Modeling and Rendering of Real Environments

Wagner T. Corrêa *

Manuel M. Oliveira †

Cláudio T. Silva ‡

Jianning Wang §

Abstract

The use of detailed geometric models is a critical factor for achieving realism in most computer graphics applications. In the past few years, we have observed an increasing demand for faithful representations of real scenes, primarily driven by applications whose goal is to create extremely realistic experiences by building virtual replicas of real environments. Potential uses of this technology include entertainment, training and simulation, special effects, forensic analysis, and remote walkthroughs. Creating models of real scenes is, however, a complex task for which the use of traditional modeling techniques is inappropriate. Aiming to simplify the modeling and rendering tasks, several image-based techniques have been proposed in recent years. Among these, the combined use of laser rangefinders and color images appears as one the most promising approaches due to its relative independence of the sampled geometry and short acquisition time. Renderings of scenes modeled with such a technique can potentially exhibit an unprecedented degree of photorealism. But before one can actually render new views of these virtualized environments, several challenges need to be addressed. This tutorial provides an overview of the main issues associated with the modeling and rendering of real environments sampled with laser rangefinders, and discusses the main techniques used to address these challenges.

Keywords: 3D scanning, 3D model acquisition, 3D photography

*Department of Computer Science, Princeton University, 35 Olden Street, Princeton, NJ 08540, e-mail: wtcorrea@cs.princeton.edu

†Instituto de Informática, UFRGS, Caixa Postal 15064, Porto Alegre, RS 91501-970, Brasil, e-mail: oliveira@inf.ufrgs.br

‡Department of Computer Science and Engineering, OGI School of Science and Engineering, Oregon Health & Science University, 20000 NW Walker Rd., Beaverton, OR 97006, e-mail: csilva@cse.ogi.edu

§Department of Computer Science, State University of New York at Stony Brook, Stony Brook, NY 11794-4400, USA, e-mail: jianning@cs.sunysb.edu

Resumo

O uso de modelos geométricos detalhados é um fator crítico para obtenção de realismo na maioria das aplicações em computação gráfica. Recentemente, tem-se observado uma demanda crescente por representações fidedignas de cenas reais. Isso se deve, principalmente, ao interesse por aplicações que objetivam a criação de experiências extremamente realísticas através da construção de réplicas virtuais de ambientes reais. Exemplos de tais aplicações incluem entretenimento, treinamento e simulação, efeitos especiais, análise forênsica, e exploração de ambientes remotos. Entretanto, a criação de modelos de ambientes reais é uma tarefa não trivial para a qual o uso de técnicas de modelagem tradicionais é inapropriado. Objetivando simplificar o processo de modelagem e rendering nesses casos, várias técnicas baseadas no uso de imagens foram propostas nos últimos anos. Entre essas, o uso combinado de digitalizadores 3D a base de laser e de fotografias coloridas tem se mostrado como uma das abordagens mais promissoras devido à sua relativa independência em relação às superficies a serem amostradas e à velocidade do processo de amostragem. A visualização de cenas reconstruídas utilizando essa técnica permite, teoricamente, a obteção de um grau de fotorealismo sem precedentes. Mas antes que se possa explorar esses ambientes virtuais, várias etapas precisam ser observadas e vários desafios superados. Este tutorial apresenta os principais aspectos relacionados à modelagem e rendering de ambientes reais amostrados por meio de digitalizadores 3D a base de laser e discute as principais técnicas utilizadas.

1 Introduction

Models of real-world objects and scenes are becoming fundamental components of modern computer graphics applications, playing an important role in helping to create realistic virtual experiences. Applications such as entertainment, training and simulation, special effects, analysis of forensic records, telepresence, and remote walkthroughs can greatly benefit from the availability of such models. However, the modeling of real scenes is a non-trivial task for which the use of traditional modeling techniques is inappropriate. Trying to overcome this problem, several image-based techniques have been proposed in recent years [7, 17, 30, 55, 71, 72, 88]. Among these, the use of laser rangefinders combined with color photographs seems to be one the most promising approaches due to its relative independence of the sampled geometry and short acquisition time. Renderings of scenes reconstructed using this approach can potentially display an unparalleled degree of photorealism.

Sampling a real environment is just the beginning of the scene reconstruction process, and before one can explore these virtualized spaces, several challenges need to be addressed. For example, to cover the entire environment, multiple datasets are usually acquired from different viewpoints, and need to be integrated. Measurements produced by laser rangefinders are inherently noisy, and some preprocessing is required to “clean” the data. Range and color information are often captured using different devices, and must be registered. The output produced by the previous stages is still a point cloud, and surface reconstruction is

required if the models are to be displayed as polygonal meshes. In many situations, despite data being collected from several viewpoints, it may still not be possible, or practical, to guarantee appropriate sampling for all surfaces: occlusions and accessibility limitations to certain regions of the scene may cause some areas not to be sampled, resulting in incomplete models. In these cases, it is desirable to reconstruct the missing geometry and texture from the incomplete data available. Another challenge involves the management of large amounts of samples. Datasets with several hundred megabytes in size are common, and out-of-core, simplification, and parallelization techniques are used to achieve interactive renderings. The ability to edit the resulting scenes is also highly desirable. Ideally, one should be able to manipulate individual objects, positioning, scaling, saving, loading, and combining them to create new scenes, as if they were conventional 3D models.

This tutorial discusses the major issues associated with the modeling and rendering of real environments sampled using laser rangefinders, and discusses the main techniques used to address the many associated challenges. It starts by describing the steps involved in 3D model acquisition pipeline, which includes range acquisition (Section 2), texture acquisition (Section 3), and reconstruction of non-sampled areas (Section 4). Scene editing is also discussed in Section 4. Section 5 presents rendering strategies for handling massive amounts of sample data. Section 6 discusses some of the research challenges ahead: the issues of inconsistent illumination among sets of photographs, lighting changes, and reconstruction of non-stochastic textures. Finally, Section 7 summarizes the key ideas discussed in this survey.

2 Range Acquisition

In this section, we describe the range acquisition phase of the model acquisition pipeline. First, we give a brief overview of range acquisition methods. Then, we describe how to register multiple range datasets in a common coordinate system. Finally, we describe how to filter the acquisition artifacts in the range datasets.

2.1 Range Acquisition Methods

There are many different methods for range acquisition (Figure 1). Curless [29] and Rusinkiewicz [91, 92] classify these methods into three main categories: contact, transmissive, and reflective methods.

Contact methods touch the surface of the object with a probe, and record the position. Contact methods, e.g., coordinate measuring machines (CMMs), can be very accurate, but they are slow, can be clumsy to manipulate, usually require a human operator, and must make contact with the surface, which may be undesirable for fragile objects [29].

Transmissive methods project energy waves onto an object, and record the transmitted energy. For example, industrial computer tomography (CT) projects x-rays onto an object, and measures the amount of radiation that passes through it. The result is a high resolution volumetric model of the object. Transmissive methods are largely insensitive to the reflective

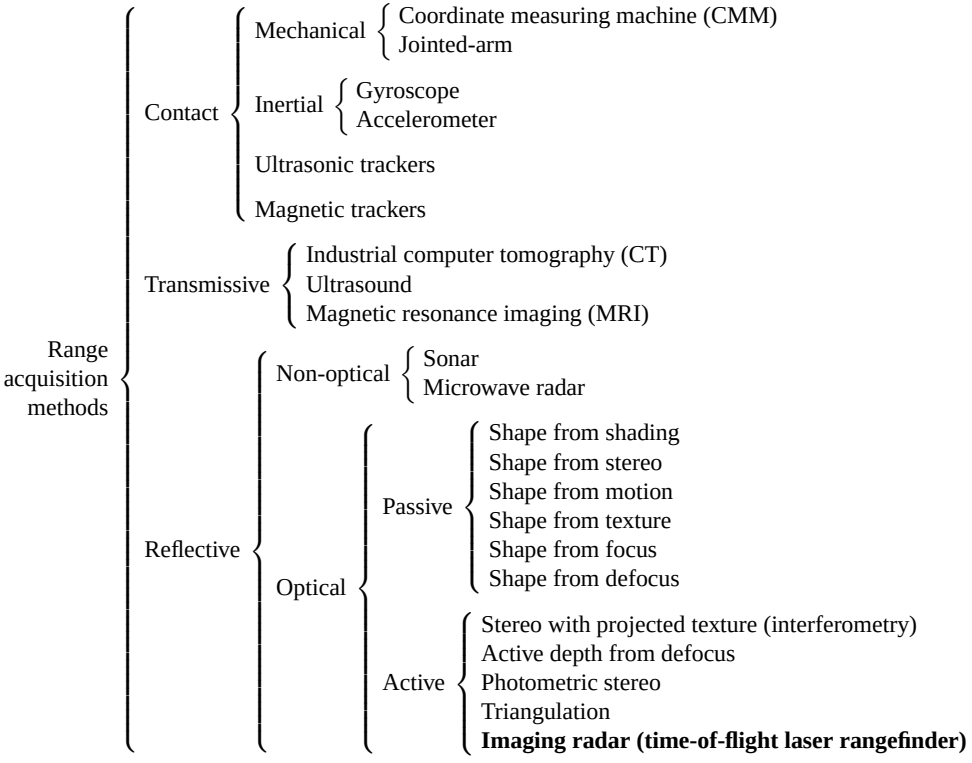


Figure 1: Classification of range acquisition methods. Adapted from Curless [29] and Rusinkiewicz [92]. In this tutorial, we focus on optical time of flight methods.

properties of the object, and they can capture details not visible from the outside. On the other hand, transmissive methods are expensive, sensitive to large variation in material density, and pontentially hazardous [29].

Reflective methods may be non-optical or optical. Non-optical methods include sonar and microwave radar, which measure the distance to an object by sending a pulse of sound or microwave energy to the object, and recording the the time it takes for the pulse to bounce back from the object. Sonars are typically inexpensive, but they are neither very accurate nor very fast, and microwave radars are typically used for long range remote sensing.

Optical methods may be passive or active. Passive methods do not interact with the object being scanned, and include computer vision techniques such as shape-from-shading for single images, stereo triangulation for pairs of images, and optical flow and factorization methods for video streams. Although passive methods require little special purpose hardware, they rarely construct accurate models, and their main application is object recognition [29, 92].

Active optical methods project light onto an object in a structured manner, and determine the object's shape from the reflections. Active optical methods include depth from defocus [79], photometric stereo [90], projected-light triangulation [47, 91], and time of flight [1, 2]. Active methods have several advantages over passive methods: active methods perform better in the absence of texture, are computationally inexpensive and robust, and produce dense and accurate sets of range samples [91]. In this tutorial, we focus on time-of-flight laser range scanning.

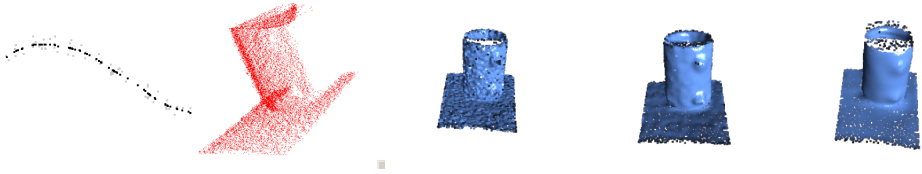
2.2 Registration of Multiple Range Datasets

A single scan usually contains holes due to occlusion, and samples near and far objects at different resolutions. Thus, to get a more complete model, or to obtain a more uniform resolution, we need to scan the environment from multiple locations. Automatically determining the best set of locations for scanning is a hard problem [38, 85]. The simplest approach is to select the scanning locations manually, trying to minimize the number of scans necessary for a good coverage of the environment, and making sure that there is some overlap between the scans [24]. After we have a set of scans, we need to align them in a common coordinate system. Most approaches first find an initial global alignment, and then refine it.

There are several approaches for finding an initial alignment between a pair of scans. High-end systems, such as coordinate measuring machines (CMM), employ accurate tracking, but these systems are often very expensive. Some low-end systems find the initial alignment by performing the scanning using a turntable, which tends to constrain the use of this approach to small objects. Other low-end systems [24] rely on a human operator, who uses an interactive tool to select matching features on each scan. From the matching features, a rigid transformation is computed that aligns the two scans [52]. Automatic feature detection, although desirable, is a hard problem, and is currently an active area of research [27, 44, 54, 89, 127].

The initial global alignment is typically not very accurate, and needs to be refined. The most common approach to refine the initial alignment is the iterative closest point (ICP) algorithm and its variants [14, 18, 94, 128]. The ICP algorithm consists of two main steps. Given two sets of points, the first step is to identify pairs of candidates corresponding points, and the second step is to compute a transformation that minimizes the distance between the two sets of candidate points in the least-squares sense. These steps are repeated until some convergence criterion is met.

ICP typically provides very accurate alignments, and converges fast, but it may get stuck in local minima. Another potential problem with ICP is that sequential alignment of multiple scans may suffer from accumulation of pair-wise errors. To avoid this problem, some systems use anchor scans, and align additional scans to the anchor scans. Other systems employ error diffusion to evenly distribute the error among the scans. Bernardini and Rushmeier [13] review several approaches to avoid accumulation of pair-wise errors.



(a) (b) (c) (d) (e)

Figure 2: The noise removal process. (a) In 2D, a moving least squares curve (black) [6] is computed from noisy sample points (gray). (b) Noisy scan of a 3D object. (c)-(e) Progressively smoother versions of (b). Images courtesy of Shachar Fleishman, Tel Aviv University.

2.3 Range Data Filtering

The raw data provided by a 3D scanner may contain several artifacts, including noise, difference in resolution between near and far surfaces, and distortions due to intensity bias. Many researchers have studied the artifacts related to triangulation scanners [49, 80, 95, 103, 110]. Here we focus on artifacts related to optical time-of-light scanners [2, 24].

A typical time-of-flight scanner, such as the DeltaSphere-3000 [2], returns a set of points, each point consisting of its spherical coordinates (r, θ, ϕ) and the intensity i of the energy returned from that point. We thus use the term "*rtpi* sample" to refer to a point sample captured by such a scanner.

One artifact related to optical time-of-flight scanners is intensity bias: points with too low or too high intensity tend to have unreliable radius. One approach to minimize the intensity bias of the scanner is to scan a calibration pattern, and build a correction table. Corrêa et al. [24] use a calibration image that changes linearly from black to white, and is surrounded by a white background. The image is placed on a flat surface, and then scanned. From the scan, a bias correction table that is used for correcting subsequent scans is built. The noise distribution in the scan is assumed to be of zero mean. The bias in the scan is in the r part of an rtpi sample, not in ϕ or θ. The background is considered to be the ground truth, and the best (in the least squares sense) plane that fits the background points is found. A bias is computed for every intensity value $i \in [0, 255]$ as the average height $\overline{H}_i$ (relative to the plane) of the sampled points with intensity i. After building the bias correction table, the radius r of every point is corrected by computing $r' = r - \overline{H}_i$.

Another acquisition artifact is noise, which manifest itself as random errors in the position of surface points (Figure 2). One approach to deal with noise is to average samples from overlapping scans [13]. Another approach is to resample the points based on a local estimate of the underlying surface. For example, Alexa et al. [6] and Corrêa et al. [24] apply the *moving least squares (MLS) projection* of Levin [63] to filter the noisy scans.

3 Texture Acquisition

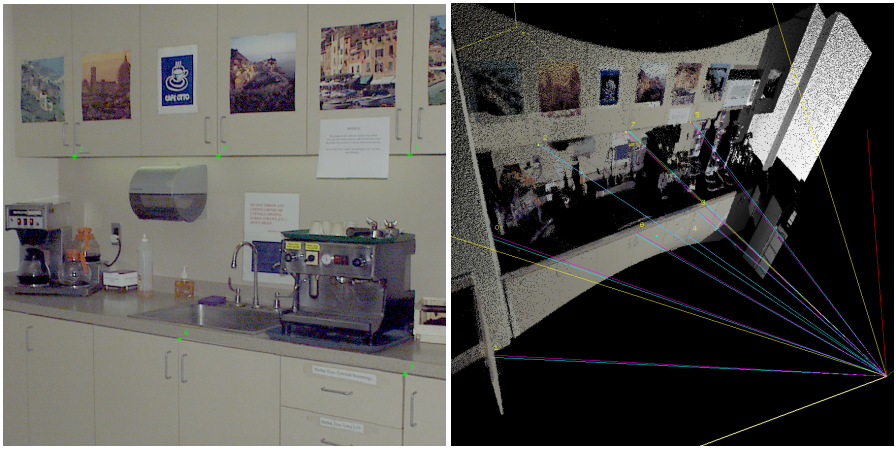
Section 2 described the acquisition of the geometric properties of a model. This section focus on object texture (colors) acquistion. Ideally, we would like to have a complete description of how a surface point reflects light depending on its normal, the incident light direction, the emerging light direction, and the light's wavelength. Such a description is known as the bidirectional reflectance distribution function (BRDF) of a surface. Measuring BRDFs accurately is a hard problem [60, 125]. Here we describe a simple approach that captures sparse samples of BRDFs, suitable for diffuse environments [24, 71].

The simplest approach for acquiring the texture of an environment is to take pictures of it, and then map these photographs onto the previously acquired geometry. To map a photograph to the geometry, we need to know the camera projection parameters (intrinsic parameters), and the position and orientation of the camera when the photograph was taken (extrinsic parameters). Real cameras do not perform a perfect perspective projection as a pinhole camera does, and present many kinds of distortions, e.g., radial distorsion. One solution to this problem is to model the action of the distorsions, and to find an inverse mapping. A widely used model was proposed by Tsai [108]. Tsai's camera model has five intrinsic parameters, namely the focal length, the first-order coefficient of radial distortion, the coordinates of the image center, and a scale factor. One way to calibrate these intrinsic parameters is to take a photograph of a planar checkerboard pattern with known geometry, and then find the image location of the checkerboard corners with subpixel accuracy. Implementing these algorithms, which require full non-linear optimization, is fairly complex. Luckily, high quality implementations are available from Willson [121] and Bouguet [15].

After calibrating the intrinsic camera parameters, the process of acquiring the images goes as follows. First, we take photographs of the environment, keeping the same camera settings for all photographs to avoid having to recalibrate the intrinsic parameters. Then, for each photograph, we first remove its radial distortion, using a warp based on the coefficient found above, and then find the position (translation) and orientation (rotation) of the camera when we took the photograph relative to an arbitrary global coordinate system.

It is hard to automatically solve the image-geometry registration problem. By specifying pairs of corresponding points between the images and the geometry, it is possible to find the extrinsic camera parameters [121]. The approach taken by McAllister et al. [71] is to keep the center of projection (COP) of the camera coincident with the COP of the 3D scanner, while acquiring panoramic images. This simplifies the registration problem by only requiring the computation of a rotation. Furthermore, this enforces that no occlusion artifacts arise. Corrêa et al. [24] uses an interactive program to specify corresponding points, typically requiring only 7 to 10 correspondences to obtain a good calibration of the camera. One advantage of this approach is that it allows the user to take pictures from any position.

Once all the parameters have been found, it is straightforward to map the colors from the photograph to the scan (or scans) that it covers. For each 3D point in a scan covered by the photograph, we find its 2D projection on the camera plane, and assign the corresponding pixel



(a) photograph (b) scan

Figure 3: The texture-geometry registration process.

color to it (Figure 3). To support view-dependent effects (such as highlights), we can store multiple color samples per point, and at runtime find the color of a point by interpolating the closest color samples [31].

4 Reconstruction of Non-Sampled Areas and Scene Editing

Even after a scene has been scanned from several viewpoints, it may still not be possible, or practical, to guarantee a complete coverage of all surfaces. Occlusions and scanner accessibility limitations to certain areas of the environment may lead to incomplete or incorrectly reconstructed models. Figure 4a shows the rendering of a real environment created using 1,805,139 samples from ten range images. Note the missing portions of the carpet, chair, and several pieces of furniture. These problems result from the limited vertical field of view used during the scanning of the scene and from occlusions among objects. For this example, the scene is being observed from a viewpoint chosen to stress the exposure of non-sampled areas. The existence of holes introduces major rendering artifacts and creating high-quality reconstructions from incomplete data is a challenging task [126]. Arbitrary geometric relationships among objects in a scene make the automatic computation of viewpoints that guarantees appropriate sampling of all surfaces essentially impossible. Solving this problem would require knowing the scene model, which is what the reconstruction process is trying to produce. As a result, the decision of where to position the scanner is usually left to the user.

(a) original scans (b) reconstructed model

Figure 4: A real environment model with 1.8 million samples from 10 range images.

Given the difficulty to avoid the occurrence of undesirable holes in the reconstructed models, one is left with two alternatives: keep acquiring more scans or try to reconstruct the missing information using the incomplete data available. Neither alternative alone is completely satisfactory in general. Holes caused by accessibility limitations cannot be filled with the acquisition of new scans, and the ability to perform reconstruction from incomplete data is highly dependent on the input data. Ideally, both approaches should be used in combination. Since acquiring and merging new scans is a relatively straightforward task, we will focus on reconstruction of missing information from incomplete data.

4.1 The Reconstruction Pipeline

Wang and Oliveira [116] proposed a pipeline for improving scene reconstruction that can significantly reduce artifacts caused by missing data. It consists of a segmentation step followed by the reconstruction of absent geometric and textural information (Figure 5). As output, the pipeline produces a modified scene model with most of the original holes removed. Figure 4b illustrates the reconstruction produced by the method for the same input samples used to create Figure 4a. Note the considerable improvement between the two renderings. The ability to reconstruct missing information requires some assumptions about the structure of the environment. Although scenes tend to differ significantly from one to another, two observations seem to hold true: first, indoor scenes usually contain a significant number of large planar surfaces, such as walls, ceiling, and floor; second, symmetry pervades both

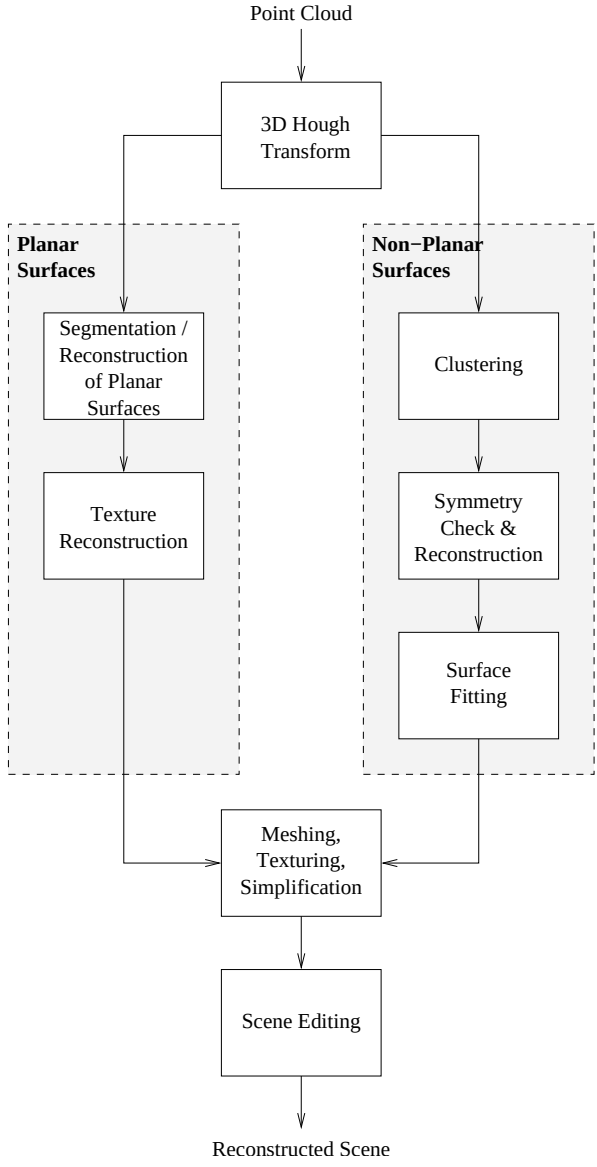


Figure 5: The segmentation and reconstruction pipeline

human-created and natural environments [48, 69, 119]. These observations were exploited in the design of new algorithms and tools that significantly simplify the reconstruction task. Since absent areas may possibly contain arbitrary geometry and texture, reconstruction from incomplete data is inherently ambiguous. Thus, although most tasks in the pipeline are performed automatically, they require some amount of user intervention. This is acceptable given the costs and difficulties still associated with the scanning of real environments.

Segmentation and Reconstruction of Planar Surfaces

Given the existence of several algorithms for registration of range, and range and color data [14, 81, 87], it is assumed that the data presented as input to the pipeline have already been registered (both range and color). The reconstruction process starts by dividing the original dataset into planar and non-planar surfaces using a 3D Hough transform [119]. The separation of the original dataset into two non-overlapping groups greatly simplifies the overall reconstruction task. For example, holes in planar surfaces can be easily restored. Also, by removing the samples associated with planar areas more expensive algorithms can be used to reconstruct the remaining surfaces (represented by smaller datasets). In order to avoid the undesired effect of finding spurious planes by the Hough transform, the normal of every point is computed before vote accumulation and each point only votes for a few Hough cells [116]. These cells are identified based on the normal at the point, which, in turn, is computed considering the point's neighborhood. The final parameters for the plane are obtained through a fitting process using singular value decomposition (SVD). Once large planar surfaces have been automatically identified using the procedure just described, the user is required to specify boundaries for the underlying planar areas. This is done interactively in 3D through the use of a graphical user interface. Note that the specification of such boundaries is often necessary due to the ambiguity introduced by the existence of missing data. For example, consider the small portion of the carpet visible in Figure 4a. It would not be possible to automatically recover the original boundaries of the carpet from the available information. The boundaries specified by the users define polygons that will replace the original samples. The texture associated with a planar surface is extracted by orthographically projecting the corresponding color samples onto the plane of the polygon. For the case of stochastic textures, a small sample is obtained and used as input for a texture synthesis algorithm [34]. The original samples are then discarded and the new synthesized texture is mapped onto the corresponding polygon. This procedure was used to reconstruct the carpet in Figure 4b. Although not identical, the new textures are statistically similar to the original ones. Since only the new textures are presented to the viewer, the actual differences between them are likely to pass unnoticed.

Replacing many thousands of samples with textured-mapped polygons allows for efficient rendering on current graphics hardware. 2D textures with symmetric patterns can also be reconstructed along the planar-surfaces branch of the pipeline. In this case, a 2D Hough transform applied to texel colors is used to identify axes of symmetry, and to mirror information from one side to another [116]. Remaining holes are filled using a push-pull strategy [43].

(a) original scan (b) symmetry plane (c) reconstructed model

Figure 6: Symmetry-based reconstruction.

Segmentation and Reconstruction of Non-Planar Surfaces

Samples corresponding to non-planar surfaces receive a different treatment. Since most objects rest on flat surfaces, once the large planar areas have been removed, it becomes relatively easier to identify and isolate individual objects. Thus, the non-planar branch of the reconstruction pipeline starts by clustering spatially close samples. This is accomplished using an incremental surface construction algorithm based on proximity of points in 3D [42]. Once clusters have been identified, each one is treated independently of the rest of the dataset. This is desirable since each cluster usually has a relatively small size when compared to the entire dataset and some of the algorithms used for cluster reconstruction run in polynomial time on the number of the input samples.

Each cluster undergoes a 3D symmetry check based on a variation of the 3D Hough transform [116]. The check consists of computing, for each pair of points in the cluster, its bisector plane (i.e., the plane perpendicular to the line defined by the two points and equidistant to both), which receives a vote. At the end, any plane with a significantly larger number of votes is elected as representative of approximate symmetry [116]. Depending on the amount of missing data and their distribution over the object's surface, it may not be possible to accurately recover the actual symmetry plane of the original object. In this case, user intervention is required to appropriately reposition the computed plane. User intervention is also required to select among several candidate planes with approximately the same number of votes. In the case of rotationally symmetric objects, the axis of revolution can be obtained as the intersection, in the least square sense, of all the candidates. In practice, however, rotationally symmetric objects are easier reconstructed by mirroring data across several of their symmetry planes. Figure 6a shows a chair automatically segmented from the dataset depicted in Figure 4a. Figure 6b shows the symmetry plane computed using the procedure just described. Note that, despite the large missing areas on the surface of the chair, the algorithm still manages to compute a good symmetry plane. This can be explained by the relative insensitivity

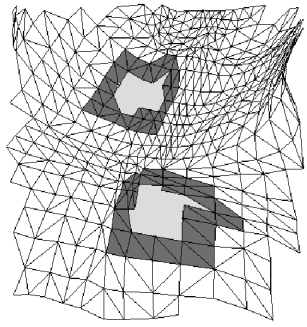


Figure 7: Identifying holes.

of the Hough transform to noisy and missing data. Finding an approximate symmetry plane does not require a consensus, but simply finding a plane with the largest number of votes. Once approximate symmetry has been identified, surface reconstruction proceeds by mirroring samples across the symmetry plane (axis) [116]. Figure 6c shows the reconstructed chair obtained from the original samples using symmetry-based reconstruction. The existence of small holes in the resulting model is due to the lack of samples in both sides of the symmetry plane. While such holes are relatively small and could have been filled during the meshing process, all surface reconstruction algorithms [9, 11, 33, 42, 51] assume that the surfaces to be restored have been appropriately sampled and, therefore, do not handle local variations in sampling density. The occurrence of such variations manifests itself as small holes visible in Figure 6c, and further processing is required to eliminate these artifacts.

Filling Small Holes by Surface Interpolation

In locally smooth surfaces, holes can be filled using surface interpolation methods. Wang and Oliveira [115] describe a procedure for automatic identification and filling of holes in point clouds. The method consists of first creating a triangle mesh using a surface construction algorithm. Holes are then automatically identified by analyzing the resulting mesh searching for cycles of non-shared edges (Figure 7). Note that since it is not possible to distinguish between an under-sampled region and a real hole in the surface, user interaction is required to guarantee proper reconstruction. Once a hole has been identified, the missing region can be interpolated from its neighborhood. Samples around the hole, called the interpolation context (see dark gray regions in Figure 7), are used to perform the interpolation. Whereas several surface fitting techniques can be used to reconstruct missing regions, for most of them, the resulting surfaces do not pass through the original samples, which tends to introduce discontinuities. This is the case, for instance, of the conventional least-square

method for surface fitting. Avoiding discontinuities between the original and filled areas is important in order to guarantee the quality of the reconstructed models. The hole filling strategy proposed by Wang and Oliveira [115] consists of adding new samples to under-sampled regions by resampling an interpolated surface reconstructed using moving least squares [59], a variation of the original least squares method that guarantees the interpolation of the original samples. An important requirement during the introduction of new samples is to enforce that the sampling density inside the hole matches the sample density of the interpolation context. This is necessary to guarantee that the surface reconstruction algorithm used will not just leave even smaller holes.

Surface fitting procedures treat the surfaces to be reconstructed as height fields (i.e., as function of the type $z = f(x,y)$) [59]. Thus, for each hole, reconstruction and re-sampling are performed at its "local tangent plane". Let N be the number of samples in the interpolation context c of a hole h, and let O be the point obtained by averaging the 3D coordinates of all samples in c. Also, let M be an $Nx3$ matrix obtained by subtracting O from all samples in c. A local coordinate system for the tangent plane can be obtained by factoring M using singular value decomposition. Such a factorization produces an orthonormal basis corresponding to the eigenvectors of MM^T [104]. The two eigenvectors corresponding to the two eigenvectors with largest absolute values span the desired tangent plane, while the third one is perpendicular to the plane. Once the plane has been computed, the samples belonging to the corresponding interpolation context are projected onto it. The problem of deciding the re-sampling positions inside the hole is then reduced to the 2D problem of finding a set of (x,y) positions inside the projection of the hole that preserves the sampling density of the projection of the interpolation context. These positions are computed, and the new samples are obtained by re-sampling the interpolated surface at these points. Once these samples have been added, a new triangle mesh is created for the object using the surface reconstruction algorithm described in [42].

In addition to reconstructing geometry, it is also necessary to reconstruct color and texture. The moving-least-squares procedure can also be used for the reconstruction of smoothly varying colors. This can be achieved simply by replacing height (the z coordinate measured with respect to the plane) with the red, green and blue color channels, one at a time. Figure 8 shows a scene displaying the chair model in three different stages of the reconstruction pipeline. The original samples are shown on the left; the image in the middle depicts the chair after symmetry-based reconstruction; on the right, one sees the result after the hole-filling procedure has been applied. A significant improvement can be observed when comparing the original and the final models.

Scene Editing

The segmentation of individual objects supported by the pipeline depicted in Figure 5 provides the users with the ability to edit the original environments. This is an extremely valuable feature, since it allows the acquired scenes to be composited in many different ways.

(a) original samples

(b) after symmetry-based reconstruction

(c) after hole filling

Figure 8: Edited scene showing a chair in different stages of the reconstruction pipeline.

Like conventional 3D geometric models, these reconstructed objects can be translated, rotated, scaled, and instantiated arbitrarily. In the reconstruction shown in Figure 4b, the chair in Figure 6c was instantiated three times at different positions and orientations, replacing the original ones. In Figure 8, the original chairs were replaced by a single instance of the reconstructed chair, and the carpet was replaced with a wooden-textured floor. The bottom shelves and their contents were edited using some imaging tools available in the system described by Wang and Oliveira [116].

4.2 Other Reconstruction Approaches

An alternative for filling holes as part of surface reconstruction from point clouds has been proposed by Carr et al. [16], and consists of using polyharmonic radial basis functions to obtain implicit representations for object surfaces. Such a technique handles large numbers of samples, produces very impressive results, and can also be used for mesh simplification and re-meshing. Mesh simplification is a highly desirable feature, since it can be used to significantly reduce the amount of primitives used to render the model while preserving its original appearance. One drawback of this approach is that the entire point cloud (extended with some off-surface points [16]) represents a single surface. As a result, the entire scene is treated as a single surface, which precludes scene editing. Handling individual objects would require the use of a segmentation procedure (not part of the original algorithm) in order to isolate the objects. This approach does not handle texture reconstruction, and would need to be combined with texture synthesis techniques.

Another approach for model reconstruction of indoor scenes sampled by range images has been proposed by Whitaker et al. [120]. In their system, the user defines correspon-

dences among planar surfaces visible in the several range images, which are used to guide the registration process. The integration of the range images is performed using a probabilistic optimization procedure that deforms the original range surfaces in order to improve the fitting. Except for requiring the user to define the correspondences, the process is completely automatic. Like in the approach proposed by Carr et al., the entire scene is modeled as a single surface, and no support for scene editing is provided. In order to handle noisy datasets, the input data undergo some smoothing pre-processing, causing all sharp edges and corners to be rounded in the reconstructed model.

5 Rendering

5.1 Interactive Exploration of Large Environments

Researchers have studied the problem of rendering complex models at interactive frame rates for many years. Clark [22] proposed many of the techniques for rendering complex models used today, including the use of hierarchical spatial data structures, level-of-detail (LOD) management, hierarchical view-frustum and occlusion culling, and working-set management (geometry caching). Garlick et al. [41] presented the idea of exploiting multiprocessor graphics workstations to overlap visibility computations with rendering. Airey et al. [4] described a system that combined LOD management with the idea of precomputing visibility information for models made of axis-aligned polygons. Funkhouser et al. [40] described the first published system that supported models larger than main memory, and performed speculative prefetching. Their system was based on the from-region visibility algorithm of Teller and Séquin [107], which required long preprocessing times, and was limited to models made of axis-aligned cells. Aliaga et al. [8] presented the Massive Model Rendering (MMR) system, which employed many acceleration techniques, including replacing geometry far from the user's point of view with imagery, occlusion culling, LOD management, and from-region prefetching. MMR was the first published system to handle models with tens of millions of polygons at interactive frame rates.

Recently, Wonka et al. [123] presented a from-region visibility preprocessing algorithm with occluder fusion. The fact that their algorithm took 9 hours to preprocess a model with 8 million triangles, and was limited to 2.5D environments highlights the difficulty of handling complex models. Another example is the work of Durand et al. [32], which presents a from-region visibility preprocessing algorithm that could handle 3D environments, as opposed to 2.5D [123]. But the algorithm took 33 hours to process a model with 6 million triangles. Schaufler et al. [101] also presented a from-region 3D visibility preprocessing algorithm, but their largest test model had only 0.6 million triangles.

The vast amount of work in the area of real-time rendering algorithms makes it impossible for us to be comprehensive. We point the readers to the survey of Cohen et al [23], and the books by Luebke et al [70] and Möller and Haines [73].

5.2 Out-Of-Core Techniques for Model Management

In this section, we briefly review the existing work on out-of-core techniques for computer graphics and scientific visualization. For a general introduction to the theory and practice of external memory algorithms, we refer the interested reader to Abello and Vitter [3].

Cox and Ellsworth [28] propose a general framework for out-of-core scientific visualization systems based on application-controlled demand paging. Leutenegger and Ma [62] propose to use R-trees [46] to optimize searching operations on large unstructured datasets. Ueng et al [111] uses an octree partition to restructure unstructured grids to optimize the computation of streamlines. Shen et al [102] and Sutton and Hansen [105] have developed techniques for indexing time-varying datasets. Shen et al [102] apply their technique for volume rendering, while [105] focusses on isosurface computations. Chiang and Silva [19] worked on I/O-optimal algorithms for isosurface generation. An interesting aspect of their work is that even the preprocessing is assumed to be performed completely on a machine with limited memory. Though their technique is quite fast in terms of actually computing the isosurfaces, the associated disk and preprocessing overhead is substantial. This lead to further research [20] on techniques which are able to trade disk overhead for time in the querying for the active cells. They developed a set of useful meta-cell preprocessing techniques. External memory algorithms for surface simplification have been developed by Lindstrom [67] and El-Sana and Chiang [36]. The technique presented in [67] is able to simplify arbitrarily large datasets on machines with just enough memory to hold the output (i.e., the simplified) triangle mesh. Lindstrom and Silva [68] extended the work into a completely memory insensitive technique. Wald et al. [113] developed a ray tracing system that used a cluster of 7 dual-processor PCs to render low-resolution images of models with tens of millions of polygons at interactive frame rates. Their system could preprocess the UNC power plant model [114] in 2.5 hours (two orders of magnitude faster than MMR [8]). Avila and Schroeder [10] and El-Sana and Chiang [36] developed systems for interactive out-of-core rendering based on LOD management, but these systems did not perform occlusion culling. Varadhan and Manocha [112] describe a system for interactive out-of-core rendering that uses hierarchical LODs [37] and from-region prefetching, but no occlusion culling. Cignoni et al. [21] developed an out-of-core algorithm for simplification of large models. Their algorithm first builds a raw (not indexed) octree-based external memory mesh (OEMM), and then traverses the raw OEMM twice to build an indexed OEMM.

5.3 Parallel Rendering and Large-Scale Displays

When interacting with large models, it is natural to want to visualize these models at high resolution. It is possible to use parallelism to increase performance, and, in particular, increase the resolution of graphics displays. A traditional approach to parallel rendering has been to use a high-end parallel machine. More recently, with the explosive growth in power of inexpensive graphics cards for PCs, and the availability of high-speed networks, using a cluster of PCs for parallel rendering has become an attractive alternative, for many

reasons: [65, 96]. First, a cluster of commodity PCs, each costing a few thousand dollars, typically has a better price/performance ratio than a high-end, highly-specialized supercomputer that may cost up to millions of dollars. Second, high-volume off-the-shelf parts tend to improve at faster rates than special-purpose hardware. We can upgrade a cluster of PCs much more frequently than a high-end system, as new inexpensive PC graphics cards become available every 6-12 months. Third, we can easily add or remove machines from the cluster, and even mix machines of different kinds. We can also use the cluster for tasks other than rendering. Finally, the aggregate computing, storage, and bandwidth capacity of a PC cluster grows linearly with the number of machines in the cluster.

Many approaches to parallel rendering have been proposed. Molnar et al. [74] classify parallelization strategies in three categories based on where in the rendering pipeline sorting for visible-surface determination takes place. Sorting may happen during geometry preprocessing, between geometry preprocessing and rasterization, or during rasterization. The three categories of parallelization strategies are sort-first, sort-middle, and sort-last, respectively.

Sort-first algorithms [53, 77, 97, 98] distribute raw primitives (with unknown screen-space coordinates) during geometry preprocessing. These approaches divide the 2D screen into disjoint regions (or tiles), and assign each region to a different processor, which is responsible for all of the rendering in its region. For each frame, a pre-transformation step determines the regions in which each primitive falls. Then a redistribution step transfers the primitives to the appropriate renderers. Sort-first approaches take advantage of frame-to-frame coherence well, since few primitives tend to move between tiles from one frame to the next. Sort-first algorithms can also use any rendering algorithm, since each processor has all the information it needs to do a complete rendering. Furthermore, as rendering algorithms advance, sort-first approaches can take full advantage of them. One disadvantage of sort-first is that primitives may cluster into regions, causing load balancing problems between renderers. Another disadvantage is that more than one renderer may process the same primitive if it overlaps screen region boundaries.

Sort-middle algorithms [5, 39, 76] distribute screen-space primitives between the geometry preprocessing and rasterization stages. Sort-middle approaches assign an arbitrary subset of primitives to each geometry processor, and a portion of the screen to each rasterizer. A geometry processor transforms and lights its primitives, and then sends them to the appropriate rasterizers. Until recently, this approach has been the most popular due to the availability of high-end graphics machines. One disadvantage of sort-middle approaches is that primitives may be distributed unevenly over the screen, causing load imbalance between rasterizers. Sort-middle also requires high bandwidth for the transfer of data between the geometry processing and rasterization stages.

Sort-last approaches [50, 75, 117] distribute pixels during rasterization. They assign an arbitrary subset of the primitives to each renderer. A renderer computes pixel values for its subset, no matter where they fall in the screen, and then transfer these pixels (color and depth values) to compositing processors. Sort-last approaches scale well with respect to the number of primitives, since they render each primitive exactly once. On the other hand, they need a

high bandwidth network to handle all the pixel transfers. Another disadvantage of sort-last approaches is that they only determine the final depth of a pixel during the composition phase, and therefore make it difficult (if not impossible) to use certain rendering algorithms, e.g., transparency and anti-aliasing.

5.4 Polygonal versus Point versus Image-Based Rendering

Interactive rendering of realistic environments has been a focus of computer graphics research for many years [4, 40]. Traditionally, researchers have modeled the geometric and photometric properties of an environment manually, and the resulting 3D models have been frequently polygonal soups or meshes. More recently, the advances in 3D scanning technology have allowed researchers to capture those properties directly from real-world environments [12, 64, 81, 106], and the use of images and points instead of polygons as rendering primitives has become widespread [45, 84, 93, 82, 83, 86].

On a high level, representing geometry with triangles, images, and points are fundamentally equivalent, and in principle, it should be possible to convert from one representation to another with proper algorithms. Unfortunately, our current knowledge is far from making this possible. Because of the way computer graphics developed, polygonal representations are the easiest to render, with extensive hardware support, and the one that we know most about in terms of its algorithmic properties. Because it has been a preferred representation for so long, quite a bit is known about transforming other representations into polygonized models. On the other hand, for data acquisition, points and images are the most natural since 3D scanners and imaging equipment in general output collections of points and/or images. Using techniques developed in computer vision, it is possible to transform (usually registered and calibrated) imagery into 3D points. Actually building models out of the raw input is quite hard (as can be seen from the material covered in this paper). With the substantial amount of research being done, it is getting easier to convert between representations. Several efforts are underway to make it easy to use points and images as basic primitives, in a similar way to how we use polygonal models. For instance, it is now possible to render points quite efficiently [45, 84, 93], and there are even some systems for editing point sets directly [129].

5.5 Case Study: The iWalk System

The iWalk system [25] lets a user walk through a large model at interactive frame rates using a PC with a small amount of memory. Figure 3b shows an image that was rendered by the iWalk system. The main parts of iWalk are the out-of-core preprocessing algorithm and the out-of-core multi-threaded rendering approach.

The out-of-core preprocessing algorithm creates an on-disk hierarchical representation of the input model. More specifically, it creates an octree [99] whose leaves contain the geometry of the input model. The algorithm first breaks the model in *sections* that fit in main memory, and then incrementally builds the octree on disk, one pass for each section, keeping in memory only the section being processed. To store the octree on disk, the preprocessing

algorithm saves the geometric contents of each octree node in a separate file. The preprocessing algorithm also creates a *hierarchy structure* (HS) file. The HS file has information about the spatial relationship of the nodes in the hierarchy, and for each node it contains the node's bounding box and auxiliary data needed for visibility culling. The HS file is the main data structure that our system uses to control the flow of data. A key assumption we make is that the HS file fits in memory. That is usually a trivial assumption. For example, the size of the HS file for a 13-million-triangle model is only 1 MB.

At run time, iWalk uses an out-of-core multi-threaded rendering approach. A rendering thread uses the PLP [57] algorithm to determine the set of octree nodes that are visible from the user's point of view. For each visible node, the rendering thread sends a fetch request to the fetch threads, which will process the request, and bring the contents of the node from disk into a memory cache. If the cache is full, the least recently used node in the cache is evicted from memory. To minimize the chance of I/O bursts, there is a look-ahead thread that runs concurrently with the rendering thread. The look-ahead thread tries to predict where the user is going to be in the next few frames, uses PLP to determine the nodes that the user would see then, and sends prefetch requests to the prefetch threads. If there are no fetch requests pending, the prefetch threads will bring the requested nodes into memory (up to certain limit per frame based on the disk's bandwidth). This prefetching scheme amortizes the bursts of I/O over frames that require little or no I/O, and produces faster and smoother frame rates.

The rendering thread and the look-ahead thread both use PLP [57] to determine the nodes that are visible from a given point. PLP is an approximate, from-point visibility algorithm that may be understood as a set of modifications to the traditional hierarchical view frustum culling algorithm [22]. First, instead of traversing the model hierarchy in a predefined order, PLP keeps the hierarchy leaf nodes in a priority queue called the *front*, and traverses the nodes from highest to lowest priority. When PLP visits a node, it adds the node to the *visible set*, removes the node from the front, and adds the unvisited neighbors of the node to the front. Second, instead of traversing the entire hierarchy, PLP works on a budget, stopping the traversal after a certain number of primitives have been added to the visible set. Finally, PLP requires each node to know not only its children, but also all of its neighbors. An implementation of PLP may be simple or sophisticated, depending on the heuristic to assign priorities to each node. Several heuristics precompute for each node a value between 0.0 and 1.0 called *solidity*, which estimates how likely it is for the node to occlude an object behind it. At run time, the priority of a node is found by initializing it to 1.0, and attenuating it based on the solidity of the nodes found along the traversal path to the node.

The key feature of PLP that iWalk exploits is that PLP can generate an approximate visible set based only on the information stored in the HS file created at preprocessing time. In other words, PLP can estimate the visible set *without* access to the actual scene geometry.

Although PLP is in practice quite accurate for most frames, it does not guarantee image quality, and some frames may show objectionable artifacts. To avoid this potential problem, the rendering thread may optionally use cPLP [58], a conservative extension of PLP that guarantees 100% accurate images. On the other hand, cPLP cannot determine the visible set

from the HS file only, and needs to read the geometry of all potentially visible nodes. The additional I/O operations make cPLP much slower than PLP.

Corrêa et al. [26] used iWalk to develop a sort-first parallel system for out-of-core rendering of large models on cluster-based tiled displays. Their system is able to render high-resolution images of large models at interactive frame rates using off-the-shelf PCs with small memory. Given a model, they use an out-of-core preprocessing algorithm to build an on-disk hierarchical representation for the model. At run time, each PC renders the image for a display tile, using iWalk's rendering approach. Using a cluster of 16 PCs, each with 512 MB of main memory, they were able to render 12-megapixel images of a 13-million-triangle model with 99.3% of accuracy at 10.8 frames per second. Rendering such a large model at high resolutions and interactive frame rates would typically require expensive high-end graphics hardware. Their results show that a cluster of inexpensive PCs is an attractive alternative to those high-end systems.

6 Challenges Ahead

Despite the great potential of the integrated use of laser scanning and color photographs, several challenges still need to be overcome, including handling inconsistent illumination, being able to dynamically change the scene lighting conditions, and reconstructing non-stochastic textures.

In order to handle possible occlusions and cover the geometric complexity usually associated with real environments, laser scans and photographs usually need to be acquired from several different positions. The existence of view-dependent effects tends to introduce shading inconsistences in sets of pictures taken from different viewpoints. As these pictures are combined to model a scene, seams due to abrupt changes in shading across the same surface can be very distracting. A possible way to address this problem is to extract surface reflectance properties, for example, in the form of BRDFs, and use them for rendering. While some techniques have been developed for extracting reflectance properties from sets of photographs associated with known geometry [61, 126], these approaches assume that the properties and positions of the lights sources in the scene is are known. As a result, they cannot be used for arbitrary scenes. Alternatively, one can consider factoring the illumination reflected by a surface into its diffuse and specular components. This kind of decomposition has been studied by several researchers in recent years [56, 66, 78, 100, 122], but satisfatory results are only achieved under controllable conditions very unlikely to be found in real environments. The ability to dynamically change the scene lighting conditions by, for example, changing the position of the light sources is a highly desirable feature closely related to the problem of factoring shading between its diffuse and specular components.

While several algorithms have been developed in recent years for synthesizing stochastic textures [34, 35, 118] and for texture synthesis on surfaces [109, 118, 124], reconstruction of arbitrary textures is a hard problem for which no general solution is likely to exist. For these cases, the use of a painting system to perform manual retouching may be the only solution.

7 Conclusion

This tutorial is an introduction to recent work in the modeling and interactive rendering of real environments. These topics lie in the frontier between computer graphics and computer vision research, and have recently been the source of substantial work. We expect this to continue to be the case for years to come, and that in the near future we might start to see complete commercial 3D photography systems based on research similar to the one described here. The usefulness of such technology is very broad, ranging from entertainment to the analysis of forensic records, and we expect it to have tremendous impact in everyday life.

We would like to point out that our tutorial is highly skewed towards our research areas, and, thus, it is not representitive of all areas of work. For a more complete description of work in 3D model acquisition, we point the reader to the excellent survey of Bernardini and Rushmeier [13]. For those interested in interactive rendering, we recommend the survey by Cohen-Or et al. [23], and the books by Luebke et al. [70] and Möller and Haines [73].

Acknowledgements

The authors would like to thank the UNC Chapel Hill IBR Group for kindly providing the Reading Room dataset. Some of the material presented here is based on work done jointly with Shachar Fleishman of Tel Aviv University and James T. Klosowski of the IBM T.J. Watson Research Center.

References

[1] 3DV Systems, Inc. ZCam. http://www.3dvsystems.com.

[2] 3rdTech. Deltasphere-3000 laser 3D scene digitizer.

[3] J. Abello and J. Vitter. *External Memory Algorithms and Visualization*, volume 50 of *DIMACS Book Series*. American Mathematical Society, 1999.

[4] J. M. Airey, J. H. Rohlf, and F. P. Brooks, Jr. Towards image realism with interactive update rates in complex virtual building environments. *1990 ACM Symposium on Interactive 3D Graphics*, 24(2):41–50, Mar. 1990.

[5] K. Akeley. RealityEngine graphics. In *Proceedings of SIGGRAPH 93*, pages 109–116, 1993.

[6] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C. T. Silva. Point set surfaces. *IEEE Visualization 2001*, pages 21–28, Oct. 2001.

[7] D. Aliaga and I. Carlbom. Plenoptic stitching: A scalable method for reconstructing 3D interactive walkthroughs. In *Proceedings of SIGGRAPH 2001*, pages 443–450, 2001.

[8] D. Aliaga, J. Cohen, A. Wilson, H. Zhang, C. Erikson, K. Hoff, T. Hudson, W. Stürzlinger, E. Baker, R. Bastos, M. Whitton, F. Brooks, and D. Manocha. MMR: An interactive massive model rendering system using geometric and image-based acceleration. *1999 ACM Symposium on Interactive 3D Graphics*, pages 199–206, 1999.

[9] N. Amenta, M. Bern, and M. Kamvysselis. A new Voronoi-based surface reconstruction algorithm. In *Proceedings of SIGGRAPH 98*, pages 415–421, 1998.

[10] L. S. Avila and W. Schroeder. Interactive visualization of aircraft and power generation engines. In *IEEE Visualization 97*, pages 483–486. IEEE, Nov. 1997.

[11] C. L. Bajaj, F. Bernardini, and G. Xu. Automatic reconstruction of surfaces and scalar fields from 3D scans. In *Proceedings of SIGGRAPH 95*, pages 109–118, 1995.

[12] F. Bernardini, I. Martin, J. Mittleman, H. Rushmeier, and G. Taubin. Building a digital model of Michelangelo's Florentine Pietà. *IEEE Computer Graphics & Applications*, 22(1):59–67, Jan. 2002.

[13] F. Bernardini and H. Rushmeier. The 3D model acquisition pipeline. *Computer Graphics Forum*, 21(2):149–172, June 2002.

[14] P. Besl and N. Mckay. A method for registration of 3D shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, 1992.

[15] J.-Y. Bouguet. Camera calibration toolbox for matlab. http://www.vision.caltech.edu/-bouguetj/calib_doc.

[16] J. C. Carr, R. K. Beatson, J. Cherrie, T. J. Mitchell, W. R. Fright, B. C. McCallum, and T. R. Evans. Reconstruction and representation of 3D objects with radial basis functions. In *Proceedings of SIGGRAPH 2001*, pages 67–76, 2001.

[17] S. E. Chen. QuickTime VR — an image-based approach to virtual environment navigation. In *Proceedings of SIGGRAPH 95*, pages 29–38, 1995.

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

[19] Y.-J. Chiang and C. T. Silva. I/O optimal isosurface extraction. *IEEE Visualization 97*, pages 293–300, Nov. 1997.

[20] Y.-J. Chiang, C. T. Silva, and W. J. Schroeder. Interactive out-of-core isosurface extraction. *IEEE Visualization 98*, pages 167–174, Oct. 1998.

[21] P. Cignoni, C. Rocchini, C. Montani, and R. Scopigno. External memory management and simplification of huge meshes. *IEEE Transactions on Visualization and Computer Graphics*, 2002. To appear.

[22] J. H. Clark. Hierarchical geometric models for visible surface algorithms. *Communications of the ACM*, 19(10):547–554, Oct. 1976.

[23] D. Cohen-Or, Y. Chrysanthou, C. T. Silva, and F. Durand. A survey of visibility for walkthrough applications. *IEEE Transactions on Visualization and Computer Graphics*, 2002. To appear.

[24] W. T. Corrêa, S. Fleishman, and C. T. Silva. Towards point-based acquisition and rendering of large real-world environments. In *Proceedings of the 15th Brazilian Symposium on Computer Graphics and Image Processing*, 2002. To appear.

[25] W. T. Corrêa, J. T. Klosowski, and C. T. Silva. iWalk: Interactive out-of-core rendering of large models. Technical Report TR-653-02, Princeton University, 2002.

[26] W. T. Corrêa, J. T. Klosowski, and C. T. Silva. Out-of-core sort-first parallel rendering for cluster-based tiled displays. In *Proceedings of the 4th Eurographics Workshop on Parallel Graphics and Visualization*, 2002. To appear.

[27] G. M. Cortelazzo, C. Doretto, and L. Lucchese. Free-form textured surfaces registration by a frequency domain technique. In *Proceedings of the International Conference on Image Processing*, pages 813–817, 1998.

[28] M. B. Cox and D. Ellsworth. Application-controlled demand paging for out-of-core visualization. *IEEE Visualization 97*, pages 235–244, Nov. 1997.

[29] B. Curless. *New Methods for Surface Reconstruction from Range Images*. PhD thesis, Stanford University, 1997.

[30] P. Debevec, C. Taylor, and J. Malik. Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. In *Proceedings of SIGGRAPH 96*, pages 11–20, 1996.

[31] P. E. Debevec, Y. Yu, and G. D. Borshukov. Efficient View-Dependent Image-Based Rendering with Projective Texture-Mapping. *Eurographics Rendering Workshop 1998*, pages 105–116, 1998.

[32] F. Durand, G. Drettakis, J. Thollot, and C. Puech. Conservative visibility preprocessing using extended projections. In *Proceedings of SIGGRAPH 2000*, pages 239–248, 2000.

[33] H. Edelsbrunner and E. P. Mucke. Three-dimensional alpha shapes. *ACM Transactions on Graphics*, 13(1):43–72, Jan. 1994.

[34] A. A. Efros and W. T. Freeman. Image quilting for texture synthesis and transfer. In *Proceedings of SIGGRAPH 2001*, pages 341–346, 2001.

[35] A. A. Efros and T. K. Leung. Texture synthesis by non-parametric sampling. In *Proceedings of the International Conference on Computer Vision 99*, pages 1033–1038, 1999.

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

[37] C. Erikson, D. Manocha, and W. V. Baxter III. HLODs for faster display of large static and dynamic environments. In *2001 ACM Symposium on Interactive 3D Graphics*, pages 111–120, Mar. 2001.

[38] S. Fleishman, D. Cohen-Or, and D. Lischinski. Automatic camera placement for image-based modeling. *Computer Graphics Forum*, 19(2):100–110, 2000.

[39] H. Fuchs, J. Poulton, J. Eyles, T. Greer, J. Goldfeather, D. Ellsworth, S. Molnar, G. Turk, B. Tebbs, and L. Israel. Pixel-planes 5: A heterogeneous multiprocessor graphics system using processor-enhanced memories. In *Proceedings of SIGGRAPH 89*, pages 79–88, 1989.

[40] T. A. Funkhouser, C. H. Séquin, and S. J. Teller. Management of large amounts of data in interactive building walkthroughs. *1992 ACM Symposium on Interactive 3D Graphics*, 25(2):11–20, Mar. 1992.

[41] B. Garlick, D. R. Baum, and J. M. Winget. Interactive viewing of large geometric databases using multiprocessor graphics workstations. In *SIGGRAPH 90 Course Notes (Parallel Algorithms and Architectures for 3D Image Generation)*. ACM SIGGRAPH, 1990.

[42] M. Gopi and S. Krishnan. A fast and efficient projection-based approach for surface reconstruction. *High Performance Computer Graphics, Multimedia and Visualization*, 1(1):1–12, 2000.

[43] S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen. The Lumigraph. In *Proceedings of SIGGRAPH 96*, pages 43–54, 1996.

[44] M. A. Greenspan and P. Boulanger. Efficient and reliable template set matching for 3D object recognition. In *Proceedings of the 2nd International Conference on 3D Digital Imaging and Modeling*, pages 230–239, 1999.

[45] J. Grossman and W. J. Dally. Point sample rendering. In *9th Eurographics Workshop on Rendering*, pages 181–192, Aug. 1998.

[46] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proc. ACM SIGMOD Conf. Principles Database Systems*, pages 47–57, 1984.

[47] O. Hall-Holt and S. Rusinkiewicz. Stripe boundary codes for real-time structured-light range scanning of moving objects. In *Proceedings of the 8th International Conference on Computer Vision*, pages 359–366, 2001.

[48] I. Hargittai and M. Hargittai. *Symmetry: A Unifying Concept*. Shelter Publications, Bolinas, California, 1994.

[49] P. Hébert, D. Laurendeau, and D. Poussart. Scene reconstruction and description: Geometric primitive extraction from multiple view scattered data. In *Proceedings of IEEE Computer Vision and Pattern Recognition*, pages 286–292, 1993.

[50] A. Heirich and L. Moll. Scalable distributed visualization using off-the-shelf components. *Symposium on Parallel Visualization and Graphics*, pages 55–59, 1999.

[51] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Surface reconstruction from unorganized points. In *Proceedings of SIGGRAPH 92*, pages 71–78, 1992.

[52] B. K. P. Horn. Closed form solution of absolute orientation using unit quaternions. *Journal of the Optical Society A*, 4(4):629–642, Apr. 1987.

[53] G. Humphreys, M. Eldridge, I. Buck, G. Stoll, M. Everett, and P. Hanrahan. WireGL: A scalable graphics system for clusters. In *Proceedings of SIGGRAPH 2001*, pages 129–140, 2001.

[54] A. E. Johnson and M. Hebert. Using spin images for efficient object recognition in cluttered 3D scenes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(5):433–449, May 1999.

[55] T. Kanade, P. Rander, and P. J. Narayanan. Virtualized reality: Constructing virtual worlds from real scenes. *IEEE MultiMedia*, 4(1):34–47, Mar. 1997.

[56] G. J. Klinker, A. Shafer, and T. Kanade. The measurements of highlights in color images. *International Journal of Computer Vision*, 2(1):7–32, 1988.

[57] J. T. Klosowski and C. T. Silva. The prioritized-layered projection algorithm for visible set estimation. *IEEE Transactions on Visualization and Computer Graphics*, 6(2):108–123, Apr. 2000.

[58] J. T. Klosowski and C. T. Silva. Efficient conservative visibility culling using the prioritized-layered projection algorithm. *IEEE Transactions on Visualization and Computer Graphics*, 7(4):365–379, Oct. 2001.

[59] P. Lancaster and K. Salkauskas. *Curve and Surface Fitting*. Academic Press, London, 1986.

[60] H. Lensch, M. Goesele, J. Kautz, W. Heidrich, and H. Seidel. Image-based reconstruction of spatially varying materials. In *Rendering Techniques 2001*, pages 103–114, 2001.

[61] H. P. A. Lensch, J. Kautz, M. Goesele, W. Heidrich, and H.-P. Seidel. Image-based reconstruction of spatially varying materials. In *Eurographics Rendering Workshop 2001*, pages 104–115, 2001.

[62] S. Leutenegger and K.-L. Ma. *External Memory Algorithms and Visualization*, volume 50 of *DIMACS Book Series*, chapter Fast Retrieval of Disk-Resident Unstructured Volume Data for Visualization. American Mathematical Society, 1999.

[63] D. Levin. Mesh-independent surface interpolation. Technical report, Tel-Aviv University, 2000. Available online at http://www.math.tau.ac.il/˜levin.

[64] M. Levoy, K. Pulli, B. Curless, S. Rusinkiewicz, D. Koller, L. Pereira, M. Ginzton, S. Anderson, J. Davis, J. Ginsberg, J. Shade, and D. Fulk. The digital Michelangelo project: 3D scanning of large statues. In *Proceedings of SIGGRAPH 2000*, pages 131–144, 2000.

[65] K. Li, H. Chen, Y. Chen, D. W. Clark, P. Cook, S. Damianakis, G. Essl, A. Finkelstein, T. Funkhouser, T. Housel, A. Klein, Z. Liu, E. Praun, R. Samanta, B. Shedd, J. P. Singh, G. Tzanetakis, , and J. Zheng. Early experiences and challenges in building and using a scalable display wall system. *IEEE Computer Graphics and Applications*, 25(4):671–680, 2000.

[66] S. Lin and H.-Y. Shum. Separation of diffuse and specular reflection in color images. In *Proceedings of IEEE Computer Vision and Pattern Recognition*, pages 341–346, 2001.

[67] P. Lindstrom. Out-of-core simplification of large polygonal models. In *Proceedings of SIGGRAPH 2000*, pages 259–262, 2000.

[68] P. Lindstrom and C. T. Silva. A memory insensitive technique for large model simplification. In *IEEE Visualization 2001*, pages 121–126, Oct. 2001.

[69] Y. Liu. Computational symmetry. Technical Report CMU-RI-TR-00-31, The Robotics Institute, Carnegie Mellon University, 2000.

[70] D. Luebke, M. Reddy, J. D. Cohen, A. Varshney, B. Watson, and R. Huebner. *Level of Detail for 3D Graphics*. Morgan Kaufmann, 2002.

[71] D. K. McAllister, L. Nyland, V. Popescu, A. Lastra, and C. McCue. Real-time rendering of real world environments. In *Rendering Techniques 99*, pages 145–160, 1999.

[72] L. McMillan and G. Bishop. Plenoptic modeling: An image-based rendering system. In *Proceedings of SIGGRAPH 95*, pages 39–46, 1995.

[73] T. Möller and E. Haines. *Real-Time Rendering*. A K Peters, 1999.

[74] S. Molnar, M. Cox, D. Ellsworth, and H. Fuchs. A sorting classification of parallel rendering. *IEEE Computer Graphics and Applications*, 14(4):23–32, 1994.

[75] S. Molnar, J. Eyles, and J. Poulton. Pixelflow: High-speed rendering using image composition. In *Proceedings of SIGGRAPH 92*, pages 231–240, 1992.

[76] J. S. Montrym, D. R. Baum, D. L. Dignam, and C. J. Migdal. InfiniteReality: A real-time graphics system. In *Proceedings of SIGGRAPH 97*, pages 293–302, 1997.

[77] C. Mueller. The sort-first rendering architecture for high-performance graphics. *1995 ACM Symposium on Interactive 3D Graphics*, pages 75–84, 1995.

[78] S. Nayar, X. Fang, and T. Boult. Removal of specularities using color and polarization. In *Proceedings of IEEE Computer Vision and Pattern Recognition*, pages 583–590, 1993.

[79] S. K. Nayar, M. Watanabe, and M. Noguchi. Real-time focus range sensor. In *Proceedings of the 5th International Conference on Computer Vision*, pages 1186–1198, Dec. 1996.

[80] P. J. Neugebauer. Reconstruction of real-world objects via simultaneous registration and robust combination of multiple range images. *International Journal of Shape Modeling*, 3(1 & 2):71–90, 1997.

[81] L. Nyland, A. Lastra, D. K. McAllister, V. Popescu, and C. McCue. Capturing, processing and rendering real-world scenes. In *Videometrics and Optical Methods for 3D Shape Measurement, Electronic Imaging 2001, Photonics West*, volume 4309, pages 107–116. SPIE, 2001.

[82] L. Nyland, D. McAllister, V. Popescu, C. McCue, A. Lastra, P. Rademacher, M. M. Oliveira, G. Bishop, G. Meenakshisundaram, M. Cutts, and H. Fuchs. The impact of dense range data on computer graphics. In *Proceedings of Multi-View Modeling and Analysis Workshop*, pages 3–10, 1999.

[83] M. M. Oliveira, G. Bishop, and D. McAllister. Relief texture mapping. In *Proceedings of SIGGRAPH 2000*, pages 359–368, July 2000.

[84] H. Pfister, M. Zwicker, J. van Baar, and M. Gross. Surfels: Surface elements as rendering primitives. In *Proceedings of SIGGRAPH 2000*, pages 335–342, 2000.

[85] R. Pito. A solution to the next best view problem for automated surface acquisition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(10):1016–1030, 1999.

[86] V. S. Popescu, A. Lastra, D. G. Aliaga, and M. M. de Oliveira Neto. Efficient warping for architectural walkthroughs using layered depth images. In *IEEE Visualization 98*, pages 211–216, Oct. 1998.

[87] K. Pulli. Multiview registration for large datasets. In *Proceedings of the 2nd International Conference on 3D Digital Imaging and Modeling*, pages 160–168, 1999.

[88] R. Raskar, G. Welch, M. Cutts, A. Lake, L. Stesin, and H. Fuchs. The office of the future: A unified approach to image-based modeling and spatially immersive displays. In *Proceedings of SIGGRAPH 98*, pages 179–188, 1998.

[89] G. Roth. Registering two overlapping range images. In *Proceedings of the 2nd International Conference on 3D Digital Imaging and Modeling*, pages 191–200, 1999.

[90] H. Rushmeier, G. Taubin, and A. Guéziec. Applying shape from lighting variation to bump map capture. In *Proceedings of the 8th Eurographics Rendering Workshop*, pages 35–44, June 1997.

[91] S. Rusinkiewicz. *Real-time Acquisition and Rendering of Large 3D Models*. PhD thesis, Stanford University, 2001.

[92] S. Rusinkiewicz. Sensing for graphics. http://www.cs.princeton.edu/courses/archive/-fall01/cs597d, 2001.

[93] S. Rusinkiewicz and M. Levoy. QSplat: A multiresolution point rendering system for large meshes. In *Proceedings of SIGGRAPH 2000*, pages 343–352, 2000.

[94] S. Rusinkiewicz and M. Levoy. Efficient variants of the ICP algorithm. In *Proceedings of the 3rd International Conference on 3D Digital Imaging and Modeling*, pages 145–152, 2001.

[95] M. Rutishauser, M. Stricker, and M. Trobina. Merging range images of arbitrarily shaped objects. In *Proceedings of IEEE Computer Vision and Pattern Recognition*, pages 573–580, 1994.

[96] R. Samanta, T. Funkhouser, K. Li, and J. P. Singh. Hybrid sort-first and sort-last parallel rendering with a cluster of PCs. In *2000 SIGGRAPH/Eurographics Workshop on Graphics Hardware*, pages 97–108, 2000.

[97] R. Samanta, T. Funkhouser, K. Li, and J. P. Singh. Sort-first parallel rendering with a cluster of PCs. In *Sketches and Applications, SIGGRAPH 2000*, page 260, 2000.

[98] R. Samanta, J. Zheng, T. Funkhouser, K. Li, and J. P. Singh. Load balancing for multi-projector rendering systems. In *1999 SIGGRAPH/Eurographics Workshop on Graphics Hardware*, pages 107–116, 1999.

[99] H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, 1990.

[100] Y. Sato and K. Ikeuchi. Temporal-color space analysis of reflection. *Journal of the Optical Society of America A*, 11(11):2990–3002, 1994.

[101] G. Schaufler, J. Dorsey, X. Decoret, and F. X. Sillion. Conservative volumetric visibility with occluder fusion. In *Proceedings of SIGGRAPH 2000*, pages 229–238, 2000.

[102] H.-W. Shen, L.-J. Chiang, and K.-L. Ma. A fast volume rendering algorithm for time-varying fields using a time-space partitioning (TSP) tree. *IEEE Visualization 99*, pages 371–378, Oct. 1999.

[103] M. Soucy and D. Laurendeau. A general surface approach to the integration of a set of range views. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(4):344–358, Apr. 1995.

[104] G. Strang. *Linear Algebra and Its Applications*. Saunders HBJ College Publishers, Orlando, Florida, 3rd edition, 1988.

[105] P. M. Sutton and C. D. Hansen. Accelerated isosurface extraction in time-varying fields. *IEEE Transactions on Visualization and Computer Graphics*, 6(2):98–107, Apr. 2000.

[106] S. Teller. Toward urban model acquisition from geo-located images. *Pacific Graphics 98*, pages 45–52, 1998.

[107] S. J. Teller and C. H. Séquin. Visibility preprocessing for interactive walkthroughs. In *Proceedings of SIGGRAPH 91*, pages 61–69, 1991.

[108] R. Y. 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):323–344, Aug. 1987.

[109] G. Turk. Texture synthesis on surfaces. In *Proceedings of SIGGRAPH 2001*, pages 347–354, 2001.

[110] G. Turk and M. Levoy. Zippered polygon meshes from range images. In *Proceedings of SIGGRAPH 94*, pages 311–318, 1994.

[111] S.-K. Ueng, C. Sikorski, and K.-L. Ma. Out-of-core streamline visualization on large unstructured meshes. *IEEE Transactions on Visualization and Computer Graphics*, 3(4):370–380, Oct. 1997.

[112] G. Varadhan and D. Manocha. Out-of-core rendering of massive geometric environments. In *IEEE Visualization 2002*, 2002. To appear.

[113] I. Wald, P. Slusallek, and C. Benthin. Interactive distributed ray tracing of highly complex models. *Rendering Techniques 2001*, pages 277–288, 2001.

[114] Walkthru Project at UNC Chapel Hill. Power plant model. http://www.cs.unc.edu/-˜geom/Powerplant.

[115] J. Wang and M. M. Oliveira. A hole filling strategy for surface reconstruction from range images. Technical Report TR02.07.18, SUNY at Stony Brook, 2002.

[116] J. Wang and M. M. Oliveira. Improved scene reconstruction from range images. *Computer Graphics Forum*, 21(3), Sept. 2002. To appear.

[117] B. Wei, D. W. Clark, E. W. Felten, K. Li, and G. Stoll. Performance issues of a distributed frame buffer on a multicomputer. *1998 SIGGRAPH / Eurographics Workshop on Graphics Hardware*, pages 87–96, 1998.

[118] L.-Y. Wei and M. Levoy. Texture synthesis over arbitrary manifold surfaces. In *Proceedings of SIGGRAPH 2001*, pages 355–360, 2001.

[119] H. Weyl. *Symmetry*. Princeton University Press, Princeton, New Jersey, 1952.

[120] R. Whitaker, J. Gregor, and P. Chen. Indoor scene reconstruction from sets of noisy range images. In *Proceedings of the 2nd International Conference on 3-D Digital Imaging and Modeling*, pages 348–357, 1999.

[121] R. Willson. Freeware implementation of Roger Tsai's camera calibration algorithm. Available online at http://www.cs.cmu.edu/˜rgw/TsaiCode.html.

[122] L. B. Wolf. Using polarization to separate reflection components. In *Proceedings of IEEE Computer Vision and Pattern Recognition*, pages 363–369, 1989.

[123] P. Wonka, M. Wimmer, and D. Schmalstieg. Visibility preprocessing with occluder fusion for urban walkthroughs. In *Rendering Techniques 2000*, pages 71–82, 2000.

[124] L. Ying, A. Hertzman, H. Bierman, and D. Zorin. Texture and shape synthesis on surfaces. In *Rendering Techniques 2001*, pages 301–312, 2001.

[125] Y. Yu, P. Debevec, J. Malik, and T. Hawkins. Inverse global illumination: Recovering reflectance models of real scenes from photographs. In *Proceedings of SIGGRAPH 99*, pages 215–224, 1999.

[126] Y. Yu, A. Ferencz, and J. Malik. Extracting objects from range and radiance images. *IEEE Transactions on Visualization and Computer Graphics*, 7(4):351–364, 2001.

[127] D. Zhang and M. Hebert. Harmonic maps and their application in surface matching. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 524–530, 1999.

[128] Z. Zhang. Iterative point matching for registration of free-form curves and surfaces. *International Journal of Computer Vision*, 13(2):119–152, 1994.

[129] M. Zwicker, M. Pauly, O. Knoll, and M. Gross. Pointshop 3D: An interactive system for point-based surface editing. In *Proceedings of SIGGRAPH 2002*, pages 322–329, 2002.