To appear in NPAR 2004

Stylized Rendering of 3D Scanned Real World Environments

Hui Xu Baoquan Chen

University of Minnesota at Twin Cities *

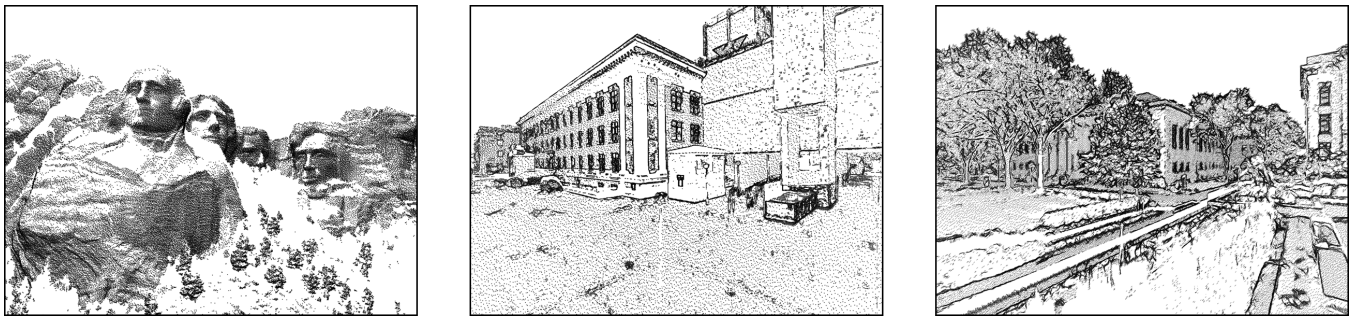


Figure 1: Stylized rendering of 3D scanned real world environments using representative sketchy styles for feature illustration (from left to right: stippling (point sprites), textured strokes + stippling, and textured strokes + hatching (short line segments)).

Abstract

This paper presents an interactive non-photorealistic rendering (NPR) system that stylizes and renders outdoor scenes captured by 3D laser scanning. In order to deal with the large size, complexity and inherent incompleteness of data obtained from outdoor scans, our system represents outdoor scenes using points instead of traditional polygons. Algorithms are then developed to extract, stylize and render features from this point representation. In addition to conveying various NPR styles, our system also promises consistency in animation by maintaining stroke coherence and density. We achieve NPR of large data at interactive rates by designing novel data structures and algorithms as well as leveraging new features of commodity graphics hardware.

CR Categories: I.3.3 [COMPUTER GRAPHICS]: Picture/Image Generation—Digitizing and scanning

Keywords: non-photorealistic rendering, multi-resolution, point-based rendering, interactive 3D graphics, 3D scanning

1 Motivation and Introduction

Recently, researchers have shown increasing interest in capturing and processing real-world 3D objects and scenes. Most of the current scanning systems, driven by applications in entertainment, telepresence and documentation of historical sites and objects, strive to enhance the realism of graphic representations and offer the ultimate realistic virtual experience [McAllister et al. 1999; Levoy et al. 2000; Bernardini and Rushmeier 2002]. In this paper we present a system that instead strives to generate stylized rendering of scanned real world scenes, especially large outdoor environments.

Our main motivation for this work is the facilitation of architectural design. A good architectural design idea should respond to its urban or natural environment. Visualizing the site digitally in its 3D form allows designers to conceptualize and evaluate new designs against their backdrops throughout the entire design process. In the early stages of conceptual design, architects often desire 3D visualization of sites with a controlled degree of abstraction and imprecision which spurs on new design ideas in the context of the site. NPR is an effective mechanism to achieve this goal. Furthermore, for the general purpose of depicting 3D scanned environments, stylized rendering helps visualize or hide the inherent uncertainty in the data, making it less distracting. Such uncertainty is caused by hardware limitations and constraints in outdoor scanning. Rendering an environment in non-photorealistic styles automatically conveys a sense of imprecision, therefore lessening the perceived impact of unwanted artifacts.

Most of the existing scanning systems generate polygon meshes out of scanned point clouds. However, this approach has several shortcomings when applied to outdoor scanning data. The main issues concern the size and quality of such generated polygon meshes, which greatly affect the quality and speed of rendering. First, unwanted artifacts like non-existing facets and jagged object boundaries may be produced by existing polygon construction algorithms due to the excessiveness, ambiguity and fragmented nature of the point clouds that represent outdoor environment scans [McAllister et al. 1999]. Second, an excessive number of polygons may be generated due to the large size of outdoor environments. Third, such algorithms used by previous scanning systems are irrelevant or ineffective to our problem domain because these systems rely on an accurate and complete input, whereas our system deals with outdoor scans of significant uncertainty and incompleteness. To overcome these limitations, we have built point-based representations in our system and developed an array of algorithms to effectively stylize and efficiently render them.

Our system offers a set of tools that can generate various degrees

*Email:{hxu, baoquan}@cs.umn.edu

and styles of abstraction. Visual attributes can be changed through interactive operations. Overall, our system offers four main features:

1. *Effective feature extraction*: Features for NPR illustrations are efficiently and effectively extracted from the large and uncertain data sets of outdoor scans.
2. *Flexibility in stylization and rendering*: Various NPR styles are supported and can be easily changed during navigation.
3. *Coherent animation*: Smooth animation and a consistent NPR quality (e.g., stroke coherence and stroke density consistency) are provided during navigation.
4. *Interactive navigation*: Easy navigation through scenes of millions of primitives on a regular PC using commodity graphics hardware is made possible.

The rest of the paper is organized as follows. We first discuss prior work (Section 2) in this area, and then briefly describe our data acquisition and processing pipeline (Section 3). After giving an overview of our NPR system (Section 4), we introduce two main operations involved, namely feature point extraction (Section 5) and illustration (Section 6). We discuss some implementation details, especially regarding hardware implementation (Section 7), present our results (Section 8), and conclude with a discussion and plans for future work (Section 9).

2 Prior Work

3D Scanning and Processing:

To process scanned data, most existing systems reconstruct polygon meshes from scanned sample points, extract corresponding texture maps out of camera-taken images, and then render textured polygons on conventional graphics hardware to achieve photorealism [Bernardini and Rushmeier 2002; Levoy et al. 2000]. The justification for constructing polygons has been the rendering support offered by available graphics hardware. However, this often results in an excessive number of polygons. The high resolution necessary for scanning large environments leads to a high polygon-per-pixel ratio during the rendering of novel views. Furthermore, fitting meshes into points, especially those of natural phenomena, may create unwanted artifacts by producing non-existing structures or jagged boundaries between objects [McAllister et al. 1999].

Point-Based Rendering:

Using points as alternative modelling primitives has been explored for over a decade and has recently received increasing attention. Points have been shown to be advantageous over polygons when representing highly detailed features [Grossman and Dally 1998; Pfister et al. 2000; Rusinkiewicz and Levoy 2000]. Thus as modern graphics hardware provides more support for the rendering of point primitives, point-based rendering will undoubtedly continue to gain popularity. Besides the efficiency points provide for rendering, they are also more flexible primitives for visualizing and modelling large scanned environments, especially for stylized rendering.

Non-Photorealistic Rendering (NPR):

Non-Photorealistic rendering has become an important branch of computer graphics in recent years. Most NPR techniques attempt to create images or virtual worlds visually comparable to renderings produced by artists. Many artwork styles have been explored in the NPR literature, such as pen and ink [Salisbury et al. 1997; Winkenbach and Salesin 1994], painting [Meier 1996], informal sketching [Raskar and Cohen 1999], and charcoal drawing [Cornish et al. 2001]. Cornish et al. summarize several commonalities in NPR techniques for generating some highly varied artistic effects [Cornish et al. 2001]. The first common feature is selecting and placing strokes, ranging from dabs with a paintbrush and streaks of charcoal to lines drawn with a pen or pencil. These strokes are usually placed with some randomness to imitate the nonuniformity in the drawing styles of human artists, but this randomness can cause flickering during animation due to the lack of consistency. Meier [Meier 1996] solved this problem by associating strokes with particles defined on the surface of objects. Since the strokes are associated with actual locations in space, they move smoothly across the screen in a consistent manner as the view point shifts. Many NPR systems have since incorporated this idea. The second common feature is defining the orientation of strokes, which can be defined by either a user-specified vector field [Salisbury et al. 1997], or normal and curvature information from a 3D model [Saito and Takahashi 1990; Meier 1996; Kalnins et al. 2003]. The third common feature is controlling the screen-space density of strokes [Raskar and Cohen 1999]. Too many or too few strokes can create a cluttered effect or fail to convey the underlying shape respectively. In styles such as pen-and-ink, stroke density also controls tone [Praun et al. 2001; Salisbury et al. 1997; Winkenbach and Salesin 1994]. Simply associating strokes with screen-space increases particle density as objects recede into the distance. An adaptive way of changing the density of the particles according to object distance is desirable.

3 Data Acquisition and Processing

We acquire real world environments through laser scanning. Here we only provide a brief introduction to our scanning and modelling system as most of the operations are conventional. Details about some issues are explained in later sections of the paper together with other operations. Our pipeline of the data acquisition and processing system mainly includes acquisition, registration, segmentation, geometry estimation, and model construction.

- *Acquisition* - Our scanning device is the Riegl Inc's LMS-Z360 3D imaging sensor, which can measure a distance up to 200m with 12mm precision. A full panoramic scan ($360^o \times 90^o$) at 3000×792 resolution takes about four minutes and data in the form of range, intensity, and color images is simultaneously obtained as demonstrated in color plate Figure 8.
- *Registration* - Multiple scans are needed to build up a unified representation of a complex environment. A pair of scans are registered in an interactive way as follows. First, a user identifies some common points existing in both scans. Then, a matrix that transforms one scan into the other is computed based on these common points by the least-squares method [Arun et al. 1987]. Finally, this transformation matrix is used as the initial guess for the iterative closest point (ICP) method to refine the alignment [Besl and McKay 1992]. Pair wise registration is used iteratively to align multiple scans.
- *Segmentation* - The scanned data is in the form of point sets. The purpose of segmentation is to divide the point sets into objects such as buildings, trees, etc. Points in each object can be further segmented to represent subparts. In this way, an object hierarchy is established, which can be used for model editing, efficient view frustum culling, geometry estimation etc. In our system, segmentation is done for every scan by user-assisted image processing methods such as edge detection. Specifically, utilizing different information from the color, intensity

and range images improves the accuracy and efficiency of segmentation.

- *Geometry estimation* - Geometric information regarding each point is necessary for operations such as visibility detection, back-face culling, and stroke placement for NPR etc. Each pixel scanned represents a 3D point with a certain size. This size can be estimated based on the range value and the pixel azimuth/altitude angle. We discuss some other geometry estimation aspects like approximating normals in Section 5.
- *Model construction* - Different from most existing scanning systems, we build point-based models as the final environment representation. These hierarchical, unified and non-redundant point-based models are constructed based on the aforementioned procedures of registration, segmentation and geometry estimation.

In summary, our system scans real world environments, merges data obtained from multiple scanning spots, and builds point-based representations with geometric information estimated at every point.

4 System Overview

Similar to most existing NPR systems, our system includes two main steps: feature point *extraction* and *illustration*. We start our process by calculating a "feature degree" for each point based on some criteria, such as data accuracy and/or geometric properties etc. Then, we choose a threshold to obtain a subset of points that are of high feature degree, termed feature points. Thus, feature degree characterizes the probability of a point to be demonstrated as a feature point. Further operations such as detecting directions to place strokes are conducted on the feature points if necessary. Once the features are extracted, we employ strokes of different styles to illustrate them. The fact that the points are grouped into objects is used to facilitate rendering efficiency and quality. For example, we implement efficient view frustum culling by testing if the bounding box of an object is in the current view. We estimate an object's screen projection area and render a suitable number of feature points in order to ensure consistent screen space density. We design and employ a special data structure called a continuous resolution queue to further maintain coherence in animation. Finally, we develop strategies to leverage modern graphics hardware and achieve interactive rendering rates of environments with millions of points.

5 Feature Point Extraction

Feature points are a subset of our point model. The feature points in our system resemble particles in [Meier 1996] for indicating the locations of strokes on an object. The general features we wish to illustrate include object boundaries, sharp corners and high frequency areas. Our observation suggests that such features largely exist in the area where the error in normal estimation varies significantly. For example, the estimated normals of leaves are somewhat random, hence the estimation error changes largely. They are among the features we are likely to illustrate. On the other hand, when estimating the normals of points on a smooth area (e.g., a curved or flat surface etc), the estimation error remains relatively constant. They are the parts we are unlikely to emphasize in the rendering. According to this observation, we define a *feature degree* for every point in our model based on the variation of the normal estimation error. This feature degree represents the likelihood of a point to be selected as a feature point. Although this method is somewhat empirical, our experiments show that it is very effective for feature extraction when combined with other properties such as point color and intensity.

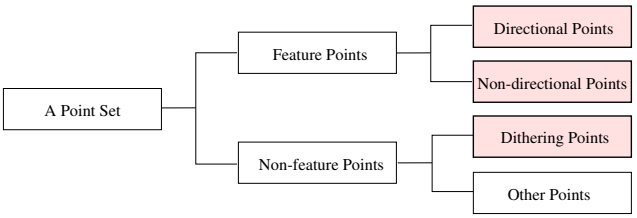


Figure 2: Classification of points (points in the shaded rectangles are used in NPR).

In this section, we first provide the details of point normal estimation. Then, we use the variation of the estimation error to define the feature degree of each point. Next, we extract feature points based on these feature degrees. Finally, we further classify the feature points. Figure 2 shows the overall classification of our point model. Points in the shaded rectangles are used to stylize the scanned scene.

5.1 Normal Estimation

For a point P_0 in our model, we first find the points adjacent to it. Then we calculate the least-squares fitting plane for these points including P_0, and we use the normal of the plane to represent the normal of the point.

The adjacent points can be located efficiently by taking advantage of the fact that input scans are stored as 2D images as shown in color plate Figure 8. With the registration information, we can project P_0 onto every scan and find its neighborhood. The union of these neighborhoods from all scans contains the candidates of the adjacent points of P_0. Then, segmentation information and the distance threshold are used to exclude candidate points belonging to objects or surfaces different from that of P_0.

We denote the adjacent points of P_0 by $P_1, P_2, \cdots, P_n$. We construct a 3×3 positive semi-definite matrix

$$M = \sum_{i=0}^{n} (P_i - \bar{P})(P_i - \bar{P})^{\mathrm{T}}, \tag{1}$$

where $\bar{P} = \frac{1}{n+1} \sum_{i=0}^{n} P_i$ is the centroid of all the points. The eigenvector v corresponding to the smallest eigenvalue of matrix M is the desired direction [Hoppe et al. 1992]. The smallest eigenvalue also indicates the least-squares error of fitting the plane. Notice that the normal can be decided by either $+v$ or $-v$. Here we pick up the one pointing towards the corresponding scanner position to obtain the consistently oriented normals.

5.2 Feature Point Generation

For every point, we calculate a value (the feature degree) between 0.0 and 1.0 to indicate its likelihood of being a feature point.

A criterion used to define this value is based on the variation of the normal estimation error. As discussed earlier in this section, an increase in the variation in the normal estimation error across points in a certain area increases their likelihood of being feature points. To measure the variation of error, we generate 2D "error

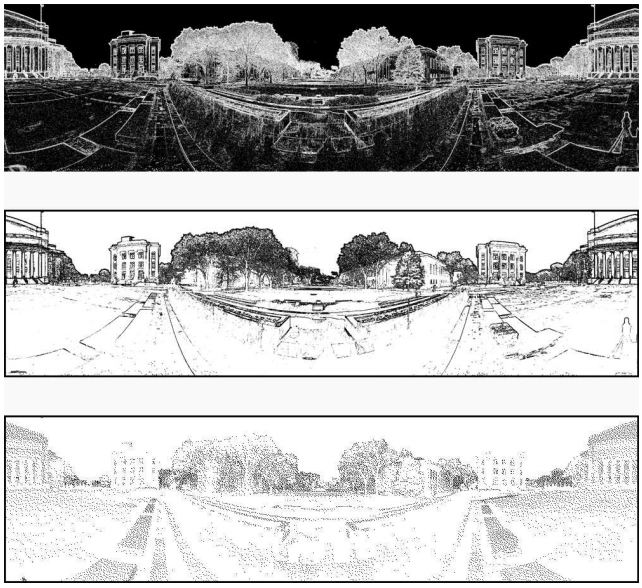


Figure 3: Feature extraction of the panoramic scan shown in color plate Figure 8. From top to bottom: image encoded by the feature degree, the feature points, and dithering points.

maps" of the points based on their 2D coordinates in the original scans. Then we process them using an edge detection filter. The results are scaled from 0.0 to 1.0 and could be defined directly as the feature degrees.

The feature degree defined as above is only based on the geometric information of the points. However, this is not sufficient to describe other features related to the color or intensity of the points. For example, the above process does not detect a boundary between two areas with different colors on a flat surface. Therefore, to extract features more effectively, we take into account the color or intensity. We do this by applying the edge detection filter on the color or intensity image to obtain a gray scale map (black is 0.0 and white is 1.0), and then linearly combine it with the feature value obtained from the error map. We use the values in the final combined image as feature degrees. In this way, features related to both the geometry and the color can be described. The gray scale image at the top in Figure 3 shows an example of the encoded feature degrees, where brighter color indicates higher feature degree.

Once the feature degree for each point is decided, we pick a threshold between 0.0 and 1.0 to extract the feature points. If the feature degree of a point is greater than the threshold, then it is a feature point, otherwise it is a non-feature point.

Although we emphasize feature points, non-feature points are also important to maintain shading patterns. Since rendering too many non-feature points would interfere with the NPR effect, we select as few of them as necessary to approximate the shading tones. To do this, we apply a dithering method on the set of non-feature points to extract a subset called dithering points. It is worth noticing that a direct dithering method would generate dithering points of uniform distribution in the 2D space, but non-uniform distribution in 3D space. Therefore, we combine information from the range image to solve this problem. We apply denser dithering patterns to pixels with higher range values, and sparser dithering patterns to those with smaller range values. A strategy to simplify this procedure is that we first blend the color image with the range image by treating the range value as a scalar or weight to adjust the RGB value of the pixel on the color image. We then apply the standard dither algorithm on the blended image. In this way, the range value of the non-feature point is automatically considered.

The two bottom images in Figure 3 show examples of the extracted feature points (middle) and the dithering points (bottom) respectively.

5.3 Feature Point Classification

Not all the feature points belong to meaningful edges that we intend to illustrate using directional strokes. For example, even though points representing wavering leaves of a tree are likely to be detected as feature points, they unlikely demonstrate consistent directions among multiple scans. Therefore, we wish to deemphasize their orientations. On the other hand, for feature points representing tree trunks and building edges which do demonstrate consistent orientation, we must compute their on-screen directions to orient the strokes to form continuous edges on object boundaries. Because of this, we further classify feature points and computer their orientations if necessary.

The method to address this issue is as follows. As discussed in Section 5.1, for a given feature point P_0, we can efficiently locate its adjacent feature points $P_1, P_2, \cdots, P_n$. We then fit a 3D line to $\{P_i, i = 0, 1, \cdots, n\}$ using the least-squares method. The errors are measured orthogonal to the proposed line. This line indicates the potential direction to place the stroke, while the least-squares error indicates the likelihood of its placement.

Let $\bar{P} = \frac{1}{n+1}\sum_{i=0}^{n} P_i$ be the centroid of the points. The least-squares fitting line can be calculated by constructing a 3×3 matrix

$$M' = \sigma I - \sum_{i=0}^{n} (P_i - \bar{P})(P_i - \bar{P})^{\mathrm{T}}, \tag{2}$$

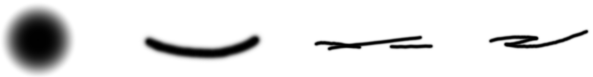
where $\sigma = \sum_{i=0}^{n} (P_i - \bar{P})^{\mathrm{T}}(P_i - \bar{P})$ is a scalar. Similar to the plane fit, the desired line direction is the eigenvector corresponding to the smallest eigenvalue (in absolute value) of matrix M', which also indicates the error of fitting this line.

Thresholds are then set to classify feature points based on the error of fitting the stroke directions. In this paper, we use one threshold to classify them into two categories: directional feature points and non-directional feature points.

6 Feature Point Illustration

Various NPR styles are achieved by illustrating feature points using different graphics primitives. The directional points are emphasized as long textured strokes while non-directional and dithering points are generally illustrated as short line segments of a constant direction or even points (point sprites). In our system, dithering points are rendered in the same way as non-directional points. Therefore, we only discuss the illustration of feature points (i.e., directional points and non-directional points).

We first present our NPR pipeline. We then introduce the graphics primitives used in this paper to illustrate feature points. Finally, we demonstrate the strategies used to ensure coherent animation and consistent stroke density.



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

Figure 4: Alpha textures: (a) point sprite; (b)-(d) stroke examples.

6.1 General Rendering Pipeline

Our system employs programmable graphics hardware and conducts a two-pass rendering pipeline to achieve interactive rendering rates. The reason for using two passes is to ensure correct visibility and alpha blending. The first pass generates a depth buffer and the second pass renders the points into the frame buffer.

In the first pass, a depth buffer called visibility mask is generated by rendering all of the points in the scene as opaque disks (a.k.a. visibility splatting [Pfister et al. 2000]). In this technique, as adjacent points are required to partially overlap with each other to avoid holes, the on-screen projection size of a point needs to be calculated with some degree of precision. The 3D point size estimated in section 3 is used for this purpose. Based on the size of this point, its distance from the viewpoint, the field of view, and the screen resolution, we decide the radius of its on-screen projection. Since the projection size is view-dependent and needs to be recomputed for every frame, we implement this calculation in vertex shaders.

In the second pass, we render only those points that we wish to illustrate. The depth buffer is initialized using the visibility mask, and its updating is disabled. In this way, correct visibility can be guaranteed even though only a subset of the points are rendered. At the same time, since the feature points are demonstrated by textured strokes, alpha blending is enabled to lessen artifacts. However, only points passing through depth testing are blended with the frame buffer during rasterization. Thus, one implementation issue is to prevent discarding of a point on the visible surface. To ensure that, a small offset value is added to every depth value in the depth buffer. This is efficiently implemented by offsetting every point along the viewing ray by a small distance in vertex shaders during the first pass [Ren et al. 2002].

Numerous strategies have been proposed to improve the rendering efficiency and quality. We perform view frustum culling using the pre-calculated bounding boxes of objects. To guarantee coherent NPR animation while maintaining consistent tones in our system, we have designed a new point-based multi-resolution method to build a "*continuous resolution queue*". Continuous levels of detail can be selected from this model. In other words, points are *progressively* added or removed when changing between levels. The model is also designed in a way so that it can be efficiently implemented in hardware.

6.2 Feature Point Stylization

In the second pass, feature points can be stylized using different primitives. In this paper, three primitives are used: point sprites, line segments, and textured strokes.

With hardware support, using point sprites is a very efficient way to render points of variable size and with an applied texture map. To avoid disk-like points, we approximate screen space splatting. We apply a pre-computed splatting filter kernel to the point sprites. Figure 4(a) illustrates a Gaussian splatting filter. Point sprites are usually applied on non-directional points, but they could also be used for directional points when they are to be softened intentionally.

Rendering feature points as line segments is essential for simulating sketching styles in NPR. For directional points, the line segments are drawn along their stroke directions calculated in Section 5.3. For non-directional points, shorter line segments are drawn along a pre-defined constant screen direction.

Different stroke styles are pre-designed and applied to feature points as textures. Figure 4 (b)-(d) show some examples of textured strokes. Similar to line segments, the textured strokes are oriented along their stroke direction for directional points. For non-directional points, the textured strokes are oriented along a pre-defined constant screen direction. The combination of these primitives leads to various artistic styles. We present our results in Section 8.

6.3 Continuous Resolution Queue

When an object is rendered during navigation, the screen density of feature points of the object can change dramatically with their distance from the viewpoint. This leads to different tones. Too many strokes will create a darker drawing with a completely different look [Salisbury et al. 1997]. Since we wish to maintain a consistent density of strokes, the number of rendered feature points has to be dynamically changed during navigation.

We build a multi-resolution representation of the model to solve this problem. In this approach, a level is dynamically selected depending on the screen projection size of an object (the bounding box of the object is pre-computed and used to estimate this screen size). One other issue that needs to be addressed is maintenance of coherence in animation. A further strategy is needed to ensure that the set of points is incrementally updated from one level to another. To guarantee this, representations at lower levels (with higher resolution) should include points in representations at higher levels (with lower resolution), so that when the level changes, only a subset of points are either added (when moving to a lower level) or removed (when moving to a higher level). However, this may still cause noticeable flickering as a group of points are suddenly added or removed if only a discrete set of levels is pre-generated. To solve this problem, we adopt a *continuous* level of detail structure, where only *one* point is either added or removed when changing from one level to its immediate adjacent level.

To construct this hierarchy, we employ a point randomization approach in which the point samples of an object are randomly selected and stored in a linear buffer during a pre-processing stage. In this way, from the first point to the end of the buffer, a continuous level of detail is naturally defined: when more points are added in, more details are added, until the whole set of points is used. This point model is termed *continuous resolution queue*. During rendering, the on-screen projection area of an object is estimated. Then a corresponding number of points, *always* starting from the beginning of the buffer, are selected. This approach shares some features of the Randomized Z-buffer [Wand et al. 2001]. The difference is that here the initial random distribution of points is pre-generated to ensure a progressive updating of points.

We design this scheme to illustrate feature points in the second rendering pass, but it can also be used in the first rendering pass. However, straightforwardly applying this approach to generate the visibility mask may cause holes, resulting in incorrect visibility as an overlap between a set of randomly selected points on the screen space is not guaranteed. Our solution to this issue is to uniformly

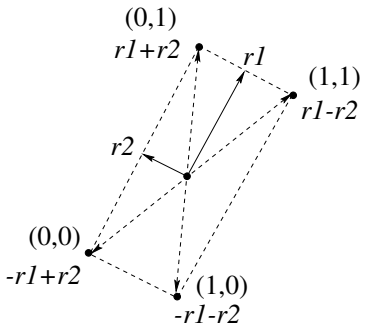


Figure 5: Quad texture coordinates definition.

increase the point size based on the current level of detail. Although theoretically it does not eliminate the problem, our experiments show that holes rarely appear and the rendering speed can be largely increased. Another justification for this approach is that in non-photorealistic rendering, a small error in visibility is not as noticeable as in photorealistic rendering. Therefore, we use the randomized approach in both passes.

7 Implementation Details

When rendering millions of primitives at interactive rates, it is not practical to exchange data between the main memory and the video card at every frame. Therefore, the aim of designing our rendering algorithms is to ensure that the data reside in the video memory. Thus, the dynamic calculation and modification of view-dependent information is done only in programmable shaders. The role of the CPU is restricted to choosing the primitives to be rendered.

In this section, we first present some details of rendering point sprites, line segments and textured strokes. We then discuss implementation of the continuous resolution queue.

7.1 Point Sprites

Point sprites are used for all points in the first rendering pass to generate the visibility mask as well as for the feature points in the second pass. There are two differences between these passes. First, as it is not necessary to enable alpha blending for generating the visibility mask, all points are rendered as opaque disks in the first pass. Secondly, the projection size of each point is precisely calculated in the vertex shaders in the first pass to ensure hole-free surfaces. In the second pass we only use a predefined constant screen size for the feature points. A uniform visual effect and improved rendering efficiency are simultaneously achieved in this manner.

7.2 Line Segments

Line segments are rendered using the primitive type D3DPT_LINELIST in DirectX (or GL_LINES in OpenGL). Therefore, for every feature point, two vertices are sent to the graphics hardware. First, both vertices are initialized to the same value by copying the geometry information directly from the feature point. Next, their 1D texture coordinates are assigned to be 1 and -1 respectively to distinguish them in vertex shaders. The vertex shader program for a line segment offsets these two vertices in opposite directions along a line direction. For a directional point, this line direction is determined by projecting its stroke direction on the screen. For a non-directional point, the line direction is a pre-assigned constant screen direction.

7.3 Textured Strokes

Although point sprites allow texture mapping on a point primitive, the current hardware implementation is relatively limited. Specifically, the texture orientation is very difficult to control. Therefore, we adopt a traditional way to apply the textured stroke using a quad. Similar to the implementation of the line segments, four vertices with same geometry information and different texture coordinates are sent to the graphics hardware. The vertex shader program for textured strokes offsets the four vertices along four different directions respectively as shown in Figure 5. For a directional point, $r1$ is the projection of its stroke direction. For a non-directional point, it is a pre-assigned constant screen direction. $r2$ is the direction perpendicular to $r1$.

7.4 Randomized Vertex Buffer

The vertex buffer supported by commodity graphics cards is ideal for implementing the continuous resolution queue. For the complete point set and the feature point set of an object, their respective continuous resolution queues are pre-computed and stored in vertex buffers. The vertex buffers reside in video memory. To render the object, the CPU only determines the points to be rendered, which takes only $O(1)$ according to our approach. Let the total number of points in the object be N and the number of feature points be N_f. First, the bounding volume of the object is projected on the screen, and the area M (the number of pixels) covered by the projected volume is estimated. Then the ratio $s = M/N$ (Here we assume the object is a single side object like a wall or a surface) is used to approximate the proportion of points necessary. As a result, in the first pass, only the first sN points in the vertex buffer are processed. Similarly, in the second pass only the first sN_f feature points in the feature point vertex buffer are processed.

Our implementation of the continuous resolution queue requires a reasonable and detailed segmentation of objects. For example, if a segmented object is too large, it should be further divided into smaller objects.

8 Results

We have implemented our NPR system using DirectX 9.0 on the nVidia GeForce FX 5800 graphics card with 128MB video memory. Our PC has a 2.4GHz Pentium 4 processor, 1GB of main memory and runs Windows XP. All example images presented in this section are rendered with 1600×1200 resolution.

Figure 6 demonstrates images rendered from point models constructed from different scans and their combinations. In the left image in this figure, we notice that the incompleteness of data from only one scan is very obvious when rendered from a novel view. With the addition of data from more scans from different positions, we obtain a relatively more complete model as shown in the middle (two scans) and the right (three scans) images. Although the model is still incomplete, we were unable to scan for more data because of the lack of accessible scanning spots. This reveals one of the inherent difficulties of outdoor scans that a complete data set is very difficult to acquire. Nevertheless, the point-based NPR shows great advantage in such situations. It conveys the information while

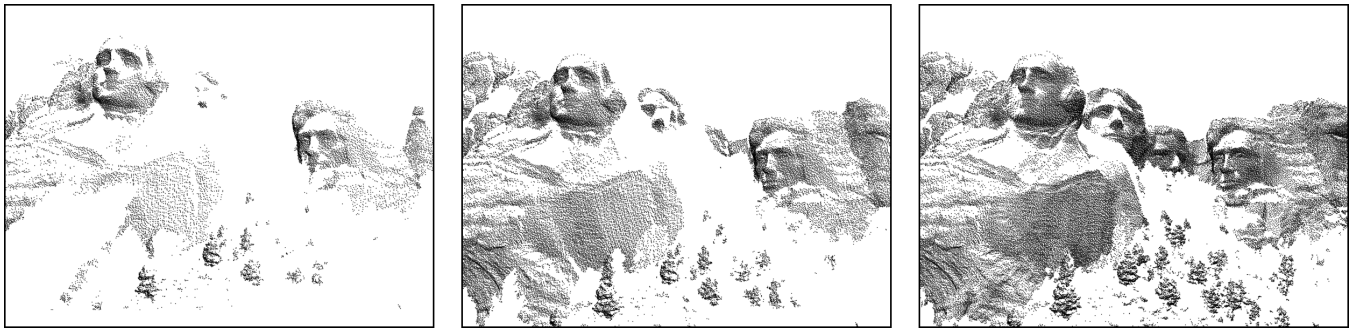


Figure 6: Images rendered from point models constructed from one scan (left), two scans (middle), and three scans (right) respectively. Feature points are combined naturally since they are extracted by taking into account their neighborhoods in 3D space from all scans.

making the incompleteness of data relatively less obvious. The figure also shows how feature points from various scans are combined seamlessly. This happens because they are extracted by taking into account their neighborhoods in 3D space from all scans.

Figure 7 shows different styles implemented in our system. The styles are generated by combining different primitives presented in Section 6.2. Figures 7(a)-(c) shows the rendering of feature points. For different aspects of view, we illustrate them differently. In Figure 7(a), both directional and non-directional points are rendered as point sprites. In Figure 7(b), only directional points are rendered using the textured stroke specified in Figure 4(b), while Figure 7(c) adds non-directional points rendered as short line segments in a constant direction. Figure 7(d) shows the visual effect of Figure 7(c) after rendering additional dithering points. Clearly, different styles cast different impressions of the same environment. Overall, according to our experience, long directional textured strokes or line segments likely convey higher confidence features; while short textured strokes or line segments drawn in constant screen direction likely convey either uncertain features (for non-directional points) or simply the tones of surfaces (for dithered points). Our experiments show that using short and constant screen direction strokes (or line segments) to illustrate dithering points makes flat surfaces more expressive.

Consistent stroke density is demonstrated by comparing the left and the middle images in color plate Figure 9. Without using the continuous resolution queue to progressively update the number of points rendered, the image in color plate Figure 9 (left) is significantly darker than the image (right) rendered at a closer view to the bridge. By using the continuous resolution queue, the middle image shows much more consistent density and less aliasing than the left image.

Finally, we show the rendering efficiency of our system in Table 1. The performance is evaluated when navigating through the left (Scene A) and the right (Scene B) scenes shown in Figure 1. Scene A is constructed from three scans and scene B is constructed from four scans. Three representative frames are reported for each scene. The frame rate is calculated based on time spent at each frame. Even though the speed is reported for these specific frames, they are adequately representative and demonstrate the range of frame rates that we generally experience. This table shows that our system offers interactivity at comfortable rates when exploring large outdoor environments on commodity graphics hardware. The table also demonstrates the effectiveness of performing view frustum culling and using the continuous resolution queue.

Table 1: Rendering performance.
N: the number of points in the scene;
$\mathbf{N_f}$: the number of feature points and dithering points;
$\mathbf{N_c}$: the number of points processed after view frustum culling;
$\mathbf{N_1}$: the number of points actually rendered in the first pass ($N_1 < N_c$ because of the continuous resolution queue);
$\mathbf{N_2}$: the number of feature points and dithering actually rendered in the second pass).

	N	$\mathbf{N_f}$	$\mathbf{N_c}$	$\mathbf{N_1}$	$\mathbf{N_2}$	FPS
A	2,878K	1,442K	854K	725K	419K	33
			1,017K	763K	512K	30
			1,655K	1,023K	572K	28
B	3,955K	1,305k	1,259K	1,140K	621K	20
			1,990K	1,518K	800K	18
			3,198K	2,214K	974K	15

9 Conclusions and Future Work

We have presented an NPR system that features stylized rendering and interactive navigation of large outdoor environments. To cope with the complexity, incompleteness, and uncertainty of such scenes, we have employed point-based approaches. To achieve common features of non-photorealistic rendering, we have developed procedures to extract feature points and have illustrated them using different rendering primitives (or strokes). We have obtained a variety of artistic effects by combining different styles and different features. We have also developed methods to maintain coherent animation and consistent screen space stroke density during navigation. Finally, we have developed strategies to leverage modern commodity graphics hardware to achieve rendering at interactive rates. The results of our experiment show that our system is both efficient, in terms of interactivity, and effective, in terms of feature extraction and illustration.

We are developing new methods to generate more artistic styles based on the current framework. Two of the new styles we are working on are profile lines and painterly styles. First, we are interested in more simplified abstractions using sparser, longer and smoother profile lines. Second, we are planning to create painterly styles and the combination with sketchy styles [Xu et al. 2004].

In our current approach, all the data is loaded into video memory at once, which limits the data set our system can handle. One way of addressing this issue is to take a progressive data loading approach. According to the current navigation direction, we release the video

memory for objects that are unlikely to appear in the view frustum soon, and load those that will appear. To further improve the rendering speed, we plan to incorporate efficient occlusion culling methods like those mentioned in [Klosowski and Silva 2001] in our rendering framework.

Our current system takes a pure point-based approach. Although points are considered to be advantageous over polygons for representing highly detailed features, polygons are more efficient for representing simple and regular geometry. In the future, we plan to adopt a hybrid approach that combines the advantages of both representations. Furthermore, for the purpose of facilitating architectural design, our colleagues from the architecture department have shown great interest of depicting various levels of abstraction for different objects in the scene. We plan to investigate how a combination of photorealistic rendering and non-photorealistic rendering of various styles can be achieved in a single visualization and how each style can be intuitively specified and controlled.

10 Acknowledgements

We thank Andrzej Piotrowski, Nathan Gossett, Amit Shesh, Erik Freed and Michael Koch for helping us scan the data used in this paper and engaging in fruitful discussions. We thank Rushmore National Park Service for park access.

Support for this work includes a University of Minnesota Digital Technology Center Seed Grant 2002, a Ted & Linda Johnson Donation, NSF ACI-0238486 (CAREER), and NSF EIA-0324864 (ITR). This work is also supported 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

ARUN, K., HUANG, T., AND BOLSTEIN, S. 1987. Least-squares fitting of two 3-d point sets. *IEEE Transactions on Pattern Analysis and Machine Intelligence 9*, 5, 698–700.

BERNARDINI, F., AND RUSHMEIER, H. 2002. The 3D model acquisition pipeline. In *Computer Graphics Forum*, vol. 21(2). 149–172.

BESL, P. J., AND MCKAY, N. D. 1992. A method for registration of 3d shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence 14*, 3 (February), 239–256.

CORNISH, D., ROWAN, A., AND LUEBKE, D. 2001. View-dependent particles for interactive non-photorealistic rendering. In *Proceedings of Graphics Interface 2001*, 151–158.

GROSSMAN, J., AND DALLY, W. 1998. Point sampled rendering. *Proc. Eurographics Rendering Workshop*.

HOPPE, H., DEROSE, T., DUCHAMP, T., MCDONALD, J., AND STUETZLE, W. 1992. Surface reconstruction from unorganized points. 71–78.

KALNINS, R. D., DAVIDSON, P. L., MARKOSIAN, L., AND FINKELSTEIN, A. 2003. Coherent stylized silhouettes. *ACM Transactions on Graphics 22*, 3 (July), 856–861.

KLOSOWSKI, J. T., AND SILVA, C. T. 2001. Efficient conservative visibility culling using the prioritized-layered projection algorithm. *IEEE Transactions on Visualization and Computer Graphics 7(4)*, 365–379.

LEVOY, M., PULLI, K., CURLESS, B., RUSINKIEWICZ, S., KOLLER, D., PEREIRA, L., GINZTON, M., ANDERSON, S., DAVIS, J., GINSBERG, J., SHADE, J., AND FULK, D. 2000. The digital michelangelo project: 3D scanning of large statues. In *Siggraph 2000, Computer Graphics Proceedings*, 131–144.

MCALLISTER, D. K., NYLAND, L. F., POPESCU, V., LASTRA, A., AND MCCUE, C. 1999. Real-time rendering of real world environnements. In *Rendering Techniques '99*, Eurographics, 145–160.

MEIER, B. J. 1996. Painterly rendering for animation. In *SIGGRAPH 96 Conference Proceedings*, 477–484.

PFISTER, H., ZWICKER, M., VAN BAAR, J., AND GROSS, M. 2000. Surfels: Surface elements as rendering primitives. In *SIGGRAPH '00 Proc.*, 335–342.

PRAUN, E., HOPPE, H., WEBB, M., AND FINKELSTEIN, A. 2001. Real-time hatching. 579–584.

RASKAR, R., AND COHEN, M. 1999. Image precision silhouette edges (color plate S. 231). In *Proceedings of the Conference on the 1999 Symposium on interactive 3D Graphics*, 135–140.

REN, L., PFISTER, H., AND ZWICKER, M. 2002. Object space EWA surface splatting: A hardware accelerated approach to high quality point rendering. In *Proceedings of Eurographics 2002*.

RUSINKIEWICZ, S., AND LEVOY, M. 2000. QSplat: A multiresolution point rendering system for large meshes. In *SIGGRAPH '00 Proc.*, 343–352.

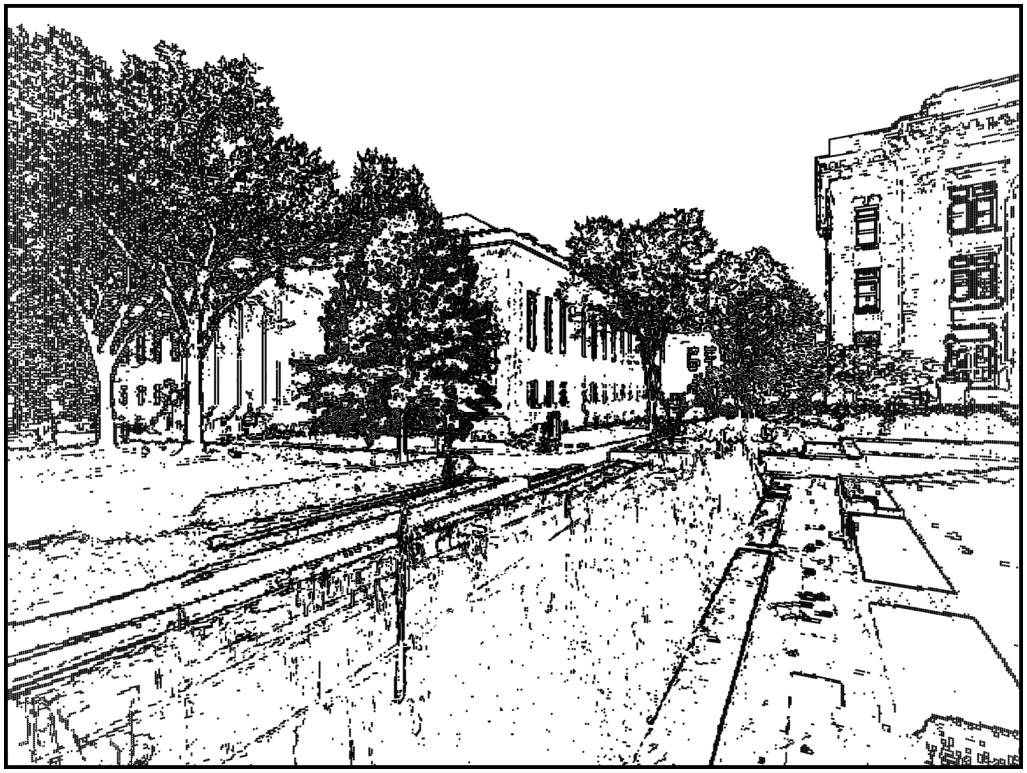
SAITO, T., AND TAKAHASHI, T. 1990. Comprehensible rendering of 3-D shapes. *Computer Graphics 24*, 4, 197–206.

SALISBURY, M. P., WONG, M. T., HUGHES, J. F., AND SALESIN, D. H. 1997. Orientable textures for image-based pen-and-ink illustration. In *SIGGRAPH 97 Conference Proceedings*, 401–406.

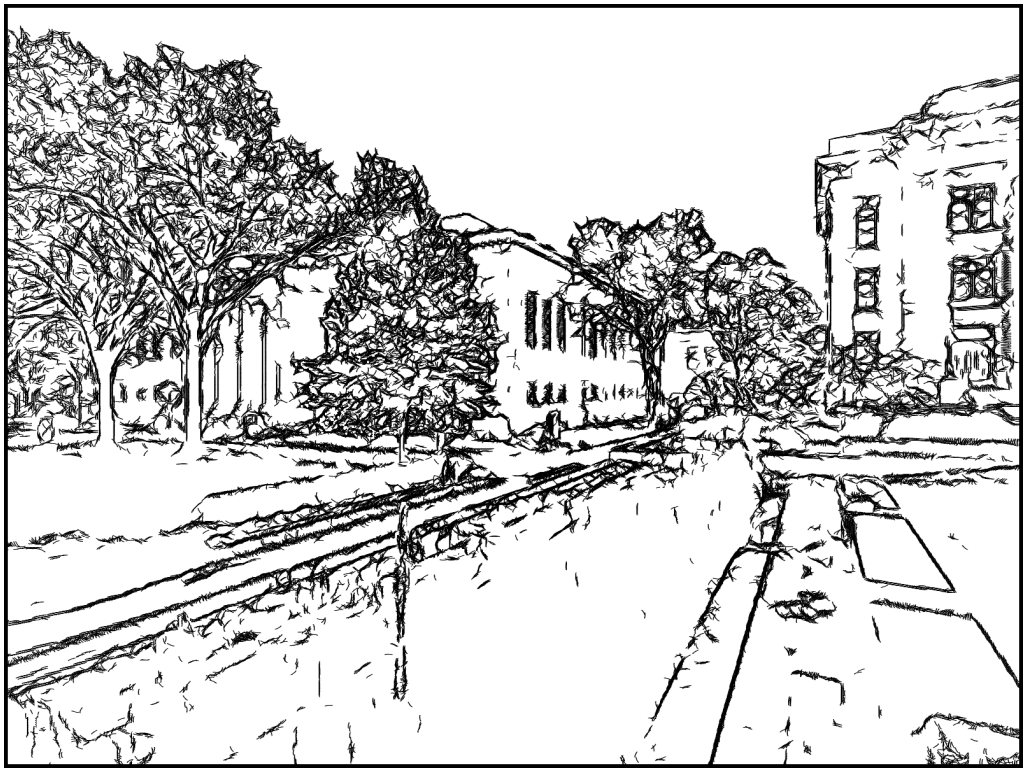
WAND, M., FISCHER, M., PETER, I., AUF DER HEIDE, F. M., AND STRASSER, W. 2001. The randomized z-buffer algorithm: Interactive rendering of highly complex scenes. In *SIGGRAPH 2001, Computer Graphics Proceedings*, ACM Press / ACM SIGGRAPH, E. Fiume, Ed., Annual Conference Series, 361–370.

WINKENBACH, G., AND SALESIN, D. H. 1994. Computer–generated pen–and–ink illustration. In *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, 91–100.

XU, H., GOSSETT, N., AND CHEN, B. 2004. Pointworks: Abstraction and rendering of sparsely scanned outdoor environments. University of Minnesota Technical Report TR-01-04.

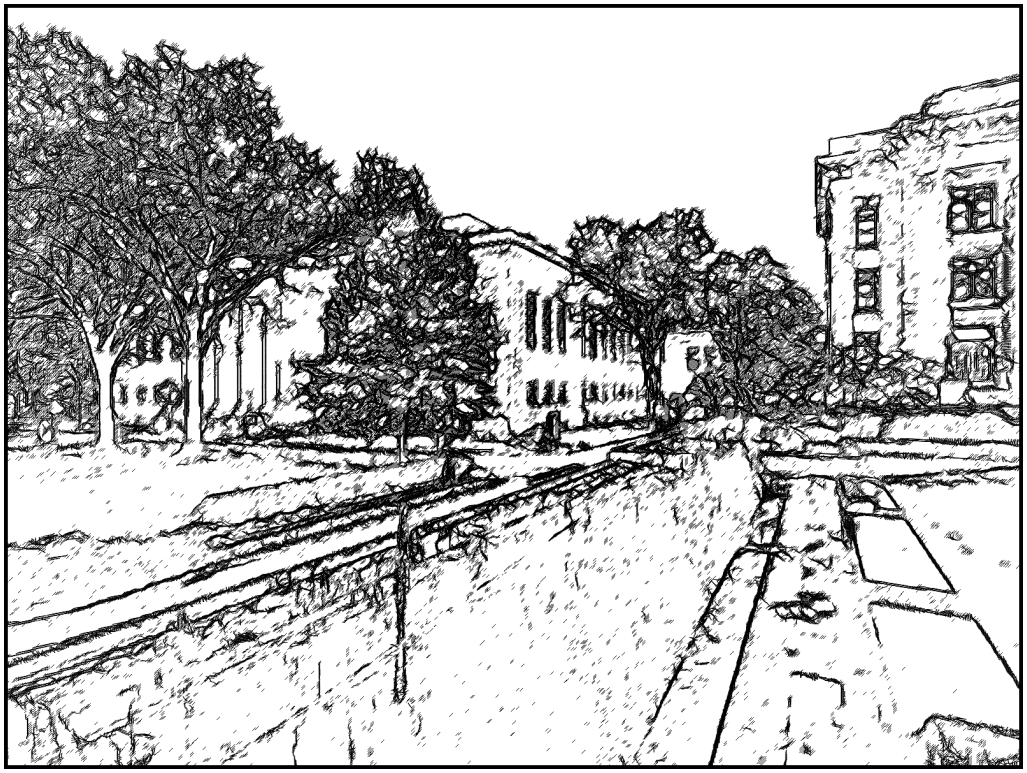


(a)

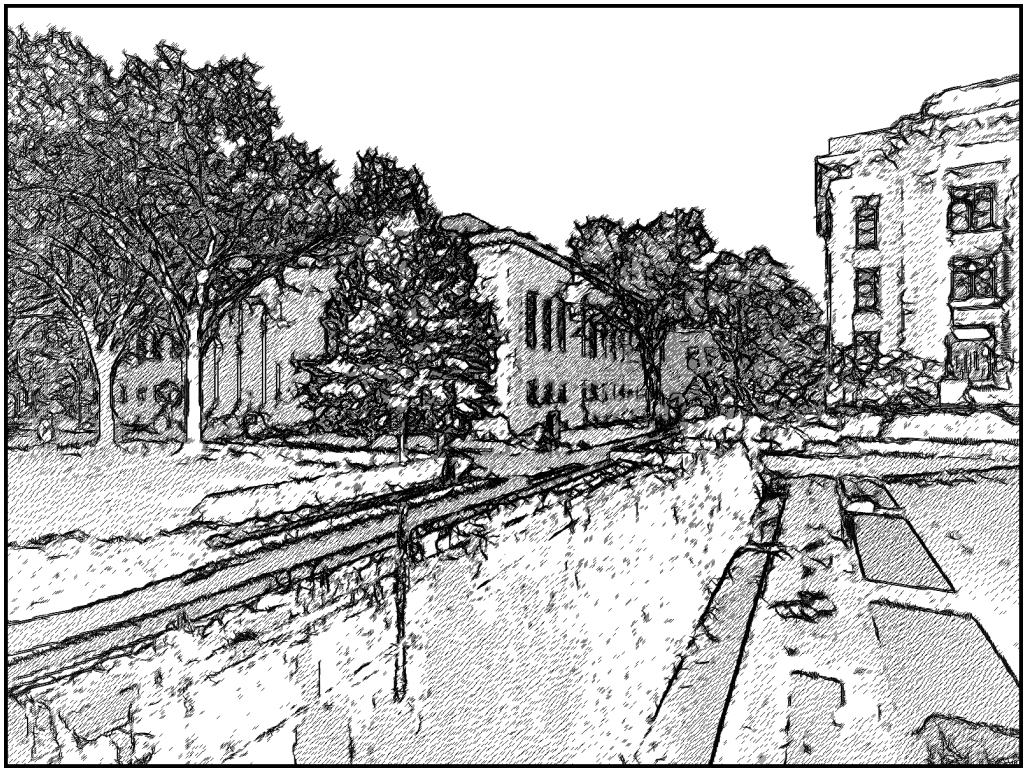


(b)

Figure 7: Rendering the same environment using different styles: (a) point sprites are used for both directional and non-directional points; (b) only directional points are displayed using long textured strokes;



(c)



(d)

Figure 7: (Contd) Rendering the same environment using different styles: (c) non-directional points using short line segments with constant screen direction are added to (b); (d) dithering points using short line segments with constant screen direction are added to (c).

Stylized Rendering of 3D Scanned Real World Environments

Hui Xu, Baoquan Chen

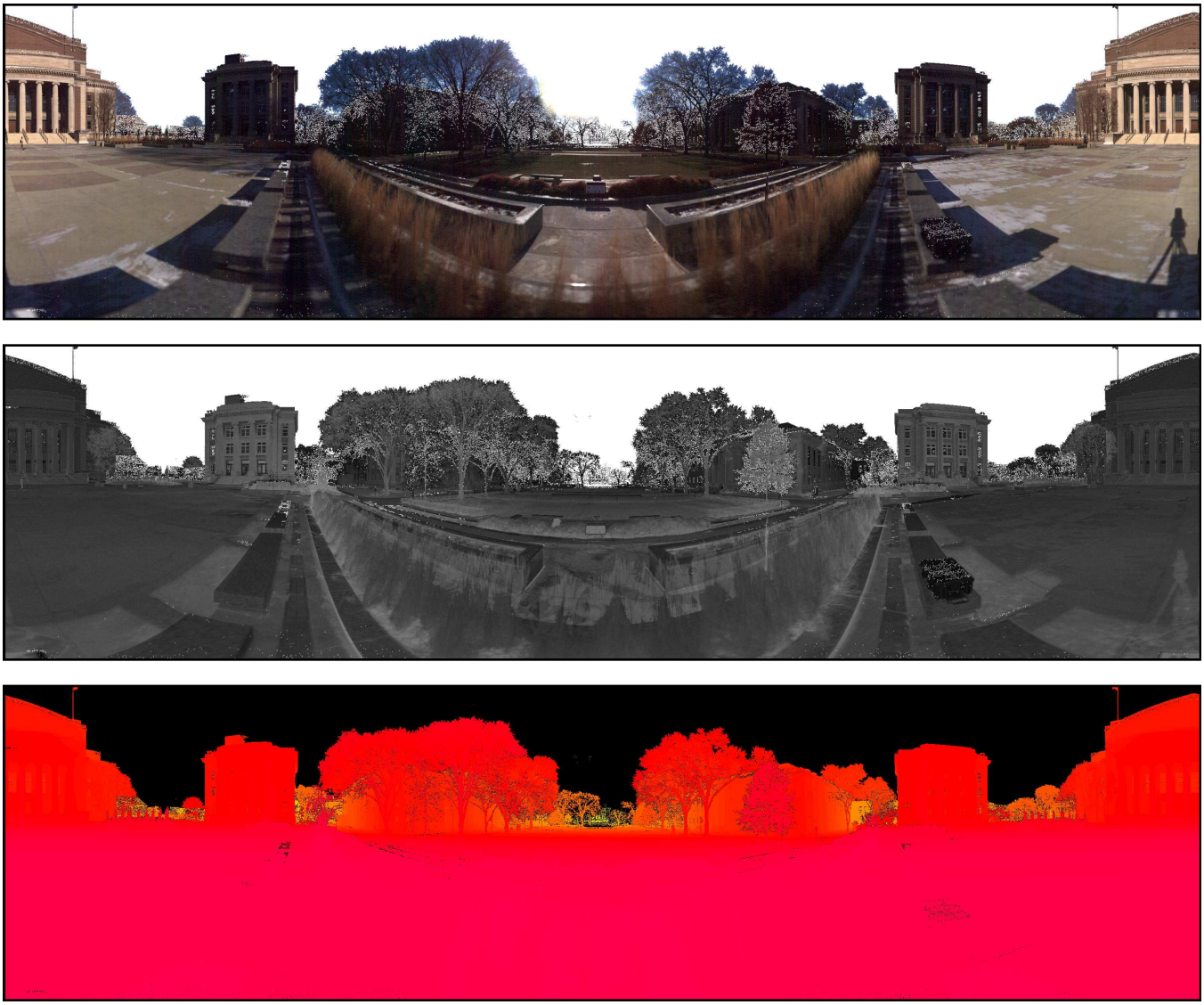


Figure 8: A single panoramic scan of a corner on the University of Minnesota campus (from top to bottom: color image, intensity image, and pseudo-color-encoded range image).

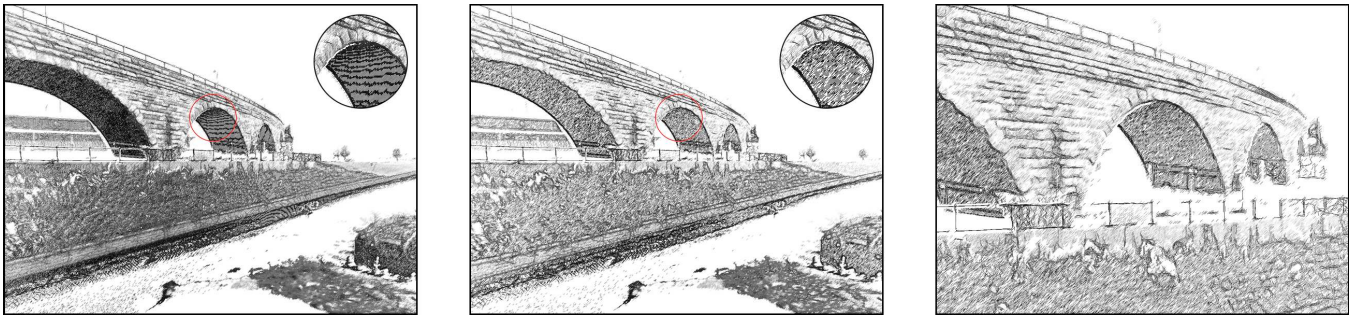


Figure 9: Images rendered without (left) vs. with (middle) the continuous resolution queue. The area marked by the small circle in each image is magnified and shown at the top right corner. A closer view taken during continuous navigation is demonstrated in the right image. The image generated without using the continuous resolution queue is obviously much darker and has obvious artifacts.