Università di Pisa
Dipartimento di Informatica
Dottorato di Ricerca in Informatica

Ph.D. Thesis: 7/06

Beyond high-resolution geometry in 3D Cultural Heritage: enhancing visualization realism in interactive contexts

Marco Callieri

Supervisor

Dott. Paolo Cignoni

Dott. Roberto Scopigno

Referee

Prof. Pere Brunet

Prof. Luc Van Gool

October 16, 2006

Acknowledgments

I wish to thank my family, for supporting me during this period and enduring my temper when I was under pressure.

Thanks goes to my Ph.D. supervisors Dott. Paolo Cignoni and Dott. Roberto Scopigno and to the head of the Visual Computing Lab Dott. Claudio Montani, who helped me in my work, in my research and in my personal growth.

This work could not have been possible without the contribution of all people in the Visual Computing Lab, who worked with me in the last few years and shared with me the hard work during on-the-field campaign.

A special thank goes to Prof. Paul Debevec, who let me work on an incredibly interesting project, and to all the people at the ICT Graphics Lab and FlatWorld.

Thanks goes to the reviewers of this thesis and to all people who worked for the Ph.D. program of the Department of Computer Science in Pisa.

Finally, I wish to thank all my friends for being there when I needed them most.

Contents

Chapter 1

Introduction

During the last decades computer graphics has become a core technique within computer science and information technology. Computer systems are more and more used to represent and simulate real or imaginary worlds. Such a task requires to model or, better, to acquire from reality, to process and to render complex objects and scenes.

The goal of computer graphics is to turn abstract information into visual images and to allow the user to interact with complex objects and data in a natural and intuitive way.

1.1 Framework

The base of the 3D computer graphics is the use of digital 3D models. The models are used for calculations on the geometry and to produce renderings. A possible way of generating the 3D models is to manually (or semi-manually) create them using software tools (CAD, 3D modelers). Another way is to obtain a digital representation of real objects trough the use of sensors. This kind of technology goes normally under the name of *3D scanning*.

3D scanning is an emergent technology that has various uses in industrial (reverse engineering, quality control), medical (crime forensic analysis, prosthetics, anatomical studies) and cultural heritage fields (documentation, restoration).

Graphics research community gave to 3D scanning lot of attentions in the last few years; in first instance, it was necessary to provide algorithms and sensors for faster and more precise geometry acquisition. Then, since the scanning hardware does not produce a ready-to-use 3D model, new algorithms were necessary to define reliable and efficient methods for the generation of complete and not redundant digital models. Moreover, the large amount of data produced posed serious memory management issues. Finding a way to efficiently manage all that data was another challenge.

Nowadays, the commercial scanning hardware has reached a very high level of accuracy and stability. The range of objects that can be acquired has significantly increased, both in terms of dimension and of materials variety. New, cheaper and more powerful 3D scanners hit the market every few months. However, the weak part of the acquisition process still resides in the software that has to be used to generate the 3D model.

At this point, the research focus has moved also in the direction of the 3D model usage. Given the presence of a highly detailed 3D model, it is possible to define new interaction paradigms which can be useful to solve problems (or give new instruments) for certain works. This kind of research is carried out normally in conjunction with people that, while not computer science and graphics experts, work in that field and can help in defining goals and establish useful methodologies.

In our case, we worked to apply the 3D scanning technology to the Cultural Heritage field.

1.2 Applications of 3D models in Cultural Heritage

Developing tools for acquisition and manipulation of 3D scanned data, it was mandatory to find specific application fields where to test such algorithms. Given the extraordinary availability of cultural heritage in our country, it seemed a natural choice to apply the potentiality of 3D scanning to the cultural heritage field.

A first reason for this choice is the possibility to obtain 3D models that are beautiful and that will be more understandable for people. Users can evaluate the quality of a 3D scan of a human statue better than a scan of some mysterious mechanical parts.

Moreover, when compared (for example) to industrial objects, cultural heritage objects often present additional challenges. Great variety of materials and surface characterizations, extremely intricate surface geometry and difficult object accessibility are some of the problems that have to be faced. These problems can be useful to demonstrate the effectiveness of a given approach or to stimulate further researches.

Having a 3D model of a cultural heritage object, its most straightforward application is to produce still images or interactive animations for documentation, didactic applications or multimedia presentation. For this particular use, the visualization realism is an essential requisite. However, the pure rendering is not the only possible outcome of this kind of data.

The availability of an accurate digital representation opens several possibilities of utilization to help cultural heritage experts (restorers, archivists, museum curators) in their work. The availability of a complete and accurate knowledge of the shape's

characteristics of Cultural Heritage artworks is the ideal base for cataloguing and technical documentation; the 3D representation grants much more information than a conventional photo (that, as today, still remains the standard representation media in catalogues), since it contains a complete representation of the object shape.

An exciting opportunity is to make an intense use of digital 3D models in artworks restoration. The integration between 3D graphic and restoration represents an open research field, aiming at the development of new techniques and new support methodologies for the work of restorers and scientists. In this context the David restoration project represents a milestone and it it has given several starting points and guidelines to the definition and development of innovative solutions.

As proposed in the David's restoration [18, 19], a 3D digital model can be used to support restoration in two different ways: as a instrument or tool for the execution of specific investigations, or as a supporting media for the archival and integration of the multimedia data produced by the different scientific studies.

We can already provide a faithful representation of the object geometry; this is essential for some of the more technical applications, but it is not enough to obtain a realistic representation of the object. However, when the user has to interact with the 3D model, a more realistic visualization can help the object perception, increasing its effectiveness.

1.3 Objectives

Many efforts have been spent on improving rendering speed for very large 3D models, introducing complex data structures for efficient data storage and retrieval or employing faster drawing algorithms. However, very few researchers focused on new solutions to increase visualization *realism* when dealing with such large datasets. Currently, we are able to render in realtime the geometry of an entire city (or even of a planet), but with a very unrealistic result.

Realism has been an objective for computer graphics researchers, but most of the available techniques for surface representation, shadows and advanced lighting have been developed in such way that they work almost only with small to medium size 3D models.

Many of these techniques rely on having a quite coarse geometry, overcoming this lack of data by introducing lot of texture-mapped data, used at runtime to recreate complex material and lighting behaviors. These approaches, while effective on such kind of models, are impossible to be used in conjunction with very large models. This because some of the requirements for these algorithms cannot be met: for very large models, the geometry cannot be stored completely in memory and it is quite uncommon to be able to build a parametrization for texture mapping.

The main objective of this thesis is to define methodologies to obtain a more realistic visualization when rendering very complex 3D models. We want to be

able to show surface characterization and to provide a realistic illumination in the visualization of very large 3D models.

This task will be accomplished by finding ways to apply existing techniques to very large 3D models, normally considered too big for such kind of processing. Moreover, we will present some additional methodologies to increase visualization realism for this kind of models.

It is important to say that this quest for realism is not just a pure aesthetic need. Human perception has been tuned to understand the environment trough senses, through an evolution of millions of years. To convey the maximum information through an image generated from digital models, that image should be able to provide a stimulus similar to the one perceivable from the real object.

Obviously, the cultural heritage field is the one that benefits the most from such realism. A work of art is, in the majority of cases, not only defined by its geometry, but also by its color, material and (even more important) by its interaction with light. When presenting such artifacts, it is mandatory to consider these effects if we want to provide a faithful representation of the object.

With the pure geometry of an artistic artifact it is possible to perform a vast range of measurements and analysis. This level of presentation and analysis can be enough from a more technical point of view: it is possible to make consideration about constructions techniques, artifact history and plan restoration actions. However, for those tasks that are centered on human perception, like documentation, dissemination and didactic purpose, a more realistic visualization of 3D models is almost essential.

1.4 Methodologies

When working with very large models, there are additional considerations that have to be taken in account when designing algorithms and techniques for data manipulation and rendering.

Data storage Having a parametrization for very large models can be impossible; there are problems related to the presence of topological irregularities, to the irregular sample of the surface and also to the sheer size of the model. An alternative storage paradigm would help in ensuring the applicability of shading algorithms even for model with such problems.

Precalculation Many algorithms for material representation and lighting have a quite long per-element execution; since we will work with such large models, the time will grow exponentially. However, a large part of the lighting calculation is invariant and depends only on the geometry. Precalculation of lighting invariant is an essential instrument to speed up realtime applications.

Locality Given the raw occupance of the 3D models, it is impossible to keep all the geometry in main memory: it will be mandatory to use some kind of secondary memory support. This will imply that just a small part of the object will be available in memory. Any algorithm we will use should be local; all the needed information should be stored within the loaded block, without the need, at rendering time, to access data from neighbors.

We believe that in most cases, given the complexity of the 3D models, working on a per-vertex basis can be enough to obtain good visual results.

On triangulated 3D models, the only *real* geometric information is contained into vertices, while the triangles are elements used to fill the unknown with interpolation. Many techniques, like texture mapping and bump mapping, are used to better describe the surface represented into the triangles, thus overcoming their lack of information. The necessity of this process increases with the size of the triangles.

When the complexity of the 3D model is very high, each triangle will be quite small and, more important, when projected on the screen, it will map on few pixels. At the same time, if the surface of the object is represented with such precision, it is quite uncommon to be so close to the surface to be able to clearly see the triangles. So, in most cases, a per-vertex processing can guarantee good results at a much lower computational cost. In the next chapters we will show different examples of this approach.

As stated before, given an illumination algorithm, it is possible to determine some parameters that depend only on geometry and can be precalculated to speed up the computation. Precalculation does not affect rendering quality; it is just basically *moving* some computation outside the rendering itself, making it faster (more or less like an algebraic simplification reduces the number of operations in an equation, without changing the result).

We will follow this approach and present methods to precalculate various lighting parameters, always considering the limitations posed from the huge size of the 3D models we will work on. Using the per-vertex encoding, we will be able to store all the precalculated data necessary to compute the shading in an efficient and simple way.

It is to be noted that encoding shading parameters into geometry is also supported by other considerations. If we are already dealing with a large geometry using a secondary memory management, even the addition of data to each geometric element does not influence the system too much, since this data can be managed using the same out-of-core approach. Moreover, this kind of storage will ensure the *locality* property we were searching for.

Having defined *local* algorithms for shading and having stored all the necessary data *inside* geometric elements, it will be easy to implement all the computation on accelerated 3D video cards. The new programmable graphics hardware offers

an incredible computational power, very complex lighting algorithms can be implemented without sacrificing performances. This is especially useful when working with Out-Of-Core technologies; in this situation the CPU is busy managing the secondary memory storage, it is then advisable to move to GPU as much computation as it is possible.

1.5 Outline

In Chapter 2 we will present some of the previous works available in literature; this technical review covers three topics that we will be the starting point of this thesis:

- Geometry acquisition: the 3D models and environments we will use for our work will come mainly from 3D scanning and other techniques for automatic digitalization of real objects. It is then necessary a (brief) explanation of how those techniques work and which is the kind of data will be the input of our work.
- Acquisition and mapping of surface attributes: many different approaches has been used to enrich the pure geometric information with color. These techniques range from the simple texturing to more complex models to describe surface properties.
- Efficient rendering of large datasets: our main ambition is to enhance rendering realism of complex 3D environments. In this part we will cover the standard approaches that are used to manage large quantities of data. We will use those techniques as a base for our work. Various algorithms able to efficiently manage huge datasets are already available in literature; it is then advisable to work to *extend* existing techniques. Developing algorithms able to work in conjunction of those techniques will yield the maximum benefit, since there will be a realism enhancement in some already efficient applications.

In Chapter 3, we will discuss the problem of how to map and integrate on a 3D model the color information coming from various photos. Images taken trough photography represent the easiest and most affordable method to gather information regarding surface characteristics. It is essential to have a solid method to map this kind of information. In general, the information provided through photos has the advantage to be redundant and at a very high resolution but, at the same time, it presents lot of photographic artefact. In order to maximize the information usage without running into quality loss (due to smoothing) we propose a weighted blending function, able to preserve sharp features while eliminating most of the artifacts.

Having a good surface characterization can improve rendering realism; however, most of the information we can perceive about an object comes from shading. In Chapter 4 we will introduce some methods to calculate and display the ambient occlusion shading. We will show that its computational overhead is quite low, while this kind of shading algorithm greatly enhances the appearance of the objects, showing a more correct interaction between light and object surface.

In Chapter 5, we will show how it is possible to build a realtime application that combines a very complex 3D scene and a realistic lighting. We will describe the design and implementation of an interactive system which reproduce in realtime the **time lapse** sequence from the short movie The Parthenon presented at Siggraph 2004 [137]. This is a good example of how unique a project can be; even if many common points remains between different projects, it is not possible to define an all-purpose methodology. The constrains of this particular project forced some interesting choices, we believe this presentation can give a good overview of how such decisions were taken and why.

In Chapter 6, we will present two systems developed in conjunction with Cultural Heritage partners. The tools, while essentially 3D graphics applications, have been developed to cope with specific needs of people working in the CH field. We will describe the results of the use of those tools in the framework of specific Cultural Heritage projects.

Finally, in Chapter 7 we will discuss effectiveness of the work described in this thesis. Using the examples provided in the previous chapters, we will analyze the main contributions and we will discuss remaining open problems, outlining possible developments and extension of the proposed techniques.

Chapter 2

State of the Art

In this chapter we will discuss some of the background work that is the base of this thesis. This review will describe what is the creation process of a 3D models and which are the problems associated with their generation and manipulation. Then we will describe the techniques to acquire and represent surface characteristics. Finally, we will describe some methods for efficient rendering of large datasets.

2.1 Geometry acquisition

3D scanning technology evolved considerably in the last few years, both in terms of hardware devices and of algorithms for processing the raw data produced by scanning devices [5].

The process of acquisition of a 3D model is not straightforward. Generally, it is not possible to obtain a geometric model in a single operation, like it is done with 2D scanners. Most of scanning systems only generates a set of 3D points named *range map* that represents the geometry of the visible part of the object. Those points have to be necessarily converted into a connected representation, usually a triangular mesh, before using them. Moreover, the construction of a complex model requires the integration of multiples scans in order to obtain the entire surface of the object. So, the input data are submitted to some phases of processing. In this section we briefly summarize the state of the art of the 3D scanning pipeline. The process of building and using 3D models from raw data acquired by 3D scanners is normally called *acquisition pipeline* (see Figure 2.1) and it is composed by various steps:

- Acquisition planning
- Range maps acquisition
- Range maps filtering
- Registration

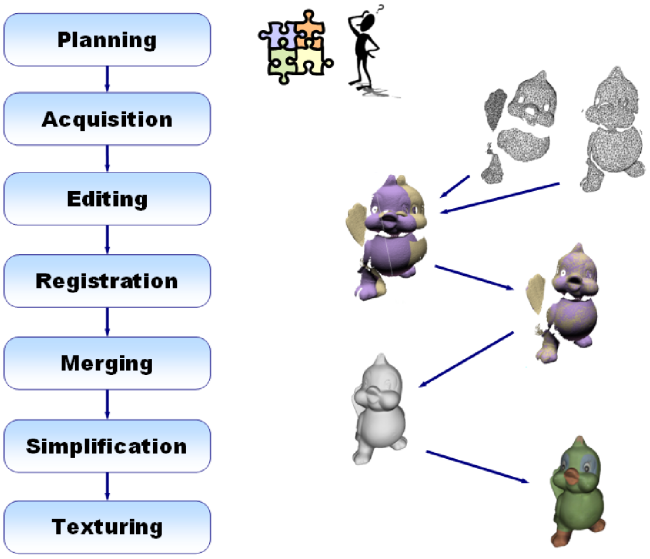

Figure 2.1: The various step necessary for the 3D digitalizion of real object; the process is often addressed as *scanning pipeline*.

- Merging
- Geometry simplification

The first step is the acquisition planning; in this phase are selected the positions of the scanner from which acquire the surface portion of the object. Raw data are acquired by a 3D scanner according to the choices of the previous phase. The set of range maps are then registered in a unique coordinate system and merged together in a single coherent surface. The obtained model must be simplified, the redundant geometry information is eliminated in order to reduce the memory occupancy of the model.

2.1.1 Sensors

A review of the available techniques for geometric acquisition is far beyond the scope of this work, a more complete overview of this technology can be found in [9]. Many different scanning devices exist, including both academic prototypes and commercial systems. Optical approaches are the most used, since they do not need to touch the objects nor to employ potentially dangerous radiations (like TAC or PET scanning). A common classification divides the available optical techniques in: passive systems (e.g. the ones based on the reconstruction from silhouette approach, which returns a nearly complete model of the object but suffers of scarce accuracy); and active systems, which sample the surface by actively projecting a

laser or structured light pattern on the object, and measure the geometry of the hit points either by triangulation or time of flight. Most of these active systems produce in output a range map, i.e. a 2D grid of points sampled on the visible surface of the object. A subset of the latter produce geometry + color range maps (i.e. range maps with sampled position and RGB values for each point).

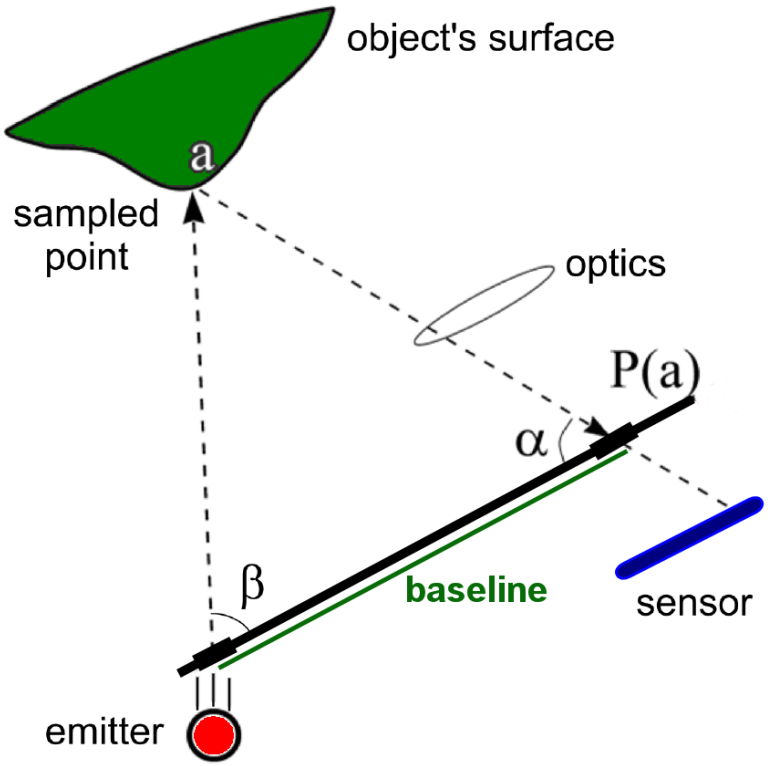

Figure 2.2: General Scheme for an active optical scanner based on triangulation. Emitter sends light on the surface, reflected light is read back by the sensor. Emitting angle β and view angle α are measured, the baseline lenghts is known; using this data it is possible to calculate the position of the probed point with respect to sensor

The most common range scanners are **triangulation systems**, shown generically in Figure 2.2. A lighting system projects a pattern of light onto the object that has to be scanned: possibly a spot or line produced by a laser, or a multi-stripe pattern produced by a projector. A sensor, frequently a CCD camera, senses the reflected light from the object. Software provided with the scanner computes an array of depth values, which can be converted to 3D point positions in the scanner coordinate systems, using the calibrated position and orientation of the light source and sensor.

A fundamental limitation of what can be scanned with a triangulation system is having an adequate clear view for both the source and sensor to see the surface point currently being scanned. In these cases it is necessary to integrate multiple

scans in order to avoid holes in the reconstructed model. Moreover, if the object to be scanned is too far from the scanner, the problem of finding the depth of the surface point is ill-conditioned. This happens because the angle between emitted and reflected light is too small, generating a skewed triangulation. Surface reflectance properties affect the quality of data that can be obtained. Triangulation scanners may perform poorly on materials that are shiny or that have low surface albedo or significant subsurface scattering.

An alternative class of range scanners are **time-of-flight systems**. These systems send out a short pulse of light and estimate distance by the time it takes the reflected light to return. An advantage of this technique is that the emitter and the receiver can lie on the same axis, consequently, these systems not suffer from shadowing effects as triangulation methods do. Time-of-flight systems have been developed with near real time rates, and can be used over large (e.g. 100 m) distances. They require high precision in time measurements, and so errors in time measurement fundamentally limit how accurately depths are measured, so they are used mainly for the acquisition of large structures. Triangulation systems, instead, are preferred in applications where accuracy is needed, such as cultural heritage.

Basic characteristics to know about a range scanner are its scanning resolution and its accuracy. **Accuracy** is a statement of how close the measured value is to the true value. The absolute accuracy of any given measurement is unknown, but a precision that is a value for the standard deviation that typifies the distribution of distances of the measured point to true point can be provided by the manufacturer. The methods used by manufacturers to determine the precision are based on standard tests for length measurement developed for coordinate measurement machines or surveying applications, depending on the scale of the application. The absolute value of error increases with distance between the scanner and object. The deviation of measurements is a thin ellipsoid rather than a sphere, the error is greatest along the line-of-sight of the sensor. The precision of the measurements may vary across a range image. There are some effects that produce random errors of comparable magnitude at each point. Other effects may be systematic, increasing the error towards the edges of the scan. Because models are built from points acquired from many different range images, it is important to understand the relative reliability of each point to correctly combine them.

Resolution is the smallest distance between two points that the instrument measures. The accuracy of measured 3D points may be different than the resolution. For example, a system that projects stripes on an object may be able to find the depth at a particular point with sub millimetrical accuracy. However, because the stripes have some width, the device may only be able to acquire data for points spaced millimeters apart on the surface. Resolution provides a fundamental bound on the dimensions of the reconstructed surface elements, and dictates the construction of intermediate data structures used in forming the integrated representation.

Range scanners do not simply provide clouds of 3D points, but implicitly provide additional information. Simply knowing a ray from each 3D point to the scanning

sensor indicates that there are no occluding surfaces along that ray, and provides an indicator of which side of the point is outside the object. Since range images are organized as two-dimensional (2D) arrays, an estimate of the surface normal at each point can be obtained by computing vector cross products for vectors from each point to its immediate neighbors. These indicators of orientation can be used to more efficiently reconstruct a full surface from multiple range images.

2.1.2 The Scanning Pipeline

Before actually starting the acquisition phase, it is necessary to plan the session, i.e. the number and the viewpoints of scans have to be chosen in order to cover the whole object's surface and guarantee a reasonable overlapping between neighboring range maps. Sensor distance, that affects resolution, is an important choice because it determines the accuracy of the final model. Overlapping of range maps is necessary for the alignment step, but a large overlap would result in an excessive number of scans, increasing the time of the acquisition and the amount of data. Find a good scanning set is a difficult task and it is a *NP-hard* problem, the solution is to find good heuristics and approximate
results.

Alignment

After the acquisition phase, multiple scans must be aligned (*registered*) into a common coordinate system so they can be integrated (*merged*) into a single 3D model. A common approach to registration consists of an initial rough alignment followed by a refining technique in a successive step. The initial placement is mainly done "by hand".

In the last years many efforts have been made in order to avoid the manual placement. Some techniques try to estimate the scanner pose by acoustic, magnetic, or optical tracking. A similar approach employs a calibrated turntable but it imposes some restrictions on the dimensions of the objects to be acquired. Good results can be obtained with some algorithms which performs the registration in automatic manner using matching techniques to find sets of corresponding points between the views without user interaction.

Since none of these techniques can give accurate alignments, a refining step must be performed on the initial rough placement. The most successful approach to solve this problem is the Iterative Closest Point (ICP) algorithm proposed by Besl and McKay [8], Chen and Medioni [29], and Zhang [150].

The ICP algorithm consists of two steps. In the first step, pairs of candidate point matches are identified in the overlapping area of two neighbouring range scans. Then, an optimization procedure computes a rigid transformation which reduces the distance between the two sets of points. The process is iterated until some conver-

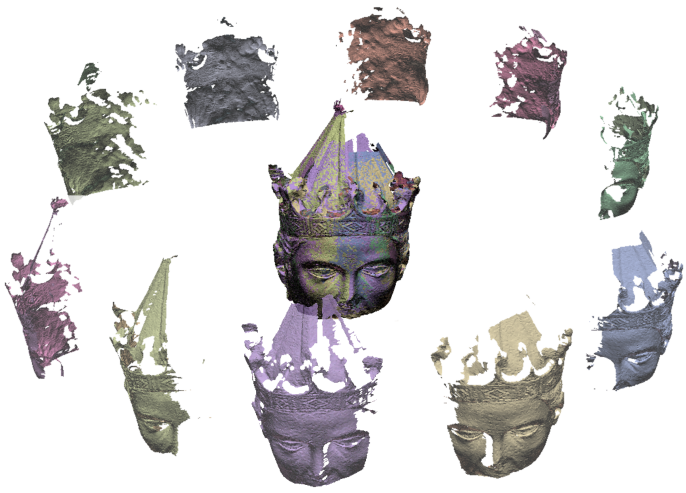

Figure 2.3: An example of range maps alignment. Toward the border, the single range maps. In the center, the range maps have been correctly aligned in a common reference system.

gence criterion is satisfied. The method is based on the assumption that at each iteration the distance between the two scans is reduced, allowing for a better identification of matching point pairs in subsequent iterations. It has been proved [8] that the process converges to a local minimum but not necessarily to a global one. Variations of the basic algorithm differ in how the candidate matching pairs are identified, which pairs are used to compute the rigid transformation, and in the optimization procedure used. Besl and McKay [8] choose the matching candidate to a point using Euclidean distance. Instead, Chen and Medioni, for each point p on scan P, find the sample point q' on Q closest to the normal ℓ to P in p. The candidate matching point is the point q on the tangent plane to Q in q', closest to p. The main advantage of this technique is that it is not much sensitive to nonuniform sampling.

Registration of multiple range maps

So far, we illustrated the ICP algorithm, the most successful solution to the problem of pair-wise range scans alignment. A set of range map can be registered pairwise, but pairwise alignment leads to accumulation of errors when walking across the surface of an object, the optimal solution should minimize distances between all range scans simultaneously: the global registration solves this problem.

The issue of how to minimize the global error is not treated in the same time of the ICP problem. There are many different global registration approaches, but it is possible to classify them in two classes: *incremental* and *global optimization*.

One of the first solution to the global registration was proposed by Turk and Levoy in [143]. They use an incremental method consisting in a first step in which a cylindrical scan that covers most of the object's surface is acquired, and then in a second step other scans are registered to it. Other approaches [10, 102] (coming from global optimization field) search for a simultaneous solution of all the rigid motions using some optimization techniques such as simulated annealing or least-squares. Execution times even for just a few views are fairly long. Another global optimization approach [135, 49] models the problem by supposing a set of springs attached to point pairs and simulating the relaxation of the dynamic system.

An important (incremental) solution of the global registration problem that is particularly right for large dataset was proposed by Pulli in [114]. Pulli's method consists of two steps: in a first step all scans are registered pair-wise, and the initial registration is used as a constraint in a second pass which evenly distributes the pair-wise registration error.

Merging

Once the range maps have been acquired and registered they are integrated in a single digital model. This process however is not straightforward and can present some difficulties. The input data are often noisy and may contain outliers, some areas may have not been reached by the scanner, or the sampling density may be not sufficient for a correct reconstruction of the object's surface. Some of these difficulties are attenuated if additional information other than the simple points positions, such as estimated surface normals, adjacency in the range maps, are available.

A **first classification** of methods can be made on whether the input data is assumed to be unorganized points (point cloud) or a set of range scans. Techniques that deal with the first kind of input are more general, but also usually less robust in the presence of noise and outliers. The second category uses information in addition to simple point position, such as estimated surface normal, partial connectivity embedded in the range scan, sensor position, to better estimate the actual surface. A **second classification** is based on the approach taken to reconstruct surface connectivity:

- A first class of techniques reconstructs the surface connectivity by triangulating the input points. Given a set S of points in the space R^3, the associated Delaunay complex $D(S)$ imposes a connectivity structure between points in S. Taking advantage from this property, Delaunay-based methods reconstruct a surface by extracting a subcomplex from $D(S)$. Notice that these methods usually take as input a set of *point cloud*.

- Another group of algorithms constructs the connectivity information by locally parameterizing the surface and connecting each point to its neighbours by

local operations. Triangulated local patches are then connected to build the global triangulation. Some of these methods take advantage of the implicit connectivity of range maps.

- A third class of algorithms is based on the computation of a signed distance field in a regular grid enclosing the data [38, 41]. Then the surface is reconstructed using the marching cube algorithm [92]. The various approaches differ in how they estimate the signed distance based on the input data.

- Finally, some algorithms work by *deforming* an initial approximation of a surface under the effect of forces and reactions, used as constraints [140, 108].

Simplification

Simplification is mandatory when one has to manage the meshes produced with 3D scanning devices. The sampling resolution of current scanning instruments is up to 10 sampled 3D points per squared millimeter; producing surface meshes composed by 20M-100M faces is therefore common. Meshes of this size usually have to be reduced to a more easily manageable size to be used in real applications. Although commercial rendering hardware is growing faster and faster, performances are still not adequate to many applications. On the other side, applications are becoming more and more demanding, and this trend is likely to go on at least for the near future. Moreover, simplification techniques are the core of multiresolution approaches; the various level of details of a given 3D models are generated trough simplification.

All simplification algorithms rely on the same basic observation: data acquired by range scanners are sampled uniformly, and thus the triangle density in the reconstructed surface depends on the sampling resolution of the device, not on the complexity of the shape. Simplification algorithms take advantage of this observation in order to exploit redundancies and reduce the number of mesh triangles representing low-curvature areas.

The simplification approaches proposed in literature can be characterized with respect to the following factors:

- updates are performed on a local or global basis

- capability to bound or precisely estimate the accuracy loss

- preservation (or not) of the geometry and/or topology

- preservation (or not) of the texture or pictorial information

- termination criterion used

- input/output domain

Local or Global Approach Simplification methods that adopt a local approach perform a series of partial updates on selected areas of the mesh; global approaches apply the simplification criterion to the whole mesh in a single action.

Bounded or Precisely Estimate Approximation Error An important characteristic of simplification algorithms is the ability to accurately estimate, and possibly bound, the approximation error. This is usually achieved by means of different heuristics. All approximation error measures estimate shape (i.e. geometry and/or topology) degradation, and some take into account attribute accuracy loss. They differ mainly in the way they weight shape vs. attribute error in the choice of the next primitive update.

Geometry and/or Topology Preservation Another fundamental aspect is the preservation of geometry or topology of the input mesh. Geometry preserving methods simplify meshes based on the extraction of a subset of the vertex set, while geometry is not preserved by those approaches that re-sample vertices (i.e. move vertices from their original position). The choice between methods that preserve and others that do not preserve the geometry depends on the specific application. For example, some applications such as those which deal with datasets of scalar fields evaluated at the mesh vertices have to re-sample the scalar field if the geometry is not preserved. On the other hand, re-sampling often leads to shapes with a better approximation accuracy, e.g. by moving vertices on the lines of maximum curvature. Similarly, the choice between topology preservation or not, is a matter of application. If the speed of the simplification process is the main concern, algorithms that do not preserve topology are better suited because they are often faster. On the contrary, if the global aspect of the shape has to be preserved, methods that preserve topology should be chosen.

Termination Criterion Simplification algorithms differ in the termination criterion adopted. Namely, there are two possibilities:

1. given an error threshold ε, the algorithm halts when the global approximation error reaches (or is close to) ε;

2. given the size of the final mesh, the aim is to minimize the approximation error ε.

Some simplification schemes exist that offer both termination criteria.

Input/Output Domain Most methods manage simplicial meshes, but only few of them can manage non-manifold surfaces. Given a triangle mesh as input, the output can be a triangle mesh too, possibly with a different topology, or some different representation, such as a series of B-spline parameters that fit a smooth surface. Some algorithms return a simplified triangle mesh together with a texture/bump map that encodes pictorial or shape details removed by the simplification process.

2.2 Acquisition and mapping of surface attributes

Beside the raw geometry, to present a photorealistic view of the 3D models, it is necessary to reproduce in a convincing manner the optical characteristics of the object surface. Real-world objects are usually composed of a number of different materials that often show subtle changes even locally. Photorealistic rendering of such objects requires accurate measurements of the reflection properties of each material, as well as the spatially varying effects.

Traditionally, a lot of research in the appearance area has been focused on the measurement and classification of individual materials for quality control and specification purposes, mostly in industrial applications.

A lot of measures for individual physical properties of surfaces have been derived and corresponding measurement tools have been developed (see Hunter and Harold [66] for a good introduction). Due to their focus on single materials and specialized measurement devices these techniques are however not applicable to the acquisition of whole objects with varying surface properties.

Figure 2.4: 3D model of "Minerva di Arezzo". Left: the raw geometry. Middle: texture mapping obtained from digital photos. Right: realistic rendering using acquired BRDF with the method proposed by Lensch et al [85]

The appearance acquisition and appearance representation of real-world objects has recently received a large amount of attention in the computer graphics and computer vision community. The common approaches to appearance acquisition can be grouped into different categories: texture acquisition, light field, measurement of spatially varying BRDFs. Most of these approaches are proof-of-concept or research

prototypes and almost none of them is currently widely used in a practical applications.

2.2.1 Texture Mapping

The most trivial method to add surface appearance to the raw geometry is to use photographic detail mapped on the 3D model surface. Texture mapping, firstly presented by Catmull in his PhD thesis (1974)[28], is a consolidated technology, nowadays implemented by all graphics library and supported at hardware level also by consumer graphics hardware.

Texture mapping principle is very simple; the color of the 3D surface is stored in an image (the *texture*), when the object is rendered, for each surface point the texture image is accessed to retrieve the object color in that point. The mapping relies in the existence of a *parametrization* of the 3D model; it is necessary to have a function that maps the object coordinates (that are in a 3D space) onto the texture image pixels (2D space).

In the last few years, research in the texture mapping field are mainly related to the *automation* of the texturing process of the 3D models acquired with 3D scanning and to the 3D model *parametrization*.

Although it is possible to find commercial software, like Rapidform [139], that provides texture reconstruct modules for real-world digitized objects, a fully automatic texture building process, able to work with all kind of objects, is still very difficult to be archived.

Many approaches have been proposed for the automatic texture building of acquired 3D models: typical input data is a complete 3D model and a series of photos of the object. What is normally done is a multi-step algorithm that can be summarized as follow:

image registration to obtain a correspondence between pixel colors on the image and 3D points on the model surface, it is necessary to retrieve the position (plus orientation, focus distance and radial distortion) of the camera that acquired the photo. This data can be either returned by the 3D scanner hardware, or it can be reconstructed by: performing a registration based on the selection of corresponding point pairs between image and 3D model [142]; using the results produced by an integrated geometric- and image-based alignment [6, 4]; or finally by adopting a silhouette-based registration approach [84, 69].

color extraction and texture building once a correspondence has been constructed, it is possible to select for all zones of the 3D model a part of an image that contains the appropriate color. Since there can be model parts that are represented in multiple photos, the choice should pay attention to the image orthogonality to that model part (more orthogonal photo direction means less

distortion) and to the absence of shadows or highlights. When all the object surface has been covered, all the extracted parts are arranged in a single texture image. Neugebauer and Klein [103], use the images to compute the texture by evaluating a color on a given point from a weighted mean of neighbors images. A disk-shaped subdivision of the surface mesh is built in [98] to reduce the possible image distortion inherent in any *flat* image to *curved* mesh mapping.

frontier smoothing the color sampled for a single zone on the object can differ from an image to another (due to different lighting condition, white balancing and so on) making possible to see frontier borders between model parts with color information coming from different images. This kind of discrepancy should be eliminated in order to obtain a continuous, realistic texture. Lensch, Heidrich and Seidel [84] perform a local interpolation in the frontier zones between different images. In Callieri, Cignoni et al. [21] a full texture color uniformation is performed, to eliminate completely luminance discrepancies.

Overcoming Texture Problems

Certainly, textured 3D models looks much more similar to real-world objects than their geometry-only counterparts, but this is mainly a perceptual issue, we are not used to see the naked geometry of objects but our perception is always mediated by the color, lighting condition and optical properties of the object. When manipulating a textured model, however all the faults of this technology appear clearly; the main problem of the poor photorealistic quality of this simple texture mapping is that *it relies on common photos*.

What firstly catch the eye looking at a texture mapped object is the strong difference from the real object in terms of rendition; color contrasts that in reality appeared evident are now soothed down, original object is always more "alive". This happens because a simple photo cannot reproduce all the subtle luminance/chrominance changes that are present in the real world. To address this problem, there exists techniques to acquire the full *dynamic range* of real-world scene. An analysis of High Dynamic Range imaging is presented in section 2.2.1.

But even using High Dynamic Range photos, the resulting texture can present artifacts related to local lighting problems like casted shadows or highlights that changes drastically from a photo to another. An idea is to use a huge number of photos, sampling object appearance from any direction to reduce such discrepancy; this approach is called *Light Field Sampling*. Drawbacks are the humongous amount of space required for storing all those images and the impossibility to obtain a precise correspondence between the photos and the 3D model. A very short presentation of Light Fields techniques can be found in section 2.2.2.

Figure 2.5: Texture Mapping Example. Results obtained using our algorithm [21]. On left the textured 3D model, on upper right the input images, on lower right the texture image

Conversely, instead of raising the number of photos, it is possible to process the photos in some manner to eliminate the disturbing artifacts and recover some real surface properties. Since object geometry is known and we want a coherent lighting all over the texture, reverse lighting calculation (assuming the object to have a simple optical behavior) can be used to reconstruct the original, "real" color of the surface. Deshading methods are explained in section 2.2.2.

A further step ahead is to effectively figure out how the surface react to incident light, obtaining the true optical properties of the surface material; in this way it will be afterward possible to recreate every possible illumination. What is needed is a way to represent how much of the incident light is reflected in each direction; this is called *Bi-directional Reflection Distribution Function*. In section 2.2.3 is discussed BRDF theory and its application.

High Dynamic Range

A first problem with photos is the brightness space reduction; nor the photographic film nor common digital CCD are able to capture the full extent of luminance level that are present in the scene. When a single exposure is used, that means that only a subset of the luminance range is represented on the photograph, the parts darker than the lower captured limit will appear black and the parts brighter than the upper limit will be white. The **dynamic range** of a scene is the contrast ratio between its brightest and darkest parts. For example, the interior of an ornate cathedral with light streaming in through its stained-glass windows presents a very high dynamic range. In fact, any scene in which the light sources can be seen directly is high dynamic range.

Figure 2.6: High dynamic range imaging. Left: HDR luminance represented in in false colors raw geometry. Middle and Left: from the HDR data, images with different exposures can be generated

A **High Dynamic Range** image is an image that has a dynamic range that cannot be directly shown on a standard display device because their limited range, or that cannot be captured with a standard camera with just a single exposure. HDR images also have the important property that their pixel values are *proportional* to the amount of light in the world corresponding to that pixel, unlike most regular images whose pixel values are *nonlinearly* encoded. In practice, high dynamic range pixels use floating-point numbers, capable of representing light quantities of one to a million and beyond. Low-dynamic range images usually represent pixels using eight bits per channel, with pixel values ranging as integers between 0 and 255.

It is possible to overcome this problem in two ways: recently, enhanced HDR digital cameras [96][61][37] have been developed, able to directly capture images with higher color resolution with respect to normal CCD (12bit or more per channel). The main problem with this kind of hardware is the very high cost and the difficulty to use it in a non controlled laboratory-like environment.

The other way is to sinthetize HDR images starting from data acquired using common hardware. For example, if multiple photos are taken using different exposure, it is possible to use them to reconstruct the full brightness range; in non-digital photography this process is known as “bracketing”. Debevec and Malik

[44] proposed an interesting algorithm to recover the effective camera capabilities from photos with different exposure and then use this response curve to obtain the true (absolute) luminance at each point of the image. The method relies on the knowledge of the various exposures used.

As said before, common hardware can only represent 8 bit for each color channel: this is not enough to display HDR images. For this reason, it is necessary to use particular techniques to manage this kind of data. In [70], high-dynamic range textures are decomposed into sets of 8-bit textures. These 8-bit textures are dynamically reassembled by the programmable graphics hardware. These operations allow the exposure level of the texture to be adjusted continuously and arbitrarily at the time of rendering, correctly accounting for the gamma curve and dynamic range restrictions of the display device.

2.2.2 Light Fields

One category of methods that aim on representing real world objects are Light Fields [87][126] which are able to capture the view dependent appearance of the object.

The general principle of light field techniques is to capture all "light" rays emanating from an object under a given illumination. This is measured by a large number of images from various view points. In order to display the object from another view point the images of the nearest view points are selected and blended appropriately.

In this category, there has been a number of approaches ranging from a relatively sparse set of images with a geometric model [45] over the Lumigraph [126] with more images and a coarser model to the light field [87] with no geometry and a dense image database. In general, the less geometric information is provided the more images are required. The same point on the objects surface will be visible at different locations in different images, resulting in so called ghosting artifacts when different images are blended. If the geometric model is given, the exact pixel location of a surface point can be calculated for each new view, avoiding most of such artifacts.

Light fields store all images in a database and reconstruct any new view directly from the acquired data without any intermediate representation. This direct interpolation requires a high sampling rate (several hundreds of images per object) [42] in order to reproduce view dependent effects, e.g. highlights moving smoothly over the surface. Although the storage cost can be significantly reduced by compression [57], it is still high and does not allow for remote access or high-resolution meshes management and rendering.

Furthermore, light fields can only capture the view dependent effects, i.e. the digitized object can be viewed from any direction, but it will always show the object in the environment where it was during the acquisition. It is not possible to adapt the incident lighting to new "virtual" environments.

In contrast to light field approaches, bidirectional texture functions (BTFs) [71]

also allow for changes in the lighting conditions, although at very high storage costs. BTFs acquire the view and lighting dependent spatially varying appearance of more or less flat samples of a given complex material by a large set of images showing the sample from different view points and different incident light direction lit by a directional light source.

While BTFs are typically used to represent a single although complex material, reflectance fields are used to capture the viewing and lighting dependent appearance of an entire object. The acquisition technique is however quite similar. Debevec et al. [42] describe a method for acquiring the reflectance field of human faces. A video stream of the face is captured while a point light source spins around his face.

Texture Deshading

Another kind of processing can be used to recover what, in informal terms, is called the "real" color of the object: the *unshaded diffuse component* of the surface. When a photo is taken, the resulting image contains not only the object original "color" but also shadows that can be produced by other objects or by self-occlusion and highlights, depending from the specular component and from light position. Moreover, the resulting color itself is greatly affected by the lighting condition. Those problems causes the 3D model to look "freezed" in the lighting situation in which it has been acquired, changing lighting direction or intensity will cause visual artifacts like double shadows or presence of false highlights.

To overcome this pre-lighting effect, various approaches have been presented to reduce this effect by using ad-hoc imaging systems [7, 124, 149, 87] where both the lighting stage and the image post-processing are designed to support the acquisition either of the albedo of the object or of an approximation of its radiance. Using a calibrated lighting setup, Rocchini et all [122] employed a method based on reverse lighting that was able to find and eliminate shadows and highlights, recovering an approximation of the original diffuse component for the probed surface.

The final color that appears on the photo can also be modified by the intrinsic limitation of the camera. This kind of color aberration can be corrected using a color reference table [100] to evaluate the real color response of the camera.

2.2.3 BRDF

To understand the concept of BRDF (*Bi-directional Reflection Distribution Function*) and how BRDFs can be used to improve realism in interactive computer graphics, we begin by discussing what we know about light and how light interacts with matter.

In general, when light interacts with matter, a complicated light-matter dynamic occurs. This interaction depends on the physical characteristics of the light as well

as the physical composition and characteristics of the matter. For example, a rough opaque surface such as sandpaper will reflect light differently than a smooth reflective surface such as a mirror.

Firstly, when light makes contact with a material, three types of interactions may occur: light reflection, light absorption, and light transmittance. That is, some of the incident light is reflected, some of the light is transmitted, and another portion of the light is absorbed by the medium itself. Because light is a form of energy, conservation of energy tells us that

$$incidentlight = reflectedlight + absorbedlight + transmittedligh$$

For opaque materials, the majority of incident light is transformed into reflected light and absorbed light. As a result, when an observer views an illuminated surface, what is seen is reflected light, i.e. the light that is reflected towards the observer from all visible surface regions. A BRDF describes how much light is *reflected* when light makes contact with a certain material. Similarly, a BTDF (Bi-directional Transmission Distribution Function) describes how much light is *transmitted* when light makes contact with a certain material.

In general, the degree to which light is reflected (or transmitted) depends on the viewer and light position relative to the surface normal and tangent. Consider, for example, a shiny plastic teapot illuminated by a white point light source. Since the teapot is made of plastic, some surface regions will show a shiny highlight when viewed by an observer. If the observer moves (i.e. changes view direction), the position of the highlight shifts. Similarly, if the observer and teapot both remain fixed, but the light source is moved, the highlight shifts. Since a BRDF measures how light is reflected, it must capture this view and light dependent nature of reflected light. Consequently, a BRDF is a function of incoming (*light*) direction and outgoing (*view*) direction relative to a local orientation at the light interaction point.

Additionally, when light interacts with a surface, different wavelengths (*colors*) of light may be absorbed, reflected, and transmitted to varying degrees depending upon the physical properties of the material itself. This means that a BRDF is also a function of wavelength.

Finally, light interacts differently with different regions of a surface. This property, known as *positional variance*, is most noticeably observed in materials such as wood that reflect light in a manner that produces surface detail. Both the ringing and striping patterns often found in wood are indications that the BRDF for wood varies with the surface spatial position. Many materials exhibit this positional variance because they are not entirely composed of a single material. Instead, most real world materials are heterogeneous and have unique material composition properties which vary with the density and stochastic characteristics of the sub-materials from which they are comprised.

Considering the dependence of a BRDF on the incoming and outgoing directions, the wavelength of light under consideration, and the positional variance, a general

BRDF in functional notation can be written as

$$BRDF_\lambda(\theta_i, \phi_i, \theta_i, \phi_i, u, v)$$

where λ is used to indicate that the BRDF depends on the wavelength under consideration, the parameters θ_i, ϕ_i, represent the incoming light direction in spherical coordinates, θ_o, ϕ_o the outgoing reflected direction in spherical coordinates, and u and v represent the surface position parameterized in texture space.

Though a BRDF is truly a function of position, sometimes the positional variance is not included in a BRDF description. Instead, it is common to see a BRDF written as a function of incoming and outgoing directions and wavelength only (i.e. $BRDF_\lambda(\theta_i, \phi_i, \theta_i, \phi_i)$). Such BRDFs are often called *position-invariant* or *shift-invariant* BRDFs. When the spatial position is not included as a parameter to the function an assumption is made that the reflectance properties of a material do not vary with spatial position. In general this is only valid for homogenous materials. One way to introduce the positional variance is through the use of a detail texture. By adding or modulating the result of a BRDF lookup with a texture, it is possibly to reasonably approximate a spatially variant BRDF.

There are two classes of BRDFs and two important properties. BRDFs can be classified into two classes: isotropic BRDFs and anisotropic BRDFs. The two important properties of BRDFs are reciprocity and conservation of energy.

Basically energy conservation property says that if the sense of the travelling light is reversed, the value of the BRDF remains unchanged. That is, if the incoming and outgoing directions are swapped, the value of the BRDF does not change. Mathematically, this property is written as

$$BRDF(\theta_i, \phi_i, \theta_o, \phi_o) = BRDF(\theta_o, \phi_o, \theta_i, \phi_i).$$

As stated before, energy conservation tells us that no light can be created in the reflectance process; quantity of light reflected from the surface is less or equal the quantity of incoming light (the difference is the quantity of light absorbed). Mathematically, the sum over all outgoing directions of the BRDF times the projected solid angle must be less than one in order for the ratio of the total amount of reflected light to the incident light to be less than one. This is written as

$$\sum_{out} BRDF(\theta_i, \phi_i, \theta_o, \phi_o) cos\theta_o dw_o <= 1$$

The term isotropic is used to describe BRDFs that represent reflectance properties that are invariant with respect to rotation of the surface around the surface normal vector. Consider a small relatively smooth surface element and fix the light and viewer positions. If we were to rotate the surface about its normal, the BRDF value (and consequently the resulting illumination) would remain unchanged. Materials with this characteristic, such as smooth plastics, have isotropic BRDFs.

Anisotropy, on the other hand, refers to BRDFs that describe reflectance properties that do exhibit change with respect to rotation of the surface around the surface normal vector. Some examples of materials that have anisotropic BRDFs are brushed metal, satin, and hair. In general, most real-world BRDFs are anisotropic to some degree, but the notion of isotropic BRDFs is useful because many classes of analytical BRDF models fall within this class. In general, most real-world BRDFs are probably more isotropic than anisotropic though many real-world surfaces have subtle anisotropy. Any material that exhibits even the slightest anisotropic reflection has a BRDF that is anisotropic.

Another simplification to the generic model can be obtained eliminating the wavelength dependence. For computer graphics utilization, every color is expressed as the combination of three basic colors (Red Green and Blue); this convention is used by both the image acquisition as well as rendering devices. For this reason it is possible to sample the BRDF for just that three frequencies; this is done very easily, since most of the acquisition devices like digital cameras outputs RGB information.

In this way the resulting function depends only from the light source and from the viewing direction $(\theta_i, \phi_i, \theta_o, \phi_o)$, reducing the BRDF from a 7D to a 4D function.

BRDF Representation

Beside the physical definition, to effectively use BRDF in lighting calculation, it is necessary to define a proper representation:

- **sampled BRDF data**: when a BRDF is measured from a real material, what is obtained is a 4D matrix containing the sampled values. This matrix can be directly used as a lookup table for rendering,
- **analytical model**: to obtain a more compact representation, the BRDF function can be approximated with some mathematical models. In this way it is not necessary to store the 4D matrix and the needed value can be calculated on the fly

BRDF analytical models can be divided into two varieties: BRDF *models* and BRDF *approximations*. A BRDF model is a reflection function that attempts to model the observed scattering behavior of a class of real surfaces, usually in a physically motivated manner. A survey of several models can be found in Shirley et al. [104]. Most BRDF models have a fixed number of parameters. Each parameter has some physical meaning, usually relating to some characteristic of the class of surfaces to which the model applies. With most models it is possible to choose parameter values that break the physical plausibility constraints.

The Phong model [110] is an empirical model that creates a highlight, but breaks all the rules of physically plausibility introduced in Section 2.2.3 (reciprocity and conservation of energy). Physically plausible variations of the Phong and other

models have been proposed. The microfacet-based Equation is a generalization of many models (Torrance and Sparrow [141]; Blinn [11]; Cook and Torrance [116]; He, Torrance et al. [62]). The Torrance-Sparrow model has been used in acquisition systems [130] [149]. Debevec et al. [42] used a modified Torrance-Sparrow suited for human skin. The Ward reflection model [81] uses an elliptical Gaussian sharpness function, and can thus create anisotropic highlights with an elliptical cross-section. Banks proposed an empirical anisotropic model that is not physically plausible.

In contrast to these models, BRDF approximations represents the BRDF by a projection onto general basis functions. Examples of basis functions that have been used for BRDFs in computer graphics include spherical harmonics, spherical wavelets, and Zernike polynomials. Spherical harmonics are spherical analogues of sines and cosines. They are localized in the frequency domain, but are global in the spatial domain.

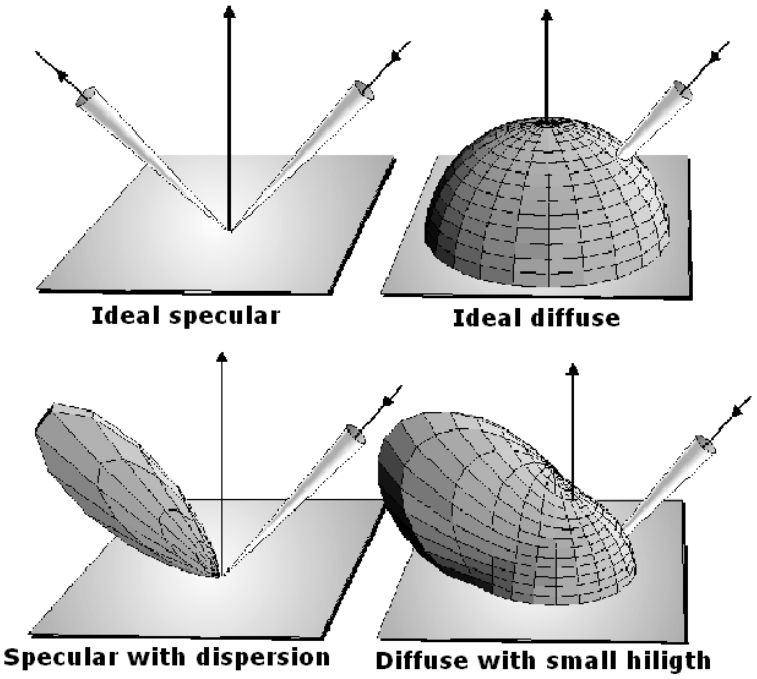

Figure 2.7: Top row: **ideal** light-matter interaction, the light is totally reflected toward specular direction or diffuse equally in every direction. Bottom row: **real world** interaction, quasi-specular surfaces presents some scattering from the specular direction (it is often addressed as material *Hardness*) and quasi-diffuse presents some small highlights too.

With considerable modification, Westin et al. [144] remap spherical harmonics from their spherical domain to the hemisphere domain of BRDFs. Zernike polynomials are used in optics to characterize lenses and have also been applied to BRDF representation (Koenderink, Doorn [76]). As with spherical harmonics,

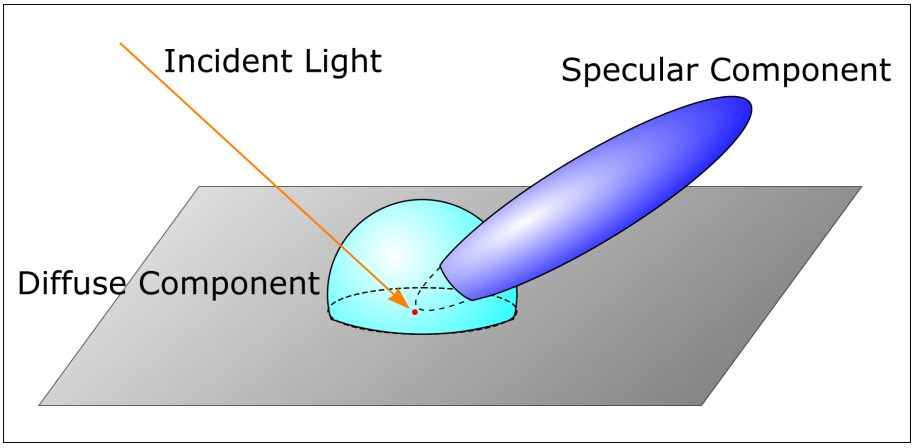

Figure 2.8: BRDF Lobe approximation (**Phong**). A part of the incoming light is diffused (quasi-hemispherical lobe), another part is reflected toward the direction specular to incidence(ellipsoid lobe).

Zernike polynomials are not spatially localized. One advantage of polynomials and spherical harmonics is that they are a linear representation, so the coefficients can be computed in closed form using a linear least squares approach. Schröder and Sweldens introduced spherical wavelets [132], and proposed their use to represent the exitant hemisphere of a BRDF with a fixed incident direction. Their method was extended to the full BRDF domain (Lalonde and Fournier [79]). Spherical wavelets are spatially localized but are multiresolution, so evaluating the BRDF still requires evaluation of coefficients at a full range of scales.

BRDF Acquisition

As we said before, BRDF represents the quantity of light reflected from the medium parametric to incidence and outgoing direction, it is then necessary to perform a 4D sampling.

In the case we are interested in the reflectance properties of a single material a particular equipment called **gonioreflectometer** can be used. With this device, shown in fig 2.9, it is possible to measure the BRDF of a small sample of material. The instruments, with a series of moving elements, can measure object reflectance for each pair incoming-outgoing angle. Obviously, this kind os measure is very slow and is not possible to use this kind of instruments for generic objects.

Moreover, using such BRDF for rendering simulates a material that have a constant optical behavior all aver the surface making the object appear as made of a single, *uniform* material. It is possible to find on the internet many material BRDF datasets acquired in this way, there also exist companies that sells accurate samples of commercial building materials.

In his PhD thesis, McAllister [99] has constructed a modified gonioreflectometer, able to capture the BRDF of many points on a surface at the same time by moving a small square sample in front of a calibrated camera. In this way it is possible to capture position-variant BRDF.

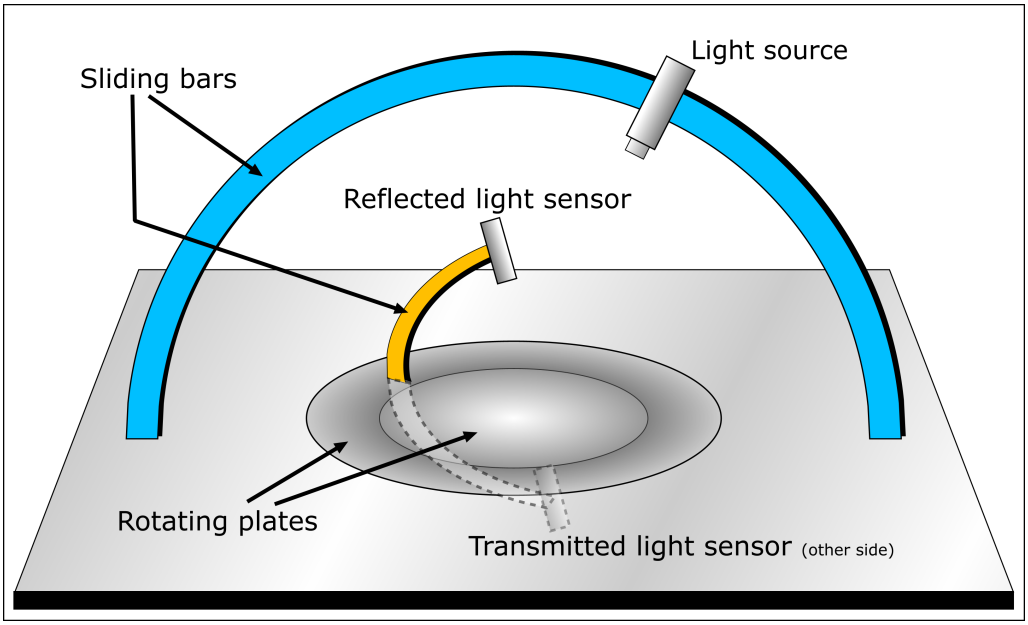

Figure 2.9: A gonioreflectometer scheme: a sample of the material is placed in the middle of the rotating plates and it is possible to obtain all incident ray - outgoing reflection pairs by sliding the light source along the outer arch, rotating the two plates and moving the inner arch. With a second sensor on the lower part of the inner arch it is possible to capture also the material BTDF.

A more suitable approach has been taken by Debevec et al. [42] who constructed the light stage where a point light source spins around the object while a video camera takes several hundreds of images for a fixed view. Hereby, the BRDF of each visible surface point is captured for exactly one viewing direction as a function of incident lighting.

More recently, image-based approaches have been proposed. These methods are able to acquire a large number of samples at once. For example, Ward Larson [81] uses a hemispherical mirror to sample the exitant hemisphere of light with a single image. Instead of using curved mirrors, it is also possible to use curved geometry to obtain a large number of samples with a single image. This approach is taken by Lu et al. [117], who assume a cylindrical surface, and Marschner et al. [127] who obtain the geometry using a range scanner.

A number of researchers have described methods for fitting reflection models to measured sample data [42] [130] [78] [99] [149] [81]. Of these methods, the ones by Ward Larson [81] and Lafortune et al. [78] do not consider spatial variations. Sato et

al. [130] fit a Torrance-Sparrow model [141] to the data, and consider high-frequency variations for the diffuse part but only per-triangle variations for the specular part. This is also the case for the work by Yu et al. [149], which also takes indirect illumination into account. Boivin and Gagalowicz [12] reconstruct arbitrary reflection properties for whole patches in a scene using just one image. McAllister [99] fits a spatially varying Lafortune model to very densely sampled planar materials. The achieved results are impressive but the technique requires flat surfaces and an automated setup to get a dense sampling of the reflection properties. In Ramamoorthi and Hanrahan [119] and Gibson et al. [59] inverse rendering algorithms are proposed that reconstruct the reflection properties and the incident light field at the same time. Ramamoorthi and Hanrahan [119] as well as Westin et al. [144] project BRDF data into a spherical harmonics basis instead of fitting an explicit reflection model.

Figure 2.10: BRDF acquisition using Lensch et al. method [85]: the sampled object is illuminated with a strong quasi-punctual light source, the surface response is captured using a digital camera. Both light and camera position are calculated using the mirroring spheres

Lensch et al. [85] compute a spatially varying BRDF of an object with known geometry. They assume that many points over the surface share the same BRDF and by only handling isotropic BRDFs they require only about 15-30 images per object (Figure 2.10). Their work focuses on finding a few basis BRDFs and representing each texel as a linear combination of these bases, with small per-texel detail. Thus, their system only applies to surfaces with a few different isotropic BRDFs.

Beside the acquisition technique, if the BRDF is not constant for the whole object, it is also necessary to map the obtained BRDF to the 3D model (to describe the material change across the surface); this is done practically in the same way as

the texture mapping process. A texture is built where, for each part of the object, is specified not just the color of that point but also its optical behavior.

BRDF Rendering

As we said, a BRDF can be either represented as a 4D matrix or by an approximate mathematical model; in both of this cases it is necessary to use some optimization techniques to make it possible the real time rendering of 3D models with BRDF surface characterization.

As opposed to a regular 2D texture, a BRDF represented as a lookup table is highdimensional and involves very large amounts of data. Store and access this data in an efficient way is a crucial task to make it possible the real-time use of this surface model. To the contrary, when the BRDF is represented as analytical function, the problem moves from storage space to computational optimization of the lighting calculations.

One of the major enhancements of 3D graphic hardware in the last two years is the introduction of programmable graphics card; more than processor evolution, the new generation graphic cards have opened the door for creating stunning photorealistic interactive 3D content. The concept of programmable GPU (Graphical Processing Unit) has been introduced by NVidia in 2002; graphic cards programming has evolved from a rigid assembly to a flexible C-like language [97].

Since its introduction, this kind of technology has received lot of attention from the research world. With the use of this kind of hardware is possible to obtain a very complex surface appearance rendering like Lensch and Gösele [63] or to perform heavy calculation otherwise CPU bound like Purcell et al [115] or Krueger and Westermann [77].

The new programmable hardware is the ideal platform to use the BRDF, with this technology it is possible to do lighting calculation for every rendered pixel, thus obtaining very realistic surface rendition. There exist lot of technical reports on BRDF lighting implementation from the various hardware producers like NVidia [36]. When BRDF is expressed as a analytical model, the lighting expression should be implemented using the specific hardware programming language. As the hardware evolves, more complex analytical models can be used without adding significant overhead to the rendering process.

When dealing with a sampled BRDF is a 4D matrix, ideally it would be nice to be able to use a hardware accelerated 4D lookup table (a 4D texture) to store the BRDF and then at run-time do a per-pixel texture lookup to compute the lighting equation. Unfortunately, the current generation of graphics hardware does not provide support for 4D textures and moreover, this kind of storage waste too much space. As a consequence, it is necessary to come up with a clever way of computing the same expression using the features of currently available graphics accelerators. One possible solution will be to, in a preprocessing stage, take the

4D BRDF and separate it into the product of two 2D matrices using, for example, Normalized Decomposition technique. Then at runtime a pair of 2D texture lookups can be modulated together to compute the BRDF weighting function as described by Kautz in [73] and in [74].

2.3 Efficient rendering of large datasets

The need for interactively inspecting very large surface meshes, consisting of hundreds of millions of polygons, arises naturally in many application domains, including 3D scanning, geometric modeling, and numerical simulation. However, despite the rapid improvement in hardware performance, these meshes largely overload the performance and memory capacity of state-of-the-art graphics and computational platforms. A wide variety of simplification methods and dynamic multiresolution models have been proposed to face the problem, but, unfortunately, none of them is able to perform both scalable simplification and interactive view-dependent visualization of very large meshes without imposing a lossy decimation of the original dataset [91].

Rapidly rendering adaptive representations of large models is a very active research area. In the following, we will discuss the approaches that are most closely related with our work. For a wider overview, it is possible to refer to recent surveys (e.g., [134]).

2.3.1 Out-of-core mesh simplification

Various techniques have been presented to face the problem of huge mesh simplification. With the exception of memoryless clustering approaches [90, 91]and stream-based methods [146, 68], most of these techniques, such as Hoppe's hierarchical method for digital terrain management [65] and the octree based structure OEMM [34], are based on some kind of mesh partitioning and subsequent independent simplification. Hoppe hierarchically divides the mesh in blocks, simplifies each block while freezing borders and then traverses the block hierarchy bottom-up by merging sibling cells and again simplifying. In this approach some of the borders remain unchanged until the very last simplification step. OEMM avoids this kind of problem, but it does not build a multiresolution structure. On the other hand, BDAM [31] allows both the independent processing of small sub-portions of the whole mesh and the construction of a multiresolution structure, but is limited to height fields. Our work generalizes this approach to arbitrary surfaces, and parallelizes the simplification process in order to efficiently build a multiresolution structure for very large meshes.

2.3.2 View-dependent triangulations

The vast majority of view-dependent simplification methods for general meshes are based on constructing a graph of possible refinement/coarsening operations at the vertex or triangle level. Early methods used edge collapse [147, 64] or vertex clustering [50, 93] as primitive operations, and assumed in-core hierarchy construction, limiting their applicability. Few techniques have been presented for both construction and rendering from external memory. Hoppe's hierarchical terrain management method [65] was an early example, later extended to arbitrary meshes [113]. More recently, El-Sana and Chiang [51] proposed a technique for segmenting the surface and ordering edge collapses to handle block boundaries without explicitly imposed constraints. Both methods have only been tested on models of a few million polygons, and their scalability is unclear. Lindstrom [91] recently proposed a scheme for out-of-core construction and visualization of multiresolution surfaces based on vertex clustering on a rectilinear octree. While it significantly improves over earlier approaches, it is still unable to retain the fidelity of the original mesh and is heavily CPU bound at rendering time, with a peak performance of 2M triangles/s and asynchronous hierarchy updates at no more than 1 frame/s.

2.3.3 Efficient host-to-graphics communication

All adaptive mesh generation techniques spend a great deal of rendering time to compute the view-dependent triangulation. For this reason, many authors have proposed techniques to alleviate popping effects due to small triangle counts [65, 35] or to amortize construction costs over multiple frames [64, 48, 91]. Our technique reduces instead the per-triangle workload by composing at run-time pre-assembled optimized surface patches. The idea of grouping together sets of triangles in order to alleviate the CPU/GPU bottleneck was presented also in the RUSTIC [112], CABTT [86], and BDAM [31] data structures for terrains, and HLOD [52] for general environments. RUSTIC and CABTT are extensions of the ROAM algorithm in which subtrees of the ROAM bintree are cached and reused during rendering. BDAM constructs instead a forest of hierarchies of right triangles, in which each node is a general triangulation of a small surface region. These methods produce adaptive conforming surfaces but are hard to generalize to surfaces with arbitrary topology. HLOD improves instead the classic LOD scene graph by providing multiple precomputed levels of details not only for each model but also for entire subtrees, however, HLOD focuses on the run-time handling of a large number of small unconnected objects, typical of large CAD assemblies.

2.3.4 Point rendering approaches

An alternative to mesh refinement is to use multi-resolution hierarchies of point primitives to render highly complex scenes in output-sensitive time [125]. Since current rendering hardware is optimized for triangle rendering, high quality filtered point splatting requires considerable effort, and high visual quality has often to be sacrificed for rendering speed. The latest approaches try to solve this problem by exploiting the programmability features of modern GPUs, leading to impressive peak performances that range from 50M points/second for low quality rendering with unfiltered splats [39] to 10M points/second for high quality filtering [15] for in-core models of a few million polygons. By using cache coherent triangle strip primitives, we are able to exceed such rates even for out of core models that are two orders of magnitude larger.

Chapter 3

Color mapping on large 3D datasets

As stated in the introduction, the starting point of our work is the availability of a faithful digital representation of the geometry of a real object. Mainly, 3D models we will work on will be produced using 3D scanning. 3D scanned models gives a very accurate description of the object shape, but usually this description is limited to geometry. If we want to perceive the object similarly to how we can perceive in reality, this description should be enriched.

The first step in performing a more realistic visualization of a 3D model is to characterize the digital model surface, in order to overcome that feeling of flatness and unreality often generated by the use of the infamous "grey plastic" material.

As presented when talking about surface characterization (Par. 2.2), the basic option to have a more faithful rendering of a real object is to display it with its apparent surface color. The easiest way to obtain this surface attribute is extracting information from photos taken at the object. Texturing from photos has been a topic of interest for research for a long time, many algorithms are available in literature to accomplish this task.

As discussed in Chapter 2, 3D scanning received lot of attention from the research world in the last years. It is nowadays technically possible to obtain very complex 3D models of real objects, thanks to the new scanning hardware and processing software. Generating a 15-20 million triangle model of a real object is quite easy; as easy as, with modern digital cameras, producing a photo set of the same object that can be much larger than 100 MPixel.

Given the size of those datasets, the problem of mapping color attributes is now also how to efficiently manage all the available data. Many of the standard techniques for color mapping and texturing are unable to work with such large datasets. For this reason we need an approach able to scale with the dataset size.

Beside the problems introduced by data management of large photographic data sampling, the main problems remain how to deal with color incoherence between different photos due to illumination changes during photo acquisition and the pres-

ence of view dependant artifacts. These problems are quite common when doing photographic mapping, some previous approach exists, but again we need to define methods able to work with large datasets. The second requirement is then to find an algorithm able to deal efficiently with such color coherency problems.

Like during 3D model generation (Par. 2.1.2), we have to face for each surface point the problem to integrate multiple samples coming from various probings and decide a consensus value. The most straightforward solution is to select one of the candidates (using some metrics) as correct value. This however, neglects the rest of available data and can produce discontinuities in the final mapped color. Since redundancy is the best way to resolve intrinsics sensor error, we want to exploit this redundancy using a blending function to obtain a more correct color to map.

Figure 3.1: A 18 million faces 3D model with color coming from a 520 Mpixel dataset (64 photos)

3.1 Description of the mapping process

Obviously, to map photographic information from photos to a 3D model, it is necessary to obtain a correspondence between the 2D image domain to the 3D model domain. Photographic process is based on prospective projection. Reversing this projection, it is possible to correlate point of the image to the 3D geometry. However, to do so, it is necessary to recover the parameters of the camera at the moment of the shot.

Different approaches exist for recomputing the inverse projection needed to solve the inverse mapping (from an uncalibrated photo to the 3D space) [142, 84, 46]. The correctness of this phase is essential; imprecisions in the image registration can result in very poor mapping results. The most common effect of a poor registration is the presence of *ghosting*. In the areas of the 3D models where different images overlap, some photographic detail can be seen multiple times in slightly different position.

In our lab, we developed a tool [56] for image registration, able to work with very large 3D models and many hi-resolution images. This tool goes beyond the concept of single-image-to-geometry alignment used by other algorithms: it introduces the possibility to use the connections between images, which in some cases are much more stronger or easy to detect visually than the ones between the image and the 3D model. Moreover, the tool implements a novel system to assist the user in the alignment process by computing the minimum set of correspondences which have to be selected. The system is based on a graph representation where the nodes represent the pictures and the 3D object, and arcs encode correspondences. This graph is used to infer new correspondences from the ones specified by the user and from successful alignment of single images.

Since it is possible to associate 3D areas with image pixels, the problem shift then to the subsequent phase, i.e. how to process redundant pixels data and how to efficiently map such information on a high resolution 3D model.

One of the more common approach is to build up a parametrization of the 3D mesh well fitting the pool of images available, and producing a new texture map [14, 103, 84, 21]. Unfortunately, the management of very large geo-pixel samplings is very complicated. Their texture-based approach is ideal when we have both low-to-moderate resolution meshes (50K-1M faces), usually produced by simplification or subsampling, and moderate pixel datasets (1M-10M).

For this reason, we will focus on exploiting the high geometric detail of the 3D model by using a per-vertex color mapping. This kind of storage may seem insufficient to preserve the detail present in the images. However, since we are working with very large models, vertex density can be enough to obtain a good mapping. In Figure 3.5 it is possible to see how a vertex-mapped detailed model can show sharp photographic detail and how slow the quality of the mapping decrease, even for a smaller model.

To deal with color incoherencies and produce a correct color mapping, we choose an approach where a multivariate blending function is used to weight all the available

pixel data w.r.t. geometric, topological and colorimetric criteria. This blending function (which operates in the image space) allows to mix weighted pixels from the various images. In this way we aim to maximize the use of information contained into the input images, to correct incoherence between different photos while avoiding the blurriness caused by simple blending.

In Section 3.4 we will show how the weighting function is constructed, combining various metrics, each one focused to address a particular aspect of the images: view angle, distance from sensor, defocusing and so on. In Section 3.6 we will discuss how to detect quality problems in the source dataset and how to correct them.

In Section 3.5 we will show how this function can be used to map the color information to the geometry, by choosing either a texture-based approach (which requires mesh parametrization) or by adopting a multiresolution per-vertex encoding. We show in Section 3.7 how this process can be implemented in an out-of-core way, to deal with huge geometric and image datasets. Then, in Section 3.8 we will present some results of color mapping relying on per-vertex encoding, since it makes possible to use all the data available by the adoption of modern multiresolution approaches; moreover, it results more robust w.r.t. the need to manage incomplete, topologically dirty and complex meshes (which are, unfortunately, the standard results of 3D scanning).

3.2 Mapping approaches

Various approaches have been proposed to select the most correct color which has to be applied to each parcel of the 3D model. As previously stated, the most standard approach is to compute a texture by assembling subparts of the original input images. Additionally, some corrections can be applied to deal with incoherence in the borders between different images.

An example of this class of approaches is our previous texturing tool, *weaver* [21]. In that case the texturing process followed more or less a standard pipeline, common in various literature algorithms. Notable contributions of the tool were the hardware-accelerated visibility calculation, used to map surface points to corresponding images and the final heuristic correction able to reduce the incoherencies between different images. According to visibility computations and some greedy optimizations, the tool would subdivide the mesh in patches and assign to each patch a section of one of the input images. These image sections would then be packed in a texture image minimizing space but maintaining image coherence. Finally, to eliminate color discontinuity between input images, the tool would apply a correction based on cross-correlation, directly on the texture image space.

The *weaver* tool was developed in 2002, when the "standard" size of a 3D model was around 500k - 1M faces; the development of the tool considered this range as the application target. This affected various design choices, providing solutions

that were correct at the time, but that subsequently limited its applicability. The tool, like other similar approaches, rely on the topology of the 3D model to manage the patching and parametrization process. This method provides good results, but unfortunately it is unusable on very large 3D models for the difficulty to generate the topology and its prohibitive storage cost. Moreover, the presence of topological noise can result in problems during patch generation and optimization. Erroneous geometry greatly affects the color mapping process and the final step of discontinuity elimination by introducing false data in cross-correlation.

Technical advance in 3D scanning, GPU and data manipulation software have made this limit grow at a very fast pace. Even if the tool has been update often, e.g. by introducing an out-of-core management for textures and image data, the tool is not able to work with the large datasets we have today as standard output of acquisition campaigns. We believe that this approach has reached its technical limit, for this reason we decided to find a different approach to color mapping.

Conversely to the approach just outlined, other methods generate color mapping without reassembling parts of the original images, but calculating color values (using various processes) and filling the texture map texel by texel (or assigning colors/materials to the 3D model). An example of this approach is [149], where the texture is filled with values coming from an inverse rendering process based on the original images. In the same fashion, we will use the input photos as a source to calculate the correct colors that will be used to fill texture map texels or to perform per-vertex coloring. This calculation will be done using a per-pixel blending function.

The idea of using a per-pixel blending function is not completely new; some other approaches that uses a per-pixel weight have been proposed. Both [4] and [3] uses a weighted blending function to compute the texture color, but without exploiting all the potentiality of this method. Per-pixel weighted blending has also been used in image processing, as showed in [120].

However, we are focusing on the definition of an extensible and flexible framework for image blending. We need to overcome the two main problems related to color mapping from photographic data: (a) difficulties related to large datasets management and (b) difficulties related to images discordancy. We propose a way to weight and blend multiple images that is able to work with an arbitrary amount of input data. We will examine benefits of the per-vertex color encoding, in terms of quality and compactness, while not excluding the algorithm applicability on standard texture mapping. Moreover, given a complete and clean definition of various different weighting schemes, a good data arrangement and application rules, we will show that it is possible to obtain very good results in color mapping, overcoming most of the image-to-image incoherences.

3.3 The Masked Blending Function

The objective is to be able to work with photos taken in arbitrary conditions since, in most cases, having a calibrated de-shaded and artifact-free photo dataset it is not a viable option. This is especially true when we are not allowed to work in controlled laboratory conditions, as is the usual case of valuable artworks. In many cases the data acquisition takes places in museums or, when working on architectural heritage, in open air conditions.

The main idea is to create a blending function (that operates in the image space) capable to mix data from the various images, weighting the quality of each contribution. If for all images we have a value for each pixel expressing its quality, the most correct color to apply can be computed as the weighted mean of all color sources.

The projective mapping principle is well known. Since the photos taken with the camera follows the laws of optics, knowing the camera parameters it is possible to determine if (and where) a point on the surface is mapped inside the image boundaries. The camera model generally used is a simple perspective projection. Some precautions should be used to deal with distortion introduced by lenses, but the problem is not so stringent, as there are various methods to undistort an image and often we are working with photos taken with a tele lens, which minimizes the distortion problems.

A depth map (produced from the 3D model rendered with the same projection parameters of the given photo) is necessary to discriminate, similarly to shadow mapping, if a point on the surface that projects inside the image is in fact visible from that point of view or occluded by other geometry. A different approach is to use a ray-tracing approach to resolve visibility issues. While much slower, in theory it guarantees a better precision. However, we found that the depth map approach is quite accurate, especially on video cards able to use a 24 bit depth buffer.

Using this method it is possible to assign a color, taken from the photo, to a point onto the surface. The real problem is when the same surface point can take the color from many sources images.

Since there is more than one candidate, it is not possible to simply choose the color from a single image, ignoring the other values. But also doing a simple blend could not be a good choice, because not all sampled colors yields the same degree of quality. Since to each pixel in the source images can be assigned a quality (defined by various metrics), it is necessary to take into account this quality while blending. The blending function we need is basically a weighted mean of all possible sources. The problem is now to choose a good weighting method.

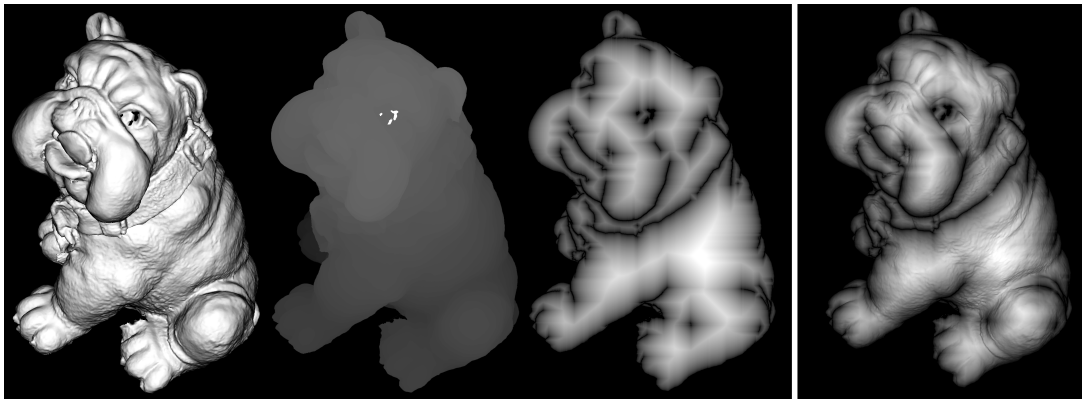

Figure 3.2: An example of the core weighting masks. From left to right: Angle Mask, Depth Mask, Border Mask. Rightmost, all the masks combined in the final mask. Caveat: the contrast of the depth and border masks has been increased for enhanced readability

3.4 The weighting mask generation

The core of the blending function is the weighting mask that is generated for each image. The weight mask states the quality of each pixel and, consequently, how much it will contribute to the final color of the 3D point it is mapped on. Various metrics can be applied to evaluate the quality of the image pixels; moreover, since we have not only the images but also a faithful geometric representation of the object, we can effectively use this information to perform a better evaluation.

We can evaluate a pixel quality using many different metrics. It is possible, for example, to state that each metric is relative to a different characteristic of the source images. For example, pixel quality can be measured based on camera orthogonality to that surface point; because more orthogonal areas appear less "stretched". But a pixel that is "good" for this particular metric can at the same time be considered "bad" by another metric that considers the distance from the sensor (farther surface, less dense sampling available in the image). We will then define multiple metrics, and mix them all in a single measure.

It is important to state that, since we need a per-pixel blending, all the masks we will use have the same resolution of the corresponding photo image. We will show how, having the camera parameters, it will be possible to calculate each mask with a series of controlled renderings and buffer processing. As we said, to perform the color projection we need a depth buffer (to test for visibility), this buffer will be calculated at the begin and used as a base for the other masks.

The weighting mask will be generated assembling various metrics through multiplication. In this way we conserve for each mask its continuity and minima. Preservation of minima is very important to remove outliers. If a given pixel is considered "bad" according to a given metric, we should be sure that it has the least probability

of being used.

Here follows the description of the basic weights used in our system.

Angle mask: the simplest quality evaluation; the weight of a particular image pixel is proportional to the angle between the viewing ray and the surface normal. Similar to lambertian illumination, the quality is higher when the view direction is orthogonal to the surface and lower when glazing. The computation of this mask is quite straightforward: for each pixel we need to know its surface normal and position. We have the image view point stored in the camera data, it is then possible to render the object with the same camera parameters, to obtain an image with pixel-to-pixel correspondence to the original photo. If we use for each vertex a color that encodes its normal, the result will be a per-pixel normal map. The depth map we calculated at the begin can be used to recover the position of each image pixel (again, with inverse projection). Having the camera point of view, it is then possible to calculate the viewing angle of each image pixel.

Depth mask: the weight of an image pixel is higher as the surface is closer to the camera. This is an approximation of the ratio pixel/surface (informally, how dense is the information on the image). Obviously this ratio depends also on orientation, but this has already been taken in account when calculating the Angle mask. Calculation is, again, quite easy, using the same depth map that has been calculated at the begin of the process. Using directly the depth map would imply a linear quality decrease due to distance. However, we had better results in using a squared distance, a better approximation of the light attenuation, that is exponential.

Border mask: this mask measures in what measure a pixel in the image is far from a discontinuity in the depth map (silhouette borders) or from image borders; farther we go, higher the quality. The first step is to mark the discontinuities of the depth map (calculated using a Kalman filter) and the image borders as point of zero value. For the rest of the pixels, the weight is calculated as the geodesic distance from the borders.

An example of the core masks described above and the final assembled weight is showed in Figure 3.2.

While the meaning of the first two weights is not difficult to understand, the third one is a bit more tricky. We use the depth map as a starting point because some discontinuities on the photo are more easily detectable when looking at the 3D models. A discontinuity on the depth map indicates a silhouette point for that point of view; on the corresponding image those pixels will map information of two non contiguous area, with problems of coherence in color, focusing and depth. Image borders are problematic too; it is well known that (thanks to lenses imperfections) the quality of picture is minimal toward borders. These two pixel regions should

ideally not contribute to color determination; the situation improves the farther we go from those problematic areas. Moreover, the geodesic distance calculation ensure continuity on the weights, that is also a nice property.

3.4.1 Additional masking

The most interesting feature of this approach is its flexibility; the system works by evaluating multiple metrics and then mixing them in a single quality measure. In order to improve our system, we can add more masks to address for specific dataset problems.

The masks described in the previous section are the most important ones, i.e. the *core masks* of the system. These masks are always applied to all datasets, because they evaluate measures that are related to photography in general and not to a specific dataset.

In our experience, we found that there exists two recurring problems in the dataset we processed. For this reason, we implemented two additional masks, that can be used if the dataset presents some particular problems. The two optional masks are the *stencil* and *focus* mask.

Stencil mask: in some cases there is the need to exclude certain parts of the source image (e.g. people in front of a building, photographic artifacts). In those cases a simple stencil masking can be applied. Images we need to correct are (manually) marked with a particular color, the system will assign a zero weight to those pixels.

Another situation that often occurs when processing photos is to detect focusing problems. When doing a photographic coverage of an objects there is the need of being close to the object to obtain enough detail. This closeness can results in problems whit the autofocus system, out of focus areas cannot be used for mapping, otherwise we will obtain a blurred result. There can be two main problems:

- An entire image can result not perfectly in focus, i.e. when the camera cannot autofocus correctly because it is too close to the surface or because the object of interest is very small.
- A part of the image can be out-of-focus, because the depth extent covered by the image is large, and some parts of the object falls beyond the focusing range of the camera.

These two problems can be resolved with the focus mask:

Focus Mask: to build this mask the focusing of each pixel of the image is evaluated in a way that is almost identical to the procedure used by digital cameras

Figure 3.3: Focus masking example. An image of the wooden statue with large depth extent. As detected by the focus mask,the hands are out of fucus due to depth of field, while the red part of the vestment is perfectly focused.

while performing autofocus. Typically, the autofocus procedure set the focus distance by maximizing the sharpness of some reference points in the framed image. We are evaluating the per-pixel "focusness" of the source image using as sharpness operator the energy of image Laplacian [138] (applied on a window of 20×20 pixels).

Figure 3.3 shows an example of focus masking; in this case the problem focus problem was due to excessive depth of field. With this masking the out of focus areas will have a very low weight. In this way they will not introduce blurring when blended with other images, but they still be available as color source if no other data can be mapped on the same surface part.

3.4.2 Weights normalization

Mixing the various masks into the final one using a simple product is fine to preserve properties of the different weights without numerical problems. What, however, can happen is numerical cancellation in the weighed mean when applying the function (Section 3.5). If weights are quite discordant (very large values vs. very small values) multiplying color values (normally in the range 0-1) for the weight, accumulating

and then dividing by the sum of all weight can produce artifacts even when using float (or doubles). As the number of used masks increases, it is necessary to be cautious to select the range of the weights; having all mask between 0 and 1 would be the best thing. This is easy to obtain for angle mask (it is already 0-1), it is possible to obtain on border mask, just considering the distance from border (in pixels) normalized with respect to the image size (in pixels). For the depth mask this can be obtained by normalizing with respect to the range between the minimum and maximum distance found among all images; this is however time consuming (require visiting all depth), experimentally we found that for object whose photos does not contain much depth of field this step can be ignored.

3.5 Application of the weighting function

Given a point on the 3D surface, it is easy (as described in Section 3.3) to determine the set of source images that can effectively "see" that surface point. The source colors will be multiplied by their weight and accumulated, using as final color for that point the weighted mean.

A very nice characteristic of this function is its generality: given a point on the surface, a color is returned. It is then possible to use it to fill a texture map or, a much simpler approach, since no parametrization is required, to calculate the per-vertex color of a 3D model. If a parametrization is present for the 3D model, there is a full correspondence between texels and points on the 3D surface; to fill the texture it is necessary to calculate the function for each of its texels. For other applications, a per-vertex coloring (obtained by evaluating the function for each vertex) can be sufficient.

The images showed in this chapter are all obtained from per-vertex colored models. When dealing with densely scanned objects, it is common to have a sub-millimetrical geometric representation. This detail can be used even in realtime, thanks to multiresolution and out-of-core rendering techniques, coupled with modern video cards. It is then worth to try using this geometric density to store the color information, without affecting the rendering quality too much. In most cases, during rendering, each triangle will be projected to less than 2-3 display pixels. A per-vertex encoding is then more then enough for a realistic visualization. This is especially true when working with multiresolution engines; doing rendering, for each part of the scene is selected a different resolution level. The selection is often based on the projected size of the triangles of that particular area.

Using all the redundancy contained in the image dataset, we obtained an high quality color mapping. Even if we are blending multiple images, high frequency details are not lost thanks to the weighting function. Image areas with more detail will have a higher weight, that will dominate over other lower, more blurred, images. At the same time, discrepancy between different photos is no more detectable.

Figure 3.4: An example of per-vertex color mapping. The 3D model is composed by 1 million triangles, with a mapped photographic dataset of 8 images 2560×1920. From left to right: original photos, geometry, color+shading, color only.

Blending the overlap area between different images ensures there are not apparent frontiers.

Since the color blending function relies on redundancy to determine the most correct color to apply, a good level of overlapping is essential to overcome strong color discrepancies due to illumination changes or color biasing between different images. Having all points of the surface covered by at least 3 image pixels can be enough to correct most of this problems. This may seem a strong requirement, but often, when doing a 3D scanning, a similar dense sampling of the surface is done. When not much overlap is present, or some images presents a poor color coherence with the others, some artifacts can be perceived in the resulted mapping. The next section presents a partial solution to this problem.

3.6 Image/Color Correction

As stated before, the weights express the quality of the pixels. For each image is then possible to evaluate its usefulness in terms of average, min and max quality. If an image results much lower in quality with respect to the other, the user receives a

warning and has the possibility to exclude the image from the process or to further decrease the weight vales for the entire image. The second option is quite helpful if the image is the only one that covers a section of the 3D model; eliminating it completely would cause a non-mapped area, while reducing the weight (to 1/2 or 1/4) will greatly reduce the influence of its overlapping area, while conserving its unshared data.

The usefullness of this feature should not be underestimated: when dealing with very large dataset (the one used for the David 3D model was 64 images), detecting imperfections and problems among all those images can be a difficult task.

A better error evaluation can be done regarding the color coherence. Every time we evaluate the blending function we sample the contributing images to retrieve the source colors (with associated weights) and then we compute the correct color as the weighted mean. The difference between the sampled color and the final color is a measure of how much the source image was coherent with the other neighbors. We can determine, for each image, how severe is this incoherence by keeping track of all those mapping errors. If we detect that the error value is above a threshold, it is possible to mark that image as discordant.

In this case it is also possible to provide a correction for the erroneous image. For each pixel sampled on this image, we know which is the color that, after blending, is mapped onto the 3D surface. Each one of those color couples represents a color transformation; we are interested in a global correction able to make the image more coherent with its neighbors. This global color transformation can be expressed as a 4×4 *color correction matrix* M, that can be evaluated by solving a linear system showed in Equation 3.1.

$$\begin{bmatrix} R_1\, G_1\, B_1\, 1 \\ \vdots \\ R_n\, G_n\, B_n\, 1 \end{bmatrix}_{sampled} \cdot \begin{bmatrix} \\ M \\ \\ \end{bmatrix} = \begin{bmatrix} R_1 G_1 B_1 1 \\ \vdots \\ R_n G_n B_n 1 \end{bmatrix}_{final} \tag{3.1}$$

The vector of sampled colors is multiplied by the correction matrix M (the unknown) to obtain the vector containing the final colors.

The correction matrix is applied to the image and the result is presented to the user; if the user believe the new image is better than the older, the image dataset is updated and the mapping process is repeated. This color correction is done, again, taking in account of the weight associated to the pixel and computing the matrix in the XYZ color space [30], to guarantee a more stable correction.

It is notable that, while this kind of correction is technically sound, in practice it could fail under certain circumstances. This is because the color couples used to build the matrix are not well distributed in the color space, since they are only associated to the colors of the object. In this case, the linear system used to compute the color correction matrix could become ill-conditioned and could introduce some

color-shifting problems. A similar approach for color correction is used in [1], but also in the dataset presented in their paper a similar color-shifting artifact is present. This is the reason why we choose to have the system prompt the correction to the user for confirmation, instead of doing the correction automatically.

To avoid this kind of color problems, it is much advisable to perform a color calibration *on the field*, during image acquisition. Using a color calibration chart (for example, the Macbeth chart) can result in a much more exact color with a low processing cost. In our project we always apply this strategy, but we had to implement this correction considering image dataset coming from other sources.

3.7 Large Dataset Management

Executing the proposed pipeline for color reconstruction from photos could require excessive memory resources on very large datasets. The problem of dealing with large dataset depends on the size of both the 3D geometry and the photographic data. To perform the projection with an "all in memory" approach it is necessary to have loaded the 3D geometry and for each photo the image, the associated depth map and the weight mask (both float vector, same size of the image). It is easy in this way to exceed the memory capacity, it is also impractical to set up a system to load and discard the whole images and the maps as needed, since there is not much locality and the size of "chunks" is quite big. It is much more efficient to calculate and organize the data in a way that is easy to access on disk.

The mask calculation can be done independently for each image, so we can calculate one mask at a time, keeping in memory just the 3D model, one image, one depth buffer, one weight mask and the temporary data used during calculation. In this way it is possible to generate all the data necessary to apply the blending function; the data for each photo can be stored in a single file with a tight packing.

The packing strategy of the buffers (image, depth and weight) is based on the type of process we have to do.

When dealing with per-vertex coloring approaches, there in no guarantee of access locality: the vertex ordering into the 3D model (or into the multiresolution structure) can be arbitrary. For this reason, rearranging all the mapping data to follow the vertex ordering can be a waste of time, loosing as much time as we would gain speeding up the subsequent access phase. In that case we use the simplest encoding: for each contributing image there is a file with an header containing the camera data and, the pixel data stored linearly. For each pixel we store the color values (from the image), the depth value (for the shadow-mapping projection) and the mask value (for blending).

Conversely, when the we are applying the weighted blending to generate texture map texels, there is a strong locality that should be exploited. The majority of texels will access the same subset of images (and possibly, in the same area) with

Figure 3.5: Marble capital. Top row: 6 million triangles, Bottom row: 1 million triangles. Photographic dataset 8 images 1800×1200. Even reducing the complexity of the 3D model, the mapping quality remains high; quality degradation due to the per-vertex approach starts to appear on the smaller model as we get very close to the object (the eye of the small figure is 6mm wide).

respect to its neighbors. A better data arrangement can be used to speed up the access to the data; in literature are available advanced storing strategies for OOCore algorithms (like [107]). However, since our problem is not so complex, even a simpler approach can grant an effective speedup. It is possible to arrange the image data in square "chunks" that can be read and stored in memory using a simple LRU strategy. The size and number of chunks can be selected according to the dataset size and the available memory. but they are out of scale to applied

The second step, the actual application, is almost identical to the in-memory implementation. When calculating the blending function for a point, the 3d point is tested for each image to see if it falls in the image frustum and if its normal is front facing with respect to the camera view direction; if not the image is skipped, otherwise the data is retrieved from the OOCore file and used for the projection as usual. In the case of the first, simplest encoding, a file seek is necessary for each pixel access. In the other case, if the chunk containing the pixel is in memory, is

accessed directly, otherwise the loaded set is updated and then accessed.

This implementation still need to maintain the whole 3D model in memory. This is normally possible, since the 3D data is much less than the image data and the additional buffers. However, the real potentiality of the per-vertex approach is to work in conjunction with multiresolution or out-of-core technologies, producing colored geometric datasets that can be browsed in realtime with high visual quality. It is then necessary to integrate the mapping process with an existing multiresolution structure. Given the simplicity of the process, this can be done for most of the existing data structures. The points in the algorithm where the geometric data is needed are two:

- During weight generation: it should be possible to render the dataset (at full geometric resolution) with the correct camera parameters to retrieve the depth map associated to each image. It is also necessary to render a normal map (used for the Angle mask calculation).
- During mapping, it is necessary to access all vertices in the structure and using their coordinates (and normal) to evaluate the blending function. The structure should also be able to store all computed colors

We used this approach to implement color mapping for our multiresolution data structure [33]. we had no problems in obtaining the renderings needed for weights generation, just a minor change to have a normal map rendering. For the latter phase, when color is generated, we used the internal data arrangement: since the geometry is stored as patches, we can load each patch in main memory, process all its vertices and store back the results in the data structure.

3.8 Mapping Results

To test the effectiveness of this approach we tested it with various datasets. In all cases, the input images has been previously registered to the geometry with a computer assisted approach [56]. All the 3D models have been produced using 3D scanning technology, so they are not watertight nor topologically clean. The images here included shows some of the models obtained; in all cases, just the per-vertex color has been calculated. This because the size of some of the objects and the difficulty of calculating a good parametrization. In Table 3.1, are listed the objects with details on their size, available images and time required for color mapping.

Additionally, to demonstrate the possibility to manage a very large dataset, we selected a test case of an extraordinary size. Mapping on the Michelangelo's David the two complete photographic campaigns (pre and post restoration). The 3D model comes from the scanning campaign of the Digital Michelangelo Project, it is composed by 18 million triangles. This is a simplified version of the original

Figure 3.6: Life-size painted wooden statue. Left: shaded+color, center: color only, right: detail. 3D model of 5 million triangles, photographic dataset 33 images 1694×2496.

56M triangles model, merged at 1mm resolution; simplification was necessary to remove all the almost degenerated faces produced by the Marching Cube merging. The two photo dataset were taken by a professional photographer before and after the restoration of the statue in 2004, they are composed respectively by 60 images 1920×2560 and 64 images 2336×3504 (294 and 523 MPixels).

The time needed to generate the OOCore data necessary for color blending was 30 minutes for the first dataset and 50 minutes for the second one. The size on disk of the temporary data was 3.6 GB and 6.2 GB respectively.

The second phase, color blending and mapping for the vertices of the model, took more or less 2,5 hours for both dataset (in this phase, time is almost only dependent on number of function evaluation).

The results are visible in the images included (Figures 3.7 and 3.1), there are still parts without color, but this is caused by a lack of photographic coverage. This is because the photographic campaign was aimed to the production of a book and

Table 3.1: Color Mapped objects

Object	Size(triangles)	Images	MPixels	Time
Dog	1 M	8 (2560×1920)	39	3 min
Capital	6 M	8 (1800×1200)	17	8 min
Painted statue	5 M	33 (1694×2496)	139	15 min
David (pre)	18 M	60 (1920×2560)	294	3.0 hr
David (post)	18 M	64 (2336×3504)	523	3.5 hr

to document the restoration process. Unfortunately, as people with experience in 3D scanning knows well, to obtain a complete surface coverage is a very hard task to accomplish.

3.9 Geometric Refinement

In the previous section, we stated that per-vertex encoding of color does not introduce a visible loss of information. This is true, assuming that the density of geometric information has more or less the same density of the photographic sampling. This equality ensures that, even when using the per-vertex approach, we will not lose information. if, conversely, the geometry is coarser than the photographic detail, the per-vertex mapping will cause a loss of information.

Geometric density depends on the acquisition sampling density; 3D scanners and photo cameras can have very different sampling densities. This disparity will affect the mapping phase. For some categories of objects, it is easy to an equivalent sampling rate. Medium size objects (let's say, from 10 cm to 2m) acquired with triangulation laser scanners yield a geometric resolution of around 0.5 - 0.2 mm. It is easy, with modern cameras, to obtain similar sampling for such objects. For example, for the statue in Figure 3.6, we had a half millimiter resolution geometry and a slightly higher (0.3mm) photographic resolution. Mapping the color on the maximum size model would not produce a visible loss of pictorial information.

A different problem arises for larger objects, maybe acquired with less precise scanners, like time-of-flight. An example of this situation are the buildings; a geometric resolution of 1 cm is considered high resolution, but the photographic information can easily be one level of magnitude higher. The same problem can happen when the photographic campaign has been carried out in a very detailed way: using macro lenses, it is possible to acquire photographic detail of even 0.1 mm.

However, as we stated at the begin, to have a clear idea of the amount of information perceived by the user it is also necessary to consider the view distance. When the projected size of the triangles is around 3-4 pixels, any quality loss will

be virtually undetectable for the user.

It is not difficult to stay below this triangle-pixels ratio in most of the applications, object visualization is generally not so close to the surface to generate this problem. However, a close inspection can be needed for some models, so it's possible for the user to get so close to the object to perceive the approximation due to the per-vertex mapping.

We have in this way restricted the problem to the cases where the geometry is much coarser than photographic detail and when the viewpoint is really close to the object. Unfortunately, we have no immediate solution for these cases. As an extension to the actual system, we are trying to find a way to deal with this additional information. Considering a multiresolution structure as a tree, with the leaves containing the original, full resolution geometry, for obvious reasons only the leaves will need to have this increased detail level.

A possibility can be the generation of a texture map valid only for the local chunks that are visualized very close to the viewpoint. The multiresolution engine will ensure that each parcel of the object is rendered at the correct resolution; we can think to differentiate geometry-only patches (which are below the triangle-pixel ratio threshold) that are rendered with per-vertex color mapping, and textured patches (above the threshold) that will be leaves with an associated textures. Since texturing will be only required for small patches, the generation of a parametrization will be easier with respect to the parametrization of the whole geometry. Advantages of this approach are the possibility to include as much more detail we need just changing the texture resolution, but at a high computational cost due to texture loading and discarding.

Another approach, would be a static geometric refinement, to continue using the per-vertex color mapping paradigm. After the leaves, it will be generated another level of refined geometry to contain the extra color information. In this way the rendering will follow the same algorithm as before, without any status change. The real problem is the incredible growth in size of the dataset.

Finally, the problem can be faced using some dynamic mesh refinement. There are various techniques for geometric refinement, but we are mainly interested in reducing the CPU load, so we need a technique able to work on the graphics hardware. Refinement by adding new geometry directly on the GPU it is at the moment impossible, since the video card can only modify existing geometry and not adding new vertices or triangles. Partial solution has been described in [16] and [133]. Even if these solutions are not really cheap in terms of execution time, it is again to be considered that only the chunks of the geometry that are very near to the viewer would be rendered in that way.

3.10 Further Developments

As stated in Section 3.4.1, this mapping system can be extended by implementing additional masks to deal with image defects. This feature grants a good degree of flexibility to the system, giving the possibility to write specific code to resolve possible problems we could face when working with future datasets. Moreover, the masks modularity will make possible the update of the existing code without too many problems.

We believe this extensibility feature is very important to keep the pace with increasing size of the datasets, and possibly to make this piece of software able to work with other kind of data. For example it is likely that, in the near future, this kind of processing will use, instead of common photos, HDR images (see Section 2.2.1). Using HDR images will grant a finer representation of lighting and colors, overcoming problems due to different exposure between images. The difference between HDR and normal images, in terms of software manipulation, resides almost only in their encoding and numeric range. All the consideration and the proposed processing methods discussed in this chapter still holds for HDR data. To use this kind of data, it will be then possible to just update the software, without the need of changing the mapping strategy.

A major (unresolved) problem when mapping photos onto a 3D geometry is the presence of shadows. For this reason an interesting goal could be to detect parts of the image that need to be excluded because in shadow. Shadows have been widely analyzed in literature for image and video processing applications and several techniques for shadow detection and removal has been developed in recent years. Most of these techniques rely on color analysis. Suitable color models are exploited to classify in-shadow pixels [58, 128]. In general, geometric information of shadows are less used due to the fact that the geometry of the image is unknown in many cases. Recently, Sato et al. [129] have addressed the problem to estimate the illumination distribution from images with known geometry and unknown reflectance properties (that is our cases). This method could be easily adapted to our purposes: after the illumination distribution is recovered, for each image, the pixels in shadow can be identified with a good degree of precision and the corresponding *shadow mask* can be built.

It is possible to object that, having the geometry and the illumination it is possible to recover the real albedo of the scene trough a deshading process. However, since the recovered illumination is just an approximation and we do not know much about surface characterization, this masking seem a much more realistic result.

Another open issue is how to deal with areas not covered by the photographic dataset, like the top of the head of the David (post-restoration). This is a recurring situation, since it is very difficult to obtain the total surface coverage. The presence of uncovered parts greatly affects the correctness of the visualization, because gives the impression of having a *badly painted* surface, rather than having a representation of the surface material. However, in cultural heritage field it is often asked to not

introduce *artificial* data; information not measured from reality does not have scientific value. For documentation and study purpose it's often preferable to leave holes than to complete it with plausible, but not original, data. The same consideration holds for geometric data.

In those cases when a completion is possible, Texture InPainting approaches [151, 148] can be adopted to synthesize color information for unsampled regions.

3.11 Considerations on Color mapping

As shown in the images, the blending function we described is able to take into account all the typical problems related to the integration and mapping of photographic datasets. With this weighting mask we can efficiently use all the redundancy present in the source images to reduce illumination artifacts and incoherence between different images. It is also notable how this weighting system is extendable, giving the user the possibility to overcome specific dataset problems, (like out of focus areas). As required, this method can be implemented as an Out-of-Core, making it usable also on huge 3D models.

However, taken as it is now, this kind of mapping is just an enrichment of the 3D dataset. The resulting mapped model definitely has a better resemblance with the original object, but is still far from being photorealistic.

The main problem reside in the nature of the information we can obtain from the source images. The color derived from photo mapping is just the observed surface response in a given illumination. In fact, the source photos contains illumination-dependent artifacts like shadows and highlights. Moreover, apparent color on photo depends on other factors, like optical response of the surface and the characteristics of the camera.

If the lighting is not constant in the various available images, is possible, during the blending phase, to obtain a partial elimination of the illumination influence on colors. If, for a given object part, we blend different colors, each one taken in a different illumination, the result is something that is more or less "illumination independent". This is **clearly not** a way to perform deshading, but the obtained results are visually quite good.

For example, the statue in Figure 3.6 has been photographed using three mobile light sources to obtain an almost diffuse lighting condition. Regardless of our efforts, there was still a predominance of the light coming from atop. The resulting mapped color seems almost free of lighting artifacts, but in areas occluded from the upper hemisphere (like the lower part of hands), the color is much darker than in reality.

In this case, if we try to relight this object with a different lighting, we would obtain fairly good results but only if using a lighting coming mainly from the upper hemisphere. This limitation is not as strong as it may seem; in most cases the illumination would be coming from atop (indoor and outdoor).

Moreover, as we found during many on-field acquisition, this level of data gathering and processing is more or less as much as it is realistic to expect possible. More complex color probing procedures, while granting a more complete surface characterization are often too cumbersome to be applied. This because of many constrains regarding time availability, physical accessibility to the object or unusual lighting condition.

Unfortunately, to perform a realistic relighting of the 3D model it is necessary to have a deshaded color information: an albedo map. This is the only way to be able to apply an *arbitrary* lighting.

However, we believe that also in a scenario where a robust deshading technology exist, it is essential to provide a solid (and functional) method to mix and blend the source information. In fact, how described in Chapter 2, many of the techniques for surface material representation are based on calculation on some source image.

Figure 3.7: Details of the David with the two mapped datasets. Leftside: before restoration, Rightside: after restoration. (upper part of head in the post restoration phase was not covered by photographic campaign)

Chapter 4

Improving rendering with Ambient Occlusion

In the context of typical interactive visualization, when very large 3D models are displayed, a simple direct lighting model is the common choice. This because the choice of more complex lighting models would decrease application performances too much. When trying to improve rendering realism in this kind of application is necessary to take in account this resource shortage. The improvements should be integrated into existing rendering engines, possibly without the need of introducing additional data structures and at a manageable computational cost.

Adding per-vertex coloring is something affordable because, beside additional storage cost, the rendering of such attributes has a very low impact on performances because color is a basic vertex attribute, already included in the standard pipeline. When using texture mapping, the cost increases a bit, but again, the modern video card have a dedicated hardware for this purpose.

It is also possible to enrich this basic lighting with a neglectable computational cost, introducing an approximation of the diffuse lighting component. In basic lighting, the surface is only shaded by the direct light received, which intensity only depends on its orientation. The effect of the light not directly coming from the primary light source has to be approximated in some way. Without resorting to more correct (and complex) global illumination solutions, many shortcuts are possible. The most common and cheap solution [111] is to use a simple per-scene constant term, but this approach leads to a notable flatness of the non directly lightened portions of the scene.

This approach has been improved by explicitly computing for each point of the surface the light accessibility value, which is the percentage of the hemisphere above each surface point not occluded by geometry [80]. This approach is commonly known as *ambient occlusion* and it is a useful technique used in many production environments for adding, at a low cost, an approximation of the shadowing of diffuse objects lit with environment lighting. This approach was, for example, used for precomputing a nicer ambient term in our old interactive visualization system

described in [13].

A variant of the ambient occlusion term, called *obscurance*, has been introduced in [152, 67], where the authors propose to exponentially weight the occlusion factor according to the distance of the occluders, in order to enhance the shadowing effects of near occluding surfaces. In all of the above proposals the computation of this extended ambient term is performed by a traditional ray-traced approach.

The difference between a standard rendering with constant ambient and a rendering that considers the occlusion term is clearly visible in Figure 4.4: in this particular case the visual gain is huge, this is due to the fine grain of this object geometry.

4.1 Ambient Occlusion Term

Let us consider a point p on the surface and n_p be the surface normal at p. According to [72] we can define the *irradiance*, E, arriving at p as:

$$E(p) = \int_{\Omega} n_p \cdot \omega L(\omega) d\omega \tag{4.1}$$

where $L(\omega)$ is a scalar with magnitude equal to the radiance arriving from direction ω, and Ω is the set of directions above the surface, i.e. the direction for which $n_p \cdot \omega > 0$. As we will see in section 4.3 this equation can be approximately evaluated by discretizing the domain Ω into a finite number of k sectors $\overline{\omega_i}$ with a possibly uniform solid angle measure $|\overline{\omega_i}|$, and, for each sector, evaluating the radiance L only for a sample direction ω_i:

$$E(p) = \sum_{i=1}^{k} n_p \cdot \omega_i L(\omega_i) |\overline{\omega_i}| \tag{4.2}$$

The above equation can be greatly simplified if we consider a uniform lighting environment (light coming uniformly from every direction as under a cloudy sky). In this case, if we discard diffuse interreflection effects and therefore we take into account only direct lighting, $L(\omega)$ can be substituted by a simple binary function $O(\omega)$ yielding 0 if a ray shoot from p along ω cross our surface (and therefore the light coming from the sky is obscured) and 1 otherwise.

Intuitively speaking this can be considered a simple first order approximation of the whole rendering equation. After these consideration and under assumption of a uniform sampling of the ω directions,

$$A(p) = \frac{1}{4\pi} \sum_{i=1}^{k} n_p \cdot \omega_i O(\omega_i) \tag{4.3}$$

Calculating this kind of shading in realtime is not a viable option, especially when we are manipulating huge 3D datasets. However, since this measure depends on the

object geometry, that is a constant in many realtime applications, it is possible to precompute the ambient occlusion term to use it when doing the final lighting in realtime.

Using only geometric considerations it is possible to determine in which measure a point of the surface is occluded by other parts of the geometry and consequently, how much light will receive (see Figure 4.1). We can store for each vertex this measure, representing how much of the diffuse light present in the scene will reach that particular point on the surface.

All the standard graphics libraries provides a way to use the ambient term by specifying a light amount (how much diffuse light is present in the environment) and a surface ambient parameter (how the surface will react to ambient light). Setting the surface ambient value as the ambient occlusion term, we will obtain the correct shading: more occluded areas will receive less lighting than more reachable parts. This is done at a very low computational cost, since this calculation is already implemented in the graphic pipeline.

This is, obviously, the simplest way to use the occlusion term. It is easy to implement, quite fast and provides good results. However, better results can be obtained integrating this calculation into more complex lighting models. This is especially easy when the illumination model is implemented using graphics hardware shaders; the necessary data is available to shaders (as a vertex attribute) and the computation is not complex to implement and to execute.

It is also possible to store this occlusion value on a texture: the shading algorithm does not change much, beside the different storage and some computation that is shifted from vertex to fragment. However, we can consider again that per-vertex encoding does not imply a quality degradation, since the geometry is detailed enough to produce good results, especially when dealing with very large 3D models. Moreover, diffuse lighting does not require very sharp features.

It is to be noted that we are not considering, at the moment, how good the surface material is simulated during rendering. We are jut considering the amount of light that will reach each point on the surface. This value can just be considered an input for subsequent lighting calculations: used to compute whatever shading function we need, from very simple to more complex lighting models.

4.2 Geometric accessibility

Geometric accessibility is a measure of how easily a surface may be touched by a probe [101]. This property is almost always restricted to spheres; we define "tangent-sphere accessibility" as the radius of a sphere which may touch a surface point while not intersecting any other surface. The size of such a sphere will depend on the surface curvature and the proximity of other parts of the probed object. The geometric accessibility can be used as an approximation of the ambient occlusion,

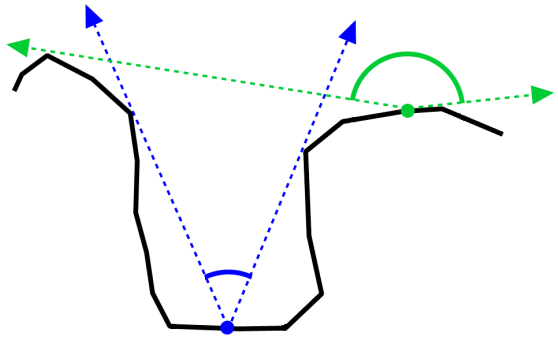

Figure 4.1: Ambient occlusion. Diffuse lighting comes from the whole skydome; light received is proportional to the solid angle of sky that is visible from the point. More exposed areas (green point) can receive light from a larger angle with respect to more hidden areas (blue point).

because zones with a lower accessibility will for sure be reached by less light. The probe size can also represent how much light can "touch" the surface, accessibility can be used to approximate the ambient occlusion term, even if sometimes they are not coherent (see Figure 4.2).

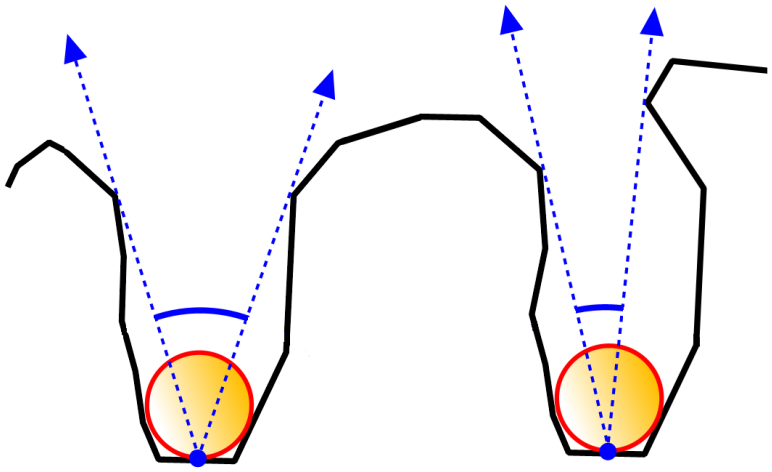

Figure 4.2: Occlusion and Accessibility are sometimes not coherent: the bottom areas of the two valleys have the same accessibility value (orange spheres) but quite different sky occlusion (blue arrows).

Accessibility is an important concept in chemistry since many chemical reactions depend on the access of one molecule to the surface of another. Introduced by Lee and Richards [82] in terms of a "solvent-accessible surface, it is used to determine which parts of a given molecule are accessible to (the spherical atoms of) another molecule. This measure is used to evaluate the possibility (and effectiveness) of

interactions between substances.

Accessibility shading has already been used in literature to enhance visualization by bringing out small surface details like carvings, normally invisible using standard illumination [88, 94]. Areas with an accessibility under a certain threshold would be displayed much darker than other, more accessible, areas. While in those cases this effects has been greatly exaggerated to provide higher contrast (or using another color, like in Figure 4.3), it is possible to use the accessibility value to provide a more realistic shading effect.

When computing accessibility at a particular surface point, the surface normal and position is known. The accessibility could be computed by iterating on the sphere radius using a sphere-object intersection test. Unfortunately, this approach would be very time consuming since the intersection tests need to be done once for each iteration. A second approach is to compute the accessibility analytically, directly from the geometry. Unfortunately, even in this case the cost is still linear to the number of primitives to test (in this case, triangles); this number can be reduced by limiting the maximum accessibility radius we are interested (every accessibility level above a certain degree is considered infinity) and employing strategies for storage and fast search of occluding entities

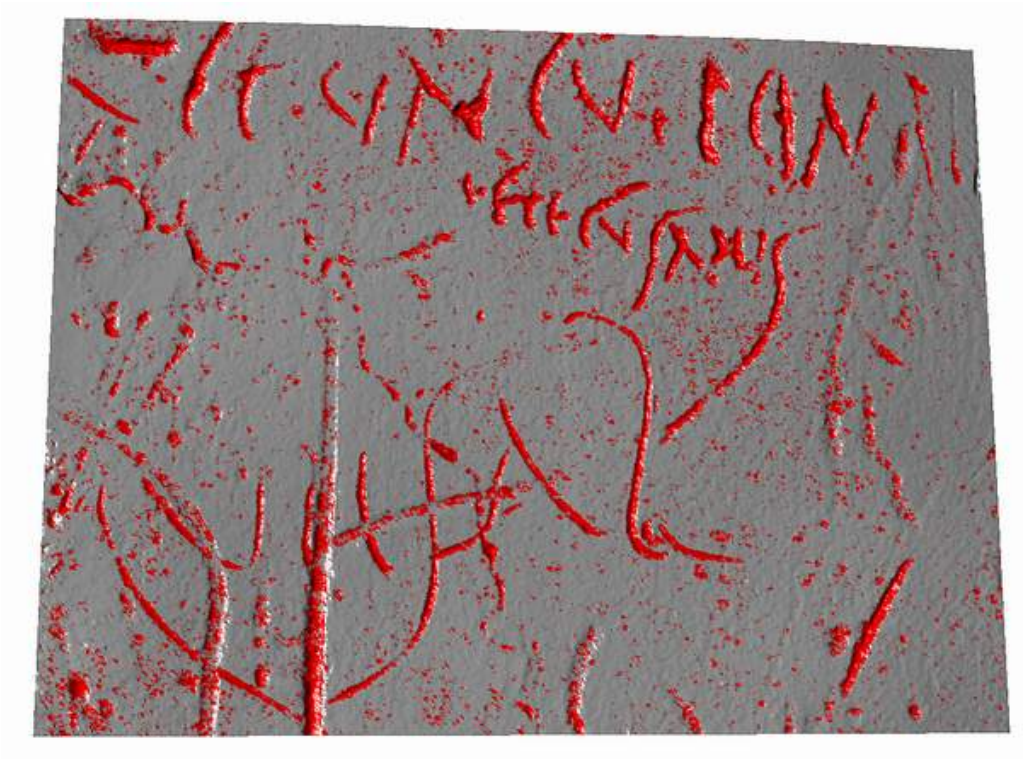

Figure 4.3: A very shallow wall inscription (from Pompei). Once the accessibility has been calculated, it is possible to color low accessible areas for enhanced readability [94]

However, even applying an optimized calculation, the execution time of the algorithm remains quite high, due to the 3D model complexity. Moreover, another problem arises when dealing with huge 3D models: implementing this computation for out-of-core or multiresolution models is not easy and requires even longer execution times due to the secondary memory access.

So, even if this approximation is plausible, for efficiency reasons we need a better way to evaluate the ambient occlusion term; faster and usable on larger 3D models.

4.3 Approximated occlusion

The exact way to compute occlusion is to proceed analytically, using the available geometric information. However, what is normally done, it is just to use rasterization or ray tracing approaches to regularly sample the hemispherical visibility of the surface points. The occlusion ratio is evaluated as the number of probes without intersection with the rest of the object divided by the number of probed directions.

Instead of manually computing occlusion using a ray-tracing approach, we use the graphics hardware to determine visibility. When doing a rendering from a certain point of view, only the parts that are *visible* form that particular point will be drawn on the screen. If we consider that point a light source, those visible parts would be the one able to *receive* light coming from that direction.

Given a good number of view direction distributed over a sphere, we can compute visibility from each of those points. Then, for each point, the number of times it has been visible among all renderings is a measure of how much light that point will receive from an ambient lighting.

Since it is rendering based, most of the calculation are performed by the graphic card. This ensure very good performances even in case of large datasets. This is by far the most easy way to compute occlusion information, but at the same time it grants very good results.

Even if it is a very coarse approximation of the diffuse lighting, it is quite difficult to detect the shading imprecisions just with a naked eye. Since the results of this algorithm are coherent, the user will perceive it as realistic.

Algorithm implementation is quite simple. Generating a series of points over the sphere is straightforward by using a regular subdivision, maybe introducing some randomness to avoid too regular patterns. Viewpoint setup is again easy, both OpenGL and DirectX have specific functions to control rendering parameters. Rendering is done in two steps (like is done for solid wireframe renderings): first, the whole geometry is rendered in solid white; then, the points of the object are rendered using a precise color coding with a slight z-offset.

Given the resulting image, the vertices that are visible from the current camera (i.e. the vertices that can receive light from that direction) are recognizable by analyzing the colors contained in the image. The vertex-to-color encoding is

straightforward, the vertex index is converted in an RGBA color using a simple cast (an *integer* is composed by 4 bytes, the 4 bytes can be independently read as *char* to specify color components).

```
unsigned char* color;
int            index;

color = (unsigned char*)&index;
glColor4ub(color[0],color[1],color[2],color[3]);
```

This approach is very fast using current accelerated graphics boards, easy to implement (with respect to a ray-tracing engine) and it guarantees good results.

On the other hand, its accuracy can be not satisfactory if multiple vertices are projected onto the same pixel (this can happen for very dense meshes or for region nearly parallel to the view direction); in this case, some of the visible vertices cannot be detected by the algorithm. We solve this problem by iterating the rendering process: in each pass we render in color-coding only the vertices not yet detected as visible, and stop the iteration as soon as no more vertices are detected as visible. With this enhancement, the accuracy does not depend anymore on the viewport resolution w.r.t. model density. Other methods to reduce this aliasing problem can be the use of tiled rendering (viewport is divided in different areas, each one rendered independently) or to lightly jitter the point of view. In our experimentations using a combination of these methods, no more than three or four rendering passes where needed, even for very detailed meshes.

The bottleneck of this algorithm is located in the reading back of the buffer from video memory to main memory to be analyzed. This operation is normally quite costly, considering the computational speed of such video cards; it is also true that video cards are designed to process data flowing from the CPU to the video, and not to bring back data. The same problem has been faced by people working with General Purpose GPU; as the video card technology advance this operation is getting faster, but there is few that can be done from the programming side. Off-screen buffers (framebuffer object) can be used to obtain better performance, since reading back the video memory has a slightly faster implementation. Moreover, the size of those buffers can be much higher than the video resolution, thus giving another option to reduce aliasing problems.

Dealing with very large 3D datasets is a problem that can be easily faced with the use of existing out-of-core techniques. The algorithm does not assume to have all geometry in main memory; it is just necessary to perform rendering from a given point of view using a particular color encoding. Almost all multiresolution or out-of-core engines can be used to implement this.

This GPU accelerated algorithm has been implemented firstly for our texturing tool [21], where it was used just to compute the visibility of vertices from some points of view. It was later extended to compute the approximated occlusion term.

Figure 4.4: Using approximated occlusion in rendering. Upper: lambertian illumination. Lower: lambertian illumination plus ambient occlusion term. Model is composed by 4 M triangles, occlusion calculation took 1 minute.

In [131], graphics hardware is exploited to efficiently compute a per vertex ambient occlusion term. The core algorithm is quite similar to our, but instead of reading back the buffer and analyzing the result, an hardware based occlusion query is used to find vertices that passed the depth test and therefore were visible for considered light source. A similar approach based on the use of the programmable GPU for the computation of the ambient occlusion term has been proposed in [109] and extended to the computation of also a first bounce of the diffuse interreflection of light in [17].

4.4 Multiple bounces

What happens in reality when light hits a surface is that part of the light is absorbed, a part is transmitted (if the object is not opaque) and a part is reflected. This reflected light is the one that determines the perceived color of the object, but

the reflected light also bounces from surface to surface, producing a wide range of shading effects (like, for example, *caustics*). The algorithms outlined in the previous sections just consider the first contact between the light and the surface. Also many of the simpler rendering models just consider the first iteration; the received light is used to compute apparent color but the computation stops here.

We are not interested in modeling very complicated lighting effects, but the light bounce also affects diffuse lighting. Areas that can be occluded from light coming directly from the environment, but can be reachable from second or third light bounce. Simple occlusion normally produces very dark areas, this is because just the first light hit is considered. In reality there are no such "black holes" not reachable by light (Figure 4.5).

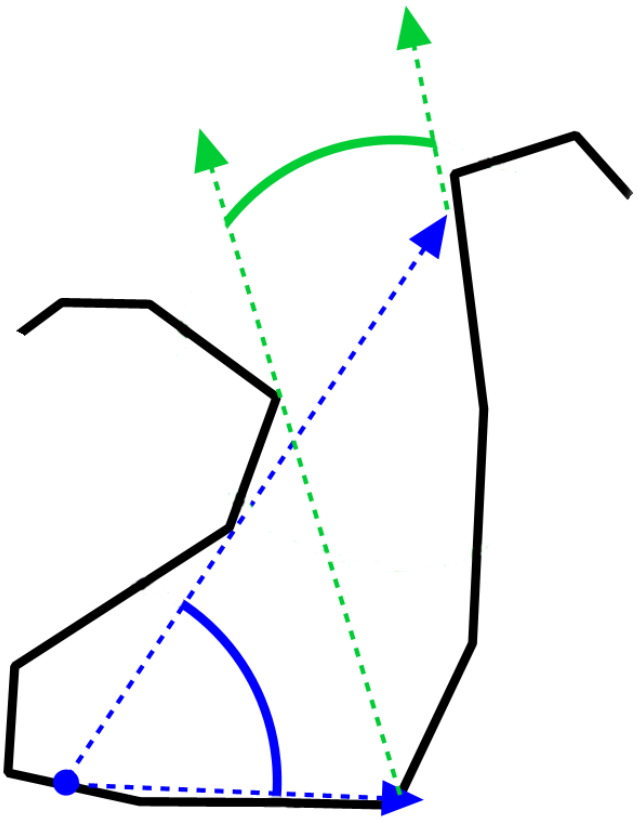

Figure 4.5: Even if the inner part of the valley (blue point) does not have direct view of the sky (blue arrows), some light will reach it, bouncing over the sides of the crease. 2^{nd} light bounce occlusion is showed with the green arrows.

Considering other bounces can produce more correct results. However, it fails to represent the color *transfer* due to reflection. For example, when a red object is close to a white wall, we can see some red color appearing on the wall, because the

light reflected from the object to the wall is red.

We are only reasoning in terms of *amount* of light received from a certain surface area. Keeping track of the color transport would require too much storage space for the kind of models we are working on.

To perform this kind of calculation is necessary to model light bounce. While the physical formulas are not so complicated, implementing such algorithm in an efficient way can be time consuming. It should also be considered that we need to apply this calculation to non-trivial models.

To obtain more precise results and better performances, instead of writing the code to perform this calculation, it is also possible to employ already existing rendering engines. Available on the net, there are several rendering engines (as commercial products, but also as free software) able to calculate global illumination effects using a given level of recursion. The advantage, beside saving the time required to implement complex rendering algorithms, is that the calculation is probably well optimized (after all, this is a rendering engine main task) and has been extensively tested.

The problem of using such instruments is how to get the results, since often the output of rendering processes are just images. A first solution can be render the information we need into an image and then re-projecting it back onto the geometry as we showed in Chapter 3 when discussing color mapping. This approach can be coupled with a strategy to select renderings able to cover the most of the object in very few renderings.

However, the best possible scenario is having access to the full programming API of the rendering engine. For some products it is also possible to obtain (beside the basic rendering program) a programming library; in those cases it is possible to write custom programs that, given the scene geometry and illumination, are able to evaluate some "value" in any point of the object. The good news about this method is that it is not limited to calculate incoming light (as we need in this case); since we have full control over the rendering engine, any kind of calculation can be implemented.

In Chapter 5 we will show how we used the Arnold rendering engine API to easily implement offline calculation of shadows and diffuse illumination.

The advent of a new generation of fast rendering engines can provide a solid base for similar calculation. Recently, NVidia released a free version of the GELATO rendering API.

Using this API it is possible the definition of custom shaders, in this way it is easy to implement whatever calculation we need. In this particular case we needed the occlusion term, we could evaluate it using as illuminator a completely white sphere and just evaluating how much light each point receive with a shader that returns as final color just the received light amount. More generally, given a lighting model, we can find out which are the invariants and write custom shaders to evaluate those quantities on the surface.

4.5 Non homogeneous lighting

Up to this point, we only mentioned diffuse lighting effects that are produced from light coming from "everywhere" in the scene. This perfectly diffuse situation can be useful when dealing with a single object, visualizing it without considering the environment for pure examination purpose. When the object is placed into a scene, the lighting should take in account the real light disposition of the scene; otherwise the object would seem even more out-of-place than without illumination at all.

Clearly, the approximated occlusion technique exposed in Par. 4.3 can easily deal with a non-uniform lighting. When the probing directions are generated, it is possible to apply the same distribution of the lights (instead of using an uniform distribution). However, even in this way we calculated a diffuse shading that is *frozen* in that particular lighting condition. This, again, can be good for some applications but in case of dynamic lighting this is unfortunately insufficient.

The best way to perform on-the-fly change of lighting is to store ambient occlusion not as a single value for each vertex (or texel), but to use spherical harmonics encoding.

4.5.1 Spherical harmonics

In mathematics, the spherical harmonics (SH) are an orthogonal set of solutions to Laplace's equation represented in a system of spherical coordinates. The spherical harmonic $Y_{lm}(\theta, \phi) - l < m < l$ is a function of the two coordinates θ, ϕ on the surface of a sphere. The spherical harmonics are orthogonal for different l and m, and they are normalized so that their integrated square over the sphere is unity. Any signal in a spherical coordinates can be encoded as a linear combination of these orthonormal basis. The mathematics behind the process of encoding and value recovering is easy, making this technique very valuable in realtime applications.

Spherical harmonic lighting is a very popular technique to calculate lighting [75, 118]; the lighting environment (seen as a sphere surrounding the scene) can easily be encoded in SH basis, as well as the local surface BRDF response. Lots of implementation details on SH calculation and usage for realtime applications can be found in [60].

Since we need to store the ambient occlusion term with respect to light distribution we can use the SH encoding. Basically, we will encode how much a certain point of the surface is occluded from each SH basis, considering the basis as the light source.

Then, given an illumination setup (that normally can be approximated by a spherical lighting environment) it is possible to encode it using the basis: thus knowing how intense is the lighting from each of those basis. At runtime, the lighting encoding will combined with the ambient encoding to recover the amount of light that the surface receives from that illumination setup.

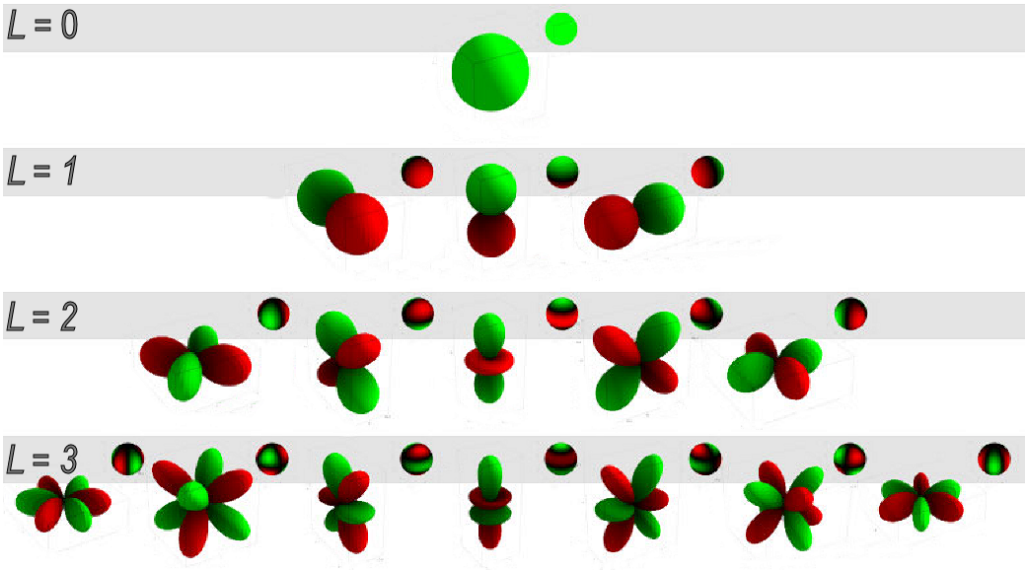

Figure 4.6: Graphical representation of the first four orders of Spherical Harmonics basis. Green is used to encode positive values, red for negative values. From Green [60]

Again, it is possible to implement the shading algorithm to perform measurement, but the most accurate way to compute spherical harmonics is using a global illumination rendering engine.

SH basis are an infinite series of functions; just like the Taylor polynomials, the more basis are used, the better is the approximation we can obtain.

In most of the publications that uses SH to compute lighting, at least 3 orders (9 basis) are used. This because they are trying to represent the whole range of lighting effects. Our case is simpler; we are just using this encoding to represent the diffuse component of the illumination (and, to be precise, just a part of it). For this reason, we reduce the number of basis used for encoding. This grants a lower memory occupation and a faster lighting process, this is especially important, given the size of 3D models we need to manipulate.

Using just 4 basis (first and second order) it is possible to obtain a good representation of the ambient occlusion term with a minimum computational effort. The results of such encoding can be seen in Figure 4.7; the two images show how the ambient occlusion term changes when changing the position of the light source. Certainly, the result is quite smooth, but it has to be considered that the nature of diffuse lighting is smooth.

In case of very large 3D models, the spherical harmonics calculation can be quite difficult; most of the available global illumination engines are implemented with the assumption to keep in memory the entire 3D model. This because having all the scene in main memory greatly simplify the implementation process.

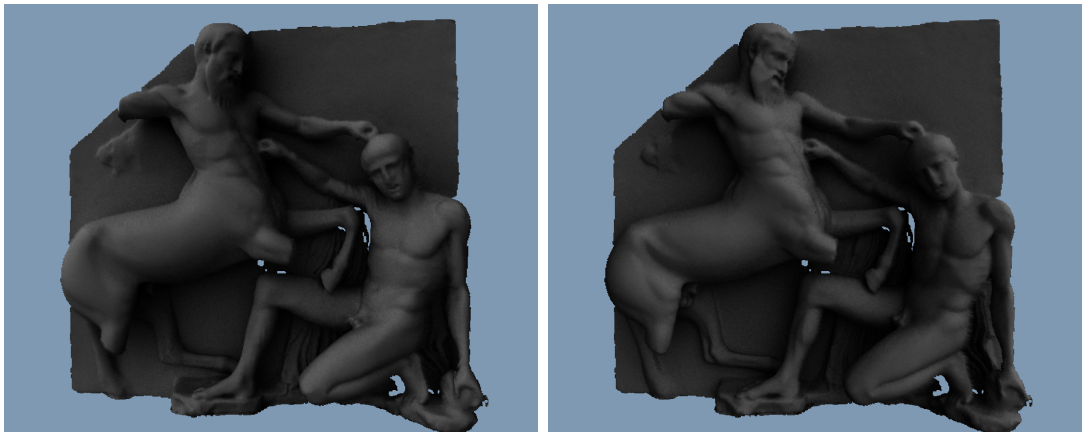

Figure 4.7: Non homogeneous Ambient Occlusion calculated and stored as Spherical Harmonics coefficients. Ambient Occlusion is stored using just 4 SH basis. Left: reconstructed occlusion term for a light coming from above-left. Right: reconstructed occlusion term for a light coming from above-right. Calculation has been done using the Arnold rendering engine [53] taking in account multiple light bounces (3 levels of recursion).

If those limitations arise, the calculation process should be split into steps. It is necessary to subdivide the 3D model in chunks and produce, for each chunk, different level of details. Then, for each chunk, it is possible to build a model that has that principal chunk at full resolution and the other chunks at lower resolution the farther they are from the principal one. Those models will hopefully be below the triangle limit of the rendering engine, but providing a good approximation when calculating occlusion. Occlusion influence from very far geometry is minimal; so, for those parts a lower resolution model does not affect quality. Occlusion is calculated for each chunk, using the appropriate model. Obviously this increase the computational time very much, but extends the usability of the method as well.

4.6 Beyond

Adding the diffuse component of lighting to a rendering engine greatly enhances the level of realism perceived. The calculation techniques outlined here give a good approximation of the occlusion value at a relatively low computational cost. They are applicable to very large 3D models and easy to implement.

Even using the techniques described here, we are still far from having a photorealistic visualization; this step of lighting is just accessory to more complex shading. Used in conjunction with the mapped color can produce good results. Even better results can be obtained with a more accurate surface representation; since many surface models have as input the amount of light received, this integration is

straightforward.

Introducing hard shadows can be another step towards the photorealism. In literature there are many techniques to calculate and display hard shadows. Probably, the most straightforward way to obtain this effects when dealing with large triangulated models is the use of *shadow maps*. This solution is quite good for static or semi-static lighting setup. However, in case of dynamic lighting, the computational cost can be very high. Even considering just a single light, when the light position changes it is necessary to update the shadow map doing another off-screen rendering; if the light motion is continuous, the frame rate drops to one half.

A partial solution, when working with multiresolution engines, can be using a lower resolution model when calculating the shadow map. Even if the shadow quality decrease, for some applications this can be a viable option. In other cases, some different techniques can be employed, as we will show on the next chapter.

Chapter 5

Realistic Realtime Rendering of complex 3D scenes

In the last chapters we have discussed methods for a more realistic rendering of 3D models. We have discussed how to efficiently map attributes on very large 3D models and how to deal with diffuse lighting for complex scenes.

We believe illumination is the main component in our perception of the world around us. Clearly, the surface characterization plays an important role, but having a realistic illumination can help in perceiving the object as real. This is even more important in the case of very large object or buildings. Design of large statues and buildings is heavily driven by their interaction with light.

There are numerous studies about our familiarity with certain illumination conditions: in our mind we perceive as "good" a lighting coming from above, slightly on the left side. This is why this lighting is used by the majority of corporate logos (like the Google one) or even by operating systems (Windows, MacOs) to shade buttons and frames of their windows managers .

Although in the past this kind of studies were not available, buildings and statues were generally built using such kind of considerations, just because it was empirically determined that this was a way to maximize their appeal. For this reason, when rendering large artifacts with an outdoor placement or buildings, a particular care should be used to represent a correct lighting. Hard shadows, while not important for small objects, have in those cases a huge impact on visualization fidelity. Since the main outdoor light source is the sun (which can be considered a directional light), it is almost impossible for a building to be completely free of shadows. When we look at a building we *expect* to see those shadows, otherwise the impression is to see a small scale model. Unfortunately, this is what often happens while looking at 3D models on the computer screen.

In the following sections, we will present the results of a project where we tried to integrate many techniques in a single application, with the aim of building a realtime demo able to compute an almost photorealistic rendering of a complex scene. The objective is to faithfully represent the interaction between the daylight

(the passing of an entire day) and a building with a detailed geometry (in our case, the Parthenon).

If there is a concept that clearly arises while working in different projects, is that there are not two projects alike. Certainly, all 3D models share the same structure and can be manipulated using the same algorithms. The project presented in the following paragraphs is a clear example of how a project can be unique; everything in this project, ranging from the available data to the visualization system, imposed some constraints and forced us to find ad hoc solutions.

In the following description, we will carefully explain each of these constrains, presenting (and motivating) our choices and, were possible, by also proposing some possible alternatives.

5.1 Project Background

In 2002, the ICT Graphic Lab of the University of Southern California started a project aimed at performing the 3D acquisition of the Parthenon building and of all its carved decorations (scattered between different museums). The aim was to obtain a complete virtual representation of the building in its current shape, but also reconstruct the original appearance [137]. A collaboration between the Visual Computing Lab (VCLab) of ISTI-CNR and the ICT Graphic Lab started in the early phase of the project, since ICT chose the VCLab tools [20] to process the raw scan data. The most spectacular outcome of the Parthenon Project has been for sure the short movie presented at the Electronic Theatre at Siggraph 2004.

The objective of this work is to reproduce in realtime a crucial sequence of the short movie: the **time lapse** sequence which shows the Parthenon during the passing of a whole day, from dawn till dusk. This sequence shows the interaction between the changing sun and sky against the building geometry, originating complex lighting effects and bringing out the surface details. Lighting plays an essential part of this sequence; replicate the accuracy of the lighting process will be the main challenge.

Realtime and off-line rendering have always been considered two separate worlds, since it is usually very hard to mix the astonishing level of realism obtained in movies with the user control available in videogames or other visualization tools. In the last few years, however, many examples of offline-to-realtime conversion have been presented, mostly due to improvements in video card technology. Famous computer-generated short movies (like the Pixar's "LUXO Jr.") have been transformed in realtime demos by video card manufacturers to demonstrate the effectiveness of their products.

The aim of the demo we have designed is to convey the same level of realism of the Sequence computed offline, reproducing the correct interaction between the light and the building and thus giving to the viewer a feeling of presence. Some peculiar characteristics of the work can be summarized as follows:

Complex Shading: the idea behind this work is not to find tricks that can produce "plausible" results, but to find an approach which allows to compute a realistic lighting even in a realtime environment. We will use a mix of preprocessing and hardware accelerated rendering to evaluate the lighting function.

HDR: High Dynamic Range calculation is considered essential to convey a sufficient level of realism, especially when we are interested in outdoor scenes. Sun intensity can be over five orders of magnitude brighter than the sky and clouds, giving a dynamic range very difficult to manage. Working on color with only 8-bit-per-channel would result in excessive loss of visual details. The new programmable graphic hardware can be used to calculate the whole lighting step using floating point values; the result will be then exposed for visualization.

Immersive Stereo: running the demo in an immersive environment will greatly enhance the experience. The VR platform used allows both to manage interaction in a very natural way, reacting to user movements and supporting a very large displaying surface (by retroprojection), rendering the scene at a *scale* much closer to reality. A stereo visualization system will help in conveying the sense of greatness given by this monument.

The use of virtual reality to better present cultural heritage artifacts and environment is quite an old trend [40, 121] and cultural heritage applications have been the standard demos for many virtual theaters or CAVE-like systems.

More recently, the need for improved realism in this kind of VR applications has been addressed by many research projects which focused on improved geometric models and more sophisticated use of textures [47, 145]. But using good geometry and textures is not sufficient to convey a sense of presence, adding a good modeling of the illumination is another basic ingredient [123]. How to add more advanced illumination models in interactive VR environments is still an open research topic.

A requirement of the project was that the interactive visualization should have been a peculiar system, based on retroprojected walls, which simulates a room with openings (virtual windows) over the external world. The need to develop the application for this specific environment introduced some constraints, discussed in the following sections.

A design constraint for the interactive demo was that it should be implemented in the framework of a specific virtual reality environment, called FlatWorld [106, 105]. In film and theatrical productions, sets are constructed using modular components called *flats*. FlatWorld utilizes a *digital flat* system; a digital flat is basically a large retro-projective display. The system is oriented to training, with the idea of providing a reconfigurable environment able to represent areas much larger than the building the system is built in. The current setup is a room with two active walls: the room can be decorated as needed and the user is free to move inside this room,

looking at an outside world displayed on these walls. The system employs polarized lenses for projectors and glasses to display stereo images, position of the user head and of other objects is tracked using an infrared systems and the sound is managed by a positional audio system.

5.2 Available ingredients

The interactive demo has been designed by using the 3D data gathered during the Parthenon project. Working on the same data used to produce the Siggraph movie was a wonderful opportunity to focus on the rendering techniques, since this data has proven to be accurate and complete, as shown in the Siggraph movie.

Obviously, the core of the interactive demo is the 3D model of the Parthenon. This model has been acquired using a *time of flight* scanner and then processed using the standard software pipeline. However, as almost all the 3D scanned models, the resulting mesh was uncomplete in many areas, especially in the parts visible from inside and above. This would cause problems during the movie rendering, because rays could pass through the object. To resolve this, the model was completed with a "filler" geometry, modeled with standard tools.

Another important data for a realistic rendering is the surface characterization, since knowledge of the reflection properties of each part of the building makes it possible to compute a physically plausible illumination. For the Parthenon dataset, an accurate description of reflection characteristics was available. First of all, the general BRDF of the building surface has been measured on the field. Capturing the exact spatially varying BRDF for the entire surface of a such large object it's impossible. However, since the surface of the building is more or less homogeneous, what has been done was to evaluate a general BRDF model, valid for the entire building, and then encode for each area of the surface the local variations as parameters of that general BRDF. The local parameters have been recovered for the two short sides, using an inverse rendering method [43]. Unfortunately, for the other two sides of the Parthenon it was not possible to obtain a good photographic coverage to perform the inverse rendering; one side was occluded by the scaffolding installed for restoration purposes and the other one is too close to the cliff side. Among the recovered local parameters, the most important value for our work is the surface albedo.

However, to obtain a realistic illumination of the building during the entire day sequence, it is necessary to have a faithful representation of the light sources. In this outdoor scene, the two major sources of light are the sun and the rest of the skydome. A day-long sky dataset has been acquired using HDR imaging techniques [44, 136], where a High Dynamic Range skydome image is available for every minute of the day. Moreover, for each image, the sun position and intensity have been detected, producing a concise directional light representation. This dataset has been used in

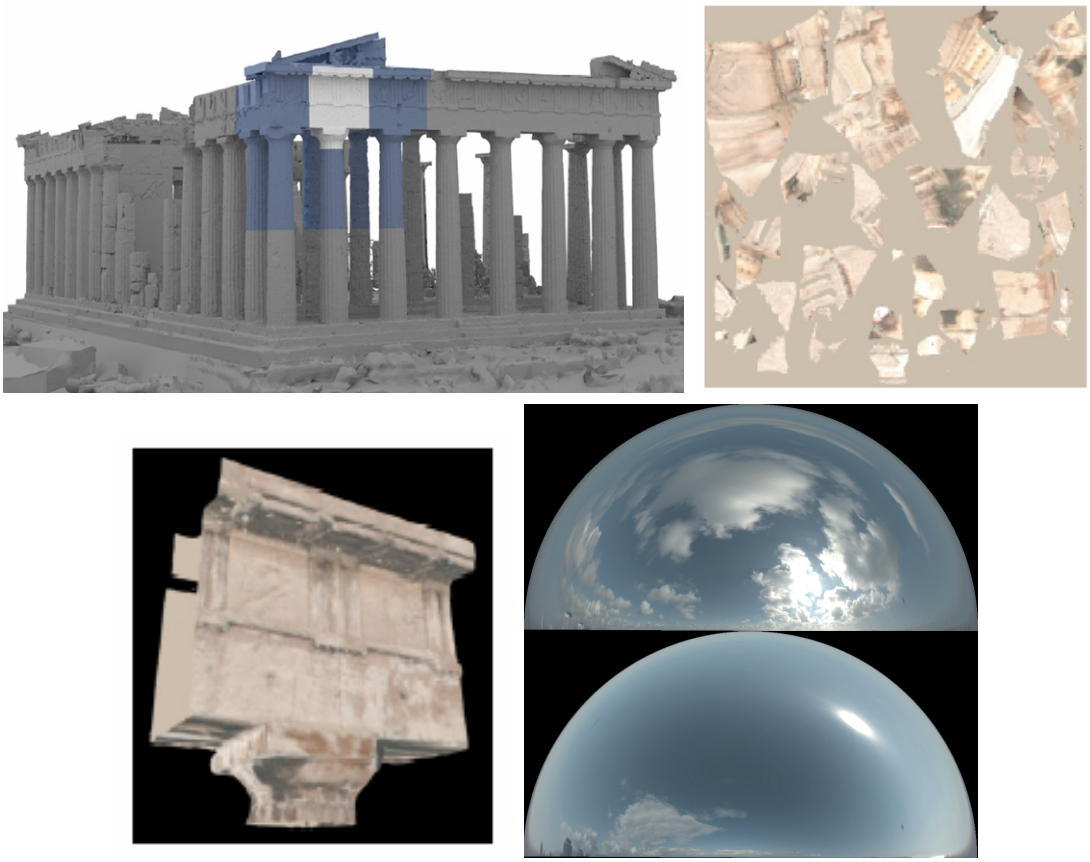

Figure 5.1: Available data. The 3D model of the Parthenon building, acquired with TOF scanning and divided in multiple chunks (one of these chunks is rendered in white); the texture atlas used for the highlighted chunk, which encodes the recovered surface albedo, and the texture-mapped chunk; finally, HDR images of the sky dome.

the movie rendering to illuminate the scene, where the sun is the major light source but also the rest of the skydome has been taken into account.

To render the Parthenon movie with a high level of realism, a physically accurate rendering engine is needed. The Arnold rendering engine [53] was used to render The Parthenon movie. It implements a Monte Carlo-based global illumination algorithm able to produce images with a very high realism. Moreover, coming as a library, it is completely configurable and it is possible to add plugins to manage custom data or to write shaders which implement new surface behaviors.

Obviously, it was not possible to directly use that very rendering engine in a realtime system because of the high computational cost. To give a reference, one hour has been necessary to compute each frame when rendering the movie. However, since we had access to the basic API of the rendering engine, it was possible, as explained in the last chapter (see Paragraph 4.4) to use this rendering engine to

calculate *invariant parts* of the shading equation and then complete the calculation using hardware shaders in the realtime demo.

To implement the demo in the Flatworld environment, we had access to the software codebase used to build applications for that system. Beside its hardware setup, the core of the FlatWorld system is an application framework able to manage the various elements of the immersive system: head tracking system, sound, lighting and so on. This framework is based on a commercial game engine called GameBryo [89].

5.3 Design of the Rendering Method

Lighting is the core component of this sequence; the passing of the day is perceived trough the illumination effects that are visible onto the Parthenon. While looking at a single frame of the animation, it is quite natural to be able to determine the time of the day, just looking at shadows length, light intensity and light color dominance. Ideally, the shading model we need should be able to provide those three aspects (shadows, light intensity, light color).

Our interactive demo should be able to represent both **direct** and **indirect** illumination: the main effect of direct illumination is the production of hard shadows and base lighting, while indirect illumination smoothes up the effect introducing additional light bounces and taking into account light coming from all the skydome.

Considering the different nature of the two components, we decided to employ two different algorithms for the computation: one for generating precise shadows, the other for producing diffuse effects. The mostly static nature of the scene suggested that the best approach was to *precalculate* as much as possible the invariants in the illumination equations and just complete the calculation at runtime when all data was available. *Hardware shaders* seemed to be the perfect choice for this kind of task since it is possible to implement efficiently the lighting calculation. Moreover, the processing power of GPUs will steadily increase in the immediate future (faster than CPU growth), providing a better frame rate and the possibility to add more complex effects. However, to maximize the benefit of using programmable graphic hardware, it is necessary to keep the computation as *local* as possible. Hardware shaders are basically short programs that are executed on each component (vertex, fragment) of the stream in the rendering pipeline. Storing all data necessary for the computation inside stream components will result in less rendering pass and thus, better performances.

Offline precalculation of shadows is also a good way to limit the geometry to be drawn each frame: using shadow maps or shadow volumes would require all the occluding geometry to be drawn (even the non directly visible parts); obviously, a subsampled version of the occluders can be used but this would affect shadow precision. Conversely, by precomputing shadows we use the whole dataset for shadow

calculation, but only a small part of the geometry will be rendered and shaded by the realtime application. The same consideration holds also for indirect illumination, since light bouncing between geometric elements is determined using the whole dataset, but only the data related to the realtime model will be stored and processed at runtime.

The direct lighting algorithm should firstly discriminate between *shadowed* and *lit* areas, this can be done with a variation of the **interval mapping** technique (described in the *GPU Gems* book [54]). Dynamic shadows are normally a difficult task, but in this case the only direct light source in the scene is the sun that is a *directional* light, easier to manage with respect to a local point source; moreover, its location in every minute of the day is well known. It is therefore possible to regularly sample the daytime and to calculate the shadows for each of this sun positions we sampled.

There is no need for storing all the calculated shadows in the scene, since the only rendered shadows will be the ones casted onto the visible geometry. The idea is to store the shadows information inside the renderable geometry. Each vertex will contain enough information to know, for each time step in the day, if in that particular moment is receiving direct sun illumination or, conversely, it is in shadows. Since for each time step the geometry will either be illuminated or shadowed, we need to store in each vertex a bit mask.

At runtime, the bit mask in each vertex will be tested to know its state with respect to the current time and the vertex will be illuminated accordingly. Lit areas will be illuminated using a Lambertian illumination calculation, which helps simplifying the computation without affecting too much the final visual quality, since the measured Parthenon BRDF proved to be quite close to a Lambertian surface.

The diffuse calculation is based on **spherical harmonics**. We already discussed advantages of using spherical harmonics in Chapter 4. In this case the ability of SH basis to encode a signal over a sphere which suits our case since the signal to be encoded is the sky image spherical probe. The preprocessing step is to calculate for each part of the geometry the lighting contribution from a particular harmonic base; similarly, the lighting source (each sky probe) is encoded using the harmonic basis.

To compute the lighting at realtime, for each vertex of the visible geometry, the influence coefficients are multiplied by the sky probe encoding for that particular time position and added up. The algorithm used in our interactive system is a standard implementation of the SH lighting described in many papers (like [60]).

The two light contributions (direct, diffuse) are added up obtaining the amount of light reflected by the surface. This value is then modulated by the surface albedo to obtain the color value of that surface point. Exposure and gamma correction are applied at the end of the lighting process.

To obtain good execution performance of this lighting process, it is necessary to move all computation to hardware shaders. The shading algorithm has been first implemented using **OpenGL Shading Language**, in order to test the algorithm

in a sample application, then it has been converted to **HLSL** to be used in the GameBryo engine. The conversion is quite straightforward and it involves almost only changing names of intrinsic functions and data types.

The described lighting does not include specular highlights, making it view-position invariant. This simplifies the calculation a bit, but it is also another approximation of the surface model. The real Parthenon surface *does* have a bit of specular behavior (when looking at almost glazing angle), but this effect can be clearly seen only when the viewer is quite close to the surface and with a local light source. We believe at this distance and with so much diffuse light coming from the skydome this kind of phenomena can be neglected. In any case, adding also this component would not be technically difficult, but at a cost of decreased performances. It would require passing to the shader also the view position and adding to the geometry some more parameters to control the angle and the amount of reflections.

5.4 Hardware Implementation of the Shading Process

All the lighting calculations are based on vertices and this choice is justified by the following arguments. The model has a very high resolution, so lighting calculated just on the vertices produces high quality result; then, given the very complicated model topology, it is very hard to produce a good (and compact) texture parametrization. In our case the geometry is dense and the usual view specifications makes each triangle project on 3-4 pixels area on the FlatWorld display. The use of a less detailed model, would require moving the computation from the vertex to the pixel/fragment, therefore requiring to encode all the needed parameters on a texture map. it is important to say that, beside the parametrization problem, there are no major problems in implementing the same light computation using texture maps to store parameters.

As stated before, the lighting process is divided in two steps; direct lighting and diffuse lighting are calculated separately, then their contribution is added up.

$$L = L_{direct} + L_{indirect} \tag{5.1}$$

5.4.1 Direct Lighting - vertex shader

As introduced in the previous section, sun direction and intensity are known at each time of the day and can be passed to the shader as a frame constant; this is enough

to compute the lambertian lighting for each point. To compute *direct lighting* on a surface point it is then only necessary to know if that vertex receives light or does not in the specific time. Therefore, together with each vertex of the object we store a bit mask containing a bit for each time step in the day, representing the lighting condition (1 for light or 0 for shadow) of that particular vertex in that particular time. Direct lighting is therefore computed as:

$L_{direct} = Lambertian$ if mask[current_time] == true
$L_{direct} = 0$ if mask[current_time] == false

When calculating the direct lighting, the status of the vertex is determined by accessing the shadow mask ant testing the appropriate bit.

The binary mask has been implemented using floating values, since floating points variables are native in graphics hardware while integers (normally used as bitmask) are just emulated and their behavior is not guarantee to be uniform from a video card to another. On videocards, float are implemented following the IEEE standard: each float has then a 23 bit mantissa that can be used to store 23 binary samples. More samples could be stored in a single float (for example, using the float sign) but that would cause problems in the hardware shader since it would be necessary to discriminate particular cases.

Just 23 samples are too few for our aim; we have to use more than one float. On video cards there are certain structures that can rely on strong hardware optimizations; one of those structures is the 4 float vectors, used to represent 3D points (XYZ plus W) or RGBA colors. We will use one 4 float to build our mask, obtaining a total of 92 samples.

According to our specific sky dataset, the sun begins to be visible at 6:52 and it remains visible until 17:44. Using the 92 samples encodable in the mask it is possible to cover the time from 6:57 to 17:34 having 7 minutes lapse between each sample. This interval between subsequential samples is acceptable, it will be possible to follow the shadows moving from one sample from the next one, but still the samples will be close enough to avoid too harsh discontinuities.

Each vertex has 4 floating point values containing the shadow mask. Since each float contains 23 samples the bit we need can be found as bit `N1` of float `N2` with:

```
N1 = CurrentTimeIndex mod 23
N2 = CurrentTimeIndex div 23
```

To avoid calculating those values for every vertex the two indexes are computed in the program and passed down as constants. Binary operations are not implemented into shaders, therefore the old assembly trick of “divide and check the rest” is used to extract the bit value:

```
(mask[N2] / 2*N1) mod 2
```

returns 1 if the bit is set, 0 otherwise. After this step the light value for the vertex is set to 0 (if in shadow) or to the standard Lambertian lighting value (if in light) calculated as:

```
DirectL = LightIntensity * max(dot(VertexNormal,LightDir), 0.0f)
```

5.4.2 Diffuse Lighting - vertex shader

For *indirect illumination*, the light contribution in each vertex is the sum of each harmonic of the influence coefficient (how much the vertex is affected by that harmonic) multiplied by the current sky encoding (how intense the sky is on that harmonic).

$$L_{indirect} = \sum_{i=1}^{4} Coeff_i * SkySH_i \tag{5.2}$$

Given the direct lighting contribution, we have now to add the diffuse component computed using the spherical harmonics. Each vertex has 4 floats which represent the amount of influence the spherical harmonic base has on that vertex; this is used as a multiplication factor for the sky probes encoded as spherical harmonics.

The lighting contribution of each harmonic to a particular vertex is obtained by multiplying the influence factor by the spherical harmonics encoding of the sky (basically, an HDR RGB value) and summing it up to the direct lighting previously calculated:

```
Directlight.r += vert_sh[0]  *  sky_sh_encode[0].r;
Directlight.g += vert_sh[0]  *  sky_sh_encode[0].g;
Directlight.b += vert_sh[0]  *  sky_sh_encode[0].b;
. . . . . [omissis harmonics 1-2] . . . . . .
Directlight.r += vert_sh[3]  *  sky_sh_encode[3].r;
Directlight.g += vert_sh[3]  *  sky_sh_encode[3].g;
Directlight.b += vert_sh[3]  *  sky_sh_encode[3].b;
```

Only the first four harmonics are used into our interactive system; in other publications nine harmonics are used to obtain a more precise representation of the high frequency shading variations. However, as also discussed in the previous chapter, we are using this kind of technique to calculate just *half* of the lighting; at this point the high frequency direct illumination is already calculated and we just need to introduce low frequency indirect lighting.

Final calculations - fragment shader At the end of this process we have the final light value incoming in that vertex. This value is then passed down toward the fragment shader. In this way the lighting values that are calculated on the vertices are interpolated across the faces of the model. Interpolation produces a smooth and plausible effect, under the hypothesis that the base geometry is detailed enough and that the user will never get too close to the object.

In the fragment shader the light value is multiplied by the surface albedo, producing the final color of the fragment. Up to this point all the computation has been performed using floating point values, it is now time to get from HDR to values in the [0..1] interval for the final rendering. The color is multiplied by the exposure level (calculated as 2^{stops} and passed to the shader by the application) and clamped in the [0..1] interval. Gamma correction is then applied to match the monitor (or videoprojector) response.

Figure 5.2: Examples of real-time shading: note how the shadows moves precisely over the geometry (top-let and bottom-left images) and how the HDR lighting calculation allows to adjust the exposure to better perceive details (images on the right) which are under shadows in the images on the left.

Clearly, considering the amount of computation required by the vertex an fragment shaders, it is possible to see a predominance of the vertex shader. The first reason of this choice was to "leave space" to the computation of HDR effects like *bloom*. Those kind of effects are computationally heavy and can only be done in the fragment processor.

5.5 Offline Parameters Calculation

As stated before, the main idea behind the interactive system is to precalculate all the lighting invariants: all the realtime shading computation will be based on the different parameters that have been computed offline. These parameters can be divided in two groups: *vertex attributes* and *lapse attributes*.

Vertex attributes are stored in each vertex of the 3D model and they are used during computation, they appear as variables in the vertex (and fragment) programs since they are model-dependant. Lapse attributes represent the light condition in a particular time of the day and they are generated from the skydome probes. In the shaders they appear as constants because they does not change during a single frame rendering.

To precalculate the invariants of lighting equation, it is necessary to have an implementation of the lighting process that can be used to compute lighting "up to" a specific point and then retrieve the partial result. To obtain this, we chose to use an already existing rendering engine, as explained in Paragraph 4.4. The Arnold rendering engine is the tool used to produce "The Parthenon" movie. Since its basic API was available, to render the offline movie have been implemented custom importers and shaders to manage the data format used for the Parthenon model, the material model and the lighting environment.

Beside its accuracy, another important feature of the Arnold engine is that, using API, it can be used to probe the shading of a particular point on the scene geometry. Using the Arnold library is therefore possible to build a program that, given the scene as input, computes for each vertex of the geometry the various values we are interested in. The use of the Arnold engine to precalculate the lighting values is not only justified by the code reuse policy we already mentioned, but it also guarantees that the shading results will be coherent with respect to the ones presented in the offline movie.

5.5.1 Direct Lighting Parameters

Two different kind of values are required to compute the direct light contribution: the shadow mask and position and intensity of the sun. Shadow mask is a vertex attribute, sun position and intensity are lapse attributes

Shadow mask: the mask represents the lighting state of the vertex during the day. To calculate these values (for each vertex and for each sun position), the Arnold rendering engine has been used in "probing" mode. The scene with all the geometry has been initialized as doing a normal rendering, then for each sun position we calculated the irradiance of each vertex using a single ray with no light bouncing. Vertices with an irradiance equals to 0 are in shadow. As an alternative to the Arnold engine, any ray tracing implementation would

Figure 5.3: Shadows precalculation: for each sun position, the scene is rendered in the Arnold Engine. In this case the engine works like a ray tracer, to determine if each single vertex is receiving light in that time of the day.

work fine since in this phase it is only important to know if the vertex can see the light source or not.

Sun position and intensity: this is the most intuitive data and comes directly from the processing of the HDR sky probes. In each skydome image, the sun position and intensity have been determined analyzing the brightest pixels. To have more details on this process, refer to the HDR sky acquisition paper [136]. For each time position there are 3 floats to encode the normalized direction [XYZ], and 3 floats for the high dynamic range color [RGB].

5.5.2 Diffuse Lighting Parameters

Again, two different values are required to compute the indirect light contribution, which are the spherical harmonics vertex response (vertex attributes) and skydome spherical harmonics encoding (lapse attributes), in particular:

Spherical harmonics response: spherical harmonic basis describes a signal over a sphere; each basis is a spherical intensity field. Using this spherical basis representation as a skydome light source to illuminate an object, it is possible to measure how much each vertex is affected by that lighting environment (i.e. from that particular base). This value, a float for each basis, will be used to modulate the spherical harmonic encoding of the real skydome during realtime shading. Again, the Arnold engine was used in probing mode, rendering a scene with no direct light sources and using the spherical harmonics as skydome. This time, light scattering has been included in the computation, using 4 levels of recursion. For each vertex in the model, 4 signed floats are generated.

Skydome spherical harmonics (SH) encoding: this value is a representation of the sky using the first four SH basis. In theory it is the sphere integral of the multiplication of the sky by the spherical harmonic. Practically it is

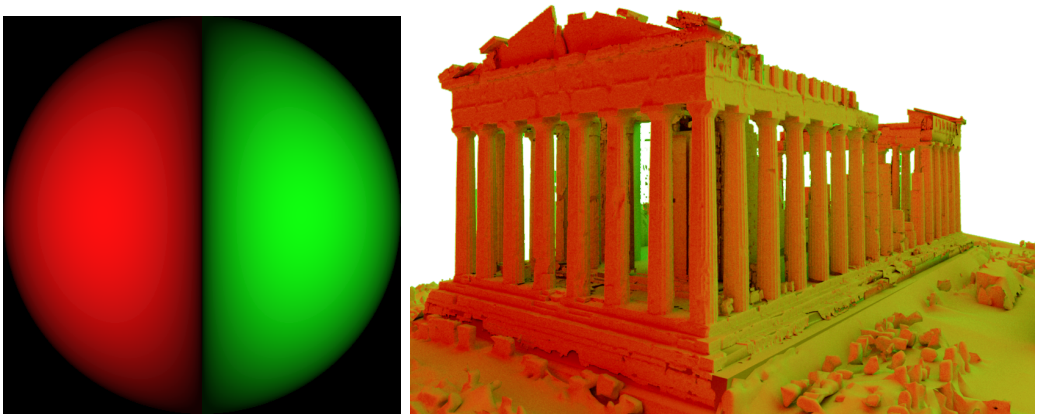

Figure 5.4: Diffuse lighting precalculation. On the left, the Spherical Harmonic base used for lighting with positive and negative values encoded in red and green. On the right, by illuminating the scene using the harmonic as a skydome light source it is possible to see how much each part of the object is influenced by this harmonic.

computed as the weighted sum for all pixels of the multiplication of the sky probe image by the spherical harmonic image. For each harmonic basis there are 3 floats, basically a signed HDR color value.

5.6 Scene Setup

The original Parthenon model was too big to be used directly into a real time application: even at a medium resolution the polygon number exceed 10 million triangles. A smaller model was required to grant a realtime frame rate. The geometric description of the Acropolis used for rendering the movie is composed by different elements:

- the *ground*, which is obtained by 3D scanning, it is stored as a single file and it is textured.
- the *temple*, which is obtained by 3D scanning, it is divided in various chunks. This division corresponds to a regular voxel-based space subdivision and was necessary during model generation and the rendering video to better manage the geometry complexity. Only the front and rear facades have texture information.
- the *filling geometry*, that is modeled with a CAD system to fill gaps in the scanned data and it is encoded to a single file.

Since the image resolution that is available in the Flatworld system is 1024x768 for each wall, an over-detailed geometry would be useless. Moreover, the Flatworld goal is to simulate a *window* over a real environment trough the walls of the visualization room. The room has to be statically placed into the scene and the user will look out from one *virtual* window, having the possibility to move inside the room.

5.6.1 Viewing parameters

The Flatworld system has been designed for military simulation and training applications, for example, for allowing a soldier to observe the external environment (buildings, other actors) from inside a building. This restricts the set of possible view position and directions that the tracked user can generate by moving in the interior space. Therefore, it is not necessary to have a model which looks good from all possible viewpoints, like in a standard browsing/manipulation framework, but just a model that looks good from the allowed viewpoints.

Our first task was then to choose a good position for the virtual Flatworld room (on top of the Acropolis). The choice of the viewpoint and view parameters is, in this case, very important. When observing a 3D model using a computer screen, sometimes the perception of the object scale is completely lost. E.g. a vase we are observing can be a small perfume container, or a very large ceremonial urn. A correct perception of the scale of an object can rely on many lateral-information, like:

Environment: to inspect a single object, often we render it alone, floating in an empty space. This does not help to perceive its size. Conversely, the presence of an environment and other familiar objects can help determine the size of an object by comparison.

Material scale: most of the real world materials have a *grain* that has a well defined size. The size of things like veins in a piece of wood, cracks on stucco and threads in a fabric, when compared to the object size can give us an idea of the scale of the object.

Field of view: we are used to perceive the world with a certain field of view, that is more or less 70; just through perspective we can have an idea of the size of an object. On computer graphics application we often use very large field of view, to be able to see more in a single rendering; this cause lots of perspective deformation, making object appear bigger than they are Conversely, using an almost orthographic view, make things smaller.

Stereoscopic Vision: using the different viewpoint of our two eyes, we are able to determine the depth of the scene we are looking at. Normal monitors cannot present the same stimulus. However, there are many different techniques available to provide stereo visualization of 3D scenes.

Lighting: given our experience in real world, we can judge the size of an object by looking at its interaction with light; our brain process informations like the presence of a distant or of a local lighting, light intensity, hardness of shadows. Also this ability depends on our experience of the real world. Moreover, even some complex material behaviors depends on object size.

In this particular project, we have to show an extremely large building, and our goal is to be able to convey its scale to the viewer. To reach this goal we will work on all the perceptive hints above mentioned; we already discussed how we will try to represent as precisely as possible the surface behavior and the lighting. At the same manner, also the choice of position and view parameters will be focused to reach this objective.

The visualization will use a stereoscopic system, this will give the correct sense of distance from the object; moreover, thanks to the head tracking, we will be able to a more natural perception of parallax. The screen size will also help, a 3×2 meters wall used as a screen will fill the field of view of the user. Finally, we will use the correct perspective angle when drawing the . All those precautions will ensure that the wall-screen will be perceived as an aperture over a space with the correct (real) proportions.

The position should be selected such that it is possible to see from the virtual window the whole Parthenon at an adequate distance and from a sufficiently low position. This will give a view quite similar to the one possible in reality. Being in a lower position will also give to the building a sense of greatness, as it is in reality. For sure this aspect has been taken in consideration also by builders; the building sits on top of the acropolis, a visitor will see it from a lower position.

Part of the sky and of the ground should have been also visible; in this way it will be possible to see the the changing of the day (in terms of sun and clouds movement) and the shadows (that will move over the ground, like a sundial). This will also help in giving a better idea of proportions.

We have chosen a position on the west side, offset towards north, looking more or less at the same area covered by the movie. This choice was also influenced by the data availability; in fact, this is also the area with better 3D scanning coverage.

5.6.2 3D model processing

The Parthenon model has then been divided in different areas, having different resolutions according to the view distance from the room position. This was done to have high detailed geometry in front of the camera and less detailed triangulation in the parts further away.

The filling geometry, that had been modeled using Maya to close the holes of the scanned geometry, required also some modifications. This added geometry was

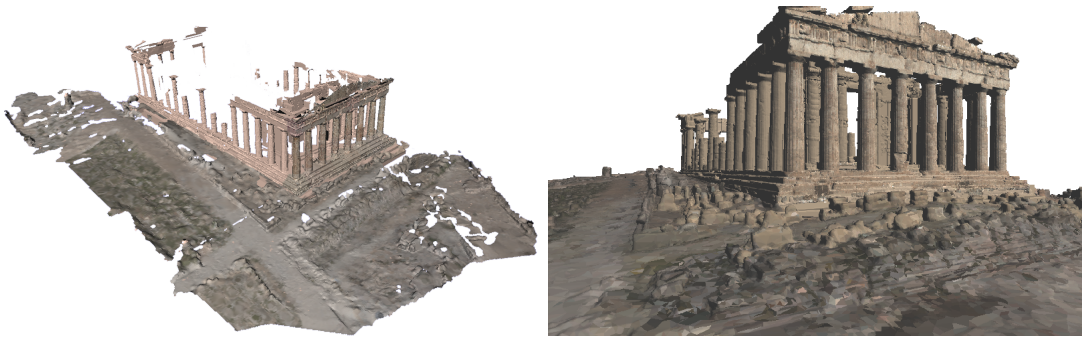

Figure 5.5: The 3D model after reduction and culling; even if the model is not complete (left) it contains all the elements visible from the FlatWorld environment (right).

good for offline rendering, but problematic for our shading algorithms because it is composed of very large triangles and self-intersecting parts. For this reason, the geometry has been recursively splitted into smaller triangles to have a more uniform triangulation and the redundant and self-intersecting parts have been eliminated. These processing steps were needed due to the characteristics of the 3D model. Scanned models are very accurate, but at the same time they are usually not topologically clean and highly incomplete, requiring intense processing.

At this point we had a simplified model, however this model still had too much geometry (2 Million triangles) to be suitable for our realtime application. By exploiting the characteristics of our constrained virtual window system (the model will be viewed from a virtual window and with a freedom of movements of a few cubic meters) we further reduced the size of the geometry by purging those model faces which will always be back-facing or occluded from all possible points of view inside the Flatworld space. A ray tracing algorithm has been used to detect those back-facing or occluded faces. The size of the model was reduced to 700,000 triangles, a number low enough to be rendered in a real time context but still enough to convey a very precise geometry when displayed on the Flatworld screen.

We adopted this static LOD approach because of the limited viewpoint placement, but also because the specific characteristics of the virtual reality environment, Flatworld, did not allow us to use more efficient and sophisticated multiresolution approaches [32]. Game engines and many other scene rendering engines are normally developed to manage lots of single *entities*, each one described by a geometry with associated materials and shaders and maybe animations. The engine is then able to perform techniques to reduce work load like frustum culling, visibility culling, and LOD switching (for entities with multiple LOD geometry). This often collides with the data structures and algorithms necessary to implement more complex strategies.

A trade-off solution would have been dividing the Parthenon geometry in chunks (or using the available division) and provide different levels of detail for each chunk. Then, inside the GameBryo engine, use each chunk to build a single entity (each

one with the same shader, to have the same behavior). This would mean having the game engine itself managing the visibility and load balance issues. This can be a viable option for low-to medium geometric load; modern engines can easily manage scenes composed by even one or two million triangles. However, for very large scene it will be impossible to directly use game engines without structural modifications.

5.7 Interactive system implementation

Building a single task interactive application, with the possibility to use every bit of the host machine and without coding restrictions, can produce very good results, but the code implemented will not be usable in conjunction with other tasks or inside more structured programs.

In our case, part of the initial specification was the need to integrate the demo in the FlatWorld framework. The idea behind this, was to implement the demo as a sort of architectural "stage" such that, afterwards, more elements (people, animals, special effects) could be added. In this perspective, the game engine structure described in the last paragraph seems a good solution. Since each entity in the game is a closed structure, describing everything necessary to be drawn (in our case, geometry and shaders), this data confinement helps in keeping a "clean" program environment. The presence of other entities, with different drawing effects will not interfere, and the engine will take care of resource allocation and state management of the graphic pipeline. So, other components can be added later by introducing new game entities.

An example of this kind of process is the grass in front of the Parthenon. In reality, in front of the building there is only naked stone, but the space appeared very empty during testing, so we decided to fill it with some waving grass. The grass is a game entity, with a very simple geometry (billboard-based) animated trough a shader. The same lighting constants used for geometry are also used by the grass shader to provide a lighting coherent with the rest of the scene.

The skeleton of the demo program was already available: a simple application able to render a scene graph and to accept input from keyboards, game controllers and the tracking system. The real problem was to integrate all the data and processing requested in one of the GameBryo engine entities.

To manage the complexity of the geometry, the game entity has multiple geometric children, since it is impossible to have such a large object in a single GameBryo geometry node. This because each geometric node is converted in an high performance indexed buffer on the video card memory and, due to hardware limitations, the size of those buffers is limited to 20k triangles.

Shader data is stored in the video memory, in the same node objects where geometry is stored. The description of a vertex is composed by multiple fields, beside geometric data, we extended the standard fields to accommodate our vertex

parameters. All data is loaded at startup and transferred to the video card. This way no further updates are necessary at runtime, all animation is done just changing the shader parameters.

The sky behind the Parthenon has been implemented as another game entity; the geometry is a rough sphere textured using the sky image dataset. The texture is updated as the time changes and the associated shader is used to control exposure. Even if 92 samples are acceptable to show the shadows moving on the objects, for the sky this is quite a poor time resolution since clouds move much faster than the sun. For this reason, in the FlatWorld demo we used 184 sky maps (two for each time position). This means that during time flow there are two kinds of transition: in odd transition only the sky changes, in even transition the sky and the shadow change.

Unfortunately, the GameBryo engine does not supports HDR textures, it was then necessary to use low dynamic range images, each one exposed to match the light value of the current time position. Extending the engine to support this kind of data may help in having a more correct sky rendering. However, we could not implement this support, due to the short time we had to developed this demo.

Beside pure rendering, the main activity inside the application is to change the time-dependant data, updating the shaders with the new frame constants, this is done by an object called TimeLapseNavigator. The lapse navigation contains the lapse attributes (sun intensity and position, SH sky encoding, exposures) for each time position. When the time position is changed (automatically or by user command) the shader frame constants are updated using the corresponding data.

Interaction with the demo is quite simple: the viewpoint is initially set to the center of the virtual room and can be modified using a keyboard, a game controller or using the tracking system.

The demo starts in automatic time flow mode, the time advances automatically throughout the day lapse, completing a cycle in more or less 30 seconds and then starting again (the speed can be specified in the configuration file). This behavior can be interrupted (and then resumed), stopping the demo in a particular time position that can be moved forward and backward with just a gesture (using a floating wireless mouse).

Exposure is automatically adjusted to a "good" level in the automatic mode but can be tuned up or down with another gesture.

Despite the complex lighting calculation, the current system implementation resulted quite fast. Beside the normal calculation load of a 3D visualization program, in our case also the stereo projection management and the head tracking influenced the performances. Nevertheless the application was able to run at 12 frames per second (considering the stereo, at 24 fps).

The host machine is a Pentium4 at 3GHz with a 1GB of ram; the used video card is a GeForce FX6800 Ultra with 256MB of video-ram. The most important part of the computer is obviously the video card, since most of the calculation is done by the hardware shaders. Moreover, the ram of the video card is where the

Figure 5.6: Screenshots of the Realtime Demo. Time flows from dawn to dusk, shadows moves across the building and the overall hue of the scene changes according to the sky illumination. The shots are taken from different positions, each time nearer to the building.

scene geometry is stored for efficiency.

The memory footprint of the demo is only 200 MB and the CPU is not completely saturated by the application. There are enough spare resources to be able to add more elements to the scene (such as dynamic actors) as required in the project specifications.

Regarding the resulting realism, even if the lighting calculation implemented in the demo is just an approximation of the one used originally for the movie rendering, many of the lighting effects that appeared on the offline movie are still visible. Shadows are precisely represented, the sky color (yellowish in the morning and reddish in the evening) influences the overall hue of the building very realistically. Finally, the HDR rendering gives the user the possibility to change the exposure to bring out different ranges of details.

5.8 Conclusions and Future Work

We have presented the development of a realtime demo, reproducing the time lapse sequence from the short movie *The Parthenon*.

The aim of this chapter was to present the various choices we had to do to implement an interactive application able to cope with a very complex (and fragmented) 3D model and with an advanced lighting technique. We discussed technical problems arisen by this project peculiarities, explaining our technical decisions and outlining possible alternatives.

In particular, there were technical problems related to the size of the dataset and related to limitations of the target VR framework. However, the main problem was how to efficiently implement a complex and faithful lighting algorithm, able to match the one used in the offline movie rendering.

To solve these problems, we presented methods to make a scanned 3D model more adequate for realtime rendering applications and we described lighting algorithms based on the separation between direct/indirect light and extensive precalculation. Finally, we explained how the described algorithms can be implemented in a pre-existing programming framework.

Even if the implementation details may vary between different projects, we believe this separation approach is a good solution. Providing separate solution for each problem can give the possibility of successive updates. This approach also makes possible the modular composition of algorithms; for each project we can select those techniques that give the best contribution to realism but are also computationally affordable.

The described algorithms have a quite local execution, all the necessary data is associated to the geometry, and all the computation is implemented in the hardware shaders. This greatly helps when it is needed to integrate those techniques inside bigger projects. As seen in this case, it was possible to integrate the algorithms in a framework as specific as a game engine. We are sure that doing the same thing in other engines is a possible and affordable action.

We showed the advantages of using an existing rendering engine to precalculate lighting invariants. Without the need of a long, difficult (and error-prone) implementation of a lighting algorithm, it was possible to calculate lighting invariants just adapting the existing Arnold rendering engine. In this case this was also useful because we used the same engine used for the movie rendering; this was a guarantee of the lighting coherence between the movie and the demo.

The chosen techniques have proven to be accurate in terms of rendering results; shading is visually comparable to the one obtained in the movie, even if some simplifications have been introduced to perform it in realtime.

At the same time, they have proven to be affordable in terms of execution time. The demo runs at a sufficient frame rate; moreover, future improvements of video cards will also confirm that the choice of using hardware shaders is a good option.

We tried to include into the demo the bloom HDR effect, but with not much

success. Having done the lighting calculation using HDR values helped determining which areas of the final image were over-saturated and should bleed over the neighbor areas. However, execution was too cumbersome and slowed the system; furthermore, the result was not really visually pleasant. So, we decided to not include this effect in the final release.

Chapter 6

Beyond Pure Rendering

In this chapter we will present two software tools developed in conjunction with Cultural Heritage partners. As stated in the introduction, we focused our research effort applying 3D computer graphics in the Cultural Heritage field. A first task was to demonstrate the potentiality of 3D scanning to acquire and record shape information of works of art. However, even more important was the objective of finding new usages, presentation and interaction paradigms to make profitable use of digital models in the CH field.

We believe this objective can only be reached trough a tight collaboration with Cultural Heritage experts; we know the technology well, but it is necessary the contribution of someone working in the CH field to define real needs and to evaluate the effectiveness of the approaches we propose. The partners we involved contributed in both the development and testing phase, giving us precious ideas and advices.

A first need that emerged from the interaction with restorers is the automatic generation of technical prints, which are the traditional instrument used to plan, document and monitor the restoration process. Therefore, we designed and implemented *Cavalieri* tool that, given a digital model of the object, can be used by restorers to define the needed views and cross-sections and to generate them in the form of high resolution images/prints.

Another important need was related to interactive exploration of 3D models for naïve users, like museum visitors. We designed *Virtual Inspector*, a tool for visualization and browsing of high quality 3D models, especially designed for museum installations.

In both cases, the interaction with the tools was a central point; it should be powerful enough to drive all the tool features but at the same time easy enough to be mastered by non experienced users. This has been obtained during development, by following the advices of our CH partners and, more important, during the assessment phase by correcting errors done in the initial design. We believe the effectiveness of tools should me measured by users and not by developers; we made the instruments available for testing in various projects (as we will show in Section 6.3), collecting impressions, requests for modifications and, obviously, signalling the presence of

bugs.

The work devoted to the enhancement of rendering realism was also part of this collaboration with Cultural Heritage operators. Interacting with the naked geometry sometimes cannot convey the level of realism necessary for a natural interaction. For this reasons, these two tools have been extended to include some of the techniques presented in the previous chapters (this is, however, still an ongoing activity).

6.1 Technical prints generations

The production of drawings is a basic activity in restoration, archeology and Cultural Heritage didactics. The manual production of technical drawings is a complex process, both in terms of time and skills required. To overcome those problems, we developed a tool to automatically produce technical drawings of the artifacts using its 3D model. Technical drawings of archeological findings are produced with two main motivations: (a) to have a graphic documentation (more synthetical and rich of meta-information than photographs) of the shape and of the preservation conditions of the artifact, and (b) to be used in restoration to plan, design and document the actions to be done.

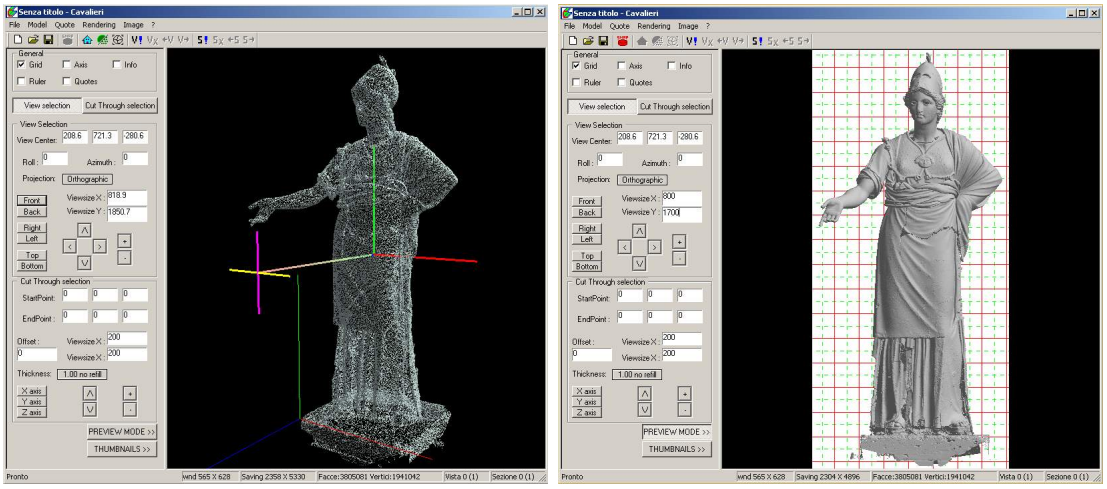

Figure 6.1: The CAVALIERI tool. Left: interface for view definition. Right: preview of the resulting image.

The *Cavalieri* tool [95] has been developed as a software devoted to the production of such graphic printouts and has been used in various cultural heritage projects we have worked in, like the restoration of Michelangelo's David [18] and of the Minerva d'Arezzo. The tool manages the huge digital models produced with 3D scanning devices and supports easy specification of orthographic drawings and

cut-through sections, which are given in output as very high resolution images. It is necessary to specify that this is not a straight rendering task, because there are some constrains not normally considered in standard CG programs:

- The kind of images required are mostly orthographic projections and cut-trough sections. User should be able to precisely control the view parameters, in order to obtain useful drawings.
- It is necessary to generate images at a known scale and at very large resolution (for the A1 format, to reach a mere 300 dpi resolution, the image should be 7000×9900 pixels)
- The tool should be able to work with huge 3D models, to effectively use the maximum available detail.

The function of the tool is to generate technical drawings; it is important to point out that this kind of images are not mere snapshots. First of all, technical drawings have a known scale (and thus a physical dimension); this permits a correct editing and printing. Moreover are also precisely defined the extent of the represented area, the viewpoint center and the view direction, giving the possibility to take measures.

The tool has been implemented with a simple user interface to help the user in defining view and sections parameters. Definition of such parameters can be accomplished using a combination of three principal instruments.

Presets user can decide to use some pre-set options, like frontal view or sagittal sections; numerical values are generated accordingly to object characteristics. Presets are available for both views and cut-trough sections and are a good starting point to define a custom view/section.

Manipulators user can also modify views and sections by using manipulators (translation, rotation) directly into the 3D view area using click-and-drag operations, like in many 3D editors. Manipulators are the fastest (and more intuitive) way to modify parameters.

numerical editors using the widgets visible on the left side of the window (Figure 6.1), the user can enter all the necessary parameters (visually checking the result on the 3D view). This gives the precision needed for fine tuning.

Examples of the drawing produced by the tool can be seen in Figures 6.4 and 6.5.

Obviously, for standard measuring purpose, a simple lambertian shading is totally sufficient. Conversely, when the image is used for documentation or for annotations, it is necessary to provide as much surface information as possible. Having a more detailed visualization helps in finding correspondence between the drawing

and the real object. We already showed in Figure 4.4 how the ambient occlusion can enhance the readability of a 3D model rendering.

Using Cavalieri, the user can select the appropriate view and than can also define the illumination environment more appropriate to exalt the interested area. It is then possible, when generating the image, to choose to include in the computation the surface color and/or the occlusion term. Is to be noted that, while occlusion can greatly enhance the object perception by providing additional surface characterization, color can in some case be confusing. Also hard shadows are normally excluded because, while exalting creases and discontinuities of the surface, they *hide* information in the obscured areas.

The system is able to produce very large drawings for printing purpose; most of the times the target printer is an A1 or A0 plotter, making the needed resolution grow incredibly (the maximum dimension requested so far was 15k x 15k pixels). It's impossible to generate such images with a single rendering and it's also impossible to keep it all in memory. To solve both problems, images are generated using a tiled rendering and progressively saved on disk. Using the LibTIFF, the space for the image is allocated on disk (using the tiled-tiff format), the total viewport is regularly divided in chunks, each chunk is generated trough a single rendering and directly stored on disk. Thus, a much lower memory footprint is required during the image generation task.

Again, since it is often needed to obtain very high resolution images the 3D models needed are quite large (above 10M triangles). The system makes full use of our multiresolution technique [32]. This data structure has been extended to support both color and occlusion as explained in the previous chapters (Chapter 3 and 4); so, also this tool has the possibility to make profitable use of that kind of data.

6.2 Virtual Inspector

The Virtual Inspector tool is an interactive system for the visualization of complex and accurate digital 3D models. The tool is mainly oriented to the detailed visual analysis (hence, inspector) of single works of art (like sculptures, pottery, etc.).

Cultural Heritage is a specific field of application of 3D graphics, where a basic issue in the design of visualization tools is the ease of use: forecasted users usually possess limited skills in managing 3D graphics technology. People like museum curators, art historians, restorers or even ordinary visitors of a museum are not used to manipulate 3D entities trough the same interfaces used by people working in 3D graphics, CAD operators and videogames players.

The design of the GUI and its overall usability play a critical role in deciding if the tool will be just a nice looking toy or a really useful technical instrument. Interaction with 3D data should be as easy and natural as possible; usability should

Figure 6.2: VIRTUAL INSPECTOR displaying two color-mapped models of the David, showing side-by-side the color before and after the restoration (the two trackball of the models are synchronized). The data is the same presented in Chapter 3

be given priority over flexibility and completeness, avoiding users to feel lost in virtual space while manipulating/inspecting the digital model.

For this reason, a main goal in the design of the system was to provide the user with a very easy and natural interaction approach, based on a straightforward "point and click" metaphor. It was designed according to the specification and needs of both restorers and art curators, and qualifies as a very handy tool for discovering the beauty and complexity of works of art, implementing a multimedia kiosk for museums or expositions.

A standard layout of the Virtual Inspector tool is shown in Figure 6.6. A complete model of the inspected object, called dummy, is visualized in the rightmost frame. It is conceived as a sort of interactive and intuitive 3D map whose role is to allow an easy selection of the view specs. The dummy is always visualized in full view. The user can rotate the dummy and see the dummy from any side view; a simple mouse click on any point of the dummy updates the current view, making the clicked point the center of the new view. User can also interact with the model, using a standard trackball on the left side of the window.

An important feature of the tool is the extreme interface configurability. To be

able to cope with the specific needs of a given artifact, it is possible to change the appearance and the functionality of the interface. Interface entities like buttons, icons and links can be choosed and arranged on the screen; geometric and rendering entities like 3D models, lights and trackball can also be specified. This has been obtained by designing a simple interpreted language, called NSP, that is used to build up the interface with its behavior. The language exploits XML for all the well known advantages of this technology (availability of syntax aware parsers, human readability, extendibility, etc.).

At startup, Virtual Inspector reads the `.nsp` file and configure itself opportunely accordingly to the instructions there specified. With this approach, the designer of the multimedia application does not have to compile a new version of the system to obtain a new type of layout, but he can simply design and distribute a new GUI layout by simply typing a new `.nsp` file. The `.nsp` file is composed by an XML element named `inspector`, inside this object can be found a sequence of XML elements that declares all the 3D entities (models, viewers, lights) that will be displayed and the current screen configuration (e.g. resolution used). The XML approach allows to change very easily the look and feel of Virtual Inspector.

As an example, compare the different layout and graphics presented in Figures 6.6, 6.3 and 6.2. This is not a pure cosmetic makeup; beside the appearance of background and icons, between the various interfaces there are also changes in the interaction modes. For example, the synchronized movement of the two models in Figure Figures 6.6 and 6.2 is specified in the configuration file.

A feature, especially required by Cultural Heritage partners, is the possibility to associate additional information or links to external data to specific points on the surface of the artifact. This feature is implemented trough *Hot Spots*, defined in the system configuration and located on the 3D surface; user can activate the Hot Spot by simply clicking on it. Triggering an Hot Spot can result in displaying an image or switching to another 3D model or to an HTML page (presented by a standard web browser).

Visualization efficiency is obtained by adopting a *continuous* level-of-detail (LOD) representation. For each frame, the best-fit *variable resolution* LOD is selected according to the current view frustum and the requested visualization accuracy. LOD selection and rendering are very efficient since we adopt a patch-based representation, where a coarse-grain multiresolution hierarchy is visited on the fly and ready-to-render geometry patches are associated to each logical node of the variable LOD produced [33].

Another interesting feature of the tool is the possibility to integrate its functionalities with a standard web browser; the user can navigate html pages with written or graphical description of the object or of its historical background and then have access to the 3D model browsing. An example of this possibility is presented in Section 6.3.3.

To give a more realistic visualization of the object, we included in the basic rendering mode the possibility to use the color mapped onto the geometry (the

system supports both texture mapped and per-vertex coloring). The use of colored models it is shown in Figure 6.2, where the same 3D models we used as mapping example in Chapter 3 are displayed.

The user can move around the object, to observe it from any point of view; it can also control the lighting direction. Since lighting is essential for our geometric perception, this feature greatly simplify the object comprehension. A simple phong lighting cannot give the correct stimulus; using the precomputed ambient term as explained in Chapter 4 we can obtain a much more correct lighting.

Figure 6.3: VIRTUAL INSPECTOR showing the Minerva head, rendered on the basis of the BRDF model sampled from the real statue.

This hybrid approach (static diffuse lighting field plus dynamic head-light) gives the best trade-off between interactivity and rendering quality, allowing the user to have a view-dependent lighting, while also having a correct darkening of all the less exposed areas.

Again, given the aim of this tool, we decided to avoid hard shadows. While not technically difficult to implement, we believed that, since the aim of the tool is reproduce a close inspection of the object, this feature would not enhance the user perception of the object.

As stated in the State of the Art (Chapter 2), the most accurate way to represent a surface characterization is trough its BRDF. Thanks to the configurability of this software's rendering engine, we implemented the possibility to render a spatially

varying BRDF, obtaining a very realistic visualization for the 3D models that have mapped this level of information.

The spatially varying BRDFs approach included in VIRTUAL INSPECTOR follows the sampling and rendering approach proposed by Lensch et al. [83]. It has been implemented, in collaboration with MPI colleagues, in an efficient manner by exploiting the programmable features of modern programmable GPU. An example of the Minerva head rendered using a sampled BRDF is shown in Figure 6.3.

6.3 Cultural Heritage projects

6.3.1 Restoration of Michelangelo's David

The *Michelangelo's David* restoration project has given several guidelines to the definition and development of innovative solutions to process and visualize 3D data in the framework of Cultural Heritage applications. Our main goal has been to demonstrate the usefulness of digital 3D models and of visualization tools in the framework of a restoration project.

Restoration is nowadays a very complex task, where multidisciplinary skills and knowledge are required. A complex set of investigations usually precedes the restoration of a valuable artwork: visual inspection, chemical analysis, different type of image-based analysis (RGB or colorimetric, UV light reflection, X-Ray, etc.), structural analysis, historical/archival search, etc.

An emerging quest is how to manage all the resulting multimedia data (text, annotations, historical documents, 2D/3D images, vectorial reliefs, numeric data coming from the analysis, etc.) in a integrated framework, making all information accessible to the restoration staff (and, possibly, to experts and ordinary people as well). Since most of the information is directly related to spatial locations on the artwork surface, 3D models can be valuable media to index, store, cross-correlate and obviously visualize all this information. 3D models can also be a valuable instrument in the final assessment phase, supporting the interactive inspection of the multiple digital models (pre- and post-restoration status) to check possible shape and color variations.

We experimented different uses of 3D graphics for the restoration of the David [18, 19], ranging from the classical *scientific visualization* tasks to more complex *information visualization* applications. According to our experience, the tools available (either commercial or academic systems) do not satisfy all the potential needs of *computer-aided restoration*. While in some cases standard visualization features are sufficient (e.g., to present a scalar field over a 3D surface), other applications often require more sophisticated tools to map multimedia information to the artwork surface and to present those data visually, e.g., joining the capabilities of a 3D GIS with the ones of a multi-media information visualization system.

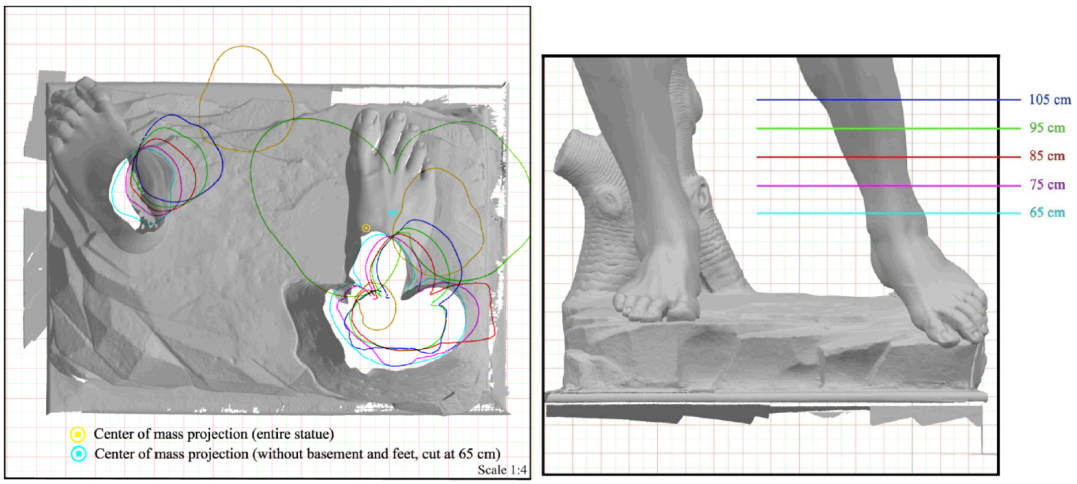

Figure 6.4: Technical illustration obtained composing images produced with the Cavalieri Tool. The illustration shows the projection of the barycenter over the base; the two views and the sections help understanding the spatiality of the object. Actually, there are two barycenters: one for the whole statue, the other one just consider the statue above the ankles. This double calculation was needed to evaluate the dangerousness of some fractures presents in the ankles.

The David restoration is has proven to be an ideal testbed, since a complex set of scientific investigations has been planned both before and after the restoration intervention. This gave us the opportunity of attempting different methodologies to support restorers/scientists with the use of visualization tools based on 3D digital models.

Graphical artifacts produced with the Cavalieri Tool were used to as maps to help planning the restoration actions. Another interesting use was to generate technical illustration of some of the calculation we performed using the 3D model, like the barycenter and the surface exposition to contaminants fall.

Figure 6.4 shows the illustration produced to illustrate the the barycenter calculation. This calculation has been performed to evaluate the stability of the statue; the barycenter has been calculated and projected on the base. Moreover, since there are some fissures in the ankle area, two barycenters have been calculated; one for the whole statue and another one for just the volume above the ankles. The distance of the projection over the base gave useful indication of the stability of the statue. The produced illustration presents the results in a visual manner, quite easy to understand.

Technical drawings were also used as a map to help managing and locating all the various analysis performed over the statue during restoration; an example of the maps produced for documentation can be seen on the left side of Figure 6.5.

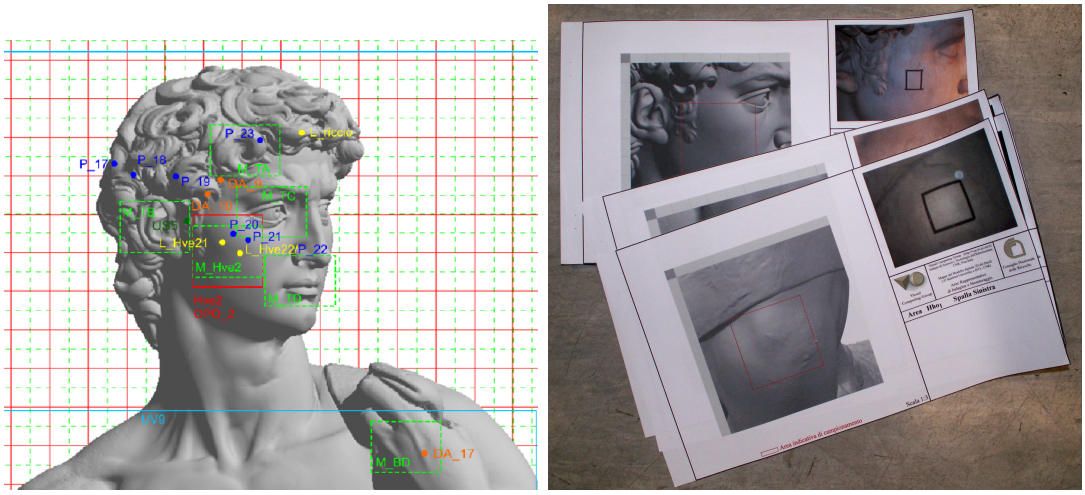

Figure 6.5: Left: the analysis performed on the statue during restoration were located on the surface of the 3D model. Here is represented a parcel of the reference scheme produced as documentation. Right: one of the booklets given to the technicians working on the David during restoration. Booklets were used as on-field annotation blocks for restorers and technicians.

Booklets were distributed to restorers and technicians; the aim was to give a common reference instrument for the analysis phase and the subsequential data entry. A series of areas were selected as target for analysis; the booklet show each area with a photo, a description and a graphical representation of the area, produced from the 3D model (see right side of Figure 6.5). The booklet was firstly used as a precise visual reference to unequivocally identify the area to be analyzed and then as an space for annotations, common for all the restorers.

The Virtual Inspector has also been used in this project but, since the tool was still under development during the restoration process, only to present some of the final results.

6.3.2 Restoration of the Minerva d'Arezzo

The Minerva d'Arezzo is a bronze statue discovered in Arezzo in 1541 whose origin is still uncertain: it could be either an original Hellenistic bronze dating back to third century B.C., or a variant produced in the Roman Imperial period (first century A.D.). The statue is usually located in the Archaeological Museum in Florence, and is now under restoration at the Archaeological Restoration Center in Florence.

The Minerva project intends to show how 3D techniques can integrate standard diagnostic methods giving useful and powerful tools to the restorers. The main goal of the project is to build 3D digital models of the Minerva statue before, at

different stages during restoration and at the end of the process in order to monitor the various phases of the restoration. The case study is very stimulating because, giving the high level of degradation of the statue, the restoration process will be highly invasive, producing variations of the shape (removal of plaster, polishing of the corroded bronze surface) and of the surface appearance (removal of the thick paint film on the bronze surface) of the statue.

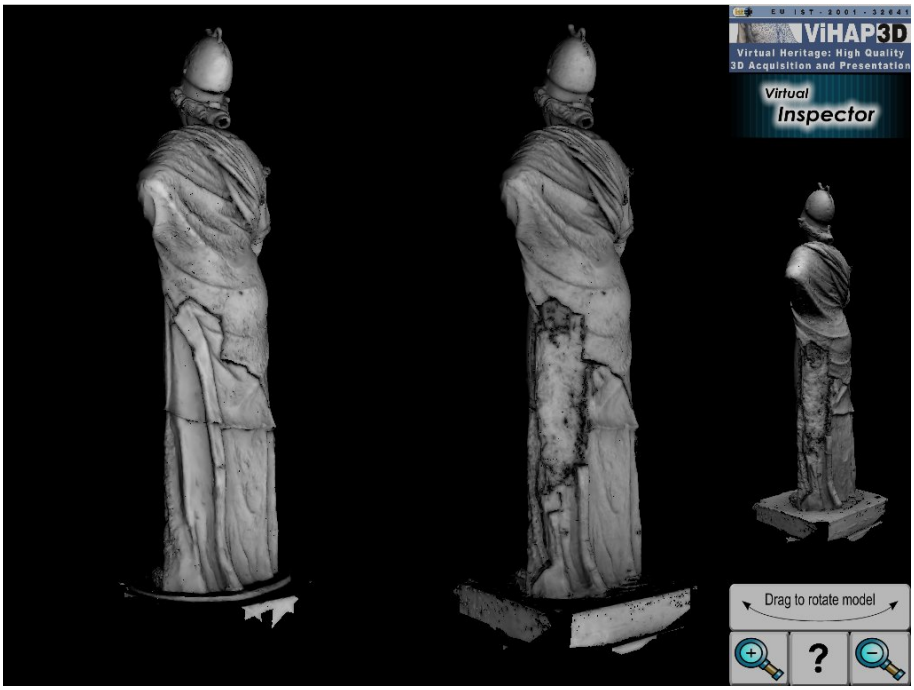

Figure 6.6: Another example of the side-by-side presentation. In this case, the Minerva of Arezzo is shown in two different steps of its restoration process, with quite different geometry. The clarity of presentation is enhanced using the occlusion term shading

Up to now, four different models of the Minerva have been scanned in the period between 2001 and 2004. The statue is now completely dismantled in its components, the restoration of each part has been completed and we are waiting for the reintegration of the restored parts to perform the last acquisition (planned for November-December 2006), to obtain the final geometry.

During restoration, the Cavalieri tool has been widely used to generate reference images for the planning of the restoration and for graphically map results of the analysis.

The statue is hollow; its surface is composed by many fragments and in the internal part there is a complicated wooden and metal structure, built in the 17th century to hold the components together. The cross sections produced with the Cavalieri Tool were the ideal support for the restorers. Sections can give a precise

representation of the internal space; restorers can take measures and draw schemes of the internal structure while they explored it.

The images representing the external surface were also very useful, by giving a precise (more precise than human draw) and flexible (any view can be generated in short time) support for technical annotations.

Virtual Inspector has been also used during the various phases of restoration. A first objective was to present the evolution of the geometry during the statue restoration. The four models can be viewed side-by-side to compare the geometries in their evolution trough time. In Figure 6.6 it is possible to see how the side-by-side comparison of the two geometries (model 2 and 4) can help in presenting the change of the shape of the statue.

Another issue was the idea of maintaining a faithful representation of an appearance that for sure would have been lost; the restoration has changed not only the shape but also the surface appearance of the object in a dramatic way. The BRDF capture and its subsequential visualization using the Virtual Inspector (Figure 6.3) is a way to preserve the surface status in an intermediate state of the restoration. BRDF acquisition has been performed in collaboration with the colleagues of the MPI, as explained in Section 6.2.

6.3.3 The Mausoleum of Arrigo VII

The aim of this project was to perform a digital acquisition of the Arrigo VII complex and to design a new approach for its archival, study and presentation, oriented both to experts and to the public [2].

The Arrigo VII Mausoleum was one of the most famous funerary monuments of the XIV century, perceived as a model by contemporaries. It was carried out by Tino di Camaino, an artist who played a big role in the birth of modern sculpture in Europe. Unfortunately, this masterpiece was dismantled soon after its completion. After almost a century of attempts to reconstruct its original disposal, art historians are now looking with interest to the help offered by new technologies.

3D graphics was perceived as the right tool for finding plausible solutions: the availability of 3D digital models of all the statues and of the architecture would in fact help a lot in the evaluation of recomposition hypothesis, offering a realistic perception of reciprocal relations between the elements of the assembly.

The activities of the project can be divided in the following tasks:

- acquisition of 3D models of all the elements of the Arrigo's monument, by adopting high-accuracy 3D scanning;

- post-processing of the raw scanned data, to build up a complete model for each statue and to derive from it optimized 3D representations (different level of details);

- modeling of the Pisa cathedral in its XIV cent. status, by both acquiring the current architecture with a scanning device and re-modeling the lost components with a CAD tool;

- design and implementation of interactive systems to present the models and the associated multimedia data both to ordinary public (museum presentations) and to experts (to support study and analysis of the complex).

Fifteen statues have been acquired, producing high resolution 3D models. The digital models, with the addition of some architectural elements modeled inside a CAD system, have been used to reconstruct of the original shape of the funeral monument.

The reconstruction has been performed under the supervision of art historians. In literature there are available different reconstruction hypothesis for the architecture of the monument; we started by reproducing them virtually. During this phase we discovered that some of them were wrong, since they do not fully considered the real size of the statues nor the orientation of the figures. Then, with the help of two experts on medieval sculpture, we proposed a new layout for the monument which, given the current knowledge, seems to be the most probable.

The results of the project are presented to the public with a multimedia kiosk build around the Virtual Inspector tool; the browsing of the digital model of the statues is the central node of an exploration path for a comprehensive (historical and artistic) knowledge the monument.

The flexibility of the tool and the possibility to integrate the 3D browsing with the navigation of HTML pages make the creation of the kiosk very easy.

The Arrigo VII visual presentation in the museum (visible in Figure 6.7) starts from the HTML pages, which provide both a detailed artistic and historic overview of the Arrigo VII complex, and providing links to activate Virtual Inspector on the different statues. These introductory/index pages hold also links to time-navigation pages, which present for each sub-set of statues the different locations during their seven century lifespan (presented to the user through some pre-computed videos).

The design of the Arrigo VII installation has been carried out with the help of a professional graphic designer. Thanks to the extreme configurability of the tool, it was easy to change the graphical appearance to uniform it to the one used in the HTML pages, giving to the whole installation the same look-and-feel.

The multimedia kiosk was installed in the Museo dell'Opera del Duomo in Pisa on October 2004, in the room where a subset of the Arrigo statues is exposed.

To validate the effectiveness of the installation, we have done a formal evaluation of the its usability. This evaluation has been carried out by an external entity (Megaride, a company spin-off of the University of Naples) specialized in this kind of actions. First the application was analyzed out of context, just in terms of accessibility and ergonomic considerations (following the guidelines of ISO 9241-11 [55]); afterwards, as a communication experiment inside a specific museum. The

first analysis resulted in changes to the installation, to meet the ergonomic and accessibility requirements. In the second phase the museum, its visitors and their interaction with the installation have also been analyzed. This has been done using questionnaires, personal interviews and by recording and analyzing the interaction of the user with the installation. The evaluation yielded very good results, proving the effectiveness of the installation and registering the approval of visitors.

Figure 6.7: The Arrigo VII museum installation. Top row: HTML pages present historical and artistic information for the monument. Trough the statues images it is possible to start the 3D browsing with Virtual Inspector. Middle and Bottom row: 3D model exploration, user can choose the viewing position, modify the lighting and click on hot spot to have access to localized information.

Chapter 7

Conclusions

The goal of this thesis was to design new techniques and algorithms to enhance visualization realism when working with high resolution 3D models.

A digital 3D model is a mathematical representation of the shape of a physical object; this abstraction from reality is essential, since a computer can only manipulate this kind of entities. When we render it, we are transforming this mathematical representation into something (an image) that can be perceived trough our vision system, resulting in a familiar perceptual stimulus.

Given this objective, the more close this representation is to reality, the more useful is to us. We will be able to manipulate, study and work with these renderings much better, if we will be able to perceive them correctly.

The quest for realism has been the focus of various researches in the recent years but, when dealing with very complex 3D models, the quality of the rendering still lacks the necessary level of realism. Our work has been devoted to define methods to obtain more realistic renderings when working in realtime with complex 3D models. At the same time, we worked to define new interaction and visualization methodologies to maximize the effectiveness of the 3D graphics technology in its application to the Cultural Heritage field.

One of our goal was to define not just new *functional* algorithms, but methods applicable in **real** projects. In our experience all projects are different from each other due the nature of the data or due to specific needs and constrains; so, it is unrealistic to be able to give a single full-spectrum solution, able to work in any situation. The techniques we discussed in this thesis are modular and self contained, each one focused to solve a specific problem. This approach gives us the needed flexibility and helps the composition of more complex solutions, each one tailored to the needs of that particular project.

7.1 Enhanced realism

To obtain a more realistic rendering of very large 3D models we worked both giving a better surface characterization (to generate good color mapping) and computing a more accurate lighting model (introducing ambient occlusion and shadows)

In Chapter 3, we described an efficient and precise way to map color information from large photo samplings to the digital model. Standard processing methods cannot cope with very large 3D models (more than 5 M triangles) and huge photographic datasets (more than 100 MPixels). We introduced a multivariate per-pixel weighting system, able to deal with most of the problems generally related to photographic mapping. The weighting function is both effective and scalable with respect to the dataset size. To show the approach robustness, we presented as test case the mapping of the Michelangelo's David photographic campaign. In this way we showed how this technique efficiently scales with the number of photos and with the size of the target 3D model.

We also discussed the advantages of mapping information on a per-vertex basis; this grants both computational and manipulation efficiency and simplicity, while yielding pleasant visual results.

In Chapter 4 we presented a way to enrich the standard phong lighting used to display very large models, with the aid of Ambient Occlusion. Ambient occlusion is an effective way to approximate the effect of a complex global illumination calculation. Again, we concentrate our work to define efficient and precise methods to calculate and display this value for very large 3D models.

We showed how the calculation can be implemented using an approximate method that uses the computational power of accelerated video cards, but also how to use existing rendering engine to obtain a more correct result.

This second method is particulary interesting, since many other useful values can be efficiently (and precisely) calculated for 3D models, by adapting existing rendering engines.

7.2 Integration flexibility

We have shown how it is possible to integrate the basic techniques into existing visualization systems and rendering libraries. This feature is particulary important, since many of the algorithms available in literature are stand-alone techniques; they often rely on the total resource availability, thus making impossible mixing various techniques or adding it to an existing system.

We tried to keep each technique we presented, well separated and self-contained. This will make possible to choose, depending on the project characteristics, the more effective techniques to employ.

As an example of how this is possible, in Chapter 5 we described in detail the

realization of a realtime demo, where a very complex dataset has been coupled with an advanced lighting algorithm, producing a nearly-photorealistic rendering. More than the single algorithms employed, it is important how each element of the demo has been chosen and tailored to follow the constrain of the project.

7.3 Cultural Heritage

As explained in the introduction, our work has been focused to the application of the 3D computer graphics to the Cultural Heritage field. In this context, beside the enhancement of the visual quality for the 3D models, it was also important to give new interaction paradigms and propose new usage methods to

Two of such tools have been presented in Chapter 6. We were requested for a tool to generate technical illustration from the 3D models, to be used in the restoration process. Another important objective was to develop a flexible application to display large 3D models using a very intuitive interface, to be used in the creation of multimedia museum installations. The *Cavalieri* tool and *Virtual Inspector* have been designed following these needs.

The tools are the results of a collaboration between Computer Science researchers and Cultural Heritage operators, such as restorers, curators or art historians. This collaboration was essential, since people from CH field know well which are the needs of their work, and helped in testing the tools. This resulted in obtaining a more effective definition of the tools functionality and simpler user interfaces.

An important point is that the effectiveness of the presented tools has been evaluated not just using testing datasets, but trough their use in *real* Cultural Heritage projects and, in the case of Virtual Inspector, also trough a specific assessment process.

7.4 Remarks and future work

We know that computer capabilities will continue to grow in the next years, providing much more computational power than today at even lower costs. However, it is not conceivable to be able to compute very complex lighting without the need of optimization even on this kind of machines. This also because, as the computational power will grow, will also grow the need for larger and more complex 3D models.

For this reason, the approach of precomputing the lighting invariants and finish the lighting process using programmable video hardware will still be a good solution. The development of the new generation of programmable video cards will also contribute, bringing even more processing power, thus giving the possibility to implement more complex lighting models.

Certainly, the quest for realism when dealing with very large datasets is still far from being completed, but we believe the approaches presented in this work can be a good base for future development.

Open issues remains in the color mapping. Even if the results are visually good, there are many photographic problems that are still to be addressed, like shadows and highlights. Moreover, implementing a deshading technique will give the opportunity to obtain a lighting independent color, that can be used for re-illumination. To maintain the generality of the method, without requiring any additional information or calibration system, an idea is to implement a method to estimate the lighting at the moment of the shot.

Regarding real-time shading, even if we are satisfied with the visual results of the Parthenon demo, we know that the implemented lighting model is still a rough approximation of the complete lighting calculation used in the rendering of the movie. Graphics hardware will give, in the near future, more possibilities to implement a more complex shading model. As new graphic hardware will be available, we will try to update the technique used, always keeping the idea of modular composition of single algorithms, to conserve application flexibility.

Another objective is the application of the realism techniques presented to the tools we have developed for Cultural Heritage operators, like the Cavalieri tool and the Virtual Inspector. This work, as showed in Chapter 6, is already started, but still far to be completed.

7.5 Related publications

The results of the research presented in this thesis has been the subject of publications on international journals as well as in the proceedings of international conferences. The main publications are as follows:

- For Chapter 3: our old texture builder tool, weaver is described in [21](Vision Modeling Visualization 2002) while the new mapping algorithm in [24](Pacific Graphics 2006).

- the work regarding the Parthenon demo, described in Chapter 5 has been published in [27](SPIE Optical Metrology 2005) [23](Computer and Graphics).

- the two systems described in Chapter 6, are presented in [22](Journal of Cultural Heritage) for Cavalieri and [25]EG Italian Chapter 2006 [26](IEEE Computer Graphics and Applications) for Virtual Inspector.

  The projects where those tools have been used are presented in [18](IEEE Computer Graphics and Applications) [2](VAST 2004)

Bibliography

[1] N. Bannai, A. Agathos, and R.B. Fisher. Fusing multiple color images for texturing models. In *3DPVT04*, pages 558–565, 2004.

[2] C. Baracchini, A Brogi, M. Callieri, L. Capitani, P. Cignoni, A. Fasano, C. Montani, C. Nenci, R. P. Novello, P. Pingi, F. Ponchio, and R. Scopigno. Digital reconstruction of the arrigo vii funerary complex. In *The 5th International Symposium on Virtual Reality, Archaeology and Intelligent Cultural Heritage (VAST 2004)*, Brussels and Oudenaarde, Belgium, December 7 - 10 2004.

[3] A. Baumberg. Blending images for texturing 3d models. In *BMVC 2002*. Canon Research Center Europe, 2002.

[4] F. Bernardini, I.M. Martin, and H. Rushmeier. High-quality texture reconstruction from multiple scans. *IEEE Transactions on Visualization and Computer Graphics*, 7(4):318–332, 2001.

[5] F. Bernardini and H. E. Rushmeier. 3D Model Acquisition. In *Eurographics 2000, State of the Art Reports Proceedings*, pages 41–62. Eurographics Association, August 24–25 2000.

[6] F. Bernardini and H. E. Rushmeier. Strategies for registering range images from unknown camera positions. In B.D. Corner and J.H. Nurre, editors, *Proc. Conf. on Three-Dimensional Image Capture and Applications II*, pages 200–206, Bellingham, Washington, January 24–25 2000. SPIE.

[7] F. Bernardini and H. E. Rushmeier. The 3D Model Acquisition Pipeline. *Computer Graphics Forum*, 21(2):149–172, March 2002.

[8] P. J. Besl and N. D. McKay. A method for registration of 3-D shapes. *IEEE Transactions on Pattern Analysis and machine Intelligence*, 14(2):239–258, February 1992.

[9] F. Blais. A review of 20 years of range sensor development. In *Videometrics VII, Proceedings of SPIE-IS&T Electronic Imaging, SPIE Vol. 5013*, pages 62–76, 2003.

[10] G. Blais and M. D. Levine. Registering multiview range data to create 3d computer objects. *IEEE Trans. Pattern Anal. Mach. Intell.*, 17(8):820–824, 1995.

[11] J. F. Blinn. Simulation of wrinkled surfaces. *Computer Graphics (SIGGRAPH 78 Proceedings)*, 12(3):286–292, August 1978.

[12] S. BOIVIN and A. GAGALOWICZ. Image-based rendering of diffuse, specular and glossy surfaces from a single image. page 107116, 2001. ISBN 1-58113-292-1 (Proceedings of ACM SIGGRAPH 2001).

[13] R. Borgo, P. Cignoni, and R. Scopigno. An easy to use visualization system for huge cultural heritage meshes. In D. Arnold, A. Chalmers, and D. Fellner, editors, *VAST 2001 Conference Proc.*, pages 121–130, Athens, Greece, Nov. 28-30 2001. ACM Siggraph.

[14] A. Bornik, K. Karner, J. Bauer, F. Leberl, and H. Mayer. High-quality texture reconstruction from multiple views, 2002.

[15] M. BOTSCH and L. KOBBELT. High-quality point-based rendering on modern gpus, 2003.

[16] T. Boubekeur and C. Schlick. Generic mesh refinement on gpu. In *In Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*. ACM Press, July 2005.

[17] Michael Bunnell. *GPU Gems 2*, chapter Dynamic Ambient Occlusion and Indirect Lighting, pages 223–233. Addison-Wesley, 2005.

[18] M. Callieri, P. Cignoni, F. Ganovelli, G. Impoco, C. Montani, P. Pingi, F. Ponchio, and R. Scopigno. Visualization and 3d data processing in david's restoration. *IEEE Computer Graphics & Applications*, 24(2):16–21, March/April 2004.

[19] M. Callieri, P. Cignoni, F. Ganovelli, G. Impoco, C. Montani, P. Pingi, F. Ponchio, and R. Scopigno. Restoring david using 3d. *IEEE Potentials - the magazine for high-tech innovators*, 23(5):4–7, 2005.

[20] M. Callieri, P. Cignoni, F. Ganovelli, C. Montani, P. Pingi, and R. Scopigno. VCLab's tools for 3D range data processing. In A. Chalmers D. Arnold and F. Niccolucci, editors, *VAST 2003*, pages 13–22, Bighton, UK, Nov. 5-7 2003. Eurographics.

[21] M. Callieri, P. Cignoni, and R. Scopigno. Reconstructing textured meshes from multiple range rgb maps. In *7th Int.l Fall Workshop on Vision, Modeling, and Visualization 2002*, pages 419–426, Erlangen (D), Nov. 20 - 22 2002. IOS Press.

[22] M. Callieri, P. Cignoni, R. Scopigno, G. Gori, and M. Risaliti. Beyond manual drafting: a restoration-oriented system. *Journal of Culturel heritage*, To appear, 2006.

[23] M. Callieri, P. Debevec, J. Pair, and R. Scopigno. A realtime immersive application with realistic lighting: the parthenon. *Computer & Graphics*, 30(3), 2006.

[24] Marco Callieri, Paolo Cignoni, Massimiliano Corsini, and Roberto Scopigno. Masked photo blending: mapping many hi-res images on dense 3d models. *Submitted to Pacific Graphics, under review*, 2006.

[25] Marco Callieri, Federico Ponchio, Paolo Cignoni, and Roberto Scopigno. Easy access to huge 3d models of works of art. In *Fourth Eurographics Italian Chapter 2006*, pages 29–36. Eurographics Association, 2006. Catania, 22-24 February 2006.

[26] Marco Callieri, Federico Ponchio, Paolo Cignoni, and Roberto Scopigno. Virtual inspector: a flexible visualizer for dense 3d scanned models. *Submitted to IEEE Computer Graphics and Applications, under review*, 2006.

[27] Pair J. Callieri M., Debevec P. Realistic realtime illumination of complex environment for immersive systems a case study: the parthenon. In *Optical Methods for Arts and Archaeology conference*. SPIE, June 2005.

[28] E. Catmull. *A Subdivision Algorithm for Computer Display of Curved Surfaces*. PhD thesis, Computer Science Department, University of Utah, December 1974.

[29] Y. Chen and G. Medioni. Object modelling by registration of multiple range images. *International Journal of Image and Vision Computing*, 10(3):145–155, April 1992.

[30] CIE. *Commission Internationale de l'Eclairage Proceedings*. Cambridge University Press, 1931.

[31] P. Cignoni, F. Ganovelli, E. Gobbetti, F.Marton, F. Ponchio, and R. Scopigno. BDAM: Batched dynamic adaptive meshes for high performance terrain visualization. *Computer Graphics Forum*, 22(3):505–514, Sept. 2003.

[32] P. Cignoni, F. Ganovelli, E. Gobbetti, F. Marton, F. Ponchio, and R. Scopigno. Adaptive tetrapuzzles: Efficient out-of-core construction and visualization of gigantic multiresolution polygonal models. *ACM Trans. on Graphics (SIGGRAPH 2004)*, 23(3):796–803, 2004.

[33] P. Cignoni, F. Ganovelli, E. Gobbetti, F. Marton, F. Ponchio, and R. Scopigno. Batched multi triangulation. In *Proceedings IEEE Visualization*, Conference held in Minneapolis, MI, USA, October 2005. IEEE Computer Society Press.

[34] P. Cignoni, C. Montani, C. Rocchini, and R. Scopigno. External memory management and simplification of huge meshes. *IEEE Transactions on Visualization and Computer Graphics*, 9(4):525–537, 2003.

[35] Daniel Cohen-Or and Yishay Levanoni. Temporal continuity of levels of detail in delaunay triangulated terrain. In Roni Yagel and Gregory M. Nielson, editors, *IEEE Visualization '96,*, pages 37–42, 1996.

[36] Chris Wynn NVIDIA Corporation. Real-time brdf-based lighting using cube-maps. www.nvidia.com/developer, 2002.

[37] PIXERA corporation. cooled ccd hdr camera. http://www.pixera.com/pixeracatalog/products

[38] B. Curless and M. Levoy. A volumetric method for building complex models from range images. In *Comp. Graph. Proc., Annual Conf. Series (SIGGRAPH 96)*, pages 303–312. ACM Press, 1996.

[39] Carsten Dachsbacher, Christian Vogelgsang, and Marc Stamminger. Sequential point trees. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2003)*, 22(3):657 – 662, 2003.

[40] P. Dave, T. Imai, J. Anstey, M. Roussou, and T. DeFanti. XP: An authoring system for immersive art exhibitions. In *Proceedings 4th International Conference on Virtual Systems and Multimedia*, Gifu, Japan, Nov 1998.

[41] J. Davis, S. Marshner, M. Garr, and M. Levoy. Filling holes in complex surfaces using volumetric diffusion. In *First Int. Symp. on 3D Data Processing, Visualization and Transmission (3DPVT'02)*, pages 428–438. IEEE Comp. Soc., 2002.

[42] P. Debevec, T. Hawkins, C. Tchou, H.-P. Duiker, W. Sarokin, and M. Sagar. Acquiring the reflectance field of a human face. In Kurt Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, Annual Conference Series, pages 145–156. ACM Press / Addison Wesley Longman, 2000.

[43] P. Debevec, C. Tchou, A. Gardner, T. Hawkins, C. Poullis, J. Stumpfel, A. Jones, N. Yun, P. Einarsson, T. Lundgren, P. Martinez, and M. Fajardo. Estimating surface reflectance properties of a complex scene under captured natural illumination. *ACM Transactions on Graphics*, Oct 2004.

[44] P. E. Debevec and J. Malik. Recovering high dynamic range radiance maps from photographs. In Turner Whitted, editor, *Comp. Graph. Proc., Annual Conf. Series (SIGGRAPH 97)*, pages 369–378. ACM SIGGRAPH, Aug. 1997.

[45] P.E. Debevec, C.J. Taylor, and J. Malik. Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 11–20. ACM SIGGRAPH, Addison Wesley, August 1996.

[46] F. Dornaika and C. Garcia. Robust camera calibration using 2d to 3d feature correspondences. In *Proceedings of the International Symposium SPIE –Optical Science Engineering and Instrumentation, Videometrics V, Volume 3174*, pages 123–133, 1997.

[47] G. Drettakis, M. Roussou, N. Tsingos, A. Reche, and E. Gallo. Image-based techniques for the creation and display of photorealistic interactive virtual environments. In *Eurographics Symposium on Virtual Environments*. EUROGRAPHICS, 2004.

[48] M.A. Duchaineau, M. Wolinsky, D.E. Sigeti, M.C. Miller, C. Aldrich, and M.B. Mineev-Weinstein. ROAMing terrain: Real-time optimally adapting meshes. In *Proceedings IEEE Visualization '97*, pages 81–88. IEEE, October 1997.

[49] David W. Eggert, Andrew W. Fitzgibbon, and Robert B. Fisher. Simultaneous registration of multiple range views for use in reverse engineering of cad models. *Comput. Vis. Image Underst.*, 69(3):253–272, 1998.

[50] J. El-Sana, J. Azanli, and A. Varshney. Skip strips: maintaining triangle strips for view-dependent rendering. In David Ebert, Markus Gross, and Bernd Hamann, editors, *IEEE Visualization '99*, pages 131–138, San Francisco, 1999. IEEE.

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

[52] Carl Erikson, Dinesh Manocha, and III William V. Baxter. Hlods for faster display of large static and dynamic environments. In *SI3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 111–120, New York, NY, USA, 2001. ACM Press.

[53] Marcos Fajardo. Montecarlo ray tracing in action. In *Course 29*. ACM SIGGRAPH, 2001.

[54] Randima Fernando, editor. *GPU Gems*. Addison-Wesley, 2004.

[55] International Organization for Standards. Ergonomic requirements for office work with visual display terminals. parts 1-11. http://www.iso.org/.

[56] T. Franken, M. Dellepiane, F. Ganovelli, P. Cignoni, C. Montani, and R. Scopigno. Minimizing user intervention in registering 2d images to 3d models. *The Visual Computer*, 21(8-10):619–628, sep 2005. Special Issues for Pacific Graphics 2005.

[57] S. Rubin G. Miller and D. Ponceleon. Lazy decompression of surface light fields for precomputed global illumination. In *9th Eurographics Workshop on Rendering*, pages 281–292, June 1998.

[58] Theo Gevers. Reflectance-based classification of color edges. In *9th International Conference on Computer Vision (ICCV'03)*, volume 2, page 856, 2003.

[59] HOWARD T. GIBSON, S. and R. HUBBOLD. Flexible image-based photometric reconstruction using virtual light sources. *Computer Graphics Forum*, 20(3), 2001.

[60] Robin Green. Spherical harmonic lighting: The gritty details. Sony Computer Entertainment America, Pdf document, 2003. http://www.research.scea.com/-gdc2003/spherical-harmonic-lighting.pdf.

[61] HAMAMATSU. High dynamic range camera. http://www.hpk.co.jp/Eng/products/SYSE/C7700E.htm.

[62] TORRANCE K. E. SILLION F. X. HE, X. D. and D. P. GREENBERG. A comprehensive physical model for light reflection. In *ACM SIGGRAPH*, volume 25, pages 175–186, July 1991.

[63] Philippe Bekaert Jan Kautz Marcus A. Magnor Jochen Lang Hendrik P. A. Lensch, Michael Goesele and Hans-Peter Seidel. Interactive rendering of translucent objects. In *Pacific Graphics*, pages 214–224, October 2002.

[64] H. Hoppe. View-dependent refinement of progressive meshes. In Turner Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 189–198. ACM SIGGRAPH, Addison Wesley, August 1997. ISBN 0-89791-896-7.

[65] H. Hoppe. Efficient implementation of progressive meshes. *Computer & Graphics*, 22(1):27–36, 1998.

[66] Richard S. Hunter and Richard W. Harold. *The measurement of appearance*. Wiley, 2nd edition, 1987.

[67] Andrey Iones, Anton Krupkin, Mateu Sbert, and Sergey Zhukov. Fast, realistic lighting for video games. *IEEE Computer Graphics and Applications*, 23(3):54–64, May/June 2003.

[68] Martin Isenburg, Peter Lindstrom, Stefan Gumhold, and Jack Snoeyink. Large mesh simplification using processing sequences. In *Visualization'03 Conference Proceedings*, pages 465–472, 2003.

[69] Y. Iwakiri and T. Kaneko. Pc-based real-time texture painting on real world objects. In *Computer Graphics Forum (Eurographics 2001)*, volume 20(3), pages 105–113, 2001.

[70] Tim Hawkins Jonathan Cohen, Chris Tchou and Paul Debevec. Real-time high dynamic range texture mapping. In *12th Annual Eurographics Workshop on Rendering*, June 2001.

[71] S. Nayar K. Dana, B. van Ginneken and J. Koenderink. Reflectance and texture of real-world surfaces. *ACM Transactions on Graphics*, (18(1)), January 1999.

[72] James T. Kajiya. The rendering equation. *Computer Graphics (SIGGRAPH)*, 20(4):143–150, 1986.

[73] J. Kautz. Hardware rendering with bidirectional reflectances. Technical Report CS-99-02, University of Waterloo, 1999.

[74] J. Kautz and M. McCool. Interactive rendering with arbitrary brdfs using separable approximations. In *10th Eurographics Rendering Workshop*, 1999.

[75] Jan Kautz, Peter-Pike Sloan, and John Snyder. Fast, arbitrary BRDF shading for low-frequency lighting using spherical harmonics. In *In Proceedings of the 13th Eurographics workshop on Rendering*. Eurographics Association, June 2002.

[76] Stavridi M. Koenderink J.J., Doorn A.J. van. Bidirectional reflection distribution function expressed in terms of surface scattering modes. *Computer Vision - ECCV '96*, 1996.

[77] Jens Krueger and Ruediger Westermann. Linear algebra operators for gpu implementation of numerical algorithms. *ACM Transactions on Graphics (TOG)*, 22(3):908–916, 2003.

[78] E.P.F. Lafortune, S.C. Foo, K.E. Torrance, and D.P. Greenberg. Non-linear approximation of reflectance functions. In Turner Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 117–126. ACM SIGGRAPH, Addison Wesley, August 1997.

[79] Paul Lalonde and Alain Fournier. Filtered local shading in the wavelet domain. In Julie Dorsey and Phillipp Slusallek, editors, *Rendering Techniques '97 (Proceedings of the Eighth Eurographics Workshop on Rendering)*, pages 163–174, New York, NY, 1997. Springer Wien.

[80] Hayden Landis. Production ready global illumination. In *Siggraph 2002 Course Notes:*, pages 331–338, 2002.

[81] G.Ward Larson. Measuring and modeling anisotropic reflection. In *ACM SIGGRAPH*, page 265272, July 1992.

[82] B. K. Lee and Fred M. Richards. The interpretation of protein structures: estimation of static accessibility. *J Mol Biol.*, 55(3):379–400, Feb 1971.

[83] Hendrik P. A. Lensch, Jan Kautz, Michael Goesele, Wolfgang Heidrich, and Hans-Peter Seidel. Image-based reconstruction of spatial appearance and geometric detail. *ACM Transaction on Graphics*, 22(2):234–257, April 2003.

[84] H.P. Lensch, W. Heidrich, and H.P. Seidel. Automated texture registration and stitching for real world models. In *Proc. 8th Pacific Graphics 2000 Conf. on Computer Graphics and Application*, pages 317–327, Los Alamitos, CA, 2000. IEEE.

[85] P. A. Lensch, Jan Kautz, Michael Goesele, Wolfgang Heidrich, and Hans-Peter Seidel. Image-based reconstruction of spatially varying materials. In *Rendering Techniques 2001 (Proc. 12th Eurographics WS on Rendering*, pages 104–115, London(UK), June, 25-27 2001. Springer, Wien.

[86] Joshua Levenberg. Fast view-dependent level-of-detail rendering using cached geometry. In *VIS '02: Proceedings of the conference on Visualization '02*, pages 259–266, Washington, DC, USA, 2002. IEEE Computer Society.

[87] M. Levoy and P. Hanrahan. Light field rendering. In Holly Rushmeier, editor, *SIGGRAPH 96 Conf. Proc.*, Annual Conf. Series, pages 31–42. ACM SIGGRAPH, Addison Wesley, August 1996.

[88] Marc Levoy, Kari Pulli, Brian Curless, Szymon Rusinkiewicz, David Koller, Lucas Pereira, Matt Ginzton, Sean Anderson, James Davis, Jeremy Ginsberg, Jonathan Shade, and Duane Fulk. The digital michelangelo project: 3D scanning of large statues. In Kurt Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, Annual Conference Series, pages 131–144. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.

[89] Numerical Design Limited. GameBryo game engine. www.ndl.com, 2005.

[90] P. Lindstrom. Out-of-core simplification of large polygonal models. In *Comp. Graph. Proc., Annual Conf. Series (SIGGRAPH 2000), ACM Press*, pages 259–262. Addison Wesley, July 22-28 2000.

[91] Peter Lindstrom. Out-of-core construction and visualization of multiresolution surfaces. In *SI3D '03: Proceedings of the 2003 symposium on Interactive 3D graphics*, pages 93–102, New York, NY, USA, 2003. ACM Press.

[92] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. In *ACM Computer Graphics (SIGGRAPH 87 Proceedings)*, volume 21, pages 163–170, 1987.

[93] D. Luebke and C. Erikson. View-dependent simplification of arbitrary polygonal environments. In *ACM Computer Graphics Proc., Annual Conference Series, (SIGGRAPH 97)*, pages 199–208, 1997.

[94] M. Fabbri A. Fasano C. Montani P. Pingi N. Santopuoli R. Scopigno F. Uccelli A. Varone M. Balzani, M. Callieri. Digital representation and multimodal presentation of archeological graffiti at pompei. In *VAST 2004*, pages 93–104, Bruxelles, 2004.

[95] R. Scopigno G. Gori M. Risaliti M. Callieri, P. Cignoni. Beyond manual drafting: a restoration-oriented system. *Journal of Culturel heritage*, 2006, to appear.

[96] DALSA Vision For Machines. 12 bit hdr camera. http://vfm.dalsa.com/products/areascan.asp.

[97] William R. Mark. Cg: A system for programming graphics hardware in a c-like language. *ACM Transactions on Graphics (TOG)*, 22(3), 2003.

[98] S.R. Marshner. *Inverse rendering for computer graphics*. PhD thesis, Cornelle University, 1998.

[99] David Kirk McAllister. *A Generalized Surface Appearance Representation For Computer Graphics*. PhD thesis, Chapel Hill, 2002.

[100] C. S. McCarmy, H. Marcus, and J. G. Davidson. A color-rendition chart. *Journal of applied Photographic Engineering*, pages 95–99, 1976.

[101] Gavin Miller. Efficient algorithms for local and global accessibility shading. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 319–326, New York, NY, USA, 1994. ACM Press.

[102] P. J. Neugebauer. Geometrical cloning of 3d objects via simultaneous registration of multiple range images. In *SMA '97: Proceedings of the 1997 International Conference on Shape Modeling and Applications (SMA '97)*, page 130, Washington, DC, USA, 1997. IEEE Computer Society.

[103] P.J. Neugebauer and K. Klein. Texturing 3d models of real worls objects from multiple unregistered photographic views. *Computer Graphics Forum (Eurographics'99 Proc.)*, 18(3):245–255, 1999.

[104] et al P. Shirley, H. Hu. A practitioners' assessment of light reflection models. In *Pacific Graphics*, June 1997.

[105] J. Pair, U. Neumann, D. Piepol, and W. Swartout. FlatWorld: Combining hollywood set-design techniques with VR. *IEEE Computer Graphics and Applications*, 23, January/February 2003.

[106] J. Pair and D. Piepol. FlatWorld: A mixed reality environment for education and training. In *International Conference on Information Systems, Analysis and Synthesis*. SCI/ISAS, 2002.

[107] V. Pascucci and R. Frank. *Hierarchical and Geometrical Methods in Scientific Visualization*, chapter Hierarchical Indexing for Out-of-Core Access to Multi-Resolution Data, pages 225–241. Springer, 2002.

[108] Alex Pentland and Stan Sclaroff. Closed-form solutions for physically based shape modeling and recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 13(7):715–729, 1991.

[109] Matt Pharr. *GPU Gems*, chapter Ambient occlusion, page 667692. Addison-Wesley, 2004.

[110] B. Phong. Illumination for computer generated images. *Communications of the ACM*, 6(18):311–317, June 1975.

[111] Bui Tuong Phong. Illumination for computer generated pictures. *Commun. ACM*, 18(6):311–317, 1975.

[112] Alex A. Pomeranz. Roam using surface triangle clusters (rustic). Master's thesis, Center for Image Processing and Integrated Computing, University of California, Davis, 2000.

[113] Chris Prince. Progressive meshes for large models of arbitrary topology. Master's thesis, Department of Computer Science and Engineering, University of Washington, Seattle, August 2000.

[114] K. Pulli. Multiview registration for large datasets. In *Proc 2nd Int.l Conf. on 3D Digital Imaging and Modeling*, pages 160–168. IEEE, 1999.

[115] Timothy J. Purcell, Ian Buck, William R. Mark, and Pat Hanrahan. Ray tracing on programmable graphics hardware. *ACM Transactions on Graphics*, 21(3):703–712, July 2002. ISSN 0730-0301 (Proceedings of ACM SIGGRAPH 2002).

[116] et K. E. Torrance R. L. Cook. A reflectance model for computer graphics. In *ACM SIGGRAPH*, volume 15, pages 301–316, July 1981.

[117] J. Koenderink R. Lu and A. Kappers. Optical properties (bidirectional reflectance distribution functions) of velvet. *Applied Optics*, (37(25)):59745984, September 1998.

[118] R. Ramamoorthi and P. Hanrahan. An Efficient Representation for Irradiance Environment Maps. In *Computer Graphics (ACM SIGGRAPH '01 Proceedings)*, 2001.

[119] R. RAMAMOORTHI and P. HANRAHAN. A signal-processing framework for inverse rendering. page 117128, 2001. ISBN 1-58113-292-1 (Proceedings of ACM SIGGRAPH 2001).

[120] V. Rankov, R. Locke, R. Edens, P. Barber, and B. Vojnovic. An algorithm for image stitching and blending. In *Proceedings of SPIE. Three-Dimensional and Multidimensional Microscopy: Image Acquisition and Processing XII*, March 2005.

[121] P. Reilly. Towards a virtual archaeology. In *Computer Applications in Archaeology*. British Archaeological reports (Int. Series 565), 1990.

[122] C. Rocchini, P. Cignoni, C. Montani, and R. Scopigno. Acquiring, stitching and blending diffuse appearance attributes on 3d models. *The Visual Computer*, 18(3):186–204, 2002.

[123] I. Roussos and A. Chalmers. High fidelity lighting of Knossos. In A. Chalmers D. Arnold and F. Niccolucci, editors, *VAST 2003*, pages 195–201, Bighton, UK, Nov. 5-7 2003. Eurographics.

[124] H. Rushmeier, G. Taubin, and A. Gueziec. Applying shape from lighting variation to bump map capture. In P. Slusallek J. Dorsey, editor, *Eurographics Rendering Workshop 1997*, pages 35–44. Springer Wien, June 1997.

[125] 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.

[126] R. Szelinski S. Gortler, R. Grzeszczuk and M. Cohen. The lumigraph. In *ACM SIGGRAPH*, pages 43–54, August 1996.

[127] E. Lafortune K. Torrance S. Marschner, S.Westin and D. Greenberg. Image-based brdf measurement including human skin. In *10th Eurographics Workshop on Rendering*, page 131144, June 1999.

[128] E. Salvador, A. Cavallaro, and T. Ebrahimi. Cast shadow segmentation using invariant color features. *Computer Vision and Image Understanding*, 95(2):238–259, August 2004.

[129] I. Sato, Y. Sato, and K. Ikeuchi. Illumination from shadows. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 25(3):290–300, 2003.

[130] Y. Sato, M.D. Wheeler, and K. Ikeuchi. Object shape and reflectance modeling from observation. In Turner Whitted, editor, *Comp. Graph. Proc., Annual Conf. Series (SIGGRAPH 97)*, pages 379–388. ACM SIGGRAPH, Aug. 1997.

[131] Mirko Sattler, Ralf Sarlette, Gabriel Zachmann, and Reinhard Klein. Hardware-accelerated ambient occlusion computation. In *VMV*, pages 331–338, 2004.

[132] Peter Schröder and Wim Sweldens. Spherical wavelets: efficiently representing functions on the sphere. *Computer Graphics*, 29(Annual Conference Series):161–172, 1995.

[133] Le-Jeng Shiue, Ian Jones, and Jörg Peters. A realtime gpu subdivision kernel. *ACM Trans. Graph.*, 24(3):1010–1015, 2005.

[134] C. Silva, Y. Chiang, J. El-Sana, and P. Lindstrom. Out-of-core algorithms for scientific visualization and computer graphics. In *Visualization'02*, 2002. Course Notes for Tutorial n.4.

[135] A.J. Stoddart and A. Hilton. Registration of multiple point sets. In *ICPR96*, page B6A.5, 1996.

[136] J. Stumpfel, A. Jones, A. Wenger, C. Tchou, T. Hawkins, and P. Debevec. Direct HDR capture of the sun and sky. In *In Proceedings of the AFRIGRAPH*, 2004.

[137] J. Stumpfel, C. Tchou, T. Hawkins, P. Martinez, B. Emerson, M. Brownlow, A. Jones, N. Yun, and P. Debevec. Digital reunification of the Parthenon and its sculptures. In *4th International Symposium on Virtual Reality, Archaeology and Intelligent Cultural Heritage*. VAST, 2003.

[138] M. Subbarao, T. Chio, and A. Nikzad. Focusing techniques. *Optical Engineering*, 32(11):2824–2836, 1993.

[139] INUS Technology. Rapidform^{TM} 2004. More info on: `http://www.rapidform.com/index.htm`, 2003.

[140] Demetri Terzopoulos, Andrew Witkin, and Michael Kass. Energy constraints on deformable models: recovering shape and non-rigid motion. In *Proc. AAAI-87*, pages 755–760, 1987.

[141] K. Torrance and E. Sparrow. Theory for off-specular reflection from roughened surfaces. *Journal of Optical Society of America*, (57(9)), 1967.

[142] R. Tsai. A versatile camera calibration technique for high accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses. *IEEE Journal of Robotics and Automation*, RA-3(4), August 1987.

[143] G. Turk and M. Levoy. Zippered polygon meshes from range images. *ACM Computer Graphics*, 28(3):311–318, 1994.

[144] ARVO J. WESTIN, S. and K. TORRANCE. Predicting reflectance functions from complex surfaces. page 255264, 1992. (Proceedings of ACM SIGGRAPH 1992).

[145] J. Willmott, L. Wright, D. Arnold, and A. Day. Rendering of large and complex urban environments for real time heritage reconstructions. In *International Symposium on Virtual Reality, Archaeology and Intelligent Cultural Heritage*, 2001.

[146] J. Wu and L. Kobbelt. A stream algorithm for the decimation of massive meshes, 2003.

[147] J.C. Xia and A. Varshney. Dynamic view-dependent simplification for polygonal models. In R. Yagel and G. Nielson, editors, *IEEE Visualization '96 Proc.*, pages 327–334, 1996.

[148] Hitoshi Yamauchi, Jörg Haber, and Hans-Peter Seidel. Image restoration using multiresolution texture synthesis and image inpainting. In *Computer Graphics International (CGI 2003)*, pages 120–125, Tokyo, Japan, July 2003. IEEE.

[149] Y. Yu, P. Debevec, J. Malik, and T. Hawkins. Inverse global illumination: recovering reflectance models of real scenes from photographs. In A. Rockwood, editor, *SIGGRAPH 99 Conf. Proc.*, Annual Conf. Series, pages 215–224. ACM SIGGRAPH, Addison Wesley, July 1999.

[150] Z. Zhang. Iterative point matching for registration of free-form curves and surfaces. *International Journal of Computer Vision*, 13:119–152, 1994.

[151] Kun Zhou, Xi Wang, Yiying Tong, Mathieu Desbrun, Baining Guo, and Heung-Yeung Shum. Texturemontage: Seamless texturing of arbitrary surfaces from multiple images. *ACM Trans. Graph.*, 24(3):1148–1155, 2005.

[152] Sergej Zhukov, Andrej Inoes, and Grigorij Kronin. An ambient light illumination model. In George Drettakis and Nelson Max, editors, *Rendering Techniques '98*, Eurographics, pages 45–56. Springer-Verlag Wien New York, 1998.