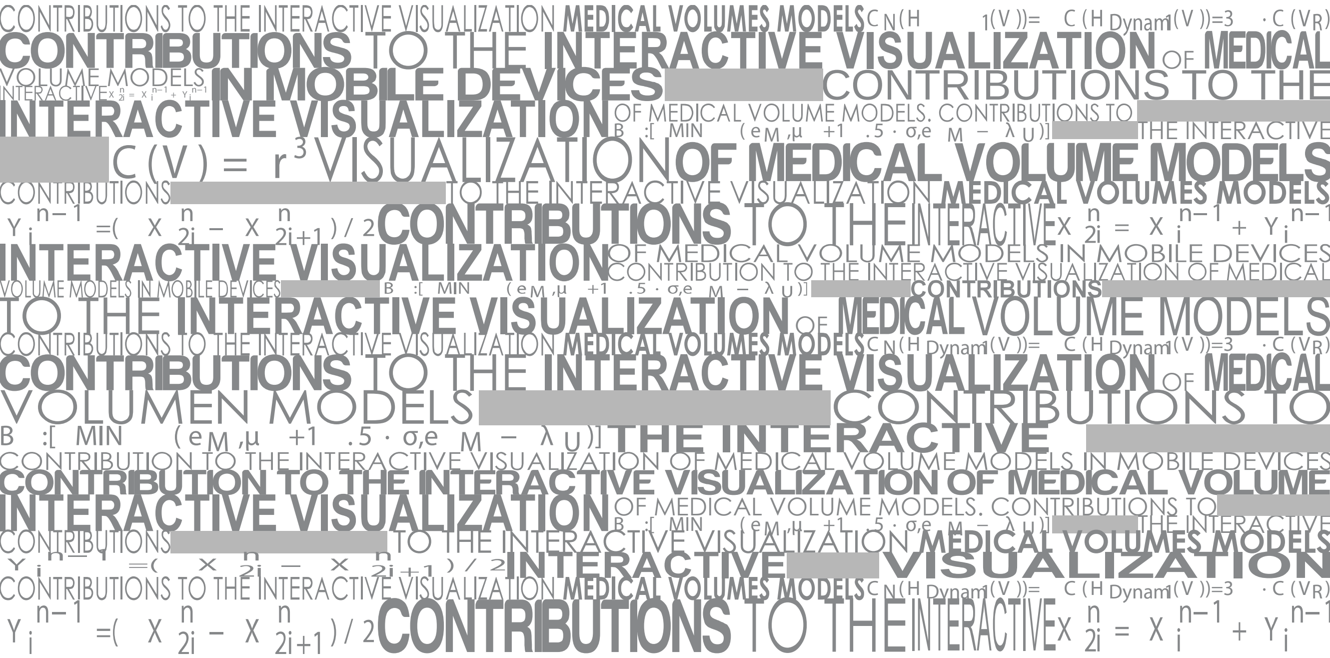
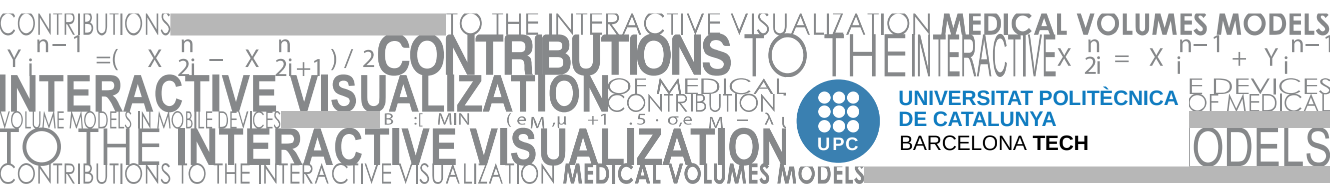




UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONA **TECH**

CONTRIBUTIONS TO THE INTERACTIVE VISUALIZATION OF MEDICAL VOLUME MODELS IN MOBILE DEVICES

PhD Programme in Computing

Doctoral Thesis

LAZARO CAMPOALEGRE VERA

ADVISORS: ISABEL NAVAZO AND PERE BRUNET

Acknowledgements

I would like to thank to my advisors Pere Brunet and Isabel Navazo. Their guide has been the guarantee to conclude this thesis. They have been exemplars, constants and patients. I cannot express enough gratitude to them, in front of the personalized attention, how they motivated me, and their unconditional support.

I also want to thank the UPC for the PhD formation Grant. I leave the UPC with the experience of the rigor, the familiarity, organization, and also the commitment that the institution puts in our hands for the development of our countries.

I would like to thank all the friends I found in this wonderful Barcelona. They opened me to their culture, they made me to love their languages and fanned my desire to explore the world. I also want to include in this acknowledgments the Cuban friends who maintained our friendship identically alive despite the distance; thanks to the worries and the sincere wishes of successes I got all the time. I have felt sheltered by all of them.

I want especially thank Jose Díaz and Eva Monclús for their experience and their technical support.

I would also like thank my great friend Pablo, for his patience in supporting my ideas and the dedication in the designing of this book cover.

Thank to my parents and sisters for the motivation they gave to me since the beginning of my studies, for their permanent emotional support and constant communication. This allowed me to close my eyes many times and feel I was at my own home.

I dedicate this thesis to my aunt Maria Esther, a woman who provided me the most pure and unconditional love until her last breath. This a very simple tribute to alleviate the nostalgia of her absence.

ABBREVIATIONS AND SYMBOLS

Abbreviations

CT	Computer Tomography
CTA	Coherent Traversal Algorithm
CTT	Coherent Traversal Tree
DCT	Discrete Cosine Transform
DFT	Discrete Fourier Transform
DPCM	Differential Pulse Code Modulation
DVR	Direct Volume Rendering
FT	Fourier Transform
FTT	Fast Fourier Transform
GUI	Graphical User Interface
HDR-VDP	High Dynamic Range-Visual Difference Predictor
IWT	Integer Wavelet Transform
LOD	Level of Detail
MC	Marching Cubes
MRI	Magnetic Resonance Images
PET	Positron Emission Tomography
PSNR	Peak Signal-to-Noise Ratio
PVTC	Packed Volume Texture Compression
RC	Ray Casting
ROI	Region of Interest
RLE	Run Length Encoding
RMS	Root Mean Square
TF	Transfer Function
VTC	Volume Texture Compression
VQ	Vector Quantization

Symbols

b	bit
$\boldsymbol{B}$	Bit stream.
B_l	Vector of Bits. Representation of Octree nodes at level l.
$C()$	Function to compute size in KBytes.
$C_N()$	Function to compute size of the transmitted data in KBytes.
D_l	Vector of gradient and materials for each node of the Octree at level l.
$\boldsymbol{D}$	Octree Data.
$\mathcal{D}(n)$	Pointer to associated data.
D_r	Vector of data at the deepest level r.
D_R	Data respresentation of the ROI Volume.
D_{R_r}	Compact representation of the ROI Volume.
e	Error matrix values.
e'	Quantized error values.
e_m	Minimum of error values.
e_M	Maximum of error values.
E_F	Group of matrices which share one face with the low frequency matrix.
E_E	Group of matrices sharing one edge with the low frequency matrix.
E_V	Matrix sharing one vertex with the low frequency matrix.
$\mathsf{f}(n)$	Pointer to the parent node.
$G(V)$	Gradient Octree of the volume V.
$G_R(V)$	Restriction of $G(V)$ to the ROI.
$H(V)$	Hybrid Model.
$H_{Stat}(V)$	Static Information of the Hybrid Model.
$H_{Dynam1}(V)$	Dynamic information of the Hybrid Model using D_R.
$H_{Dynam2}(V)$	Dynamic information of the Hybrid Model using D_{R_r}.
l	Level in the tree.
$l(n)$	Level of a node.
L	Laplacian filter.
$LB(n)$	Labeling function to represent nodes as bits.
$LD(n)$	Labeling function when $\mathsf{t}(n) = 1$.

λ_T	Smoothing Factor in the Laplacian Filter.
λ_U	Scalar to define pieces of the Uniforming Function U.
M_l	Number of grey nodes at level l.
n	Node.
O_l	Vector of indexes to the son nodes at level l.
σ	Standard Deviation.
p	Point of the original volume model.
P	Perceptual Metric.
Q	Quantization.
Q^{-1}	Dequantization.
R	Rendering Function.
r	Resolution of the volume model V.
S	Segmentation Transformation.
s_m	Pointer to son nodes.
$\boldsymbol{S}$	Octree structure.
$\mathcal{S}_{\mathcal{N}}$	List of octree structure arrays B_l transmitted over the network.
$\mathcal{S}_R^E$	Segmented regions of volume V.
T	Wavelet Transform (Chapter 3) and Gradient computation plus hierarchical downsampling (Chapter 4).
T_S	Planar Section.
$\mathtt{t}(n)$	Node type, 0 or 1.
t	Transfer Function in TF_k.
t_r	Tree root.
T^{-1}	Inverse of T.
$\{TF_k\}$	Set of Transfer Functions.
U	Piecewise Linear Uniforming Function.
U^{-1}	Inverse of the Piecewise Linear Uniforming Function.
V	Volume Model.
V_c	Vole with quantized Wavelet coefficients.
V_E	Edge Volume Model.
V_T	Volume Transformed.
V_R	Volume of the ROI.
V_R^E	Edge volume.
$V^{'}$	Reconstructed Volume.
V^t	Transformed and quantified volume.

W	3D Haar Wavelet Transform.
W^{-1}	Inverse of the 3D Haar Wavelet Transform.
w_l	Number of Wavelets reconstruction steps.
W_N	Transmitted information in the Wavelet region for the hybrid model.

Summary

With current medical imaging improvements, specialists are being able to obtain correct information of the anatomical structures of the human organism. By using different image visualization techniques, experts can obtain suitable images for bones, soft tissues, bloodstream among others. Present algorithms generate images with better and better resolution and information accuracy. Medical doctors are being more familiarized with three-dimensional structures reconstructed from bi-dimensional images. As a result, hospitals are becoming interested in tele-medicine and tele-diagnostic solutions. Client-server applications allow these functionalities. Sometimes the use of mobile devices is necessary due to their portability and easy maintenance. However, transmission time for the volumetric information and low performance hardware properties make quite complex the design of efficient visualization systems on these devices.

The main objective of this thesis is to enrich user experience during the interactive visualization of volumetric medical models in low performance devices. To achieve this, a new transfer-function aware compression/decompression mechanism adapted to transmission, reconstruction and visualization has been studied. This work proposes several schemes to exploit the use of transfer functions (TFs) to enhance volume compression during data transmission to mobile devices. As far as we know, this possibility has not been considered by any of the described approaches in the previous work.

The Wavelet-Based Volume Compression for Remote Visualization approach is a TF-aware compression scheme. It supports inspection of complex volume models with maximum level of detail in selected regions of interest (ROIs). It uses a GPU-based, ROI-aware ray-casting rendering algorithm in the client, with a limited amount of information being sent over the Network, decreasing storage size in the client side.

Regarding the Remote Exploration of Volume Models using Gradient Octrees scheme, we have shown that this technique can efficiently encode volume datasets. It supports high-quality visualizations with Transfer Functions from a predefined TFs set. In the present implementation, Transfer Function sets can encode up to ten different volume materials. Gradient Octrees are multi-resolution, supporting progressive transmission and avoiding gradient computations in the client device. That is, Gradient Octrees encode pre-computed gradients to save costly computations in the client, and support illumination-based ray-casting without extra computations in the client GPU. The proposed scheme presents a minimum loss of visual quality as compared to the state of the art ray-casting renderings. The octree structure is compacted into a small volume array and a set of texture-coded arrays, with only one bit per octree node. The proposed scheme supports planar volume sections which are visualized with high-resolution volume information, besides interactive extrusion of specific structures.

As a final contribution, a Hybrid ROI-based Visualization Algorithm has been proposed. It inherits the advantages of the previously described contributions while keeping a good performance in terms of bandwidth requirements and storage needs in client devices. The scheme is flexible enough to represent several materials and volume structures in the ROI area at high resolution with a very limited information transmission cost. The Hybrid approach has been proved to be specially well suited in the case of large models. Experimental results show that this Hybrid approach is a scalable scheme, with compression rates that decrease when the size of the volume model increases.

Resumen

Los adelantos actuales en imagenes médicas están permitiendo a los especialistas obtener información cada vez más precisa de las estructuras anatómicas del organismo humano. Mediante la utilización de diferentes técnicas de visualización, los expertos pueden obtener imágenes de calidad para los huesos, tejidos blandos, torrente sanguíneo, entre otros.

Los actuales algoritmos de procesamiento de imágenes garantizan el equilibrio entre la resolución y la exactitud de la información. Paralelamente, los médicos están más familiarizados con las estructuras tridimensionales reconstruidas a partir de imágenes en dos dimensiones. Por otro lado, los hospitales están incorporando la tele-medicina y el tele-diagnóstico entre sus soluciones técnicas. Las aplicaciones cliente-servidor permiten estas funcionalidades. En ocasiones el uso de dispositivos móviles es necesario debido a su fácil mantenimiento y a su portabilidad. Sin embargo, el tiempo de transmisión de la información volumétrica así como el bajo rendimiento del hardware en estos dispositivos, hacen que el diseño de sistemas eficientes de visualización sea todavía una tarea compleja.

El objetivo principal de esta tesis es enriquecer la experiencia del usuario en la visualización interactiva de modelos volumétricos de medicina en dispositivos de bajo rendimiento. Para conseguir esto, se ha puesto en práctica la implementación de un mecanismo de compresión/descompresión que depende de funciones de transferencia para optimizar la transmisión, reconstrucción y la visualización en estos dispositivos. Esta tesis, por lo tanto, propone varios esquemas para aprovechar el uso de las funciones de transferencia (TFs) e incrementar el ratio de compresión del volumen durante la transmisión a los dispositivos móviles. De acuerdo con nuestros conocimientos, ninguna de las técnicas descritas en los trabajos presentados anteriormente ha considerado esta posibilidad.

El esquema de compresión de volumen basado en Wavelets para la visualización remota, es una propuesta para compresión que tiene en cuenta la función de transferencia. Permite la inspección de modelos de volumen complejos con máximos niveles de detalles en regiones de interés seleccionados. El rendering ejecuta un ray-casting adaptado a modelos con regiones de interés orientado a la GPU en el cliente con una cantidad de información muy limitada que se envía por la red.

La otra contribución de esta tesis es la implementación de un esquema para la exploración remota de modelos volumétricos mediante **Gradient Octrees**. Esta técnica codifica de manera eficiente datos de volumen mientras garantiza visualizaciones de alta calidad con funciones de transferencias predefinidas en un determinado conjunto. La actual implementación permite codificiar hasta 10 materiales diferentes en los datos de Volumen. **Gradient Octrees** es una técnica multi-resolución, permite la transmisión progresiva y evita los cálculos del gradiente en el dispositivo cliente. En efecto, esta aproximación codifica gradientes previamente calculados para reducir el coste de los cálculos en la GPU del cliente y garantizar el ray-casting con iluminación en la GPU del dispositivo. En comparación con las propuestas estudiadas la pérdida de la calidad visual en los Gradient Octrees es mínima. La estructura del octree es compacta, compuesta de un pequeño vector de volumen y un conjunto de vectores de texturas codificadas, que utilizan sólo 1 bit por nodo del octree. El esquema soporta además secciones planas de volumen que contienen información de alta resolución, además de la extrusión de estructuras en los modelos visualizados.

La contribución final de esta tesis se concentra en regiones de interés. La propouesta aprovecha las ventajas de la implementación de un algoritmo de visualización híbrida de datos de volumen basado en regiones de interés. La propuesta incluye las ventajas de las contribuciones anteriores y mantiene un correcto rendimiento en términos de ancho de banda y capacidad de almacenamiento en el cliente. El esquema es lo suficientemente flexible como para representar varios materiales y estructuras de volumen en la región de interés con alta resolución; garantizando el bajo coste del proceso de transmisión.

CONTENTS

CHAPTER 1

INTRODUCTION

An introduction to our research methods and proposed algorithms.

Contents

1.1 Motivation

With the evolution of medical image techniques, specialists are being able to obtain correct information of the anatomical internal structures of the human organism. By using different image visualization techniques, experts can obtain suitable images for bones, soft tissues, bloodstream among others. At present, these systems generate images with better and better resolution and information accuracy. Handling these images allows the interpretation of large volume data sets. The standard DICOM (Digital Imaging and Communication in Medicine) allows portability and manipulation of these data, easing visualization and interaction with models.

Visualization of some damaged tissues and tumors helps the treatment in patients with oncological pathologies. A key use in chemotherapy is to know whether the tumor is growing or shrinking. The application of current visualization algorithms can improve the highlighting of these pathologies during medical examination.

Recently, several important research areas in three-dimensional techniques for multimodal imaging have appeared. Applications include neurological imaging for brain surgery [1], tissues characterization, medical school teaching, plastic surgery, surgical simulators [2] and others. At the same time, scientists are more familiarized with three-dimensional structures reconstructed from bi-dimensional images.

Volumetric models are generally achieved by using voxel datasets. A voxel representation consists of a three-dimensional array of voxels, each voxel being a cube which is the basic element in a volume representation. According to the medical structure to be highlighted during the visualization, a transfer function is applied to assign color and opacity to the values which represent the voxel properties. For large volume datasets, interactive visualization techniques require last generation graphic boards, due to the intensive calculation and memory requirements during rendering.

Nowadays, hospitals are more interested in tele-medicine and tele-diagnostic solutions. Tele-medicine [3] [4] is defined as the use of medical information

exchanged from one site to another via electronic communications to improve the clinical health status of patients. This concept includes a growing variety of applications and services using two-way video, email, smart phones, wireless tools and other forms of telecommunications technology. Clinically oriented specialities can capture and remotely display physical findings, transmit specialized data from tests and carry out interactive examinations [5].

Tele-diagnostic is allowed by using tele-medicine, known as a process whereby a disease diagnosis, or prognosis, it is made by the electronic transmission of data between distant medical facilities.

Some applications for the remote visualization of medical images can be considered as tele-medicine approaches. The handling of three-dimensional information requires efficient systems to achieve fast data transmission and interactive visualization of high quality images.

Client-server applications allow these functionalities. Sometimes the use of mobile devices is necessary due to their portability and easy maintenance. However, transmission time for the volumetric information and low performance hardware properties make quite complex the design and implementation of efficient visualization systems on these devices.

1.2 Thesis statement

The objective of this thesis is to enrich user experience during the interactive visualization of volumetric medical models in low performance devices, by studying new transfer-function aware compression/decompression mechanisms adapted to transmission, reconstruction and visualization in those devices.

1.3 Formal Framework

Our initial hypothesis is to exploit the use of standard transfer functions as a way to compress datasets. What we want to achieve while visualizing medical images in mobile devices, is to obtain the best possible image quality with an acceptable size of data in a suitable time. This therefore is a **Constrained Optimization Problem**, as shown in what follows:

Let us assume that we have a volume model V. Let us consider a rendering function $R(V,t)$, which generates rendered images from the original model V through a specific transfer function t. The compression in the server is based on three operators (S, T and Q) that work on V:

$$V^t = Q \cdot T \cdot S(V,t) \tag{1.1}$$

A first segmentation S transforms the data to minimize the loss of visual quality during rendering (R). We will note $S(V,t)$ the result of segmenting V by using the transfer function t. After that, a transformation T is applied followed by a quantization scheme Q.

The first segmentation transformation, S, filters those details in V that will not appear in the rendered images. The goal of S is to achieve:

$$R(S(V,t),t) \simeq \ R(V,t) \tag{1.2}$$

with $S(V,t)$ being less complex than V. The transformed and quantified volume V^t is then transmitted to the client, where a reconstruction V' is computed:

$$V' = T^{-1} \cdot Q^{-1} \cdot V^t \tag{1.3}$$

The inverse transformations of T and Q being simple enough to support their real time execution in the client.

Our intention is to achieve the highest perceptual quality rendering in the client device. Assuming a perceptual metric P and naming $C(V)$ the spatial

complexity (memory requirements in KBytes) of a volume model V, the first goal is to maximize P:

$$Max\ [P(R(V',t),R(V,t))]\ \mid\ C(V') \leq\ Max\ Size \tag{1.4}$$

Where P compares the rendered images of V in the server and of V' in the client with values between 0 and 1, with P=1 meaning a maximum of perceptual similarity. The maximization is subjected to the constraint of a bounded $C(V')$. $C(V')$ must be small enough in the client to allow proper user interaction.

We want to send the whole volume model on demand and in a compressed way to the client device. This ensures full interaction facilities in the client without further server-client communication. We must therefore consider network transmission as a further constraint. If we name $C_N(V)$ the amount of information that is transmitted over the network, we must also consider $C_N(V)$ in our final goal. Hence, the final objective is:

$$Max\left[\ (1-\lambda)P(R(V',t),R(V,t)) + \lambda\cdot\left(1-\frac{C_N(V)}{C(V)}\right)\right]\ \mid\ C(V') \leq\ Max\ Size \tag{1.5}$$

where λ is a normalization constant.

We propose two basic techniques to achieve this objective, both based on perceptual metrics and transforming the pre-processed volume in a hierarchical multi-resolution volume data representation.

- In Chapter 3: S is a Laplacian Filter and T the Wavelet transform. The Laplacian Filter is a smoothing operator that is applied in segmented regions defined by a transfer function. It forces voxel density values to remain in the interval of density values defining this region. After smoothing, the volume is subdivided in blocks to independently compute a Haar Wavelet (which is the transform T in the previous scheme) and it is quantized through a well-suited Q algorithm.
- In Chapter 4: S is a volume segmentation process $V \rightarrow V_E$ and T includes the gradient computation plus a hierarchical downsampling. The volume segmentation process is induced by each transfer function of the predefined set of transfer functions to create an Edge Volume

V_E, which contains the basic information for the construction of a multiresolution voxel model by means of a hierarchical downsampling. This process computes and compresses the vector gradient for *grey* nodes of this representation.

1.4 Contributions

Medical doctors usually adopt standard transfer functions to visualize anatomical structures, this being a common practice in radiology. Based on this fact, the thesis presents two transfer function-aware schemes for remote interactive inspection and expressive visualization of volume models in client-server architectures. These approaches codify volume models from CT (Computed Tomography) data.

The main contribution of this thesis is the analysis and use of:

- **Transfer Function-aware Compression Schemes:** The proposed schemes exploit the use of standard transfer functions to compress the volume dataset during its transmission to mobile devices. As far as we know, this possibility has not been considered by any of the described approaches in the previous work, (see Chapter 2).

This main contribution develops into the following specific contributions:

- **A Wavelet-Based Volume Compression for Remote Visualization Scheme:** A Wavelet-based, structure-aware compression scheme for 3D voxel models is proposed. It is designed for client-server architectures and offers interactive volume visualization on mobile devices. The scheme is block-based, supporting adaptive visualization in the client. Decompression is simple enough to be run in real time in the client side. A two-level ray-casting allows focusing on small details on targeted regions while keeping bounded memory requirements on both the CPU and the GPU of the client.

- The contributions of this scheme resulted in the publication of the paper [6]: Lázaro Campoalegre, Pere Brunet and Isabel Navazo. Interactive visualization of medical volumes in mobile devices. *Personal and Ubiquitous Computing*, Volume 17, Issue 7, 2013.

- **A Remote Exploration of Volume Models using Gradient Octrees:** A transfer function-aware scheme for remote interactive inspection of volume models in client-server architectures is proposed, with the objective of supporting multi-resolution, avoiding gradient computations in the client device and sending a very limited amount of information through the network. The novel GPU-oriented data representation (named *Gradient Octrees*) can be progressively transmitted to the client in a compact way while achieving a minimum loss of visual quality as compared to state of the art ray-casting renderings. It is based on the concept of Coherent-Traversal Trees and Coherent-Traversal Algorithms (CTA). Visual volume understanding can be complemented by showing 2D sections of the original volume data on demand, and by a number of additional interactive tools.

- The contributions of this scheme resulted in the publication of the paper [7]: Lázaro Campoalegre, Pere Brunet and Isabel Navazo. Gradient Octrees: A new Scheme for Remote Interactive Exploration of Volume Models. *Proceedings of the CAD/Graphics*, Hong Kong, 2013.

- **A Hybrid ROI-based Visualization Algorithm:** A technique that inherits the advantages of the previous contributions while keeping a good performance in terms of bandwidth requirements and storage needs in client devices is also proposed. The scheme is flexible enough to represent several materials and volume structures in the ROI area at high resolution using Gradient Octrees, all in all at a very limited information transmission cost. The rest of the volume is represented by the Wavelet-based approach.

- The contributions of this scheme resulted in the paper [8]: Lázaro Campoalegre, Isabel Navazo and Pere Brunet. Hybrid, ROI-Based Inspection of Medical Volume Models in Mobile Devices. Spanish Conference on Computer Graphics, CEIG 2014.

1.5 Organization of the Manuscript

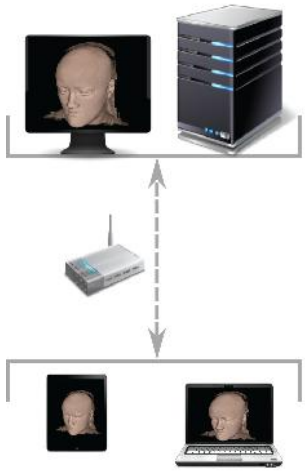
The remainder of this document is organized as follows:

Chapter 2 describes and compares current client server architectures for volume visualization and discusses the status of the visualization in low-end devices. Chapter 3 presents a wavelet-based approach for volumetric medical images visualization. Chapter 4 describes the Gradient Octrees approach and its contributions to the remote exploration of medical volume models. In Chapter 5 the Hybrid visualization approach and a comparison among the previously studied techniques are presented. Finally, Chapter 6 includes the thesis conclusions and future work.

CHAPTER 2

CLIENT-SERVER VOLUME VISUALIZATION

This chapter briefly describes and compares current client server architectures for volume visualization and volume compression techiques.



Contents

2.1 Introduction

The term client-server refers to a popular model for computer applications that utilizes client and server devices each designed for specific purposes. This kind of applications can be used on internet settings or local network as well [9].

Client-server architectures have grown in popularity since many years ago as personal (PCs) computers became an alternative to mainframe computers. Therefore, client devices are typically PCs with network software applications installed that request and receive information over the network [10]. Mobile devices as well as desktop computers can both function as clients.

A server device typically stores files and databases. These devices often feature higher-powered central processors, more memory, and larger disk drives than clients.

For computer graphics applications the mechanism is the same, a powered machine works as server while low-performance devices are used as client [11] [12].

In this chapter we present a state of the art describing and comparing some recent and significant papers related to the client-server architecture proposals for volume visualization.

Remote visualization of medical images is a highly selected area for scientists during the last years [13, 14, 15]. Many authors have published research results in the remote volume visualization area. However there is still scarce specific bibliography for volume visualization in mobile devices. The majority of the proposals use known algorithms like Ray-Casting, 2D Textures, and isosurface modeling to render volume data. In general, in order to compensate limitations in low performance devices or to reduce costs, high number of client-server schemes have been proposed.

2.2 Client-Server Architectures

A wide study and revision of the published work have allowed us to identify four principal schemes of client-server architectures for volume rendering. Basically we classify them according to the way they reduce volume information for transmission: volume compression, image compression, data partitioning or volume pre-processing.

Sending the compressed volume: In the first group of approaches (see Figure 2.1), the dataset is compressed in the server side and sent to the client where the transfer function is applied after decompression and before rendering the recovered data. We also include in this group the approaches that send the whole volume to the clients, without compression.

Concerning this mechanism, Callhan *et al.* [16] presented an isosurface based method for hardware assisted progressive volume rendering. Their approach seeks to minimize the size of the data stored in the final client in each step during data sending. The approach sends a compressed vertex array to the client during the reconstruction of the model.

Moser and Weiskopf [17] proposed a 2D texture-based method which uses compressed texture atlas to reduce interpolation costs. The approach proposes a hybrid high/low resolution rendering, to combine volume data and additional line geometries in an optimized way. By doing this, they achieve interactive frame rates. The technique runs on a mobile graphics device natively without remote rendering. Mobeen *et al.* [18] proposed a single-pass volume rendering algorithm for WebGL platform. They built an application with a transfer function widget which enables feature enhancement of structures during the rendering. To avoid 3D texture limitations of some devices, they mapped the volume into a single 2D texture to develop an application able to run in any modern device with a basic graphic processor.

A recent application developed by Balsa *et al.* [19] allows interacting with volume models using mobile devices hardware. They are able to apply different transfer functions to volumes while selecting among 2D, 3D, and

ray-casting methods according to the hardware capabilities. Their scheme is not compressing the volume data.

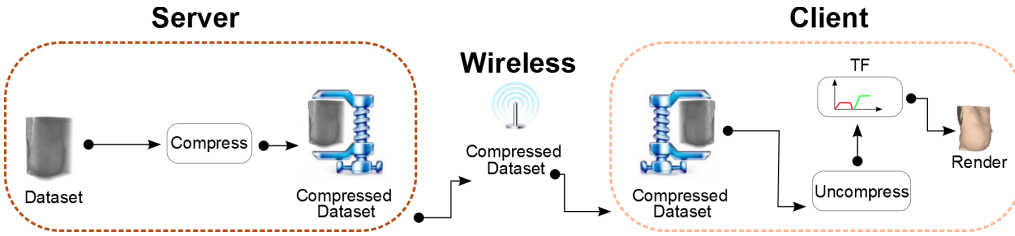


Figure 2.1: Client-Server Architecture, Case A: the dataset is sent to the client in a compressed or uncompressed way. The client applies the transfer function after decompression and before rendering.

Sending compressed 2D rendered images: In the schemes where the transmitted data is a compressed image [20], [21], (see Figure 2.2), the transfer function is applied at the beginning of the pipeline, followed by a 2D rendering on a texture, all done in the server side. A compressed image is sent to the client where decompression and image rendering take place. This scheme is frequently named ***Thin Clients*** [22].

Engel *et al.* [23] developed an approach that uses Open Inventor Application which provides a scene graph programming interface with a wide variety of 3D manipulation capabilities. The application renders images off-screen, encodes images on-the-fly and transmits those images to the client side. Once in the client, images are decoded and copied into a framebuffer. The client interface also provides a render drawing area with mouse event handling capabilities to display images.

A new remote visualization framework is proposed in [24], here the dataset is loaded into slicing tool in the client side. The designed tool allows axial, coronal and sagittal direction inspections of medical models. The application allows the selection of a sub-region by using object aligned textures. Volume data is transferred to the server side to increase visualization quality. In a similar way to other techniques, the server first renders images off-screen, compress the image and the result is sent to the client. Once in the client side, the image is decompressed and rendered. Mouse and GUI events are sent to the server for re-rendering operations. Qi *et al.* [25] designed a medical application to send images in a progressive way. The approach

creates a reference image of the entire data by applying transforms. The encoding scheme allows the gradual transmission of the encoded image, which is reconstructed on-the-fly during the rendering in the client side.

Constantinescu *et al.* [26] implemented an application that incorporates Positron Emission Tomography/Computer Tomography (PET/CT) data into Personal Health Record for remote use on internet-capable or handheld devices. It is a client/server application designed to display images in final low-end devices as mobile phones. Users can control brightness and contrast, apply color look up table and view the images in different angles. The approach allows the transmission of images with a enough refresh rate to achieve interactive exploration of 2D images.

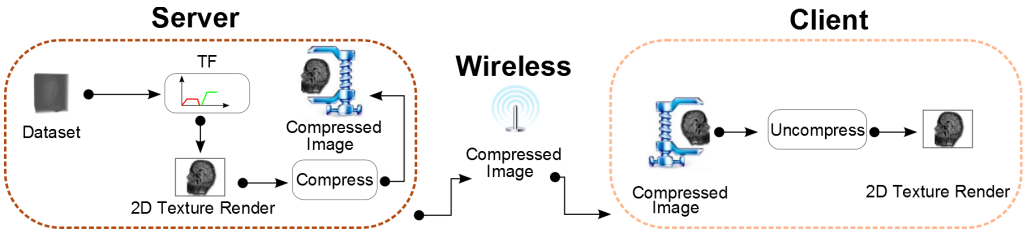


Figure 2.2: Client-Server Architecture, Case B: the transfer function is applied in the server, which also renders the volume data. The information sent to the client consists on compressed 2D images.

Jeon and Kaufman [27] implemented a virtual colonoscopy application using wireless with a Personal Digital Assistant (PDA) as client device. In this scheme the server performs a GPU-based direct volume rendering to generate an endoscopy image during the navigation, for every render request from the client.

An explorable images technique is proposed by Tikhonova *et al.* [28]. The approach converts a small number of single-view volume rendered images of the same 3D data set into a compact representation. The mechanism of exploring data, consists in interact with the compact representation in transfer function space without accessing the original data. The compact representation is build by automatically extracting layers depicted in composite images. Opacity and color are achieved by the different combination of layers.

Partitioning the volume data: Some approaches achieve a reduction

of the sent information by partitioning the rendered volume (see Figure 2.3). This kind of proposals perform a partition of the volume to be sent to the client, where the composition of the entire volume and the rendering is applied.

Bethel. [29] proposes to subdivide and render volumes in parallel. The resulting set of 2D textures is sent to a viewer which uses 2D texture mapping to render the stack of textures, and provides interactive transformations.

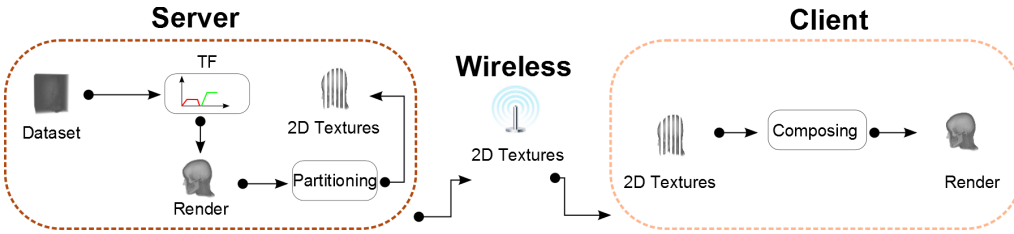


Figure 2.3: Client-Server Architecture, Case C: the server partitions the volume into a set of 2D slices, represented as 2D textures. The information sent to the client consists on a stack of 2D textures.

Sending compressed multiresolution volume information: In some proposals data preprocessing ensures the reduction of the information, combined with different techniques for quantization, encoding and multiresolution representation. (see Figure 2.4)

In this group of approaches, a networking application is proposed by Lippert *et al.* [30]. Here, a local client with low computational power browses volume data through a remote database. The scheme allows the treatment of intensity and RGB volumes. A Wavelet based encoding scheme produces a binary output stream to be stored locally or transmitted directly to the network. During rendering, the decoded Wavelet coefficients are copied into the accumulation buffer inside GPU. The bandwidth of the network and the frame rate control the transmission of the Wavelet coefficients in significance order to guarantee the rendering quality.

Boada *et al.* [31] proposed an exploration technique where volume data is maintained in the server in a hierarchical data structure composed of nodes. The server receives the user parameters to select the correct list of nodes to be rendered in the client side according to its hardware capabilities. As a second rendering possibility, the user can select a region of interest

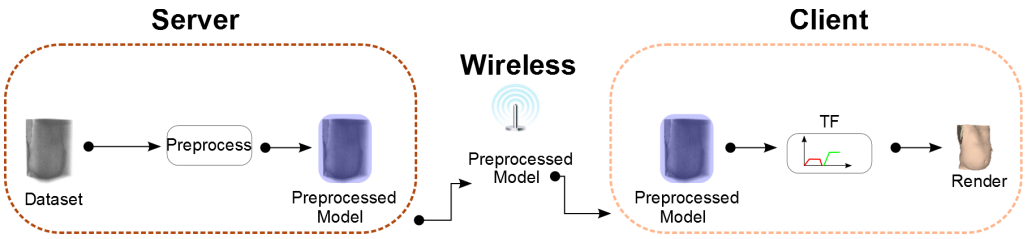


Figure 2.4: Client-Server Architecture, Case D: the server enriches the volume dataset by computing a multiresolution volume hierarchy, which is then compressed and sent to the client.

(ROI) of the entire volume. To achieve this, the server transmits data in an incremental fashion. Recently Gobebtti *et al.* [32], proposed a progressive transmission scheme to send compact multiresolution data from the server to the client stage. The technique allows fast decompression and local access to data according to the user interaction in the client side.

Table 2.1 shows a description of the studied proposals. The columns show for each approach, the above client-server schemes which applies, the kind of data sent through the network and the compression technique implemented to compact the data. It also shows where a mobile device is used as client, and an estimation (based on the *fps* of the clients) of latency and interactivity of the implemented approaches.

A simple analysis of this table, shows that few techniques were designed to run on mobile devices as final clients. Although techniques presented in [17, 18] achieve visualizations in mobile devices, the limited size of the models and the lack of advance lighting shading implementations leave a gap for further research in the area.

Latency and interactivity are strongly associated concepts in client-server architectures for volume visualization. The ability of achieving interactive frame rates depends on the transmission procedure and the rendering algorithm implemented in both servers and clients. Some techniques achieve a good combination of this properties by applying progressive transmission schemes[16] and adaptive rendering algorithms[17, 32]. But unfortunately this techniques are still quite complex to be run in low-end devices.

Table 2.1: Client-Server Architecture Schemes. Columns show the client-server architectures (Figures 2.1...2.4, the rendering algorithm, the kind of data being sent to the clients, the compression scheme (Table 2.2). The last columns show whether a mobile devices is a client or not, and finally a estimation of latency and interactivity for each technique.

						Latency			**Interactivity**		
Ref.	**Arch.**	**Rendering Algorithm**	**Data sent**	**Comp. Scheme**	**Mobile**	**L**	**M**	**H**	**L**	**M**	**H**
[29]	C	2D Texturing	Images	-	×	✓				✓	
[31]	D	3D Texturing	Vertices, Indexes	-	×		✓		✓		
[16]	A	Iso-Surface	Vertices	A	×			✓			✓
[26]	B	2D Texturing	Images	-	✓		✓		✓		
[24]	B	2D Texturing	Images	A	×		✓			✓	
[23]	B	3D Texturing	Images	A	×		✓			✓	
[32]	D	RC	Octree Nodes	C	×	✓					✓
[27]	B	2D-Texturing	Images,Points	A	✓		✓			✓	
[30]	D	2D Texturing	Wavelet Coeff.	A	×	✓				✓	
[17]	A	2D Texturing	Intensities	C	✓	✓					✓
[18]	A	2D Texturing, RC	2D Texture Atlas	-	✓		✓			✓	
[33]	B	2D Texturing	Images	A	✓	✓				✓	
[19]	A	2D,3D , RC	Intensities	A	✓	✓				✓	
[28]	B	2D Texturing	Images	-	✓	✓				✓	
[25]	B	2D Texturing	Images	A	✓	✓			✓		
[34]	B	2D Texturing	Images	A	×	✓					✓

2.3 Compression Techniques for Volume Models

As we have mentioned above, dataset transmission from servers to final clients is considered a very important stage in client-server architectures for volume visualization. Efficient schemes require optimized algorithms to reduce the data and to send them through the networks. The algorithms must achieve the maximum compression possible while allowing an easy decompression in the client side, where sometimes hardware and memory constraints decrease performance.

Compression algorithms can be classified in lossless and lossy [35, 36]. These are terms that describe whether or not all the original data can be recovered when the information is decompressed. With lossless compression, data that was originally in the file, is recovered after the entire information is uncompressed. On the other, lossy compression schemes reduce data by permanently eliminating certain information, especially redundant information.

Some recent and relevant compression techniques for volume visualization have been studied, few of them have been included in client-server architectures solutions. In Table 2.2, we present a comparison among these proposals. Columns show the stage of the pipeline where decompression takes place, whether the compression is lossless or lossy, and the applied compression technique. We also specify those techniques designed for medical image applications [37] and whether progressive transmission is allowed. The last columns show an estimation of the compression ratio as well as a qualitative measure of the reconstruction quality.

Wavelets and Vector Quantization are popular techniques for those approaches in which decompression takes place in CPU. Wavelet transforms offer considerable compression ratios in homogeneous regions of an image while conserving the detail in non-uniform ones. The idea of using 3D Wavelets for volume compression was introduced by Muraki [38]. One of the limitations of this approach was the cost to access individual voxels. In [39] a lossy implementation of $3D$ Wavelets transform was applied to a real volume data generated from a series of 115 slices of magnetic resonance

images (MRI). By applying a filtering operation three times, the approach obtains a multiresolution representation of a volume of 128^3 voxels. Using the coefficients from the Wavelet functions, they reconstructed a continuous approximation of the original volume at maximum resolution. The rendering technique prevents an interactive scheme, due to the cost of finding the intersection point of the ray with a complexm 3D function, and consumes a considerable amount of time. Ihm and Park [40] proposed an effective 3D 16^3-block-based compression/decompression Wavelet scheme for improving the access to random data values without decompressing the whole dataset and allowing volume render.

Focusing now on those techniques where decompression takes place in GPU, Guthe *et al.* [41] proposed a novel algorithm that use a hierarchical Wavelet representation. The Wavelet filter is locally applied and the resulted coefficients are the basic parameters for a threshold quantization based scheme. During rendering, the required level of the Wavelet representation is decompressed on-the-fly and rendered using graphics hardware. The scheme allows the reconstruction of final images without noticeable artifacts.

Current bottlenecks of Wavelet based volume compression schemes are the lack of locality and the complexity of the decompression in low-end devices. Moreover, present approaches are always compressing the whole volume, even if the TF is forces most of the medical structures to become invisible. In the first group of approaches (see Decomp. Stage: In CPU, from table 2.2) most of the implementations are lossy due to the application of quantization/encoding schemes (see also table 2.3). Nguyen *et al.* [42] proposed a block based technique to compress very large volume data sets with scalar data on a rectilinear grid. The method works in Wavelet domain. By using different quantization step sizes, the technique encodes data at several compression ratios. Although they ensure that compared to similar proposals their approach achieves better reconstruction quality, the resulting images show the existence of small blocking artifacts due to the block based coder. Furthermore, they can only perform two compression steps with too limited multiresolution capabilities.

Many methods try to maintain genuine volumetric data during the quantization stage. Rodler [43] proposes instead, to treat two dimensional slices in position or time and draw on results developed in the area of video coding. The first step of their encoder removes the correlation along the z-direction,

assuming that two-dimensional slices are divided along this direction. A 3D Wavelet decomposition should be ideal to further remove correlation in the spacial and temporal directions. But in order to decrease computational costs, they adopt as second step a 2D Wavelet transform to handle the spacial redundancy. Finally, the quantization continues by removing insignificant coefficients to make the representation even sparser. The method is capable of providing high compression rates with fairly fast decoding of random voxels. They achieve high compression rates with notable cost in the decompression speed. The approach in [25] works on 3D medical image sets, but it is finally a 2D visualization scheme to be performed in CPU. The proposal has also been restricted to Magnetic Resonance Images (MRI) datasets with relative large distance between slices. Even though authors say the approach is a lossless compression scheme; the averages and thresholds performed to the images, result on an accuracy reduction of the information and therefore in a lossy compression approximation. Although the working with slices has the advantage that memory format is identical to that of the final 3D texture used for rendering, this comes at the cost of losing spatial coherence.

Vector Quantization [44] is one of the most explored techniques for volume compression. Basically consists on decreasing the size of volumetric data by applying a specific encoding algorithm. The basic idea of this lossy compression method is to code values from a multidimensional vector space into values of a discrete subspace of lower dimension. Ning and Hesselink [45] were the first ones in applying vector quantification to volume models. In their scheme, the volume dataset is represented as indexes into a small codebook of representative blocks. The approach is suitable for a CPU-based ray-casting render. The proposed system compresses volumetric data and renders images directly from the new data format. A more efficient solution was proposed in [46], here the volume dataset is presented as indexes into a small codeboook of representative blocks. This structure, allows volume shading computations to be performed on the codebook, and image generation is accelerated by reusing precomputed block projections. Schneider and Westermann [47] implemented a Laplacian pyramid vector quantification approach that allows relatively fast volume decompression and render on the GPU. However this method does not allow using the linear filtering capabilities of the GPU and the render cost increases when using high zoom factors. Eric B. Lum *et al.* [48] propose a palette-based

decoding technique and an adaptive bit allocation scheme. This technique fully utilizes the texturing capability of 3D graphics card.

Bricking techniques subdivide large volumes into several blocks, named bricks, in such a way each block fits into GPU. Bricks are stored in main memory, then they are sorted either in front-to-back or back-to-front order with respect to the camera position, depending on the rendering algorithm [49, 50].

The idea in multiresolution model schemes [51, 52, 53] is to render only a region of interest at high resolution and to use progressively low resolution when moving away from that region. Both bricking and multiresolution approaches need a high memory capacity on the CPU for storing the original volume dataset. Moreover, bricking requires a high amount of texture transfers as each brick is sent once per frame; multiresolution techniques have been built for CPU purposes and its translation to GPUs is not straightforward due to the required number of texture fetching. As Table 2.2 shows, different techniques allow multiresolution. An old technique proposed by Ghavamnia *et al.* [54], consists in the use of the Laplacian Pyramid compression technique, which is a simple hierarchical computational structure. By using this representation, a compressed volume data can be efficiently transmitted across the network and stored externally on disk.

Progressive transmission has become an important solution for client-server architectures, allowing sending large volume dataset to the clients according to the rendering possibilities as well as hardware and network constraints in both servers and clients [55]. Xiaojun Qi *et al.* [25] propose a progressive transmission capable approach. Here the information reduction addresses the transmission scheme. The approach basically compresses data by reducing noise outside the diagnostic region to each image of the 3D image set. They also reduce the inter-image and intra-iamge redundancy adjusting pixel correlations between adjacent images and within a single image. By applying a Wavelet decomposition feature vector, they select a representative image from the representative subset of the entire 3D medical image set. With this reference image, they achieve good representation of all data with a best contrast and anatomical feature details. The encoding technique (Table 2.3) ensures the progressive transmission scheme. The codified version of the reference image is transmitted gradually from the coarse version to the fine one. In medical applications, radiologists can determine during transmission

whether the desired image is reconstructed and stop the sending of the information before the transmission of the entire image set. Menmann *et al.* [56] present a hybrid CPU/GPU scheme for lossless compression and data streaming. By exploiting CUDA they allow the visualization of big out-of-core data with near-interactive performance.

More recently, Suter *et al.* [57] have proposed a multiscale volume representation based on a Tensor Approximation within a GPU-accelerated rendering framework. It can be better than Wavelets at capturing non-axis aligned features at different scales. Gobbetti *et al.* [32] have proposed a different multi-resolution compression approach using a sparse representation of voxel blocks based on a learned dictionary. Both approaches allow progressive transmission and obtain good compression ratios, but they are lossy and require huge data structures and a heavy pre-process.

Table 2.3 summarizes the compression pipeline for the techniques presented in Table 2.2. The studied approaches have not been designed to use in mobile devices as clients, and none of them are transfer-function aware. Some of these techniques are not even designed for client-server architectures, but for compressing data from disk or to decrease bandwidth limitations.

Compression quality is usually measured by computing rate distortion curves for representative datasets. A common measure in image compression is the Peak Signal-to-Noise Ratio (PSNR) [58]. Fout *et al.* [59] designed a hardware-accelerated volume rendering using the GPU. The approach is a block-based transform coding scheme designed specifically for real-time volume rendering applications. An efficient decompression is achieved using a required data structure that contains 3D index volume to the codes and codebooks textures. Guitián *et al.* [60] implemented a complex and flexible multiresolution volume rendering system capable to interactively drive large-scale multiprojector. The approach exploits view-dependent characteristics of the display to provide different contextual information in different viewing areas like field displays. The proposal achieves high quality rendered images in acceptable frame rates.

Table 2.2: Compression Schemes. Columns show the stage of the pipeline where decompression takes place, whether the compression is lossless or lossy, and the applied compression technique. Table also specifies those techniques designed for medical image applications and whether progressive transmission is allowed. The last columns show an estimation of the compression ratio as well as a qualitative measure of the reconstruction quality.

			Compression								Comp. Ratio			Reconst. Quality		
Decomp. Stage		Ref.	Lossless	Lossy	Comp. Technique	Medical Images	Multiresolution	Prog. Trans	Mobile	L	M	H	L	M	H
In CPU		[39]		✓	Wavelets	✓	✓	✓	×		-		✓		
		[61]	✓		Wavelets	-	✓	✓	×	✓				✓	
		[42]		✓	Wavelets	-	✓	-	×			✓		✓	
		[43]		✓	Wavelets	-	✓	-	×	✓				✓	
		[41]		✓	Wavelets	✓	✓	✓	×	✓				✓	
		[62]		✓	Wavelets	-	✓	-	×	✓				✓	
		[30]	✓		Wavelets	✓	✓	✓	×		-				✓
		[63]		✓	Wavelets	✓	✓	-	×		✓				✓
		[45]		✓	VQ	✓	✓	-	×		✓				✓
		[46]		-	VQ	✓	-	-	×		-		✓		
		[64]	✓		Huffman	✓	-	-	×		-		✓		
		[25]	✓		Wavelets	✓	-	-	×	✓			✓		
		[65]		✓	Fourier Transf.	✓	✓	-	×			✓		✓	
		[66]		✓	Fourier Transf.	✓	✓	-	×		✓			✓	
		[67]		✓	Fourier Transf.	-	-	-	×			✓		✓	
		[48]		✓	VQ	-	✓	-	×		✓				✓
		[68]		✓	Wavelets	-	✓	✓	×		-			✓	
		[57]		✓	Tensor Approx.	-	✓	✓	×			✓			✓
In GPU	During Rendering	[47]		✓	VQ	✓	✓	✓	×	✓			✓		
		[56]		✓	LZO	-	✓	✓	×		✓			✓	
		[69]		✓	Block-Based Codf.	✓	-	✓	×		-			✓	
		[70]		✓	VTC-LZO	-	-	-	×	✓				✓	
		[60]		✓	Frame Encod.	-	✓	-	×			✓			✓
		[71]		✓	DCT	✓	✓	-	×			✓	✓		
		[54]		✓	Laplacian Pyramid	✓	✓	-	×		✓			-	
		[32]	✓		Linear Comb.	✓	✓	✓	×		✓				✓
		[59]		✓	texture Comp.	-	-	-	×			✓			✓

Table 2.3: Compression Schemes Pipeline. Columns show the kind of Input data used in each approach, the preprocessing and encoding techniques applied. The table also shows the data structure or function to represent data, and finally, the decoding and rendering algorithms implemented.

Ref.	Input Data	Preprocessing/Encoding	Data Representation	Decoding	Rendering
[39]	MRI	-	3D Orthonormal Wavelets	-	Ray Casting
[61]	RM Instability	RLE+Huffman Coding	3D Haar Wavelets	RLE+Huffman Decoding	Ray Casting
[42]	CT	Quantization of Wavelet Coefficients	3D Haar Wavelets	Dequant. Wavelet Coefficients	-
[43]	CT	Quantization of Wavelet Coefficients	3D Haar Wavelets	Dequant. Wavelet Coefficients	Ray Casting
[41]	-	Block Subdivision/Encoding Wavelet Coefficients	Wavelets	Decoding W. Coefficients	3D texturing
[62]	Scanning Model	Encoding Wavelet Coefficients	3D Orthogonal Wavelets	Decoding W. Coefficients	Ray casting
[30]	MRI,CT	Encoding Wavelet Coefficients	Wavelets(Haar/B-Splines)	Decoding W. Coefficients	3D texturing
[63]	MRI,CT	Encoding Wavelet Coefficients	3D Haar Wavelets	Decoding W. Coefficients	Ray Casting
[45]	-	VQ. Encoding	Code Books	VQ. Decoding	2D Texturing
[46]	CT	VQ. Encoding	Code Books	VQ. Decoding	2D Texturing
[64]	MRI,CT	DPCM+Huffman Coding	Encoding Stream	Decoding DCPM + Huffman	2D Texturing
[25]	MRI,CT	DPCM+Huffman Coding	Encoding Stream	Decoding DCPM + Huffman	2D Texturing
[65]	-	-	Packing Textures	Decod. of Compressed Blocks	2D Texturing
[66]	MRI	-	3D Fourier Transform	-	Ray Casting
[67]	-	Quantization + Entropy Encoding	Fourier Transform	Dequant. + Entropy Decoding	Ray Casting
[48]	MRI	Filtred and threshold/RLE+Arithmetic Coding	IWT	RLE+Arithmetic Decoding.	2D Texturing
[68]	Time Varying Models	DCT	Quant. DCT Coefficients	RLE+DCT Decod.	2D Texturing
[57]	MRI,CT	Sorting, normalization/RLE+Huffman Coding	Wavelets	RLE+Huffman Decoding	2D Texturing
[47]	CT	BLE Based on Tensor Quantization	Wavelets	Tensor Recost. and Decoding	Ray Casting
[56]	CT	VQ Encoding	Code Books	VQ Decoding	3D Texturing
[69]	Hydrodynamical Simulation	Brick Decomp./Variable Lenght Encoding	LZO Compression of Blocks	Variable Lenght Decod.	Ray Casting
[70]	CT	Block Subdivision/VQ Encoding	Code Books	VQ Decoding	3D Texturing
[60]	CT	-	PVTC-LZO	PVTC-LZO Decoding	Ray Casting
[71]	CT	Frame Ecoding	-	-	Ray Casting
[54]	-	-	DCT	-	2D-Textures
[32]	CT	-	Laplacian Pyramid	-	Ray Casting
[59]	CT	Rectilinear to Compact Multiresolution Volume	Octree	Decod. of Compressed Blocks	Ray Casting

2.4 Visualization on Low-End Devices

The display resolution, the lack of power capacity, as well as memory and storage capability constraints, make mobile devices to be qualified as low-end devices in client-server architecture implementations [72]. Low performance computers are also in this group of devices when comparing to sophisticated computers designed to run complex graphics applications.

The advance of the mobile telephony and mobile devices in general, encourage the developing and updating of visualization algorithms, most of them for video games. Besides libraries updating, new software and platforms for embedded systems have also appeared. These properties as well as the easy maintenance and portability make mobile devices the preferred alternative for scientists and developers. Because of this, their limitations are becoming a strong field of discussion for computer graphics researchers [72].

In parallel, volume models have grown continuously during the last years. The amount of memory of modern GPUs is also growing, but unfortunately the increasing rate of the size of volumetric data sets is much higher surpass it.

Most of the recent approaches for volume visualization in mobile devices have been implemented using OpenGL ES [17, 18, 19], which is an application programming interface for advanced 3D graphic target at handheld and embedded devices. This library addresses some device constraints like preprocessing capability and memory availability, low memory bandwidth, sensitivity to power consumption , and the lack of floating-point hardware. However the scarce amount of proposals and effective solutions in the area, as well as the user interaction requirements for this kind of applications make volume visualization algorithms in mobile devices a new challenging problem for research.

2.5 Conclusions and Discussion

The study of client-server architectures for volume visualization is nowadays a wide area of research for computer graphics scientists . We have made a comparative analysis of some relevant approaches in the area. But of course it is impossible to fully cover all published in this work.

Although there have been interesting practical solutions in the last years, a lot of open problems remain for further investigation.

Sending the whole volume model in a compressed way to the client device is still a challenging problem. This ensures full interaction facilities in the client without further server-client communication. Many proposals achieve good results, but the small allowed size of models and network latency during real time visualization, decrease interaction capabilities.

As an alternative most techniques transmit images through the network, but this falls in the lack of volume information in the client side, decreasing performance when new data is demanded and affecting image quality when reconstruction takes place in clients.

Compressed volume rendering method is a good solution to render those models whose size exceeds current hardware capabilities. But performance during decompression is being still a field needed of improvements. Although some proposals decompress data using CPU, the current trend is to move all the decompression to the last part of the graphic pipeline. This way the data gets compressed to the GPU, which result in less memory consumption and better exploiting available bandwidth.

Data transformation, quantization, encoding and progressive transmission, are concepts strongly united. Some approaches convert volume data into compact data structures ready to be gradually sent from servers to clients. In cases where no client-server architectures are implemented, compact data structures are used to transfer data between CPU and GPU or also to efficiently perform out-of-core methods.

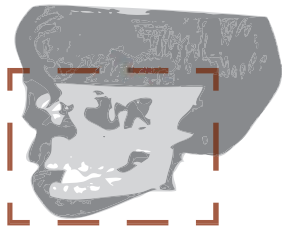
Medical visualization requires a special treatment. Current solutions propose novel improvements without fully exploiting the fact that doctors usually

adopt standard transfer functions. This fact can be a new tool to compress even more the datasets. Moreover, low-end devices as mobiles are being preferred due to it easy maintenance and portability but their limitations demand a further revision of client-server algorithms to optimally increase interactivity while inspecting large volume models.

CHAPTER 3

WAVELET-BASED VOLUME COMPRESSION FOR REMOTE VISUALIZATION

This chapter presents a Wavelet-based transfer-function dependent approach for interactive volume visualization in small mobile devices



Contents

3.1 Introduction

The complexity of present volume models in medical applications is continuously increasing, therefore increasing the gap between the available models and the rendering capabilities in low-end mobile clients. New and efficient rendering algorithms and interaction paradigms are required for these small platforms.

In this Chapter, we propose a transfer-function aware compression and interaction scheme for client-server architectures, with visualization on standard mobile devices. The scheme is block-based, supporting an adaptive ray casting in the client. A two-level ray-casting allows focusing on small details on targeted Regions of Interest (ROIs) while keeping bounded memory requirements in the GPU of the client. The approach includes a Transfer Function-aware compression scheme based on a local Wavelet transformation, together with a bricking scheme that supports interactive inspection and levels of detail in the mobile device client. We also use a quantization technique that takes into account a perceptive metrics of the visual error.

Specifically, the approach includes a compression scheme based on a previously defined Transfer Function and on a local Haar Wavelet transformation that works on individual $16 \times 16 \times 16$ blocks of the volume. Unlike previous scheme, the compression technique allows on-the-fly decompression in the client and saves memory requirements in the model transmission.

Results show that we can have full interaction with high compression rates and with transmitted model sizes that can be of the order of a single photographic image.

3.2 Preliminaries

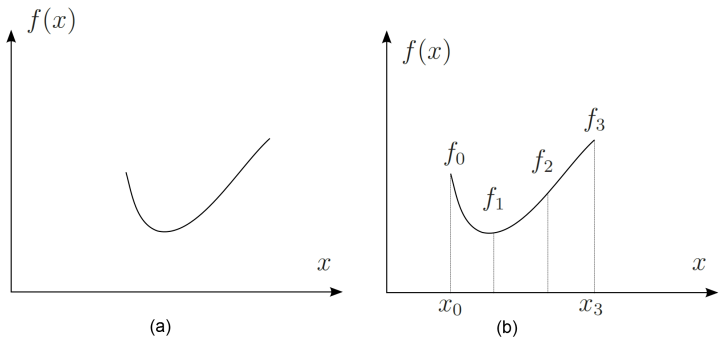
3.2.1 Volume Transformations

Real functions in Computer Graphics and Geometric Modeling are usually defined on a 3D domain space or in domains of lower dimension subspaces. They map values from the domain D into the real line $\Re$, $f : D \in \Re^n \rightarrow \Re$

Analytical functions defined in one dimensional domain can be written in the way $y = f(x)$ where $y \in \Re$ and $x \in D \subset \Re$. In general, the necessary information to represent a general function can be unbounded (infinite), as Hilbert spaces which have an infinite dimension. In order to represent and use them in a computer, an approximation is required. Functions can be approximated by their coefficients in a finite set of basis functions, or, simply, by sampling them at a finite (most times uniformly spaced) number of values $x_0...x_n$, see an illustrative example in Figure 3.1. The function $f(x)$-(Figure 3.1-(a)), can be represented by the coefficients $[c_0...c_N]$, or it can be defined by the values $[f_0...f_N]$. In both cases, $f(x)$ is approximated by a one-dimensional array of coefficients or values. In Figure 3.1-(b) the function $y = f(x)$ is represented by the discrete set of values $f_0...f_N$ or $f(x_0)...f(x_N)$ or $\bar{f} = [f_0...f_N]^T$, where $x_k = x_0 + k\dot{h}$ means that it is approximated by a finite number of samples.

The Nyquist-Shannon Sampling theorem [73] tells that a function $f(x)$ will be completely determined by a discrete sampling with an spacing h (like $f(x_0)...f(x_N)$, where $x_k = x_0 + k\dot{h}$), only if it contains no details of frequency greater or equal than $1/(2h)$ and if $N \rightarrow \infty$. In other words, we can never recover details with period smaller than 2h. Any sampling looses information.

Functions approximated by arrays can be transformed by standard linear transformations represented as $(N+1) \times (N+1)$ matrices $[T]$, $\bar{f}_T = [T] \cdot \bar{f}$. Non-singular linear transformations represent invertible function transformations. Examples include the Discrete Fourier Transform (DFT) [74], having



$$f(x) = c_0 B_0(x) + c_1 B_1(x) + c_2 B_2(x) + c_3 B_3(x)$$

(c)

Figure 3.1: The function $f(x)$ in (a), defined by the values $[f_0 ... f_N]$ in (b) and represented by $[c_0 ... c_N]$ in (c).

an efficient FFT computation algorithm, and kinds of different Wavelet transforms (Haar, Daubechies, and others, see [75]).

A wide range of mathematical transformations, are important tools for reducing data. One of the most used techniques to compress data is Vector Quantization [44], which is a lossy compression scheme. The basic idea is to code values from a multidimensional vector space into values from a discrete subspace of lower dimension. Because the lower-space vector requires less storage space, data gets compressed. The transformation into the subspace is either achieved through projection or by using a codebook.

The Discrete Cosine Transform (DCT) [76] maps sequences of scalars into single scalar indices. This is a compression method that transforms data into a set of coefficients that are then quantized to create a more compact representation.

3.2.2 Wavelet Volume Transformations

Wavelets are transformations which work by analyzing data components with a variable scale resolution. These transformations satisfy certain mathematical requirements and are used to represent data and time signals

[75]. Wavelets started by moving from the previous notion of frequency to the notion of scale analysis, and by exploring orthogonal systems and the Fourier convergence. Therefore, Fourier Transform (FT) [77] can be considered as an ancestor of the Wavelet Transform. The ability of analyzing functions through their frequency content makes Fourier Transform a very important tool. Since the Fourier coefficients of the transformed function represent the contribution of sine and cosine functions at each frequency, the signal can be analyzed through its frequency content [78].

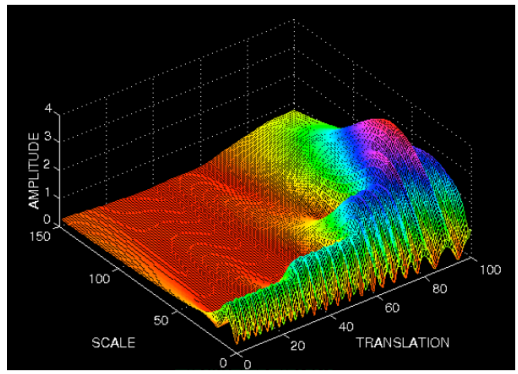


Figure 3.2: Wavelet Transform of a signal. From [78].

Figure 3.2 shows the continuous Wavelet Transform of a time signal in which the axes represent translation and scale, instead of time and frequency. In time signals, translation is strictly related to time, and it indicates where Basis Wavelet functions are located. In our proposal, we use transforms in the discrete domain to efficiently transform 3D volume signals while keeping a bounded complexity of the transformed data.

During a Wavelet transformation the signal goes through a number of high pass and low pass filters, which prune out either high frequency or low frequency portions. With a correct selection of the Wavelet function and truncating the coefficients below a threshold, data can be compactly represented, this possibility makes Wavelets to be an excellent tool for compression. The different Wavelet families make different trade-offs between how compactly the basis functions are localized in space and how smooth they are. Haar Wavelets [79], Daubechies [80], Biorthogonal [81], Coiflets [82], Symlets [83], Morlet [84] and Mexican hat [85] are some of the Wavelet schemes that have been used in different applications [75].

Transform, quantization and encoding are the three basic components of a typical Wavelet compression algorithm. During the transform stage, the input data is separated into different bands of frequencies using Wavelet filters. These filters are usually linear. They can be represented as constant matrices which transform a 1D data array X^n of dimension 2^n into a half-size data array of size 2^{n-1}. Given a data array coefficient vector X^n, classical Wavelets use a rectangular matrix A^{2^n} to compute

$$X^{n-1} = A^{2^n} X^n \tag{3.1}$$

and an accompanying matrix B^{2^n} to capture details

$$Y^{n-1} = B^{2^n} X^n \tag{3.2}$$

These two matrices (called analysis filters) have independent columns, and the process can be inverted using the synthesis filters A_R^n and B_R^n:

$$X^n = A_R^n X^{n-1} + B_R^n Y^{n-1} \tag{3.3}$$

without loss of information. Furthermore, the space required to store X^{n-1} and Y^{n-1} is the same as the required for X^n. This process can be iterated yielding a multiresolution scheme (with each level using half as much detail), which can be stored using the same space as X^n: It stores the approximation X^0 and a sequence of detail vectors $Y^1, Y^2, Y^3, ..., Y^{n-1}$.

The quantization step restricts the values of the coefficients. Then, the encoding step phase represents the results of the quantization as efficiently as possible with minimum loss.

A three dimensional Wavelet approximation for volume data sets was presented by Shigeru Muraki [38]. In this approach, a 3D Orthogonal Wavelet Basis is built by using a tensor product of one dimensional basis functions. This Wavelet transform can be applied to volume data [39]. The authors remove the insignificant coefficients and only use the remaining coefficients to reconstruct an approximation of the original data. E. Schiavi and C. Hernandez [86] presented an study that shows the advantages of using 3D-Wavelets Functions for medical image visualization. Our proposal uses this Wavelet Transform.

3.2.3 Truncated 3D Haar Transformations

The Haar Wavelet Transform has a very small local support. It is very efficient in applications that require fast decomposition and reconstruction, since it can be implemented by a few additions, subtractions and shift operations.

For a simple understanding of Haar Wavelet Functions, let us consider a sampled array:

$$X^n = X_i^n \mid 0 \leqslant i < 2^n \tag{3.4}$$

with a power of two size. A new array:

$$X_i^{n-1} = (X_{2i}^n + X_{2i+1}^n)/2 \mid 0 \leqslant i < 2^{n-1} \tag{3.5}$$

is obtained by averaging (down-sampling) each sample pair from X^n. This new array can be regarded as a coarser representation of X^n with half of the original size [40]. By applying a Haar Wavelet transform, part of the information gets lost during the down-sampling. Therefore, to recover the original array, detail information is computed in a second array as:

$$Y^{n-1} = Y_i^{n-1} \mid 0 \leqslant i < 2^{n-1} \tag{3.6}$$

where:

$$Y_i^{n-1} = (X_{2i}^n - X_{2i+1}^n)/2 \tag{3.7}$$

The 2^{n-1} averages and 2^{n-1} differences (or detail coefficients) are obtained by applying 2-channel subband filters: the smoothing filter and the detail (or Wavelet) filter respectively. The reconstruction of the original samples can be achieved by reversing the operations:

$$\left.\begin{aligned} X_{2i}^n &= X_i^{n-1} + Y_i^{n-1} \\ X_{2i+1}^n &= X_i^{n-1} - Y_i^{n-1} \end{aligned}\right\} 0 \leqslant i < 2^{n-1} \tag{3.8}$$

Decomposition can continue until we obtain the global average $X_0 = X_0^0$ and a sequence of detail arrays Y^0, Y^1, Y^{n-1}. During the reconstruction of the original data at any resolution, a number of additions and subtractions are applied to detail coefficients according to the desired level of compression.

In short, the first step of the Haar Wavelet transformation of a one-dimensional data array X with $N = 2^n$ components, computes a *filtered* and smoothed data array of dimension $N/2$ by simply adding (or averaging) each odd-element of X with its next, even element. The *footprint* of the k step of a specific discrete Wavelet transform is defined as the number of data elements of the initial 1D array X^n which are used in the computation of any single element of X^{n-k}. The *footprint* of the first step of the Haar Transformation is 2, while other Wavelet schemes like Daubechies and Coiflets have a *footprint* of $N = 2^n$. The second step of this Haar Wavelet transformation will average every two neighbour (odd and even) elements of the array computed in the first step to generate a smoothed data array of dimension $N/4$, with a *footprint* of 4. The complete Haar Wavelet transformation requires $log_2(N)$ steps and ends up with a single, scalar value. The *footprint* after k steps is 2^k, meaning that computations are local with respect to a 2^k-blocked partition of the original data. In other words, if the original array X of dimension N is partitioned into a set of non-overlapping 2^k intervals, the first k steps of the Haar Transformation make completely independent computations within each of these intervals. The complete Haar Wavelet transformation obviously involves the whole initial data and uses no blocked partitions, however. The same argument applies to 3D volume data, but now independent block supporting local computations after k iterations are non-overlapping volume bricks of size $2^k \times 2^k \times 2^k$.

Truncated 3D Haar Transformations are Haar transformations that stop after k transformation steps. They do not use inter-brick information, and stop after reducing the volume by a factor of 2^k in each spacial dimension. They are not explicitly encoding low frequency volume information such as the overall average density, but they are in fact local approximations of the volume, and this is a very attractive property for transmission, multi-resolution and interaction purposes. Unfortunately, other Wavelet basis like Daubechies, Coiflets or Spline-based Wavelets are global, their truncated versions being not local and not supporting spatial subdivisions. In what follows, we will restrict ourselves to truncated 3D Haar Wavelet Transformations.

3.2.3.1 Truncation Levels in the 3D Haar Transformation Wavelet

As discussed, the *footprint* of the 3D Haar Wavelet Transformation depends on the number of Wavelet steps. Several options have been analyzed in the framework of our volume compression application:

- Constraining the 3D Haar Wavelet Transformation to 2 steps results on a *footprint* with volume blocks of size $4 \times 4 \times 4$.
- Constraining the 3D Haar Wavelet Transformation to 3 steps results on a *footprint* with volume blocks of size $8 \times 8 \times 8$.
- Constraining the 3D Haar Wavelet Transformation to 4 steps results on a *footprint* with volume blocks of size $16 \times 16 \times 16$.
- 3D Haar Wavelet Footprints smaller than $8 \times 8 \times 8$ result on Wavelet matrices difficult to quantize (so compress), due to the relevance of the information that they encode. We have therefore discarded these options.
- 3D Haar Wavelet Footprints larger than $16 \times 16 \times 16$ result on a slightly better Wavelet approximation, but the spatial subdivision is too coarse for practical interaction purposes. When Regions of Interest (ROI) with high quality volume visualizations are to be supported they must include a certain number of full blocks of the volume, to ensure that every volume block can be reconstructed at a different and suitable level of detail (LOD). This is due to the fact that any block acts as a reconstruction-unit, all block voxels having to be reconstructed at the same and identical LOD.

We use a block size of 16 together with a 4-steps Haar Transform, being rather efficient in compression and still offering acceptable interaction facilities (see Section 3.8). We preferred a 4-steps scheme instead of a 3-steps Haar Wavelet Transform in order to maximize the compression rate and the final visual quality in the client. Using a 4-steps Haar Transform means that any volume block will have four possible LODs during reconstruction. The four-level reconstruction of a block generates a full piece of $16 \times 16 \times 16$ voxels that represent the corresponding part of the volume. Reconstructions of the same block at three, two or one levels generate pieces of $8 \times 8 \times 8$,

$4 \times 4 \times 4$ or $2 \times 2 \times 2$ voxels, representing the same part of the volume at lower resolutions.

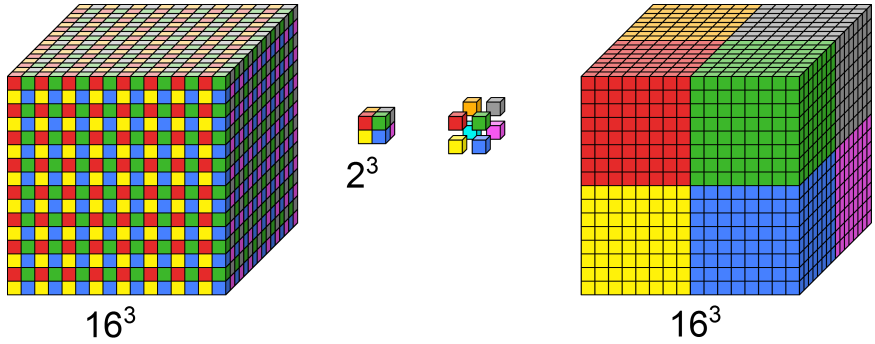


Figure 3.3: A $16 \times 16 \times 16$ block of the original volume (left). One of the $2 \times 2 \times 2$ atomic subregions for Wavelet computation (center). The resulting low frequency matrix (in red) together with the seven error (or high-frequency) matrices (right).

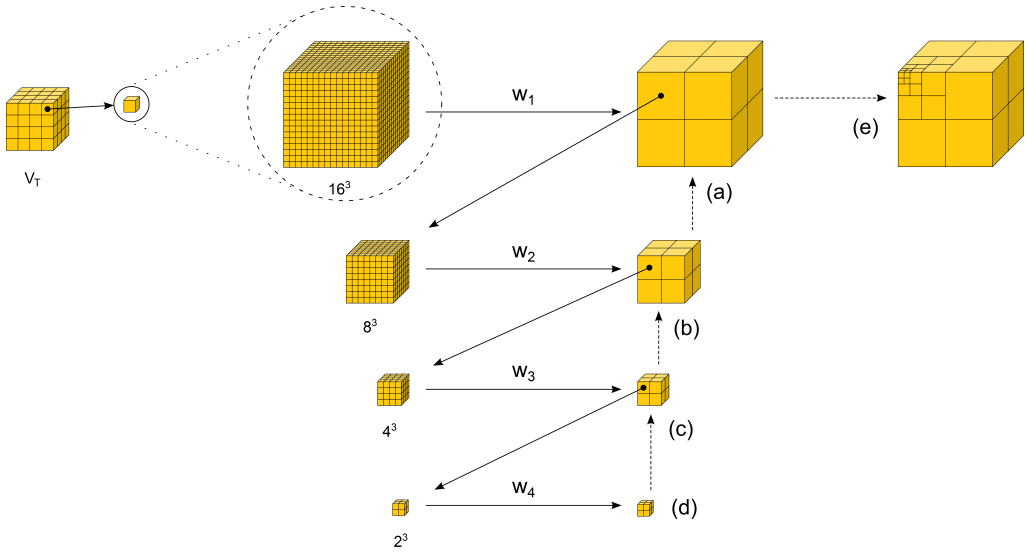


Figure 3.4: Generation of the four levels of the 3D Haar Wavelet Transformation in a 16^3 block. In (a), the result of applying one Wavelet step as explained in Figure 3.3. Same process is done for the second Wavelt step (b), third Wavelt step (c), and the fourth Wavelet step (d). The result in (e) shows the four-step Haar Transform applied to the initial 16^3 block.

To compute the Wavelet transformation, we have implemented an approach similar to [40]. After the subdivision of the volume in blocks of $16 \times 16 \times 16$ voxels, we then subdivide each block in sub-regions of $2 \times 2 \times 2$ voxels as

shown in Figure 3.3. Then, we compute the 8 possible linear combinations (with coefficients $+1$ or -1) among the 8 values inside a $2 \times 2 \times 2$ voxel subregion to compute the Haar Wavelet Transformation. After that, each one of the computed values is distributed inside the $16 \times 16 \times 16$ voxels block to create one low frequency matrix and seven high frequency matrices. This process is then repeated for the second, third and fourth steps of the Wavelet transform, see Figure 3.4. The whole computation is performed in the integer domain, to avoid rounding errors (data generated in each of the four Wavelet steps gets multiplied by 8, 64, 512 and 4096 respectively).

In what follows, we will refer to the high frequency or error matrices according to their location with respect to the low frequency matrix in each level of transformation. Therefore we call E_F the group of matrices which share one face with the low frequency matrix, E_E the group of matrices sharing one edge and E_V the matrix sharing only one vertex with the low frequency matrix. In Figure 3.3, E_F matrices are painted yellow and green, the E_V matrix is painted pink, and the rest are E_E matrices. After four Wavelet steps (see Figure 3.4) we get a hierarchy with 12 groups of error matrices: E_{F_1}, E_{E_1}, E_{V_1}, E_{F_2}, E_{E_2}, E_{V_2}, E_{F_3}, E_{E_3}, E_{V_3}, E_{F_4}, E_{E_4}, E_{V_4} (the index indicates the Wavelet step).

3.2.3.2 Statistical Analysis of 3D Haar Error Matrices

This Section presents a statistical study of the Wavelet coefficients behavior, which will drive the design of the quantization scheme (see Section 3.5). Figure 3.5 shows the models we used for analyzing data. The Skull model (left) with a $256 \times 256 \times 112$ resolution, a foot model of $256 \times 256 \times 256$ resolution and a jaw model of $512 \times 512 \times 48$ resolution. The amount of 16^3 blocks in these models is $(16 \times 16 \times 7)$, $(16 \times 16 \times 16)$ and $(32 \times 32 \times 3)$ respectively. Density values are between [0...255], each voxel being codified using 1 byte.

We made an study of the statistical behavior of the error matrices for each one of the four 3D-Haar Wavelet Transformation steps. Our conclusion is that there exists a statistical similarity among matrices in the same group (E_F, E_E or E_V). In fact, this is a direct result from the properties of 3D Wavelets

Figure 3.5: The three volume datasets used in our experiments.

and the isotropy of the medical volume models: E_F matrices result from applying 1D Wavelet transform to the volume in each of the three spatial directions (x, y, z), while E_E matrices result from applying two orthogonal 1D Wavelet transforms (x-y, x-z, y-z) to the volume. The resulting evidence is that the statistical behavior of the values of the elements of matrices in the same group is similar. Figure 3.6 shows, for instance, the histogram of the values of 20 random E_F matrices. We observe a Gaussian behavior in their element values. Matrices E_E and E_V have the same Gaussian behavior.

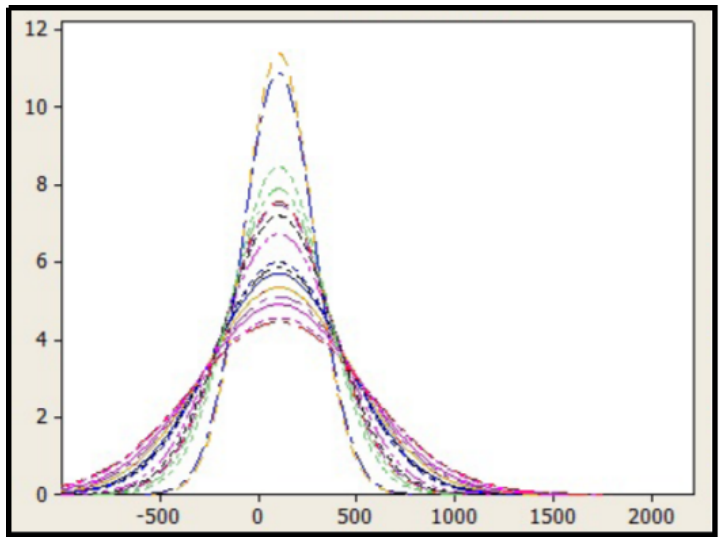


Figure 3.6: Histogram of the values in 20 E_F error matrices, randomly chosen. A clear Gaussian behaviour can be observed.

3.3 Overiew

In this Section we briefly introduce the general framework of our client-server approach, which is shown in Figure 3.7. In the next paragraphs we first define the concepts of transfer function, blocks and region of interest.

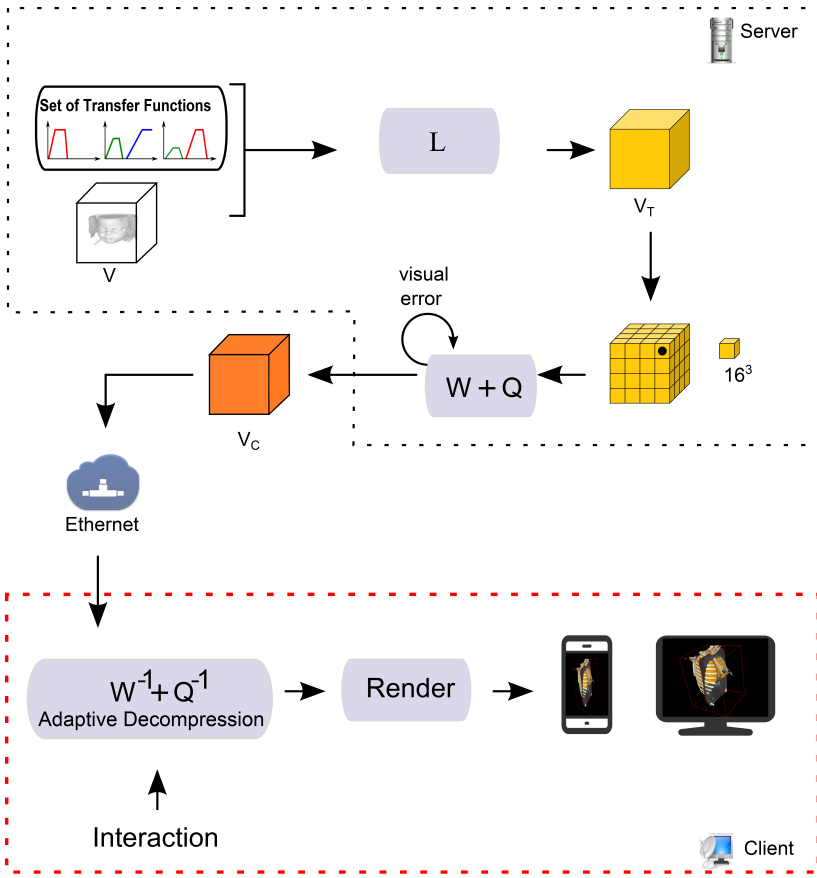


Figure 3.7: Overview of the transformations in the server side. The transformed volume V_T is obtained from the initial volume V by using a constrained Laplacian transformation L. Then, V_T is partitioned into 16^3 blocks, and the quantization Q of the Wavelet transformation W is tuned according to a precomputed estimation of the visual errors. The compressed volume V_C is sent over the network. In the client side, the user interaction drives the model reconstruction followed by an adaptive Ray Casting-based rendering algorithm.

Let V be a volume model with voxels representing scalar densities. Let TF be a rendering transfer function defined as a function of one variable: TF assigns opacity and color values for any value of the volume density. The region of interest (ROI) is usually defined as the user-defined region where the maximum resolution and detail is required during interactive inspection. Our approach supports regions of interest consisting on a connected set of blocks. The user will usually accept a lower resolution outside the ROI while requiring the maximum resolution in the areas inside the ROI. Choosing a suitable block size is a compromise. Large n values support more wavelet steps but result on too rough ROI shapes. On the other hand, small n values produce nice and flexible ROI shapes at the cost of too few wavelet steps and poor multiresolution behaviour in the client, as discussed in item 2 below. We conclude that the two most performing options are $\mathsf{n} = 3$ and $\mathsf{n} = 4$.

We divide the volume into equal-sized bricks, which we call blocks in what follows. We use the property that Haar wavelets are local in 2^{n}-sized blocks: applying n steps of the Haar transformation to V gives the same result as locally applying these n Haar steps to each of the individual blocks (this is not true for $\mathsf{n}+1$ steps, as the Haar transform would require volume data from neighbour blocks).

In our scheme (see Figure 3.7), the server builds a TF-dependent compressed model file by following these steps:

1. A TF-aware smoothing is applied to V. The objective of this preprocess is to remove local features in V that would unnecessary increase the size of the compressed file. This smoothing algorithm is presented in Section 3.4. By using prior knowledge on the Haar Wavelet transform, we succeed in decreasing the size of the compressed file while maintaining a reasonable visual quality.

2. The next step is the subdivision of the obtained volume data set V_T in blocks of $16 \times 16 \times 16$. This granularity with $\mathsf{n} = 4$ supports four independent Wavelet transformation steps, which is a good compromise between interaction performance and compression rate, as discussed in Section 3.5. Each block is independently transformed, allowing independent decompression, when required, in the mobile device. The option of using $8 \times 8 \times 8$ blocks, with $\mathsf{n} = 3$, would result in a

more flexible definition of the regions of interest. However, it would only support three wavelet steps, which we consider too limited for multiresolution purposes.

3. *Nil* blocks are detected and tagged. *Nil* blocks are irrelevant for the specific TF being used, and can easily be detected by simulating the final rendering in the server. A particular block B is *Nil* **if** $render(V_T)$ is equal to $render(V_T - \mathsf{B})$ (we use the notation $V_T - \mathsf{B}$ to represent volume V_T without block B). *Nil* blocks can contain parts of the volume with zero opacity, or they can be fully occluded by other visible, opaque blocks (anatomical structures inside a cavity, which are invisible from any viewpoint). *Nil* blocks are not wavelet-transformed: their identifiers are the only information that will be sent from the server to the client.

4. After that, four Haar Wavelet transformation (W) steps are applied to every non-*Nil* block, and Wavelet coefficients are quantized (see Section 3.5). Quantization is optimized based on a specific visual error metrics with threshold values obtained from a user study. As a result, the compressed volume file V_C is obtained.

The compressed volumetric information is sent to the client side, where an adaptive decompression is efficiently performed in its CPU according to the camera and region of interest (ROI), see Section 3.7. By only decompressing the required blocks, we obtain significant memory savings in the client, being able to show areas of interest at the maximum resolution. Finally, the client performs a ray-casting based rendering which accesses an implicit multiresolution volume model represented by bricks of different resolutions.

3.4 Transfer-Function Dependent Volume Transformation

In this Section, we explain the Smoothing Laplacian Preprocess L which is performed previously to the Wavelet transform (see Figure 3.7).

We start from the volume data V and a certain transfer function TF (see Figure 3.7). Then, the server transforms V by computing a TF-aware

smoothing. By using the well-known property that linear scalar fields result in constant error coefficients in their Haar Wavelet transform, we succeed in decreasing the size of the compressed file while maintaining the final visual quality.

We assume standard piecewise linear transfer functions [49] (see Figure 3.7). By using a piecewise linear opacity function, we virtually segment the volume V in as many regions as linear segments defined by TF. Voxels with a density d such that $TF.Opacity(d) = 0$ belong to *Nil* regions. The rest of the voxels belong either to ramp regions (when d belongs to one of the slanted linear segments of $TF.Opacity$) or to constant regions (when d belongs to one of the constant value intervals in $TF.Opacity$). The total number of constant and ramp regions equals the number of segments in the TF opacity function.

We implement a TF-aware smoothing algorithm as a constrained Laplacian filter in each of the regions: we perform a Laplacian smoothing for each volume voxel ($V_{i,j,k}$) in each region and then force it to remain in the interval of density values defining this region. In constant regions, this smoothing has obviously no visual effect; however, it spatially smoothes the field, tending to produce spatially linear density fields. In ramp regions, Laplacian smoothing is also generating linear density fields, as desired. However this smoothing leads to small visual artifacts as we are locally smoothing the opacity. To minimize these effects, we use a more conservative Laplacian filter for ramp regions:

$$L_{i,j,k} = \lambda_T V_{i,j,k} + \frac{(1-\lambda_T)}{6}(V_{i+1,j,k} + V_{i-1,j,k} + V_{i,j+1,k} + V_{i,j-1,k} + V_{i,j,k+1} + V_{i,j,k-1}) \tag{3.9}$$

where $\lambda_T = 0$ in constant regions and $\lambda_T = 0.5$ in ramp regions. In order to guarantee the filter isotropy, at each Laplacian filter iteration we compute a second array L, then we clamp the density values in L to the interval values of their initial regions, and finally assign L to the smoothing volume V_T which will be Wavelet transformed. In our implementation, the total number of iterations is set to some value between 400 and 800 to maintain a good compromise between compression rate and quality, see Section 3.8.

3.5 Block-Based Volume Compression and Quantization

After smoothing, we subdivide the volume V_T in blocks, and use them to independently compute a Haar Wavelet transform. We then quantize the obtained coefficients and sent them (V_C) to the client. The advantage of having a block-based computation with independent block transformation and quantization is two-fold: we can use stronger, localized quantization policies in the server, and the client can implement a block-based rendering acceleration and use block-aware interaction paradigms. As already mentioned, our implementation uses a block size of 16 together with a 4-steps Haar transform, being rather efficient in compression and still offering acceptable interaction facilities.

Once transformed with the above TF-aware Laplacian smoothing algorithm, the volume V_T is subdivided into non-overlapping $16 \times 16 \times 16$ blocks (or bricks) which are transformed through a four-step classical Haar Wavelet transformation. The algorithm is straightforward, as observed in Section 3.2.3.1, only requiring additions and subtractions. After four Wavelet steps we obtain a hierarchy with 12 groups of error matrices for every $16 \times 16 \times 16$ block of V_T: E_{F_1}, E_{E_1}, E_{V_1}, E_{F_2}, E_{E_2}, E_{V_2}, E_{F_3}, E_{E_3}, E_{V_3}, E_{F_4}, E_{E_4}, E_{V_4}, where the index indicates the Wavelet step (see Figures 3.3 and 3.4).

The design of our quantization algorithm derives from the above-mentioned properties of the error matrices (see Section 3.2.3.2). Our intention is to increase the compression rate while maintaining a correct user perception after the reconstruction of the model in the client. The process starts by spreading in a uniform way the values in the matrices of the first two steps, E_{F_1}, E_{E_1}, E_{V_1}, E_{F_2}, E_{E_2}, E_{V_2}. We consider the Gaussian behavior of these error matrices and compute, the mean and standard deviation values for all of them. Then, we transform their error matrix values by applying a piecewise linear uniforming function U that approximates the accumulated Gaussian probability function. The function U is shown in Figure 3.8, depending on three parameters: λ_U, A and B.

To compute U we need the mean μ and the standard deviation σ of the

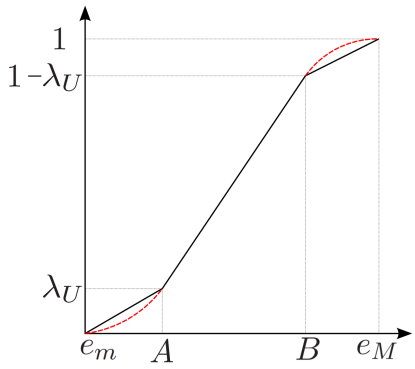


Figure 3.8: Piecewise linear uniforming function U. U is designed in a way such that the statistical distribution of the error matrix values $U(e)$ is approximately uniform.

error matrix values e. We do this as follows:

$$\mu = \left[\frac{\sum_{k=1}^{N} e}{N} \right] \quad | \quad N = 3 \cdot 2^n \tag{3.10}$$

$$\sigma = \left[\frac{\sum_{k=1}^{N} (e - \mu)^2}{N} \right] \quad | \quad N = 3 \cdot 2^n \tag{3.11}$$

Where N is the number of components of each processed error matrix. Error matrices are parametrized by their μ and σ values, plus the maximum and minimum values of their elements $e_M = Max\{e\}$, $e_m = Min\{e\}$. To approximate a cumulative Gaussian distribution, the parameters defining U are computed as:

$$\lambda_U = 0.0567 \cdot (e_M - e_m) \tag{3.12}$$

$$A : [MAX(e_m, \mu - 1.5 \cdot \sigma, e_m + \lambda_U)] \tag{3.13}$$

$$B : [MIN(e_M, \mu + 1.5 \cdot \sigma, e_M - \lambda_U)] \tag{3.14}$$

Transforming error values by U, obviously results in a much more uniform distribution of these values along their range. We name U a uniforming

function because U tends to separate dense elements around the mean value, while grouping those that are far from the mean and producing a more uniform distribution. The result is an increment in the overall compression rate. We have however observed that this uniforming function U is not required for the error matrices of the third and fourth Wavelet steps, because of the lower significance of their error values.

After applying U, the final compression step consists of a *Uniform Scalar Quantization* Q [42] (see Figure 3.7). The main parameter of our quantization scheme is the vector of intervals I_i, that drives this Q step. I_i is 12-dimensional, each element being the number of quantization intervals for the corresponding group of error matrices. Before quantization, the server tunes the elements of I_i to guarantee that we will get a maximum compression rate with a minimum loss of visual quality, as will be discussed in Section 3.6. Having the above described parameters, the final quantization formulas are:

$$e' = U(e) \tag{3.15}$$

$$q = \left[\left(\frac{e' - e_m}{e_M - e_m} \right) \cdot I_i \right] \;\mid\; q \in [0, I_i - 1] \tag{3.16}$$

Quantized values are finally Run-Length encoded (RLE) and sent to the client together with the transfer function TF.

In the client, blocks are processed individually, under request. The client keeps the list of compressed blocks. Dequantization of specific blocks takes place in the CPU of the client after RLE decoding and before Wavelet reconstruction. The client decompression of any block works by computing the inverse of the above three functions: $W^{-1}(U^{-1}(Q^{-1}(CompressedBlock)))$. Dequantization and de-uniforming are simple and efficient operations involving linear scaling and piecewise linear transformations of the quantized values q:

$$e' = \left[\left(\frac{q + \frac{1}{2}}{I_i} \right) \cdot (e_M - e_m) + e_m \right] \tag{3.17}$$

$$e = U^{-1} \cdot e' \tag{3.18}$$

Wavelet decompression is performed by simply adding and subtracting elements of $2 \times 2 \times 2$ submatrices, [63].

3.6 Error Metrics for Perceptual Quantization

We use a perceptual metrics to automate the compression technique. To set up a reasonable set of perceptual parameters for the vector I_i of intervals, we devised an experiment to compute the perceptual accuracy for each of the 12 error matrices in the *Interval Encoding Quantization*.

The test involved 16 users that were presented a total of 720 images (3 cameras, 2 zooms, 12 quantized error matrices, with 10 images in each case). The Skull volume model was selected for the experiments. Figure 3.9 presents the interface for producing these test images. The interface shows the quantization values in the left column, a reference view of the initial model and a view of the quantized and reconstructed model (bottom right of the interface). The two interface examples shown in Figure 3.9 respectively present a case with a fine quantization (with the quantization intervals set to their maximum possible values) and a second case with too small values for the quantization intervals and poor visual quality. Users were presented a number of layouts with 10 images each, see images in Figure 3.10. Each layout corresponded to one of the 12 sets of matrices and to one of the 6 camera-zoom settings. Images in a particular layout were sorted in a descendent visual quality order, based on the values in the quantization intervals vector I_i. This decreasing visual quality can be observed in the image sequences in the layout shown in Figure 3.10. The only parameter changing along the images in a certain layout was the value of the corresponding element of the vector I_i; the other elements were set to maximum values to avoid disturbances. Users had to select the first one of these 10 images which was perceptually different from the original model image in each case. For the selected image, we computed the Root Mean Square Error RMS [87], the Laplacian error (computed as the RMS between the Laplacians of the two images), a weighted average between these two errors (WErr), and the HDR-VDP-2 [88] error metrics between the original image and the selected image.

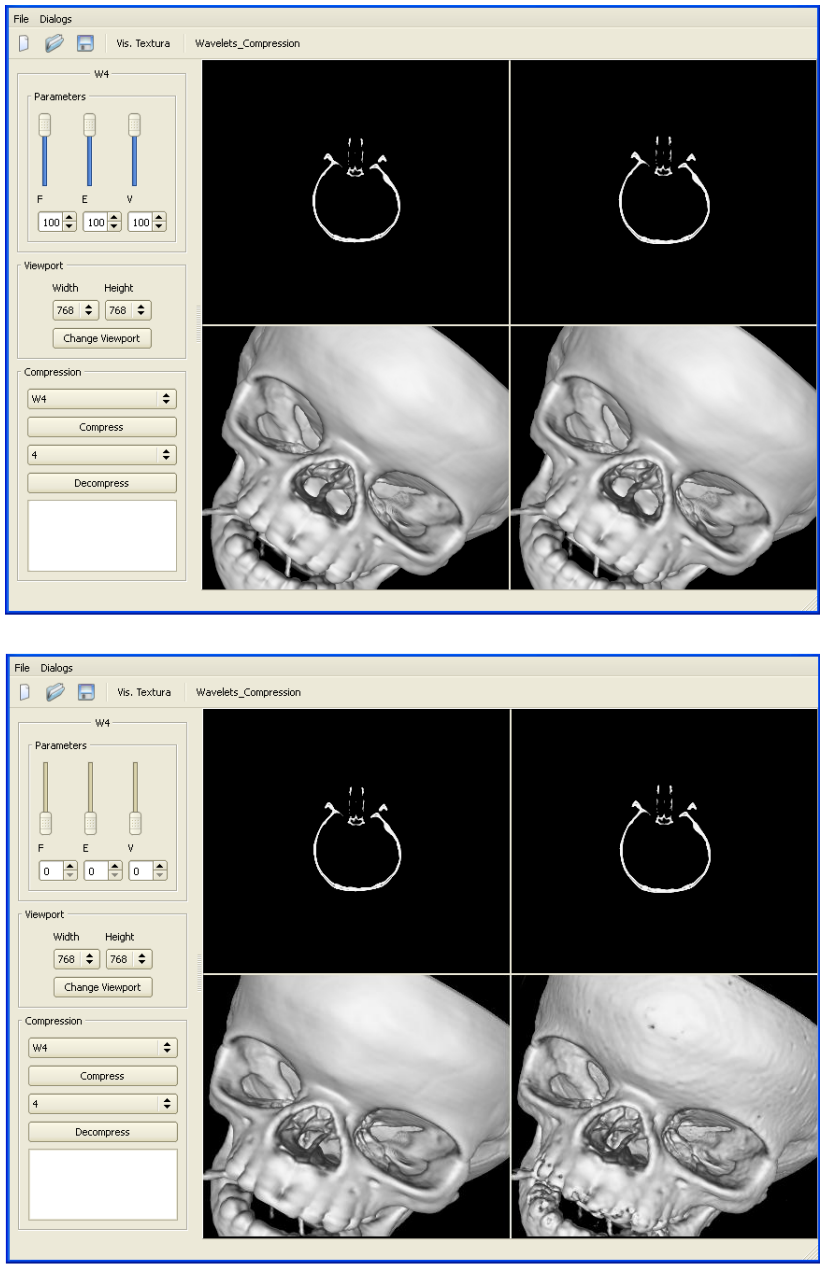


Figure 3.9: Interface of the perceptual test.

The results of our experimental test are shown in Figures 3.11 and 3.12. The number of quantization intervals (elements of the vector I_i) which were

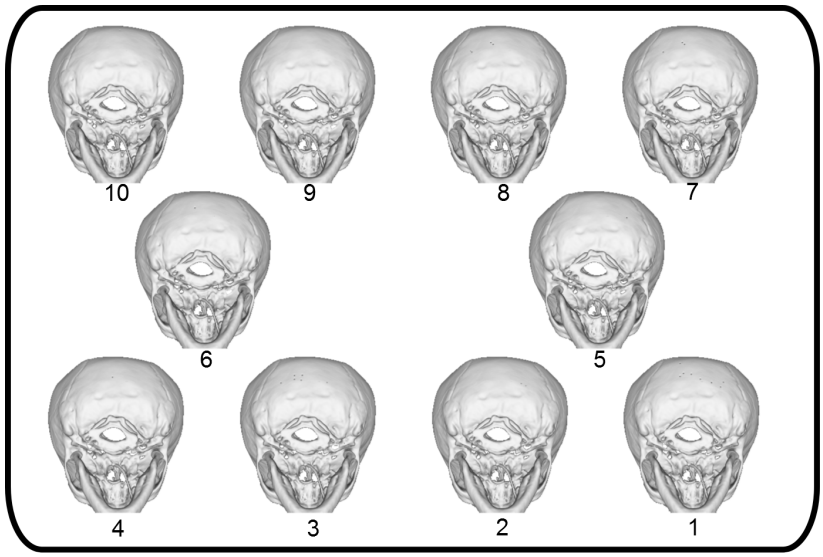


Figure 3.10: One of the layouts shown to the users during the perceptual experiment. Images are sorted in descendent visual quality order.

required to reach a perceptually correct visual quality was coherent with the expected results. We observed that a finer quantization is required in the error matrices of the first Wavelet step, with interval values around 350 for F_1, 160 for E_1 and 70 for V_1. The number of quantization intervals is uniformly decreasing within each Wavelet step (values are always larger for F matrices, smaller for E matrices and even smaller for V error matrices). They are also uniformly decreasing when we move from the first Wavelet step to the second and afterwards to the third and fourth. Moreover, we observed that uncertainty follows the same rule: it decreases along the components of the vector I_i, with more and more similar quartile values. In short, the optimal subjective quantization values and their quartile distribution behaves in a smooth and predictable way. Observe that this quartile distribution is identical in all four images in Figures 3.11 and 3.12.

We also computed the image-quality measure for the median values of the quantization intervals in each case of the diagrams in Figures 3.11 and 3.12. These Figures show the values of the Root Mean Square error, the Laplacian error, an average between RMS and Laplacian, and the HDR-VDP-2 error. The values should be constant, as they correspond to quantization scenarios

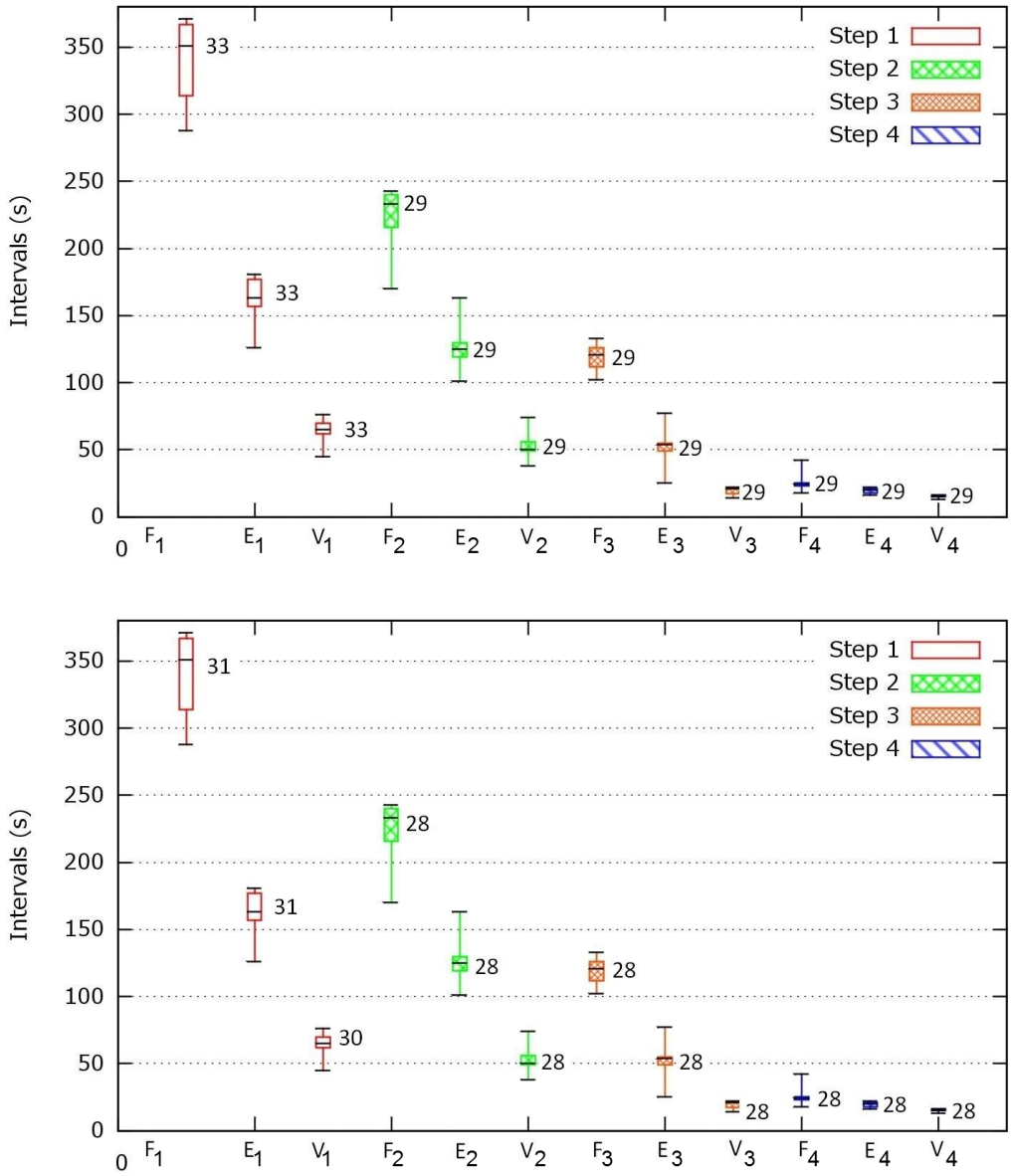


Figure 3.11: Results of the Perceptual Quality Test. Top: RMS error. Bottom: Laplacian error. The quartile values of each of the 12 error matrices are shown, together with the image-quality error measure between the original image and the image quantized with the median value of the quantization intervals in each case.

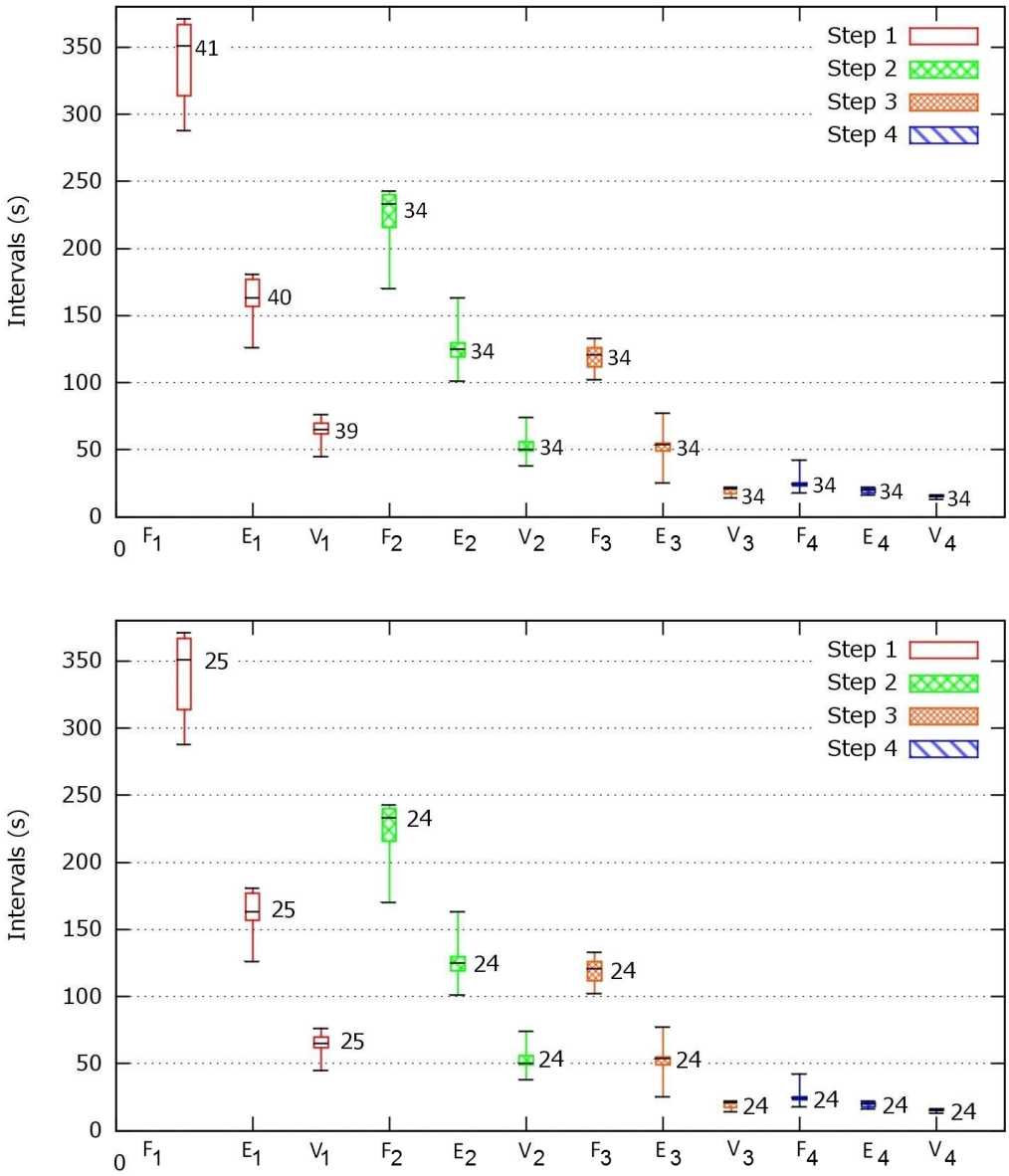


Figure 3.12: Results of the Perceptual Quality Test. Top: Weighted error (WErr). Bottom: HDR-VDP-2 error. The quartile values of each of the 12 error matrices are shown, together with the image-quality error measure between the original image and the image quantized with the median value of the quantization intervals in each case.

that were perceived as equally acceptable by our test users. We therefore concluded that the best error metrics for our purposes is HDR-VDP-2.

As a conclusion, we have an automatic way to adjust the quantization parameters in the server, before quantizing and sending the compressed volume to the client. The server sets up an initial set of quantization parameters, quantizes the model and next, it simulates the dequantization and decompression of the client. The server can finally render several views of the model, computing the value of the HDR-VDP-2 error metrics. Quantization parameters can be automatically adjusted until reasonable visual errors are obtained, according to the values presented in the bottom image in Figure 3.12. This automatic iterative refinement of the quantization parameters will produce acceptable perceived visual qualities in the final visualization in mobile clients. However, to speed up the encoding process in the server, in our implementation we simply use the median of the quantization intervals in Figure 3.12-bottom. Although a logic option would be the selection of values adapted to each model and TF, we use always median values from Figure 3.12-bottom as described before.

3.7 Volume Rendering with Regions of Interest

Once the volume model is sent to the client, it is stored as a list of compressed blocks. *Nil* blocks are stored as a simple *Nil* flag, whereas non-*Nil* blocks are represented by their run-length encoded, quantized expression. Block reconstruction takes place in the client CPU, on demand, depending on the local interaction requirements. The reconstruction of any of the blocks within the entire volume can be performed at one, two, three or four Wavelet levels. The four-level reconstruction of a block generates a full piece of $16 \times 16 \times 16$ voxels that represent the corresponding part of the volume. Reconstructions of the same block at three, two or one levels generate pieces of $8 \times 8 \times 8$, $4 \times 4 \times 4$ or $2 \times 2 \times 2$ voxels, representing the same part of the volume at lower resolutions. This local treatment allows multiresolution and bricking, by applying region-based reconstruction as described below. We achieve strong memory savings in the CPU of the client by only storing the 3D textures of the reconstructed blocks in a dynamic way, see Section 3.8.

The interaction interface allows the user to perform basic camera interactions like rotation and zooming. During zooming in a region of interest in a model, the highest resolution can be retrieved and displayed. The algorithm also allows both resolutions displayed at once in areas selected by the user.

A usual interactive session starts by inspecting the whole volume model at a low resolution. All blocks are reconstructed at one or two Wavelet levels (see Figure 3.23, column 1 and 2), the corresponding 3D Texture is computed in the CPU of the client, sent to its GPU and ray-casting rendered. Observe that the size of this 3D texture, in the case of two reconstruction levels, is 1/64 of the size of the original volume model V. In the case of only one reconstruction level, its size is 1/512 of the size of the original volume model.

The user can then select a Region of Interest, ROI, and zoom-in to this region. In this case, the blocks of the ROI are reconstructed at four levels, the corresponding 3D Texture is computed in the CPU of the client, sent to its GPU, and ray-casting rendered. In this case, the size of the 3D texture is also small but depends on the extend of the selected ROI.

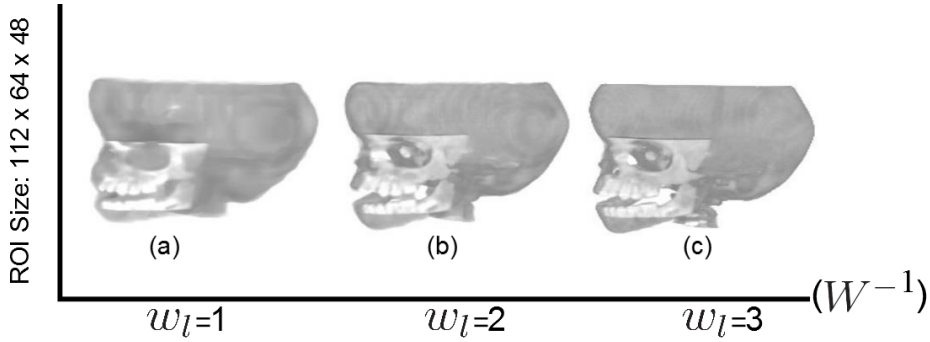


Figure 3.13: The Skull model with the ROI at the maximum level of detail. The low resolution region outside the ROI rendered at 1 (a), 2 (b) or 3 (c) Wavelet reconstruction steps.

Alternatively, the user can decide to inspect the whole volume model at a low resolution (two levels of reconstruction, for instance) with the ROI rendered at the maximum level of detail (Figure 3.13-(b)). To achieve this last interactive visualization, two 3D textures have to be sent to the client GPU where an adaptive ray-casting algorithm is performed, as detailed

below (see Algorithm 1). Since the whole model is available at the client side, rotation and zooming operations can be autonomously performed in the client without any further transmission from the server. If a region of the model needs to be detailed, the higher resolution can be displayed on demand.

Summarizing, our implemented ray-casting algorithm allows:

1. Visualizing the entire model in one of the four possible resolutions, by creating a 3D texture from the decompressed data. That is, we apply w_l reconstruction steps, $1 \leq w_l \leq 4$, as explained in Section 3.2.3.1. If $(R_x \times R_y \times R_z)$ is the spatial resolution of the initial volume V, the reconstruction creates a 3D texture ($VolumeLow$ texture) of: $\frac{2^{w_l} R_x}{16} \times \frac{2^{w_l} R_y}{16} \times \frac{2^{w_l} R_z}{16}$.
2. Visualizing a selected region of interest at maximum resolution, by creating a 3D texture from the decompressed data. Selected regions of interest must contain a subset of the volume bricks, as shown in Figure 3.14. Blocks (corresponding to the initial $16 \times 16 \times 16$ blocks) are our atomic multi-resolution and rendering structures. In this case, we apply four reconstruction steps $w_l = 4$. Let $(i_{min}, i_{max}, j_{min}, j_{max}, k_{min}, k_{max})$ be the block range defining the selected ROI (see Figure 3.14 Then, the 3D texture ($VolumeHigh$ texture) which represents the volume data within the ROI has a dimension of: $((i_{max} - i_{min} + 1) \cdot 16, (j_{max} - j_{min} + 1) \cdot 16, (k_{max} - k_{min} + 1) \cdot 16)$.
3. Visualizing the model in two resolutions at once, with areas that need to be detailed in high resolution and the rest of the areas in low resolution (Figure 3.13). To achieve it, we compute and send to the GPU both 3D textures (the $VolumeLow$ and $VolumeHigh$ textures).

The rendering algorithm is based on a classical GPU-Based Ray-Casting [89, 90]. The fragment shader algorithm is presented in Algorithm 1. We show its most complex version, visualizing two resolutions at once. For simplicity we only show the code related to the ray traversal. Previously, the $EntryPoint$ of the ray in the volume, the maximum distance to travel ($LengthMax$) and the $DeltaVector$ which defines the advancing along the ray have been computed.

Algorithm 1 The Adaptive Raycasting Algorithm

$position \leftarrow EntryPoint$
$lenght \leftarrow 0$
while $length < LengthMax$ & $color.a < 1$ **do**
  $k \leftarrow texture3D(BlocksID, position).r$
  $HighPosition \leftarrow k \times getHighCordinates(position, HighResolutionSize)$
  $scalarLow \leftarrow texture3D(\text{VolumeLow}, position).r$
  $wLow \leftarrow 0.5$
  $scalarHigh \leftarrow texture3D(\text{VolumeHigh}, HighPosition).r$
  $wHigh \leftarrow 1.0$
  $scalar \leftarrow k \times scalarHigh + (1 - k) \times scalarLow$
  $W \leftarrow k \times wHigh + (1 - k) \times wLow$
  $colorSample \leftarrow getColor(TF, scalar)$
  $alphaSample \leftarrow getOpacity(TF, scalar)$
  {Front-to-back compositing}
  $color \leftarrow color + (1 - color.a) \times colorSample \times alphaSample \times \mathcal{W}$
  $color.a \leftarrow color.a + alphaSample \times (1 - color.a) \times \mathcal{W}$
  {Advance Ray Position}
  $position \leftarrow position + DeltaVector$
  $length \leftarrow +Delta$

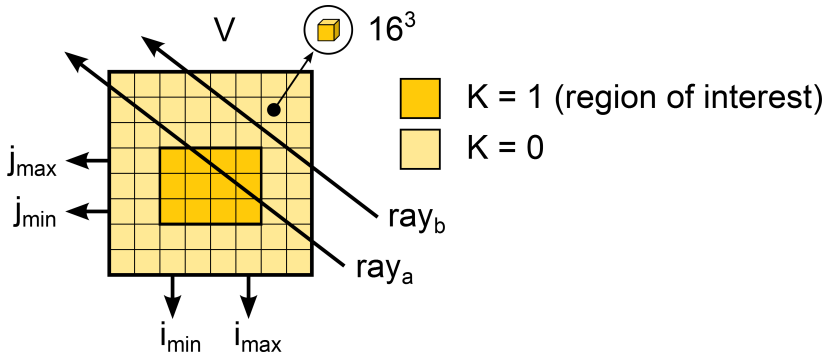


Figure 3.14: The block structure of the model, a region of interest (in orange) and the specification of the *BlocksID* texture (shown in 2D, for clarification).

In the simultaneous visualization of two resolutions, the algorithm must detect if the ray intersects the ROI (ray_a in Figure 3.14) or if it only traverses the low-resolution region (as ray_b, in Figure 3.14).

In the first case, and depending on the block we are traversing, we must

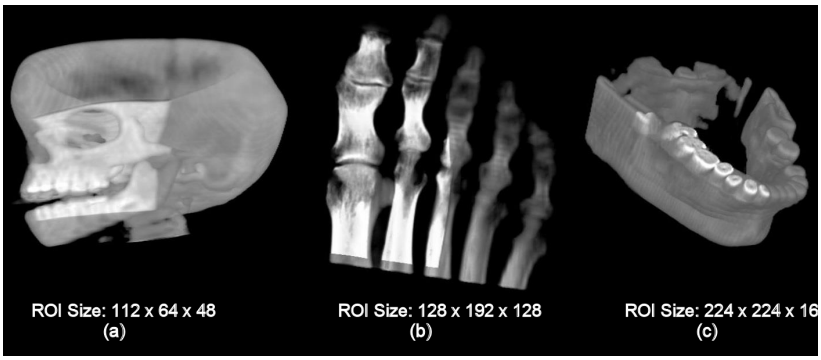


Figure 3.15: The Skull, Foot and Jaw models with the ROI at the maximum level of detail. The low resolution has been rendered after applying 2 Wavelet reconstruction steps in all cases.

sample the $VolumeLow$ 3D texture or the $VolumeHigh$ one. After many experiments, we concluded that having a driving 3D texture was much more efficient than including an ***if*** clause in the shader algorithm. Our solution consists on having a small 3D texture (called $BlocksID$ in what follows) with no linear interpolation (the access is defined as NEAREST) which characterizes blocks in the ROI from blocks outside the ROI. This texture has one texel per block of the initial volume, with a single bit information per texel. In a $256 \times 256 \times 256$ volume model, the size of the $BlocksID$ texture is therefore $16 \times 16 \times 16$. When a ROI is selected on the mobile device by the user, the $BlocksID$ texture is constructed accordingly. Figure 3.14 shows the values in this texture: $k = 0$ outside the ROI, $k = 1$ inside the ROI. A simple access to this texture in the fragment shader retrieves the value k, which is afterwards used as a weighting factor between the $VolumeHigh$ and $VolumeLow$ values and avoids the use of ***if*** clauses.

The Algorithm 1 computes $HighPosition$ so that ray positions within the ROI are scaled in accordance to the $VolumeHigh$ texture coordinates, and multiplies it by the factor k to avoid out-of-range texture fetch. The density values ($scalarLow$ and $scalarHigh$) on the textures $VolumeLow$ and $VolumeHigh$ are then retrieved, being weighted according to the k value. A second weighting factor $\mathcal{W}$ is computed to enhance accumulated values in the ray positions inside the ROI (the values $wLow$ and $wHigh$ can be tuned by the user to better enhance the high resolution details). Then, the $colorSample$ and $alphaSample$ values are computed according to

the transfer function, and they are accumulated with the standard volume ray-casting discrete equations. The final steps of the fragment shader loop are devoted to updating the position in the ray and the traversed distance (*length*) along it.

Figure 3.15 shows some snapshots of the interaction with the models we used during our experiments. In this case the ROIs are rendered at the maximum level of detail, while the low resolution has been rendered after applying two wavelets reconstruction steps.

3.8 Experimental Results and Discussion

We have performed different experimental tests to evaluate the compression, quality and efficiency of our approach.

Table 3.1 summarizes the compression results obtained for the three models used in our tests (views of the models are presented in Figure 3.5). Model resolutions spread from $256 \times 256 \times 112$ to $512 \times 512 \times 48$. Compression rates are computed with 800 iterations of the initial Laplacian transfer function-aware smoothing. The initial sizes of the models are 6 MB (Skull), 16 MB (Foot), and 12 MB (Jaw). The sizes of the compressed files if the U uniforming function is not used, are 3 MB, 5.5 MB and 5.06 MB. However, it can be observed that higher compression rates are achieved when both the uniforming function U and the Laplacian smoothing L are used: the sizes of the compressed volume in this case are 1.91 MB, 2.3 MB, 3.06 MB and 5.4 MB respectively. They correspond to compression rates of 32%, 14%, and 25%, with compressed file sizes that in some cases are of the order of a single photographic image. Table 3.1 also shows the model sizes corresponding to intermediate Wavelet steps.

Figures 3.16 and 3.17 show the effect of Laplacian smoothing as a function of the number of iterations in the server. Compression rates of the order of 90% (size of the compressed file of the order of the 10% of the initial volume model size) are obtained when 800 smoothing iterations are performed. The Figure shows that the model size is monotonically decreasing when this number of iterations increases. Perceived visual quality (Figure 3.17)

Table 3.1 Resolution and compression rates for tested models. The first column for each model (marked as $U + L$) presents the final model sizes (in MB) when the Uniforming function and the Laplacian Smoothing are applied. The Mark (✕) in the second column corresponds to the case where neither Uniforming function nor Laplacian Smoothing are applied. Dequantization and de-uniforming parameters (see Section 3.5) are classified as **Other**. The sizes confirm the almost insignificant influence of this data in the models data streams.

Model	**Skull**		**Foot**		**Jaw**	
	$U + L$	✕	$U + L$	✕	$U + L$	✕
Data Size MB	6		16		12	
Resolution	$256 \times 256 \times 112$		$256 \times 256 \times 256$		$512 \times 512 \times 48$	
Step 1 MB	1.2	2.1	1.1	2.3	1.8	3.0
Step 2 MB	0.5	0.8	1.0	1.2	1.0	1.8
Step 3 MB	0.09	0.1	0.2	0.2	0.2	0.2
Step 4 MB	0.01	0.01	0.03	0.03	0.03	0.03
Other MB	0.01		0.01		0.03	
Total MB	1.91	3	2.3	5.5	3.06	5.06

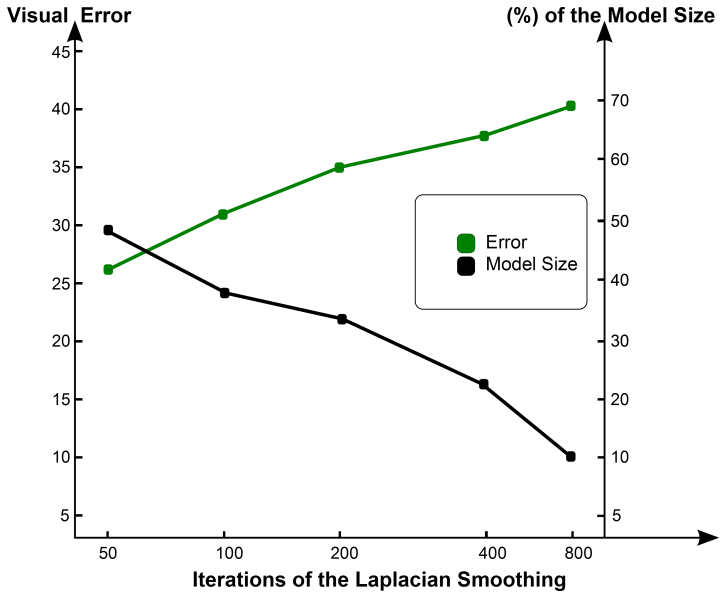


Figure 3.16: Laplacian smoothing. Perceived Visual Quality computed with the HDR-VDP-2 error (green curve shows the visual error) and Compression Rates (black curve represents the model size) for the Skull model.

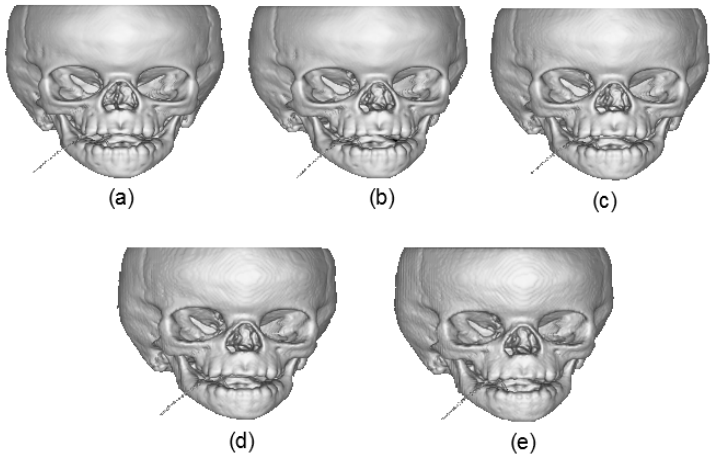


Figure 3.17: Visualization of the Skull model after 50 (a), 100 (b), 200 (c), 400 (d) and 800 (e) iterations of the Laplacian filter and with Phong illumination. The artifacts in cases with higher iterations (and higher visual error measures) result from poor gradient estimation.

is however decreasing because of smoothing in semi- transparent volume regions and subsequent poor gradient estimation. We have observed that a good compromise is achieved between 400 and 800 Laplacian iterations.

Renders in Figure 3.20 use ROI and context region at different decompression Wavelet transform. Figure 3.20-(a) low-resolution 3D textures of $32 \times 32 \times 14$ and high-resolution 3D textures of $64 \times 64 \times 64$. The corresponding memory requirements in the GPU of the client are 115 KB and 197 KB, respectively. In Figure 3.20-(b), the low-resolution 3D texture has a resolution of $64 \times 64 \times 28$ and high-resolution 3D textures are $64 \times 64 \times 64$. The corresponding memory requirements in the GPU of the client are 230 KB and 197 KB, respectively. Finally, Figure 3.20-(c) shows a low-resolution 3D texture of $128 \times 128 \times 56$ and high-resolution 3D texture of $64 \times 64 \times 64$. These resolutions require 345 KB and 197 KB respectively. Presented values are significantly lower than the 7.34 MB required when the whole 3D Skull model is ray-casted.

Images in Figure 3.21 show the Skull model after applying two and three wavelet reconstruction steps using a TF to simulate the skin.

Renders in Figures 3.13, 3.22 and 3.23 have been obtained on a HTC One smartphone. Models were transmitted to the mobile client device, individual blocks were decompressed on demand in the CPU of the client and they were ray-casting rendered in its GPU. The server was an Intel Core Duo CPU with 2.33 GHz and a GeForce 9600 GT Graphic Card of 512 MB of memory.

The curves in Figure 3.18 show the interaction Frames per second (*fps*) for different Wavelet reconstruction steps in both a PC and a mobile client, for the models in Figure 3.5. The *fps* decreases while better resolutions are selected with more Wavelet reconstruction steps (W^{-1}).

Figure 3.19 shows some snapshots of the interaction with the Skull, Foot and the Jaw model after applying planar sections to achieve a better analysis of internal regions in both ROIs and low resolution portion of the models.

Our visualization scheme uses ray-casting rendering in order to support multiresolution. Ray-casting algorithms strongly facilitate the visualization of different regions at several levels of details, supporting different sampling

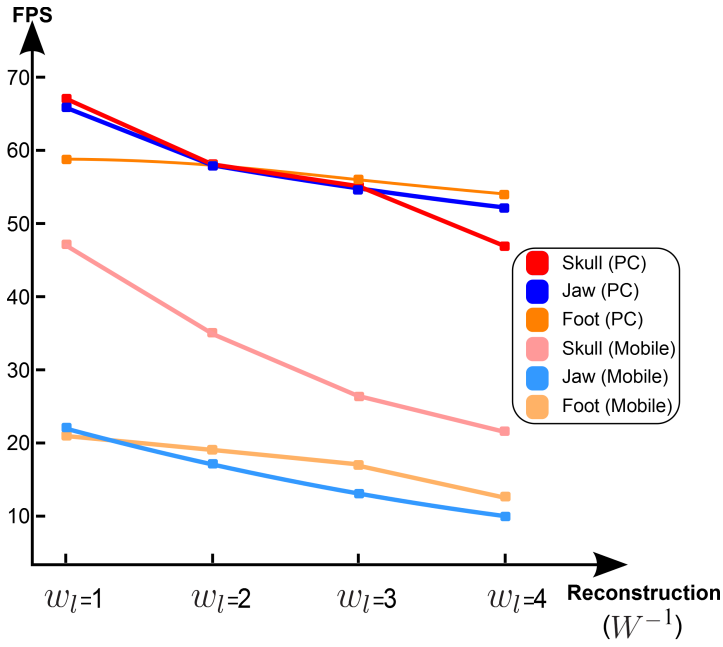


Figure 3.18: Frame rates of the interaction with the studied models. Frame Per Seconds (*fps*) for the Skull and Foot model at three different Wavelet reconstruction steps and the *fps* for the Jaw model at four different Wavelet reconstruction steps.

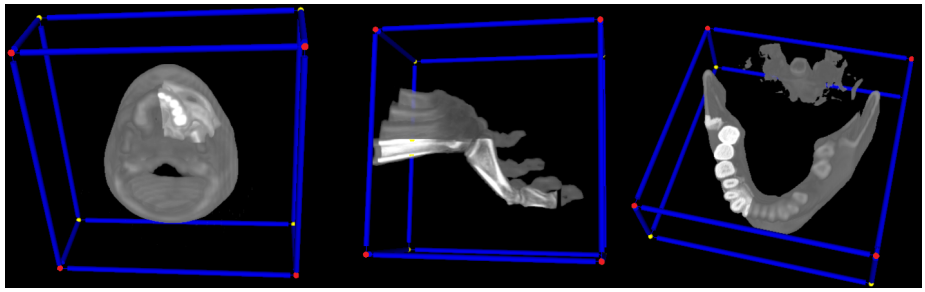


Figure 3.19: Planar sections in a multiresolution rendering of the Skull, Foot and the Jaw model. The plane selected is the one defined by the red points in each case.

rates in neighbor regions. They provide a higher performance in multiresolution models, when compared to slicing-based techniques. Schemes like [91] and [19] use texture slicing and do not support neither multiresolution nor decompression in the mobile device. It is therefore difficult to further compare our results with those in previous papers, although our scheme

could also be used in a texture slicing rendering context in cases where multiresolution is not required.

According to our experiments, the use of illumination algorithms avoids user interaction in the client side. Therefore, it is recommended to apply illumination only in cases where static images are required. Instead, we used a weighting factor $\mathcal{W}$ to enhance accumulated values in the ray position inside the ROI. As is explained in Section 3.7.

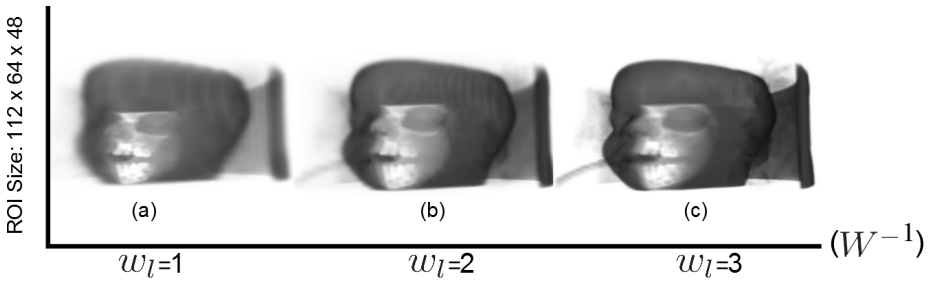


Figure 3.20: Multiresolution rendering of the Skull model. The selected transfer function shows the semi-transparent skin to visualize the Skull bones and the teeth. An eighth resolution model (one Wavelet reconstruction step) with a ROI at maximum resolution, column (a). A quarter resolution model (two Wavelet reconstruction steps) with maximum resolution ROI, column (b) and a half resolution model (three Wavelet reconstruction steps) with a maximum resolution ROI, column (c).

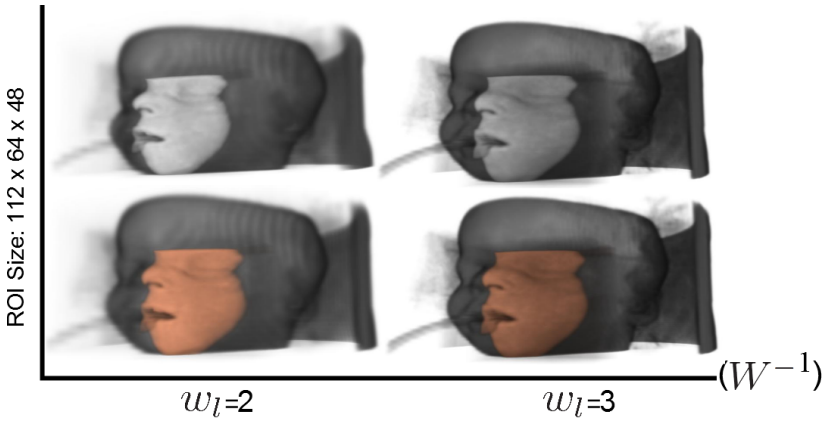


Figure 3.21: Multiresolution rendering of the Skull model. A quarter resolution model (left) and a half resolution model (right), with a transfer function applied to represent skin. In the second column, the RGBA simulates the skin tone without ilumination.

Figure 3.22: Rendering of the Skull, Foot and Jaw models in the HTC One Smartphone. The four columns show these models without illumination and two Wavelet reconstruction steps, without illumination and four reconstruction steps, with Phong illumination and two reconstruction steps, and with Phong illumination and four Wavelet reconstruction steps.

Figure 3.23: Rendering of the Skull, Foot and Jaw models in the HTC One Smartphone. The four columns show these models with Phong illumination and after applying one, two, three and four Wavelet reconstruction steps, from left to right respectively.

3.9 Conlusions

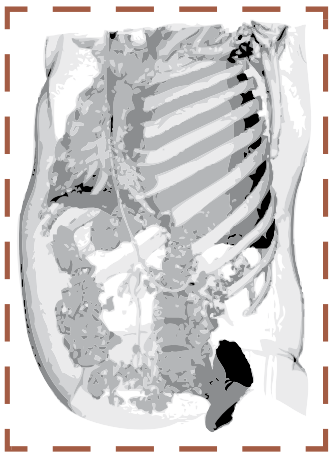
We have presented a new scheme for interactive volume in small mobile devices which has a number of new and unique features. Clients receive the whole model, compressed with a proposed TF-aware scheme which is able to achieve high compression rates. Decompression in the CPU of the client is efficient and can be performed on the fly during interaction. Our compression scheme is local, block-based and uses a Haar-Wavelet approach together with perceptual-based quantization. The consequence of this particular choice is that the size of the 3D textures in the mobile device is significantly smaller. Inspection of complex volume models with maximum level of detail in selected regions of interest becomes feasible: our scheme supports model sizes that otherwise could not be handled. The ray-casting algorithm in the GPU of the client is adapted to the block structure, being able to simultaneously deal with regions having different levels of detail.

One of the present limitations of this approach is that the preprocess in the server depends on the transfer function TF. Modifications of this TF require a new transmission of the model to the client. We think that this aspect is not critical, as the standard workflow in medical diagnose is usually based on a small number of pre-established transfer functions that enhance particular structures. Moreover the rendering frame rate slows down when Phong illumination is used in the ray-casting algorithm, because of the extra texture queries for gradient computations.

CHAPTER 4

REMOTE EXPLORATION OF VOLUME MODELS USING GRADIENT OCTREES

This chapter presents a transfer function-aware scheme for the remote interactive inspection of volume models in client server architectures.



Contents

4.1 Introduction

In this Chapter, we present a transfer function-aware scheme for remote interactive inspection of volume models in client-server architectures with the objective of supporting multi-resolution, avoiding gradient computations in the client device and sending a very limited amount of information through the network. Our novel data representation (named *Gradient Octrees*) can be progressively transmitted to the client in a compact way while achieving a minimum loss of visual quality as compared to state of the art ray-casting renderings. Visual volume understanding can be complemented by showing 2D sections of the original volume data on demand, and by a number of additional interactive tools. The main contributions of our approach are:

- An octree, multi-resolution volume data representation for client-server architectures. Our representation is based on a predefined set of transfer functions and succeeds in producing good quality expressive visualizations in the client with a limited loss of visual quality.
- An efficient compression algorithm which encodes gradients and material properties, together with a data transmission scheme which is able to progressively send deeper octree levels with a minimum amount of octree structure information (one bit per son node). A recovering algorithm that succeeds in reconstructing a full copy of the octree in the client from the received information.
- A GPU-oriented encoding of the proposed hierarchical data structure with explicit volume gradient information in octree nodes, to avoid gradient computations during GPU ray-casting.
- An interaction paradigm in the client which supports planar volume sections with high resolution volume information and interactive extrusion of specific structures.

In Section 4.2 we start by presenting a general framework (named Coherent-Traversal Trees) for the coherent maintenance of identical copies of hierarchical data structures in different computers. Gradient Octrees are then

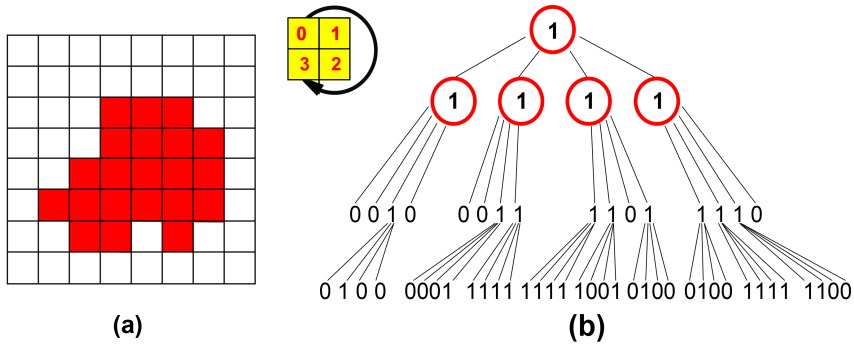
introduced in Section 4.3 as a particular case of Coherent-Traversal Trees for multiresolution volume representation.

4.2 Coherent-Traversal Trees

Distributed and client-server applications may require hierarchical data structures that must be shared among different computers. The usual goal is to maintain identical copies of the trees in all platforms while sending a minimum amount of information over the network. To fulfil this purpose, in this Section we present a special kind of hierarchical data structures which we name Coherent-Traversal Trees (CTTs in what follows).

Although the Coherent-Traversal Trees concept is general and can be used for any hierarchical data structure, let us illustrate it in the case of a four-level quadtree space subdivision, as shown in the example in Figure 4.1. Relevant data is in the red region in (a). By adopting a clockwise ordering of son nodes as shown in the top, the resulting quadtree structure is the quadtree in (b). Nodes are labeled as "0" if they encode a void region (*White* nodes) or "1" if they encode a region with relevant data. Type "1" nodes in all levels except the deepest one are *Grey* nodes, whereas type "1" nodes at the deepest level are *Terminal Grey* nodes that point to localized data in the corresponding voxel. Node circles have been omitted in the last two tree levels for clarity.

Hierarchical data structures can be classified into *Maximal Subdivision* Trees and General Trees. In *Maximal Subdivision* Trees, type "1" nodes with relevant data are always subdivided and refined until reaching the deepest level. The case shown in Figure 4.1 corresponds to a *Maximal Subdivision* tree. *Maximal Subdivision* Trees are well suited for multiresolution volume representations, as any voxel with relevant data (as shown in (a) in Figure 4.1) can be represented at different levels of detail when its tree ancestors are visited in a bottom-up way, see the tree structure (b) in Figure 4.1. On the other hand, General Trees can have type "1" nodes with relevant data at any level, without requiring them to be further subdivided. In General Trees, branches can end up in void regions (*White* nodes), in regions with relevant data not requiring further refinement, or regions corresponding to



(c) 1111 0010 0011 1101 1110 0100 0001 1111 1111 1001 0100 0010 1111 1100

(d) 1(001(0110)0)1(001(0001)1(1111))1(1(1111)1(1001)01(0100))1(1(0100)1(1111)1(1100)0)

(e) 10010110010010001111111111111110010101001101001111111000

Figure 4.1: A four- level quadtree space subdivision. Relevant data (a). Quadtree structure (b). Bit stream $\boldsymbol{B}$ when a *per level* CTA is performed (c). Bit sequence of the *preorder* traversal of the CTA (d) and (e).

the deepest tree level. An example of General tree is shown in Figure 4.2. In what follows, nodes with relevant data not requiring further refinement in General Trees (named as Black nodes) will be noted as type "1" nodes with all their sons being of type "0", to remark that they mark the end of the corresponding tree branch. The example in Figure 4.2 shows one of these *Black* nodes. It represents a set of 4 voxels in Figure 4.1 (a) which we assume that can share a common joint representation.

Let us start with some **definitions**. Without loss of generality, in what follows we will assume that trees are represented as a list (or array) of individual nodes. Nodes contain structural information and, in some cases, pointers to data information. Structural information of any node n includes the node type $\mathtt{t}(n)$ (in our proposal, 0 or 1) and information about its location in the tree and related nodes. We will consider verbose *m-ary* trees with well-defined node functions that encode the level $l(n)$ of the node, the set of m pointers to its son nodes $s_1(n), .., s_m(n)$, a pointer $\mathtt{f}(n)$ to its parent, and a pointer $\mathcal{D}(n)$ to its associated data. The level of any node is 1 plus its parent level, the level of the root node being zero by definition. The depth of the tree is the maximum level of its nodes. The tree root t_r has $\mathtt{f}(t_r) = nil$. In *Maximal Subdivision* trees, the node n is a tree leaf **iff** $s_k(n) = nil \ \ \forall k$.

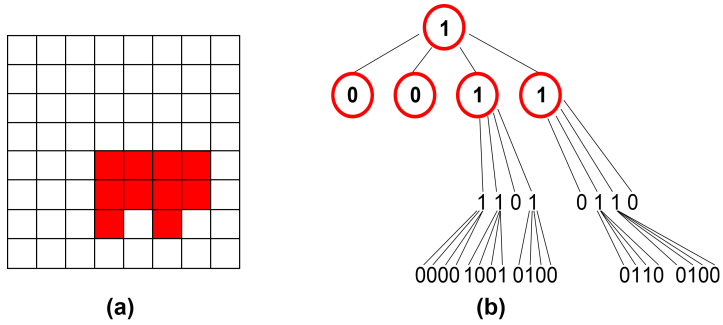
In General trees, however, leaves include nodes fulfilling $s_k(n) = nil \ \ \forall k$ and nodes having only White sons, $s_k(n) = 0 \ \ \forall k$. In hierarchical space subdivision trees, sibling nodes are ordered by following a pre-established geometric order, like the clockwise sorting used in Figure 4.1. In what follows, tree information will be classified into two different data structures: the tree structure $\boldsymbol{S}$ and the tree data $\boldsymbol{D}$. The tree structure $\boldsymbol{S}$ includes basic information allowing to retrieve $\mathtt{t}(n)$, $l(n)$, $\mathtt{f}(n)$, $s_1(n), .., s_m(n)$ and $\mathcal{D}(n)$ for all tree nodes n. Tree data $\boldsymbol{D}$ is a repository of application-dependent node data, as pointed by $\mathcal{D}(n)$. For the sake of simplicity, in what follows we will consider that $\boldsymbol{D}$ has an array structure, $\mathcal{D}(n)$ being indexes to specific $\boldsymbol{D}$ elements. Depending on the application, $\mathcal{D}(n)$ can be defined for all nodes n with $\mathtt{t}(n) = 1$, or only for a subset of them. For instance, in Figure 4.1, we could consider that all *Grey* and *Terminal Grey* nodes have a well-defined $\mathcal{D}(n)$ with associated data, as in multi-resolution trees. Alternatively, we could consider that only leaf nodes have associated data and a well-defined $\mathcal{D}(n)$. Nodes with undefined $\mathcal{D}(n)$ will be assumed to have $\mathcal{D}(n) = nil$. Anyway, and without loss of generality, in what follows we will consider that $\mathcal{D}(n)$ is well defined for all nodes n with $\mathtt{t}(n) = 1$.

Coherent-Traversal Trees are trees that use a common Coherent Traversal Algorithm (CTA), implicitly shared by all computers in a distributed application (or by client and server). Identical copies of the Coherent-Traversal Tree are maintained in distant machines by sending compact bit streams $\boldsymbol{B}$ everytime the tree (or a subtree) structure $\boldsymbol{S}$ is modified in one of the computers. The bit stream $\boldsymbol{B}$ contains one bit per tree node, encoding its type $\mathtt{t}(n)$. The sequence $\boldsymbol{B}$ is $\mathtt{t}(n_0)$, $\mathtt{t}(n_1)$, ..., $\mathtt{t}(n_N)$ where N is the total number of tree nodes and the ordering n_0, n_1, ..., n_N is given by the Coherent Traversal Algorithm. We will name b_k the bit $b_k = t(n_k)$, the index of the bit b_k being k from now on. The sender computes $\boldsymbol{B} = CTA(\boldsymbol{S})$. Then, under certain conditions, the receiver will be able to reconstruct the new tree structure $\boldsymbol{S}$ by using a CTA-based reconstruction algorithm, $\boldsymbol{S} = R_{CTA}(\boldsymbol{B})$.

Figure 4.1-(c) shows the bit stream $\boldsymbol{B}$ in (c) for our illustrative example when a **per-level CTA** is performed (the root node is considered *Grey* and it is not included in $\boldsymbol{B}$). Sibling nodes are underlined to facilitate reading. In this case 4.1-(c) , we start visiting nodes at level l just when all $l-1$-level nodes have been visited in a left-to-right constant-level traversal

of the representation in Figure 4.1-(b). Figure 4.1-(d) and Figure 4.1-(e) show the resulting bit sequence when a ***preorder* CTA** is performed. In (d), parenthesis containing the son nodes for every *Grey* node have been included for clarification. The bit stream ***B*** in Figure 4.1-(e) is exactly the same as in Figure 4.1-(d), with parenthesis removed.

Figure 4.2 shows an additional example with a General tree. It presents results from the same CTAs as Figure 4.1, in a different case with a smaller data region, resulting from clipping the data in Figure 4.1. In Figure 4.1, the bit stream in (c) is 7 Bytes long, while the bit stream in the case (c) of Figure 4.2 is 4 Bytes.



(c) 0011 1101 0110 0000 1001 0100 0110 0100

(d) 001(1(0000)1(1001)01(0100))1(01(0110)1(0100)0)

(e) 00110000110010101001010110101000

Figure 4.2: A 4-level, General quadtree. Initial scene (a). Quadtree structure (b). Bit stream ***B*** when a *per level* CTA is performed (c). Bit secuence of the *preorder* traversal of the CTA (d) and (e).

The number of required bits to send the tree structure ***S*** over the network depends on the particular CTA being use. Bit streams resulting from per-level CTAs in *Maximal Subdivision* trees are exactly as long as the number N of nodes in the tree, with one bit in ***S*** per tree node. This is due to the fact that in this case ***S*** is self-informative, as shown in Figures 4.1-(c) and 4.2-(c). For *m-ary* trees, the bit stream can be read iteratively in blocks through a per-level traversal, the number of bits to be read at each level being m times the number of "1" bits contained in the block read in the previous iteration. This is clear in Figure 4.1-(c), for instance. The first 4

bits of the stream correspond to level one. The second iteration will read a block of $4 \cdot 4 = 16$ bits, which correspond to the complete level 2 of the quadtree. Finally, the third iteration will read a block of $4 \cdot 9 = 36$ bits, as the previous block contains a total of nine "1" bits. Bit streams resulting from per-level CTAs in General Trees are however longer, see Figure 4.2. In our proposal, these Bit streams for General *m-ary* Trees are as long as $N + m \cdot N_B$ where N is again the number of nodes in the tree and N_B is the number of Black nodes in the tree ($N_B = 1$ in Figure 4.2). In both *Maximal Subdivision* and General trees, reading bit streams with a per-level CTA has an automatic checking mechanism, as the end of the bit stream must occur exactly after the last block has been read, but never during a block reading operation.

Unfortunately, other CTAs do not have this automatic checking mechanism, as it can be observed from Figures 4.1 and 4.2. The main problem of *preorder* sequences is that the bit stream does not provide any distinction between *Grey* nodes and *Terminal Grey* nodes in General trees, and an explicit encoding of the tree depth in *Maximal Subdivision* trees. This means more formally, and given a bit b_k with a $b_k = 1$ value in the stream, its subsequent bit b_{k+1} can either represent the first son node $s_1(n_k)$ of the node n_k corresponding to b_k, the next sibling of n_k or even a quite distant node in the tree: just observe the corresponding node locations in Figure 4.1-(b) of the two "1" bits in Figure 4.1-(e) that correspond to the sub-sequence "1))1" in Figure 4.1-(c), or similarly, the sub-stream corresponding to the piece "0))1" in Figure 4.1-(c). In other words, the bit stream in Figure 4.1-(c) is unambiguous, whereas the one in Figure 4.1-(e) is ambiguous. The general solution for disambiguating the bit stream in Figure 4.1-(e) is including the information lost in the transformation from (d) to (e), which basically requires a separate encoding for *Grey* and *Terminal Grey* nodes in General Trees and an explicit encoding of the tree depth in *Maximal Subdivision* trees. This means doubling the length of the bit sequence by having two-bit node encoding in General Trees. In short, we can state the following property:

Property 1: Per-level CTAs in *Maximal Subdivision* trees are minimal, in the sense that they are able to send a tree structure $\boldsymbol{S}$ over the network with the most compact possible bit stream: one bit per tree node. Any other CTA requires extra information on the tree structure which must be included in the bit stream.

In *Maximal Subdivision* trees like the one shown in Figure 4.1, non per-level CTAs require an explicit distinction between *Grey* and Terminal *Grey* nodes, which basically forces the bit stream to have a two-bit encoding of the tree nodes. In other words, per-level CTAs are able to encode *Maximal Subdivision* tree structures $\boldsymbol{S}$ in bit streams with one bit per node, while other CTAs must be encoded in bit streams with at least two bits per node. On the other hand, and using our proposed Black node encoding, we have seen that the per-level CTA bit stream of a General Tree will have $N + m \cdot N_B$ bits, while the length of a preorder CTA will be $2 \cdot N$. Per-level CTAs will be minimal for General Trees in all cases where $N_B \ll N/m$, which is a reasonable and common hypothesis.

Property 2: Given a CTA being shared between two distant applications and a bit stream $\boldsymbol{B} = CTA(\boldsymbol{S})$ computed in one of them, the receiver application can reconstruct $\boldsymbol{S}$ from $\boldsymbol{B}$.

In the rest of this paragraph, we prove it in the case of per-level CTAs by showing that the information in the bit stream can drives an algorithm to generates the complete required information for any node in the tree. As every bit in the stream corresponds to a tree node, reconstructed nodes n_k can be labeled with the index of their bit b_k in the stream, $LB(n_k) = k$, $LB()$ being the labeling function. We will also define a second label $LD(n)$ for nodes n such that $\mathtt{t}(n) = 1$. In this case, $LD()$ is defined as $LD(n_k) = \sum_{i=1}^{k} b_i$. Now, observe that iterated block reading of the bit stream as defined above is automatically providing the level of all tree nodes, as $L(n_k) = Iter(b_k)$ where $Iter(b)$ is the iteration count of the block in which the bit b has been read. Moreover, $\mathtt{t}(n_k) = b_k$. The tree structure $\boldsymbol{S}$ can be then completed by defining parent and son indexes $\mathtt{f}(n_k)$ and $s_l(n_k)$. Given the bit blocks Bl_i and Bl_{i+1} corresponding to two subsequent iterations of the bit stream reading loop, and defining $Bl_{i+1,j}$ as the sub-blocks of consecutive m bits in Bl_{i+1}, we will say that two bits $b_1 \in Bl_i$ and $b_2 \in Bl_{i+1}$ are related if the the index $j2$ of the sub-block $Bl_{i+1,j2}$ containing b_2 is $j2 = \sum_{i=1}^{k} b_l$, where k is the index of b_1 within Bl_i and b_l represent bits of Bl_i. Parent and son indexes are immediately derived from couples (b_1, b_2) of related bits, as the parent of the node corresponding to b_2 is the node associated to b_1. Similarly, son nodes of the node corresponding to b_1 are all nodes associated to b_x such as that (b_1, b_x) are related, their ordering being the ordering of the bits b_x in the block Bl_{i+1}.

We will now show that bit sequences generated by CTAs contain the necessary information to reconstruct the tree structure $\boldsymbol{S}$, and that this is true for *Maximal Subdivision* trees and for General trees.

Terminal Grey nodes are easily identified as the nodes n of the deepest tree block Bl_d having $\mathtt{t}(n)$, and Black nodes of levels $l < d$ are the ones having an associated bit b_1 such that $b_x = 0$ for all related couples (b_1, b_x). Moreover, in the general case where $\mathcal{D}(n)$ is defined for all nodes n with $\mathtt{t}(n) = 1$, $\boldsymbol{D}$ can be represented as a compact array with dimension equal to the number of *Grey* nodes in the tree, as stated by the following property:

Property 3: In trees where $\mathcal{D}(n)$ is defined for all nodes n with $\mathtt{t}(n) = 1$, the labeling $LD(n)$ can be used to define a compact set of pointers (indexes) to $\boldsymbol{D}$. By simply defining $\mathcal{D}(n) = LD(n)$, data in $\boldsymbol{D}$ can be efficiently represented during tree encoding in a compact array without null elements. The sender machine must only obtain the sequence of labels $LD(n)$ from the CTA-based bit stream. Then, data elements can be suitably organized in the $\boldsymbol{D}$ array.

Moreover, per-level CTAs can also be used in the case of optimized partial tree updates, by computing and sending bit streams of subtrees $\mathbf{B_S} = CTA(\mathbf{S_S})$. In this case, $\mathbf{S_S}$ is the structure of the updated subtree and $\mathbf{B_S}$ is the transmitted bit sequence.

A similar property can be stated for trees in which $\mathcal{D}(n)$ is only defined on *Terminal Grey* nodes, as they all belong to the deepest tree block Bl_d and they can be enumerated.

In fact, it must be observed that per-level CTAs are able to send tree structures $\boldsymbol{S}$ over the network with less bits than tree nodes, simply by compressing the bit streams $\boldsymbol{B}$ with any lossless compression scheme.

4.3 Gradient Octrees

Our main objective is to compress an initial volume model V, send it to the client device, and generate visualizations in this client device which have good visual quality when compared with Direct Volume Rendering (DVR)

of V see Figure 4.5. More specifically, our goal is to design a hierarchical, multi-resolution volume data structure supporting compact progressive transmission to the clients together with decompression and high-quality rendering in the client GPU.

In this Section, we present a new representation of a voxel model which fulfills the properties of the Coherent Traversal Trees, so that it may be compactly transmitted by the server and easy to decompress at the client side.

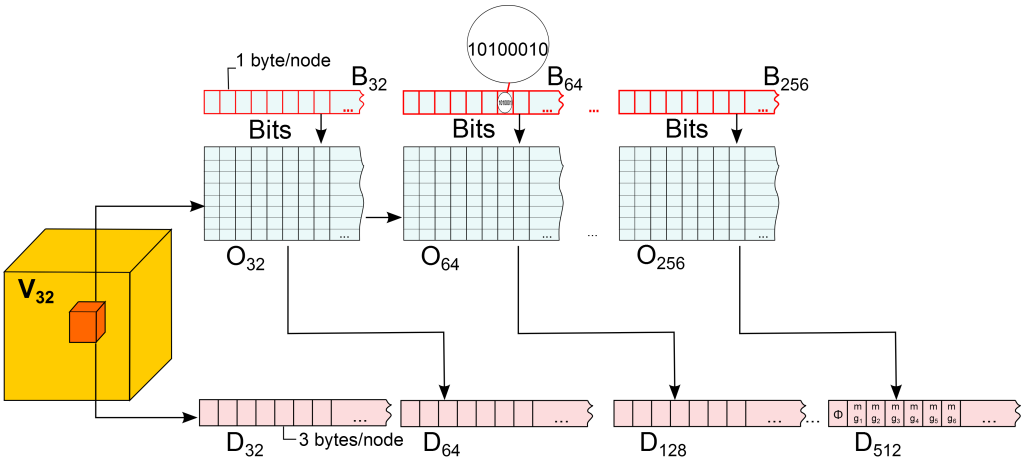


Figure 4.3: Encoding Gradient Octrees on a compact volume V_{32}, a set of index arrays O_l and a multi-resolution set of node data arrays D_l. O_l octree arrays are not send to the client. They are reconstructed from the more compact set of arrays B_l.

The Gradient Octree is a multi-resolution representation of a segmented volume V_E. Segmented volumes will be discussed in the next Section. Its name comes from the fact that its *Grey* nodes contain information on the volume gradient at different resolutions. It is based on a standard octree space subdivision, being computed in a bottom-up way. The depth of the Gradient Tree is defined by the resolution of V_E and the leaves of the Gradient Octree are voxels of V_E. Nodes in the Gradient Octree can be either *Grey* or *Nil*. *Grey* nodes store gradient and material information, encoding 3D cubical relevant regions of V_E. *Nil* nodes represent 3D cubical regions which after segmentation, only contain void voxels.

A Gradient Octree of depth d can be represented as d different lists of nodes, one of them per each tree level. Any octree level l is represented as a list of

its *Grey* nodes. *Grey* nodes in level l list contains their gradient vector, an index to its materials and eight indexes to their son nodes. Indexes to *Nil* son nodes are set to zero, while indexes to *Grey* son nodes point to their location (component) in the list of the next octree level. The cardinality M_l of the level l list is exactly the number of *Grey* nodes at this octree level l. For the sake of simplicity, we represent the *Grey* node list corresponding to any tree level l by two separate arrays with M_l elements each: the octree array O_l and the data array D_l. O_l stores indexes to the eight son nodes, while D_l stores node gradient and materials index.

The Gradient Octree is a Coherent Traversal Tree (CTT), (see Section 4.2) being the structure $\boldsymbol{S}$ on the Gradient Octree is the set of all O_l arrays, while the set of arrays D_l form the Gradient Octree data $\boldsymbol{D}$.

However, lists at coarser levels of the tree are too small and it is wise not to store them. The final Gradient Octrees representation consists on a small volume model V_{32} and two sets of per-level arrays, O_l and D_l.

Figure 4.3 shows an illustrative example of a complete octree representation of a volume V of resolution $r = 512$, with gradient and materials stored in the data array D_{512}.

For the sake of clarity, octree levels in Figure 4.3 and in what follows will be identified by their resolution. Data arrays of coarser octree levels (D_{256}, D_{128}, D_{64} and D_{32}) store gradient and materials data of *Grey* octree nodes at these levels. But, instead of storing data at levels coarser than D_{32}, we directly address D_{32} from the $32 \times 32 \times 32$ volume array V_{32}. V_{32} elements are either *Nil* (if the corresponding node of level 32 is *Nil*) or contain a two-byte index i_{32} to the data of this *Grey* node in the D_{32} array. This index i_{32} also indentifies the component of O_{32} which stores the information (indexes) to access the data of its sons in D_{64} and O_{64}.As it will shown in Section 4.5 pruning Gradient Otrees at resolution 32 saves texture addressing steps in the client GPU at a very limited overhead, the size of V_{32} being only $32 \cdot 32 \cdot 32 \cdot 2 = 65$ KBytes. In other words, we will start by sending V_{32} and D_{32} to the client, which represent a volume of resolution 32^3 that can already be ray-casted. Having a low-resolution volume model V_{32} in the client GPU will also facilitate ray traversal at rendering time, as shown in Section 4.5. To refine the octree one further level up to resolution 64, arrays O_{32} and D_{64} must be sent to the client. We use a two-Bytes encoding for O_l indexes when

the number of gray nodes in D_{2l} is below 2^{16}, and a three-Bytes encoding otherwise. This process must be repeated for all subsequent levels, the deepest level in the example in Figure 4.3 being reached when O_{256} and D_{512} are sent to the client.

Gradient Octrees are generated in the server and must be transmitted to the clients for its visualization. By using a Coherent Traversal Tree scheme with a per-level CTA, we compact the O_l arrays of the Gradient Octree in a lossless way. Instead of sending the arrays O_{32}, ..., O_{256}, we send a set of arrays B_{32}, ..., B_{256} with one byte per component as shown in red in Figure 4.3. Note that this is equivalent to store and transmit only one bit per node: the components in B_l simply represent the type of each son in one bit (0 if *Nil*, 1 if *Grey*) as already discussed in Section 4.2. We compute the compressed arrays B_l in the server, send them to the client, and reconstruct again the set of octree arrays O_l in the client from the information in B_l. Let us represent the j-index stored in the $O_l[i]$ element as $O_l[i][j]$. The algorithm is based on the CTA invariant, known by both server and client: taking into account that indexes in $O_l[i][j]$ (i=1..M_l, j=0..7) are either *Nil* or relevant indexes to son nodes, the subsequence of relevant indexes is the sequence of Natural numbers {1, 2, .. M_{l+1}} when $O_l[i][j]$ is traversed through an outer i loop and an inner j loop. Then, in the encoding step and for the i-Grey element in the list represented in O_l, the j-bit of the component $B_l[i]$ is set to one **iff** $O_l[i][j]$ is relevant, and it is zero **iff** $O_l[i][j]$ is *Nil*. The client can reconstruct O_l from B_l by simply traversing all bits in B_l and counting. First, all $O_l[i][j]$ are initialized to *Nil*. Then, every 1 in the j-bit of $B_l[i]$ is translated into an increment of one in a global counter of *Grey* nodes which is then assigned to $O_l[i][j]$. Note that instead of sending indexes, we simply send bits.

Progressive transmission of the Gradient Octree to the client device consists on initially sending the volume V_{32}, its data array D_{32} and the materials look-up table, followed by the transmission of a progressive sequence of array pairs B_l, D_{2l} representing individual levels of the octree for l=32, 64, 128, .. $r/2$ where r is the resolution of V_E. For every received level, the client generates a B_l-driven, increasing sequence of indexes to create a local copy of the array of indexes to son nodes, O_l. Note that O_l indexes point simultaneously to D_{2l} and O_{2l}. The volume V_{32} is sent to the GPU as a 3D texture, whereas arrays O_l and D_l are encoded as 1D textures.

Moreover, for transmission and storage purposes, *Grey* node data (gradient vectors and materials) is compressed to 3 Bytes. Consequently, D_l arrays contain elements of 3 Bytes each. Gradient vectors are lossly encoded by their dominant orientation and by the quantized value of the two remaining coordinates. Let us consider a cube centered at the origin, having a 512 x 512 grid in each of its six faces. Given a gradient vector, it stabs a grid cell in one of the cube faces. We encode the gradient by storing the index (0, .., 5) of the stabbed face and the two coordinates of the grid cell in the cube face. As these two coordinates require 9 + 9 bits, we use the remaining 6 bits to store the cube face and the node materials as a single value $v = f + 6 * m$ where f is the index of the face and m is an index to a look-up table of (up to ten) material pairs. If more materials/structures are needed to store the information of the segmented volume V_E we can increase from three to four the number of Bytes per element in the D_l arrays, but we have experimentally found that supporting 10 different boundaries between segmented regions is reasonable in most practical cases (see Section 4.4).

Figure 4.4 shows the main steps of our approach. First, the preprocess in the server where volume representation V_E is generated. Then, a compact Gradient Octree is created. Then, it is progressively sent to the client as a sequence of arrays with compressed information (see Figure 4.4). The client enlarges the received information in order to reconstruct a local copy of the Gradient Octree. Gradient decompression and ray-casting rendering is directly performed in the client GPU, as described in Section 4.5. Interaction in the client drives its GPU visualization, sends queries to the server to request further levels of the Gradient Octree and sends planar section queries when interactive cuts of the volume must be filled out with original volume information to fully understand the original data in V. In this last case, the sections server generates a texture with the appropriate sliced information from V, and sends it to the client. Section 4.6 details how these section textures are handled in the GPU. Interaction is autonomous and no further information is needed from the server, except when a new planar section is required.

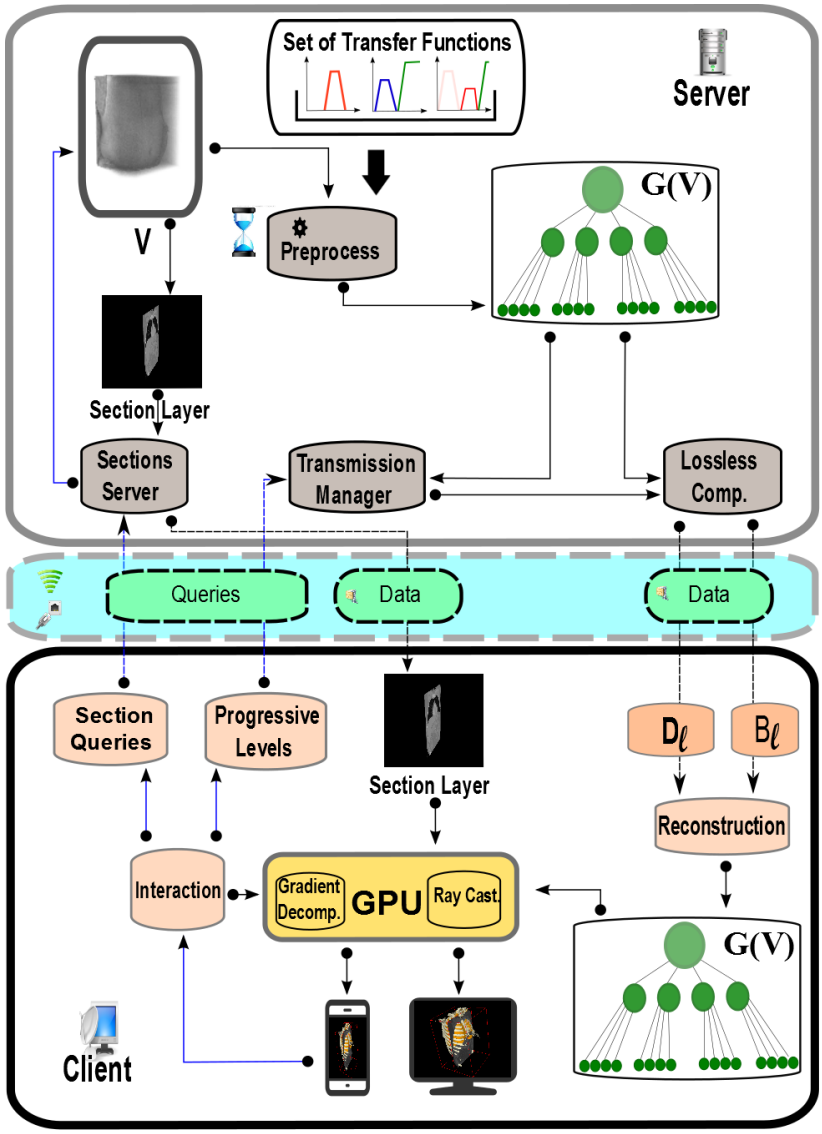


Figure 4.4: Overview of the proposed scheme. First the preprocess in the server generates the $V_E(TF_k)$. Then, the Gradient Octree $G(V)$ is created from the initial volume data V and the predefined set $\{TF_k\}$. $G(V)$ is progressively sent to the client as a sequence of arrays of compressed data. The client enlarges the received information in order to reconstruct a local copy of $G(V)$. Gradient decompression and ray-casting rendering is directly performed in the client GPU. Interaction in the client drives its GPU visualization, sends queries to the server to request further levels of the gradient tree and sends section queries when interactive planar cuts of the volume must be filled out with original volume information. In this last case, the sections server generates a texture with the appropriate sliced information from V and sends it to the client. Interaction is autonomous and no further information is needed from the server, except when a new planar section is required.

Note that our approach includes three compression steps. The first, is transfer-function aware and uses V and TF_k to compute the segmented V_E. On a second step, we compress gradient information to a total of three Bytes per *Grey* tree node (including material information) in D_l arrays. Finally, we succeed in sending the tree structure O_l with only one bit per node and in a lossless way through a CTT scheme. By adding these three ingredients, we are able to send multi-resolution volume information to the clients in a strongly compact way, as shown in Section 4.7.

4.4 Transfer Function Dependent Models: Edge Volumes

In this Section, we present a simple approach to compute a segmented volume useful to visualize models in a remote way.

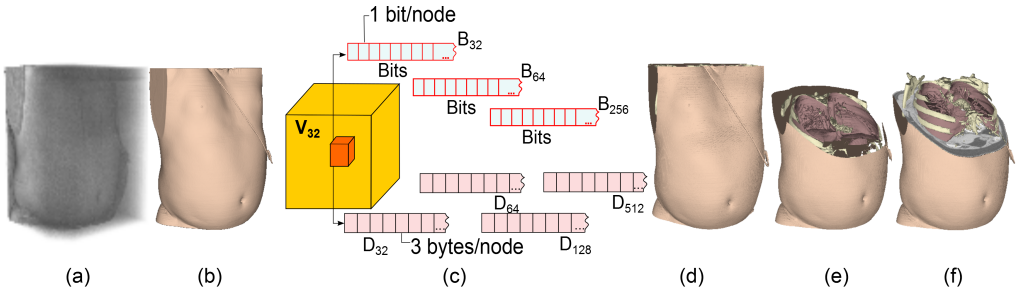


Figure 4.5: (a) We start from an initial volume model V and a set of transfer functions S, to create a (b) segmented volume model $V_E(TF_k)$. Gradient Trees (c) are multi-resolution representations of $V_E(TF_k)$ which can be progressively sent to client devices in a very compact way. Clients are able to reconstruct a copy of the Gradient Tree and ray-cast it in their GPU to interactively inspect volume structures, (d). Clients can perform planar sections (e) and render them with high-res information from the initial volume V, even adding offset structures in front of the section plane (f).

We start from the observation that in the clinical practice DVR visualizations are mostly based on well established transfer functions which enhance specific tissues and organs. Transfer functions are able to assign a color $c(p)$ and an opacity value $o(p)$ to any point p in a volume model V according to its properties. Transfer functions can be based on the density value of the scalar field V or they can be functions of densities, gradient values and other

attributes [92]. Our compression scheme is based on a predefined set $\{TF_k\}$ of transfer functions, $TF_k = \{t_1, t_2, ..., t_n\}$, each TF_k in $\{TF_k\}$ being able to classify a certain specific material in V. In our approach, the goal is to minimize the loss of information and quality when using transfer functions from the set $\{TF_k\}$, at the cost of having removed volume information that could have been relevant for TFs not in $\{TF_k\}$. Our approach is lossy in a selective way, as it tries to maximize the visual quality when using any of the transfer functions in $\{TF_k\}$.

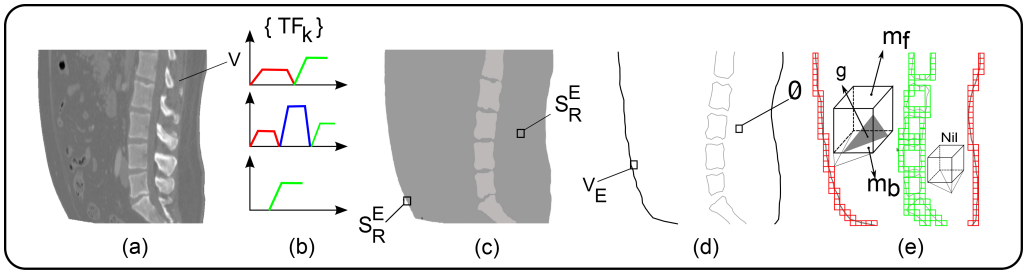


Figure 4.6: In (a), the initial volume model V. In (b) the set $\{TF_k\}$ of transfer functions which generate the segmented volume in (c). The *Edge* Volume model $V_E(TF_k)$ (with an *Edge* and a *Nil* voxel) is presented in (d), whereas (e) shows the gradient and materials stored in an *Edge* voxel of $V_E(TF_k)$ (a green triangle is shown for figure clarification).

Given a volume model V (Figure 4.6-a), let us consider the segmentation induced by each transfer function TF_k in $\{TF_k\}$, Figure 4.6-b. The segmented region $\mathcal{S}_R^E$ (Figure 4.6-c) is defined by assigning all volume points p having an opacity $o(p) > \varepsilon$ to the region $\mathcal{S}_k^E$ corresponding to TF_k, where ε is a given threshold. Alternatively, any other segmentation algorithm could be used to segment the same structure. In $\mathcal{S}_R^E$, voxels can be classified as *Uniform* voxels or *Edge* voxels. A voxel in V_R^E is *Uniform* **iff** its eight vertices belong to the same segmented region $\mathcal{S}_k^E$. A voxel in V_R^E is an *Edge* voxel **iff** its eight vertices come from two different regions $\mathcal{S}_j^E$ and $\mathcal{S}_k^E$.

Note that *Uniform* voxels in $\mathcal{S}_R^E$ are redundant, as $\mathcal{S}_R^E$ can be reconstructed from its set of *Edge* voxels. We therefore can define a compact auxiliary representation for $\mathcal{S}_R^E$, named *Edge Volume* V_E (Figure 4.6-d). Uniform voxels in $\mathcal{S}_R^E$ are stored as *Nil* voxels in V_E, while *Edge* voxels contain an index to a look-up table storing attribute information of the materials pair ($\mathcal{S}_j^E$, $\mathcal{S}_k^E$), plus the normalized gradient of V at the voxel center. As we have already introduced in Section 4.3, we store gradients in order to avoid extra

computations in the client GPU. V_E is a lossy compressed representation of V based on TF_k. Given a volume V and a set $\{TF_k\}$, its $\{TF_k\}$-based representation $V_E(TF_k)$ can be defined (Figure 4.6, d and e) as the union of all V_E:

$$V_E(TF_k) = V_E^1 \cup V_E^2 \cup ... \cup V_E^n \tag{4.1}$$

This union operation is performed in a voxel per voxel basis. Union between two *Nil* voxels is *Nil*, whereas the union between a *Nil* and an *Edge* voxel is an *Edge* voxel. An attempt to perform a union between two *Edge* voxels shows that V contains structures which are too narrow for the current voxel resolution. In this case, the resolution of the voxel model $V_E(TF_k)$ should be increased. Gradient Octrees are then generated from $V_E(TF_k)$ by recursive bottom-up simplification.

Node information in any upper level of the Gradient Octree is computed as the result of applying a simplification downsampling function to its eight son nodes.

The downsampling function must consider two different cases. Nodes in the upper level having only *Nil* son nodes are assigned a *Nil* node type. Otherwise, when at least one son node is not *Nil*, the gradient of the parent node is computed by applying a *SetGradient* function to the gradients of its non-*Nil* son nodes. One option for *SetGradient* could be to average the gradients of the son nodes, but this algorithm produces artifacts in upper tree levels when region boundaries with almost opposite gradients are merged. In our implementation, we concluded that the best option was to use a *selection* algorithm. We estimate an importance weight for each son gradient vector $g.s1$, .. $g.sn$, and then copy the gradient of the most important one to its parent's gradient. This is then repeated for the material information: we copy the materials pair of the most important son to the parent. After several experiments, we concluded that importance weights could be estimated in a suitable way as the width of the visibility cone of the node cube when observed from different view directions. As the exact computation of visibility cones may be slow, we have finally used a more simple heuristics: we compute the average of the son gradient vectors after removing outlier sons (outliers are sons being too different from their

average) and estimate importances by the scalar product between the node's gradient and this average.

4.5 Rendering Gradient Octrees

The proposed rendering algorithm is base on a standard ray casting algorithm in the client GPU, the only difference with classical algorithms being that rays traverse a virtual volume model with the same resolution r as $V_E(TF_k)$, instead of traversing $V_E(TF_k)$ itself. In Figure 4.7, an example with resolution $r = 512$ is shown. Ray-casting proceeds as usual by advancing along rays r_k from the observer with a uniform sampling of the volume along r_k. Then, for each sample s of r_k addressing a virtual voxel (i, j, k) of $V_E(TF_k)$, its volume information is found in the corresponding D_l.

Given a virtual voxel (i, j, k) the search of its data is based on the octree addressing properties. It is driven by the base-2 representation of i, j and k, as shown in Figure 4.7. In this case, their first 5 binary digits indicate the corresponding voxel in the low-res texture V_{32}. The index i_{32} found in this V_{32} voxel element points to the low resolution data in D_{32} (which we don't use if a higher resolution is required) and also to the array of its eight son indexes in O_{32}. A well-known property of binary octree subdivision ensures that next "three bit columns" in the binary representation of (i, j, k) are in fact son indexes s_l. Son indexes point to deeper octree levels and are able to drive the octree traversal to the right element in D_r containing the data of the virtual (i, j, k) voxel. Subtree traversal from the low-res voxel in V_{32} to the virtual voxel data is based on the recursion equation,

$$i_{2l} = O_l[i_l][s_l] \tag{4.2}$$

for l=32, 64, 128, .. $r/2$ The final index i_{512} points to the high-res data in D_{512}, but tree traversal can stop earlier if the virtual voxel is void and any index i_l in the chain is found to be zero (void voxel).

Let's assume that ray r_k is crossing voxel $i = 171$, $j = 312$, $k = 237$ in the virtual volume, Figure 4.7-left. In this case, Figure 4.7-right, the octree search starts in the voxel $(10, 19, 14)$ of V_{32} and is then driven by four son

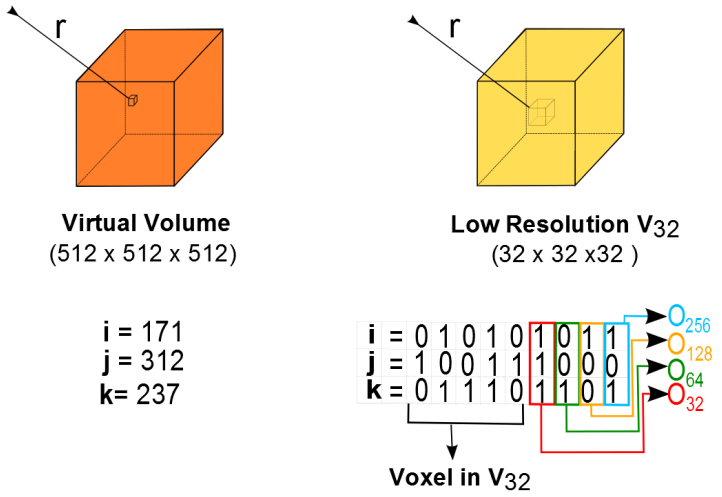


Figure 4.7: The virtual volume (left) and the low-res volume texture V_{32} (right), as used by the ray-casting algorithm in the client GPU. Octree addressing is based on the binary representation of voxel coordinates in the virtual volume.

indexes: $s_{32} = 7$, $s_{64} = 1$, $s_{128} = 4$ and $s_{256} = 5$ which recursively generate the indexes i_{64}, i_{128}, i_{256} and i_{512}. Reaching the deepest level information in a Gradient Octree of resolution $r = 512$ involves a maximum of six texture queries, to V_{32}, O_{32}, O_{64}, O_{128}, O_{256} and finally to D_{512}. Obviously, everything also works when lower resolution virtual volumes are considered, for multiresolution visualization.

After retrieving high-res data in D_{512}, materials and the gradient vector are decompressed on the fly in the GPU. The gradient face f, the two grid-coordinates of the gradient and the material index are decoded through simple bitwise operations and by one division by 6 to decouple f and the material index. Final gradient decompression involves specific computations for each of the six possible values of f, where the coordinate in the direction of the f normal vector is assigned a value 1 or -1 and the other two coordinates are linearly dependent on the coordinates of the grid cell within the cube face, followed by gradient normalization.

4.6 Interaction in the Client Device

In this Section, it is presented the user interaction facilities supported by the Gradient Octrees model in the client side. We will show that the interaction in the client is efficient and does not depend on further communication with the server, except when new planar sections are requested (see Figure 4.4).

Users can play with the camera by rotating around the volume and zoom in. They can also render the volume at different resolutions, if necessary. In this case, ray-casting rendering stops the Gradient Octree traversal before reaching its deepest level. Moreover, users are able to:

- Select transfer functions from the predefined set $\{TF_k\}$ to only visualize specific materials.
- Perform planar sections of the volume and enrich them with a visualization of the corresponding section of the initial volume V, if needed. To support this functionality, we have implemented rendering of 2D sections of the original volume data on demand.
- Render the cut volume from any viewpoint.
- Visualize offset structures in front of the section plane, and change its offset distance. In his case, we use the algorithm from [93].
- Change opacities of any of the visualized volume structures and render them as partially transparent.

Planar sections are managed by the ray-casting algorithm in a straightforward way. Section textures T_S of the original volume V are sent to the client by the Sections Server, on request (see Figure 4.4). The GPU shader receives the normalized section plane equation $a \cdot x + b \cdot y + c \cdot z + d = 0$ as four uniforms (a, b, c, d), together with the texture T_S. The alpha channel of T_S texels which correspond to external air is set to zero, to ensure transparency through air in external areas, as discussed in Section 4.7. Then, as the rendering algorithm is casting a ray, the signed distance to the section plane is computed for each sample. Samples on the plane (computed with a tolerance ε) are assigned the corresponding value of the texture T_S. Opacity

in non-transparent texels is set to the maximum value, to ensure that structures behind the plane will be occluded. Samples with a negative signed distance are processed in the standard ray-casting way. Moreover, samples having a positive signed distance s_d are only considered when some selected structures have been assigned a non-zero offset distance o_d. In this case, the ray-casting algorithm processes *Grey* octree nodes if $s_d < o_d$ and otherwise skips them.

4.7 Experimental Results and Discussion

We have tested the proposed multiresolution visualization approach with a number of volume models and transfer function. Results presented here correspond to the Thorax model with a (512^3) resolution and the Head model with a resolution of ($512 \times 512 \times 485$). Their initial volume sizes are 121 and 128 MBytes respectively, as shown in Table 4.1 . Tests have been run on a server with 6 GB of RAM, Intel Core 2 Duo at 3.16 GHz and a client with 4 GB of RAM, Intel Core 2 Duo at 3.06 GHz and Nvidia GeForce GTX z80.

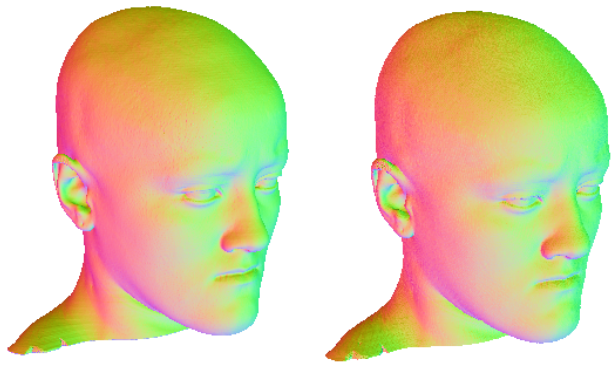


Figure 4.8: Quality of the proposed gradient compression scheme. The Head model without (left) and with (right) gradient compression is shown.

Rows in Table 4.1 correspond to resolutions from 32 to 512, showing the number *Grey* octree nodes, the size of textures B_k and D_k in KBytes, the amount of information (stream) sent over the network and the client storage

requirements for the Gradient Octree. As shown in Table 4.1, the most complex case (Thorax with skin, bones and lungs) requires 100 KBytes for sending the octree data at resolution 32, plus 165 KBytes for the 64-res octree level plus 738 KBytes for the 128-res octree level. The octree at 256-resolution requires sending $100 + 165 + 738 + 3217 = 4.22$ MBytes, whereas the full octree at the highest resolution requires 17.59 MBytes, equivalent to 3.4% and 13.5 % of the initial data size respectively. For the model with skin and bones in Table 4.1, these ratios are 2.4% and 10%, whereas the percentage values for the model in Table 4.1-(bottom) are 2% and 8.4%. High-res sections of the initial volume V, which are sent on demand, require less that 200 KBytes each. This is a drastic improvement over thin-client schemes which must send one image per frame. In short, compressed volumes at a 256-resolution require between 2 and 3.3 percent of the initial data, whereas these percentages grow up to values between 8.4 and 14.5 at the highest resolution.

We succeed in sending the tree structure $\boldsymbol{S}$ in a lossless way and with only one bit per node, through a sequence of compact arrays B_k. To the best of our knowledge, this outperforms all previous octree-based schemes, including [94], [95], [57] and [32]. Moreover, compressing gradients and materials in three Bytes is efficient, supports GPU decompression and suffers from a very limited loss in visual quality, as shown in Figure 4.12. In Figure 4.8, illumination renders when using the gradient vector itself instead of normal vectors are shown. Results with uncompressed gradients (left) are rather similar to renderings with compressed gradients (right).

Progressive model transmission and multi-resolution rendering is shown in the different columns of Figure 4.9. Some artifacts appear in resolutions lower than 256, but these coarse octrees are only used for approximate model rendering during progressive transmission. Anyway, artifacts can be avoided by using more sophisticated downsampling functions during the generation of the Gradient Octree, as already discussed in Section 4.4.

Figure 4.15 shows some snapshots of the interaction results in the client with different transfer functions. Images show planar sections with high-res information of the initial volume V, the bottom row presenting two images with offset structures in front of the section plane.

Preprocess times in the presented examples have always been between three

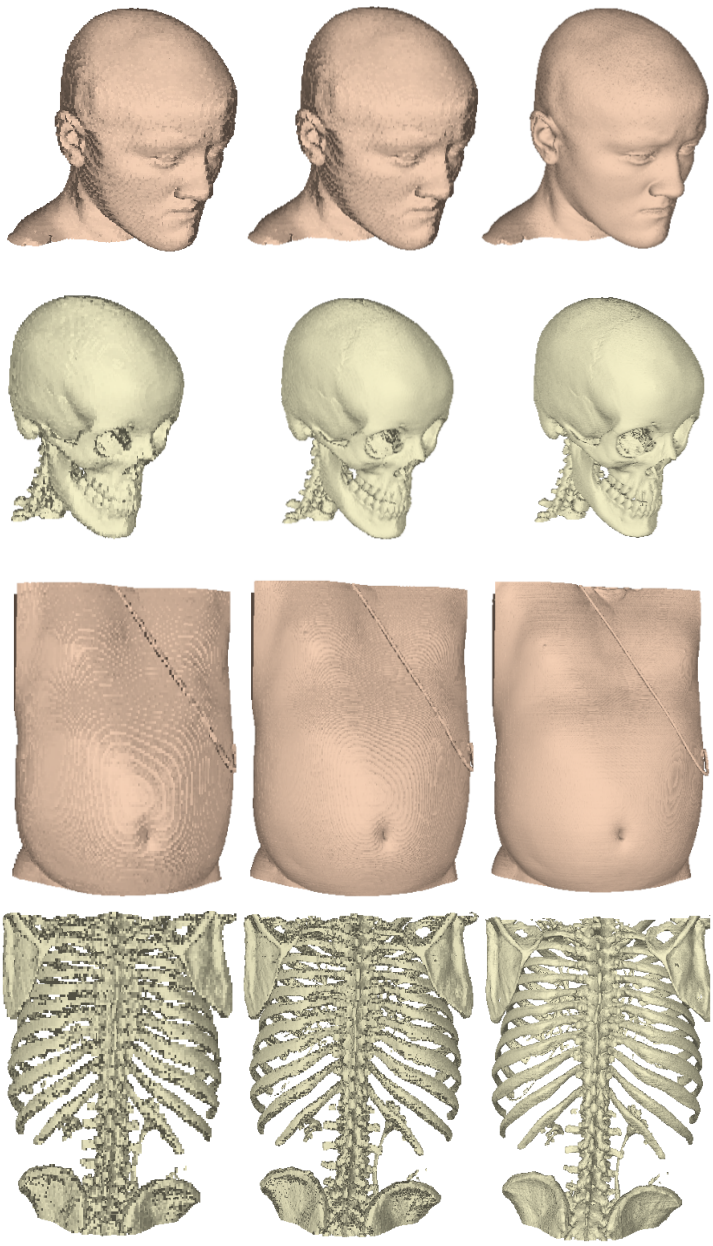


Figure 4.9: Progressive model transmission and multi-resolution rendering. The three columns show the Head and the Thorax models with two different transfer functions each, at resolutions of 128, 256 and 512.

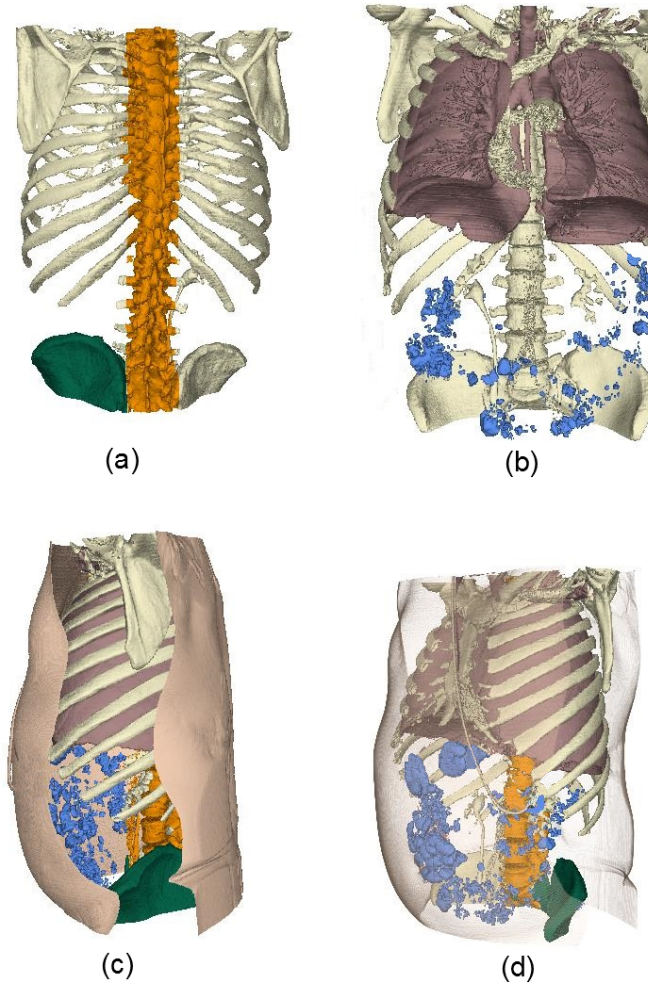


Figure 4.10: Snapshots of the interacion in the server side with six different structures of the Thorax model. Bones with three materials (a). Sectioned model, representing bones, lungs and intestinal gas (b). Ribs, backbone, kip, lungs, intestinal gas and the skin (c). Semi-transparent skin visualization (d).

and five minutes with a non-optimized code. Inspection and ray-casting in the client is fully interactive, as shown in Figure 4.14.

The curves in Figure 4.14, show the frame rates (*fps*) for the interaction with the Thorax and Head models in the server side. In the Thorax case, six tissues and structures were visualized during the user interaction (see Figure 4.10), while in the Head model interaction included: the skin and

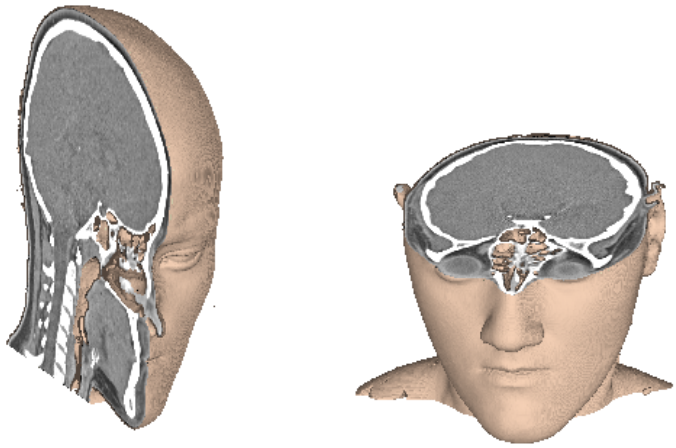


Figure 4.11: Two snapshots of the interaction in the client with different transfer functions. Images show planar sections with high-res information of the initial volume V.

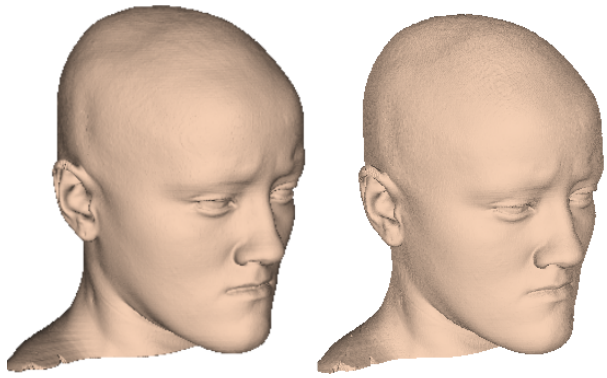


Figure 4.12: Ray casting of the original volume V a predefined transfer function (left) in $\{TF_k\}$ (left), compared to Gradient Tree ray casting in the client device (right).

skull bone.

Our compression results are better than similar client-server schemes for volume rendering, and compare favorably to Marching Cubes, *MC*-based approaches. While these last schemes must send an average of three triangles per voxel in segmented volumes like $V_E(TF_k)$, we succeed in encoding octree nodes at a rate of 3 Bytes/node, which is much lower than using MC triangles.

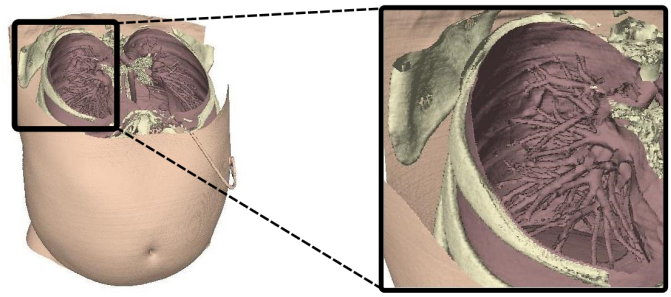


Figure 4.13: Interaction in the server application. The Thorax model sectioned by a cut plane (left) and a zoom of the alveolar region (right).

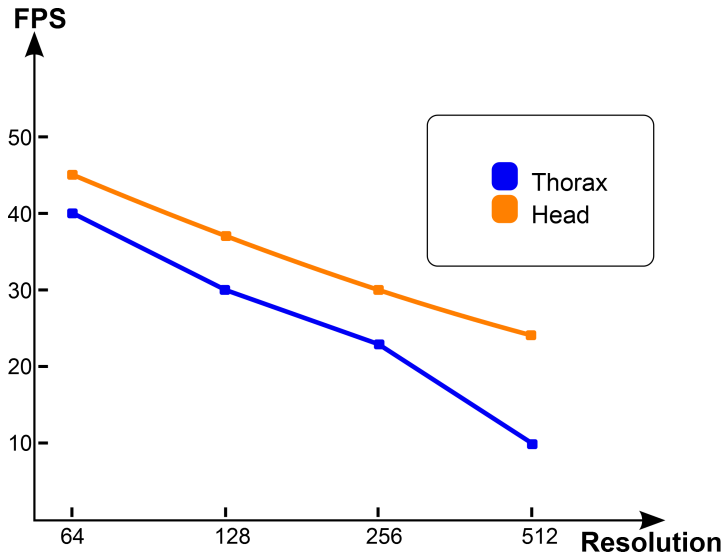


Figure 4.14: Frame rates of the interaction with the studied models (Head: 512×512×485 resolution, and 2 tissues; Thorax: $512 \times 512 \times 512$ resolution, with 6 tissues). The viewport resolution was 817×534 pixels.

Although out tests have been performed on a client with a PC architecture, in the next future we plan to migrate the decompression and interactive rendering algorithms to mobile devices.

Table 4.2 presents a study of the information transmitted from server to client when the base volume is V_{16}, V_{32} and V_{64} respectively, for the Head model. It can be observed that the main difference in all cases is the size of

the base volume model. Table 4.3 contains the most significant data from Table 4.2 in a compact way, and shows a comparison among the streamed data when using different base volume sizes. Starting the transmission with V_{64} increases in 437 KB the size of the data stream, whereas starting from V_{16} increases the amount of texture access during ray casting due to the inclusion of an unnecessary resolution level, without significantly decreasing the total amount of streamed information. A base volume V_{32} is adopted as a compromise in all our test and examples.

Table 4.4 shows a comparison among our approach and two recent client-server proposals for volume rendering [94], [32]. The comparison assumes that the same transfer function has been applied in all cases. The comparison has been made on the Thorax model with a resolution of 512^3. We conclude that our approach outperforms these two previous algorithms both in streaming the octree structure and the corresponding volume data.

The main limitation of the proposed approach is that the visualization in the client is restricted to the medical structures which have been pre-defined in the set $\{TF_k\}$ of transfer functions. We think that this should be acceptable in most practical cases, as medical doctors usually adopt standard transfer functions to enhance specific tissues and organs. Anyway, we are enhancing visual volume understanding by showing 2D sections of the original volume data on demand and by a number of additional interactive tools (see Figures 4.10,4.15 and 4.13). Any strong volume compression must be lossy and specific details are always lost. Our proposal focuses on respecting features which are visible with the TF's of the predefined set and on displaying real sections of the initial volume, on demand.

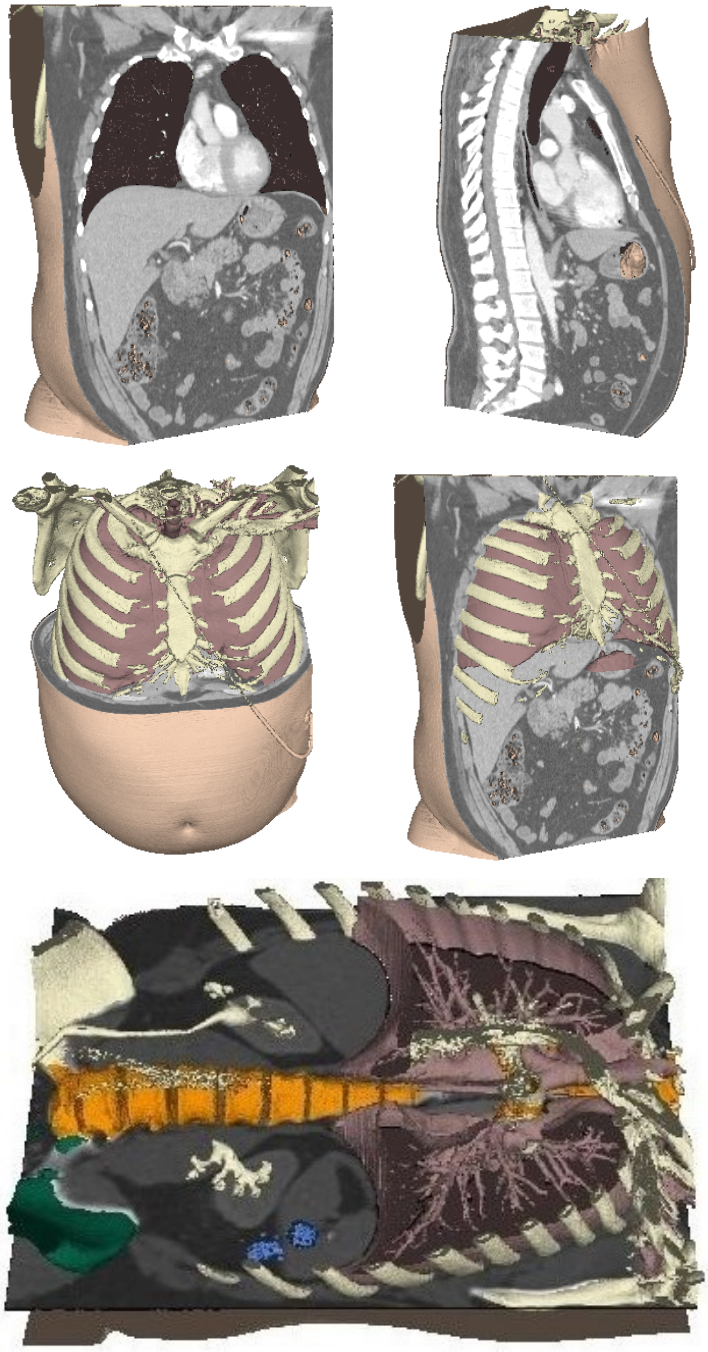


Figure 4.15: Some snapshots of the interaction in the client with different transfer functions. Images show planar sections with high-res information of the initial volume V, and three images show offset structures in front of the section plane.

Table 4.1 The Thorax model at resolution 512^3 with an initial size of 128MB, and the Head model at resolution $512 \times 512 \times 485$ with an initial size of 121MB.

Thorax

Tissues: Skin, Bones, Lungs

Resolution	**Grey Nodes**	***B***	***D***	**Stream**	**Client Storage**
		KB	KB	KB	KB
32	11489	0	34.5	100	100
64	51126	11.5	153.3	165	337.3
128	229203	51.1	687	738	1504.6
256	995610	229	2988	3217	6652
512	4123820	996	12372	13368	28308

Tissues: Skin, Bones

Resolution	**Grey Nodes**	***B***	***D***	**Stream**	**Client Storage**
		KB	KB	KB	KB
32	8816	0	26.4	92	92
64	38028	8.8	114	123	255
128	170394	170	510	548	1422
256	737342	737	2211	2381	6291
512	2988561	2989	8967	9704	26655

Head

Tissues: Skin, Bones

Resolution	**Grey Nodes**	***B***	***D***	**Stream**	**Client Storage**
		KB	KB	KB	KB
32	5641	0	16.8	81.3	81.3
64	29541	5.6	88.5	94.1	178.1
128	135964	29.5	405	434	877
256	577888	135	1731	1866	3891
512	2390093	577	7170	7747	16402

Table 4.2 The Head model at resolution $512x512x485$. Data transmission starting from V_{16}, V_{32}, V_{64}. The uncompressed model size is 121MB.

Tissues: Skin, Bones

Base Volume: V_{16}

Resolution	***B*** KB	***D*** KB	V_{16} **Size** KB	**Stream** KB
16	0	3.2	8.2	11.4
32	1.1	16.8	-	17.9
64	5.6	88.5	-	434
128	29.5	405	-	434
256	135	1731	-	1886
512	577	7170	-	7747

Base Volume: V_{32}

Resolution	***B*** KB	***D*** KB	V_{32} **Size** KB	**Stream** KB
32	0	16.8	64.5	81.3
64	5.6	88.5	-	94.1
128	29.5	405	-	434
256	135	1731	-	1866
512	577	7170	-	7747

Base Volume: V_{64}

Resolution	***B*** KB	***D*** KB	V_{64} **Size** KB	**Stream** KB
64	0	88.5	524.2	612.7
128	29.5	405	-	434
256	135	1731	-	1866
512	577	7170	-	7747

Table 4.3 Size of the data streamed when transmission starts from V_{16}, V_{32} or V_{64}.

	Stream		
Vector	**Total Octree**	**Total Data**	**Total**
V_{16}	760 KB	9411 KB	10171 KB
V_{32}	829 KB	9394 KB	10223 KB
V_{64}	1354 KB	9306 KB	10660 KB

Table 4.4 Comparison of the octree structure and data size among our technique and two previous approaches, for the Thorax model at 512^3 with a size of 134 MB.

	Stream	
Approaches	**Octree Structure**	**Data**
Crassin *et al.*2009[94]	5.2 MB	65 MB
Gobbetti *et al.*2012 [32]	31 MB	48 MB
Our approach	1.3 MB	16 MB

4.8 Conlusions

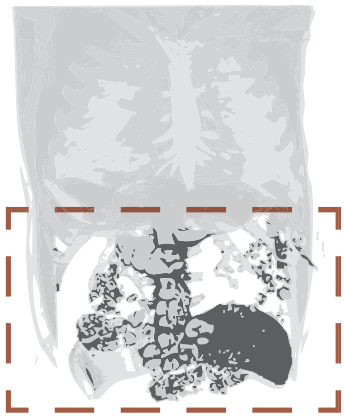
We have presented a transfer function-aware scheme for the remote interactive inspection of volume models in client-server architectures. Our approach aims at supporting progressive transmission, avoiding gradient computations in the client device and sending a very limited amount of information through the network. Our multi-resolution volume representation, Gradient Octrees can be progressively transmitted to the client in a compact way while achieving a minimum loss of visual quality as compared to state of the art ray-casting renderings. To offer users the possibility of fully understanding the volume data, the scheme supports planar volume sections which are shown with high-resolution volume information, besides interactive extrusion of specific structures. Interaction is autonomous and no further information is needed from the server, except when a new planar section is required.

Gradient Octrees are suitable for client-server architectures where non-low performance clients are used. Instead, the proposal that will be presented in Chapter 5, exploits the advantages of this technique in the design of a hybrid scheme that guarantees full interactive visualization of volume models in mobile devices.

CHAPTER 5

HYBRID ROI-BASED VISUALIZATION

This chapter describes a Hybrid volume representation scheme, that inherits advantages of the techniques presented in Chapters 3 and 4 as well as a benchmark to compare the proposed schemes with previous approaches.



Contents

5.1 Introduction

This chapter starts by discussing current relevant previous work in hybrid visualization approaches for volume models, Section 5.2. In Section 5.3, a hybrid approach that inherits the advantages of the algorithms presented in chapters 3 and 4 while keeping a good performance in terms of bandwidth requirements and storage needs in client devices is presented. Then, 5.4 presents a benchmark which compares the proposed schemes with three existing approaches: a classical volume visualization platform [96] [97], a scheme that sends the whole volume model to the client [19], and a thin-client application that performs volume rendering tasks in the server and sends 2D images to the client devices. These two last schemes correspond to cases A and B in Figures 2.1 and 2.2. Tests have been performed with two different volume data sets. We analyze the schemes regarding the compression rate, the amount of data being sent over the network, the rendering efficiency, the client requirements, the frame rate during interaction, and whether the techniques support multi-resolution and progressive transmission.

5.2 Hybrid Schemes for volume visualization

The majority of current volume rendering schemes are based on Direct Volume Rendering (DVR) algorithms. By applying this technique, images result from computing the contributions of all the voxels in a volumetric representation. On the other hand, Indirect Volume Rendering (IVR) is based on iso-surface extraction from the volume data.

The combination of DVR and IVR allows representing mixed models with surface and volume data. This merge conveys better understanding of models, providing better perception of the visualized structures, as well as the relationship among them. This fact increases the applicability in real environments such as medical applications.

Most of the presented techniques for volumetric hybrid visualization, use hybrid models that combine volume data and geometrical data together in the same volume dataset. Our proposal however, exploits the use of Gradient Octrees to reduce information while highlighting structures, also using volume data in the graphic pipeline.

Although combining both surfaces (polygons) and volume data is not the only hybrid scheme for volume rendering, it has been widely used by researchers. Several approaches can be identified to integrate surface and volume rendering. Some of them transform both types of data to a uniform representation. In some proposals a voxelization of data is applied, by performing a scan-conversion from geometric data into volumetric form [98] [99].

When the objective is to convert volumetric data into a polygonal representation, algorithms such as Marching Cubes are used to perform iso-surface extraction [100] [101]. These approaches require binary classification of volume data and may result in distorted structures in form of disconnected or missing features and surfaces.

By considering that ray casting can handle simultaneously different models by merging their contributions along the ray, many authors have developed ray casting-based hybrid visualization algorithms. Marc Levoy [102], was the first one in casting a set of polygons and volume data during the rendering algorithm. In the approach, samples of each type of data are at equally spaced intervals along the rays, while colors and opacities are composed in a depth sorted order. Following this concept other schemes have implemented ray casting-based hybrid rendering algorithms [103, 104, 105, 106]. The algorithm described in [107] optimizes the classical ray casting technique for direct volume rendering applied to both, volume and geometry data for volume visualization.

A more recent hybrid approach [108], uses 3D texturing to render opaque and translucent polygons together with semitransparent volumes. In order to preserve a correct depth composition, the technique renders alternatively the translucent polygons clipped at the boundary of the blocks defined by two consecutive slices and the volume data at each slice. Boada *et al.* [109] proposes a hybrid octree approach where the texture associated to the volume can be generated at different levels of resolution. This fact simplifies

the sorting of surface polygons between slices and achieves interactive hybrid volume visualizations.

Jainek *et al.* [110] also uses texture slices to implement a technique that interleaves illustrative meshes and volume rendering at interactive frame rates. To achieve this, they split the scene into different layers that are rendered individually, stored in intermediate textures and composed to generate the final image. With the objective of achieving a clear and concise output, the proposal includes some render styles such as silhouette rendering and ambient occlusion shading. These features increase hardware requirements and constrain the use of the approach to specific and sophisticate devices. Meyer *et al.* [111] developed a 3D texture-map-based framework to interactively render medical data sets derived from CT and MRI scanners as well as geometric data from CAD systems.

Other techniques employ independent Z-buffer processes for mixed surfaces and combine the image buffers according to their associate depths [112] [113].

Zakaria *et al.* [114] describes a new approach for extending the shear-warp rendering to simultaneously handle polygonal objects. The scheme integrates scan conversion with shear-warp rendering of run-length encoded volume data to obtain quality images in real time.

Other techniques advocate the use of hybrid region-based volume rendering, by applying different shading algorithms inside the volume model [115, 116, 117], or by implementing multiresolution region-based schemes [6] [109]. Luo *et al.* [118] developed a technique for focusing on a user-driven ROI while preserving context information. The approach uses a distance function to define the region of interest. This function controls voxel opacity, exploits silhouette enhancement and non-photorealistic shading.

Few techniques employ hybrid schemes to achieve remote visualization of volume data. Block based compression schemes, progressive transmission and multiresolution algorithms are needed to fullfill the requirements of these systems. An approximated solution to these problem was proposed by Sun k Yoo *et al.* [119]. The approach uses VRML (Virtual Reality Modeling Language) to represent compressed 3-dimensional images and to represent them through polygon reduction in web browsers.

In contrast to the previously mentioned schemes, we propose a hybrid framework that exploits the use of standard transfer functions as an alternative to compress volume dataset. Our scheme is hybrid based and transfer function-aware. It combines Wavelet preprocessed volumetric data to reduce information in non-interest regions, and highlighted segmented data in regions of interest (ROI), through a Gradient Octree scheme. Instead of a traditional iso-surface extraction stage, our technique applies the advantages of the algorithm presented in Chapter 4.

5.3 Hybrid, ROI-Based Approach

Let us assume that we are interested in inspecting a volume data model V which is too large in terms of network transmission and/or client storage facilities. The Wavelet compression algorithm presented in Chapter 3 supports block-based regions of interest (ROIs) while gradient octrees (Chapter 4) can be rendered with advanced illumination models and at a higher visual quality level. Both approaches have advantages and drawbacks:

- The TF-aware Wavelet compression scheme succeeds in sending to the clients a very limited amount of information in the areas outside the region of interest (ROI). The high quality volume information in the ROI is also compact, because the 3D texture is smaller and restricted to the blocks in the ROI area. Ray-casting visualization in the client uses two compact 3D textures which are suitable for many client devices. The main drawback of this approach, however, is twofold. It is restricted to a single Transfer Function and volume material, and it is not well suited for illumination computations that would require too many texture fetching.
- Gradient Octrees overcome these limitations by supporting multiple Transfer Functions and materials (up to 10 in our present implementation) and by precomputing gradients in the server. Gradient Octrees support advanced illumination models, thus achieving a higher visual quality level. However, they are not direct candidates for ROI-based visualization paradigms, as their low level volume representations present

a flat-face appearance with poor gradients. These representations at coarse tree levels are well suited for progressive transmission but they perform worse than similar-quality low-level Wavelet reconstructions.

A hybrid scheme which inherits the best of both approaches can also be devised by using the Gradient Octree representation for the ROI and a Wavelet representation for the context. In this case, apart from the volume model V, the user must supply a set of Transfer Functions $\{TF_k\}$ which identify the desired anatomical structures/materials and select one of them as a canonical transfer function. Without loss of generality, in what follows we will consider that the canonical TF is TF_1.

The server starts by computing the Gradient Octree $G(V)$ (Figure 5.1) from V and the set $\{TF_k\}$, and it also computes the quantified representation $W(V)$ of the Wavelet transform of V with TF_1. Given $G(V)$, $W(V)$ and a ROI, the corresponding hybrid model $H(V)$ is the union of $W(V)$ and $G_R(V)$, where $G_R(V)$ is the restriction of $G(V)$ to the ROI. Let us note as V_R the subvolume corresponding to the ROI portion, with $V_R \subset V$.

$G(V)$ includes three parts, as shown in Chapter 4: The base volume V_{32}, the octree structure $\boldsymbol{S}$ and the octree data $\boldsymbol{D}$, where $\boldsymbol{S}$ and $\boldsymbol{D}$ are lists of pointers and data arrays, with a pair (O_l, D_l) per octree level as defined in the previous Chapter. In other words, $\boldsymbol{S} = \{O_{32}, O_{64}, ..O_{r/2}\}$ and $\boldsymbol{D} = \{D_{32}, D_{64}, ..D_r\}$. Observe that the last array D_r represents the deepest octree level, its resolution r is the resolution of the initial volume V; whereas the last $\boldsymbol{S}$ array, $O_{r/2}$, represents the octree level previous to the deepest one, as pointers are unnecessary in the deepest octree level. In what follows, $r2$ will be used as a short name for $r/2$ and we will also write rl instead of r/l. The list of octree structure arrays B_l that is sent over the network will be noted as $\mathcal{S}_\mathcal{N} = \{B_{32}, B_{64}, ..B_{r2}\}$, and $W_N(V)$ will represent the transmitted information in the Wavelets case, after compression and quantization.

Although ROI-dependent $\boldsymbol{S}$ and $\mathcal{S}_\mathcal{N}$ could be defined, we have observed that the corresponding compression improvements (mainly in $\mathcal{S}_\mathcal{N}$) are negligible. In what follows, we will therefore consider that the hybrid model $H(V)$ is the union of $W(V)$ and $G_R(V)$, where $G_R(V)$ inherits the structure S of $G(V)$. However $G_R(V)$ and $G(V)$ are different regarding their $\boldsymbol{D}$ lists. Data arrays D_R of $G_R(V)$ are defined as the restriction of $\boldsymbol{D}$ to the ROI volume.

The $H(V)$ information can be sent from the server to the client (or clients) in two parts:

- The static information is sent only once, at the beginning of the interaction session. It consists on a low-resolution version $W_N(V)$ of the Wavelet-compressed volume (in our implementation and in most results presented in Section 5.4, we have used a Wavelet compression with two levels $w_l = 2$ of compression, $w_l = 2$), the $32 \times 32 \times 32$ pointers volume V_{32} of the Gradient Octree, the set $\mathcal{S}_\mathcal{N}$ of arrays which encode the $G(V)$ octree structure and the materials table of the Gradient Octree. The size of this last table is very small and will be omitted in the next paragraphs.
- The dynamic information is sent on demand (when a new ROI is defined) and consists on the subset D_R of the data arrays $\boldsymbol{D}$ of the Gradient Octree, as we know in advance that only voxels in the ROI will be retrieved.

In the client side, a low-resolution volume model V_W is reconstructed by de-quantization and $w_l = k$ Wavelet steps in each block, whereas the hierarchical structure of octree pointers (the set $\boldsymbol{S}$) is generated from the compact arrays in $\mathcal{S}_\mathcal{N}$. The only difference with the algorithm presented in Chapter 4 is that octree pointers in the O_k arrays must support direct access to dynamic and ROI-dependent subsets in D_R representing ROI 3D sub-volumes, as discussed below.

Let us first discuss the size of the static information that must be transmitted over the network for the hybrid approach $H(V)$. We use the notations defined in Section 1.3, writing the memory requirements of a model m in KBytes as $C(m)$, and the size of the transmitted information for this model m (also in KBytes) as $C_N(m)$. Assuming, for the sake of simplicity and without loss of generality, that V is a cubic volume of resolution r and that the initial density values are 1-Byte with a range between [0..255], we can obviously write,

$$C(V) = r^3 \tag{5.1}$$

And we also know that V_{32} has two-byte pointer elements,

$$C_N(V_{32}) = 2 \cdot (32^3) = 65 \ \ KBytes \tag{5.2}$$

If we use a 2-level Wavelet compression outside the ROI with $w_l = 2$, our lossy data reduction during quantization (see Section 3.5) ensures that,

$$C_N(W(V)) = C(W_N(V)) \ll \frac{1}{64} \cdot C(V) \tag{5.3}$$

Finally,

$$C_N(\boldsymbol{S}) = C(\mathcal{S}_\mathcal{N}) = \alpha.C(V) \tag{5.4}$$

being α the compactness of the transmission of the octree structure. From the experiments presented in Chapter 4 we can experimentally observe that α is between 0.005 and 0.01. Therefore, $C_N(\boldsymbol{S}) \lesssim 0.01 \cdot C(V)$. We use the symbol "$\lesssim$" to remark that this is an experimentally based inequality.

We can conclude that the total volume of static information (independent on the ROI) is bounded by a 2.5% of the initial volume size $C(V)$:

$$C_N(H_{Stat}(V)) = C_N(V_{32}) + C_N(W(V)) + C_N(\boldsymbol{S}) \lesssim 0.025 \cdot C(V) \tag{5.5}$$

Regarding the dynamic information in $H(V)$, let us first assume that we are only interested on gradient octree data at the deepest octree level r. Then let us consider two cases: sparse and compact D_R data.

(a) Sparse D_R data

A simple and efficient option is to translate the gradient octree O_k indexes to son data in $\boldsymbol{D}$ at the deepest octree level into triplets (i,j,k) of indexes to the gradients and materials of the voxels of the ROI sub-volume V_R. As defined, any element $O_{r2}[k]$ of the deepest O_{r2} array is a sub-array with structural information on its 8 sons, $O_{r2}[k][0]$ .. $O_{r2}[k][7]$. The elements $O_{r2}[k][m]$ can be either *Nil* or a certain triplet (i, j, k). In the first case, the corresponding leaf node will be considered transparent by the ray-casting algorithm, whereas in the second case, $O_{r2}[k][m]$ contains the indexes to the voxel of the maximum resolution volume that contains the desired data (this data includes material information and gradient, see Section 4.3). We

can provide direct access to data in the 3D volume in the ROI, at the cost of having a non-compact D_R data representation of the ROI volume: the compact D_r array of the gradient octrees has been translated onto a standard voxel representation D_R of the whole 3D ROI volume. D_R can be now represented as a sparse 3D array (represented as its 1D scan-line traversal). Storage requirements are usually rather limited, as shown in the next paragraphs.

In conclusion, the size $C(V_R)$ of the dynamic information in this case is obviously three times (the D_r array contains 3 Bytes per voxel) the number of ROI high-resolution voxels. In other words,

$$C_N(H_{Dynam1}(V)) = C(H_{Dynam1}(V)) = 3 \cdot C(V_R) \tag{5.6}$$

and

$$C_N(H(V)) = C_N(H_{Stat}(V)) + C_N(H_{Dynam1}(V)) \lesssim 0.025 \cdot C(V) + 3 \cdot C(V_R) \tag{5.7}$$

Observe that the octree structure information $\boldsymbol{S}$ in the client device is ROI-dependent, and that it must be reconstructed (in the client) from $\mathcal{S}_\mathcal{N}$ anytime the ROI is changed (see Figure 5.1). The bounding box information of the new ROI is used to traverse the $\boldsymbol{B}$ arrays in $\mathcal{S}_\mathcal{N}$ while generating the O_k arrays in $\boldsymbol{S}$ in a way that O_k contain triplets (i, j, k) indexing voxels withing the ROI data.

(b) Compact D_R data

There is a second option, regarding the dynamic information in $H(V)$. Instead of sending the complete ROI volume data as a 1D scan-line traversal of D_R, we can send a compact D_R array only containing those voxels with a non-*Nil* gradient value.

Let us define an octree traversal CTA algorithm which numerates only non-*Nil* voxels in V_R and that is shared by the server and the client: non-*Nil* voxels in V_R are assigned sequential Natural numbers while non-*Nil* voxels in V but outside V_R are simply skipped. An interesting observation is that any such CT Algorithm can produce a compact D_{R_r} array that can

afterwards be successfully accessed by the client, provided that this client has reconstructed $\boldsymbol{S}$ from $\mathcal{S}_{\mathcal{N}}$ through the same CTA. In this case as V_R is sparse, obviously:

$$C_N(H_{Dynam2}(V)) = C(H_{Dynam2}(V)) \ll 3 \cdot C(V_R) \tag{5.8}$$

and

$$C_N(H_{Dynam2}(V)) \ll C(H_{Dynam1}(V)) \tag{5.9}$$

Sending compact D_R arrays as dynamic information is better than sending the subvolume V_R, as:

However, sending less dynamic information comes at the cost of creating a new octree structure $\boldsymbol{S}$ from $\mathcal{S}_{\mathcal{N}}$ in the client at every ROI change during the user interaction, as $\boldsymbol{S}$ depends on the specific CTA being used, and CTA are ROI-dependent.

By considering that $C(H_{Dynam1}(V))$ is reasonable small (according to our experimental tests) we have finally chosen the first option with sparse D_R data in our implementation and in the discussion in the rest of the present Chapter. Investigation of the use compact D_R arrays are a potential avenue for future research.

Multi-resolution D_R data at upper octree levels could be considered and managed in a similar way, but in this case O_k arrays should have double sets of pointers to son nodes, as pointers to O_{2k} will obviously differ from pointers to D_R data in O_{2k}.

The deduced equations and bounds for $C_N(H(V))$ obviously depend on the size of the ROI. To show their real values in some cases, let us consider a cubic volume model V with a $256 \times 256 \times 256$ resolution, $r = 256$. The size of the uncompressed model is $C(V) = 16$ MBytes, being too high for smooth network transmission. The static information includes 4096 blocks of Wavelet coefficients which, at a two-level Wavelet compression, result on a maximum of $C_N(W(V)) = 262$ KBytes of information. The pointers volume of the gradient octree V_{32} requires a fix amount of 65 KBytes, whereas the S arrays encoding the tree structure can be successfully transmitted by using 170 KBytes. The size of the materials table is irrelevant, as already mentioned.

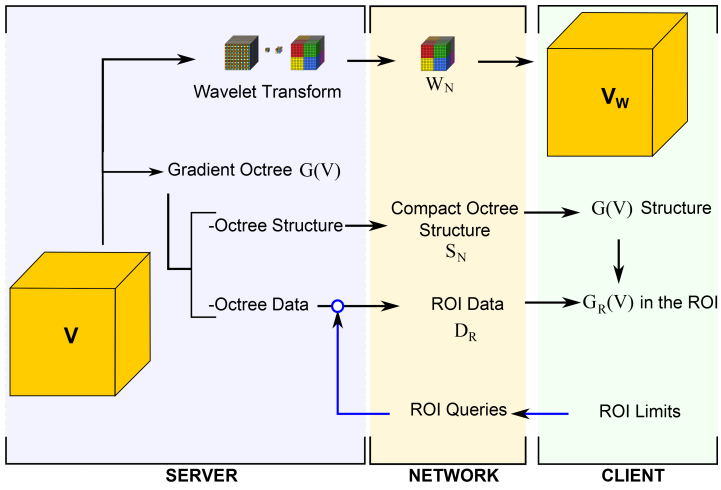


Figure 5.1: Overview of the Hybrid Scheme, Showing the preprocess in the server, the data transfer through the network and the data structures in the client device.

The amount of required static information that must be sent through the network is therefore $262 + 65 + 170 = 497$ KBytes, a 3% of the initial volume data information. The amount of dynamic information depends of the ROI size. For a ROI of $48 \times 48 \times 32$ voxels (as defined in Chapter 3), the total of dynamic information will be bounded by $48 \cdot 48 \cdot 32 \cdot 3 = 221$ KBytes, a 1 % of the initial volume data information. The client will be able to visualize the hybrid volume representation with a total of transmitted information (static+dynamic) of $497 + 221 = 718$ KBytes, a 4.5% of the initial data size.

In the case of a cubic volume model with a resolution of $512 \times 512 \times 512$ resolution, the size of the initial model V is 128 MBytes. The static information includes 32768 blocks of Wavelet coefficients which, at a two-level Wavelet compression, result on a maximum of $C_N(W(V)) = 2$ Mbytes of information. The pointers volume V_{32} of the gradient octree requires a fix amount of 65 KBytes, whereas S requires 1.36 MBytes. The amount of required static information that must be sent through the network is therefore 3.43 MBytes, a 2.6% of the initial volume data information. For a ROI also of $48 \times 48 \times 32$ voxels, the total of dynamic information is 221 KBytes, a 0.17% of the initial volume data information. The total of the transmitted information is 3.65 MBytes, less than a 3% of the initial data

size.

The main conclusion from the above figures is that the hybrid scheme is flexible enough to represent several materials and volume structures in the ROI area at a very limited static and dynamic information transmission cost, as discussed in Section 5.4.

5.3.1 Rendering of the Hybrid Scheme

The ray casting algorithm to render the Hybrid approach is similar to the one explained in Section 4.5 for the Gradient Octrees, where rays traverse a low resolution 3D texture representing the whole virtual volume (see Figure 5.2). For samples along the ray that do not belong to the ROI, the ray-casting uses density values from the low-resolution model V_W. Ray casting within the ROI is based on the octree addressing properties described in Section 4.5. The octree seach of any virtual voxel (i, j, k) is directly driven by the base-2 representation of i, j and k, as shown in Figure 5.2.

Samples in the ROI retrieve densities from a virtual volume with the same resolution R as V_R, instead of traversing V_R itself. After retrieving high-res data in D_r, materials and the gradient vector are decompressed on the fly in the GPU. The gradient face f, the two grid-coordinates of the gradient and the material index are decoded as in Section 4.5 for the Gradient Octrees model.

Figure 5.3 shows two snapshots of an interaction session with the hybrid scheme. The resolution of the volume models are $256 \times 256 \times 112$ (Skull) and 512^3 (Thorax), ROIs are $128 \times 128 \times 64$ and 256^3 blocks, respectively. The total size of the static information is 349 KBytes and the size of the dynamic information is 0.99 MBytes. The compression rate is 18 %, as the total size of the volume data is 7.34 MBytes.

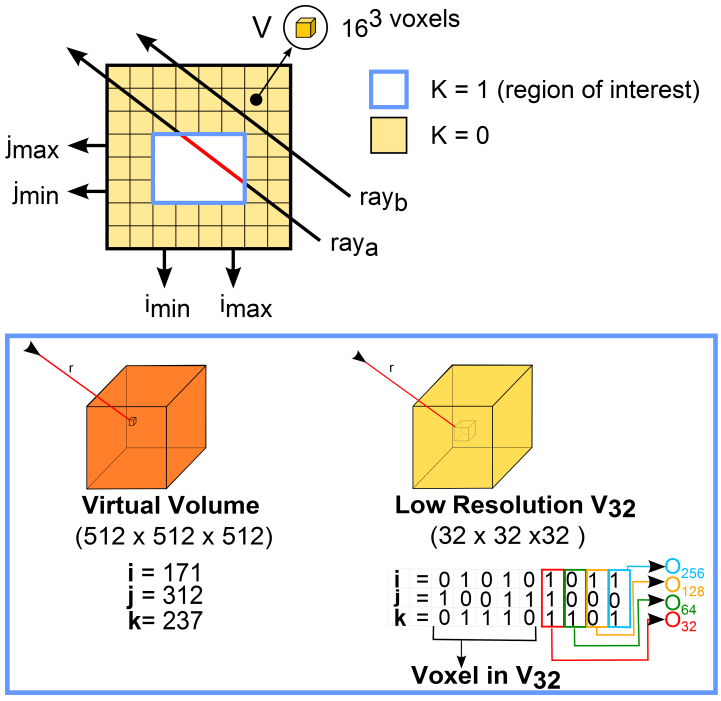


Figure 5.2: The block structure of the model, a region of interest (in white) and the specification of the BlocksId texture (shown in 2D, for clarification).

5.4 Comparison with Previous Schemes

This Section presents a comparison between the volume representation schemes proposed in this thesis and several previous approaches for Remote Volume Rendering. We have chosen three accessible applications to achieve this comparative study: The *VrMed Viewer* which is a classical volume visualization platform, the *Volume Viewer*, a scheme that sends the whole volume model to the client (case A, Figure 2.1) and the *VrMed-Thin Client* platform that performs volume rendering tasks in the server and sends 2D images to the client devices.

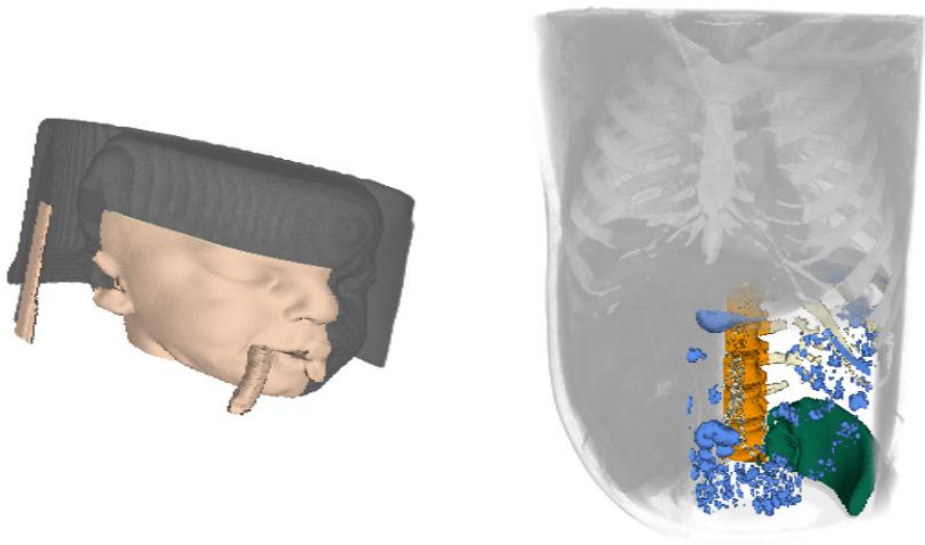


Figure 5.3: Hybrid Visualization. Snapshots of an interaction with two models in the server application. In (a), the Skull model after applying $w_l = 2$ in the low resolution region, while the ROI with a resolution of $128 \times 128 \times 64$ shows the skin. In (b), the Thorax model after applying $w_l = 2$ in the low resolution region while the ROI, with a resolution of $256 \times 256 \times 256$ is highlighting four different anatomical structures.

5.4.1 Previous Schemes

VrMed Viewer: Is a volume application, designed to achieve interactive visualization in PCs and Virtual Reality Systems [96] [97]. It has been developed by the MOVING research group from the UPC.

Volume Viewer: Is an Android based application, implemented to run on mobile devices as tablets and smart phones. It allows interactive visualization of models with a transfer function editor with easy handling, specially designed for small screens. Moreover, the software generates benchmarks that allow analyzing performance of moderns GPU by implementing three classical algorithms for volume rendering [19].

VrMed-Thin Client: The approach is based on [120]. It achieves remote visualization of volume models with basic user interaction tools in mobile

devices. A server running VRMed Viewer on Linux operative system, renders images which are sent to the client through the network. Clients generate control commands as OpenGL parameters which are sent to the server using a TPC/IP socket.

5.4.2 Comparative analysis

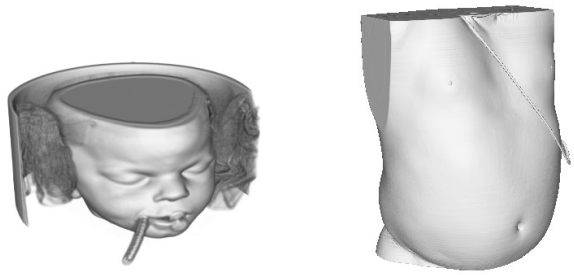


Figure 5.4: Models used during data analysis with the hybrid scheme. The skull model with a resolution of $256 \times 256 \times 112$ (left), and the thorax model with a resolution of 512^3 (right).

Figure 5.4 shows the two models that we have used for the performance analysis. The skull model (left) with a $256 \times 256 \times 112$ resolution and the thorax model of a 512^3 resolution. Density values are in the rang [0...255], hence each voxel is codified using only 1 byte. The total memory requirements of the volume models are 7.4 MBytes and 128 MBytes respectively.

Tables 5.1 and 5.2 show a comparison among the three remote visualization approaches proposed in this thesis (Wavelet-based scheme, Gradient Octrees and Hybrid-based approach) and the three platforms described in Section 5.4.1, using the previously described models. Table rows show for each scheme whether multi-resolution and progressive transmission is allowed and the compression rate achieved for each case. Both Tables also show the size of the transmitted data through the network and the client requirements to performs volume rendering followed by an estimation of the average frame rates in two cases: rendering in the PC server and rendering in the mobile device.

The server application runs on PC with 6 GB of RAM, Intel Core 2 Duo at 3.16 GHz and the PC client is a 4 GB of RAM , Intel Core 2 Duo at 3.06 GHz and Nvidia GeForce GTX z80. Client tests were also performed on the HTC One smartphone whith a screen resolution of 1080 x 1920 pixels 2 GB RAM and an Adreno 320 Graphics processor.

The size of the ROI in Tables 5.1 and 5.2 is $128 \times 128 \times 64$ voxels. A ROI-based visualization has been considered in the Wavelets-based scheme and in the Hybrid approach, while in the rest of columns, the whole volume V has been rendered without ROI. Zoom has been adjusted in a way that the total amount of rendered ROI pixels in the application viewport is a 25% of the total of viewport pixels. The amount of ROI pixels in the viewport is relevant, as it measures the total amount of required high-quality casted rays during ray-casting rendering. Compression rates correspond to the amount of data sent over the network with respect to the original volume size. The compression rate for the hybrid approach has been computed by adding the compression achieved in the low resolution ($w_l = 2$) representation of the model, plus the compression rate represented by the Gradient Octree in the region inside the selected ROI.

In contrast to the previous schemes, our proposed techniques allow multi-resolution rendering with progressive transmission of volume data. The average frame rate in the mobile device is between 8 and 16 *fps* as shown in the tables 5.1 and 5.2. As already discussed, the Wavelet-based rendering requires a client GPU being able to manage 3D textures.

In case of the Gradient Octree approach, data includes the octree structure plus material and gradient information at its deepest, maximum resolution level. In the Hybrid scheme, the *information over the network* represent both the necessary data to reconstruct two levels of Wavelet compression for the low resolution model and the nodes representing the Gradient Octree leaves in the selected region of interest (ROI). This approach also requires a client GPU being able to manage 3D textures for rendering. The compression rate in this case is between 20% and 22%.

The Hybrid approach as presented in Section 5.3 results in a compression rate which is between 4% and 18%, with an average frame rate in the mobile device between 8 and 16 *fps* when $w_l = 2$. It also requires a client GPU being able to manage 3D textures.

The *VrMed viewer* is presented for comparison purposes. Some of the parameters in the Tables do not apply to this case, as *VrMed* is a stand-alone application without network transmission. The average frame rates, 48 and 20 *fps*, are obviously higher than those in the previous cases but these figures show that our proposed approaches are performing within reasonable efficiency limits.

The *Thin Client* based approach sends a maximum of 0.18 MB of data through the network per frame during an interactive session with a single client (of course, the total amount of transmitted data depends on the number of interaction frames). This is due to the fact that the technique requires the transmission of renderized images from the server when the user interacts with the model in the client side. This fact makes this scheme a full-time network dependent, with frame rates which decrease in network congestion cases. We have observed that our architecture becomes useless when the number of clients is above 8. On the other side, this scheme does not require sophisticated client GPUs, as clients must only decompress and show pre-rendered images. This can be an advantage for basic client devices, but result on a under-utilization of client GPUs in the case of most present devices.

Figure 5.5: Hybrid multiresolution visualization of the skull model. Skin visualization in both low resolution and ROI (left). Semi-transparent skin in ROI (middle). Bone visualization in both, low resolution and ROI.

Comparing *Thin clients* to Wavelets, Gradient Octrees and the Hybrid approach, we can define the break-even interaction period as the number of frames required to have an equivalent amount of information sent over the network. In the case of the skull model in Table 5.1, this break-even is 11 frames for Wavelets, 21 frames for Gradient Octrees and 11 frames for the Hybrid approach. In the case of the thorax model in Table 5.2, the break-even is 51 frames for Wavelets, 79 frames for Gradient Octrees and 17

frames for the Hybrid approach. By considering the number of frames per second in each case, we can conclude that the information we are sending is equivalent to the total information sent by the Thin Client approach during an interaction period in between 1 and 10 seconds. In the case of the Hybrid approach, break-evens are 11 frames and 17 frames, meaning this Hybrid scheme as proposed in Section 5.3 will outperform Thin Clients in interaction sessions longer than 17 frames. This is due to the fact that break-evens are computed as the ratio between the size of the compressed model as sent over the network in our approaches and the size of a single Thin client frame image.

The *Volume Viewer* approach as presented in the last column of both Tables does not require sophisticated client GPUs, as clients are rendering stacks of 2D textures. Frame rates in the client are reasonable. The main drawback in this case, however, is the amount of information being sent over the network, which makes it unusable in the case of large volume models.

Some snapshots of the interaction with the thorax model using the Hybrid approach are shown in Figure 5.6. In all cases two Wavelet reconstruction steps ($wl = 2$) have also been applied without lighting computation, in the low resolution area: ROI showing ribs and lungs (a), internal gases and lungs (b) and Skin, ribs and lungs (d). The snapshot in (e) shows the alveoli in the ROI, magnified in (f) by interacting with zoom and a section plane. In these cases Wavelets are precomputed after applying to the model a TF covering all structures in the low resolution area. The snapshot in (c), shows a TF for bones visualization in both, the low resolution area, and the ROI. Image in (d) shows a zoom-in of (c) for showing up the quality of the hybrid model.

The asterisks in the *Thin client* column in Tables 5.1 and 5.2 mean that data sizes are per frame sizes. The compression rates obviously depend on the number of transmitted frames.

The Hybrid approach is specially well suited in the case of large models. The comparison between Tables 5.1 and 5.2 show that this is a scalable scheme, with compression rates that decrease when the size of the volume model increases. The corresponding frame rates are larger than in the case of Gradient Octrees, being of the same order of magnitude than Thin Clients.

Table 5.1 Comparison among the approaches presented in this paper and some previous schemes for remote visualization of volume models. Study of the Skull model with a resolution of $256 \times 256 \times 112$ and 7.3 MB of uncompressed size and a TF showing bone and skin. The value of C_N in the case of the Thin Client application corresponds to the size of the transmitted data per frame. The Marks in the two first rows, indicate whether the described functionalities are supported (✓) or not (×) by the presented techniques.

	Wavelets Sheme [6]	**Gradient Octrees** [7]	**Hybrid Approach** [8]	**VrMed Viewer** [96]	**VrMed Thin Client** [120]	**Volume Viewer** [19]
Multiresolution	✓	✓	✓	×	×	×
Progressive Transmission	✓	✓	✓	×	×	×
Compression rate (%)	32	22	18	-	*	100
C_N(MB)	1.91	3.8	2.04	-	0.18*	7.3
Client requirements	3D Tex	3D Tex	3D Tex	-	Image	Stack
Frame rates (PC version)	52	24.12	46.53	48.24	48.24	-
Frame rates (mobile)	24.2	-	16.32	-	20.23	17.0

Table 5.2 Comparison among the approaches presented in this paper and some previous schemes for remote visualization of volume models. Study of the Thorax model with a resolution of 512^3 and 128 MB of uncompressed size and a TF showing bone and skin. The value of C_N in the case of the Thin Client application corresponds to the size of the transmitted data per frame. The Volume Viewer does not support this Thorax model in the mobile device due to its large model size. The Marks in the two first rows, indicate whether the described functionalities are supported (✓) or not (✕) by the presented techniques.

	Wavelets Sheme [6]	**Gradient Octrees** [7]	**Hybrid Approach** [8]	**VrMed Viewer** [96]	**VrMed Thin Client** [120]	**Volume Viewer** [19]
Multiresolution	✓	✓	✓	✕	✕	✕
Progressive Transmission	✓	✓	✓	✕	✕	✕
Compression rate (%)	21	20	4.2	-	$*$	100
C_N(MB)	16.4	27	5.3	-	0.32*	128
Client requirements	3D Tex	3D Tex	3D Tex	-	Image	Stack
Frame rates (PC version)	14	10.31	18.03	20.34	20.34	-
Frame rates (mobile)	13.20	-	8.07	-	20.23	-

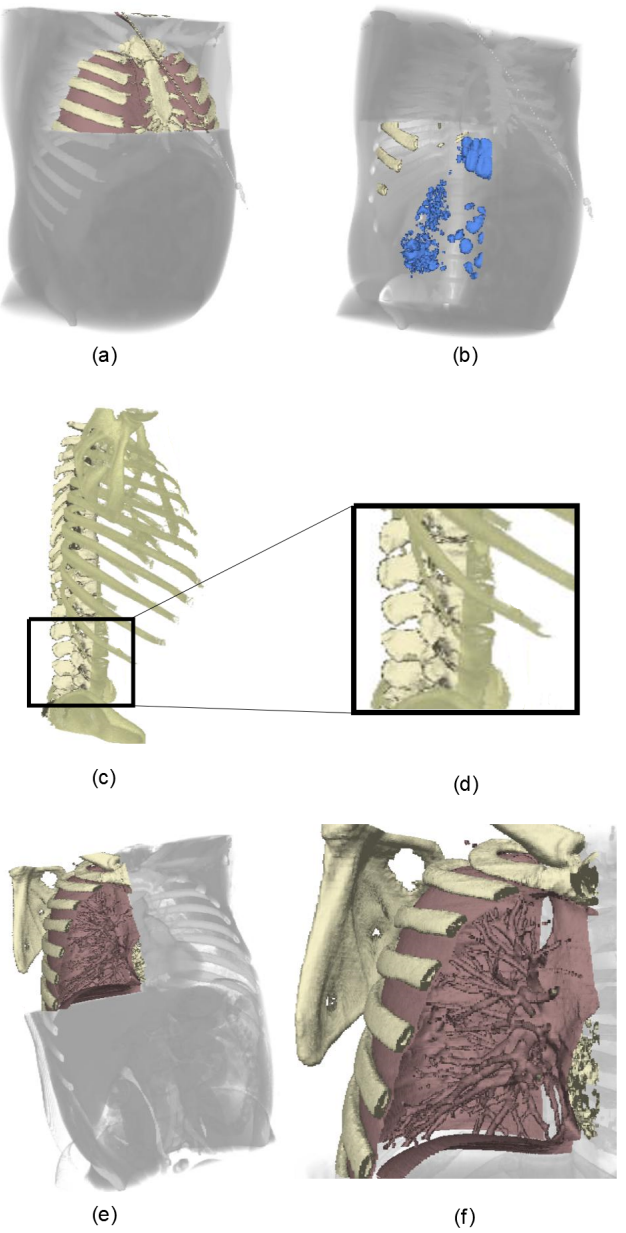


Figure 5.6: Hybrid Visualization. Interaction with the Thorax model. ROI size: $416 \times 224 \times 224$ (a), $256 \times 160 \times 256$ (b), $96 \times 128 \times 480$ (c), and $256 \times 512 \times 256$ (d), (e) and (f). Images in (d) and (f) Show zooms of local regions in (c) and (e).

5.5 Conclusions

We have proposed a Hybrid volume representation that inherits the advantages of the models presented in Chapters 3 and 4 while keeping a good performance in terms of bandwidth requirements and storage needs in client devices. Information over the network is classified into static information (being only set once) and dynamic information. Dynamic information must be re-sent anytime the ROI is redefined by the user. The complexity (memory and data transmission requirements) of the static and dynamic information has been analyzed, and two different dynamic information options have been discussed and compared. The main conclusion is that the hybrid scheme is flexible enough to represent several materials and volume structures in the ROI area at a very limited static and dynamic information transmission cost.

We have also presented a benchmark to compare the proposed schemes with three existing approaches: a classical volume visualization platform, a scheme that sends the whole volume model to the client and a thin-client application that performs volume rendering tasks in the server and sends 2D images to the client devices. Tests have been performed with two different volume data models. The compression rate, the amount of data being sent over the network, the rendering efficiency, the client requirements, the frame rate during interaction, and the support of multi-resolution and progressive transmission have been compared and discussed.

The break-even interaction period is between 11 frames and 79 frames in the presented examples. This is between one and 10 interaction seconds in all cases. In the case of the Hybrid approach, break-evens are 11 frames and 17 frames, meaning this Hybrid scheme as proposed in Section 5.3 will outperform Thin Clients in interaction cases longer than around 20 frames.

The Hybrid approach has been proved to be specially well suited in the case of large models. The presented experimental Tables show that the Hybrid approach is a scalable scheme, with compression rates that decrease when the size of the volume model increases. Corresponding frame rates are larger

than in the case of Gradient Octrees, being of the same order of magnitude than Thin Clients.

CHAPTER 6

CONCLUSIONS

This chapter presents the conclusions and the future work of the thesis.

6.1 Conclusions

The main objective of this thesis has been to enrich user experience on interactive visualization of volumetric medical models in low performance devices, by studying new transfer-function aware compression/decompression mechanisms adapted to transmission, reconstruction and visualization in those devices. Several schemes have been proposed to exploit the use of transfer functions to enhance the volume compression during its transmission to mobile devices. As far as we know, this possibility has not been considered by any of the described approaches in the previous work.

The study of client-server architectures for volume visualization is nowadays a wide area of research for computer graphics scientists. We have made a comparative analysis of some relevant approaches in Chapter 2. Although there have been interesting practical solutions in the last years, a lot of open problems remain for further investigation. In case where no client-server architectures are implemented, compact data structures can be used to transfer data between CPU and GPU and also to efficiently implement out-of-core solutions. Sending the whole volume model in a compressed way to the client device is still a challenging problem. This ensures full interaction facilities in the client without further server-client communication. Many proposals achieve good results, but small storage client capacity and network latency during real time visualization decrease interaction performance. As an alternative, some techniques transmit images through the network, but this falls in the lack of volume information in the client side, decreasing performance when new data is demanded and sometimes affecting image quality. Some approaches convert volume data into compact data structures ready to be gradually sent from servers to clients. Compressed volume rendering is a good solution to render models whose size exceeds current hardware capabilities, but performance during decompression is still a field requiring of improvements. Although some proposals decompress data in the client CPUs, the current trend is to move decompression to the GPU. In this case, data gets compressed to the client GPU, resulting in less memory consumption and better exploiting available bandwidth. Our main conclusions from Chapter 2 are:

- Low-end mobile devices are being preferred due to easy maintenance and portability but their hardware limitations demand a further revision of client-server algorithms to optimally increase the interactivity while visualizing volume models. Small storage client capacity and network latency during real time visualization decrease interaction performance and require novel and specific solutions.
- Current solutions are not fully exploiting the fact that medical doctors usually work with a restricted set of standard transfer functions (TFs). This fact can be a new tool to further compress volume datasets.
- Efficient solutions should take advantage of novel hardware architectures in current mobile GPUs, like 3D texture handling.

In Chapter 3, a new scheme for interactive volume in small mobile devices has been presented, having a number of novel features. Clients receive the whole model, compressed with a TF-aware scheme which is able to achieve high compression rates. Decompression in the CPU of the client is efficient and can be performed on the fly during interaction. Our compression scheme is local, block-based and uses a Haar-Wavelet approach together with perceptual-based quantization. The consequence of this particular choice is that the size of the 3D textures in the mobile device is significantly smaller. Inspection of complex volume models with maximum level of detail in selected regions of interest becomes feasible: our scheme supports model sizes that otherwise could not be handled. The ray-casting algorithm in the GPU of the client is adapted to the block structure, being able to simultaneously deal with regions having different levels of detail. Conclusions of Chapter 3 can be summarized as:

- A new Wavelet-based, TF-aware compression scheme has been proposed. It supports inspection of complex volume models with maximum level of detail in selected regions of interest (ROIs). It uses a GPU-based, ROI-aware ray-casting rendering algorithm in the client, with a limited amount of information being sent over the network and stored in the clients.
- One of the present limitations of this Wavelet-based approach is that the preprocess in the server depends on the transfer function TF. Modifications of this TF require a new transmission of the model to

the client. This aspect may not be critical, as the standard workflow in medical diagnose is usually based on a small number of pre-established transfer functions that enhance particular structures. Anyway, the new proposals in Chapters 4 and 5 have been designed with the objective to overcome this limitation.

Chapter 4 presents a second Transfer Function-aware scheme for the remote interactive inspection of volume models in client-server architectures. This approach aims at supporting progressive transmission, avoiding gradient computations in the client device and sending a very limited amount of information through the network. Our multiresolution volume representation (Gradient Octrees) can be transmitted to the client in a compact way while achieving a minimum loss of visual quality as compared to state of the art ray-casting renderings. To offer users the possibility of fully understanding the volume data, our scheme supports planar volume sections which are shown with high-resolution volume information, besides interactive extrusion of specific structures. Interaction is autonomous and no further information is needed from the server, except when a new planar section is required. Our main conclusions from this Chapter are:

- Gradient Octrees can efficiently encode volume datasets. They support high-quality visualizations with Transfer Functions from a predefined TFs set. In our present implementation, Transfer Function sets can encode up to ten different volume materials.
- Gradient Octrees are multiresolution, supporting progressive transmission and avoiding gradient computations in the client device. They encode precomputed gradients to save costly computations in the client, and support illumination-based ray-casting without extra computations in the client GPU. The proposed scheme presents a minimum loss of visual quality as compared to state of the art ray-casting renderings.
- Gradient Octrees send a very limited amount of information through the network. The octree structure is compacted into a very small volume array and a set of texture-coded arrays, with only one Bit per octree node.
- The proposed scheme supports planar volume sections which are shown with high-resolution volume information, besides interactive extrusion of specific structures.

In Chapter 5, we have proposed a Hybrid approach that inherits the advantages of the algorithms presented in Chapters 3 and 4 while keeping a good performance in terms of bandwidth requirements and storage needs in client devices. Our Hybrid approach uses Gradient Octrees with high-resolution rendering in user-defined regions of interest (ROIs) while visualizing the rest of the volume in low resolution with the approach presented in Chapter 3. Information over the network is classified into static information (being only set once) and dynamic information. Dynamic information must be re-sent anytime the ROI is redefined by the user. The complexity (memory and data transmission requirements) of the static and dynamic information is analyzed, and two different dynamic information options have been discussed and compared. The main conclusion is that the hybrid scheme is flexible enough to represent several materials and volume structures in the ROI area at a very limited static and dynamic information transmission cost. We have also presented a benchmark to compare the proposed schemes with three existing approaches: a classical volume visualization platform, a scheme that sends the whole volume model to the client and a thin-client application that performs volume rendering tasks in the server and sends 2D images to the client devices. Tests have been performed with two different volume data models. The compression rate, the amount of data being sent over the network, the rendering efficiency, the client requirements, the frame rate during interaction, and the support of multiresolution and progressive transmission have been compared and discussed. The break-even interaction period (as defined in 5) is between 11 and 79 frames in the presented examples. This means between 1 and 10 interaction seconds in all cases. In the case of the Hybrid approach, break-evens are 11 frames and 17 frames, meaning that this Hybrid scheme as proposed in Section 5.3 will outperform Thin Clients in interaction sessions longer than around 20 frames. The Hybrid approach has been proved to be specially well suited in the case of large models. The presented experimental Tables show that the Hybrid approach is a scalable scheme, with compression rates that decrease when the size of the volume model increases. Corresponding frame rates are larger than in the case of Gradient Octrees. Conclusions of Chapter 5 can be summarized as:

- A Hybrid approach that inherits the advantages of the algorithms presented in Chapters 3 and 4 while keeping a good performance in terms of bandwidth requirements and storage needs in client devices has been proposed. The scheme is flexible enough to represent several

materials and volume structures in the ROI area at high resolution and very limited information transmission cost.

- A benchmark has been presented to compare the proposed schemes with three existing approaches. Tests have been performed with two different volume data models.
- The Hybrid approach has been proved to be specially well suited in the case of large models. Experimental results show that this Hybrid approach is a scalable scheme, with compression rates that decrease when the size of the volume model increases.

In conclusion, this thesis has proposed and discussed several transfer-function aware compression/decompression mechanisms for volume model transmission, reconstruction and visualization in small client devices. We consider that they may enrich the user experience during the inspection of volume medical models in these low performance devices.

6.2 Future Work

Some aspects of the presented work can be further investigated and extended:

- Current mobile graphics boards are notably less powerful than the complex desktop graphic devices. However during the last years, an important improvement of these capabilities have been appeared on smartphones and tablets. For the wavelet-based approach, a future study of block-aware rendering acceleration algorithms could be improved through paralelizable implementations in GPU like CUDA.
- Related to the Gradient Octrees based approach, we would like to investigate local algorithms for octree-based gradient filtering, by taking into account gradient information in the parent node. Besides this, a good avenue for further research is to optimize thresholds during the generation of the edge volume ($\mathcal{S}_R^E$) generation to minimize the visual error between the ray-casting renderings of V and $\mathcal{S}_R^E$ for any viewing direction and for any TF.

- The proposed improvements for the previous schemes could also upgrade the Hybrid ROI-Based scheme. However, the study of ROI-dependent inspection of Coherent-Traversal Trees could also be a starting point for future research in this field. We would like to investigate Coherent-Traversal Trees with compact data representations $\boldsymbol{D}$ trying to reach an optimized compromise between storage requirements in the client GPU and the number of texture fetching during the ray casting algorithm.

BIBLIOGRAPHY

[1] Goretti Echegaray, Imanol Herrera, Iker Aguinaga, Carlos Buchart, and Diego Borro. A brain surgery simulator. *IEEE Computer Graphics and Applications*, 34(3):12–18, 2014.

[2] Giuseppe Turini, Nico Pietroni, Giuseppe Megali, Fabio Ganovelli, Andrea Pietrabissa, and Franco Mosca. New techniques for computer-based simulation in surgical training. *International Journal of Biomedical Engineering and Technology (IJBET)*, 5(4):303–316, 2011.

[3] Odysseus Argy and Michael Caputo. Introduction to telemedicine. In *Information Technology for the Practicing Physician*, pages 227–233. 2001.

[4] Richard Wootton. Telemedicine: a cautious welcome. *Bmj*, 313(7069):1375–1377, 1996.

[5] Douglas A Perednia and Ace Allen. Telemedicine technology and clinical applications. *Jama*, 273(6):483–488, 1995.

[6] Lázaro Campoalegre, Pere Brunet, and Isabel Navazo. Interactive visualization of medical volume models in mobile devices. *Personal Ubiquitous Computing*, 17(7):1503–1514, 2013.

[7] Lázaro Campoalegre, Pere Brunet, and Isabel Navazo. Gradient octrees: A new scheme for remote interactive exploration of volume models. *Proceedings of the CAD/Graphics 2013*, 2013.

[8] Lázaro Campoalegre, Isabel Navazo, and Pere Brunet. Hybrid, roi-based inspection of medical volume models in mobile devices. In *Spanish Conference on Computer Graphics*, 2014.

[9] Keir Davis, John W Turner, and Nathan Yocom. Client-server architecture. In *The Definitive Guide to Linux Network Programming*, pages 99–135. 2004.

[10] Miodrag M Kekic, Grace N Lu, and Eloise H Carlton. Client-server computer network management architecture, 2003.

[11] Jouni Smed, Timo Kaukoranta, and Harri Hakonen. Aspects of networking in multiplayer computer games. *The Electronic Library*, 20(2):87–97, 2002.

[12] Greg Humphreys, Mike Houston, Ren Ng, Randall Frank, Sean Ahern, Peter D Kirchner, and James T Klosowski. Chromium: a stream-processing framework for interactive rendering on clusters. In *ACM Transactions on Graphics*, volume 21, pages 693–702, 2002.

[13] Yonggang Huang, ChunYang Hu, Yongwang Zhao, and Dianfu Ma. Web-based remote collaboration over medical image using web services. In *Global Information Infrastructure Symposium*, pages 1–8, 2009.

[14] M. Balpande and U. Shrawankar. Medical image fusion techniques for remote surgery. In *IEEE India Conference (INDICON)*, pages 1–6, Dec 2013.

[15] Wang Xiao-min, Wang Peng-cheng, and Xie Jin-dong. A new design of remote diagnosis system for medical images. In *International Conference on Management and Service Science*, pages 1–3, 2009.

[16] Steven P Callahan, Louis Bavoil, Valerio Pascucci, and Claudio T Silva. Progressive volume rendering of large unstructured grids. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1307–1314, 2006.

[17] Manuel Moser and Daniel Weiskopf. Interactive volume rendering on mobile devices. In *Vision, Modeling, and Visualization VMV*, volume 8, pages 217–226, 2008.

[18] Movania Muhammad Mobeen and Lin Feng. Ubiquitous medical volume rendering on mobile devices. In *International Conference on Information Society (i-Society)*, pages 93–98. IEEE, 2012.

[19] Marcos Balsa Rodríguez and Pere Pau Vázquez Alcocer. Practical volume rendering in mobile devices. In *Advances in Visual Computing*, pages 708–718. 2012.

[20] Carlos Antonio Andújar Gran, Jonás Martínez Bayona, et al. Locally-adaptive texture compression. *Proceedings of the Spanish Conference in Computer Graphics*, 2009.

[21] D. Taubman. High performance scalable image compression with ebcot. *IEEE Transactions on Image Processing*, 9(7):1158–1170, 2000.

[22] P. Simoens, P. Praet, B. Vankeirsbilck, J. De Wachter, L. Deboosere, F. De Turck, B. Dhoedt, and P. Demeester. Design and implementation of a hybrid remote display protocol to optimize multimedia experience on thin client devices. In *Telecommunication Networks and Applications Conference*, pages 391–396, Dec 2008.

[23] Klaus Engel and Thomas Ertl. Texture-based volume visualization for multiple users on the world wide web. In *Virtual Environments*, pages 115–124. 1999.

[24] Klaus Engel, Thomas Ertl, Peter Hastreiter, Bernd Tomandl, and K Eberhardt. Combining local and remote visualization techniques for interactive volume rendering in medical applications. In *Proceedings of the conference on Visualization'00*, pages 449–452, 2000.

[25] Xiaojun Qi and John M Tyler. A progressive transmission capable diagnostically lossless compression scheme for 3d medical image sets. *Information Sciences*, 175(3):217–243, 2005.

[26] Liviu Constantinescu, Jinman Kim, and Dagan Feng. Integration of interactive biomedical image data visualisation to internet-based personal health records for handheld devices. In *International Conference on e-Health Networking, Applications and Services*, pages 79–83, 2009.

[27] Seok-Jae Jeong and Arie E Kaufman. Interactive wireless virtual colonoscopy. *The Visual Computer*, 23(8):545–557, 2007.

[28] Anna Tikhonova, Carlos D Correa, and Kwan-Liu Ma. Explorable images for visualizing volume data. In *IEEE Pacific Visualization Symposium*, pages 177–184, 2010.

[29] Wes Bethel. Visualization dot com. *IEEE Computer Graphics and Applications*, 20(3):17–20, 2000.

[30] Lars Lippert, Markus H Gross, and Christian Kurmann. Compression domain volume rendering for distributed environments. In *Computer Graphics Forum*, volume 16, pages C95–C107, 1997.

[31] Imma Boada and Isabel Navazo. Optimal exploitation of client texture hardware capabilities on a client-server remote visualization framework. In *Computational Science and Its Applications ICCSA*, pages 468–477. 2003.

[32] Enrico Gobbetti, José Antonio Iglesias Guitián, and Fabio Marton. Covra: A compression-domain output-sensitive volume rendering architecture based on a sparse representation of voxel blocks. In *Computer Graphics Forum*, volume 31, pages 1315–1324, 2012.

[33] Fabrizio Lamberti, Claudio Zunino, Andrea Sanna, Antonino Fiume, and Marco Maniezzo. An accelerated remote graphics architecture for pdas. In *Proceedings of the International Conference on 3D Web Technology*, pages 55–ff, 2003.

[34] Hariharan G Lalgudi, Michael W Marcellin, Ali Bilgin, Han Oh, and Mariappan S Nadar. View compensated compression of volume rendered images for remote visualization. *IEEE Transactions on Image Processing*, 18(7):1501–1511, 2009.

[35] Chuan-kai Yang. Integration of volume visualization and compression: A survey. *CiteSeer*, 2000.

[36] Marcos Balsa Rodriguez, Enrico Gobbetti, José Antonio Iglesias Guitián, Maxim Makhinya, Fabio Marton, Renato Pajarola, and Susanne Suter. A survey of compressed gpu-based direct volume rendering. In *Eurographics State of the Art Report*, 2013.

[37] Josien P.W Pluim, J.B Antoine Maintz, and Max A Viergever. Mutual-information-based registration of medical images: a survey. *IEEE Transactions on Medical Imaging*, 22(8):986–1004, 2003.

[38] Shigeru Muraki. Approximation and rendering of volume data using wavelet transforms. *Proceedings of the Conference on Visualization*, pages 21–28, 1992.

[39] Shigeru Muraki. Volume data and wavelet transforms. *IEEE Computer Graphics and Applications*, 13(4):50–56, 1993.

[40] Insung Ihm and Sanghun Park. Wavelet-based 3D compression scheme for interactive visualization of very large volume data. *Computer Graphics Forum*, 18:3–15, 1999.

[41] Stefan Guthe, Michael Wand, Julius Gonser, and Wolfgang Straßer. Interactive rendering of large volume data sets. In *IEE Visualization*, pages 53–60. IEEE, 2002.

[42] Ky Giang Nguyen and Dietmar Saupe. Rapid high quality compression of volume data for visualization. In *Computer Graphics Forum*, volume 20, pages 49–57, 2001.

[43] Flemming Friche Rodler. Wavelet based 3d compression with fast random access for very large volume data. In *Proceedings Of the Pacific Conference on Computer Graphics and Applications*, pages 108–117, 1999.

[44] R.M. Gray. Vector quantization. *ASSP Magazine, IEEE*, 1(2):4–29, April 1984.

[45] Paul Ning and Lambertus Hesselink. Vector quantization for volume rendering. In *Proceedings of the Workshop on Volume Visualization*, pages 69–74, 1992.

[46] Paul Ning and Lambertus Hesselink. Fast volume rendering of compressed data. In *Proceedings of the IEEE Conference on Visualization*, pages 11–18, 1993.

[47] Jens Schneider and Rüdiger Westermann. Compression domain volume rendering. In *IEEE Visualization*, pages 293–300, 2003.

[48] Eric B Lum, Kwan Liu Ma, and John Clyne. Texture hardware assisted rendering of time-varying volume data. In *Proceedings of the Conference on Visualization*, pages 263–270, 2001.

[49] Markus Hadwiger, Joe M. Kniss, Christof Rezk-salama, Daniel Weiskopf, and Klaus Engel. *Real-time Volume Graphics*. A. K. Peters, Ltd., 2006.

[50] Magnus Strengert, Marcelo Magallon, Daniel Weiskopf, Stefan Guthe, and Thomas Ertl. Hierarchical visualization and compression of large volume datasets using GPU clusters. *Parallel Graphics and Visualization*, D:41–48, 2004.

[51] Eric LaMar, Bernd Hamann, and Kenneth I. Joy. Multiresolution techniques for interactive texture-based volume visualization. In *IEEE Visualization*, pages 355–361, 1999.

[52] Imma Boada, Isabel Navazo, and Roberto Scopigno. Multiresolution volume visualization with a texture-based octree. *The Visual Computer*, 17(3):185–197, 2001.

[53] Manfred Weiler, Rüdiger Westermann, Charles D. Hansen, Kurt Zimmerman, and Thomas Ertl. Level of detail volume rendering via 3D textures. In *Volvis*, pages 7–13, 2000.

[54] Mohammad H Ghavamnia and Xue D Yang. Direct rendering of laplacian pyramid compressed volume data. In *Proceedings of the conference on Visualization*, page 192, 1995.

[55] F. J. Melero, P. Cano, and J. C. Torres. Bounding-planes octree: A new volume-based lod scheme. *Computer Graphics*, 32(4):385–392, 2008.

[56] Jörg Mensmann, Timo Ropinski, and Klaus Hinrichs. A gpu-supported lossless compression scheme for rendering time-varying volume data. In *Proceedings of the 8th IEEE/EG international conference on Volume Graphics*, pages 109–116, 2010.

[57] Susanne K Suter, Jose A Iglesias Guitian, Fabio Marton, Marco Agus, Andreas Elsener, Christoph PE Zollikofer, Meenakshisundaram Gopi, Enrico Gobbetti, and Renato Pajarola. Interactive multiscale tensor reconstruction for multiresolution volume visualization. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2135–2143, 2011.

[58] Quan Huynh-Thu and Mohammed Ghanbari. Scope of validity of psnr in image/video quality assessment. *Electronics letters*, 44(13):800–801, 2008.

[59] Nathaniel Fout, Hiroshi Akiba, Kwan-Liu Ma, Aaron E Lefohn, and Joe Kniss. High quality rendering of compressed volume data formats. In *Proceedings of the Eurographics/IEEE VGTC conference on Visualization*, pages 77–84, 2005.

[60] José Antonio Iglesias Guitián, Enrico Gobbetti, and Fabio Marton. View-dependent exploration of massive volumetric models on large-scale light field displays. *The Visual Computer*, 26(6-8):1037–1047, 2010.

[61] Han-Wei Shen. Visualization of large scale time-varying scientific data. In *Journal of Physics: Conference Series*, volume 46, page 535, 2006.

[62] Markus H Gross, Markus H Gross, and Markus H Gross. *Integrated volume rendering and data analysis in wavelet space*. Swiss Federal Institute of Technology, Computer Science Department, 1996.

[63] Insung Ihm and Sanghun Park. Wavelet-based 3d compression scheme for very large volume data. In *Graphics Interfaces*, volume 98, pages 107–116, 1998.

[64] James E Fowler and Roni Yagel. Lossless compression of volume data. In *Proceedings of the symposium on Volume visualization*, pages 43–50, 1994.

[65] Shane Dunne, Sandy Napel, and Brian Rutt. Fast reprojection of volume data. In *Proceedings of the First Conference on Visualization in Biomedical Computing*, pages 11–18, 1990.

[66] Takashi Totsuka and Marc Levoy. Frequency domain volume rendering. In *Proceedings of the Conference on Computer Graphics and Interactive Techniques*, pages 271–278, 1993.

[67] Tzi-cker Chiueh, Chuan-kai Yang, Taosong He, Hanspeter Pfister, and Arie Kaufman. Integrated volume compression and visualization. In *IEEE Proceedings on Visualization*, pages 329–336, 1997.

[68] Markus H Gross, Lars Lippert, and Oliver G Staadt. Compression methods for visualization. *Future Generation Computer Systems*, 15(1):11–29, 1999.

[69] Nathaniel Fout and Kwan-Liu Ma. Transform coding for hardware-accelerated volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1600–1607, 2007.

[70] Daisuke Nagayasu, Fumihiko Ino, and Kenichi Hagihara. Two-stage compression for fast volume rendering of time-varying scalar data. In *Proceedings of the International Conference on Computer Graphics and Interactive Techniques*, pages 275–284, 2006.

[71] Boon-Lock Yeo and Bede Liu. Volume rendering of dct-based compressed 3d scalar data. *IEEE Transactions on Visualization and Computer Graphics*, 1(1):29–43, 1995.

[72] T. Capin, K. Pulli, and T. Akenine-Moller. The state of the art in mobile graphics research. *IEEE Computer Graphics and Applications*, 28(4):74–84, 2008.

[73] Christopher L Farrow, Margaret Shaw, Hyunjeong Kim, Pavol Juhás, and Simon JL Billinge. Nyquist-shannon sampling theorem applied to refinements of the atomic pair distribution function. *Physical Review B*, 84(13):134105, 2011.

[74] Zhongde Wang. Fast algorithms for the discrete w transform and for the discrete fourier transform. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 32(4):803–816, 1984.

[75] A. Graps. An introduction to wavelets. *IEEE Computational Science Engineering*, 2(2):50–61, 1995.

[76] N. Ahmed, T. Natarajan, and K.R. Rao. Discrete cosine transform. *IEEE Transactions on Computers*, C-23(1):90–93, 1974.

[77] Ronald N Bracewell. Fourier transform and its applications. *McGraw-Hill Education*, 1980.

[78] Robi Polikar. The wavelet tutorial. 1996.

[79] E.J. Stollnitz, T.D. Derose, and D.H. Salesin. Wavelets for computer graphics: a primer.1. *IEEE Computer Graphics and Applications*, 15(3):76–84, 1995.

[80] Ingrid Daubechies. Orthonormal bases of compactly supported wavelets. *Communications on pure and applied mathematics*, 41(7):909–996, 1988.

[81] Albert Cohen. Biorthogonal wavelets. *Wavelets: A Tutorial in Theory and Applications*, 2:123–152, 1992.

[82] T. Cooklev and A. Nishihara. Biorthogonal coiflets. *IEEE Transactions on Signal Processing*, 47(9):2582–2588, Sep 1999.

[83] Zhongwei LI, Li CHENG, and Weiming TONG. Study of symlets wavelet amplitude algorithm. *Electric Power Automation Equipment*, 3:017, 2009.

[84] Jing Lin and Liangsheng Qu. Feature extraction based on morlet wavelet and its application for mechanical fault diagnosis. *Journal of Sound and Vibration*, 234(1):135–148, 2000.

[85] Joaquin Gonzalez-Nuevo, F Argüeso, Marcos Lopez-Caniego, Luigi Toffolatti, JL Sanz, P Vielva, and Diego Herranz. The mexican hat wavelet family: application to point-source detection in cosmic microwave background maps. *Monthly Notices of the Royal Astronomical Society*, 369(4):1603–1610, 2006.

[86] Emanuele Schiavi, C Hernández, and Juan A Hernández. Fully 3d wavelets mri compression. In *Biological and Medical Data Analysis*, pages 9–20. 2004.

[87] Rafat Mantiuk, Kil Joong Kim, Allan G. Rempel, and Wolfgang Heidrich. Hdr-vdp-2: A calibrated visual metric for visibility and quality predictions in all luminance conditions. *ACM Transaction on Graphics*, 30(4):40:1–40:14, 2011.

[88] Rafal Mantiuk, Kil Joong Kim, Allan G. Rempel, and Wolfgang Heidrich. Hdr-vdp-2: a calibrated visual metric for visibility and quality predictions in all luminance conditions. *ACM Transaction on Graphics*, 30(4):40, 2011.

[89] Jens Krüger and Rüdiger Westermann. Acceleration techniques for GPU-based volume rendering. In *IEEE Visualization*, pages 287–292, 2003.

[90] Stefan Röttger, Stefan Guthe, Daniel Weiskopf, Thomas Ertl, and Wolfgang Straßer. Smart hardware-accelerated volume rendering. In *Proceedings of the Symposium on Data Visualisation*, 2003.

[91] José M. Noguera, Juan-Roberto Jiménez, Carlos J. Ogáyar, and Rafael Jesús Segura. Volume rendering strategies on mobile devices. In *GRAPP/IVAPP*, pages 447–452, 2012.

[92] Cheuk Yiu Ip, A. Varshney, and J. JaJa. Hierarchical exploration of volumes using multilevel segmentation of the intensity-gradient histograms. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2355–2363, 2012.

[93] Jose Díaz, Eva Monclús, Isabel Navazo, and Pere-Pau Vázquez. Adaptive cross-sections of anatomical models. *Computer Graphics Forum*, 31(7):2155–2164, 2012.

[94] Cyril Crassin, Fabrice Neyret, Sylvain Lefebvre, and Elmar Eisemann. Gigavoxels: Ray-guided streaming for efficient and detailed voxel rendering. In *Proceedings of the Symposium on Interactive 3D Graphics and Games*, pages 15–22, 2009.

[95] Samuli Laine and Tero Karras. Efficient sparse voxel octrees. *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 55–63, 2010.

[96] Eva Monclús, José Díaz, Isabel Navazo, and Pere-Pau Vázquez. The virtual magic lantern: An interaction metaphor for enhanced medical data inspection. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, pages 119–122, 2009.

[97] José Díaz, Eva Monclús, Isabel Navazo, and Pere-Pau Vázquez. Adaptive cross-sections of anatomical models. *Computer Graphics Forum*, 31(7-2):2155–2164, 2012.

[98] Arie Kaufman. Efficient algorithms for 3d scan-conversion of parametric curves, surfaces, and volumes. In *ACM SIGGRAPH Computer Graphics*, volume 21, pages 171–179, 1987.

[99] Arie Kaufman et al. An algorithm for 3d scan-conversion of polygons. In *Proceedings of the Eurographics Conference*, pages 197–208, 1987.

[100] William E Lorensen and Harvey E Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *ACM Siggraph Computer Graphics*, volume 21, pages 163–169, 1987.

[101] Raj Shekhar, Elias Fayyad, Roni Yagel, and J Fredrick Cornhill. Octree-based decimation of marching cubes surfaces. In *Proceedings of the Visualization Conference*, pages 335–342, 1996.

[102] Marc Levoy. A hybrid ray tracer for rendering polygon and volume data. *IEEE Computer Graphics and Applications*, 10(2):33–40, 1990.

[103] Martin Frühauf. Combining volume rendering with line and surface rendering. In *Eurographics Association*, volume 91, pages 21–32, 1991.

[104] Tatsuo Miyazawa and Koji Koyamada. A high-speed integrated renderer for interpreting multiple 3d volume data. *The Journal of Visualization and Computer Animation*, 3(2):65–83, 1992.

[105] Lisa M Sobierajski and Arie E Kaufman. Volumetric ray tracing. In *Proceedings of Symposium on Volume Visualization*, pages 11–18. ACM, 1994.

[106] Carlos Buchart, Gaizka San Vicente, Aiert Amundarain, and Diego Borro. Hybrid visualization for maxillofacial surgery planning and simulation. In *International Conference on Information Visualisation*, pages 266–273, 2009.

[107] Marcelo Rodrigo Maciel Silva, Isabel Harb Manssour, and Carla Maria Dal Sasso Freitas. Optimizing combined volume and surface data ray casting. In *WSCG*, 2000.

[108] Kevin Kreeger and Arie Kaufman. Mixing translucent polygons with volumes. In *Proceedings of the Conference on Visualization*, pages 191–198, 1999.

[109] Imma Boada and Isabel Navazo. 3d texture-based hybrid visualizations. *Computers and Graphics*, 27(1):41–49, 2003.

[110] Werner M Jainek, Silvia Born, Dirk Bartz, Wolfgang Straßer, and Jan Fischer. Illustrative hybrid visualization and exploration of anatomical and functional brain data. In *Computer Graphics Forum*, volume 27, pages 855–862, 2008.

[111] Jörg Meyer, Steffen Gelder, Kay Kretschmer, Karsten Silkenbäumer, and Hans Hagen. Interactive visualization of hybrid medical data sets. *Proceedings of WSCG*, pages 371–380, 1997.

[112] David S Goodsell, I Saira Mian, and Arthur J Olson. Rendering volumetric data in molecular systems. *Journal of Molecular Graphics*, 7(1):41–47, 1989.

[113] Arie Kaufman, Roni Yagel, and Daniel Cohen. Intermixing surface and volume rendering. In *3D Imaging in Medicine*, pages 217–227. 1990.

[114] M Nordin Zakaria and N Selvanathan. Hybrid shear-warp rendering. *Malaysian Journal of Computer Science*, 12(2), 1970.

[115] Jianlong Zhou, Manfred Hinz, and Klaus D Tönnies. Hybrid focal region-based volume rendering of medical data. In *Bildverarbeitung für die Medizin*, pages 113–116. 2002.

[116] Helwig Hauser, Lukas Mroz, G Italo Bischi, and Eduard Groller. Two-level volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 7(3):242–252, 2001.

[117] Balázs Csébfalvi, Lukas Mroz, Helwig Hauser, Andreas König, and Eduard Gröller. Fast visualization of object contours by non-photorealistic volume rendering. In *Computer Graphics Forum*, volume 20, pages 452–460, 2001.

[118] Yanlin Luo, José Antonio Iglesias Guitián, Enrico Gobbetti, and Fabio Marton. Context preserving focal probes for exploration of volumetric medical datasets. In *Proceedings of the International Conference on Modelling the Physiological Human*, pages 187–198, 2009.

[119] Sun K Yoo, Jaehong Key, Kuiwon Choi, and Jinho Jo. Web-based hybrid visualization of medical images. In *Image and Video Retrieval*, pages 376–384. 2005.

[120] C. Rezk-Salama, K. Engel, M. Bauer, G. Greiner, and T. Ertl. Interactive volume on standard pc graphics hardware using multi-textures and multi-stage rasterization. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics Hardware*, pages 109–118, 2000.

This thesis was typeset by the author using LaTeX 2_ε. The main body of the text was set using a 11-points Time Roman font.

This work was carried out in the MOVING research group in the Department of Software (LSI) at the Universitat Politècnica de Catalunya and in the Research Center for Visualization, Virtual Reality and Graphics Interaction (ViRVIG), which is a Joint Research Unit (JRU) of the European Comission with the support of the Spanish Government.

This project has been supported by the TIN2010-20590-C02-01 Project of the Spanish Government. The author has been supported by a UPC PhD grant.

Printed by *El Taller Graphic*-BCN PRINT HOUSE S.L.

A catalogue record is available from the Library of the Universitat Politècnica de Catalunya. An electronic version is also available at the Doctorate Thesis Network (http://tdx.cat).