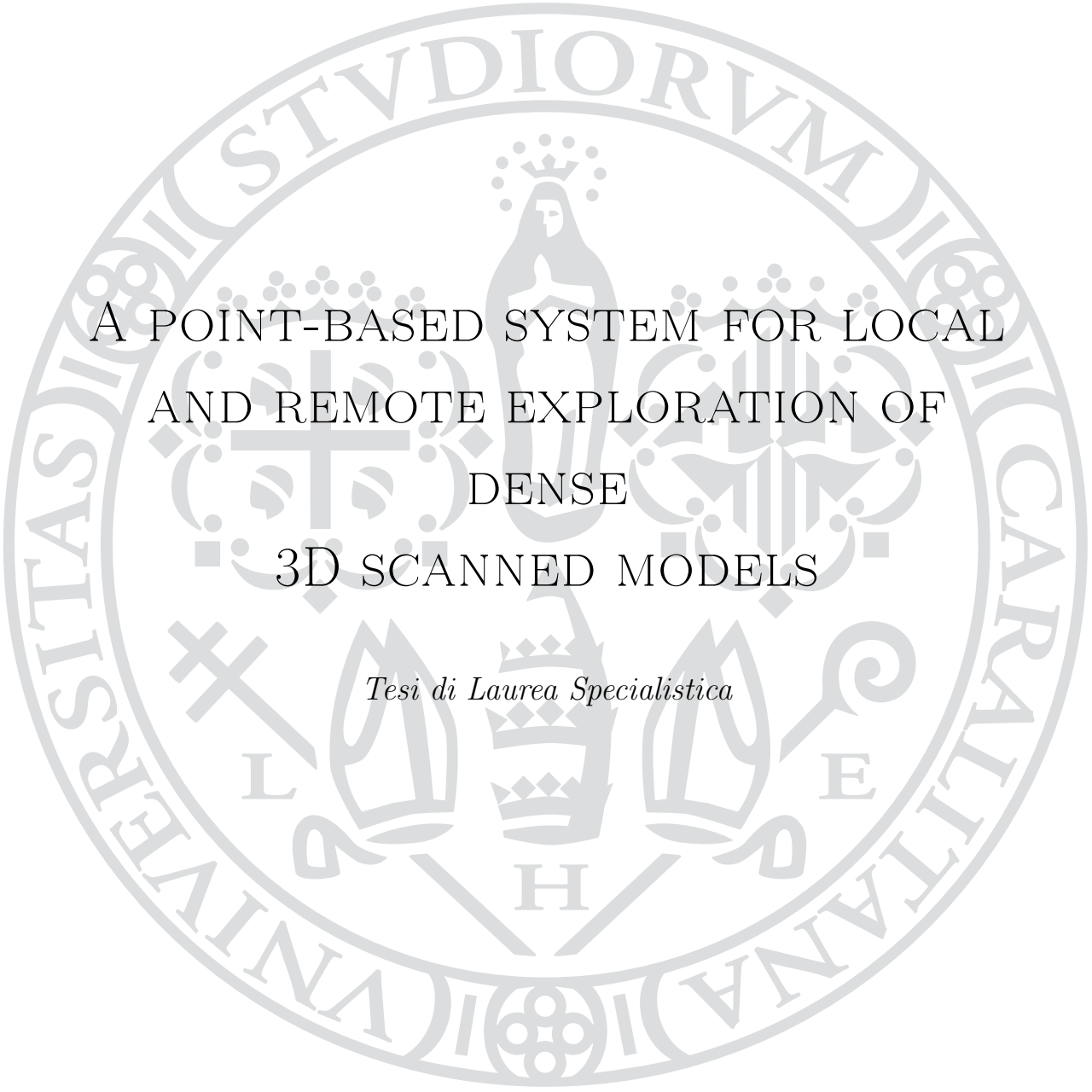

A point-based system for local and remote exploration of dense 3D scanned models

Tesi di Laurea Specialistica

Università di Cagliari

16 Dicembre 2011

Università di Cagliari

Facoltà di Scienze MM.FF.NN.

Dipartimento di Informatica

Corso di Laurea Specialistica in Tecnologie Informatiche

A point-based system for local and remote exploration of dense 3D scanned models

Tesi di
Alex Tinti

Relatori
Prof. Riccardo Scateni .....

Fabio Marton .....

Candidato
Alex Tinti .....

Sessione di Laurea del 16 Dicembre 2011
Anno Accademico 2011/2012
Consultazione Consentita

Sommario

Le odierne tecnologie di scansione laser, così come le tecniche di fotografia digitale, permettono di acquisire facilmente modelli composti da svariati milioni di punti, consentendoci quindi, di ottenere facilmente una rappresentazione digitale in alta qualità dell'oggetto di interesse ad un dettaglio sub-millimetrico.

Sfortunatamente, i tradizionali algoritmi di visualizzazione non sono in grado di gestire un ammontare di dati di tale dimensione in tempo reale, su piattaforme grafiche di uso comune e di conseguenza, all'utente finale viene generalmente mostrato solo una versione semplificata in bassa risoluzione del modello acquisito.

Questa tesi si focalizza su un sistema client-server in grado di distribuire sulla rete e visualizzare in tempo reale, modelli 3D di grandi dimensioni su elaboratori di fascia media. La strategia di visualizzazione è basata su una struttura gerarchica in multi-risoluzione, un *BSP tree*, composta da nodi compressi con migliaia di punti ciascuno. I nodi vengono costruiti tramite un processo a 2 fasi: durante la prima si estraggono i nodi foglia dal dataset iniziale, mentre nella seconda, partendo dalle foglie si creano i nodi interni filtrando le coppie di figli sino ad ottenere un'unica radice.

La rappresentazione attuale e dipendente dalla vista della struttura in multi-risoluzione viene aggiornata incrementalmente a tempo d'esecuzione da un processo di raffinamento adattivo che accede ad un dataset out-of-core locale o remoto. I Fragment ed i Vertex Shaders vengono utilizzati, oltre che per i classici effetti di ombreggiatura ed illuminazione, per disegnare la rappresentazione del modello memorizzata nella RAM del processore grafico attraverso una tecnica basata su ellissi opportunamente orientati.

L'interfaccia utente single-touch, che permette agli utenti finali la fruizione del modello, include un sistema di hyperlink bidirezionali per l'accesso a contenuti multimediali, connettendo le diverse parti del modello a differenti sorgenti d'informazione. Sono inoltre stati implementati strumenti di misurazione

per distanze, superfici ed angoli, in aggiunta alle classiche funzioni per il controllo del modello, della camera e dell'illuminazione.

Il sistema vuole fornire un'esplorazione intuitiva di modelli 3D ad alto dettaglio, da piccoli artefatti a siti più vasti. Può inoltre essere distribuito come web-plugin, come applicazione stand-alone o come installazione per un kiosko museale, da posizionare vicino all'opera d'arte originale per migliorare l'esperienza dell'utente.

Abstract

Today's 3D laser scanning technologies and digital photography techniques allow to easily acquire multi-million points models at a sub-millimetric detail, providing a hi-quality digital representation of the object of interest. Unfortunately, standard rendering algorithms are not capable to deal, at run-time, with a such huge amount of data on commodity graphic platforms and often just a coarse grained low-res model can be shown to the end-user.

This thesis focuses on a client-server system for real-time point based rendering and network distribution of large 3D models on low-end platforms. Rendering strategy is based on a hierarchical multi-resolution structure, a *BSP tree*, composed by compressed nodes with thousands of point samples. Nodes are built in a 2-phases process: the first phase extracts leaf nodes from the raw input dataset; the second instead, starting from leafs, builds inner nodes by merging and filtering pairs of children to obtain the parent node, until the unique root is constructed.

The actual view dependent representation of the multi-resolution model gets incrementally updated at run-time by an adaptive refinement process that fetches from the local or remote out-of-core multi-resolution structure dataset. Vertex and fragment shaders are used to render the GPU cached model representation with a hi-quality elliptical sample drawing and for other several shading effects.

Single-touch user interface, which allows end-users the model inspection, includes a bidirectional-hyperlink system to access to remote multimedia contents, connecting different parts of 3D model to several information sources. Area, distance and angle measurements instruments have also been implemented as well as more common tools like model, camera and light control.

The system can provide an effortless exploration of hi-detailed 3D models, from small artifacts to larger sites and can be distributed as web-plugin, stand-alone application or even as museal kiosk installation, to be placed next to the real 3D artwork in order to improve visitor's experience.

... to my parents

Contents

List of Figures

Preface

The content of this thesis has been published by the same author along with other CRS4 researchers (Center for Advanced Studies, Research and Development in Sardinia) at the VAST conference (The 10th International Symposium on Virtual Reality, Archaeology and Cultural Heritage VAST) held in Malta on September 2009.

This thesis consists in a detailed summary of the work started in the year 2008 with the CRS4 Visual Computing group (ViC) under the direct guidance of Enrico Gobbetti and Fabio Marton, whom I have both to thank for giving me this unique opportunity to work in a such skilled group and learn from their over ten-years experience in the Computer Graphics field. Concerning this, I really own a more thank to Enrico for trusting me giving me the chance to present this work to the VAST conference for my very first time.

I would also like to express my infinite gratitude to Fabio Bettio, good friend and mentor, for his invaluable advices and constant support; and to Lab 3D colleagues: Roberto Combet, Gianmauro Cuccuru, Emilio Merella and Claudio Mura, as well as other ViC members left: Ruggero Pintus, Antonio Zorcolo, Marco Agus, Giovanni Pintore and José A. Iglesias Guitián, who all have always given me an hand every time I needed.

My last and valuable thank goes to my family, in particular my parents, my sister Jessica and my life partner Gabriella; I will never been here writing these lines without their persisting sustain and trust on my doings.

December 16th, 2011, Pula, Italy *Alex Tinti*

Chapter

1 Introduction

Nowadays last laser scanning technologies have such amazing performance, like acquisition speed, hi-precision and wide operating ranges, that make it possible to acquire hi-detailed representations of huge slices of the real world in just a few minutes of work. Of course, a such high detail needs great amounts of data to be stored in our hard drives, which cannot entirely fit in the central memory of any of consumer PCs. This brings to light the most important issue we had to deal with: the real-time visualization of hi-detailed 3D scanned models.

Moreover, we also wanted this data to be accessible from anywhere on the Internet; allowing people to interactively browse a 3D model from his home position and giving him access to some basic measurement tools to get quantitative informations like distances, areas and angles.

1.1 Motivations

Cultural heritage is probably the sector which takes major advantages from nowadays laser scanning technologies, for archiving, restoration and dissemination purposes. However, for several reasons, visiting a site could not always be within everyone's reach and the ability to access to the same digital representation, sometimes with a detail even higher than that reachable by the human eye, can be a really valuable way to experience and understand an artwork. Just think about a large statue like Michelangelo's David; many points of view are not directly visible to ordinary visitors, while the chance to move, rotate and zoom any detail of his scanned representation can even enhance the user experience. In the same way, consider a larger archaeological site. Often is not possible get close enough to some particular point of interest or artifact, due to necessity to keep evidences of past activities preserved or sometimes just for safety reasons. A virtual tour of the scanned site instead, can give ordinary visitors access to unaccessible places enriching their experiences through additional multimedia contents properly integrated with real scanned objects.

Figure 1.1: Virtual Interaction. A real site digital representation displayed on a wide touch screen by our multi-resolution renderer.

1.2 Objectives

Complete and final 3D scanned models are usually the result of many single scans, cleaned, aligned and merged together to compose a multi-million point geometry. Most of the times, in order to be streamed and visualized by remote end users, these large models need to be simplified into smaller coarse objects, hindering the original detailed scans. Despite the constant and rapid improvement of the graphic hardware in the recent years, interactive and real-time rendering of multi-giga dataset, as well as Internet streaming, still remains one of the most challenging issue, being capable to overload both performance and memory capacity of state-of-the-art graphic and network system. For such complex and large geometries, multi-resolution hierarchies of point primitives can be considered a valuable alternative to the more traditional mesh refinement (CGG*04). These techniques are based on the fact that hi-resolution models, like those generated by scanning devices, have such high sample rates that triangles will be projected on really small areas of the screen while, properly selected points splat are sufficient to accurately reproduce the same model portion. Furthermore, the release of the mesh connectivity information will result in a lightweight data-packet to be faster streamed through the net.

Figure 1.2: Distance Tool. The implemented framework running on a wide touch screen.

1.3 Achievements

With the work carried out for this thesis, have been accomplished the following achievements :

- The design of an easy to use point-based streaming solution for remote high performance view-dependent visualization of very large static point sampled models on consumer graphics platforms. The progressive block-based refinement nature of the rendering traversal is well suited to hiding out-of-core data access latency, and lends itself well to incorporate view frustum culling, as well as compression and view-dependent progressive transmission.

- The implementation of client-server framework, able to give end user a real-time interaction experience of the full resolution model using a limited bandwidth. The model description is enhanced through a tree of information layers, each of them made of several 3D hot spots which allow users to visualize additional textual information. With a bidirectional hyperlink system the user can move from the 3D model to a web page and vice versa.

- The integration of potentially extensible toolset, to improve model exploration by

Figure 1.3: Nora Underground Tank. The underground tank, recently discovered in the archeological site of Nora, is not accessible anymore but can still be explored by our framework.

providing basic measurement functionalities to get quantitative informations like distances, areas and angles.

- And finally, the user interface design, based on context and gesture recognition, to allow users to navigate within a 3D environment, as well as manipulate a 3D model/tool in an intuitive manner. The interface requires only a single-button stylus or mouse (or a single- touch screen) to directly invoke specific operations within a single 3D view. The system is thus appropriate both for local (museum) settings and remote viewing.

1.4 Related work

The London Charter (RBN) for the use of 3-dimensional visualisation in the research and communication of cultural heritage seeks to establish what is required for 3D visualisation to be, and to be seen to be, as intellectually rigorous and robust as any other research method. Niccolucci et al. (ND06) presented an ontology for such complex objects that is sufficiently general to encompass very different artifacts, from pottery sherds to historical landscapes and simple enough to be used and understood by heritage practitioners and

professionals with moderate computer skills. Havemann et al. (HSB*08) proposed a similar application, but based on the Collada file format, which can render 3D models labeled with updatable textual content similar to hyperlink and link anchor. These applications anyway, are still unable to solve the problem of rendering huge 3D models, like those that can be generated by 3D laser scanners. Virtual inspector (CPCS08) lets naive users inspect large 3D models at interactive frame rates on commodity PCs. The system obtains visualization efficiency without sacrificing quality by adopting a continuous level-of-detail representation. Visual inspector's use of XML to encode the GUI's structure and behavior makes it flexible and configurable. The framework proposed in this thesis takes a similar approach to Virtual Inspector, but focusing on a point-based multi-resolution hierarchy rather than a triangulated one. In this way, discarding mesh connectivity information, can be provided both local and remote explorations.

1.4.1 Point-based Rendering Techniques

The rendering of dense 3D models through point-based 3D graphics techniques has been studied firstly by Grossman (LW85, Gro98). The improved approach QSplat (RL00, RL01) instead, has been considered for a long time a reference system in the particular area. The system for representing and progressively displaying these meshes, combines a multi-resolution hierarchy based on bounding spheres with a rendering system based on points. A single data structure is used for view frustum culling, backface culling, level-of-detail selection, and rendering. The compact representation can be quickly computed, making it suitable for large data sets. Other several authors have presented different way to improve rendering performance up the its limit, often through a retained-mode interface working with blocks of points (WFP*01, SD01, DVS03, GM04, PSL05, WS06). Layered Point Clouds (GM04) was the first work to introduce a coarse grained multi-resolution technique. This system exploits a partitioning of the model into clouds to improve the efficiency of CPU/GPU communication through a batched communication protocol and to support conservative occlusion culling for high-depth complexity models. Our rendering subsystem is based on Layered Point Clouds, but instead of using finer hierarchy levels to increase resolution of the coarser representation by using all the nodes from the root to the current cut, in our case all nodes are self-contained and the model representation is given only by the leaf nodes in the current cut. Moreover, our inner nodes are produced by a high quality simplification methods, while Layered Point Clouds are constrained to work on uniformly sampled models using pure subsampling.

Figure 1.4: Layered Point Cloud. The first work to introduce a coarse grained multi-resolution technique.

1.4.2 Elliptical Splat Drawing

There are several works that try to enhance the rendering quality of point-sampled models. For very large models, it's common to use spheres (RL00), tangential disks (PZvBG00, ZPvBG01), or high degree polynomials (ABCO*01) instead of raw point primitives, as well as improving filtering in image space (ZPvBG01) or object space (RPZ02). The system proposed in this thesis can provide two different rendering mode: the first one is based on simple OpenGL smooth points; while the second one, for most advanced graphic cards, supports vertex and fragment shaders to perform elliptical splatting as proposed in (BK03). The rendering pipeline also supports screen space ambient occlusion as well as tone mapping.

1.4.3 Single-touch User Interface

In order to allow naive users to explore 3D large sites and models, these kind of systems have to provide a quite intuitive user-friendly interface. Our single-touch interface can be considered similar to Unicam (ZF99), with some additional options like model manipulation and ambient exploration. Moreover, has been integrated a direct camera manipulation interface and a system for selection predefined views, as done, e.g., in the recent Safe 3D navigation interface (FMM*08).

Chapter

2 Data Representation

Interactive real-time 3D models rendering frameworks are usually composed by a client that constantly fetches data packets from the server, to provide smooth updates to the previous rendered model representation in response to the viewer motion. To this end, a compressed data representation needs to meet several requirements: high compression ratio, in order to reduce transmitted data; random access to different portions of the dataset; hierarchical structuring able to support variable resolution; low decoding complexity at both server and client-side, to ensure scalability and support heterogeneous clients.

We have managed to meet these requirements defining a data representation based on coarse-grained multi-resolution point-based hierarchy. We've also made a specialized version of the end-user viewer, in order to allow the access to external layers of multimedia contents from hot-spots attached to the model.

2.1 Attributes Precomputation

The input model we need to process is composed by a certain number of sample points, each of which has associated attributes including position, normal and eventually color and radius information. During the pre-processing step, data pre-filtering and radius computation operation have performed as well as others using a streaming approach as proposed by (Paj05). Points are sorted along a direction in Euclidean space (assumed to be the z-axis without loss of generality).

Before sorting, we rotate the point set to align the dominant axis of its covariance matrix with the z-axis. In this way, we can reduce the maximum complexity encountered during streaming. Then different kinds of operators can be applied to compute attributes which depend only on a local neighborhood, by only loading into memory a small layer of data around the currently processed point.

In our system, the first preprocessing step consists in associating an influence radius to each point, which is done either by using a PCA approach (PGK02) or by fast density estimation (CGM*09).

```
class bsp_data {

  box_t box_;
  float mean_sample_distance_;
  bool is_leaf_;

  std::vector<point_t>  points_;
  std::vector<normal_t> normals_;
  std::vector<color_t>  colors_;
  std::vector<float>    sample_distances_;
};
```

Figure 2.1: Node Data Representation. Node attributes – bounding box, mean samples distance and the is_leaf_ boolean – followed by point associated attributes that are stored separately inside an *std::vector* container.

2.2 Multi-resolution Model Construction

The implemented multi-resolution structure is a coarse-grained kd-tree partition of the input dataset, with leaves containing less than a predefined target sample count (few thousands of samples) and with inner nodes containing a filtered version of their children, with a number of samples equal to the target count. We construct this structure off-line, starting from the generic point cloud model and perform a simple I/O efficient recursive clustering algorithm, which is realized on top of a single out-of-core component: a standard C++ array (compatible with `std::vector`), which keeps a small in-core window for requested data, in order to handle, efficiently, even very large datasets.

The construction procedure, explained in the following subsections, consists of two phases.

2.2.1 1st Phase: Building Leaf Nodes

The first phase of the hierarchical multi-resolution structure construction, consists in building and extracting leaf nodes from the raw input dataset, partitioning it into a kd-tree of point clouds. This is done by processing an out-of-core array of uniformly distributed point samples, together with its bounding volume. This recursive procedure firstly considers the N samples of the current points set, if N is less than M, the maximum predefined point count for a single node, than a new leaf node is generated; otherwise the current points set gets parted by bisecting the bounding box at the midpoint of its longest axis. The two new children will be later processed, recursively continuing the partitioning procedure.

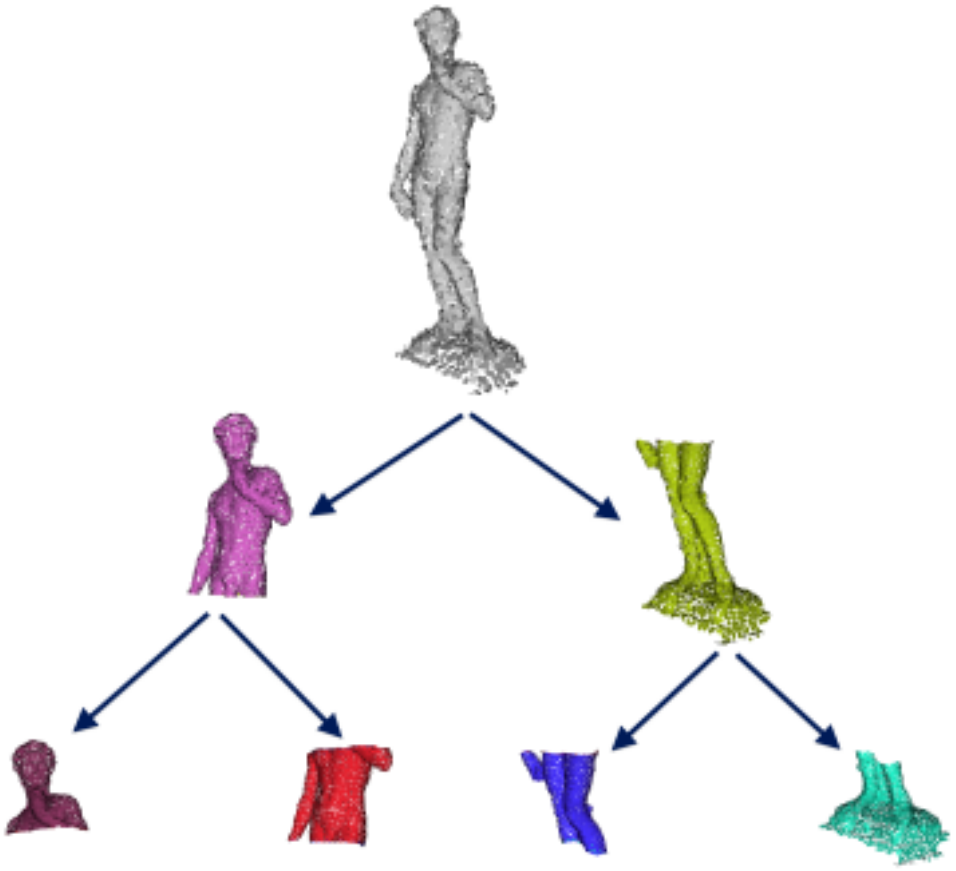

Figure 2.2: Multi-resolution structure. From the coarser representation of the root to leaves with finer informations.

2.2.2 2nd Phase: Building Inner Nodes

During the second phase inner nodes are built by merging and filtering children samples to create the parent node composed by M samples. The filtered version is obtained using an edge collapse simplification procedure adapted to point clouds. We connect each sample to its eight nearest neighbors and build a local kd-tree with all the samples of the two children in order to compute the neighbor information. A map of edges sorted by edge length is instead used for the simplification procedure, which firstly collapses the two points with the lowest edge length and then updates all new edge lengths; structures that allow the vertex neighbors identification need to be updated as well.

This procedure goes on until the parent node has only M samples left. The new obtained point position is equal to the midpoint of the two sample merged and a weight, proportional to sample splat area, is used to linearly combine their colors and normals. The radius value instead, is assigned as the minimum radius that covers both original samples from the new sample position.

2.3 Compression

Unfortunately, the only multi-resolution approach alone, is not enough to permit a smooth and low-latency inspection of the model in remote settings. The hierarchical tree, with different levels at different resolutions, allows to fetch only nodes we actually need to renderer, obtaining a view-dependent model representation, but it will be even more advantageous to compress nodes data to better exploit available network bandwidth.

The multi-resolution structure contains nodes composed by per-node data, basically a simple header, and per-point data. The header, that is stored and transmitted uncompressed, contains a flag to indicate whether the node is a leaf or not, the node's bounding box, the radius range (minimum, maximum, and average) and the total count of points contained in the associated point cloud. The per-point data instead, consists different arrays of sample attributes (position, normal, color, and radius); gets compressed and later stored in the multi-resolution structure and de-compressed at run-time just before entering the view-dependent model representation.

We chose to use a simple wavelet-based compression scheme, which is quite fast and is also able to reduce memory occupancy by almost an order of magnitude.

During the first phase of compression, we sort the point cloud in an order that minimizes the Euclidean distance among subsequent points, to produce a *point strip*. Points attributes in the point strip are then stored in a the rows of a 2D array. These rows are then transformed using a reversible n-bit to n-bit transform based on the Haar wavelet transform in order to reduce entropy (SLDJ04).

The transform is performed iteratively and at each iteration is applied to the low-pass coefficients resulting from the previous iteration until there is only a single low-pass coefficient remaining. We next map resulting coefficients to positive integers using a simple Elias gamma code (Eli75) to encode them; therefore, the positive integer x is represented by: $1 + \lfloor \log_2 x \rfloor$ in unary (that is, $\lfloor \log_2 x \rfloor$ 0-bits followed by a 1-bit), followed by the binary representation of x without its most significant bit. To de-compress data we just undo above steps in the reverse order, with the exception of the sorting. To transform the input data to coefficients, we perform an adaptive quantization which also considers the local sample spacing. Positions are expressed in relation to the node bounding box and quantized to the minimum number of bits per component that produces a quantization error less than a quarter the minimum sample radius. The same error threshold is used for radii which are expressed in relation to radius range and during decompression, in order to ensure that no additional hole is introduced, radii are enlarged by the quantization error. Normals are quantized to 16 bits using radial projection. Colors are mapped to the YCoCg-R color space (MS03) to reduce correlation and mapped to 5 bits for the luminance and 6 bits for each chroma components.

The average compression rate obtained on our test datasets is of 3.6 bytes per sample, which means that, for a standard patch of 2000 samples, the compressed occupied space is about 7KB instead of 62KB for the original uncompressed data. This is a quite competitive result if compared with other point based systems; for instance, QSplat (RL00) uses 5.375bytes/sample to encode the same attributes.

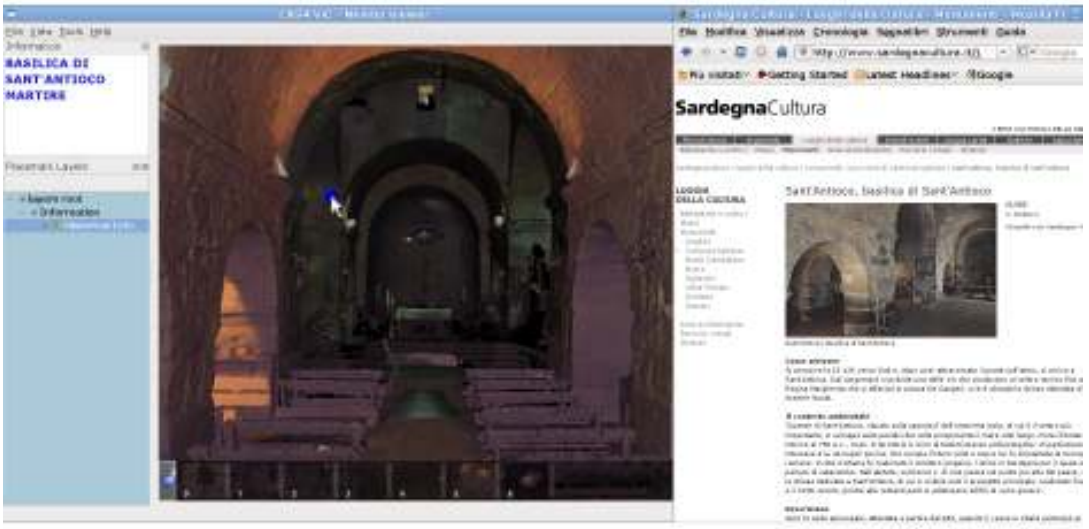

Figure 2.3: Sant'Antioco Basilica. Information layer structure, hot spot, and web browser additional information.

2.4 Information Layers

We wanted to enhance the user experience by permitting the access to different types of external multimedia contents, linked to various parts of the model through 3D hot-spots. Through the hot-spot interface, the user can so enable/disable different layers hierarchically organized in a tree, which can provide several kind of information; textual data for instance, could contain historical informations, as well as useful scanning annotations used to keep track of the acquisition procedure. Moreover, in order to prevent the model perception to be spoiled when too many hot-spots are present at the same time, can be enabled an additional function that visualize just hot-spots inside a small invisible round area centered in the mouse cursor. Another interesting feature is the bidirectional connection between model and multimedia layer; this means that we can access external informations (text, pictures, websites ...) by clicking the related hot-spot, but can also visualize the model from a particular point of view through an hyperlink in a webpage. All these additional informations can be held in separated servers and stored as hierarchical tree in xml files.

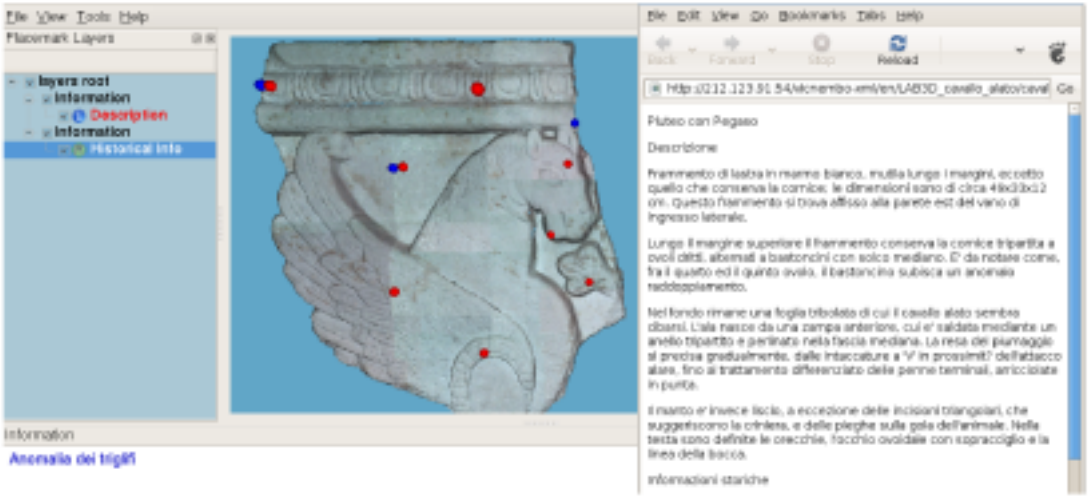

Figure 2.4: Cavallo Alato Bas-relief. Information layer structure, hot spot, and web browser additional information.

Chapter

3 Client-server Framework

This chapter contains detailed informations about the client-server framework; in particular about the data storage, server-side distribution components and the client-side streaming and rendering methods. The main objective is to better describe how the compressed multi-resolution point cloud representation is able to adapt to client characteristics and exploit available network bandwidth. A block diagram of our system's design is presented in figure 3.1.

3.1 Server design and implementation.

The server-side system has been designed to support and handle different data repositories at the same time. A repository, looked from the server side, simply consists in a database which stores the entire compressed information of a single 3D model. The implemented system uses a unique node ID to access the bytes block containing the compressed node information of the hierarchical multi-resolution structure.

The storage component instead, has been implemented through the Oracle Berkeley DB library, which provides complex data management features including high throughput, low-latency reads, high concurrency, data scalability and in-memory caching. It's a reliable solution with over 15 years of production capable to deliver fast, scalable and flexible data management services and handle up to terabytes of data. A Berkeley DB subsystem within an environment is described by one or more regions, containing all of the per-process and per-thread shared information; including configurable caches. Thanks to these caches, different instances of the same database are able to share index memory and reduce memory load for servers dealing in parallel with hundreds of clients.

Data serving is done through an Apache2 server extended with an ad-hoc module. Apache2 is a secure, efficient and extensible server for modern operating systems that provides HTTP services in sync with the current HTTP standards. Besides, it is scalable, multithreaded and includes features like persistent server processes and proxy load balancing, which are essential for the performance of our application. The implemented

custom apache module supports a simple HTTP-based connectionless protocol and allows clients to send simple data requests, just specifying the server and the repository names followed by the node ID. When the apache module parses the request string, first extracts the repository name and the ID, represented by unsigned integer, and then fetches the compressed node accessing the Berkeley database. In case the node would not be found, than an appropriate error signal is sent to the client.

The chosen approach is based on open-source API; it's also simple to implement, maintain and even upgrade with new features. Most importantly, offers a very good performance, mainly thanks to concurrency features of Apache which permit to handle thousands of clients in parallel through the forking of multiple child servers sharing the same database.

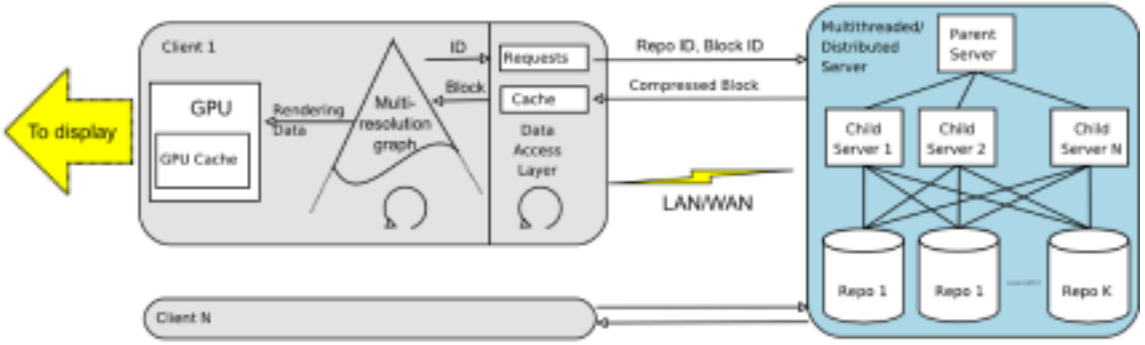

Figure 3.1: Client Server Architecture.

3.2 Multi-threaded clients for remote visualization

3.2.1 Incremental view-dependent refinement.

The client manages a binary tree that represents the 3D model from the coarser representation (given by the root) to the current view-dependent representation, which is the current cut of the graph.

The traversal algorithm, which extracts the view dependent representation of the multi-resolution model from the current point of view, is based on a stateless coarse-to-fine refinement of our structure. The refinement exploits the progressive nature and coarse granularity of the multi-resolution hierarchy to reduce CPU refinement costs and to improve repository-to-host and host-to-graphics communication.

In particular, asynchronous repository requests hide the out-of-core data access latency, and the communication with the GPU is made exclusively through a retained mode interface that reduces bus traffic by managing a least-recently-used cache of point clouds maintained on-board as OpenGL Vertex Buffer Object.

Figure 3.2: Patch Refinement. 1665 patches with a total of 1.3M samples.

The user-selected pixel threshold is the value that drives the refinement of the rendering algorithm: this value represents the required average sample distance between adjacent splats on the screen. The refinement algorithm, performing a single-pass recursive traversal of the multi-resolution structure, selects the nodes that have to be rendered. For each node, we use its bounding box to test whether the node is totally outside the view frustum. In this case, recursion stops, discarding the entire branch of the tree, otherwise we can render the node, or possibly continue the refinement with its children. We project the node's average sample distance onto the screen to obtain its average splat size. A consistent upper bound on the projected size is obtained by measuring the apparent size of a sphere with a diameter equal to the object space average sample distance and centered at the bounding box point closest to the viewpoint. If the projected splat size is less than the threshold, we select the node's point cloud for rendering and we coarsen the subtree underlying the node, to remove overrefined data from the current representation. Otherwise, if the projected splat size is higher than the threshold, we try to refine the node. In that case, in order to avoid

blocking the renderer because of data access latency, especially in the case of rendering data over wide-area networks, we first check whether the node's children data is immediately available, i.e., if it is already in the GPU cache or considered in-core by the data access layer. If data is available, we proceed with refinement, otherwise, we select the node for rendering. When refinement is completed, all the selected nodes can be rendered, using their cached GPU version if already available or creating a new VBO entry in the GPU, for the nodes which were just created in the last traversal (see figure 3.2).

Figure 3.3: Sant'Antioco Basilica. Rendered at 100 fps on a 1024x768 window, with elliptical splats enabled.

3.2.2 Multithreaded data access layer.

A multithreaded data access layer hides from the application the technique used for accessing the model repository. In our current implementation, we use a HTTP/1.1 persistent connection approach and optionally employ HTTP pipelining. The combination of these two techniques improves bandwidth usage and reduces network latency. We can

also keep the protocol simple from API point-of-view, since clients benefit from an underlying connection-based implementation hidden under a reliable connectionless interface. The pipelining approach allows multiple HTTP requests to be written together out to the socket connecting client and server ,without waiting for the corresponding responses. The client then waits for the responses to arrive in the same order in which they were requested. Requests pipelining can turn out in a dramatic response times improvement, especially over high latency connections.

The client data access layer is subdivided into two threads that communicate through a shared cache of compressed point clouds indexed by node IDs to hide network latencies. The main thread requests the node data that is needed to refine the kd-tree graph from the current point of view. Requests are pushed in a priority queue. At the end of the frame, only as many new requests as those allowed by the estimated network bandwidth are issued and managed by a separate network access thread, and the remaining ones are ignored. Since issued requests are sorted coarse to fine and by estimated projected error, and unhandled requests are repeated at each frame, a simple limited memory first-in/fist-out queue induces a request ordering that is both I/O efficient and ensures to download the most relevant data as soon as possible. The second thread manages the network by accepting the incoming compressed bitstream from the server and storing them in the shared cache.

3.3 Rendering process.

At the end of the incremental refinement process, graph's leaves contain the current model approximation, in a format suitable for graphics rendering. Since we have to manage heterogeneous clients with varying graphics capabilities, in our implementation, at the beginning of rendering, we detect which kind of GPU is available, thus tuning the graphics approach that will be used to enable our application to work also on low-end GPUs or, in the extreme case, in software-only environments. Communication with the GPU is made exclusively through a retained mode interface. We manage a cache of Vertex Buffer Objects in the GPU to exploit spatial and temporal coherence, reusing the same data for several frames without no need to move it again to the GPU. A backup solution is also used for the CPU - GPU data communication if no Vertex Buffer Objects are available, and it is based on the standard Vertex Arrays, which are provided since OpenGL 1.1. Two rendering modes are supported: a first simple straight rendering method that uses a single splat size for each point cloud, and draws circular splats through glPointSmooth. A higher quality representation has instead been obtained through a vertex and a fragment shader to draw an oriented 3D circle, depicting a textured quad for each sample (BK03). This second rendering approach tries to simulate a smooth surface drawing each splat with his

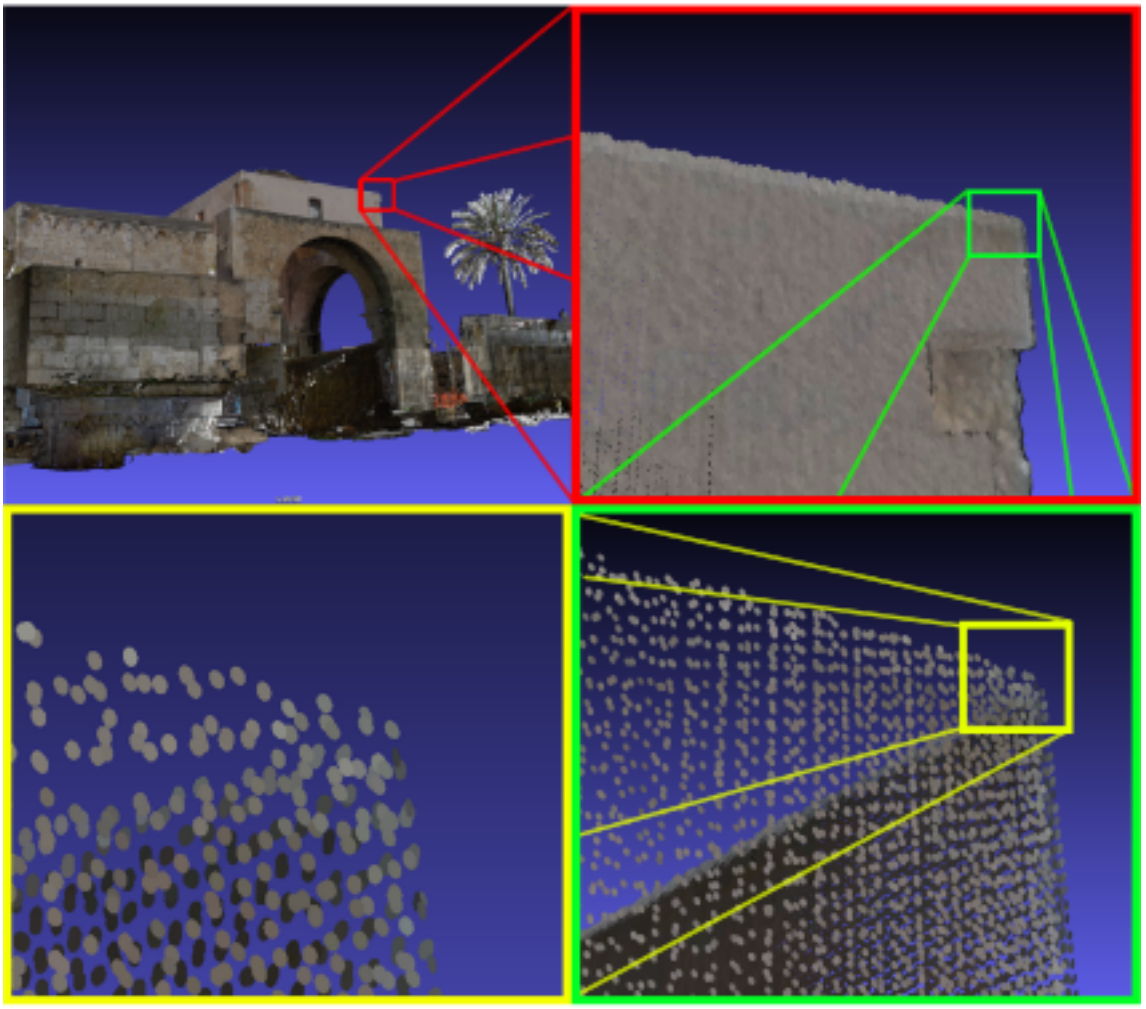

Figure 3.4: Oriented Splat Rendering. Vertex and a fragment shader allow to render overlapped and oriented 3D splats in order to simulate smooth surfaces.

own size, that is proportional his average sample distance, and orienting it so that the surface is perpendicular to the normal vector.

Chapter

4 User Interaction

Typical 3D navigation applications provide modal tools like pan, zoom, and rotation to facilitate freeform navigation in the 3D scene. Mastering these navigation tools requires a significant amount of learning, while we want to provide an inspection experience which could be enjoyed also by new-to-3D users. The application can be used as a remote exploration tool through a web plugin to facilitate dissemination of huge 3D models, but can also be placed in a museal kiosk installation next to a real 3D artwork to improve museum visitor's experience. In both cases, we must allow for inexperienced users. User interaction must be kept as simple as possible and the application should not only be operated with standard 2D mouse, but also through a touch-screen for multimedia kiosk installations. In our approach, all the user interface is designed to require input from a single-button mouse or a single-touch screen. Context information and gestural recognition are exploited to intuitively choose among different actions.

4.1 Single-touch interaction.

Our approach, based on context and gesture recognition, allows users to navigate within a 3D environment, as well as to manipulate a 3D model/tool in an intuitive manner. It requires only a single-button stylus or mouse (or a single-touch screen) to directly invoke specific operations within a single 3D view. No 2D widgets or keyboard modifiers are necessary. Our viewing window is subdivided into two areas: a rectangular *center* region and a small bar on the right, highlighted in a light semi-transparent color (see figure 4.2). If the mouse is over the right bar, the movement along the y screen axis is interpreted as moving forward/backward. When the mouse is over the main area, we need to differentiate among four different actions: rotation around a pivot, rotation around camera, panning, and moving toward a target. A single click on the 3D model activates a target, a small sphere below the 2D mouse position is displayed. Subsequent actions have the following meanings: click and drag within the main area performs a rotation around the target. Click on the target will automatically animate the camera from its current position to a new position that looks at the target. The target is deactivated by clicking

Figure 4.1: Touch Screen. Selection and animation moving toward precomputed view.

outside the small sphere.

If the target is not active, actions within the main area are interpreted in two different ways. A film-plane translation (panning) is performed by starting the movement vertically, while a rotation around the viewpoint is performed by starting the movement horizontally. This last gesture is particularly useful when the user explores an environment. In addition to camera motion, single touch interaction is also used for operating simple 3D tools (see below). For camera motion, the single-touch interaction system has been enriched with the possibility of browsing a list of precomputed view positions, displayed as a series of thumbnails in the lower part of the screen. When a view is selected, the camera is smoothly animated from the current position until it reaches the selected view (see figure 4.1).

4.2 Tools

The client framework provides some simple tools to perform different kinds of measurements: lenghts, areas and angles. The tools can be optionally enabled in the viewer, depending on configuration. All these tools are based on the ability to project the 2D mouse position onto the model to find a 3D intersection. This is achieved by casting a ray toward the 3D model and exploiting the kd-tree structure for traversing all the intersected leaf nodes in a front-to-back order until an intersection with the point cloud is found. Examples of tools are measuring tools (for distances, areas, and angles). The

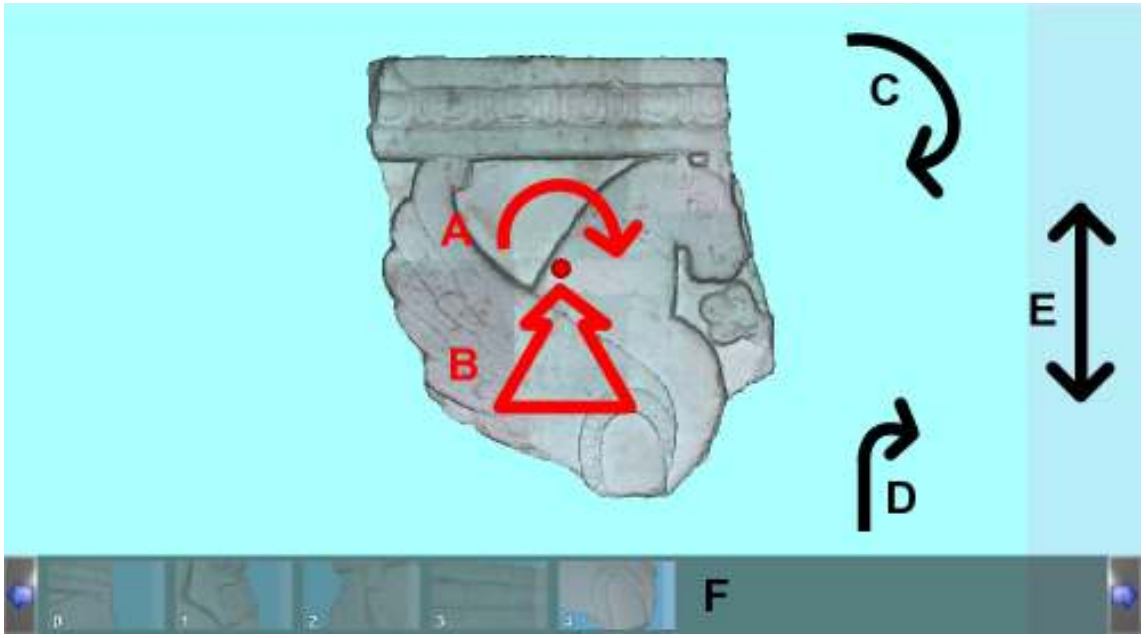

Figure 4.2: Single-touch interaction scheme. Red arrows, activated when an anchor is present: A) rotation about anchor; B) animation toward anchor. Black arrows, activated without anchor: C) rotation about camera; D) x-y pan. Anchor-independent: E) Move inward/outward; F) precomputed views.

polyline tool, which measures an area and a perimeter defined by a polyline, performs a minimum-area triangulation of the polyline and calculates the area by summing up all triangle areas. Other tools include clipping and the possibility of drawing axis-aligned grids (see figure 4.3).

4.3 Browser Web Plugin

The viewer can also be used as a remote exploration tool through a web plugin to facilitate dissemination of huge 3D models, or can be placed in a museal kiosk installation next to a real 3D artwork to improve museum visitor's experience. The plugin has been implemented by using the Nokia Qt API. In particular with the QtBrowserPlugin framework we managed to build a browser plugin that can be used in Mozilla FireFox, Safari, Opera, Google Chrome, QtWebKit and any other web browser that supports the "Netscape Plugin API", NPAPI. The Windows version is a single DLL file and can easily be installed in all popular browsers on Windows; the Unix is instead a single static object library file (.so).

Javascript can also be used for user interaction, to open a particular model or even select the desired view directly with browser components. Moreover, shaders can be installed, in the same directory of the plugin library file, to allow a better rendering quality on supported GPU.

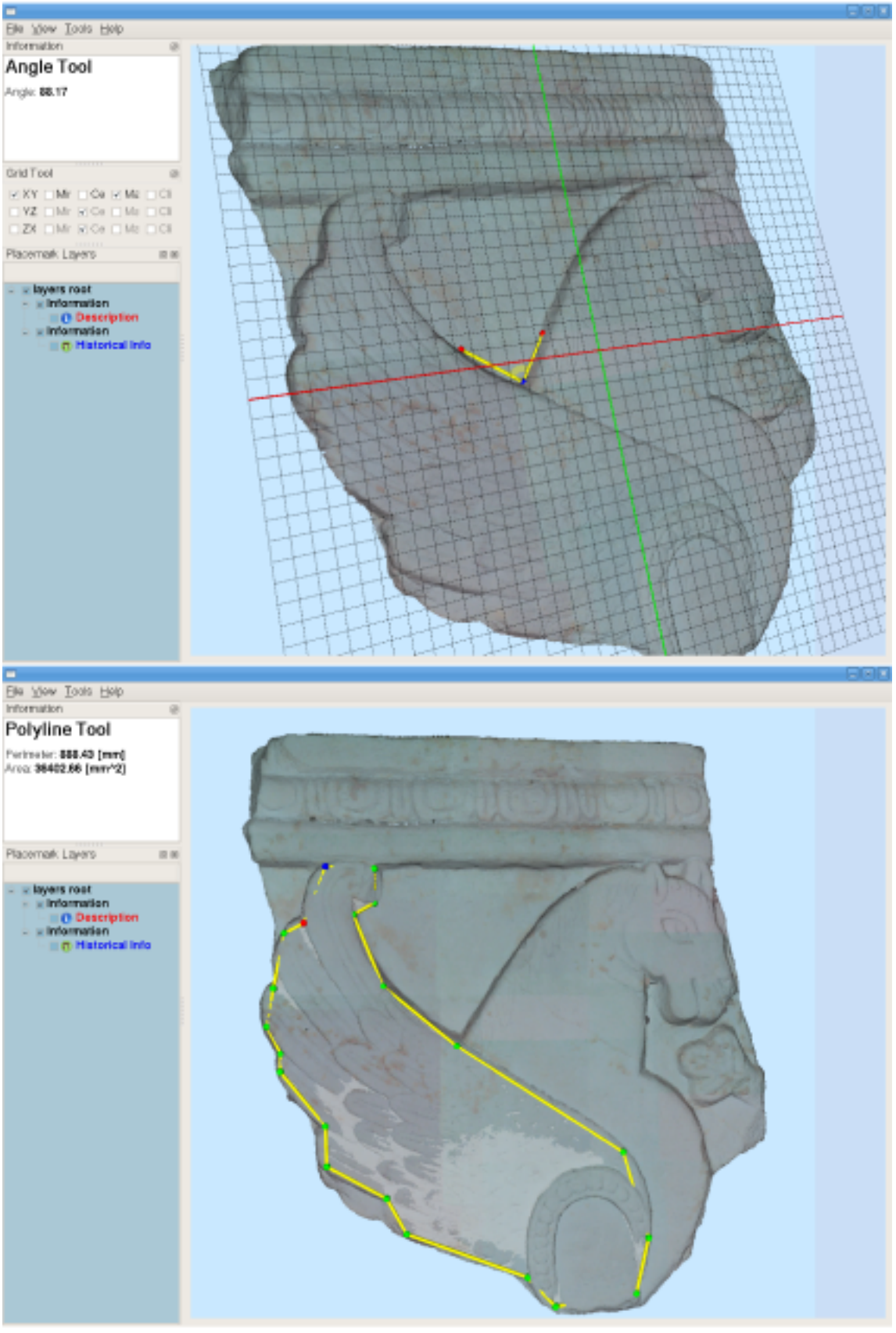

Figure 4.3: Cavallo Alato Bas-relief from Sant'Antioco Basilica. Angle tool and axis-aligned grid. Poly- line tool for area and perimeter measurement.

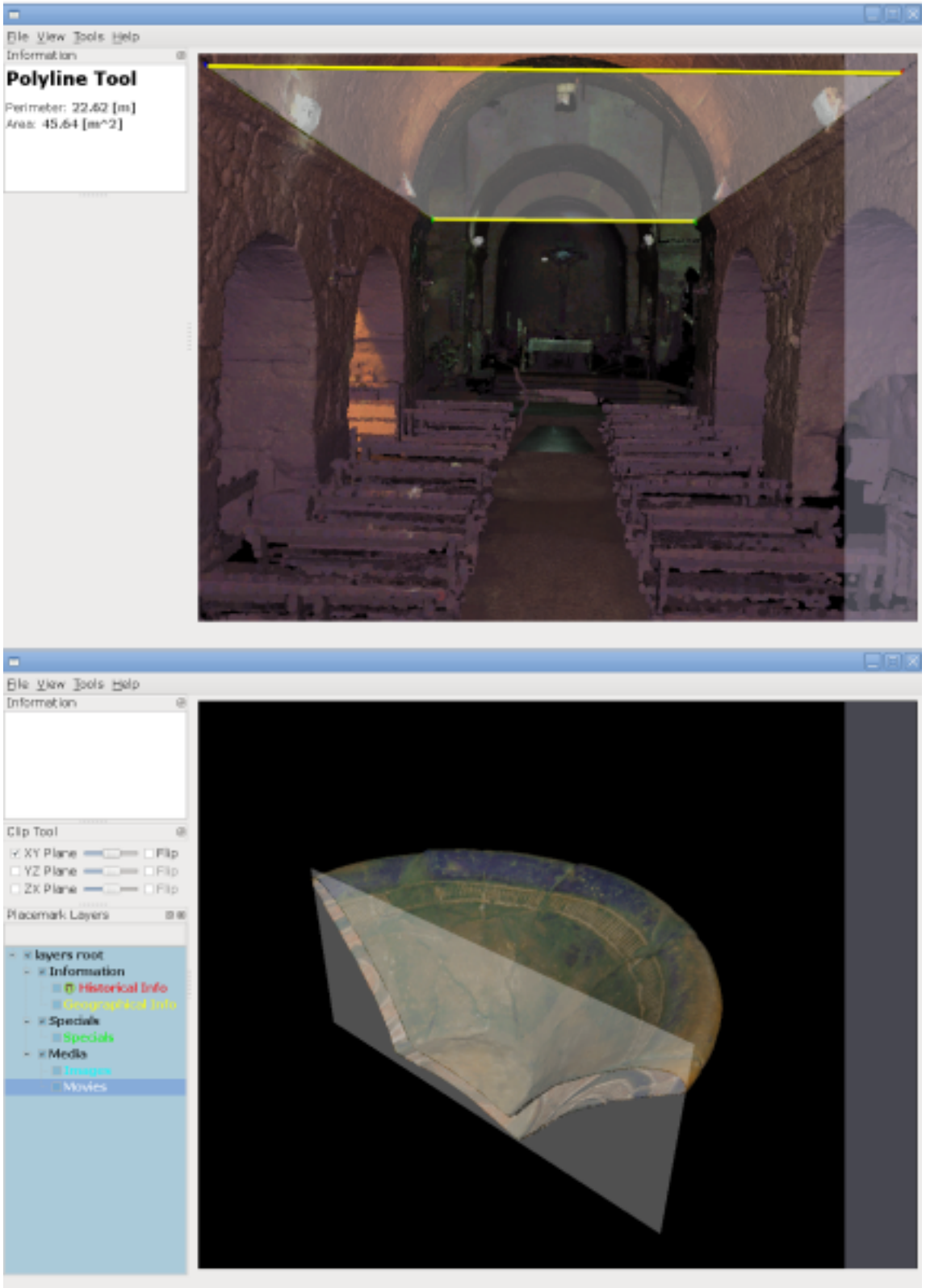

Figure 4.4: Model Inspection Sant'Antioco Basilica inspection with the polyline tool for area measurement. Clip plane tool on an ancient bowl.

Chapter

5 Results and Conclusions

Two case studies are described here: the Sant'Antioco Cathedral (see figure 5.4) made of 37M samples, which has been acquired with a Leica ScanStation2, and a detail of the same cathedral: the Cavallo Alato bas-relief (8M samples), acquired with a Minolta Vivid-9i. The cathedral was digitalized in two one-day sessions by three people, while the Cavallo Alato was acquired in about two hours, by the same group of people. Both models were acquired under a project in cooperation with Roberto Coroneo, professor of Medieval Art History at University of Cagliari, who also produced the multimedia information related to the two models.

5.1 Laser Scanning Technologies Overview

Tridimensional laser scanners devices analyze a real-world environment or object to acquire its geometry and appearance (i.e. color); the collected data is typically a colored point cloud that can be used to digitally visualize 3D models. Many different technologies are used to build these advanced scanning devices, each with a different cost, acquisition speed, precision and measurement error. Non-contact 3D scanners can be divided into two main categories, active scanners and passive scanners; passive devices detect the reflected ambient radiation, instead the active one, like laser scanners, emit some kind of radiation or light and detect its reflection in order to probe a model or surrounding environment.

In the following sections can find some detail of the two particular kinds of laser scanners we used to acquire several models and environment well suited to be rendered with our system.

5.1.1 Triangulation-based Systems

These 3D scanners are so called due to their use of the principle of triangulation. That is, typically a thin stripe of laser light is projected on the surface of an object and is detected by

a digital camera. Being the laser emitter and camera positions fixed and known, geometry can be simple computed along the laser stripe in 3-dimensions.

Triangulation-based laser scanners typically have a really high precision making them ideal for accurately recording fine details on sculptured stonework.

Furthermore, the high accuracy also allows to directly measure changes in the surface of the artifact either caused by decay, or maybe even vandalism. The flip-side is, of course, that extremely large datasets can be generated, but this has already been exploited by our framework.

The Konica Minolta The VIVID 9i is the scanner we used to acquire several medium size artifacts. It requires only 2.5 seconds per scan to acquire accurate 3D data and has different scan ranges, from 0.6 up to 2.5 meters. Tele, middle and wide lenses can be selected to accommodate the size of the measurement target; the accuracy is ± 0.05 mm using tele lens at distance of 0.6 m and the precision on the z-axis at the same distance is 0.008 mm.

Figure 5.1: Minolta Vivid 9i. Triangulation Laser Scanner.

5.1.2 Time-of-flight Systems

Time-of-flight scanners operate on a different set of principles to triangulating scanners. A time-of-flight scanner simply shoots a laser pulse at the object and then measures the time taken for the pulse to return to the scanner. Given that the speed of light is constant, the distance from the scanner to the surface of the object can be calculated quite easily. The scanner's motors move the laser emitter backwards and forwards across the object shooting a laser pulse out at regular time intervals. The 3-dimensional points are calculated as a combination of the horizontal and vertical angles of the motors plus the measured distance.

Time-of-flight scanners cannot be used for recording fine detail, such as the small artifacts, but can give great results when acquiring really wide areas.

The Leica ScanStation 2 is the device we used to acquire large environments, like the St. Antioco Basilica. With a maximum instantaneous scan speed of 50,000 pts/sec and a detection range up to 300 m, this device permitted us to acquire the church's indoor in a few hours of work.

Figure 5.2: Leica ScanStation. Time-of-Flight Laser Scanner.

5.1.3 Main differences

Triangulation and time-of-flight range scanners both have pros and cons that make them suitable for different situations. An advantage of time-of-flight range devices is that they are capable of operating over quite long distances, up to kilometers. These scanners are thus can be used to acquire large structures like buildings or geographic features. The flip-side of using time-of-flight range finders is their precision. Due to the high speed of light, detecting the round-trip time is difficult and consequently the distance measurement's accuracy is relatively low, on the order of millimeters.

Triangulation range scanners are exactly the opposite. Their accuracy is relatively high, on the order of tens of micrometers, but have a limited range of some meters.

Furthermore, time of flight scanners accuracy can be lost when the laser hits the edge of an object since the information sent back to the device comes from two different locations for one laser pulse. For a point that has hit the edge of an object, the coordinate relative to the scanners position will be computed based on an average and therefore the point will be positioned in the wrong place.

With a high resolution scan on an model the chances of the laser ray hitting an edge are higher and the resulting data will have noise right behind edges . This issue can be solved by using scanners with a smaller beam width, but the operating range will decrease as the beam width will increase over distance. Software can also be very useful, indeed we can determine when the first object to be hit by the laser ray should cancel out the second.

With a time-of-flight devices, an high resolution scans with millions of samples can take several dozen of minutes; this can create distortion from motion. Being each point sampled at a different time, any motion in the subject or the scanner will distort the collected data. Thus, it's usually necessary to mount the scanner on stable platforms and minimize vibration. Moreover, some modern scanners, can automatically perform multiple scans at different times of the same area, in order to exclude object in motion like people or vehicles.

5.2 Test Results

The framework has been implemented in C++ and OpenGL on a standard Linux PC. Tests have been performed on an Intel Core2 Quad CPU Q6600 2.40GHz - RAM 4GB running Gentoo Linux 2.6.24, with nVidia GeForce 9800 GX2 - 1GB, and a SATA2 hard-drive 500GB. Rendering tests were also performed on a lower performace laptop Intel Core2 CPU T7200 2.0 GHz RAM 2GB with GeForce Go 7400 graphic board, running a Gentoo Linux 2.6.25 distribution, and connected with a standard 7Mb ADSL. The cathedral was preprocessed in 2h47m, producing a BerkeleyDB database of 439MB. The bas-relief was

preprocessed in 40m, producing 111MB. It should be noted that these figures include BerkeleyDB overhead.

Figure 5.3: Sant'Antioco Basilica. The Sacred Heart of Jesus Statue and Altar.

Rendering of the two models has been tested on the two PCs in a variety of situations, on a window of 1000x900 pixels, with pixel tolerance 2. On the high-end PC frame rates exceed on average 100 fps with a mean throughput of about 110Msplat/sec. On the lower-end PC average frame rates are 25 fps, and go down to 5 fps for an extreme closeup in the cathedral environment, with an average throughput of 8Msplat/sec. The system is fully usable in a standard remote settings. The measured bandwidth on the ADSL link was 1.3Mbps, which enables loading full refined views from scratch in few seconds. Thanks to our compressed representation, this bandwidth permits to upload about 47K point samples per second. The user interface has been found profitable and easy-to-use from different user kinds, ranging from new-to-3D users, to Cultural Heritage scholars, to 3D graphics experts. In particular, the single-touch interface has proven easy to learn even without any explanation. Figure 4.1 illustrates a simple interaction sequence using a wall-mounted touch screen.

5.3 Scanning Activity

Many models have been scanned and digitalized with scanning devices described above but two in particular have been tested to show our framework performance with different kind of inspection, that are the large environment navigation and middle sized hi-detailed object exploration. To demonstrated and test navigation and interactive rendering performance we've choosen the Palaeo-Christian Basilica of Sant'Antioco model; for the middle sized object test we've instead taken the Cavallo Alato Bas-relief. In the table of the figure 5.5 will find some detail about the acquisition campaign and building/rendering tests performed on an Intel Core2 Quad CPU Q6600 2.40GHz - RAM 4GB running Gentoo Linux 2.6.24, with nVidia GeForce 9800 GX2 - 1GB, and a SATA2 hard-drive 500GB.

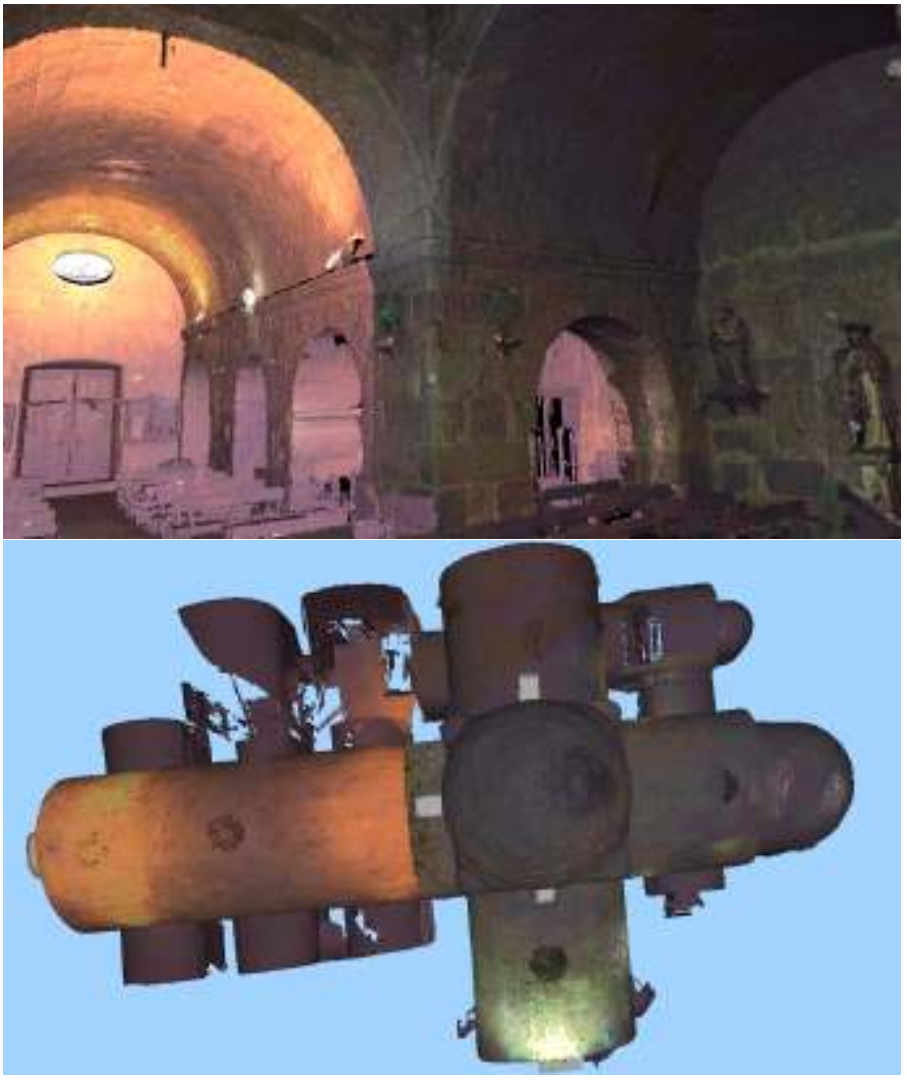

Figure 5.4: Sant'Antioco Basilica. Gateway and Cathedral Plant.

Model:	**Sant'Antioco Basilica**	**Cavallo Alato bas-relief**
Device:	Leica Scanstation 2	Minolta Vivid i9
Technology:	Time-of-Flight	Triangulation Based
Model Size:	18.95m x 28.8m x 21.5m	38.8cm x 41.6cm x 5.7cm
Tot Points Count:	36.6 millions	8 millions
Average Resolution:	6mm	0.15mm
Raw Data Size:	1812MB	314MB
Compressed Multires. Structure Size:	312MB	217MB
Scans Total Time:	1.5 days	2/3 hours
Scansions Count:	10	20/25 (Tele Lens)

Figure 5.5: Main Scanning Activities and Performed Test Results. Tests have been performed on an Intel Core2 Quad CPU Q6600 2.40GHz - RAM 4GB running Gentoo Linux 2.6.24, with nVidia GeForce 9800 GX2 - 1GB, and a SATA2 hard-drive 500GB.

5.4 Conclusions

We presented a distributed system for exploring massive 3D models. It supports streaming and rendering of very large datasets, multiple multimedia information layers, and simple measurement tools. Our future work includes setting up a publicly available web server to disseminate our 3D acquisition repository, as well as setting up multimedia kiosks to support on-site inspection of 3D artworks.

Figure 5.6: Montessu Necropolis. Real environment and digital representation comparison.

Figure 5.7: San Saturnino Basilica. The Jesus statue: real picture versus digital representation.

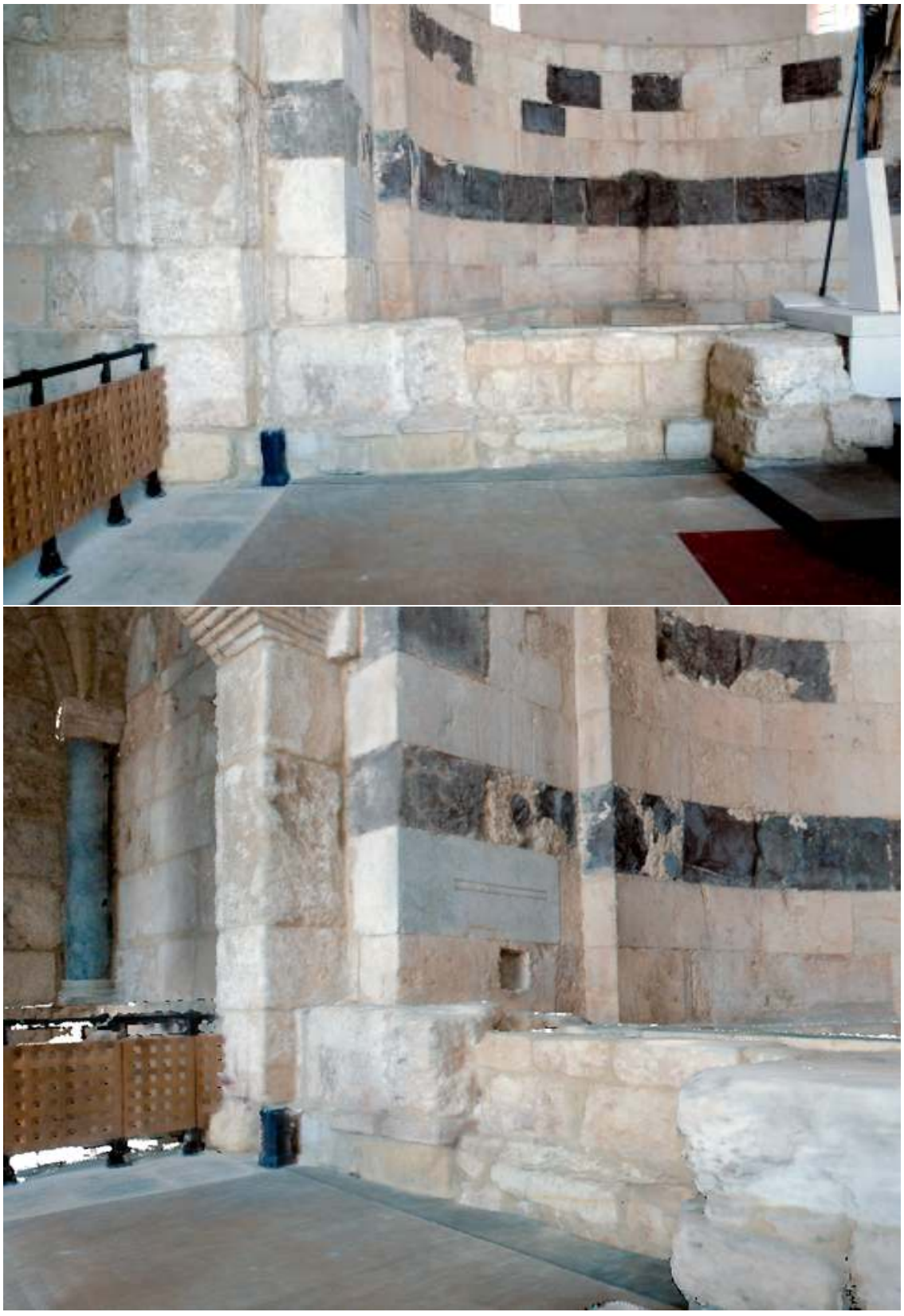

Figure 5.8: San Saturnino Basilica. Altar: real picture versus digital representation.

References

[ABCO*01] Alexa M., Behr J., Cohen-Or D., Fleishman S., Levin D., Silva C. T.: **Point set surfaces**. In *Proc. IEEE Visualization* (2001), pp. 21–28. 6

[BK03] Botsch M., Kobbelt L.: **High-quality point-based rendering on modern GPUs**. In *Proc. Pacifig Graphics* (Oct. 2003), pp. 335–343. 6, 17

[CGG*04] Cignoni P., Ganovelli F., Gobbetti E., Marton F., Ponchio F., Scopigno R.: **Adaptive TetraPuzzles – Efficient Out-of-core Construction and Visualization of Gigantic Polygonal Models**. *ACM Trans. Graph. 23*, 3 (August 2004). 2

[CGM*09] Cuccuru G., Gobbetti E., Marton F., Pajarola R., Pintus R.: **Fast low-memory streaming MLS reconstruction of point-sampled surfaces**. In *Graphics Interface* (May 2009), pp. 15–22. 7

[CPCS08] Callieri M., Ponchio F., Cignoni P., Scopigno R.: **Virtual Inspector: a flexible visualizer for dense 3D scanned models**. *IEEE Computer Graphics and Applications 28*, 1 (Jan.-Febr. 2008), 44–55. 5

[DVS03] Dachsbacher C., Vogelgsang C., Stamminger M.: **Sequential point trees**. *ACM Trans. Graph. 22*, 3 (2003), 657–662. 5

[Eli75] Elias P.: **Universal Codeword Sets and Representations of the Integers**. *IEEE Trans. Inform. Theory 21*, 2 (Mar. 1975), 194–203. 10

[FMM*08] Fitzmaurice G., Matejka J., Mordatch I., Khan A., Kurtenbach G.: **Safe 3D navigation**. In *Proc. Symposium on Interactive 3D Graphics and Games* (2008), pp. 7–15. 6

[GM04] Gobbetti E., Marton F.: **Layered Point Clouds**. In *Proc. Eurographics Symposium on Point Based Graphics* (2004), pp. 113–120,227. 5

[Gro98] Grossman J. P.: *Point Sample Rendering*. Master's thesis, Dept. of Electrical Engineering and Computer Science, MIT, 1998. 5

[HSB*08] Havemann S., Settgast V., Berndt R., Eide O., Fellner D. W.: **The Arrigo Showcase Reloaded toward a sustainable link between 3D and semantics**. In *Proc. VAST* (Dec 2008), pp. 125–132. 5

[LW85] Levoy M., Whitted T.: *The use of points as a display primitive*. Tech. Rep. TR 85-022, University of North Carolina at Chapel Hill, 1985. 5

[MS03] Malavar H., Sullivan G.: **YCoCg-R: A color space with RGB reversibility and low dynamic range**. In *JVT ISO/IEC MPEG ITU-T VCEG*, no. JVT-I014r3. JVT, 2003. 10

[ND06] Niccolucci F., D'Andrea A.: **An Ontology for 3D Cultural Objects**. In *Proc. VAST* (Oct. 2006), pp. 203–210. 4

[Paj05] Pajarola R.: **Stream-Processing Points**. In *Proc. IEEE Visualization* (2005), p. 31. 7

[PGK02] Pauly M., Gross M., Kobbelt L. P.: **Efficient Simplification of Point-Sampled Surfaces**. In *Proc. IEEE Visualization* (Oct. 27– Nov. 1 2002), pp. 163–170. 7

[PSL05] Pajarola R., Sainz M., Lario R.: **XSplat: External Memory Multiresolution Point Visualization**. In *Proc. VIIP* (2005), pp. 628–633. 5

[PZvBG00] Pfister H., Zwicker M., van Baar J., Gross M.: **Surfels: Surface Elements as Rendering Primitives**. In *Proc. SIGGRAPH* (2000), pp. 335–342. 6

[RBN] Richard Beacham H. D., Niccolucci F.: **London Charter**. London charter initiative. 4

[RL00] Rusinkiewicz S., Levoy M.: **QSplat: A Multiresolution Point Rendering System for Large Meshes**. In *Proc. SIGGRAPH* (July 24-28 2000), pp. 343–352. 5, 6, 11

[RL01] Rusinkiewicz S., Levoy M.: **Streaming QSplat: A Viewer for Networked Visualization of Large, Dense Models**. In *Proc. Symposium on Interactive 3D Graphics* (2001), pp. 63–68. 5

[RPZ02] Ren L., Pfister H., Zwicker M.: **Object Space EWA Surface Splatting: A Hardware Accelerated Approach to High Quality Point Rendering**. *Computer Graphics Forum 21*, 3 (Sept. 2002), 461–470. 6

[SD01] Stamminger M., Drettakis G.: **Interactive Sampling and Rendering for Complex and Procedural Geometry**. In *Rendering Techniques* (2001), pp. 151–162. 5

[SLDJ04] Senecal J. G., Lindstrom P., Duchaineau M. A., Joy K. I.: **An improved N-bit to N-bit reversible Haar-like transform**. In *12th Pacific Conference on Computer Graphics and Applications* (Oct. 2004), pp. 371–380. 10

[WFP*01] Wand M., Fischer M., Peter I., auf der Heide F. M., Strasser W.: **The Randomized z-Buffer algorithm: Interactive Rendering of Highly Complex Scenes**. In *Proc. SIGGRAPH* (2001), pp. 361–370. 5

[WS06] Wimmer M., Scheiblauer C.: **Instant Points**. In *Proc. Symposium on Point-Based Graphics* (July 2006), pp. 129–136. 5

[ZF99] Zeleznik R., Forsberg A.: **UniCam: 2D Gestural Camera Controls for 3D Environments**. In *Proc. Symposium on Interactive 3D Graphics* (1999), pp. 169–173. 6

[ZPvBG01] Zwicker M., Pfister H., van Baar J., Gross M.: **Surface Splatting**. In *Proc. SIGGRAPH* (2001), pp. 371–378. 6