b).

5.3.4 Regiões que não representam buracos

Outro problema comum é o preenchimento incorreto de regiões que não representam buracos gerados por oclusões, mas que fazem parte objeto capturado: ocorrem casos de preenchimento total, ou parcial, de regiões que correspondem a buracos no objeto real, e que portanto não deveriam ser preenchidas, como ilustra a Figura 5.8.



(a) (b) (c)

(d) (e) (f)

Figura 5.8: Preenchimento de buracos na cabeça da *tartaruga*: Acima modelos gerados pelo PSR, abaixo modelos gerados pelo VDSC.

Exemplos de preenchimento parcialmente incorretos podem ser vistos nas Figuras 5.8a e 5.8b. A Figura 5.8c mostra diversos buracos totalmente preenchidos incorretamente. Os resultados equivalentes para o VDSC podem ser vistos nas Figuras 5.8d, 5.8e e 5.8f. Por

ser um erro bastante comum é possível de ser visto também em quase todas as outras figuras já apresentadas.

5.4 Experimentos com buracos criados

Os experimentos realizados até aqui comprovam qualitativamente a superioridade dos resultados alcançados pelo preenchimento de buracos usando VDSC sobre o PSR. No entanto, buscou-se ainda uma maneira de analisar quantitativamente os resultados. Mas, assim como não existem na literatura trabalhos que empreguem métricas para comparar quantitativamente os métodos de integração, também desconhecemos trabalhos que comparem quantitativamente o preenchimento de buracos.

Já que em buracos reais, ou seja, aquelas regiões do modelo que não foram capturadas pelo *scanner*, não se tem nenhuma informação capaz de corroborar quantitativamente os resultados do preechimento, buscou-se aqui uma alternativa. A ideia consiste em "criar" buracos em modelos que originalmente não os contém (ao menos não buracos consideráveis), através da eliminação de uma ou mais vistas. Dessa forma, já que existe a informação real capturada do objeto, é possível comparar qual método preencheu os buracos de maneira mais próxima a informação original.

5.4.1 Distância quadrada entre os pontos

A primeira análise busca comparar a distância entre a posição dos pontos que preenchem os buracos e a posição que esses pontos originalmente possuem no modelo sem buracos, ou seja, a posição em que foram capturados do objeto real. Para isso, foi usada a mesma ideia utilizada para detectar e eliminar o preenchimento do PSR (ver seção 4.4). Nos experimentos para cada ponto identificado como buraco uma busca foi feita, nas vistas eliminadas, pelo ponto que seria seu correspondente. Como resultado, a busca retorna além do ponto correspondente a distância quadrada entre o ponto original e o ponto resultante do preenchimento. Já que a informação de erro de *scanner* (*scannerError*) é

conhecida, sabe-se que todos os pontos originais se encontram dentro desse erro. Assim, é possível comparar qual dos métodos gerou os vértices cujas posições mais se aproximam das originais capturadas.

Nos experimentos buscou-se também limitar um limiar, onde verificou-se que as distâncias começaram a se afastar muito das posições originais de forma a influenciar negativamente nos resultados. O limiar escolhido nos experimentos foi 10**scannerError*, o que significa que para um *scannerError* igual a $0.7mm$ um vértice é considerado bem posicionado se está em até $7mm$ de distância da posição original.

A Tabela 5.1 resume o percentual de vértices para cada um dos modelos testados, para cada um dos métodos, que se encontram a uma distância de até: (1) abaixo do limiar: 2**scannerError*, 5**scannerError*, 10**scannerError*; (2) acima do limiar 15**scannerError*, 25**scannerError* e 50**scannerError*. Além disso, são apresentadas também a porcentagem de vértices que estiveram abaixo e acima do limiar.

Tabela 5.1: Percentual de Pontos Encontrados em Cada um dos Grupos

Modelo	Metodo	2	5	10	15	25	50	Abaixo	Acima
Alamito	PSR	16.6	25.0	22.6	13	12.3	10.5	64.3	35.8
	VDSC	22.1	25.2	20.6	12.4	9.8	9.9	67.9	32.2
Anta	PSR	23.6	28.9	19.8	10.9	10.5	6.4	72.3	27.7
	VDSC	22.6	45	17.8	5.8	5.2	3.7	85.3	14.7
Cesto	PSR	24.3	30.8	19.3	9.3	8.6	7.8	74.4	25.7
	VDSC	26.1	28.7	19.6	10	8.6	7	74.4	25.7
Cypreacassis	PSR	14.3	16.2	13.1	9.3	11.5	37	42.3	57.8
	VDSC	25.1	29.9	15.2	5.2	7.1	17.5	70.2	29.8
Duende	PSR	33.1	35.4	23.9	7.2	0.4	0.0	92.4	7.6
	VDSC	42.4	34.2	19.7	3.8	0.0	0.0	96.3	3.8
Galo	PSR	38.6	31.1	11	5.1	5.6	8.6	80.7	19.3
	VDSC	37.5	42.8	19.3	0.4	0.0	0.0	99.7	0.4
Onça	PSR	17.3	20.4	16.3	9.4	11.8	24.8	56.6	43.5
	VDSC	13.5	24.3	21.3	11.8	11.4	17.8	60.6	39.4
Padre	PSR	54.1	30.8	13.8	1.3	0.0	0.0	98.7	1.3
	VDSC	62.1	31.3	6.6	0.0	0.0	0.0	100	0.0
Pato	PSR	57.9	31.6	9.9	0.7	0.0	0.0	99.3	0.7
	VDSC	85.7	14.3	0.0	0.0	0.0	0.0	100	0.0
Shrek	PSR	25.1	20.6	15	10	12.6	16.7	60.7	39.3
	VDSC	26.2	20.1	14.7	8.2	11.9	18.9	61	39

Observa-se na Tabela 5.1, que em todos os casos de teste o VDSC alcançou mais

de 60% de pontos preenchidos cuja distância para o ponto originalmente capturado está abaixo do limiar, o que não ocorre em alguns casos para o PSR. Para compreender melhor a diferença entre cada um dos modelos testados pode-se classificar os buracos criados nos modelos de três formas: buracos com (1) área pequena (*cesto*, *padre*, *pato*); (2) área média (*alamito*, *anta*, *duende*, *galo*); (3) área grande (*cypreacassis*, *onça*, *shrek*).

Antes de avaliar os resultados de cada grupo, é importante conhecer não apenas as porcentagens mas também a média das distâncias quadradas para cada um dos grupos. Além dos percentuais já apresentados, a Tabela 5.2 resume a média das distâncias quadradas dos vértices que foram encontrados em cada um dos grupos. Também são apresentadas a média geral das distâncias (MG) e a média das distâncias acima do limiar (ML).

Tabela 5.2: Média da Distância Quadrada dos Pontos em Cada um dos Grupos

Modelo	Metodo	2	5	10	15	25	50	MG	ML
Alamito	PSR	1.0	2.5	5.2	8.9	14.3	43.4	9.4	20.9
	VDSC	1.1	2.5	5.4	8.9	14.3	42.5	8.7	20.9
Anta	PSR	1	2.4	5.2	8.7	13.8	39.6	6.9	17.7
	VDSC	1.1	2.4	5.1	8.7	13.9	40.8	5	18.5
Cesto	PSR	1	2.3	5.2	8.4	13.9	40.1	7.1	19.8
	VDSC	1	2.3	5.1	8.6	13.9	38.8	7	19
Cypreacassis	PSR	1.1	2.6	5.5	9	14.8	50.6	22.5	37
	VDSC	1.1	2.5	5.1	8.8	14.4	49.2	11.9	33.8
Duende	PSR	0.9	2.2	4.8	7.8	12.9	-	2.8	8.1
	VDSC	0.9	2.2	4.7	7.9	-	-	2.4	7.9
Galo	PSR	1.1	2.6	5.5	8.8	14.5	51.9	7.6	29.7
	VDSC	1.1	2.7	5.3	9.2	-	-	2.6	9.2
Onça	PSR	0.7	1.7	3.7	6.3	10	33.1	10.5	21
	VDSC	0.7	1.8	3.7	6.3	10	32.3	8.6	18.1
Padre	PSR	1	2.6	5.8	8.5	-	-	2.3	8.5
	VDSC	1.1	2.6	5.9	-	-	-	1.9	-
Pato	PSR	0.7	1.8	3.8	5.2	-	-	1.4	5.2
	VDSC	0.7	1.4	-	-	-	-	0.8	-
Shrek	PSR	1	2.4	5.3	8.8	14.1	45	11.7	25.9
	VDSC	1	2.5	5.2	8.8	14.2	47.1	12.9	29

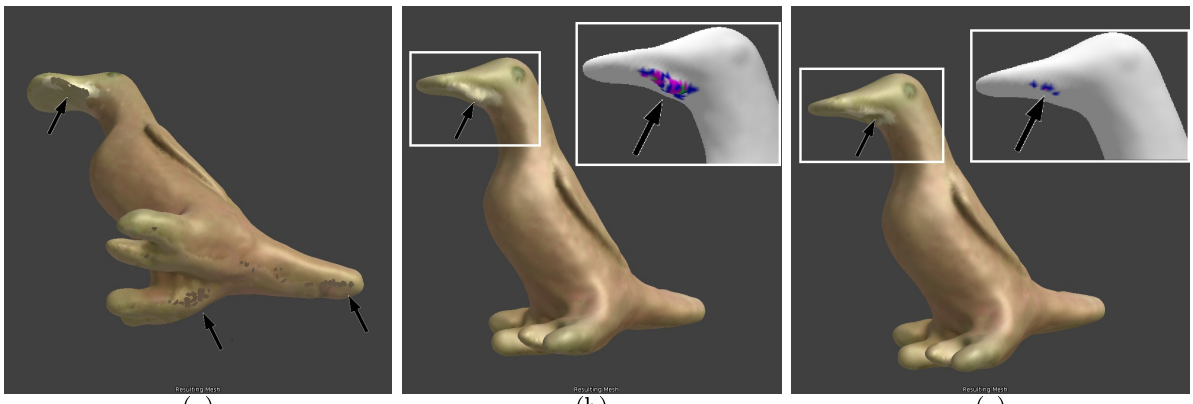
Na Tabela 5.2 verifica-se que em todos os casos, com excessão da *anta* e do *shrek*, a média das distâncias tanto geral, quanto acima do limiar são menores para o VDSC. A média das distâncias acima do limiar é importante, pois possibilita comparar o quanto os resultados do VDSC e do PSR se distanciam do ideal. Na maioria dos casos a dife-

rença é pequena, no entanto em alguns casos como o do *galo* a diferença de distâncias é consideravelmente grande, e será discutida posteriormente.

No caso da *anta*, a média acima do limiar é ligeiramente maior para o VDSC, no entanto o percentual de pontos com distância acima do limiar é bem menor comparado ao PSR, o que contribui para uma média geral de distâncias também menor para o VDSC. Os casos do *cesto*, *alamito* e *duende* são um pouco melhores para o VDSC em comparação ao PSR, tanto nas porcentagens, quanto nas médias de distâncias. O *padre* e o *pato* não tiveram resultados acima do limiar para o VDSC, e seus resultados em média ficaram melhores que o PSR, que teve resultados acima do limiar.

5.4.2 Experimento com o *pato*

A Figura 5.9 ilustra um exemplo de casos de buracos tipo 1 (pequenos) como o *pato*. A Figura 5.9a ilustra o buraco criado no modelo. O resultado para o PSR está na Figura 5.9b e o do VDSC na Figura 5.9c. No resultado do PSR, observa-se uma "protuberância" criada ao preencher o buraco, o que não ocorre no resultado do VDSC. A região destacada nas figuras representa em cores as distâncias dos vértices que preenchem o buraco: azul (2**scannerError*), roxo (5**scannerError*), verde (10**scannerError*) e amarelo (15**scannerError*, acima do limiar). Os vértices da região que aparecem em branco se encontram abaixo do *scannerError*, que para o caso do *pato* era de $0.7mm$.

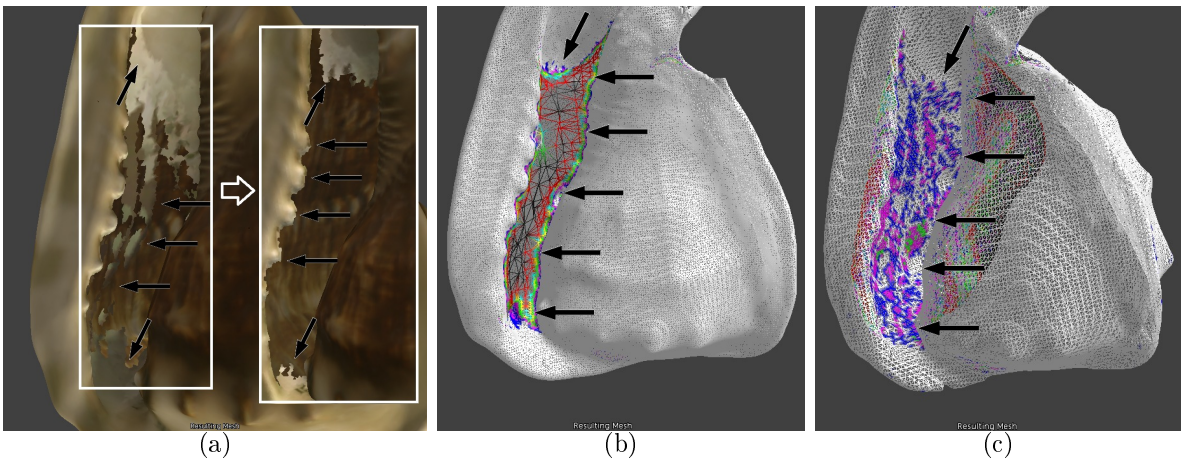


(a) (b) (c)

Figura 5.9: Experimento com o *pato*: (a) buraco criado; (b) resultado do PSR; (c) resultado do VDSC. A região em destaque representa em cores cada um dos grupos de distâncias.

5.4.3 Experimento com o *cypreacassis*

Outro caso de teste particular é o *cypreacassis*. Enquanto os outros modelos não possuem buracos consideráveis, o modelo do *cypreacassis*, como já visto, possui uma grande região de buracos no interior da concha. Então, nesse teste buscou-se observar o quanto a informação obtida pelo *space carving* contribui com o preenchimento. Para isso duas vistas da região interna do modelo, que anteriormente contribuiam para o *space carving* foram retiradas, ampliando o buraco na região interna, como mostra a Figura 5.10a. Sem a informação nessa região, buscou-se comparar se os vértices que a preenchem se aproximam de suas posições originais, mesmo com um número menor de vistas para retirar informação, e com uma área de buraco maior. As Figuras 5.10b e 5.10c trazem os resultados do PSR e do VDSC respectivamente, com cores significando: azul (2**scannerError*), roxo (5**scannerError*), verde (10**scannerError*), amarelo (15**scannerError*, acima do limiar), azul claro (25**scannerError*), vermelho (50**scannerError*) e preto (região original do buraco).



(a) (b) (c)

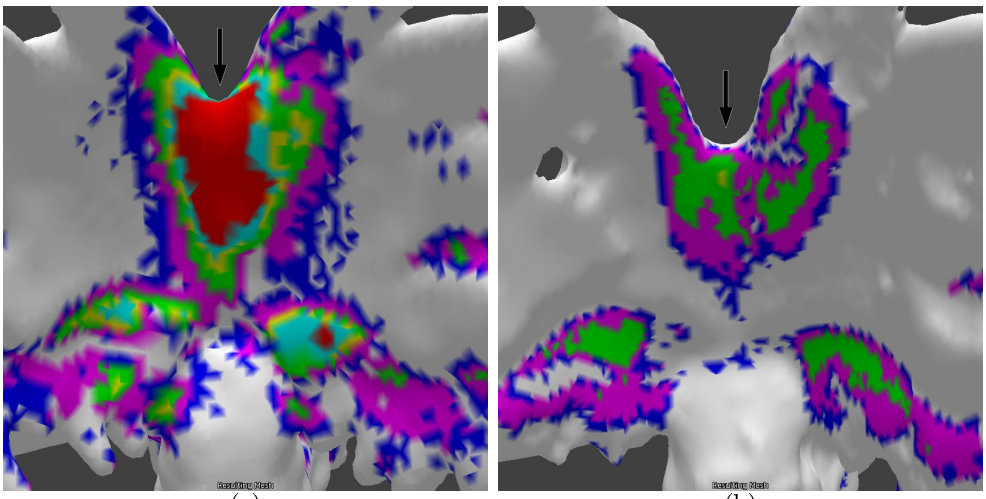
Figura 5.10: Experimento com o *cypreacassis*: (a) buraco criado; (b) resultado do PSR; (c) resultado do VDSC. Destaque para cores de cada um dos grupos de distâncias.

Além de conseguir melhores resultados para as regiões com dados originais, o VDSC ainda não foi afetado pela retirada de algumas informações (Fig. 5.10c), conseguindo manter a mesma região preenchida (Fig. 5.4d) quando existiam as informações retiradas (região em vermelho e em preto). Já o resultado do PSR (Fig. 5.10b) foi afetado negativamente pela retirada das informações, e além de não se aproximar nas regiões de dados

originais, ainda piorou o resultado geral comparado ao anterior ilustrado pela Figura 5.4b.

5.4.4 Experimento com o *galo*

O experimento com o *galo* também obteve resultados interessantes, ilustrados na Figura 5.11. As Figs. 5.11a (PSR) e 5.11b (VDSC) ilustram em cores os grupos de distância azul, roxo, verde, amarelo (acima do limiar), azul claro e vermelho. Nelas é possível ver que o PSR se afasta muito das posições originais, inclusive preenchendo indevidadmente algumas regiões.



(a) (b)

Figura 5.11: Experimentos com o *galo*: (a) resultado do PSR; (c) resultado do VDSC.

5.4.5 Desvio padrão das distâncias

Como um último experimento, procurou-se também verificar o desvio padrão das distâncias acima do limiar. Foram considerados apenas os modelos em que pontos foram encontrados em todos os grupos de distância, a Tabela 5.3 traz os resultados. Percebe-se que mesmo em alguns modelos onde a média das distâncias acima do limiar foi próxima nos dois métodos (*alamito*, *cesto*) ou maior (*shrek*), o desvio padrão foi inferior no VDSC em relação ao PSR, ou seja, o grau de dispersão dos pontos acima do limiar é maior para os pontos preenchidos pelo PSR em relação ao VDSC. Isso, já podia ser verificado visualmente através de figuras como a Fig 5.11.

Tabela 5.3: Desvio Padrão dos Pontos Acima do Limiar

Modelo	PSR	VDSC
Alamito	53.7638	51.3992
Anta	57.1373	48.9700
Cesto	29.6933	24.7980
Onça	30.5069	21.9920
Shrek	39.1446	33.5895

5.4.6 Eliminação de bordas

As média das distâncias acima do limiar, para o caso do *shrek*, se apresentaram maiores para o VDSC do que para o PSR (ver Tabela 5.2. Verificou-se que o responsável por esse problema é que em casos de buracos grandes, o PSR tende a começar a se afastar das posições originais dos vértices, assim as bordas das regiões dos buracos já se apresentam de um pouco afastadas da sua posição original, como ilustra a Figura 5.12a, 5.12c. Com isso, após eliminar a região preenchida e repreenchê-la usando o VDSC, os resultados do preenchimento tendem a ser iguais ou piores do que o PSR, pois o algoritmo de difusão volumétrica (VD) vai difundir os dados a partir das bordas que já se encontram afastadas das suas posições, o que prejudica o preenchimento.

Para resolver esse problema e melhorar o resultado do preenchimento, optou-se por aplicar a eliminação de borda, como é realizado pelo IVIA [48] [43] (ver seção 3.3). Assim, o Algoritmo 4.2 apresentado na seção 4.6, tem adicionado a etapa de cálculo de distância com sinal (linha 4) a eliminação de bordas, ou a redução de seu peso durante os cálculos. A Figura 5.12b, 5.12d ilustra os resultados após a geração do novo modelo, eliminando parte das bordas como apareciam no modelo gerado pelo PSR (Figs. 5.12a, 5.12c), nota-se uma melhora no resultado visual.

Os resultados dos experimentos apresentados neste capítulo, em relação a qualidade da malha (seção 5.1), da resolução da malha (seção 5.2) e do preenchimento de buracos, tanto qualitativos (seção 5.3), quanto quantitativos (seção 5.4), confirmam a viabilidade das técnicas empregadas. Os problemas relacionados a malha (seção 3.4) foram resolvidos através da geração da nova representação volumétrica (seção 4.3). Os problemas quanto ao

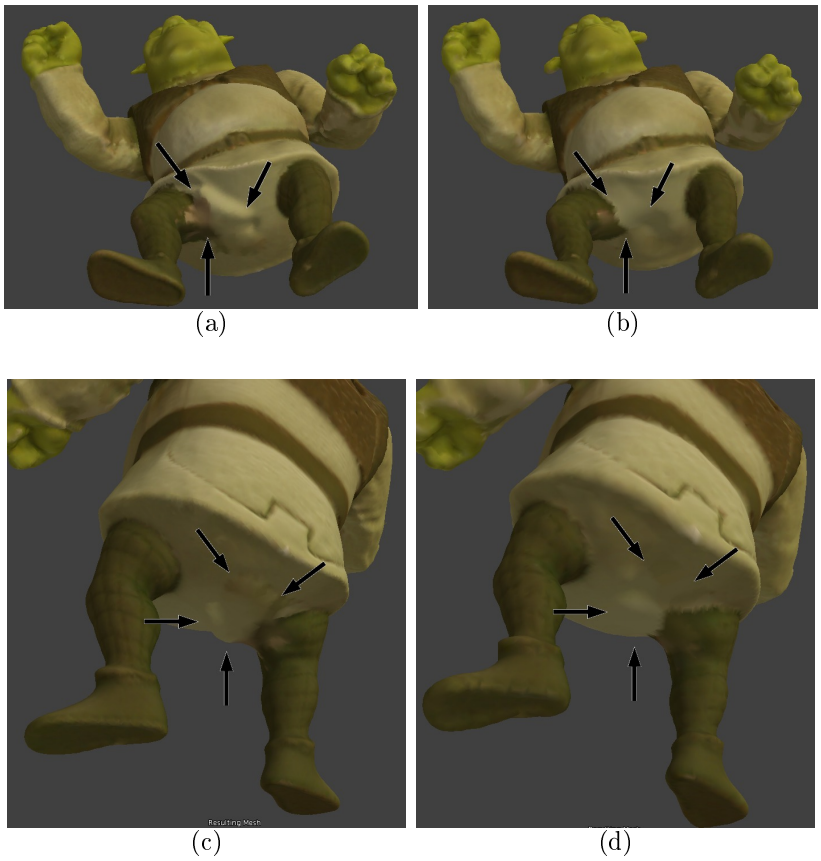


Figura 5.12: Eliminação de bordas: (a, c) resultados gerados pelo PSR com destaque para as boradas dos buracos; (b, d) resultados gerados pelo VDSC após a eliminação das bordas.

preenchimento de buracos foram resolvidos eliminando o preenchimento original do PSR (seção 4.5) e repreenchendo-os utilizando o algoritmo de *Difusão Volumétrica* (VD) de Davis *et al.* [14], juntamente com a técnica de *space carving* 4.5.1. Enfim, o novo modelo resultante da integração do PSR, retriangulação em *voxels* e preenchimento usando VDSC, foi extraído usando o *Marching Cubes* (MC) [28] em sua implementação para representação em *voxels*.

CAPÍTULO 6

FERRAMENTA DE RECONSTRUÇÃO: *RECTOOL*

A principal contribuição que este trabalho busca gerar para o *pipeline* do IMAGO é o desmembramento das etapas (b) e (c) da Figura 1.1. Para isso, buscou-se desenvolver uma ferramenta contendo as seguintes características:

- Maior interação entre o usuário e o modelo 3D: na ferramenta usada anteriormente toda a interação com o modelo 3D era feita por meio de teclas de atalho e a configuração de parâmetros era feita usando um arquivo de texto. A nova ferramenta deve facilitar a interação com o usuário, através da utilização de janelas de configuração, botões, entre outras funcionalidades, mantendo também teclas de atalho;
- Melhor acompanhamento das etapas da reconstrução: as etapas devem ser realizadas uma a uma, e informações importantes visualizadas na janela principal;
- Facilidade de realizar testes/análises: deve ser a principal contribuição da ferramenta, de forma que as etapas sejam realizadas independentemente da etapa de alinhamento, e separadas umas das outras. Isso, agiliza o processo de reconstrução, evita sobrecarga de memória, facilita a identificação de erros. As novas funcionalidades para a interação com o modelo 3D devem permitir uma melhor avaliação do resultado.

Considerando as características descritas acima, esta seção apresenta a ferramenta para a reconstrução (integração, preenchimento de buracos, extração de malha, texturização) *RecTool*. Seguindo a ideia ilustrada no capítulo 1 pela Figura 1.2, a *Rectool* recebe como entrada o arquivo contendo as matrizes de transformação que alinham as vistas do modelo, gerado como saída da nova ferramenta *ICPTool* apresentada por Gomes [18]. Como saída a *RecTool* fornece para as próximas etapas diversas informações:

- modelo 3D *watertight*: modelo integrado, sem buracos e com malha de alta qualidade. Se este modelo contém apenas coloração por vértice então ele é salvo em um arquivo *.m2* (contém informações dos vértices, faces, e cordenadas (u,v)), se ele possui informação de textura (baixa qualidade) é salvo em um arquivo *.m* (contém informações dos vértices e faces);

- informação de textura: juntamente com o arquivo *.m*, outro arquivo *.tga* é gerado contendo as informações de textura do modelo, assim como era realizado pela ferramenta anterior. Além disso, um novo arquivo *.vd* é gerado contendo as matrizes de projeção das imagens coloridas capturadas pelo *scanner* (usadas para geração da textura de baixa qualidade), para auxiliar a etapa de geração de textura de alta qualidade (Andrade [2]);

- *script* de saída: contém as informações de cada etapa da reconstrução (valores dos parâmetros usados), bem como o tempo de execução para cada etapa.

As opções para a interação com a ferramenta, tanto nas etapas da reconstrução, quanto com o modelo 3D gerado são apresentadas nas seções 6.1 e 6.2 respectivamente. A ferramenta foi desenvolvida usando a biblioteca Glui (OpenGL) e conhecimentos de IHC na disposição intuitiva das janelas e na organização das funcionalidades.

6.1 Processo de reconstrução

A *RecTool* tem duas principais janelas para a interação com o usuário durante o processo de reconstrução: uma para visualização e configuração das etapas do processo de reconstrução (*Settings*); e outra que exibe as informações de cada etapa bem como o modelo 3D reconstruído (*RecTool (Main)*).

A janela *Settings* ilustrada na Figura 6.1b permite configurar cada uma das etapas do processo de reconstrução, desde o método de integração usado até as opções para a geração de textura a partir das imagens do *scanner*. Nela estão presentes, na etapa de integração,

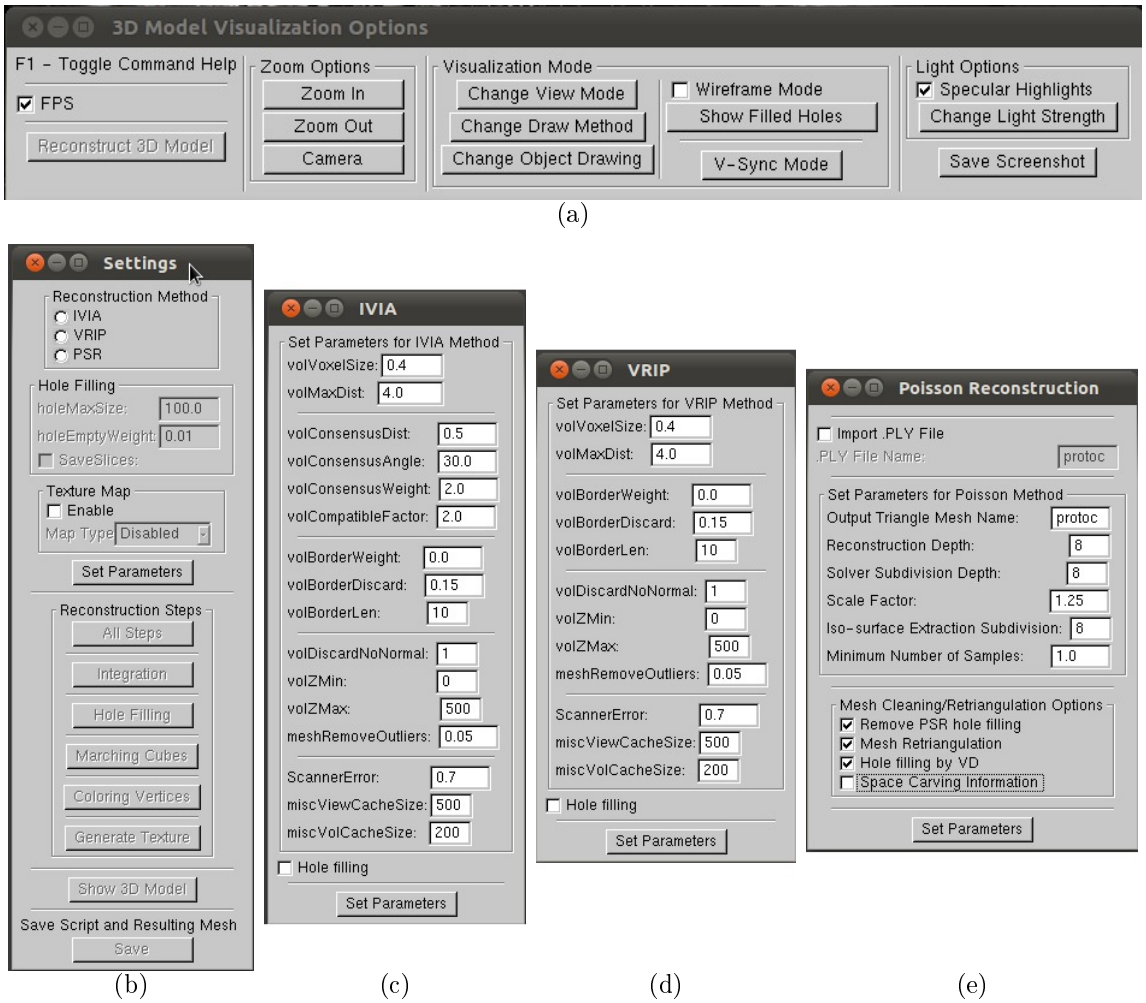


Figura 6.1: Janelas para a configuração dos métodos de integração: (a) opções de visualização; (b) janela principal de configurações; (c) configuração do IVIA; (d) configuração do VRIP; (e) configuração do PSR.

os métodos volumétricos VRIP e IVIA, e o novo método PSR. A Figura 6.1 (c, d, e) ilustra as demais janelas que são acionadas a partir da *Settings* para a configuração dos parâmetros para cada um dos métodos de integração. Note que a janela do PSR (Fig. 6.1e) contém não apenas os parâmetros de controle, mas novas funções para complementar a geração do modelo, assim como descritas pelo capítulo 4.

Após as configurações para a etapa de integração, a *Settings* permite configurar as etapas de preenchimento de buracos e texturização. Depois de configurados os parâmetros o processo de reconstrução pode ser acompanhado pelos botões agrupados no *Reconstruction Steps*. Note que o processo pode ser realizado todo de uma vez, ou controlando cada etapa. Finalizada a reconstrução, o modelo pode ser visualizado pelo botão *Show 3D*

Model e salvo pelo botão *Save*.

6.2 Visualização do modelo 3D

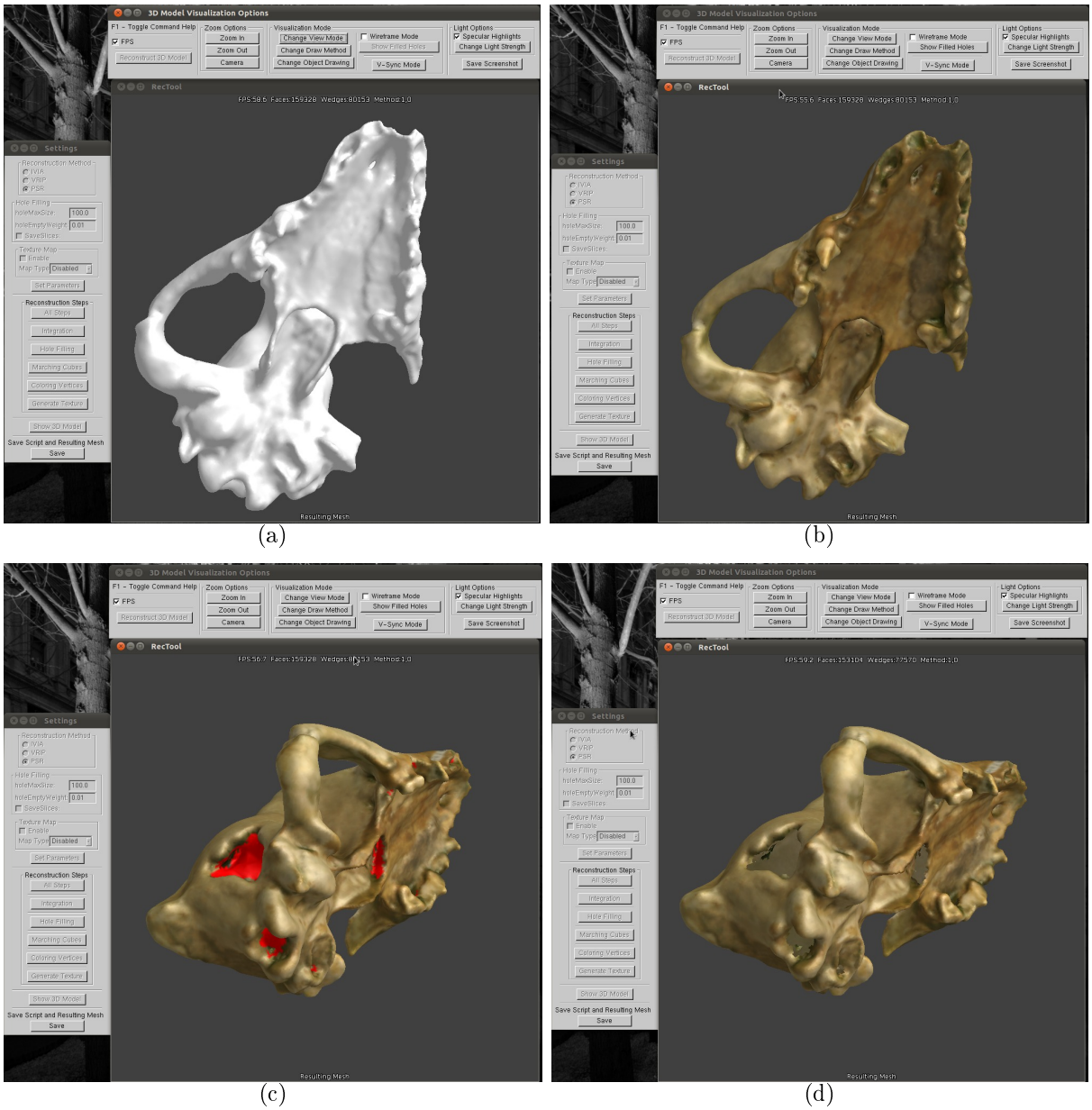


Figura 6.2: Ilustração do modelo *protocyon* sendo reconstruíŋdo pelo PSR: (a) modelo integrado sem textura; (b) modelo 3D após a texturização; (c) detecção de buracos preenchidos (em vermelho); (d) eliminação do preenchimento.

Depois que o objeto foi reconstruído, é possível utilizar diversas opções de visualização na janela *3D Model Visualization Options*, como ilustra a Figura 6.1a. Ela está organizada em grupos de opções. Da esquerda para direita se encontram as opções de visualização de

informações do modelo (FPS) e o botão que aciona a janela *Settings*, seguem as opções de *zoom*, de textura (com ou sem), aparência do modelo (*wireframe* ou sólido), a opção de visualizar os buracos preenchidos (quando não houve preenchimento é possível visualizar as bordas dos buracos), as opções de iluminação e a possibilidade de salvar imagens do modelo reconstruíŋdo. Toda essa interação com o modelo pode ser feita também utilizando teclas de atalho, para ver as teclas de atalho basta apertar a tecla F1.

Finalmente, a janela principal *RecTool* é a responsável pela visualização das informações nas etapas de reconstrução e da interação com o modelo 3D. A Figura 6.2 ilustra a interação com o modelo *protocyon* sendo reconstruído utilizando o método PSR, onde podem ser vistos o modelo inicial gerado (Fig. 6.2a), o modelo com textura (Fig. 6.2b), o modelo com buracos preenchidos detectados (Fig. 6.2c) e o modelo com preenchimento eliminado (Fig. 6.2d).

Este capítulo apresentou a nova ferramenta de reconstrução *RecTool*. As funcionalidas implementadas nessa ferramenta permitem uma maior interação entre o usuário, as etapas de reconstrução e o modelo 3D. Além disso, a medida que outras funcionalidades se façam necessárias, a ferramenta pode ser facilmente atualizada para agrupá-las.

CAPÍTULO 7

CONCLUSÃO

A preservação de acervos culturais e naturais, através da reconstrução 3D de modelos virtuais que representam com fidelidade e precisão os patrimônios reconstruídos, é um desafio. Desafio esse que é ainda maior, considerando a importância da praticidade na execução do conjunto de etapas que compõem o processo de reconstrução. Nesse contexto, o capítulo 1 apontou três principais objetivos desse trabalho. Recaptulando-os:

- Realizar um estudo sobre os métodos de integração de vistas;
- Adicionar na etapa de integração os métodos que possam contribuir com o desempenho, adaptando-os as etapas subsequêntes;
- Implementar uma ferramenta que realize as etapas (b) e (c) do *pipeline* (figura 1.1) desmembrando-o. A nova ferramenta deve conter as modificações realizadas nas etapas (b) do *pipeline*, e adaptá-las para serem acopladas com as etapas da parte (c).

Todos os objetivos apontados foram alcaçados. O capítulo 2 revisou as principais abordagens para reconstrução 3D, sobretudo os métodos de integração de vistas. Essa mesma seção discutiu os métodos já existentes no *pipeline* de reconstrução do IMAGO e um novo método estudado, o PSR. O capítulo 3 descreveu os experimentos realizados com as abordagens já existentes anteriormente no *pipeline* e o método PSR. As vantagens e desvantagens de cada método descritas no capítulo 2 foram confirmadas pelos experimentos, concluiu-se então que o PSR poderia ser usado como uma opção na reconstrução em alta resolução de grandes volumes de dados, no entanto as seguintes para superar algumas limitações, precisaria ser garantido:

- Geração de um modelo *manifold*: de forma a não prejudicar as etapas que seguiam no processo de reconstrução e sem provocar uma super-suavização no modelo;
- Geração de um modelo livre de *outliers*: garantindo a precisão e fidelidade ao objeto preservado;
- Correção do preenchimento de buracos: aumentando a fidelidade do modelo final.

O capítulo 4 descreveu o conjunto de estratégias implementadas para superar as limitações do PSR. As seguintes soluções foram adotadas:

- Geração de uma nova representação volumétrica: solucionou os problemas de malha, garantindo a geração de um modelo *manifold*, sem suavização excessiva, de alta resolução, livre de *outliers*, e compatível com as etapas posteriores;
- Eliminação do preenchimento de buracos: identificou e eliminou regiões incorretamente preenchidas pelo PSR;
- Repreenchimento de buracos: solucionou os problemas de preenchimento incorreto. Para isso a informação da linha de visão do *scanner* (captada usando a técnica de *space carving*) foi considerada no processo de difusão volumétrica para o repreenchimento dos buracos.

Os resultados satisfatórios alcançados com o preenchimento de buracos proposto foram apresentados no capítulo 5, que trouxe uma série de comparações com o PSR. Finalmente, o capítulo 6 apresentou a nova ferramenta de reconstrução *RecTool* desenvolvida. A partir de então, as etapas (b) e (c) da Figura 1.1 passam a ser realizadas pela ferramenta *RecTool*, de acordo com o esquema apresentado na Figura 7.1. A Figura 7.2 ilustra os seis primeiros profetas de Aleijadinho reconstruídos pelo grupo IMAGO usando a ferramenta *RecTool* proposta nesse trabalho.

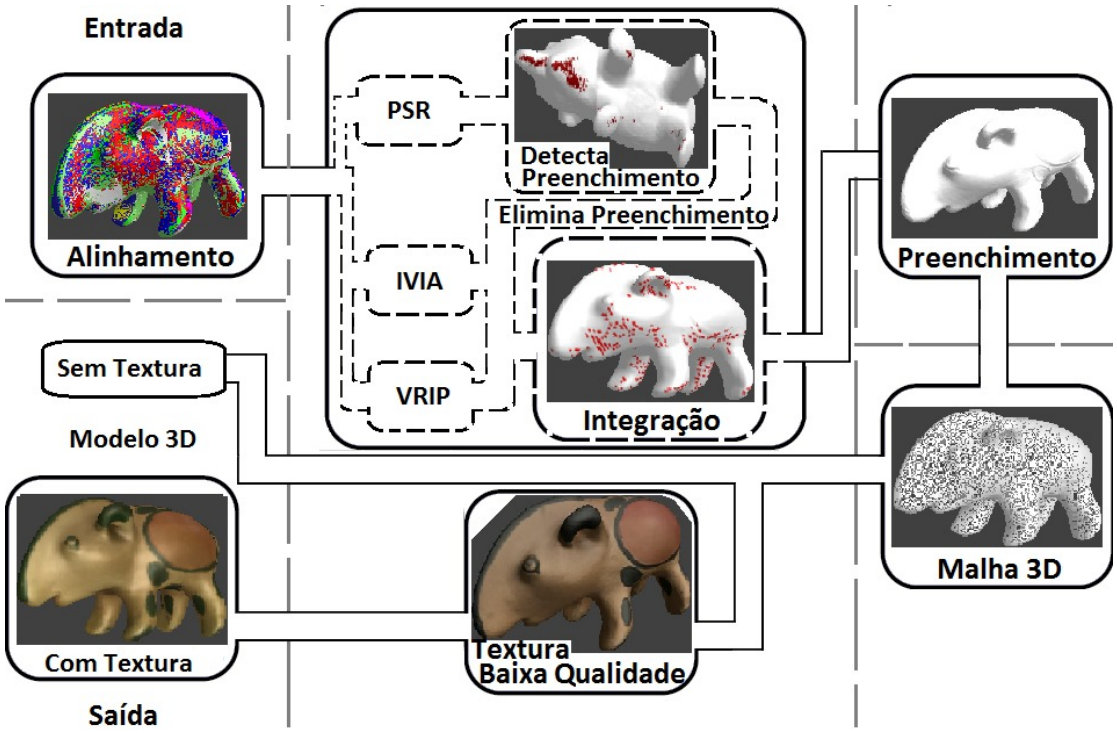
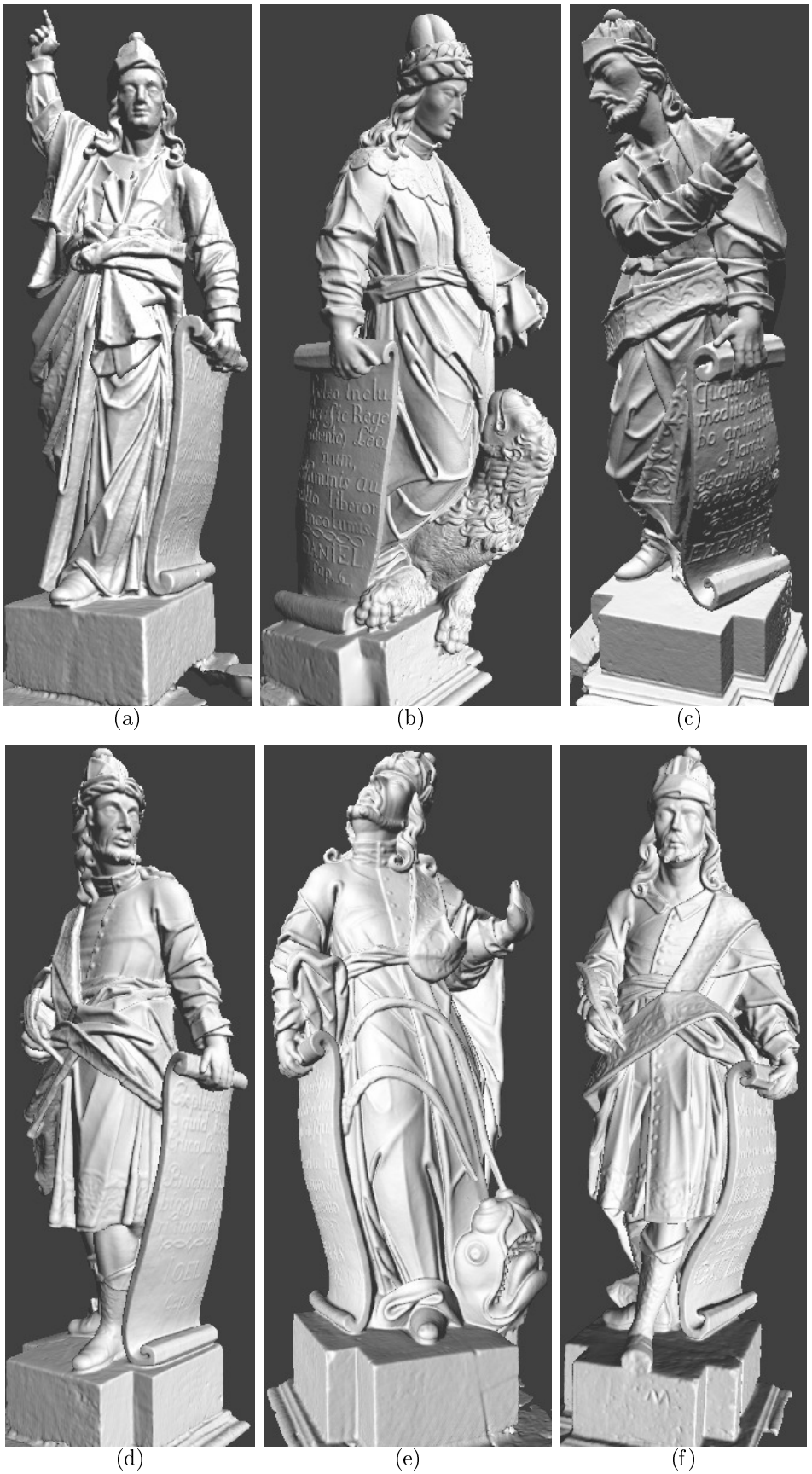


Figura 7.1: Etapas do processo reconstrução na ferramenta *RecTool*.

7.1 Trabalhos futuros

Dentro do conjunto de soluções apresentados para superar as limitações do PSR e adaptá-lo ao *pipeline* do IMAGO, uma etapa que pode ser posteriormente melhorada é a etapa de *space carving*. Apesar dos detalhes de implementação que ajudam a acelerá-la essa é ainda assim etapa mais demorada, no entanto é de fundamental importância para a geração de resultados com maior fidelidade.

Uma grande dificuldade deste trabalho foi comparar a precisão dos modelos reconstruídos pelas abordagens estudadas. Já que não existem métricas adequadas para comparar quantitativamente resultados de reconstrução, essa seria uma área interessante de pesquisas.



(a) (b) (c)

(d) (e) (f)

Figura 7.2: Profetas de Aleijadinho: (a) Abdias; (b) Daniel; (c) Ezequiel; (d) Joel; (e) Jonas; (f) Oséias.

BIBLIOGRAFIA

[1] N. Amenta, S. Choi, e R. K. Kolluri. The power crust, unions of balls, and the medial axis transform. *CGTA*, 19(2-3):127–153, 2001.

[2] B. T. Andrade, O. Bellon, A. Vrubel, e L. Silva. Digital preservation of brazilian indigenous artworks: Generating high quality textures for 3D models. *JCH(2011)*, 2011.

[3] B. T. Andrade, C. M. Mendes, J. Santos Junior, O. Bellon, e L. Silva. 3D preserving XVIII century barroque masterpiece. *JCH (2011)*, 2011.

[4] Olga Regina Pereira Bellon. *Imagens de Profundidade: Segmentação e Representação por superfícies Planares.* Tese de Doutorado, Universidade Estadual de Campinas, Campinas, São Paulo, Brasil PA, 1997.

[5] F. Bernardini, C. L. Bajaj, J. Chen, e D. Schikore. Automatic reconstruction of 3D CAD models from digital scans. *IJCGA*, 9(4/5):327–369, 1999.

[6] F. Bernardini, J. MittleMan, H. Rushmeier, C. T. Silva, e G. Taubin. The ball-pivoting algorithm for surface reconstruction. *IEEE TVCG*, 5(4):349–359, 1999.

[7] F. Bernardini e H. Rushmeier. The 3D model acquisition pipeline. *CGF*, volume 21, páginas 149–172, 2002.

[8] F. Bernardini, H. Rushmeier, I. M. Martin, J. MittleMan, e G. Taubin. Building a digital model of michelangeloâĂŹs florentine pieta. *IEEE CGA*, 22(1):59–67, 2002.

[9] Paul J. Besl e Neil D. McKay. A method for registration of 3D shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, 1992.

[10] M. Bolitho, M. Kazhdan, R. Burns, e H. Hoppe. Multilevel streaming for out-of-core surface reconstruction. *Eurographics*, 2007.

[11] M. Bolitho, M. Kazhdan, R. Burns, e H. Hoppe. Parallel poisson surface reconstruction. *ISVC*, páginas 678–689, 2009.

[12] E. V. Chernyaev. Marching cubes 33: Construction of topologically correct isosurfaces. Relatório técnico, CERN, 1995.

[13] B. Curless e M. Levoy. A volumetric method for building complex models from range images. *SIGGRAPH*, páginas 303–312, 1996.

[14] J. Davis, S. R. Marschner, M. Garr, e M. Levoy. Filling holes in complex surfaces using volumetric diffusion. *3D-DPVT*, páginas 428–432, 2002.

[15] T. K. Dey e S. Goswami. A water-tight surface reconstructor. *ACM SMA*, páginas 127–134, 2003.

[16] H. Edelsbrunner. Shape reconstruction with delaunay complex. *Latin-Amer. Symp. Theoretical Informatics*, páginas 119–132, 1998.

[17] S. Fleishman, D. Cohen-Or, e C. T. Silva. Robust moving least-squares fitting with sharp features. *ACM TG*, 24(3):544–552, 2005.

[18] Leonardo Gomes. Pré-alinhamento automático de imagens de profundidade para modelagem 3d de objetos. Dissertação de Mestrado, Universidade Federal do Paraná, 2011.

[19] M. Gopi, S. Krishnan, e C. T. Silva. Surface reconstruction based on lower dimensional localized delaunay triangulation. *CGF*, 19(3):467–478, 2000.

[20] A. Hilton, A. J. Stoddart, J. Illingworth, e T. Windeatt. Reliable surface reconstructiuon from multiple range images. *ECCV*, páginas 117–126, 1996.

[21] A. Hornung e L. Kobbelt. Robust reconstruction of watertight 3D models from non-uniformly sampled point clouds without normal information. *Eurographics SIGRAPH*, páginas 41–50, 2006.

[22] K. Ikeuchi, T. Oishi, J. Takamatsu, R. Sagawa, A. Nakazawa, R. Kurazume, K. Nishino, M. Kamakura, e Y. Okamoto. The great buddha project: Digitally archiving, restoring, and analyzing cultural heritage objects. *IJCV*, 75(1):189–208, 2007.

[23] M. Kazhdan, M. Bolitho, e H. Hoppe. Poisson surface reconstruction. *Eurographics SIGRAPH*, páginas 61–71, 2006.

[24] R. Kolluri, J. R. Shewchuk, e J. F. OâĂŹBrien. Spectral surface reconstruction from noisy point clouds. *SIGRAPH*, páginas 11–21, 2004.

[25] Kyong-Ho Lee, Oliver Slattery, Richang Lu, Xiao Tang, e Victor Mccrary. The state of the art and practice in digital preservation. *Journal of Research of the National Institute of Standards and Technology*, 107(1):93–106, Janeiro de 2002.

[26] M. Levoy, K. Pulli, B. Curless, S. Rusinkiewicz, D. Koller, L. Pereira, M. Ginzton, S. Anderson, J. Davis, J. Ginsberg, J. Shade, e D. Fulk. The digital michelangelo project: 3D scanning of large statues. *SIGGRAPH*, páginas 131–144, 2000.

[27] M. Levoy e G. Turk. Zippered polygon meshes from range images. *SIGGRAPH*, páginas 311–318, 1994.

[28] W. E. Lorensen e H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. *SIGGRAPH*, páginas 163–169, 1987.

[29] T. Masuda. Object shape modelling from multiple range images by matching signed distance fields. *3D-DPVT*, páginas 439–448, 2002.

[30] Caroline M. Mendes, Dyego R. Drees, Olga R. P. Bellon, e Luciano Silva. Sistema para visualização de museu virtual 3D. *V Workshop of Undergraduated Work, SIBGRAPI*, 2007.

[31] D. Miyazaki, T. Oishi, T. Nishikawa, R. Sagawa, T. Tomomatsu, Y. Takase, e K. Ikeuchi. The great buddha project: Modelling cultural heritage through observation. *VSM*, páginas 138–145, 2000.

[32] Y. Ohtake, A. Belyaev, M. Alexa, G. Turk, e H. P. Seidel. Multilevel partition of unity implicits. *ACM TG*, 22(3):463–470, 2003.

[33] G. Pavlidis, A. Koutsoudis, F. Arnaoutoglou, V. Tsioukas, e C. Chamzas. Methods for 3D digitization of cultural heritage. *JCH*, 8(1):93–98, 2007.

[34] A. Pentland e S. Sclaroff. Closed-form solutions for physically based shape modeling and recognition. *IEEE TPAMI*, 13(7):715–729, 1991.

[35] Kari Pulli. Multiview registration for large data sets. *Proceedings. International Conference on 3D Digital Imaging and Modeling*, páginas 160–168, 1999.

[36] Chaua C. Queirolo, Luciano Silva, Olga R. P. Bellon, e M. Pamplona Segundo. 3D face recognition using simulated annealing and the surface interpenetration measure. *IEEE Trans. Pattern Anal. Mach. Intell.*, 32(2):206–219, 2010.

[37] C. Rocchini, P. Cignoni, F. Ganovelli, C. Montani, P. Pingi, e R. Scopigno. The marching intersections algorithm for merging range images. *VC*, 20(2-3):149–164, 2004.

[38] S. Rusinkiewicz, O. Hall-Holt, e M. Levoy. Real-time 3D model acquisition. *ACM TG*, 21(3):438–446, 2002.

[39] R. Sagawa e K. Ikeuchi. Taking consensus of signed distance field for complementing unobservable surface. *3DDIM*, páginas 410–417, 2003.

[40] R. Sagawa e K. Ikeuchi. Hole filling of a 3D model by flipping signs of a signed distance field in adaptive resolution. *IEEE TPAMI*, 30(4):686–699, 2008.

[41] R. Sagawa, K. Nishino, e K. Ikeuchi. Robust and adaptive integration of multiple range images with photometric attributes. *IEEE CVPR*, páginas 172179, 2001.

[42] R. Sagawa, K. Nishino, e K. Ikeuchi. Adaptively merging large-scale range data with reflectance properties. *IEEE TPAMI*, 27(3):392–405, 2005.

[43] J. Santos Junior, A. Vrubel, O. Bellon, e L. Silva. Improving 3D reconstructions for digital art preservation. *ICIAP*, 2011.

[44] J. Santos Junior, A. Vrubel, Olga R. P. Bellon, e Luciano Silva. Integração de malhas para a preservação digital de patrimônios culturais. *WTD SIBGRAPI*, Agosto de 2011.

[45] M. Soucy e D. Laurendeau. A general surface approach to the integration of a set of range views. *IEEE TPAMI*, 17(4):344–358, 1995.

[46] D. Terzopoulos, A. Witkin, e M. Kass. Constraints on deformable models: Recovering 3D shape and non-rigid motion. *JAI*, 36(1):91–123, 1988.

[47] A. Vrubel, O. Bellon, e L. Silva. A 3D reconstruction pipeline for digital preservation of natural and cutural assets. *IEEE CVPR*, páginas 2687–2694, 2009.

[48] Alexandre Vrubel. Reconstrução digital de objetos com scanners 3D e triangulação laser, 2008. Dissertação de Mestrado.

[49] M. D. Wheeler, Y. Sato, e K. Ikeuchi. Consensus surfaces for modeling 3D objects from multiple range images. *ICCV*, páginas 917–924, 1998.

[50] R. T. Whitaker. A level-set approach to 3D reconstruction from range data. *IJCV*, 29(3):203–231, 1998.

[51] Y. Yemez e C. J. Wetherilt. A volumetric fusion technique for surface reconstruction from silhouettes and range data. *CVIU*, 105(1):30–41, 2007.

[52] H. K. Zhao, S. Osher, e R. Fedkiw. Fast surface reconstruction using the level set method. *IEEE WVLSM*, páginas 194–201, 2001.