ActivePoints: Acquisition, Processing and Navigation of Large Outdoor Environments

Hui Xu Baoquan Chen

Department of Computer Science and Engineering
University of Minnesota at Twin Cities
http://www.cs.umn.edu/∼{hxu,baoquan}
Email:{hxu, baoquan}@cs.umn.edu

Abstract

A longstanding challenge for computer graphics has been capturing images of real-world objects and building their 3D digital representations accurately and efficiently. Recently, rapid advancements in laser range scanning hardware have much improved the accuracy and efficiency of 3D scanning. In meeting the needs of various applications, the main graphics issues are how to process the paramount size of scanned raw data and to build concise and accurate 3D representations so that scanned objects or environments can be efficiently visualized and manipulated.

In this paper, we describe a system, dubbed ActivePoints, for acquiring images of large outdoor environments by employing one of the most advanced scanning devices currently available: Riegl USA's LMS Z-360 3D image sensor. We have designed and developed algorithms for creating efficient 3D representations and rendering tools for efficient interaction with the data. Even though ActivePoints shares a common scanning pipeline with most existing scanning systems, it features several unique approaches. First, ActivePoints aims to generate *a hierarchical point-based model* as the final representation. Second, to build this hierarchical model, ActivePoints employs *an interactive segmentation and object organization tool* coupled with automatic segmentation algorithms. And finally, ActivePoints benefits from a novel algorithm that can capitalize on new features in modern graphics hardware for *efficient and high quality point-based rendering*. We offer examples to demonstrate that point-based models are better suited for representing and visualizing large outdoor environments than earlier methods and that ActivePoints proves to be an efficient system that provides a platform for continuing exploration.

Keywords: 3D scanning, laser range acquisition, 3D photography, point-based rendering, interactive 3D graphics.

1 Introduction

For enhancing the realism of graphic representations, computer graphics practitioners have been employing real-world 2D photographs as textures mapped on objects. Recently, researchers have shown an increasing interest in capturing and processing real-world 3D objects and scenes to achieve the ultimate realistic virtual experience. Many applications, such as in architecture design, entertainment (games, movie special effects), telepresence, and documentation of ancient architecture and archeology sites, would benefit from such work. To these ends, new ways of digitizing existing environments and viewing them are required.

However, capturing real-world objects and scenes, especially in outdoor environments, poses great challenges, requiring the development of new software and hardware systems. Current software systems take an image-based modeling approach, often employing computer vision techniques for scene model construction [8], that can be combined with interactive modeling tools [3]. These methods are inexpensive, as no special scanning hardware is needed, and they can capture objects and scenes of any size. However, the constructed representations are often inaccurate, as the methods are sensitive to lighting conditions. Present hardware systems employ 3D scanning devices for measuring object distances, which until recently were limited to small objects (e.g., toys or statues) and indoor scenes [6, 5, 1]. In addition, traditional scanners only record range images of the scene; therefore, color images must also be taken with digital cameras, and registration of the range and color images must be conducted.

In this project, we employ one of the currently most advanced acquisition devices for acquiring 3D models of outdoor scenes in order to meet our objective of fast, accurate, and high resolution scanning and interactive virtual navigation of the scanned scenes. Our scanning device is Riegl USA's LMS-Z360 3D imaging sensor, which can measure a distance up to 200m with 12 mm precision and obtain range, intensity, and color images simultaneously. This eliminates the need to register separate range and color images, hence improving scanning accuracy and speeding up the acquisition process. Scanning outdoor scenes is more challenging than scanning indoor objects for a couple of reasons. For one, outdoor scene scanning is more constrained in terms of setting up scanning locations; natural landscape obsta-

cles like bushes and trees can make it impossible to obtain a complete image of the environment. The most significant challenge of outdoor scene scanning is the unprecedented amount of data that needs to be acquired, processed, and rendered efficiently. Because the scanning resolution must be high enough to acquire sufficient details, capturing 1cm details at 100m distance, for example, requires scanning over 6000 points for 360 degrees. A full panoramic scan ($360^\circ \times 90^\circ$) takes about 4 minutes to do, producing over 5 million points.

To process scanned data, most of the existing systems reconstruct scanned sample points into polygon meshes, extract corresponding texture maps out of camera-taken images, and then render textured polygons on conventional graphics hardware. The justification for constructing polygons has been the rendering support offered by the available hardware. But in fact, this usually results in an excessive number of triangles. The high resolution necessary for scanning large environments leads to a high triangle-per-pixel ratio during the novel view rendering. Furthermore, fitting meshes into points, especially those of natural phenomena, may create unwanted artifacts by producing non-existing structures or jagged boundaries between objects [6].

In this system, we instead construct point models from scanned samples. Using points as an alternative modeling primitive has been explored for over a decade and has recently received increased attention from researchers. Points have been shown to be advantageous over polygons when representing highly detailed features [10], and thus as modern graphics hardware provides more support for the rendering of point primitives, point-based rendering will undoubtedly continue to gain in popularity [7]. Besides the efficiency they provide for rendering, points are also more flexible for visualizing and modeling scanned large environment. Although substantial further research will be needed to develop a full-fledged point-based system (with necessary polygons) for processing acquired large outdoor scans, here we present two major efforts we are making in that direction. The first is an interactive segmentation and object organization process coupled with traditional automatic segmentation approaches. The aim of this process is to build up a hierarchical representation of the environment. This modeling process is designed to also compute additional geometric information for points. The second effort is an efficient and high-quality rendering system capable of rendering an environment of millions of points interactively on commodity graphics hardware. The rendered scenes are of a high quality and present the appearance of solid surfaces.

We will begin our discussion of this system with a review of a representative scanning pipeline. We then describe our scanning system in section 3, point-based modeling in section 4, and point-based rendering in section 5. We will present our results in section 6, and finally conclude with a discussion and plan for future work (section 7).

2 A Representative Scanning Pipeline

Although different scanning approaches and applications have resulted in scanning pipelines that emphasize different operations and vary in their order, in this section, we present a representative pipeline consisting of four main steps: (1) denoising and multi-channel registration (of a single scan); (2) multi-scan registration; (3) segmentation, object editing, and organization; and (4) rendering.

- Step 1: Depending on which range scan device is used, noise may be prominent in the scanned samples and hence has to be removed from the range images, a procedure called denoising. And as noted, because most laser scanners can obtain only range images, color images taken separately must be registered with the scanned ranged images.
- Step 2: To obtain an integrated model, a sufficient number of scans has to be taken and then registered and merged into a single 3D representation.
- Step 3: In this step, individual objects may be segmented out and then transformed, copied, or deleted [9]. Various editing operations can also be applied (e.g., filling holes).
- Step 4: Once the complete models are obtained, the last step is to perform rendering for manipulation and navigation.

The goal of most previous systems has been to reconstruct polygon mesh. Therefore, in the first step, after registering color images with range images, texture maps are extracted from color images for constructed polygons [1]. The main task in the second step is removing redundancies among overlapping scans, where meshes from different scans have to be seamlessly connected [5]. Lastly, in the third step, the rendering of textured polygon models is straightforwardly performed on conventional graphics hardware.

There are also existing systems that do not reconstruct polygon mesh. In the system described in [6], image-based rendering is used to produce the color depth images that are the desired final representation. In this system, step 1 resamples color images on range image grids so that each sample contains both color and range information. In step 2, after merging, points are still stored in their original image format and redundant points are removed as groups (or tiles). During the rendering, multiple color depth images are directly projected onto the novel view to form a new rendering through 3D warping or projecting regular mesh constructed out of the depth image.

Our scanning system also follows this representative pipeline, but aims to generate an organized point set, which is different from [6]. Because our scanning device can obtain color and range images simultaneously and offers a high

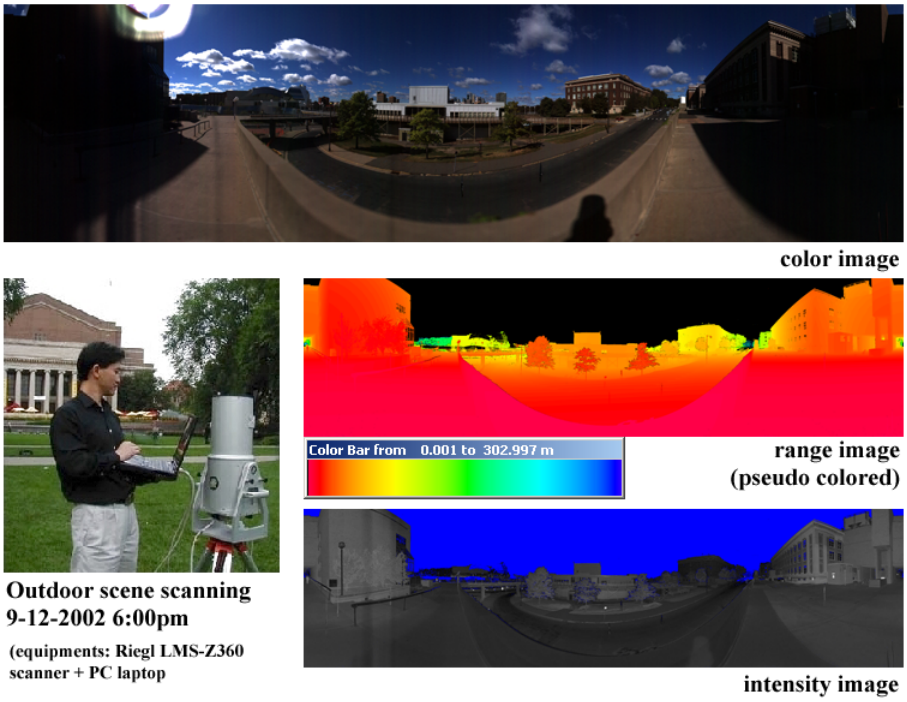


Figure 1: *Scanning devices and images acquired through a single scan of a corner on the University of Minnesota campus. The images have the same resolution of 5 million points.*

level of precision, we do not need to take the operations in steps 1. Thus the following presentation of our system will include only the operations in step 2 to 4. We will begin this description by introducing our scanning hardware system.

3 Scanning Hardware System

Our scanning device is the Riegl LMS-Z360 3D Imaging Sensor, one of the most advanced 3D laser scanners currently available, which records range, intensity, and RGB color. The scanner features long-range measurement and high accuracy (12mm precision up to 200m), a large field of view ($90^\circ \times 360^\circ$), and a high scanning speed (24,000 pts/sec). Figure 1 illustrates the scanning equipment, scanning action, and sample images of a single scan. The lower left image of Figure 1 shows the system setup during the scan. The scanner is mounted on a tripod, which can be further mounted on a cart for moving around easily. The laptop is connected to the scanner through an enhanced parallel port for collecting scanned data, which can also be replaced with a wireless connection for unpleasant weather conditions or avoiding the operators appearing in the scanned scene.

The laptop runs the interface software RiScan to control the scanning process, such as setting the scanning starting point, angles (vertical and/or horizontal), and speed. The scanner scans three channels of information: color (RGB), range, and intensity. In the scanned raw data, each RGB channel is represented using 16 bits. In RiScan, one can define color transformation to change the 16 bits color channel to 8 bits for display. The top and right images of Figure 1 show the color, range, and intensity images of one scan. The bright spot at the top left corner of the color image is the sun. The range value is pseudo color-encoded. Even though the manufacturer specifies the maximum range of this scanner as 200m, as can be seen from Figure 1, sometimes the acquired range can be beyond that. Air conditions do have some influence on the scanning quality, as the best quality can be obtained in clear and dry weather conditions. The scanning resolution has to be high enough to acquire sufficient details. In this report we demonstrate our system using scans of two locations on the University of Minnesota Campus, all scans have high resolution of 5 million points.

4 Point-Based Modeling

We have devised algorithms and tools that directly operate on point sets to obtain an optimized point representation. Our approach involves two main operations. The first is to compute additional geometric information for each point, such as size and normal vector. The second is to group points into clusters based on scene semantics and their local geometry, after which redundant points are removed for optimization. In this approach, no connectivity information is necessarily built up between points. The following details five procedures that are involved in creating point-based representation: segmentation, point geometry calculation, multi-

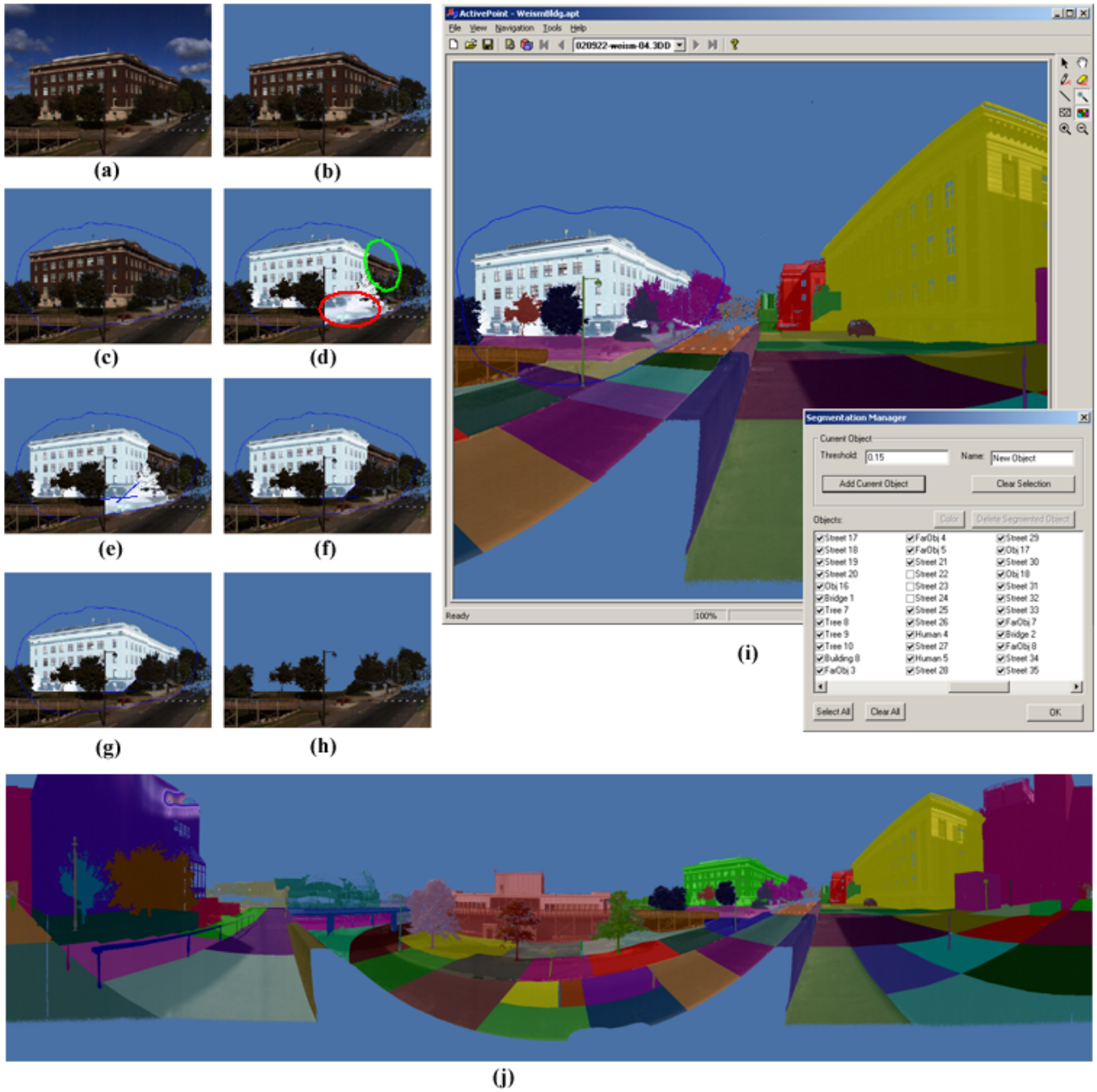


Figure 2: *Example of segmentation procedure: (a) original image; (b) invalid data (without range information) removed; (c) searching area defined; (d) automatic segmentation with mis-classified region (red) and unclassified part of the building (green); (e) assistant lines drawn to indicate boundaries; (f) updated segmentation; (g) combined with other part of the building; (h) the segmented entire building removed and added to object list; (i) in user interface of segmentation tool, segmented objects listed in the "Segmentation Manager" window where they can be selected, deleted, and grouped; (j) completed segmentation of a scene.*

ple scan registration and merging, hierarchical data structure construction, and point model editing.

4.1 Segmentation

Segmentation is an important step in extracting objects for building an individual object hierarchy. For example, points for each building are segmented out and collected into a group; within each building group, points can be further grouped together according to local geometric structures. The final representation is essentially a data hierarchy, with individual points at leaf nodes and clusters of surfaces, walls, and buildings at intermediate nodes.

Although sophisticated segmentation methods are already available for range images (e.g., [6]), these methods are fully automatic and not reliable for complex environments. Therefore, we employ interactive operations to enhance the results of segmentation and speed up the process. Figure 2i shows the segmentation tool interface. A typical segmentation procedure in our system is demonstrated in Figure 2a-h. First, the system loads in the scanned images (Figure 2a (only the color image is shown and here and hereafter). Then samples with invalid range values are removed (Figure 2b). Next, the user draws a close curve (Figure 2c, marked as blue) on the image around the building that needs to be segmented, so that the search is constrained within the close curve for efficiency. The user selects any point on the building by clicking on it. An automatic segmentation algorithm is performed to complete this operation; the system provides several segmentation algorithms to choose from. The user may also choose whether to run the segmentation operation on the range, color, or intensity image, as one channel may provide better segmentation than another for a given object. For example, using the color image for segmentation may give poor results for a building in relative darkness; however, using the range or depth images may produce better segmentation. Figure 2d shows the intermediate segmentation result based on range image. The automatic segmentation algorithms usually need further refinement, especially for large complex environments. As shown in Figure 2d, points inside the red circle are misclassified, and points on the building inside the green circle are not classified. To remove the misclassified points, assistant lines are drawn to indicate boundaries. Figure 2f shows the selection updated after two assistant lines (blue lines in Figure 2e). For points not classified, the user can select a new point and continue the segmentation. The additionally classified points are added to the existing ones. Figure 2g shows the final segmentation of the building. In this process, a set of additional functions like separation, undo, and redo are provided to ensure an efficient and user-friendly process. Finally, the user inserts the segmented building into an object list (lower right window of Figure 2i), removes the building from the scene (Figure 2h), and then continues working on the rest of the scene until all objects are segmented. Figure 2j shows the completed segmentation of the scan. Notice that the ground is manually partitioned for efficient view frustum culling as discussed in 5.3.

4.2 Point Geometry Calculation

The second procedure in creating an optimized point representation is point geometry calculation. Although it is not our objective to reconstruct a complete polygon model, point geometric information is necessary for estimating the on-screen projection size of a point, performing back-face culling and resampling, and so on. Typically, the size, tangent, and normal vectors are calculated for each point.

Each pixel scanned represents a 3D point with a certain size. The pixel azimuth angle is δ_h and altitude angle is δ_v, which can be calculated based on the scanning field-of-view and image resolution. Since angles δ_h and δ_v are small, the horizontal distance m_h and vertical distance m_v in 3D space between two points R-distance away from the scanner can be approximated as $m_h \approx R\delta_h$ and $m_v \approx R\delta_v$, correspondingly. Usually δ_h and δ_v are equal; therefore, $m_h = m_v$. For two adjacent points representing the same surface, we require that their geometries (circular disks) cover the local surface without holes in between. To ensure this, the disk of the point must have a minimum radius of $\frac{\sqrt{2}}{2}m_h$ or $\frac{\sqrt{2}}{2}m_v$.

We take advantage of the regularity of the initial scanned images for computing normals and tangent planes. Point adjacency (even connectivity) information can be obtained from pixel adjacency in the scanned images. Then a local geometry (e.g., a plane or curved patch) can be fitted into the points within the small neighborhood (space) of a point. The geometry (normal and tangent) of the point can then be computed from this fitted local geometry.

4.3 Multiple Scan Registration and Merging

The third procedure in creating an optimized point representation is multiple scan registration and merging. To build up a unified representation of a complex environment, we have to take multiple scans and register and merge points from them into one point set. There are numerous methods that have been developed for registration of multiple scans. In this system, we use PolyWorks, software sold with the scanner, to perform this registration and obtain the transformation matrix for transforming each scan into a central coordinate system. Differently than in a polygon reconstruction approach, our merging is done directly on points. Generally speaking, merging is achieved by removing redundant points with a lower sample rate. For example, when two scans of the same building are merged, the building is represented by the scan taken closest to it (i.e., with higher point resolution). The sample rates of points from two range images are compared by computing the determinant of the Jacobin matrix of the transformation between the two range images, a method similar to measuring the local area change in image warp-

ing. A sampling rate comparison between two points that come from different reference images and correspond to the same region in the scene is then employed. Two points with similar 3D positions and normals are determined to be corresponding points. Unlike the approach of [6], our redundancy test and removal is operated on every sample point, rather than on tiles.

4.4 Hierarchical Data Structure Construction

The fourth procedure in creating an optimized point representation is the construction of a hierarchical data structure. A hierarchical data structure based on segmented objects is essential for point model editing (discussed next). Moreover, it allows view frustum culling to be performed efficiently. Segmented objects, with already-computed additional local geometry information, are either further partitioned or grouped to form the data hierarchy. Partitioning or grouping is determined automatically by the localities of the objects or manually by the user. For example, a building may be partitioned as several components (e.g., walls). The user may then group it with other buildings in the same spatial neighborhood as a single object node in the hierarchical structure. This grouping continues until a single root group is obtained. To facilitate efficient view frustum culling, for each intermediate node we calculate its bounding box and store the information in the data structure.

4.5 Point Model Editing

The last procedure in creating an optimized point representation is point model editing. Here we emphasize visual attributes editing rather than geometry editing [9]. We have developed a tool that allows users to edit the color value (brightness, contrast) and hue of a selected object. The tool also allows for cloning colors, which sometimes proves to be the easiest way to correct local color problems. This kind of color editing is also very important for correcting light and color inconsistencies caused by the movement of the sun during the long scanning process. This is also a very practical tool for dealing with such color inconsistency problems because an automatic method still takes a long way to achieve. This is a problem unique to outdoor scanning, as in indoor scanning light conditions can be controlled. We can use similar image editing tools to conform the colors of two images to one another. We are convinced that more such tools will be useful.

5 Rendering

For point rendering, we employ the traditional splatting method [10], where a point is represented as a disk of certain size (termed splats). We employ efficient view frustum culling for eliminating invisible points, thereby spending computation mostly on visible points, resulting in acceleration. We also leverage modern commodity graphics hardware, e.g., point sprite primitive, for accelerated point rendering. Using the point primitive, points can be rendered as a textured point with texture coordinates (0,0) in the top left corner and (1,1) in the bottom right corner. Our rendering system utilizes hardware-supported point sprites to perform surface splatting with alpha blending [10] and conducts a two-pass rendering approach, much like [7]. Such rendered images promise high quality, but are expensive to compute, especially for complex environments. We have developed several strategies to enable interactive rendering on commodity graphics hardware while compromising little on image quality.

(a) (b) (c)

Figure 3: *Alpha textures: (a) Gaussion* $e^{\frac{r^2}{2\delta^2}}$ *(*$\delta = 0.4$*), (b)* $1 - r^2$*, and (c)* $(cos\pi r + 1)/2$*, where* r *is the radius and alpha is 0 if* $r > 1$*.*

Two-pass rendering is used to ensure correct alpha blending for point sprite rendering (sorting point primitives for this purpose is impractical due to the size of the data). The first rendering pass generates a depth buffer of visible objects. Then in the second pass, the depths of points are compared with the corresponding values in the depth buffer. Only point sprites having comparable depth values are blended with the current framebuffer. In this pass, no depth buffer is updated. Below we will discuss in more detail the two-pass rendering process and the strategies designed for accelerated rendering.

5.1 Two-Pass Rendering

In the first pass, a depth buffer is generated by rendering the scene as opaque points (visibility splatting). In the second pass, the depth buffer is initiated as the depth buffer obtained in the first pass and the depth buffer updating is disabled. Then points are rendered as point sprites with alpha blending enabled. During the rendering, only points passing through depth testing are blended with the frame buffer. One implementation issue is making sure points *on* the visible surfaces get past the depth testing. One solution is to push the visible surfaces slightly away from the viewpoint during the first-pass rendering. Simply increasing all depth values by the same amount will not work because, depending on the point's distance from the view, the same depth deviation translates to different distances in 3D space. The solution

proposed in [7] is to calculate a z offset for each vertex using a vertex shader. We employ a simpler approach here. Rather than using costly computation to translate a depth buffer by a small threshold, we simply push the far clipping plane away from the camera by a certain distance before performing the second-pass rendering. According to the projection formula, the calculated depth values will become slightly smaller so that points on visible surfaces can pass through depth testing. This method was used in shadow rendering as an alternative to hardware-supported z-bias techniques, which are not widely supported.

There may be a problem with this approach for some distant objects in a scene. The problem is caused by the uneven distribution of the traditional perspective-correct depth within the depth buffer range. When the far/near clipping plane ratio is large, as in our outdoor scenes, a large percentage of the depth buffer range is used on the scene depth range of short distance objects. For example, for a far/near ratio of 100, 90 percent of the depth buffer range is used on the first 10 percent of the scene depth range. This means depth values for distant objects are sparsely quantified. Therefore, pushing the far plane by a certain distance results in little depth deviation for distant objects. To address this problem, we use a w buffer in place of a traditional depth buffer. In a w buffer, the homogeneous w-coordinate from the point's (x,y,z,w) location in projection space is used, rather than perspective-correct z (i.e., z/w). Using a w buffer, the buffer bits are more evenly allocated between the near and far clipping planes in world space. Therefore, a w buffer is ideal for our large environment scene rendering since it allows applications to support large ranges while still getting relatively accurate depth value. The w-buffer is supported by most graphics hardware today.

The performance of the rendering can be further improved when scenes are rendered from front to back because a point failing the depth test will not be rasterized. Although a precise sorting on points is impossible for each frame, we can sort segmented objects based on their distances from the camera.

Hardware-supported point sprites allow a point to be rendered as a textured point with texture coordinates (0,0) in the top left corner and (1,1) in the bottom right corner. In the second pass, we combine this technique with a traditional alpha blending method to approximate screen space splatting for each point. The texture of the point is pre-generated alpha texture. Figure 3 shows the alpha textures that we have experimented with. Images in the results section are rendered using the cosine-based alpha texture (Figure 3c), which seems to provide better results most of the time.

5.2 Rendering Point Primitives

When rendering a point sprite, its screen size needs to be specified in addition to its coordinates, color, and texture. Once the point's 3D size is calculated, its screen size can be calculated. We seek hardware support for this calculation as much as possible. Figure 4 illustrates a point P in 3D space, where E is the viewpoint; d is the distance between the viewpoint E and the point P; fov is the field of view; φ the angle between eye ray EP; and the center view direction $\vec{v}$; V is the screen resolution. The point size S_{3D} in 3D space is the diameter of the point disk, whose radius is provided in section 4.2.

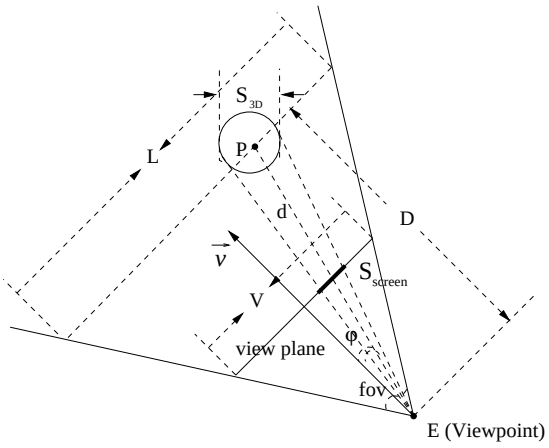


Figure 4: *Estimation of point's screen projection size.*

Most modern graphics hardware can calculate S_{screen} using the following formula:

$$S_{\text{screen}} = \frac{S_{3D}V}{\sqrt{A + Bd + Cd^2}}, \tag{1}$$

where parameters A, B, and C are user-predefined constants; d is calculated by hardware based on the point's 3D coordinates.

Let us compute these constants. Based on similar triangles and with slight approximation, we have $S_{\text{screen}}/V \approx S_{3D}/L$, where $L = 2D\tan\frac{fov}{2}$ and $D = d\cos\varphi$. Thus, we have

$$S_{\text{screen}}{}' = \frac{S_{3D}V}{2d\cos\varphi\tan(fov/2)}. \tag{2}$$

Comparing quation 1 and 2, we have $A = 0$, $B = 0$, and $C(\varphi) = 4\cos^2\varphi\tan^2\frac{fov}{2}$. It is expensive to compute $C(\varphi)$ for every point and to pass it to hardware. Therefore, we use the average angle $\varphi = \frac{fov}{4}$ as an approximation and obtain a constant parameter C:

$$C = 4\cos^2(\frac{fov}{4})\tan^2(\frac{fov}{2}). \tag{3}$$

Unfortunately, most current graphics hardware does not take 3D size S_{3D} as one of the point's attributes (like coordinates and color), but simply take it as a place holder. For example, in DirectX8.1, the 3D point size has to be specified by calling function *SetRenderState* with parameter D3DRS_POINTSIZE. However, it is not applicable to call *SetRenderState* for every point to set its 3D size, as it will

slow down the rendering significantly. We solve this problem by specifying an average point size for each segmented object instead of each point. Considering that the number of objects ranges from only tens to hundreds, the overhead of calling *SetRenderState* function is almost negligible. As we are using an averaged 3D point size, to ensure a reasonable approximation for each point, we have to group points tightly together. As discussed in 4.2, the size of a point is decided mainly by the distance between the point and the scanner since the per pixel azimuth angle and altitude angle are nearly constant. Therefore, this size averaging is only reasonable when the variation of distance values of points within a segmented object is small corresponding to the object's distance to the viewer.

In the current hardware implementation, the point's orientation is not considered when evaluating the point's screen size, as can be seen from the equation 1. We expect that future graphics hardware will eventually address this issue.

5.3 Visibility Culling

We perform both back face culling and view frustum culling for speeding up the rendering. The back face culling is performed using the calculated point normals. To perform view frustum culling, a bounding box is calculated for each node in the data hierarchy (section 4.4). During the rendering, we perform efficient view frustum culling before sending points to hardware. Starting from the root node, we test the intersection between its bounding box and the view frustum. If the bounding box is totally outside the view frustum, all points inside the box are discarded; otherwise, if it is totally inside the view frustum, all points are rendered. If there is only partial intersection between the bounding box and the view frustum, we then test lower level nodes in the data hierarchy until leaf nodes are reached; they then are either discarded or rendered.

6 Results

We have implemented our scanning pipeline on an 800MHz Pentium III PC with DirectX 8.1 under Windows 2000. The main memory is 1 GB. We used an nVidia GeForce4 Ti 4600 graphics card with 128MB video memory. All images are rendered using 800×600 resolution.

We demonstrate the rendering quality of our system in Figure 5 and 6. Even for close-up views (e.g., Figure 5b and Figure 6c), the objects have a solid appearance. In Figure 5b, the Fraser Hall building is pulled close to about 20m from the camera although it was originally scanned from about 100m away. The rendering quality can be better appreciated by comparing Figure 6(d) and (e), zoom-ins of the marked region in Figure 6(c). Figure 6(d) is rendered using simplest points without splatting, leading to holes and incorrect visibility (back points leak through front points), while Figure 6(e) is rendered using point sprite splatting, where gaps are filled and the texts on the wall become clearly readable.

Table 1: Rendering performance (image resolution: 800x600)

	View Culling	# of Points Rendered	# of Points Discarded	fps
Figure 5a	Yes	531,490	787,724	16.0
	No	1,319,214	0	7.1
Figure 5b	Yes	141,628	1,177,586	22.6
	No	1,319,214	0	6.6
Figure 6b	Yes	450,593	867,310	16.9
	No	1,317,903	0	7.1
Figure 6c	Yes	565,472	752,431	14.1
	No	1,317,903	0	7.1

As we try to leverage graphics hardware to evaluate points' screen projection sizes, we are specifying one 3D size for a group of points, as discussed in 5.2. This approximation performs well for distant objects, but is susceptible to unwanted artifacts for close objects. This is demonstrated in Figure 5a. As we can see in the green circle area, holes appear between points. Another phenomenon is the obvious boundary between two regions, each of which uses a different averaged 3D point size. Even if we systematically increase the average point size to avoid holes (e.g., by setting the 'average' size as the largest one in the region), the boundary between adjacent regions will still be visible as the average point size changes. The ultimate solution to this problem is to calculate each point's screen size using its own 3D point size, a feature we hope for in new-generation commodity hardware.

Finally, we demonstrate the rendering efficiency of our system in Table 1. The performance is evaluated when interacting with scenes in Figures 5 and 6. The frame rate is calculated based on time spent at each frame. Even though the speed is reported for four specific frames, they are adequately representative and demonstrate the range of frame rate that we normally experience. Our system can sustain comfortable interactivity when exploring fairly large environments (about 2 million points) on middle-level hardware. We also demonstrate the effectiveness of performing view frustum culling. An average of a two times speed-up has been achieved. When the environment gets larger, this approach will become more effective.

7 Conclusions and Future Work

We have presented a scanning system for large outdoor environment acquisition employing one of the most advanced scanning devices today. To cope with the complexity of such scenes, we have proposed and experimented with a point-based modeling approach. In particular, we have demonstrated an interactive segmentation and data/object organiza-

tion tool that can efficiently build a hierarchical representation of scanned environment. Given an unprecedented data size and dynamic scanning scenario, this data organization is extremely effective and important for future object indexing, updating, and editing. We have also investigated novel methods of interactively rendering high-quality large point-based scenes consisting of millions of points. This is achieved partially by leveraging new features of current commodity graphics hardware.

The above efforts have formed the framework for our future research in this domain. We already have a few investigations underway on different aspects of the system. First, we will extend the purely point-based modeling to a hybrid modeling using both points and polygons [2]. The polygons are naturally more efficient for representing flat surfaces, while points are generally more efficient for high feature surfaces. In the same direction, we will investigate ways of optimizing point-based models for efficient modeling,. Second, we will develop tools for fixing some of the holes. Again we emphasize the development of an array of interactive tools for tasks ranging from simple transformation and coping to editing to texture synthesis. Third, to further speed up rendering, we will incorporate efficient occlusion culling methods (e.g., [4]) to our rendering framework. We will also explore levels of detail in addition to our current data hierarchy.

8 Acknowledgements

Our thanks to Minh X. Nguyen and Xiaoru Yuan for numerous discussions.

Support for this work has included a Computer Science Department Start-Up Grant and a Grant-in-Aid of Research, Artistry, and Scholarship, 2002-2003, all from the University of Minnesota; University of Minnesota Digital Technology Center Seed Grant 2002, Ted & Linda Johnson Donation, and NSF CAREER ACI-0238486. This work was supported also in part by the Army High Performance Computing Research Center under the auspices of the Department of the Army, Army Research Laboratory cooperative agreement number DAAD19-01-2-0014. Its content does not necessarily reflect the position or the policy of this agency, and no official endorsement should be inferred.

References

[1] F. Bernardini and H. Rushmeier. The 3D model acquisition pipeline. In *Computer Graphics Forum*, volume 21(2), pages 149–172. 2002.

[2] B. Chen and M. X. Nguyen. POP: A hybrid point and polygon rendering system for large data. *Proc. of IEEE Visualization '01*, pages 45–52, Oct. 2001.

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

[4] 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, 2001.

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

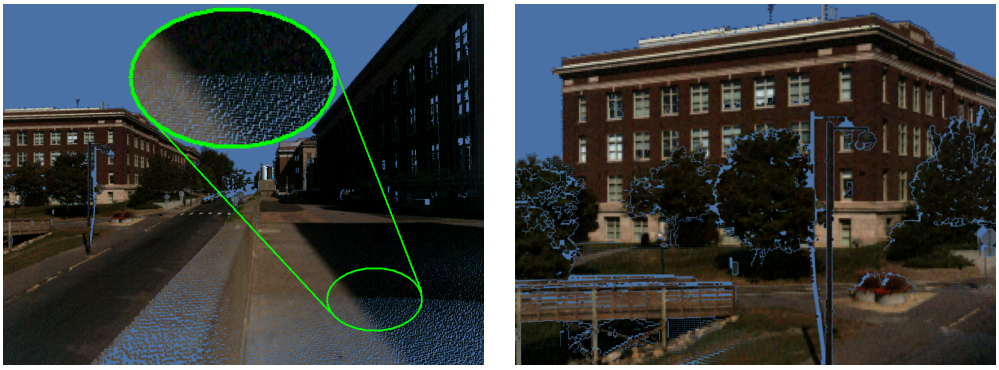
[6] D. K. McAllister, L. F. Nyland, V. Popescu, A. Lastra, and C. McCue. Real-time rendering of real world environnements. In *Rendering Techniques '99*, Eurographics, pages 145–160, 1999.

[7] L. Ren, H. Pfister, and M. Zwicker. Object space EWA surface splatting: A hardware accelerated approach to high quality point rendering. In *Proceedings of Eurographics 2002*.

[8] S. Teller. Toward urban model acquisition from geo-located images. In *Proceedings of the Conference on Computer Graphics and Applications (PacificGraphics'98)*, pages 45–52, Oct. 26–29 1998.

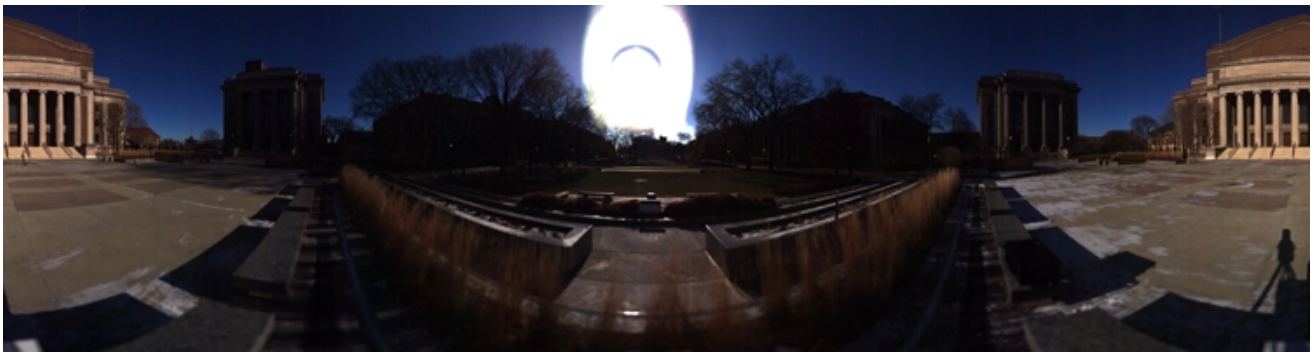
[9] J. Wang and M. M. Oliveira. Improved scene reconstruction from range images. In *Proceedings of Eurographics 2002*.

[10] M. Zwicker, H. Pfister, J. van Baar, and M. Gross. Surface splatting. In *SIGGRAPH 2001 Conference Proceedings*, pages 371–378, 2001.

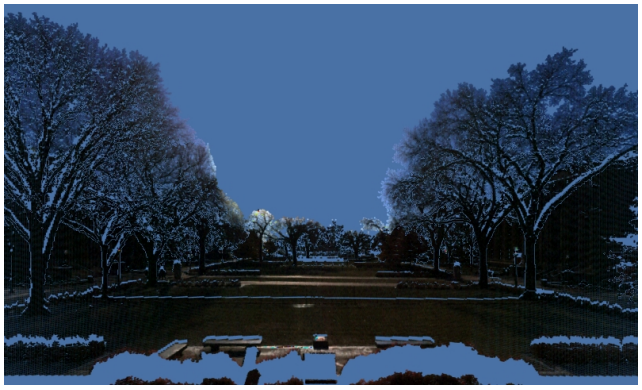


(a) (b)

Figure 5: *Two novel views of Walter Library and Fraser Hall (1,319,214 points and* 800×600 *image resolution): (a) using averaged 3D point size for each region leads to visible boundary between regions and/or holes (within green circles). The region segmentation on the ground is shown in Figure 2j; (b) a close-up view of Fraser Hall (segmented in Figure 2(a-h)).*

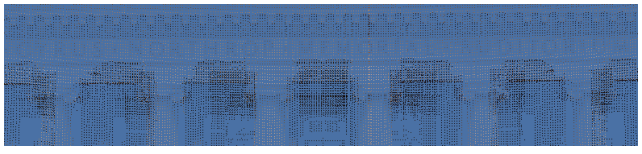


(a)



(b)

(c)



(d)

(e)

Figure 6: *Northrop Mall (1,317,903 points and* 800×600 *resolution): (a) original scanned panoramic image (only color image shown); (b) trees and grounds (looking south); (c) Northrop Auditorium (looking north); (d) rendering using raw points of marked area of (c) with holes and incorrect visibility (back points leaking through front points); (e) rendering using point sprites of marked area of (c) clearly depicting solid appearance (readable texts on wall) and correct visibility.*