The 10th International Symposium on Virtual Reality, Archaeology and Cultural Heritage VAST (2009)
K. Debattista, C. Perlingieri, D. Pitzalis, and S. Spina (Editors)

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

Fabio Bettio, Enrico Gobbetti, Fabio Marton, Alex Tinti, Emilio Merella, Roberto Combet

Visual Computing Group, CRS4, Pula, Italy – `http://www.crs4.it/vic/`

Abstract

We present a client-server framework for network distribution and real-time point-based rendering of large 3D models on commodity graphics platforms. Model inspection, based on a one-touch interface, is enriched by a bidirectional hyperlink system which provides access to multiple layers of multimedia contents linking different parts of the 3D model many information sources. In addition to view and light control, users can perform simple 3D operations like angle, distance and area measurements on the 3D model. An authoring tool derived from the basic client allows users to add multimedia content to the model description. Our rendering method is based on a coarse grained multiresolution structure, where each node contains thousands of point samples. At runtime, a view-dependent refinement process incrementally updates the current GPU-cached model representation from local or remote out-of-core data. Vertex and fragment shaders are used for high quality elliptical sample drawing and a variety of shading effects. The system is demonstrated with examples that range from documentation and inspection of small artifacts to exploration of large sites, in both a museum and a large scale distribution setting.

Categories and Subject Descriptors (according to ACM CCS): Computer Graphics [I.3.3]: Picture and Image Generation—; Computer Graphics [I.3.7]: Three-Dimensional Graphics and Realism—.

1. Introduction

Today's 3D laser scanning and digital photography systems make it possible to rapidly acquire multi-million samples 3D textured models at sub-centimetric resolution, providing detailed multi-gigabyte descriptions of real 3D objects. Cultural heritage particularly benefits from these technologies, for archiving, restauration and dissemination purposes. Beyond the possibility of visiting a site, which is not always within everyone's reach, the ability to interactively browse a 3D model from different points of view provides an invaluable tool to experience and understand an artwork. Sometimes this virtual approach can also enhance on-site experiences, when, as the case with a large statue like Michelangelo's David, many point of views are not directly visible to the ordinary visitor. The virtual knowledge experience can be further enriched if additional layers are available contextually to 3D interaction, supplying a way to move from/to a particular 3D hot spot to textual, visual, and other multimedia data.

The 3D scanning process is able to produce at high speed millions up to billions of samples, and this information needs an intensive, not completely automated, processing to produce a final high quality triangulated textured mesh. Generally this mesh is heavily simplified before being disseminated to allow unspecialized end users to browse a streamable, relatively small 3D model, thus hindering the quality of the original high resolution scan. In fact, despite the rapid improvement in graphics hardware performance, rendering at interactive speeds the multi-gigabyte datasets generated by the acquisition pipe-line remains a very challenging problem, since they still largely overload the performance and memory capacity of state-of-the-art graphics, computational and networking resources. For such complex and dense models, multiresolution hierarchies of point primitives have in recent years emerged as a viable alternative to the more traditional mesh refinement [CGG*04]. These methods are based on the assumption that model sampling rate is so high that triangles are projected onto such small screen areas that the advantages of scanline coherence are lost, and appropriately selected point samples are sufficient to accurately reproduce the model. One of the major benefits of this approach is its simplicity, stemming from the fact

DOI: 10.2312/VAST/VAST09/025-032

that there is no need to explicitly produce and maintain mesh connectivity during both preprocessing and rendering.

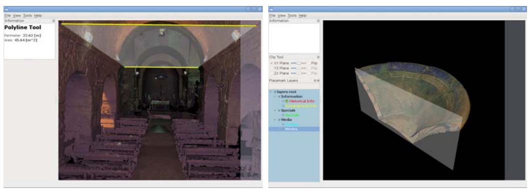

Figure 1: ***Model Inspection*** *Sant'Antioco Basilica inspection with the polyline tool for area measurement. Clip plane tool on an ancient bowl.*

Our contribution is 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 our 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 implemented client-server framework is 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. Moreover, a potentially extensible tool set improves model exploration by providing basic measurement functionalities to get quantitative informations like distances, areas and angles (see figure 1). A user interface, 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. 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.

2. Related work

Cultural heritage 3D inspection applications, as proposed by the London Charter [RBN], should promote transparency, adding sufficient information to allow computer-based visualisation methods to be understood and evaluated in relation to the contexts and purposes for which they are developed. Niccolucci et al. [ND06] demonstrated the integration of 3D into semantic databases based on the X3D format. Havemann et al. [HSB*08] proposed an application similar to the previous one, but based on Collada, which is able to visualize 3D models with updatable textual content similar to hyperlink and link anchor. This application does not solve the problem of visualizing huge unsimplified 3D models, as the ones that can be obtained by 3D laser scanner. Virtual Inspector [CPCS08] is a system that allows naive users to inspect a very dense 3D model at interactive frame rates, presenting the 3D model and all the multimedia content that has been linked to selected points of its surface. Our system is similar to Virtual Inspector, but is centered on a point-based multiresolution hierarchy rather than a triangulated one, thus requiring less preprocessing of acquired data. Moreover, we tackle both local and remote model inspection.

Point-based 3D graphics techniques for processing and rendering of dense models are an old idea [LW85, Gro98]. QSplat [RL00, RL01] has for long been the reference system in this particular area. The system is based on a hierarchy of bounding spheres maintained out-of-core, locally or remotely, which is traversed at run-time to generate points. This algorithm is CPU-bound, because all the computations are made per point, and CPU/GPU communication requires a direct rendering interface; so the graphic board is never exploited at its maximum performance. A number of authors have proposed various ways to push the rendering performance limits in particular situations, in most cases using a retained-mode interface working with blocks of points [WFP*01, SD01, DVS03, GM04, PSL05, WS06]. Layered Point Clouds [GM04] was the first system to propose coarse grained multiresolution approach, which 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 builds on Layered Point Clouds, but while they use finer hierarchy levels to increase resolution of the coarser representation, so that their representation is made 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. There is a large body of work that aims at improving the rendering quality of point-sampled models. For dense models, it is 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]. In our work, we provide two rendering modes: a simple one based on OpenGL smooth points for basic graphics cards, and a second one which uses a vertex and a fragment shader to perform elliptical splatting as proposed in [BK03]. Tone mapping and screen-space ambient occlusion are also available in the rendering pipe-line.

Systems which allow new-to-3D users to inspect 3D models need to provide an intuitive and easy-to-use interface. A great body of research has studied related issues. Our single-touch interface is similar in spirit to Unicam [ZF99], while offering additional possibilities that cover both ambient exploration and model manipulation. In addition, we integrate our direct camera manipulation interface with a system for selecting predefined views, as, done, e.g., in the recent Safe 3D navigation interface [FMM*08].

3. Data representation

In high quality real-time networked model viewing applications, the client has to rapidly adapt a visually seamless level-of-detail representation in response to viewer motion, using data coming from the server. In this context, a compressed data representation must possess a number of requirements: ability to support random access to different portions of the dataset; resolution scalability for progressive resolution increase; hierarchical structuring able to support variable resolution seamless LOD; high compression ratio, to reduce transmitted data; low server side complexity, to ensure scalability with a number of clients; low client-side decoding complexity, to support heterogeneous clients. We meet these requirements by proposing a data representation based on a coarse-grained multiresolution point hierarchy. We also need to associate the 3D model framework to a service able to provide different layers of multimedia information which can be related to the model through a set of 3D hot spots. This information is edited using a specialized version of the end-user viewer, and is then made available through the layer service.

3.1. Stream processing points

We assume that the input model is represented by a set of N sample points with associated attributes including position, normal and possibly color and radius information. In a preprocessing step we perform data prefiltering, radius computation and other operations 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. This rotation reduces 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].

3.2. Multiresolution model construction

Our multiresolution 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. The multiresolution point-cloud structure is constructed off-line, starting from a generic point cloud model. We have implemented a simple I/O efficient recursive clustering method, which is realized on top of a single out-of-core component: a standard C++ array (compatible with `std::vector`), which encapsulates a resizable file accessed through system memory mapping functions. The procedure consists of two phases.

The first phase partitions the input dataset into a kd-tree of point clouds. The partitioning procedure takes as input an external memory array of uniformly distributed point samples, together with its bounding volume. If the number of points N is less than the predefined node sample count M, a leaf node is generated, otherwise samples are distributed to the two children by bisecting the bounding box at the midpoint of its longest axis, and recursively continuing the partitioning procedure.

In the second phase we build inner nodes from their children: the parent node contains M samples created by merging children samples. In order to obtain a filtered version, we proceed using an edge collapse simplification procedure adapted to point clouds. Each point is connected to its eight nearest neighbors. Neighbor information is computed by building a local kd-tree with all the samples of the two children, while a map of edges sorted by edge length is used for the simplification. The simplification procedure collapses the two points with the lowest edge length, and then updates all the structures that permit to identify the vertex neighbors and the new edge lengths. This procedure continues until onlyt M samples remain. When merging two samples, their new position is placed at their midpoint. A weight proportional to the sample splat area is used tolinearly combine their colors and normals. The new sample radius is the minimum radius that covers both original samples from the new sample position (i.e., the distance of one input sample from the target position plus the maximum of the two sample radii).

3.3. Compression

Even though multiresolution approach builds a view-dependent model representation for a particular point of view, using only the necessary data for that view, it is advantageous to compress node data to better exploit available network bandwidth, especially in remote settings.

Each node in our data structure contains per-node data and per-point data. The per-node data consists in a flag (indicating whether the node is a leaf or not), the node's bounding box, the radius range (minimum, maximum, and average) and the number of points contained in the associated point cloud. The per-point data consists in the sample attributes (position, normal, color, and radius) stored in parallel arrays. Per-node data is stored and transmitted uncompressed, while we focus on compressing per-point data.

We use a simple wavelet-based compression scheme, which is fast and able to to reduce memory occupancy by almost an order of magnitude. Compression proceeds in the following phases. First, the samples within the point cloud are sorted in an order that minimizes the Euclidean distance among subsequent points, to produce a *point strip*. The attributes of the points 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, at each iteration applying the transform to the low-pass coefficients resulting from the previous iteration, until there is only a single low-pass coefficient remaining. The resulting coefficient are then mapped to positive integers and encoded using a simple Elias gamma code [Eli75], in which a 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. Upon decompression, all the steps are undone in the reverse order (with the obvious exception of sorting).

The quantization used to transform input data to coefficients is adaptive and takes into account local sample spacing. Positions are expressed in relation to the node bounding box and quantized to minum number of bits per component that produce a quantization error less than a quarter the minimum sample radius. Radii are also expressed in relation to radius range, using the same error threshold. Upon decompression, radii are enlarged by the quantization error to ensure that no additional hole is introduced. 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. Since the compression subsystem is used by both the preprocessing and the run-time systems, quantization effects in a given node are taken into account when constructing its parent.

With this compression scheme, we obtain on our test datasets an average compression rate of 3.6 bytes per sample, using on average 7KB bytes for a patch of 2000 samples which would otherwise occupy about 62KB. The compression rates are competitive with those of other point based systems. For instance, QSplat [RL00] uses 5.375bytes/sample to encode the same attributes.

3.4. Information layers

The interaction experience is enriched by the possibility to access further multimedia information of different types. This kind of information is linked to various parts of the 3D models through 3D hot spots which allow the viewer to display textual information. Bidirectional connection is supported: clicking on a hot spot a related web page is shown on a standard web browser, while links from a web page can open the viewer with different models or set a particular point of view. A hierarchical tree structure enables different layers to be activated, selecting various types of information which can be reached through the hot spot interface. Information layers can provide the user not only with historical information, but also with useful scanning annotations used to keep track of the acquisition procedure. When too many hot spots are active, a focus function allows to activate only the ones inside a circle around the current cursor to avoid damaging the overall model perception. A possibly separeted server takes care of deploying these additional informations, which are structured as a hierarchy of xml files.

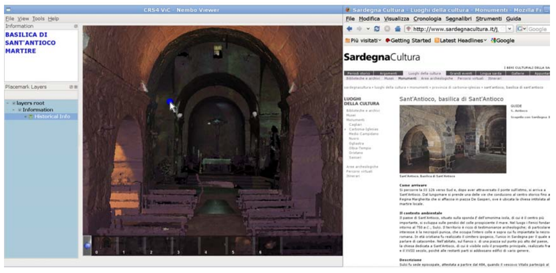
CRS4 ViC - Nembo Viewer
BASILICA DI SANT'ANTIOCO MARTIRE
Placemark Layers
SardegnaCultura
Sant'Antioco, basilica di Sant'Antioco

Figure 2: ***Sant'Antioco Basilica.*** *Information layer structure, hot spot, and web browser additional information.*

4. Client server framework

Our compressed multiresolution point cloud representation forms the basis of a scalable 3D model streaming system able to adapt to client characteristics and to exploit available network bandwidth. A block diagram of our system's design is presented in figure 3. In the following sections, we discuss the data storage and server side distribution components, as well as the client-side streaming and rendering methods.

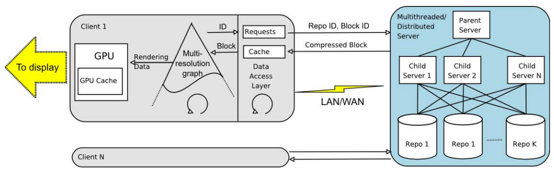

Figure 3: ***Client-server architecture.***

Server design and implementation. The server is able to manage different repositories. From the server's point of view, a repository is just a database with a unique key for indexing a block of bytes containing an encoded bitstream which represents a compressed node point cloud. In order to increase server-side scalability, no component is present in the server, whose only action is to return a block of bytes if present. This approach makes it possible to leverage existing database components instead of forcing us to implement a storage manager. In this work, storage management is done through Berkeley DB, and data serving is done through an Apache2 server extended with an appropriate module.

Berkeley DB is a widely tested open source embeddable database which provides a fast, scalable, transactional database engine with industrial grade reliability and availability, able to manage up to terabytes of data. Moreover, the amount of per-process replicated cache is configurable, and different instances of the same database are able to share index memory, thus reducing memory load for servers dealing in parallel with hundreds of clients. Apache2 is a secure, efficient and extensible open source server that provides many 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.

A custom apache module manages a connectionless protocol based on HTTP. Clients requests include database name and node identifiers. The node identifier is a single unsigned int value which addresses the node as if it stored in a full binary tree. The module parse the request, queries the associated database, and sends in response either a compressed bitstream extracted from the database encoding a point cloud, or an empty message if no data is available.

This approach based on open-source components is simple to implement and maintain, and provides very good performance. In particular, the concurrency features of apache are exploited to handle thousands of clients in parallel through the forking of multiple child servers sharing the same database.

4.1. Multi-threaded clients for remote visualization

Incremental view-dependent refinement. The client manages a binary graph which 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 multiresolution model from the current point of view, is based on a stateless coarse-to-fine refinement of our structure, which exploits the progressive nature and coarse granularity of the multiresolution hierarchy to reduce CPU refinement costs and to improve repository-to-host and host-to-graphics communication. In particular, asynchronous repository requests hide out-of-core data access latency, and communication with the GPU is made exclusively through a retained mode interface, which reduces bus traffic by managing a least-recently-used cache of point clouds maintained on-board as OpenGL Vertex Buffer Object.

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 performs a single-pass recursive traversal of the multiresolution structure and 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 4).

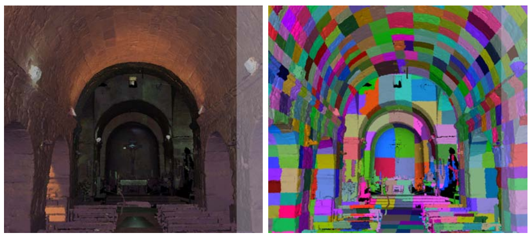
Figure 4: ***Sant'Antioco Basilica.*** *Patch refinement: 1665 patches with a total of 1.3M samples, rendered at 100 fps on a 1024x768 window, with elliptical splats enabled.*

Multithreaded data access layer. A multithreaded data access layer hides from the application the technique used for accessing the terrain 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, while keeping 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 order in which they were requested. The act of pipelining the requests can result in a dramatic improvement in response times, 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 which 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/first-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 managing the network receives the compressed bitstream from the server and stores them in the shared cache.

Rendering process. At the end of the incremental refinement process, the leafs of the graph 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 comunication if no Vertex Buffer Objects are available, and it is based on the standard Vertex Arrays which are provided since OpenGL 1.1. We support two rendering modes: a first simple straight rendering method which uses a single splat size for each point cloud, and draws circular splats through glPointSmooth. A higher quality representation can be obtained using a vertex and a fragment shader to draw an oriented 3D ellipse, depicting a textured quad for each sample [BK03].

4.2. 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. Our 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.

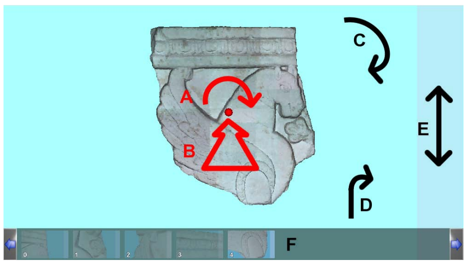

Figure 5: ***Single-touch interaction scheme.*** *Red arrows, activated when an anchor is present: A) rotation about anchor; B) animation toward anchor. Black arrows, activated without anchor: C) rotation about camera; D) x-y pan. Anchor-independent: E) Move inward/outward; F) precomputed views.*

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 5). 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 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 6).

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 polyline tool, which measures an area and a perimeter defined by a polyline, performs a minum-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 7).

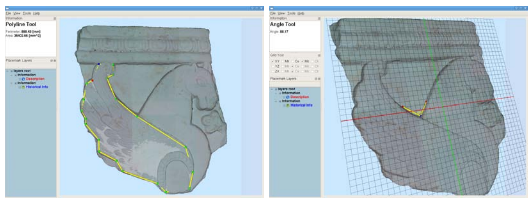

Figure 7: ***Cavallo alato from Sant'Antioco Basilica.*** *Polyline tool for area and perimeter measurement. Angle tool and axis-aligned grid.*

5. Results

Two case studies are described here: the Sant'Antioco Cathedral (see figure 8) 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.

Our 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. 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 6 illustrates a simple interaction sequence using a wall-mounted touch screen.

6. 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.

Acknowledgments. This work was partially supported by Sardegna DISTRICT (P.O.R. Sardegna 2000-2006 Misura 3.13).

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.

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

[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).

[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.

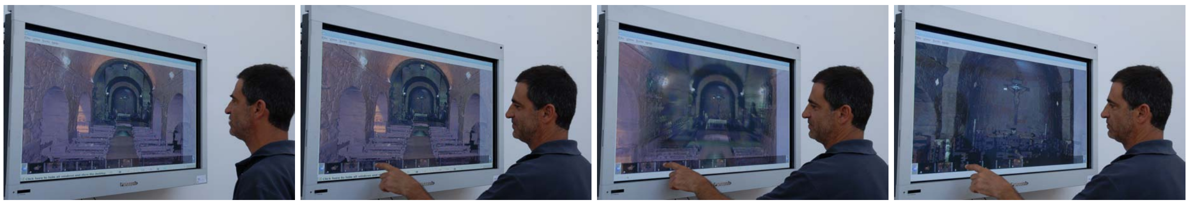

Figure 6: ***Touch screen.*** *Selection and animation moving toward precomputed view.*

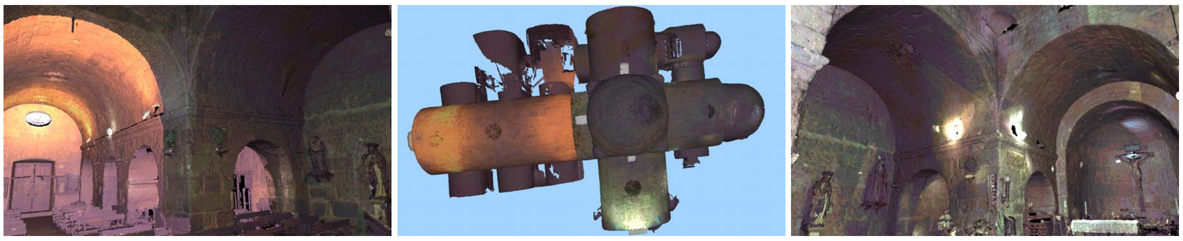

Figure 8: ***Sant'Antioco Basilica.*** *Three views: gateway, cathedral plant, transept with Sacred Heart of Jesus statue and altar.*

[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.

[DVS03] DACHSBACHER C., VOGELGSANG C., STAMMINGER M.: Sequential point trees. *ACM Trans. Graph. 22*, 3 (2003), 657–662.

[Eli75] ELIAS P.: Universal codeword sets and representations of the integers. *IEEE Trans. Inform. Theory 21*, 2 (Mar. 1975), 194–203.

[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.

[GM04] GOBBETTI E., MARTON F.: Layered point clouds. In *Proc. Eurographics Symposium on Point Based Graphics* (2004), pp. 113–120,227.

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

[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.

[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.

[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.

[ND06] NICCOLUCCI F., D'ANDREA A.: An ontology for 3D cultural objects. In *Proc. VAST* (Oct. 2006), pp. 203–210.

[Paj05] PAJAROLA R.: Stream-processing points. In *Proc. IEEE Visualization* (2005), p. 31.

[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.

[PSL05] PAJAROLA R., SAINZ M., LARIO R.: Xsplat: External memory multiresolution point visualization. In *Proc. VIIP* (2005), pp. 628–633.

[PZvBG00] PFISTER H., ZWICKER M., VAN BAAR J., GROSS M.: Surfels: Surface elements as rendering primitives. In *Proc. SIGGRAPH* (2000), pp. 335–342.

[RBN] RICHARD BEACHAM H. D., NICCOLUCCI F.: London charter. London charter initiative.

[RL00] RUSINKIEWICZ S., LEVOY M.: QSplat: A multiresolution point rendering system for large meshes. In *Proc. SIGGRAPH* (July 24-28 2000), pp. 343–352.

[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.

[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.

[SD01] STAMMINGER M., DRETTAKIS G.: Interactive sampling and rendering for complex and procedural geometry. In *Rendering Techniques* (2001), pp. 151–162.

[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.

[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.

[WS06] WIMMER M., SCHEIBLAUER C.: Instant points. In *Proc. Symposium on Point-Based Graphics* (July 2006), pp. 129–136.

[ZF99] ZELEZNIK R., FORSBERG A.: UniCam: 2D gestural camera controls for 3D environments. In *Proc. Symposium on Interactive 3D Graphics* (1999), pp. 169–173.

[ZPvBG01] ZWICKER M., PFISTER H., VAN BAAR J., GROSS M.: Surface splatting. In *Proc. SIGGRAPH* (2001), pp. 371–378.