**The Visual Computer manuscript No.**
(will be inserted by the editor)

**Adam Runions · Faramarz Samavati · Przemyslaw Prusinkiewicz**

# Ribbons

## A representation for point clouds

**Abstract** Point clouds are usually represented either globally, as surfaces, or locally, as sets of points with small neighbourhoods. We propose an intermediate representation, called *ribbons*, which is obtained by partitioning a point cloud into one-dimensional strips. This representation is well suited to the placement of strokes in non-photorealistic rendering, and can be visualized efficiently using quad strips. Methods for performing hatching, cross hatching, and silhouette renderings are presented. Ribbons also allow for the application of curve-based operations to the point cloud.

**Keywords** point cloud · point based rendering · non-photorealistic rendering · geometric modeling

### 1 Introduction

The digitization of real-world objects often produces point clouds. The increasing availability of point cloud data has made it necessary to develop techniques to process and manipulate unorganized point clouds. Over the past decade point-based modeling and rendering has developed into a new paradigm in computer graphics.

Point-cloud research initially focused on converting point sample data to better understood representations, such as polygon meshes and implicit functions. An alternative approach is to work with point clouds directly, without imposing the topological and geometric constraints of other representations. Most of the current representations can thus be categorized as surface- or point-based. In *surface-based* representations, the point cloud is modeled as a single surface [2,8], as opposed to *point-based* representations that rely entirely on the points as individual objects (perhaps using neighbouring points to determine the normal)[21].

The spectrum from points to surfaces is wide enough to include other, intermediate representations. For example, line drawings play an important role in artistic representations of an object [7,9]. These lines represent a connectivity beyond that of point samples, but less extensive than that of surfaces, which may be an over-representation for this task.

In this work, we propose *ribbons* as an intermediate representation of object geometry that lies between point-based and surface-based representations. Ribbons are particularly useful in non-photorealistic rendering, where objects are represented as collections of lines. We present a data structure for ribbons, an algorithm for converting a point cloud to a ribbon representation, and a method for rendering objects represented by ribbons. This method is inspired by ink-pen illustration techniques such as hatching, cross-hatching, and silhouette extraction [14], but it also introduces an original texture, which is aesthetically appealing in some applications. We also introduce a simple level-of-detail representation based on ribbons, and provide a number of examples demonstrating the proposed techniques.

Our method transforms the point cloud into long one-dimensional strips, called *ribbons*, by connecting points into sequences according to their neighbourhood relations. The resulting representation can be computed efficiently, with the only non-linear operation being a k-nearest-neighbour search at each point in the pre-processing phase. The ribbons may be rendered using OpenGL to obtain standard or non-photorealistic (NPR) renderings of the surface. Additionally, the linear structure of ribbons allows for the direct application of one-dimensional operations, such as forward and reverse subdivision.

The structure of this paper is as follows. Section 2 provides a discussion of related literature on point clouds

Adam Runions
2500 University Dr. NW
Calgary AB, Canada
T2N 1N4
Tel.: +1-403-210-9498
E-mail: runionsa@cpsc.ucalgary.ca

Faramarz Samavati
E-mail: samavati@cpsc.ucalgary.ca

Przemyslaw Prusinkiewicz
E-mail: pwp@cpsc.ucalgary.ca

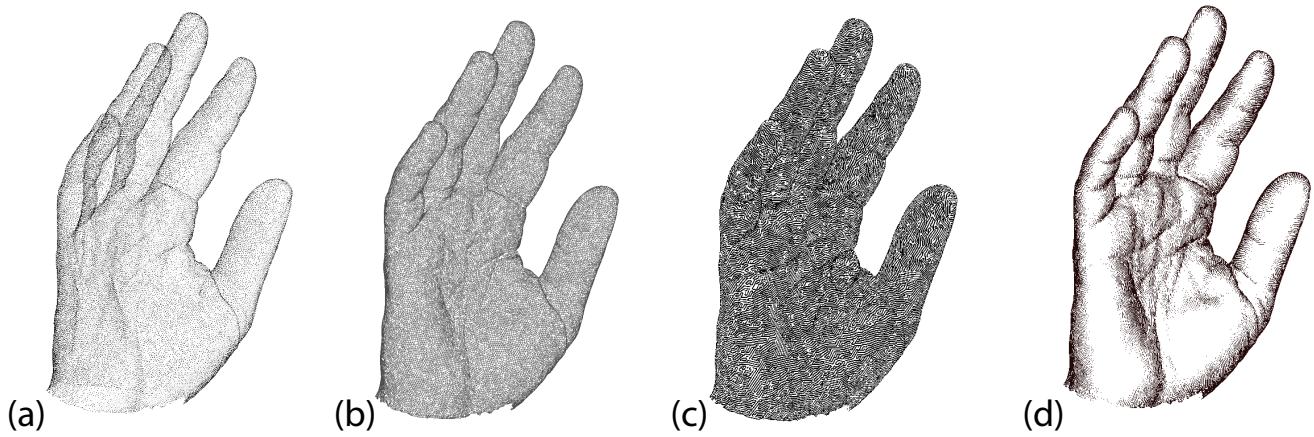


**Fig. 1** Outline of ribbon generation a) Input consists of an initial point cloud b) Connectivity computed using WPCA c) Ribbons generated by our algorithm d) Ribbons rendered using an NPR technique

and NPR. In Section 3 we show how to construct ribbons by first connecting the points (Section 3.1), then partitioning the point cloud into ribbons (Section 3.2). The rendering of ribbons is discussed in Section 4. Applications to stroke placement in NPR are presented and illustrated in Section 5. Finally, in Section 6 we outline future research directions.

## 2 Related Work

### 2.1 Point clouds

Point-cloud representations can be coarsely divided into surface- and point-based. The first category was addressed initially by constructing an implicit representation. Hoppe et al [10] proposed one of earliest methods, in which a distance function was constructed on a point set. Reconstruction of a polygon mesh from point-cloud data has also been implemented using 3D Delaunay triangulations in the Cocone[8] and PowerCrust[2] algorithms.

Some of the first point-based representations were splatting techniques[21, 4]. Splatting generates a local characterization of the point cloud at each point based on a small neighbourhood. The points are then represented by simple geometric primitives, such as ellipses, which make it possible to efficiently render the point cloud without using connectivity.

The method of moving least squares (MLS), proposed by Levin[12], is a projection approach to surface approximation. MLS uses a mapping operator that projects a given point onto a local approximation of a surface, generated from a small neighbourhood of the point. The MLS projective surface has been used to obtain an implicit representation of point cloud data and to perform operations such as denoising. Adamson et al.[1] proposed a similar technique using weighted principal component analysis instead of MLS.

A notable exception to the surface vs point-based characterization of representations is the work of Boubekeur et al. [5], which offers an intermediate representation. This representation partitions the point cloud using an octtree and triangulates each partition. By refining each triangulation using subdivision it is possible to achieve visual continuity and efficiently render the surface using the standard graphics pipeline.

### 2.2 Non-Photorealistic Rendering

Placing strokes on surfaces is an important facet of *Non-Photorealistic Rendering* (NPR). Strokes are used to provide visual cues to the viewer, conveying the structure of the surface in simple terms. Silhouettes, suggestive contours[7], creases, and hatching are often employed to this end.

Silhouettes are a well established NPR visualization technique[18]. Frame coherence of meshes, however, can be an issue. If frame coherence is not maintained, strokes can appear or disappear suddenly when a model is animated, decreasing the visual quality of results. The work of Brosz et al[6] addresses this problem by varying the width and shading of silhouette lines based on the stability of each line.

The automatic hatching or cross-hatching of a surface was examined outside the context of point clouds[18, 9]. In the work of Praun et al[18] and Hertzman et al.[9], hatching is constructed by placing roughly parallel strokes on the surface. Cross hatching is generated by placing two sets of parallel strokes perpendicular to each other. Hertzmann et al. create strokes directly on the surface, whereas Praun et al. use textures to obtain real-time hatching.

The application of NPR to point clouds has been relatively unexplored. Kawata et al.[11] proposes a technique for interactive point-based painterly rendering. Xu

et al. [19] extract feature points but do not construct strokes, and Zakaria et al. propose a method for dithering and silhouette extraction which creates strokes in image space[20]. Pauly et al.[16] propose a method for extracting feature lines, which refines the k-nearest neighbours of the point set down to a small number of feature lines.

## 3 Generating Ribbons

Let $P$ be a point cloud sampled from a 2d manifold surface embedded in $\Re^3$ (Fig.1a). Ribbons partition the point cloud into one-dimensional strips. Maximizing the length of each ribbon and maintaining a relatively consistent direction reduces the number of short or erratic ribbons, which do not characterize the point cloud well. It also serves to improve the results of NPR visualization and one-dimensional operations performed on the ribbons. In this section we propose a method to generate ribbons that works well for these purposes. It is also possible to create ribbons under other constraints, which may lead to different generation methods.

Partitioning $P$ into ribbons relies upon connectivity obtained from a local parameterization computed at each point (§3.1, Fig.1b). $P$ is locally parameterized by performing *weighted principal component analysis* (WPCA) centred on each point taking into account its neighbourhood[1]. The scope of local parameterizations increases as connectivity is established, yielding regions with a parameterization similar to tensor surfaces (§3.1.2).

Given the connectivity of each point, the point cloud is partitioned into ribbons by traversing the connected structure. (§3.2). By following the local parameterization at each point two classes of ribbons, *u-ribbons* or *v-ribbons*, can be constructed (Fig.1c).

With the exception of computing the neighbourhoods of the points all operations are linear in time, and can be implemented efficiently using simple data structures such as queues or linked lists.

### 3.1 Defining connections

#### *3.1.1 Computing the local frame*

The local parameterizations used take the form of coordinate systems, initially computed at each point $p \in P$ using, $N_p$, a local neighbourhood of $p$. To this end a subset of the *k-nearest neighbours* (k-NN) of $p$ is employed[15] (Fig.2b).

We use *principal component analysis*(PCA) to find a coordinate frame for each local parameterization. To include the impact of the distance of points, we employ a weighted form of PCA[1,17]. A WPCA estimate is said to be *centred on* $p \in \Re^3$ if it is computed with $N_p$ as the neighbourhood. For a local weighted function, we use the function employed by Levin[12]

$$w(p_i) = e^{-\frac{\|p-p_i\|}{H_p^2}}$$

where the local scale factor $H_p$ is calculated as the average distance between $p$ and each of the points in $N_p$.

It is possible that the Euclidean distance between $p$ and one of its k-neighbours is quite large. A distant neighbour can distort the parameterization of $N_p$ . To guard against this possibility, we use $H_p$, the local scale factor at $p$. All points in $N_p$ further than $H_p$ from $p$ are removed prior to performing WPCA. As $H_p$ is a local estimate, it changes adaptively across the surface.

Now, $N_p$ is used to perform WPCA centred on $p$ to obtain $\alpha_{p1}$,$\alpha_{p2}$, and $\alpha_{p3}$, the sorted eigenvectors of the weighted covariance matrix. As the point clouds being considered approximate a 3d surface, these vectors have a meaningful geometric interpretation: $\alpha_{p1}$ and $\alpha_{p2}$ estimate the tangent plane and $\alpha_{p3}$ the normal to the point cloud at $p$. By projecting the points on the tangent plane, a local parameterization can be determined. As such, these three vectors thus serve as a basis for a local parameterization of the point cloud. The vector $\alpha_{p1}$ indicates the first coordinate, denoted $u$, and $\alpha_{p2}$ the second coordinate, denoted $v$. Point $p$ and these two vectors also form a coordinate frame. We denote by $l_{p1}$ and $l_{p2}$ the lines that pass through the vectors of this frame.

#### *3.1.2 Alignment of local frames*

Computing a global parameterization for the entire point cloud is difficult, but creating a number of smaller parameterizations with enough structure to generate ribbons is easily addressed due to their linear connectivity. This is accomplished by choosing a point and propagating its parameterization to its neighbours, and in turn to their neighbours and so on until it can be expanded no further. Then another point is chosen and the process repeated. This grows the parameterization at a point to encompass a larger portion of the point cloud. The expanded parameterizations provide locally consistent directions which are followed to generate ribbons. As the local parameterization is represented by local frames this involves aligning the local frames at each $p \in P$.

In order to align the local frames in a linear fashion, the local frames at each point in $P$ are aligned by first picking an unprocessed point randomly from the point cloud (initially all points are unprocessed). Next, using the vectors $\alpha_{h1}$ and $\alpha_{h2}$ the local connectivity of $h$ is computed (Alg.1 lines 7-10). The connectivity of $h$ is given by the points $Left(h)$, $Right(h)$, $Up(h)$ and $Down(h)$.

Once the connectivity of $h$ has been calculated, each point connected to $h$ is processed in order to align their coordinate frames with $h$'s. This helps to maximize the length of each ribbon.

If $k$ is connected to $h$ (one of the Left(h), Right(h), Up(h), Down(h)) but has not yet been processed then

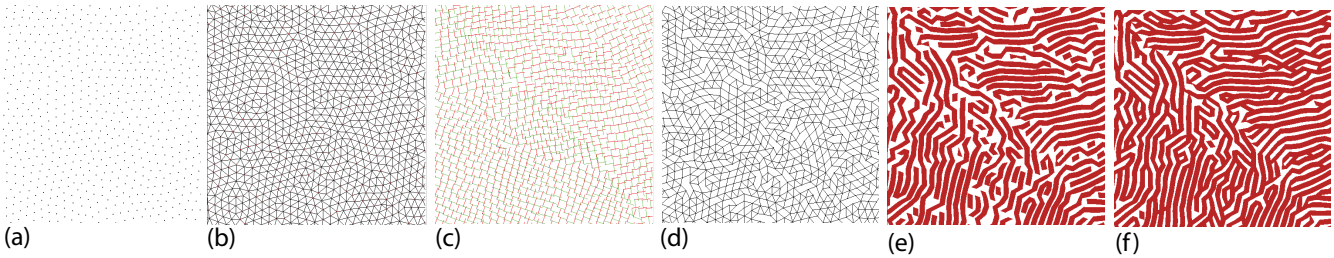


**Fig. 2** Overview of ribbon generation a)Initial points b)$N_p$ of point set c)Local coordinate frames ($\alpha_{p1}$ in orange, $\alpha_{p2}$ in green) d)$Up(p)$,$Down(p)$,$Left(p)$, and $Right(p)$ relations for each point e) Ribbons generated on point set f) Result of rendering ribbons as described in text

it must be processed. To do this it is necessary to first orient $\alpha_{k1}$ and $\alpha_{k2}$ as close to $\alpha_{h1}$ and $\alpha_{h2}$ as possible while keeping both vectors in the tangent plane at $k$. This is done by projecting $\alpha_{h1}$ and $\alpha_{h2}$ onto the tangent plane at $k$ to produce $\alpha_{new1}$ and $\alpha_{new2}$ respectively. A Gram-Schmidt orthogonalization is performed on $\alpha_{new1}$ and $\alpha_{new2}$ to guarantee orthogonality. These vectors are then used in place of $\alpha_{k1}$ and $\alpha_{k2}$ respectively. The coordinate frames are visualized for an example point cloud in Fig.2c.

As $k$ is now consistently oriented with the neighbouring point $h$ its connectivity can be computed, so it is added to the queue. Once all the points connected to $h$ have been processed, the process is repeated for the next point in the queue as described in Alg.1.

This process is repeated starting with an unprocessed point until the entire point cloud has been processed.

```
1  Unprocessed := P
2  While p ∈ Unprocessed do:
3    Q.push(p)
4    Unprocessed := Unprocessed − p
5    While Q contains at least one point do:
6      h := Q.pop()
7      Order N_p along α_h1
8      Calculate Left(h) and Right(h)
9      Order N_p along α_h2
10     Calculate Up(h) and Down(h)
11     S := {Up(h), Down(h), Left(h), Right(h)}
12     For all k ∈ S do:
13       if k ∈ Unprocessed then:
14         Unprocessed := Unprocessed − k
15         Reorient α_k1 and α_k2
16         Q.push(k)
17       End if
18     End For
19   End While
20 End While
```

Algorithm 1: Pseudocode for generating connectivity

### *3.1.3 Creating local connectivity*

The local connectivity at a point $p$ is determined by finding points preceding and following $p$ along the lines $l_{p1}, l_{p2}$ that follow the local parameterization at $p$ (illustrated in Fig.2d).

Let $N_{l_{p1}}$ be the set of points in $N_p$ with Euclidean distance to $l_{p1}$ less than that to $l_{p2}$ or equidistant from both lines (Fig.3(b)). Once $N_p$ has been reduced to $N_{l_{p1}}$ it becomes possible to determine $Left(p)$ and $Right(p)$ by ordering the points along $l_{p1}$ and selecting the points preceding and following $p$ respectively (Fig.3 c-d). If no such point exists then $p$ is selected instead. Determining $Up(p)$ and $Down(p)$ is similar but uses $l_{p2}$ in place of $l_{p1}$ (Fig.3e). It should be noted that the connections between points may not be symmetric (e.g. $Left(Right(p))$ might not equal $p$).

By using the connectivity at each point two functions are defined which are used to create ribbons. The first function $next_{LR}$ takes two points in a progression and produces the next point in the same direction using the local orderings of the points along $l_{p1}$. The second function $next_{UD}$ performs the same function along $l_{p2}$. These functions are essential to the chaining process as they provide the methods necessary to stitch local orderings together.

## 3.2 Partitioning the Point Cloud into Ribbons

By using the generated connectivity the point cloud can be partitioned into ribbons. U-ribbons are produced by following the connectivity provided by $Left(p)$ and $Right(p)$ using $next_{LR}$, and v-ribbons by following the connectivity provided by $Up(p)$ and $Down(p)$ using $next_{UD}$.

Each ribbon is started at some unprocessed point $p \in P$. The ribbon is extended in two stages. First, the ribbon is extended along opposing directions (up and down, or left and right) until unprocessed points can no longer be added.

In the second stage if either end of the ribbon terminates at the end of another ribbon then the two ribbons are linked into a single ribbon. Linking the ribbons serves to connect existing ribbons in order to create longer ribbons, when possible (§3.2.2). The result of the ribbon generation process can be seen in Fig.2e and Fig.1c.

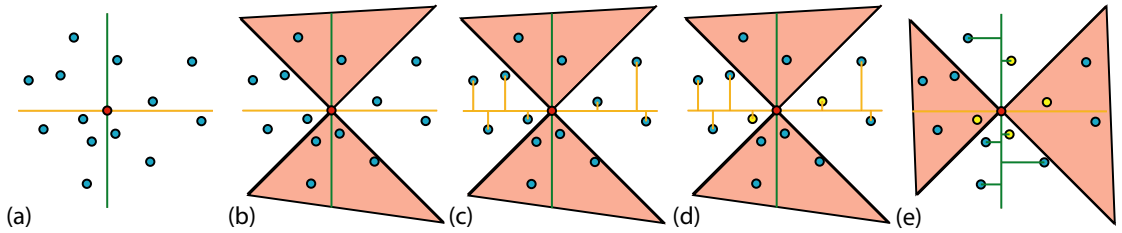


**Fig. 3** Establishing the connectivity at a point $p$ a)The point $p$ is shown in red, its $N_p$ in blue, and the lines $l_{p1}$ and $l_{p2}$ in orange and green, respectively. b) The points closer to $l_{p2}$ than $l_{p1}$ fall in the shaded region and are excluded from consideration c) The remaining points are ordered along $l_{p1}$ d) $Left(p)$ and $Right(p)$ are determined e) The process is then repeated for $l_{p2}$ to establish $Up(p)$ and $Down(p)$

Formally, each ribbon is defined by an ordered set of points and their respective coordinate frames. When considering a single set of ribbons (only the u-ribbons or v-ribbons, not both) the ribbon containing $p$ is labelled $R_p$; this notation is well-defined as each set of ribbons partitions the point cloud. The result is a set of ribbons $R = \{R_1, R_2, \cdots, R_j\}$.

Starting each new ribbon as far from pre-existing ribbons as possible helps to maximize ribbon length by allowing the ribbon more room to extend prior to linking. Unfortunately, finding the point farthest from existing ribbons is an expensive computation; thus the $p$ used to create $R_p$ is chosen randomly from the points still not contained in a ribbon, which decreases the likelihood that points near pre-existing ribbons will be selected.

### *3.2.1 Creating a ribbon*

Each ribbon is generated by staring at $p$ and traversing the connections to neighbouring points. Here, the generation of u-ribbons is described, but modifying this process to create v-ribbons is straightforward and is discussed at the end of this section.

Ribbon $R_p$ is initialized to contain only a single unprocessed point $p$, chosen randomly from $P$. While ribbons are being created each point in $P$ is marked to indicate whether it resides at the end or beginning of a ribbon, resides in a ribbon but not at the end or beginning, or is not yet part of a ribbon. This allows ribbons to be efficiently created.

The process proceeds by adding $Left(p)$ to the front of the ribbon. $next_{LR}$ is then used to move along the $Left()$ and $Right()$ connections until $next_{LR}$ returns a point that has already been processed. The process is repeated in lines 7-10 of Alg.2 starting with the point $Right(p)$ with points added to the end of the ribbon, in order to respect the ordering of ribbons(Fig.4). When Alg. 2 terminates the ribbon $R_p$ is ready to be linked to existing ribbons.

To generate v-ribbons it suffices to repeat the same procedure, but replace $Left(p)$, $Right(p)$, and $next_{LR}$ with $Up(p)$, $Down(p)$, and $next_{UD}$, respectively.

| | |
|---|---|
| 1 | Add p to $R_p$ |
| 2 | p := $Left(p)$ |
| 3 | While p is not part of a ribbon : |
| 4 | Place $p$ at the front of $R_p$ |
| 5 | $p := next_{LR}(R_{p,1}, R_{p,2})$ |
| 6 | End While |
| 7 | $p := Right(R_{p,end})$ |
| 8 | While p is not part of a ribbon: |
| 9 | Place p at the end of $R_p$ |
| 10 | $p := next_{LR}(R_{p,end}, R_{p,end-1})$ |
| 11 | End While |

Algorithm 2: : Pseudocode for generating a single ribbon

### *3.2.2 Linking ribbons*

Once the ribbon $R_p$ is generated it is linked to existing ribbons if it starts or terminates at the end of another ribbon (or both). This is handled by connecting $R_p$ and the adjoining ribbon(s) into a single ribbon by connecting the appropriate ends.

Adjoining ribbons can be identified by examining $Left(R_{p,0})$ and $Right(R_{p,end})$ for u-ribbons and $Up(R_{p,0})$ and $Down(R_{p,end})$ for v-ribbons. Linking of ribbons is a local process, thus if multiple non-intersecting point clouds are present, no ribbon should link them.

## 4 Rendering Ribbons

Although a set of ribbons presents only a partial reconstruction of a surface, the ribbons can be rendered in a manner that approximates visual continuity. The idea of approximating, instead of guaranteeing, visual continuity is also explored in the work of Boubekeur et al[5].

WPCA provides an estimate of the surface normal, but the normal may not be correctly oriented. Computing the correct orientation is a relatively complex and time consuming operation[10,13]. To avoid this, each segment of a ribbon is rendered as three parallel quads (Fig.5). The first quad is translated positively along the normal, and the second negatively along the normal. The quads are given width by moving the corners orthogonally with respect to the vector connecting the two points and the normal at each end of the quad, as is illustrated

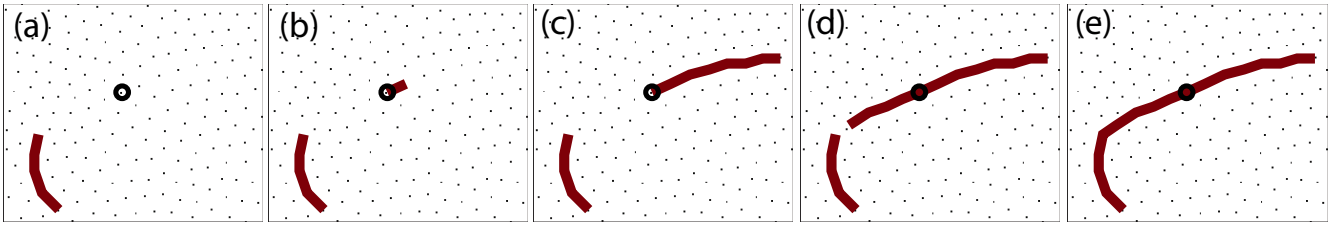


**Fig. 4** Creation of a single ribbon a) Point cloud with one ribbon already present, and $p$ highlighted b) $Right(p)$ is calculated and added to ribbon c) process continues until $Right(p) = p$ d) Process is repeated starting with $Left(p)$ e) The new ribbon ends at the end of an existing ribbon, so the two ribbons are joined

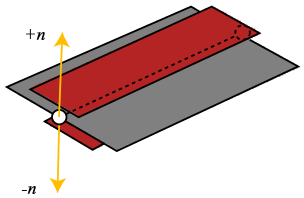


**Fig. 5** The three quads connecting two points: the top and bottom quads are shaded, whereas the middle quad is not shaded and coloured the same as the background. The two orientations of the normal are shown in orange

in Fig.5. The third quad is not displaced along the normal and is wider than the other quads. Colouring the middle quad the same colour as the background serves to occlude quads behind the current quad when performing NPR rendering. The width of the quads can be set to a constant value, or vary between quads. Rendering each ribbon as three quad strips allows for interactive rates to be achieved, and the user can adjust the maximum quad width $W_{max}$ in order to visualize the point cloud. Although ribbons do not branch or form loops, the visual quality of results is increased when they are allowed to do so (Fig.2e-f). This is achieved by extending the beginning and end of each ribbon along $next_{LR}$ for u-ribbons and $next_{UD}$ for v-ribbons when rendering. A shaded model rendered as described is shown in Fig.6. Splatting techniques such as Phong splatting[4] offer better visual quality, but require modification of the graphics pipeline to be efficient and even then become inefficient as image resolution increases[5].

It should be noted that one quad strip may be omitted when oriented normals are provided, as is sometimes the case with point cloud data.

## 5 Non-Photorealistic Rendering

Ribbons are constructed to maximize length and maintain a relatively consistent direction, properties that are desirable when ribbons are used to visualize strokes on a surface. We have approximated three pen and ink styles: silhouette rendering, hatching, and cross-hatching.

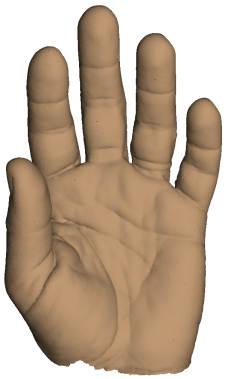

**Fig. 6** Rendering the ribbons directly to visualize the point cloud

Given a point $p \in P$ with normal $n$ and the eye position $e$ the *view angle*, denoted as $\theta$, is defined as the angle between $p - e$ and $n$. All three techniques are implemented by varying the width of each stroke based on the view angle.

### 5.1 Hatching

To emulate the placement of strokes by an artist, portions of the model almost parallel to the view plane should be represented with a small number of thin strokes. The number and width of strokes should increase as the surface turns away from the viewer, which occurs as the view angle $\theta$ decreases.

Ribbons are used as strokes on the surface with width determined by $\theta$. The width of each shaded quad starting or ending at $p$ is calculated as

$$W = W_{max}(1 - cos(\theta) - \tau)$$

where $\tau$ is a user-defined threshold. When $W \leq 0$ all quads containing $p$ are ignored. A similar formulation was employed for stable silhouettes by Brosz et al.[6] for polygon meshes. Continuous changes in the view angles result in continuous variation in width, so that stable and continuous strokes are produced. This allows frame coherency to be achieved when models are animated.

Varying width based on view angle, and rendering ribbons as described in §4 suffices to render the point

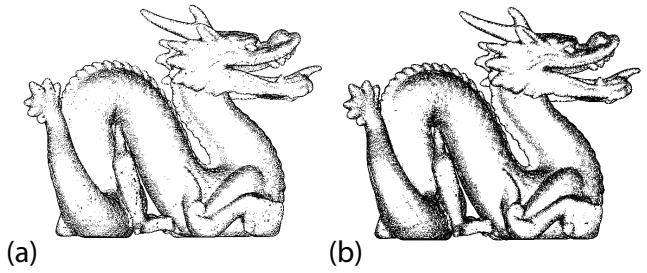


**Fig. 7** Point cloud consisting of around 100k points, a) v ribbons rendered as hatching b) Rendered with cross hatching (Model is provided courtesy of Stanford Computer Graphics Laboratory)

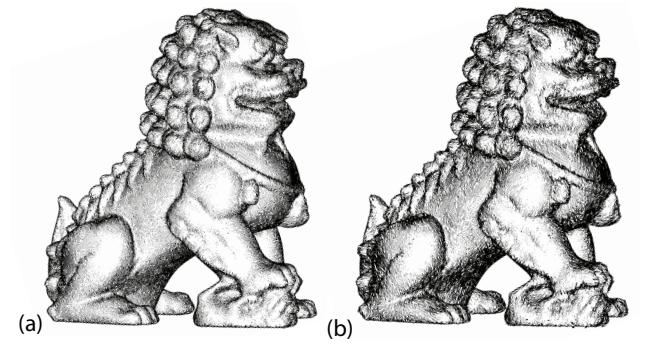


**Fig. 8** The point cloud from Fig 8 visualized using hatching a) After one level of reverse subdivision, approx. 96k b) After two levels of reverse subdivision, approx. 69k points

cloud in a style reminiscent of a hatched pen-and-ink drawing. The unshaded quad occludes strokes that should not be displayed. A cross-hatched effect can be generated by rendering two sets of chains following orthogonal directions at each point as the u and v-ribbons do. Examples of hatching and cross hatching using ribbons are given in Fig.7,9,10.

Although visually similar to hatching and cross-hatching, the proposed methods do not emulate real pen-and-ink techniques in detail. Yet, ribbons create a specific texture, which may lead to aesthetically pleasing results. For example, in Fig.10, ribbons suggest the papillary lines seen on the palm of the hand and fingers. In this sense, ribbons may be viewed as an extension rather than emulation of existing pen-and-ink styles.

When illustrations are small, they have more of a precise ink drawing appearance, but as the size is increased, additional details and individual strokes become apparent. Some irregularity is introduced into the placement of strokes by the distribution of points in the underlying point cloud; this allows conspicuous regularity to be avoided without special consideration.

### 5.2 Silhouette extraction

Silhouette lines occur on a object where the surface turns away from the viewer and becomes invisible[7]. To emulate this only the ribbon segments with normals almost perpendicular to the view vector are rendered. When compared to the method proposed in [20] our method emulates a more precise style of stroke placement and is calculated entirely in object space.

Silhouettes are produced by varying the stroke width as described for hatching, but using only two quads between points. The shaded quad is rotated to coincide with the normal instead of the tangent plane and the unshaded quad is rendered as previously described. Thus increasing the visibility of quads on the silhouette. Visualizing silhouettes require higher values of $\tau$ which restricts the strokes displayed to the silhouette. The parameter $\tau$ can be interactively chosen by the user.

### 5.3 Level of detail

The density of ribbons is tied to the density of points in the underlying point cloud. As point clouds may be very dense, it is desirable to have some type of control over the level of detail. As ribbons are linear strips we can treat them like one-dimensional curves. In that context subdivision and reverse subdivision provide an effective technique for increasing and, more importantly, decreasing the density of points[3]. By exploiting the fact that ribbons are one-dimensional curves, level of detail of the ribbons can be controlled using forward and reverse subdivision. An application of reverse subdivision to ribbons is illustrated in Fig.8.

## 6 Conclusion

Ribbons provide an intermediate representation for point clouds, falling in-between surface reconstruction and point-based splatting. With the exception of the preprocessing step k-NN searches the ribbon representation of a point cloud can be efficiently computed using only simple data structures, and can be directly rendered as quad strips.

The linear structure of ribbons is well suited to stroke placement, and this allows for a number of NPR visualization techniques. In addition, one-dimensional operations, such as forward and reverse subdivision, can be easily applied to ribbons.

Although ribbons are created by following the coordinate frame computed at each point it is possible to generate ribbons along any direction field defined at each point of the point cloud. To follow an arbitrary direction field it suffices to omit the alignment of local coordinate frames and use the direction field to connect points. This should allow for the visualization of direction fields

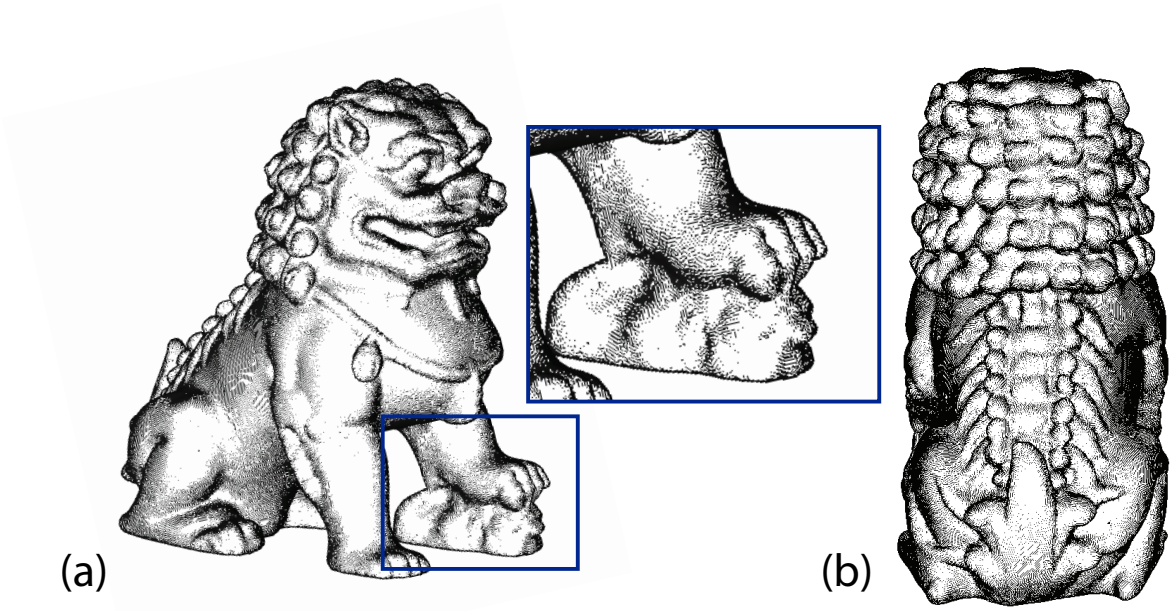


**Fig. 9** Point cloud consisting of around 153k points, a) v ribbons rendered as hatching, with an inset showing more detail b) u ribbons rendered as hatching (Model is provided courtesy of INIRIA by the AIM@SHAPE Shape Repository)

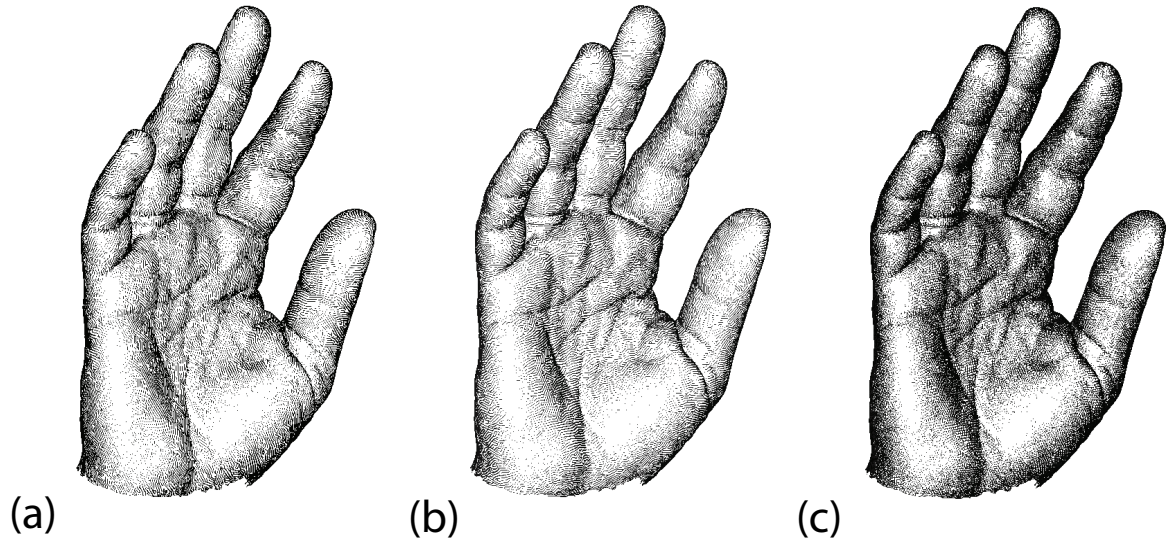


**Fig. 10** Point cloud consisting of around 53k points, a) u-ribbons rendered as hatching b) v-ribbons rendered as hatching c) The cloud rendered with cross hatching (Model is provided courtesy of INIRIA by the AIM@SHAPE Shape Repository)

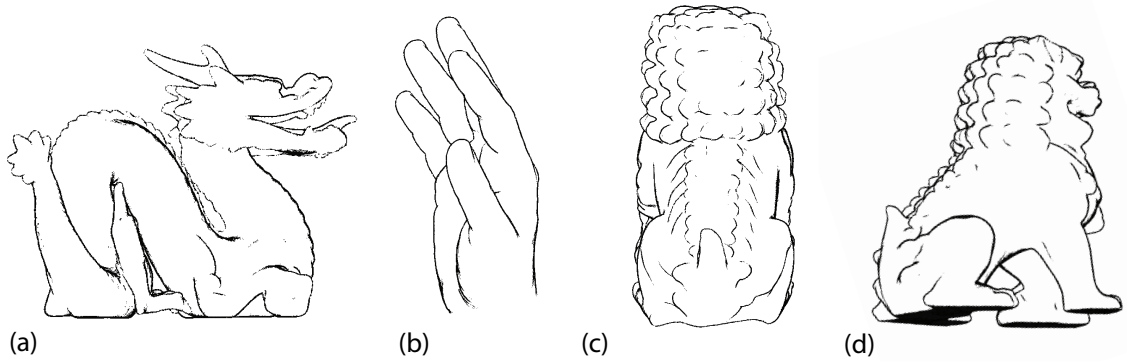


**Fig. 11** Silhouette technique applied to a)dragon b)Olivier's hand, c-d) and the Chinese dragon

on the point cloud, including the principal directions of curvature and user defined direction fields.

**Acknowledgements** We would like to acknowledge the support from the Natural Sciences Research Council of Canada to AR, FS, and PP, and from the Informatics Circle of Research Excellence to AR.

## References

1. Adamson, A., Alexa, M.: Approximating and intersecting surfaces from points. In: SGP '03: Proceedings of the Eurographics/ACM SIGGRAPH Symposium on Geometry Processing, pp. 230–239 (2003)
2. Amenta, N., Choi, S., Kolluri, R.K.: The power crust. In: SMA '01: Proceedings of the ACM Symposium on Solid Modeling and Applications, pp. 249–266 (2001)
3. Bartels, R., Samavati, F.: Reversing subdivision rules: Local linear conditions and observations on inner products. Journal of Computational and Applied Mathematics **119**(1-2), 29–67 (2000)
4. Botsch, M., Spernat, M., Kobbelt, L.: Phong splatting. In: Symposium on Point Based Graphics, pp. 25–32 (2004)
5. Boubekeur, T., Reuter, P., Schlick, C.: Visualization of point-based surfaces with locally reconstructed subdivision surfaces. In: Shape Modeling International, pp. 23–32 (2005)
6. Brosz, J., Samavati, F., Sousa, M.C.: Silhouette rendering based on stability measurement. In: SCCG '04: Proceedings of the spring conference on Computer Graphics, pp. 157–167 (2004)
7. DeCarlo, D., Finkelstein, A., Rusinkiewicz, S., Santella, A.: Suggestive contours for conveying shape. ACM Trans. Graph. **22**(3), 848–855 (2003)
8. Dey, T.K., Giesen, J., Hudson, J.: Delaunay based shape reconstruction from large data. In: PVG '01: Proceedings of the IEEE Symposium on Parallel and large-data Visualization and Graphics, pp. 19–27 (2001)
9. Hertzmann, A., Zorin, D.: Illustrating smooth surfaces. In: SIGGRAPH '00, pp. 517–526 (2000)
10. Hoppe, H., DeRose, T., Duchamp, T., McDonald, J., Stuetzle, W.: Surface reconstruction from unorganized points. In: SIGGRAPH '92, pp. 71–78 (1992)
11. Kawata, H., Gouaillard, A., Kanai, T.: Interactive point-based painterly rendering. In: CW '04: Proceedings of the International Conference on Cyberworlds, pp. 293–299 (2004)
12. Levin, D.: Mesh-independent surface interpolation. In: Geometric Modeling for Scientific Visualization, pp. 37–49 (2003)
13. Mello, V., Velho, L., Taubin, G.: Estimating the in/out function of a surface represented by points. In: SM '03: Proceedings of the ACM symposium on Solid Modeling and Applications, pp. 108–114 (2003)
14. Nice, C.: Drawing in Pen and Ink. North Light Books (1997)
15. Pauly, M., Gross, M., Kobbelt, L.P.: Efficient simplification of point-sampled surfaces. In: VIS '02: Proceedings of the conference on Visualization '02, pp. 163–170 (2002)
16. Pauly, M., Keiser, R., Gross, M.: Multi-scale feature extraction on point-sampled models. In: Proceedings of Eurographics, pp. 281–289 (2003)
17. Pauly, M., Kobbelt, L.P., Gross, M.: Point-based multiscale surface representation. ACM Trans. Graph. **25**(2), 177–193 (2006)
18. Praun, E., Hoppe, H., Webb, M., Finkelstein, A.: Real-time hatching. In: SIGGRAPH '01, p. 581 (2001)
19. Xu, H., Chen, B.: Stylized rendering of 3d scanned real world environments. In: NPAR '04: Proceedings of the international symposium on Non-photorealistic animation and rendering, pp. 25–34 (2004)
20. Zakaria, N., Seidel, H.P.: Interactive stylized silhouette for point-sampled geometry. In: Proceedings of ACM Graphite, pp. 242–249 (2004)
21. Zwicker, M., Pfister, H., van Baar, J., Gross, M.: Surface splatting. In: SIGGRAPH '01, pp. 371–378 (2001)

**Adam Runions** is a graduate student in Computer Science at the University of Calgary pursuing his Masters of Science. Adam Runions' research interests are Computer Graphics, Geometric Modeling, and Biological Modeling and Visualization

**Faramarz Samavati** is an associate professor in the Department of Computer Science, University of Calgary. He is also an adjunct associate professor in Computer Engineering Department, Technical University of Lisbon, Portugal. He received his PhD degree from Sharif University of Technology in 1999. He was a research visitor at University of Waterloo in 1997. Prof. Samavatis research interests are Computer Graphics, Geometric Modeling, Visualization, and Computer Vision. He has authored more than 40 research papers in Subdivision Surfaces, Sketch based Modeling, Multiresolution and Wavelets, Surface Modeling and Scientific Visualization.

**Przemyslaw Prusinkiewicz** is a Professor of Computer Science at the University of Calgary. He holds a Ph.D. from the Technical University of Warsaw. His research interests include computer graphics, and modeling, simulation and visualization of biological patterns and structures, in particular plants.