An Easy-to-use Visualization System for Huge Cultural Heritage Meshes

R. Borgo, P. Cignoni, R. Scopigno
Istituto Scienza e Tecnologie dell'Informazione – Consiglio Nazionale delle Ricerche*

Abstract

The technology for the automatic shape reconstruction evolved rapidly in recent years, and huge mass of data can be easily produced. Due to the supported accuracy, works of art are one of the ideal fields of use of these devices. Given this particular application domain (Cultural Heritage), two issues arise: how to manage these complex data on commodity computers, and how to improve the ease of use of the visualization tools (as potential users are often not expert at all with interactive graphics).
We present a new visualization system that allows naive users to inspect a large complex 3D model at interactive frame rates on off-the-shelf PC's. A main goal in the design of the system has been to provide the user with a very easy and natural interaction approach, based on a straightforward *"point and click"* metaphor. Visualization efficiency is obtained by adopting a LOD representation and on-line automatic selection of the best-fit level of detail (according to the current view frustum).
The system has been used by restores and art curators for the inspection of some high resolution models of statues, and its evaluation was successful.

Keywords: Large mesh visualization, Surface Simplification, LOD representation, Visualization systems.

1 Introduction

Interactive rendering of complex 3D models is important for many applications, such as architectural design, graphics simulation and scientific visualization. These systems need to provide the user with both real-time navigation and a realistic visual representation of the scene. One basic problem in the design of a visualization systems to be used in the Cultural Heritage (CH) field concerns its ease of use: visualization tools should take into account the specific needs of the forecasted users, which usually possess limited skills in managing 3D graphics technology (museum curators, art historians, restorers, ordinary visitors of a museum, etc.). The design of the GUI and its overall usability play a critical role in deciding if the tool itself can be just a nice toy or a truly and completely useful technical instrument.
Another problem is that realistic looking models often contain more graphics primitives than the interactive capabilities of graphics workstations. The focus of our activity is how to process the huge meshes which are produced by the 3D scan of works of art (e.g. statues or any other work of art [6, 11, 12]). Efficient data management and rendering is a central issue in processing huge dataset. Several techniques have been developed to cope with this problem: some of them keep a triangle-based representation and adopt *geometry simplification* and *multiresolution representation* to reduce data complexity and rendering times [5]; others adopt a point-based rendering approach coupled with keen heuristics for dynamic data sub-sampling [13, 9].

The work described in this paper was twofold: first, we designed the interface of a new visualization tool, specifically oriented to the inspection of complex digital representations of works of art, giving priority to the easy-of-use of the system rather then to the flexibility. Second, we have designed its internal architecture by taking into account two contrasting constraints: the system should support huge meshes inspection, and real-time behaviour should be ensured. Therefore, our goals are:

- **Graphics Workstation tailoring**. The tool needs to be designed for inexpensive platforms (PCs with available state of the art graphics boards), to ensure a wide potential user community;
- **User tailoring**. Because a major fraction of the forecasted user community is composed by naive users, interaction with 3D data should to be as easy and natural as possible, avoiding users to feel lost in virtual space while manipulating/inspecting the digital representation;
- **On-line selection of best-fit drawable geometry**. Geometry should be represented at different accuracy, using in each instant of time the accuracy adequate to the current visualization task;
- **Real-time visual update**. When the gaze moves, the mesh regions to be updated have to be located quickly and the update rate should be maintained as low as possible.

* Area della Ricerca CNR, Via Moruzzi 1, 56125 Pisa, ITALY. WWW: http://vcg.iei.pi.cnr.it/ Email: {borgo | cignoni}@iei.pi.cnr.it, roberto.scopigno@cnuce.cnr.it

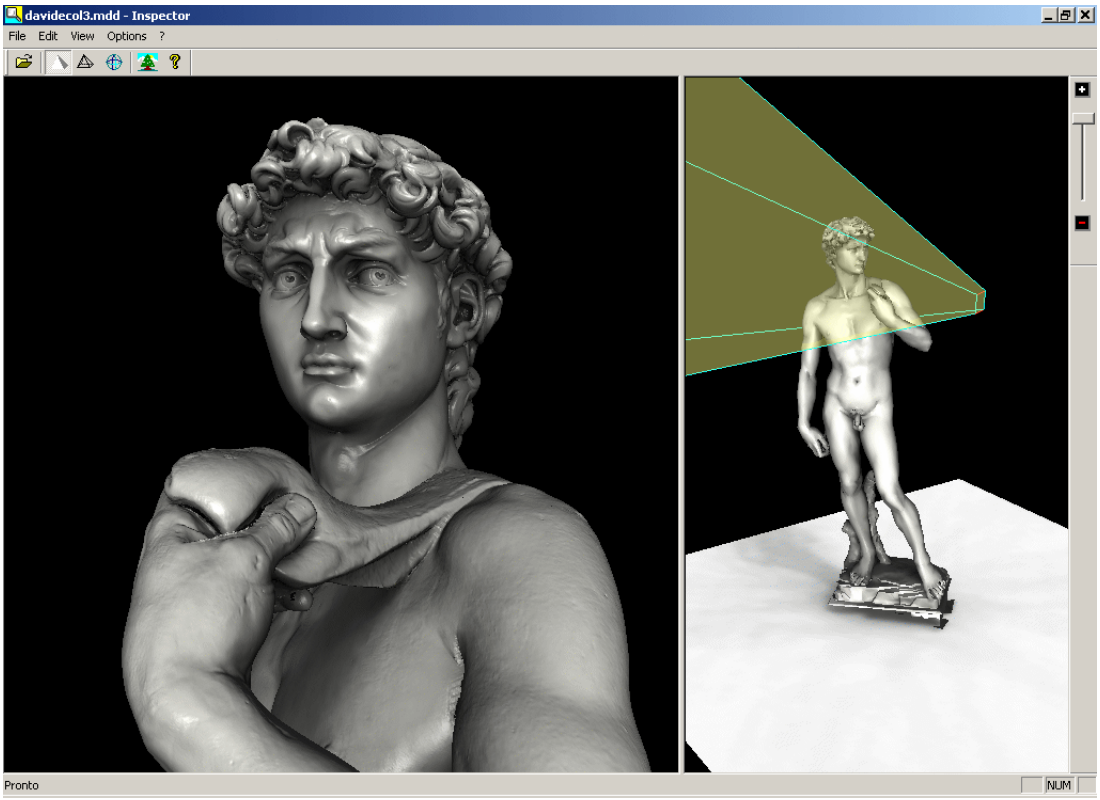

Figure 1: User interface of the *Inspector* tool, showing the David mesh (Digital Michelangelo Project).

- **High Quality rendering**. With the above constraint in mind the rendering quality should be the highest attainable without sacrificing frame rate.

As far concerns the GUI of the system, we decided to structure the user interaction providing a sort of small digital replica of the visualized object. Main scope of this replica (called *dummy* in the following) is to support the selection of the area of the artifact to be rendered (using a very simple *point-and-click* approach) and to mirror in a consistent manner the location the user is currently looking at (in oder to prevent him to get lost in space or to loose orientation). Once the selected region is presented to the user, the visualization tool enables only a constrained set of navigation capabilities. This set includes only commands useful to the visual analysis of the current selected region (panning and rotation are limited to small variations of the standard preselected view specification).

The system architecture has been designed by choosing a triangle-based approach. This is justified by the rather impressive improvements of inexpensive graphics board performances and by recent detail recovery approaches which allows a very efficient representation of detail (either color or bump) on simplified meshes [3, 14]. Given a 3D mesh which represents the given work of art, the digital model is converted into an LOD representation. We preferred an LOD model (composed of 5-8 different levels) to a more flexible multiresolution model, in order to reduce time and space overheads; see Section 2.2 for a detailed description and justification of our LOD choice. The LOD representation is managed following a time-budget approach: the visualization tool must run in real time, and it must support a fixed frame rate. The LOD used in each frame is therefore chosen dynamically by performing an approximate estimation of the size of the mesh section contained in the current view frustum and of the rendering speed of the PC used. Geometric data are encoded in triangle strip format, to improve rendering efficiency.

We have applied our LOD control technique to a few large models (triangle-based meshes) representing Cultural Heritage (CH) artifacts, produced with 3D scanning devices. Some results are reported in the last section.

In Section 2 we give an overview of the architecture of the tool itself and of the gaze-driven approach, in Section 3 we presents some results upon real applications and collected performance of the developed tool followed in Section 3 by some conclusions.

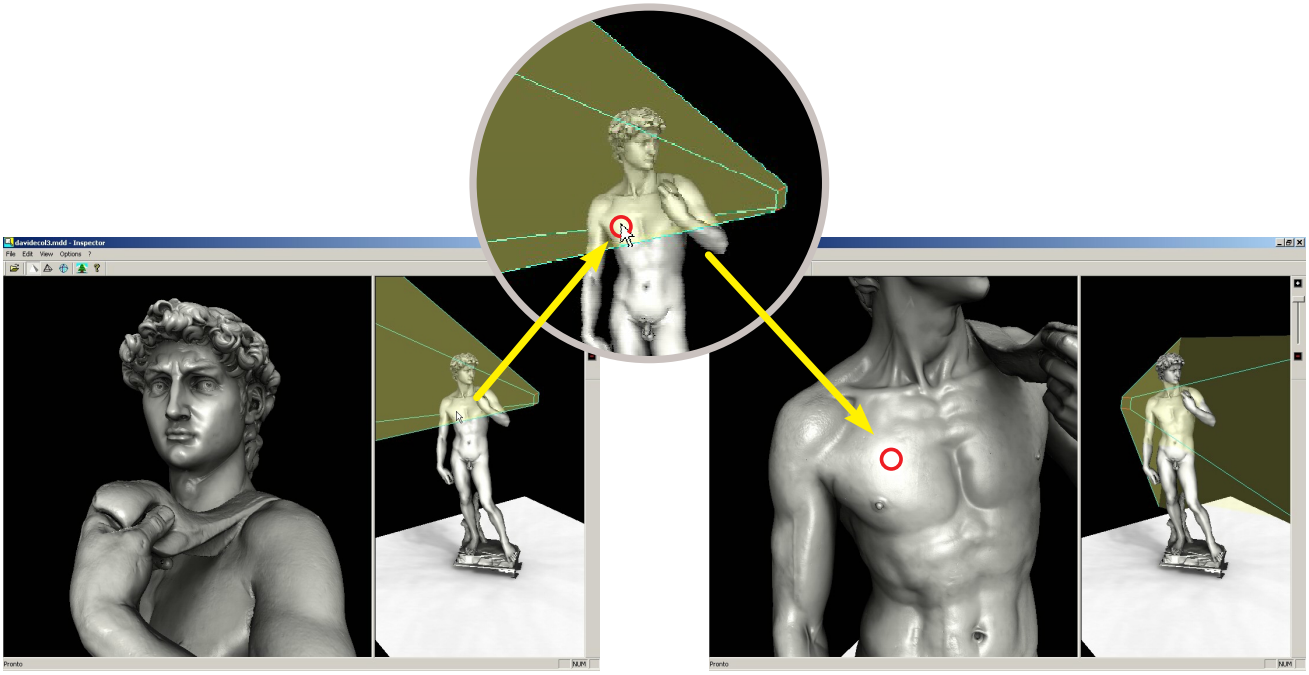

Figure 2: The current gaze point is marked by a red spot in the zoomed image fragment; a new gaze point is selected by a simple mouse click on the corresponding location on the dummy (see the position of the white arrow cursor).

2 The Inspector Tool

To fulfill the above requirements, we have developed a tool which is able to manage huge triangulated polygonal objects with a dynamic LOD selection driven by the user-selected gaze. We describe first the GUI of the tool, leaving all the technicalities regarding efficient visualization and enhanced rendering to the following subsections.

2.1 GUI Design

The layout of the *Inspector* tool is shown in Figure 1. The output window of the tool is divided into two main frames: the one on the right is dedicated to the interactive selection of the desired view and to monitor visually the current view specifications; the one on the left constitutes the main visual output region. The object portion visible at run time in the leftmost frame depends on the viewing parameters that the user selects by means of the rightmost window region. Let us describe how the user can interact with it.

A complete model of the inspected object, called *dummy*, is visualized in the rightmost frame. It is conceived as a sort of interactive 3D map whose role is: to allow an easy selection of the view specs, and to maintain the user clearly located in space. The dummy is always visualized in full view, its appearance is not affected by the main user-selected view, neither its level of detail (the object is visualized in the right-most frame using a fixed low-resolution model, with high frequency detail painted through a bump-texture [3] to improve visual accuracy). The user disposes of a constrained virtual trackball on the rightmost frame, which allows only to rotate the dummy on its vertical axis. In this manner, the user can rotate the dummy and see the dummy from any lateral direction (see Figure 3).

The selection of the viewing parameters is implemented according to a very simple direct manipulation approach. Clicking on any point of the dummy on the rightmost frame (see Figure 2) updates the current view as follows: the point selected on the dummy surface becomes the *gaze point*, and the *view direction* is set by default to be equal to the surface normal in the gaze point. The *field of view* and the *view point* are set initially using default values (an example of these default values is shown in Figure 2, where approximately 25% of the object is in the current view volume), or take the value used/set in the previous interactive action. They can be updated (zooming-in/zooming-out) clicking on the +/- icons or sliding the cursor on the right (see Figure 4).
To be more specific, the *view direction* is set equal to an *average* surface normal computed in a small mesh section centered on the gaze point. This to prevent potential erratic or random values of the view direction in locations where the surface normal changes rapidly (e.g. on rough surfaces, or on highly convex or concave regions). For each selected pixel in view space, a 3*3 or 5*5 sampling kernel is evaluated: for each of these pixels, we compute the corresponding point on the dummy surface, and the best fitting plane to these locations in modeling space gives the *average* surface normal.

Coupled with the dummy is a 3D iconic representation of

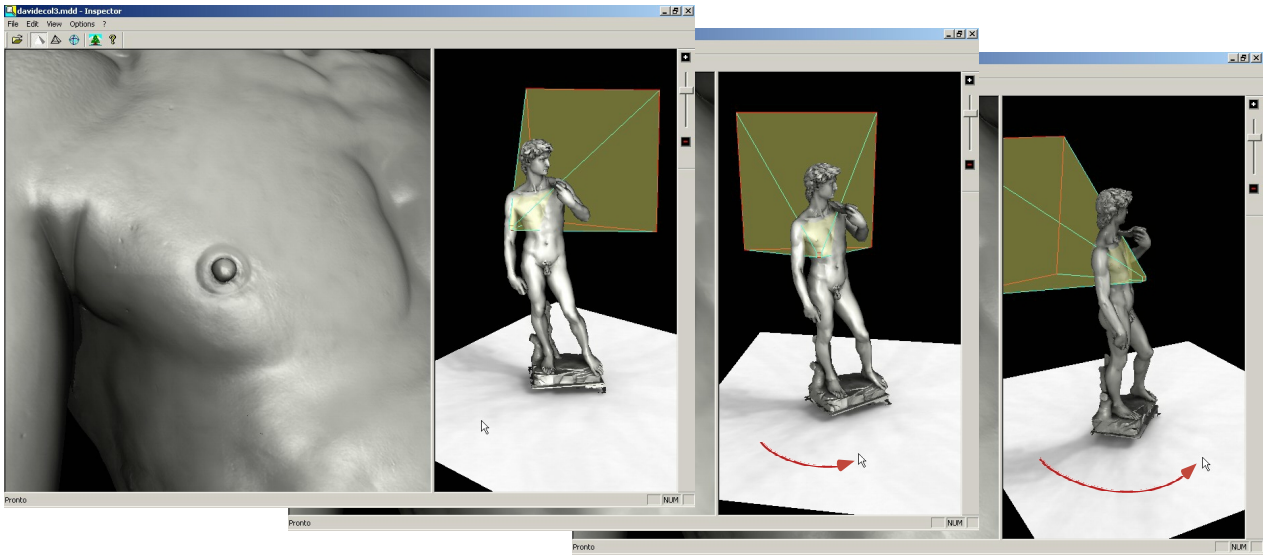

Figure 3: Dragging onto the ground in the rightmost window you can rotate the dummy model around its vertical axis.

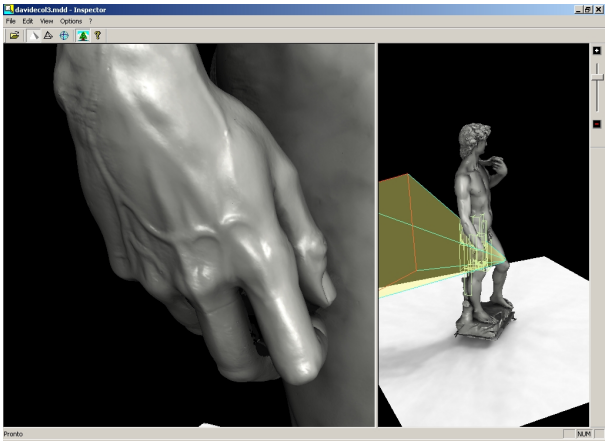

Figure 4: A tighter field of view is chosen, by using the vertical slider on the right.

the current view, rendered as glass-like view frustum. The goal of this visual representation is to prevent the users to be lost in space during inspection of a complex model. Global orientation can be easily lost when we zoom to a small region of the surface.

To resume, the two frames refer to two different coordinate systems: the rightmost frame uses a fixed coordinate system that models a space containing the dummy and the iconic representation of the current view volume; the leftmost frame uses the coordinate sistem induced by the current view specs, selected by the user in the rightmost frame.
Once the user selects a new view, the portion of the triangulated surface contained in the new view volume is extracted on the fly, and it is rendered in the leftmost region (see following section for details).
The user is now free to inspect visually the region chosen and, if needed, to modify in a *constrained* manner the current view, by a direct manipulation of the mesh rendered. A virtual trackball is available also in the leftmost frame, which implements either rotations or pan actions on the selected mesh section.

2.2 LOD Management

It is well known that simplification produce data at different resolutions; these data can be managed following either the *Level of Detail* approach (a few different representations of the geometric object having different levels of accuracy and complexity), or the *Multiresolution* approach (which provides a nearly continuous set of different representations).
The *Inspector* tool is oriented to the visualization of complex, digital representations of single objects. Therefore, complexity resides in the accuracy and resolution of the data, rather than the number of objects represented in the scene. Let us assume that the target object has a limited extent in space, as it is in our case. Therefore, we can also assume that in most rendering tasks all visible geometry is approximately at the same distance from the viewer (this is in contrast with other applications, such as terrain rendering, where view distances can vary substantially in the same frame). Under these assumptions, the use of a variable resolution representation, like the one of [4], is not a mandatory. Moreover, the interaction stream of the *Inspector* browser goes through a number of discrete events (selection of a new view by point-and-click on the object dummy); the so called popping effect is therefore intrinsic in this GUI design, and using a well-chosen discrete number of LOD does not create problems (again, in contrast with the case of applications where the change in

resolution has to be very smooth, such as for terrain rendering, and popping effects have to be avoided). An approach based on discrete LOD is therefore sufficient for the application considered in this project, and the use of static LOD allows: a simpler implementation; a faster retrieval of the requested LOD at dynamic time; and, finally, a better exploitation of graphics hardware by building packed triangle strips that are rendered very efficiently and that are a very compact way of storing a mesh.

We implement multiple levels of details by adopting an Octree hierarchical representation of the object surface. Octrees [7] are spatially sorted, hierarchical data structures capable of storing "true" 3-D objects – the spatial structures and properties of the represented object are represented in a compact form amenable to spatial search. All of this makes very suitable an organization where each LOD is represented by means of an Octree structure, such that all the LODs together construct a sort of Octree forest.
The LOD representation of an input mesh is produced in a pre-processing phase from raw data using an external-memory simplifier based on edge-collapse [2]. We maintain a small number of levels of detail (5-8) and each level is around four times more complex (in no. of geometrical elements) than the previous one.
Each level of detail is maintained by a hierarchical space-subdivision structure (an octree). Non-leaf nodes simply hold pointers to their descendants, while each leaf node stores a pointer to a rendering-oriented representation of the geometric elements contained (they are encoded statically as a set of triangle strips). This rendering-oriented representation consists of three vectors containing, respectively: vertex positions, normals and triangle strip indices. All these vectors are usually larger than current RAM, and therefore are packed together and stored on a file which is memory-mapped. The distribution of the data on the disk is by construction well suited to the on-the-fly access driven by view specifications. During off-line octree construction, recursive subdivision of nodes terminates when the number of geometric elements contained is less than a selected threshold (16K triangles in our current implementation).

For each update of the current view-volume (e.g. according to the interactions either with the dummy or with the rendered region in the left-most frame), the system extracts a representation of the surface section contained in the current view-volume from just one of the octrees. This representation has to satisfy some constraints: the mesh rendered has to be as much detailed as possible, and at the same time it has to fulfill the requested frame rate. The data extraction process firstly accesses the octree used in the previous iteration, selects the set of nodes contained in the current view-volume and measures the complexity of the resulting mesh (each leaf node stores also a counter which measure the complexity of the corresponding mesh portion, therefore evaluating the sub-mesh size is very efficient). If the size is in the optimal range (which depends on the speed of the graphics subsystem available), this mesh is accepted and rendered. Otherwise, the octree immediately preceding in the LOD order [or following, according to the data size required] is selected, and another octree visit is performed and evaluated. Because for each single frame we extract the representation to be rendered form just an octree, all visible geometry is rendered from the same LOD mesh. We do not produce representations where adjacent regions are rendered with data coming from different octrees, to avoid the cracking problem (by construction, there are no cracks between the sub-meshes extracted from different nodes of the same octree).

The selection of the octree leaves contained in the current view volume can be performed in an straightforward manner by using the selection feature of OpenGL (drawing only the bounding boxes of the leaves). The extraction of the current representation is very efficient, and its cost depends on the graphics subsystem performance; it is usually in the order of 0.01-0.05 seconds on an 800MHz PC. Figure 4 shows the octree leaves that are intersected by the current view frustum (the yellow wireframe boxes in the rightmost frame).

The space overhead is very low, because the size of the octree is negligible. Using a threshold of 16K faces for each leaf node, the octree associated to a mesh composed of 15M-30M faces (at the maximal accuracy) is composed of 2.5K-5K nodes. Each node stores: no. of vertices (4 bytes), no. of faces (4 bytes), bounding box (24 bytes) and a pointer to the data (4 bytes), resulting in an overall node size of 36 bytes. Therefore, the space overhead of the octree representing the most detailed LOD of a huge mesh is around 100KB.

The key idea for controlling LODs is what we address as the gaze-driven approach. We assume the object to be inspected is being observed by a viewer, this implicitly defines a viewing angle and as a consequence a viewing volume. The viewing angle defines the gaze point while the viewing volume itself describes a region on the surface of the object. The closer the gaze gets the higher is the level of detail required. Thus the gaze position dictates the LOD required while the system chooses the best-fitting representation level.

The chosen LOD representation is then used during all the interactive phases in order to guarantee a frame rate of approximately 10 frame per second. When the user stops interaction with the model (e.g. he stops to rotate/pan in the left frame) the visualization system starts in the background an interruptible rendering of the highest resolution model under the same view specs. This background process is automatically interrupted if the user starts again to rotate the model. The time required for rendering the highest resolution model depends obviously on its size and on the percentage of the mesh that is in the current view frustum; in the case of the David model, where the high resolution model is composed by 26 million of triangles, the hi-resolution background rendering can take up to a few seconds. Figure 5 shows the difference between the low resolution model used during interactive phases and the high resolution one in the case of a narrow view of the left foot of the Michelangelo's David. We

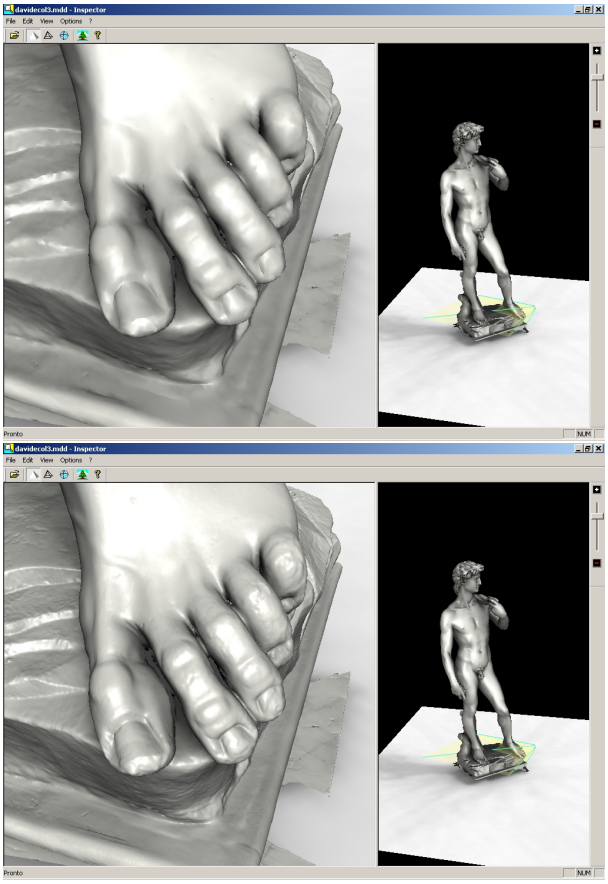

Figure 5: A tight look of left David's foot. In the top image we show the lower resolution model choose for interactive phases while in the bottom image we show the high resolution model rendered in background when the user stop interaction.

have used a very narrow view in order to see some difference between the low and high resolution models. When looking at the whole David the difference between the various LOD's is not really noticeable, because most of the triangles in the most refined LOD map to an area smaller than a single pixel.

2.3 High Quality and Efficient Rendering

A common problem in computer graphics is the evaluation of the trade-off between quality and efficiency in the rendering process. In the context of the visualization of models of real statues we have found that the standard lighting model used in interactive graphics is not satisfactory. Standard OpenGL lighting [8] does not take into account the effects of cast shadows and these effects are very important in our context. Infact the artist usually take into account these lighting effects during the creation of a statue. The most common example is the shape of the pupil of the David's eye. To create the effect of something similar to a shiny reflection Michelangelo sculpted the pupil as an hole with a small triangular inset that remains lighted against the dark background of the hole. It is evident the change of the expression of David's face when considering cast shadows or not (see Figure 6). Note that the only difference between the left and right rendering is in the lighting: on the left a standard OpenGL rendering with Phong lighting, on the right an OpenGL rendering with a more correct lighting.

Unfortunately, current graphics hardware is not able to correct render self-cast shadow in a easy and efficient way. Most recent graphics hardware can do it but with a sensible performance penalty: to render the same mesh you need a time that is approximately two or three times larger. Moreover even using these techniques it is difficult to simulate soft lighting environments where lights are not point-size and shadows are not very sharp.

For the sake of efficiency we have chosen to partially fix the lighting environment, precompute the shadow and store it as a per-vertex color onto the mesh itself and then add a standard headlight. The fine tessellation of the mesh guarantees a good sampling of the shadow map onto the surface. The approach of using only precomputed shadows results in a lighting environment that is too *static*, the common metaphor of the headlight (move always the light together with the observer around the object) provides a better feedback that a static fixed light. For this reason we have chosen a rather hybrid approach: we use a headlight, but we substitute the ambient lighting term with the exact lighting that each face of the mesh would receive from an anisotropic diffuse lighting environment. In other words in place of the ambient lighting term we simulate the effect of an environment where the light comes from any directions but with higher intensity from above (something like an object lighted under a cloudy day). Correct diffuse lighting can be evaluated by computing the solid angle of the *sky* that can be seen from each face. Larger the solid angle more luminous will be the face. We show this concept in Figure 7: face f_1 is darker than face f_2 because it sees a smaller portion of the sky. This lighting process is therefore done statically in a pre-processing phase; more details on how this lighting can efficiently be computed using graphics hardware are described in [1].

Since we use the diffuse lighting in place of the ambient term we still use a head-light directed along the viewer line of sight; in this way we still have a view-dependent lighting that makes all the shiny reflections of the surface correctly move together with the viewer (as you can see onto the hand in Figure 4), but we still have a correct darkening in the less accessible areas (like among the fingers in Figure 4 or the forehead below the hair curls in Figure 6).

3 Results and Conclusions

Some images presented in the paper have been obtained by visualizing the Michelangelo's David mesh, one of the results of the Digital Michelangelo Project [6], which is probably the most impressive mesh ever produced (thanks to the

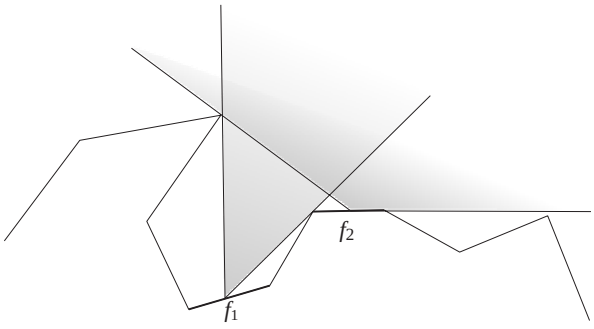

Figure 7: Correct diffuse lighting can be evaluated by computing the solid angle of the *sky* that can be seen from each face. Larger the solid angle more luminous will be the face.

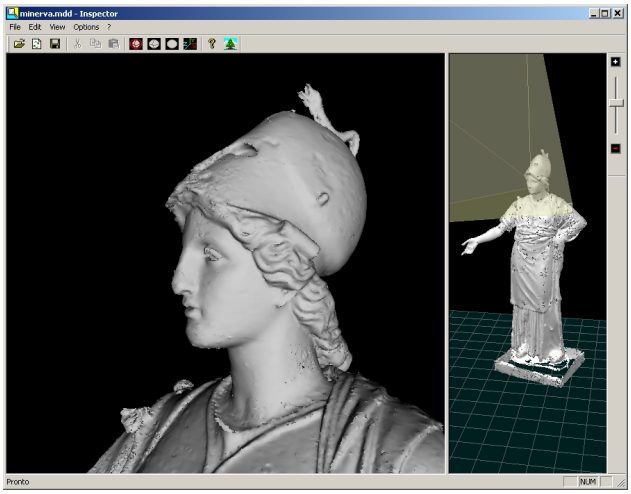

Figure 8: A snapshot of the Minerva model is shown.

skill of both the sculptor and the scanning team). As a matter of fact, the *Inspector* tool was developed in the framework of a cooperation with the Centro di Restauro (Restoration Laboratory) of the Soprintendenza Archeologica Toscana in Florence. The Restoration Laboratory started in year 2000 the restoration of the Minerva, a statue of the Archeological Museum at Florence, and this was immediately perceived as the ideal testbed for the development and the experimentation of restoration-oriented 3D digital technology. The Minerva is a bronze statue, 1.70 cm tall, discovered in Arezzo in 1541 during excavations of a dwell in the S. Lorenzo church and whose origin is still unclear (i.e. whether it is an original Greek statue or a later Roman copy). The bronze layer presents extensive corrosion. Moreover, it was extensively restored in the past: previous restorations dates back to the sixteenth and the eighteenth centuries. All these actions modified in a significant manner the aspect and integrity of the art work (e.g. reconstructed sections in plaster, use of greenish paint to get a uniform appearance between the original and the reconstructed parts).

The statue has been scanned in October-December 2000 using a prototypal 3D scanner based on structured light [10]. A raw model, composed of 26 million triangles, has been obtained (see Figure 8). The planned use of the 3D model has different objectives: to monitor the initial and the post-restoration status of the artifact (a major modification of shape and appearance is forecasted); to be used to map and correlate the results of different survey (photo under visible and UV light, X-ray imaging, results of chemical analysis, etc.); to be part of a multi-media document which should document the complete restoration; to be used for visual presentation (museum installations or multimedia CD). One of the tool designed was therefore the *Inspector* browser, which has to support easy inspection of the 3D model to naive users.

The first experimentation of the *Inspector* tool seems very encouraging. A beta-release of the system was evaluated by the restorers and was considered sufficiently handy. Obviously visualization of the digital mesh was not considered "the goal" by the restorers. We are now working on an extended version of the tool, to support: the mapping and visualization of different sources of data (such as 2D images, or hot spot and hyperlinks) on top of the digital statue surface; and interactive measuring (for computing lengths, surface area, sections, diameters on the digital model).

Acknowledgements We would like to thank the Stanford Computer Graphics Group for giving us access to the results of the Digital Michelangelo Project.

References

[1] M. Callieri, P. Cignoni, and R. Scopigno. Efficient diffuse lighting precomputation for rendering. Technical Report TR-15, I.E.I. – C.N.R., Pisa, Italy, Nov. 2001.

[2] P. Cignoni, C. Montani, C. Rocchini, and R. Scopigno. External memory management and simplification of huge meshes. Technical Report TR-01, I.E.I. – C.N.R., Pisa, Italy, Jan. 2001.

[3] P. Cignoni, C. Montani, C. Rocchini, R. Scopigno, and M. Tarini. Preserving attribute values on simplified meshes by re-sampling detail textures. *The Visual Computer*, 15(10):519–539, 1999. (preliminary results appeared in IEEE Visualization '98 Proceedings).

[4] J. El-Sana and Y.-J. Chiang. External memory view-dependent simplification. *Computer Graphics Forum*, 19(3):139–150, August 2000.

[5] M. Garland. Multiresolution modeling: Survey & future opportunities. In *EUROGRAPHICS'99, State of the Art Report (STAR)*. Eurographics Association, Aire-la-Ville (CH), 1999.

[6] M. Levoy, K. Pulli, B. Curless, S. Rusinkiewicz, D. Koller, L. Pereira, M. Ginzton, S. Anderson, J. Davis, J. Ginsberg, J. Shade, and D. Fulk. The Digital Michelangelo Project: 3D scanning of large statues. In *SIGGRAPH 2000, Computer Graphics Proceedings*, Annual Conference Series, pages 131–144. Addison Wesley, July 24-28 2000.

[7] D. Meagher. Geometric modeling using octree encoding. *Computer Graphics and Image Processing*, 19(2):129–147, 1982.

[8] J. Neider, T. Davis, and M. Woo. *OpenGL Programming Guide*. Addison Wesley, 1993.

[9] H. Pfister, M. Zwicker, J. van Baar, and M. Gross. Surfels: Surface elements as rendering primitives. In Kurt Akeley, editor, *SIGGRAPH 2000, Computer Graphics Proceedings*, Annual Conference Series, pages 335–342. ACM Press - Addison Wesley Longman, 2000.

[10] C. Rocchini, P. Cignoni, C. Montani, P. Pingi, and R. Scopigno. A low cost 3D scanner based on structured light. *Computer Graphics Forum (Eurographics 2001 Conf. Issue)*, 20(3):299–308, 2001.

[11] C. Rocchini, P. Cignoni, C. Montani, P. Pingi, R. Scopigno, R. Fontana, L. Pezzati, M. Cygielman, R. Giachetti, and G. Gori. 3D scanning the Minerva of Arezzo. In *ICHIM'2001 Conf. Proc., Vol.2*, pages 265–272. Politecnico di Milano, 2001.

[12] H. Rushmeier, F. Bernardini, J. Mittleman, and G. Taubin. Acquiring input for rendering at appropriate levels of detail: digitizing a pietá. In G. Drettakis and N. Max, editors, *Rendering Techniques '98*, pages 81–92. Springer Wien, 1998.

[13] S. Rusinkiewicz and M. Levoy. QSplat: A multiresolution point rendering system for large meshes. In *Comp. Graph. Proc., Annual Conf. Series (SIGGRAPH 00)*, pages 343–352. ACM Press, July 24-28 2000.

[14] Pedro V. Sander, Xianfeng Gu, Steven J. Gortler, Hugues Hoppe, and John Snyder. Silhouette clipping. In *SIGGRAPH 2000, Computer Graphics Proceedings*, Annual Conference Series, pages 327–334. Addison Wesley, 2000.

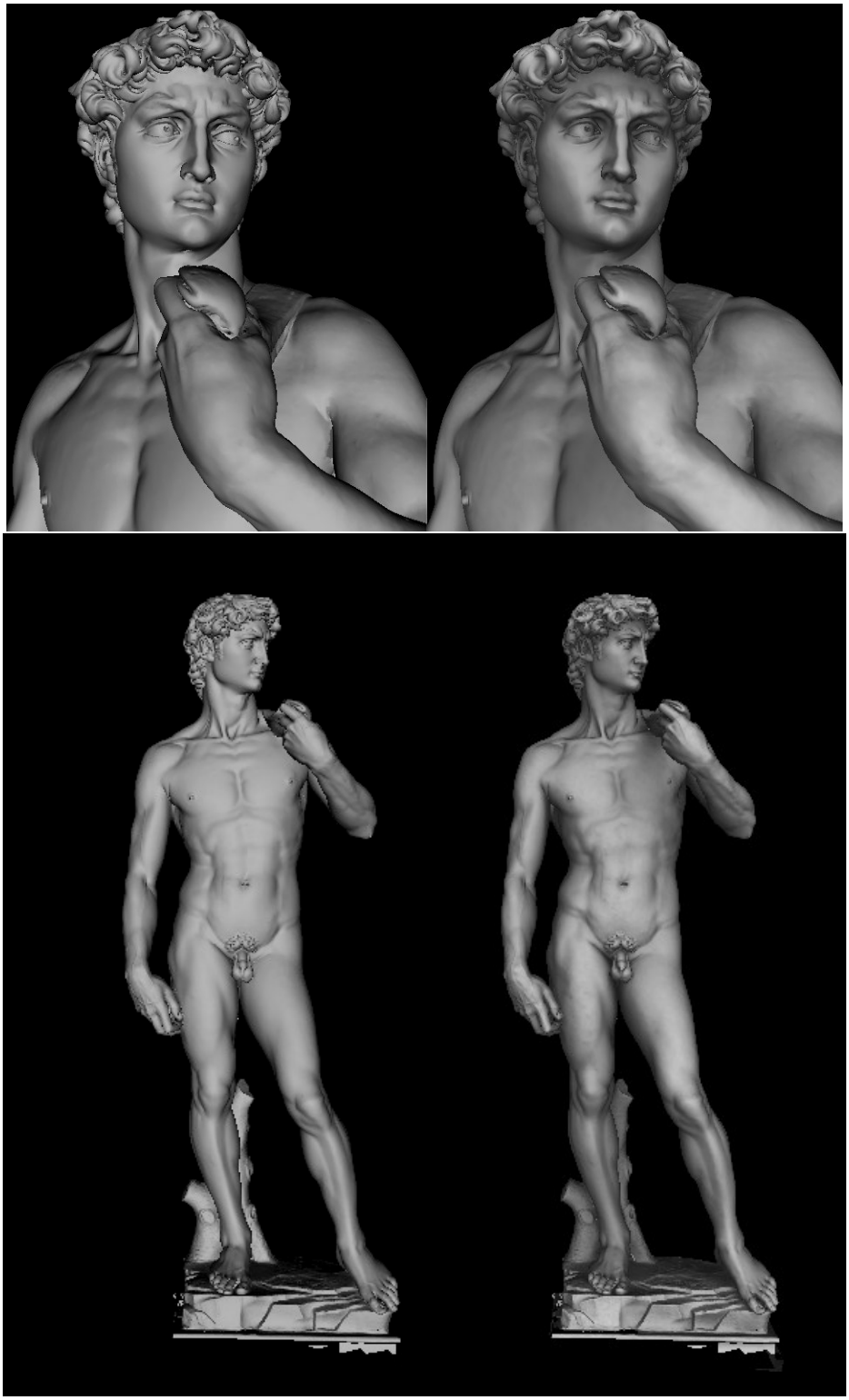

Figure 6: Rendering cast shadows greatly affects the resulting expression of the David's face. On the left a standard OpenGL rendering with Phong lighting, on the right a OpenGL rendering with a more correct precomputed lighting.