Joint EUROGRAPHICS - IEEE TCVG Symposium on Visualization (2003)
G.-P. Bonneau, S. Hahmann, C. D. Hansen (Editors)

Interactive Exploration of Remote Isosurfaces with Point-Based Non-Photorealistic Rendering

Guangfeng Ji, Han-Wei Shen and Jinzhu Gao

Department of Computer and Information Science
The Ohio State University
Columbus, Ohio 43210
E-mail: {jig/hwshen/gao}@cis.ohio-state.edu

Abstract

We present a non-photorealistic rendering technique for interactive exploration of isosurfaces generated from remote volumetric data. Instead of relying on the conventional smooth shading technique to render the isosurfaces, a point-based technique is used to represent and render the isosurfaces in a remote client-server environment. The non-photorealistic nature of the proposed rendering method enables the server to transmit only the essential surface features, which substantially reduces the network traffic. The algorithm also utilizes frame coherence and efficiently encodes the isosurface configuration inside each voxel cell to further minimize the network overhead. Finally, our algorithm can adjust the point distributions using different illumination settings to adapt to different network speeds.

1. Introduction

Although an increasing number of visualization tasks can be processed on today's low cost desktop or portable computers, those machines are limited by their smaller memory and disk space, and relatively slower processor speed. For this reason, visualization of large scale data sets is still heavily relying on high-end graphics workstations. Unfortunately, not everyone can access those machines conveniently. The need of high-end workstations also discourages remote collaboration between researchers who live in geographically dispersed regions.

To overcome the limitations, researchers have proposed various algorithms [1, 2, 3, 4] to facilitate remote data visualization. Remote visualization typically involves visualization servers which are responsible for retrieving data and computing visualization results, and visualization clients which share the load of visualization computations and display the final images. Designing efficient remote visualization algorithms for large scale data sets is challenging because the amount of information, including data, geometry, or images, to be transmitted from the server to the client can be quite large. As a result, the high network transmission cost often becomes the processing bottleneck.

This paper presents an algorithm for interactive visualization of isosurfaces in a remote client-server environment. Unlike the conventional visualization methods which scan convert smooth or flat shaded polygons to render the isosurfaces, we propose a non-photorealistic rendering (NPR) technique that can effectively convey the shapes of the isosurface to the remote client without sending a complete isosurface geometry from the server. In our algorithm, the server computes the isosurface interactively, and extracts important isosurface features such as silhouettes and illuminations on the fly. The surface features are transmitted to the client in a compact form. On the client site, based on the received information, a point-based shading algorithm is used to produce a non-photorealistic rendering of the remote isosurfaces. Our algorithm can significantly reduce the communication overhead, and is highly adaptable to different network speeds. Our method differs from the conventional geometry compression and streaming approaches for remote visualization in that the isosurfaces to be visualized are not pre-generated. Instead, our focus is on dynamic exploration of remote data where isosurfaces are computed, transmitted, and rendered at an interactive speed.

The rest of the paper is organized as follows. In section 2, we review different remote visualization strategies, as well

as the use of non-photorealistic rendering for scientific visualization. In section 3, we describe our technique in detail, which includes the non-photorealistic rendering of isosurfaces, and the implementation of the NPR technique in a client-server environment. Section 4 shows experimental results. Section 5 concludes this paper and provides future research directions.

2. Background

Generally speaking, existing remote visualization techniques can be classified into three categories based on the nature of the network messages. The first one is to have the server responsible for the entire visualization computation and send only the final images to the client for display [6]. The advantage of this approach is that the remote message size is independent of the data set size, which is an attractive feature for visualizing very large data sets. The drawback of this approach, however, is that the client can not change the view without having the server resend a new image even when the update is very small. The second type of remote visualization techniques is to send intermediate visualization objects such as isosurfaces or particle traces to the client for rendering [7]. The advantage of this approach is that the visualization objects can be rendered at an arbitrary scales or viewing angles by the client without requiring any communication. The main challenge for this type of techniques is that the size of the visualization objects is dependent on the underlying data size, so can be quite large. The third type of remote visualization techniques is to transmit the raw or subsampled data from the server to the client [4]. Compared to the first two types of techniques, sending data requires fast networks since the size of data to be visualized can be potentially very large.

Although not directly related to remote visualization, there has been an increasing number of non-photorealistic techniques for scientific visualization applications. Ebert and Rheingans applied various NPR techniques such as silhouette enhancement, tone shading, and oriented fading to volume rendering and received very impressive results [8]. Lu et.al. developed a direct volume illustration system that simulated traditional stipple drawing and applied several feature enhancement techniques to render complex volume datasets in a concise, meaningful, and illustrative manner[9]. Interrante [10] applied Line Integral Convolution following principle line directions to illustrate the shapes of semi-transparent surfaces. Kirby, Marmanis, and Laidlaw [11] utilized concepts from oil painting and applied brush strokes in layers to visualize multivariate data for 2D incompressible flows. In this paper, we demonstrate that non-photorealistic rendering can assist remote visualization. Our method falls into the second category of the remote visualization methods mentioned above, i.e., sending intermediate geometry to the client. We use a point-based technique to illustrate important surface features and show that it is possible to completely avoid sending isosurface meshes across the network. As a result, the network traffic is reduced significantly.

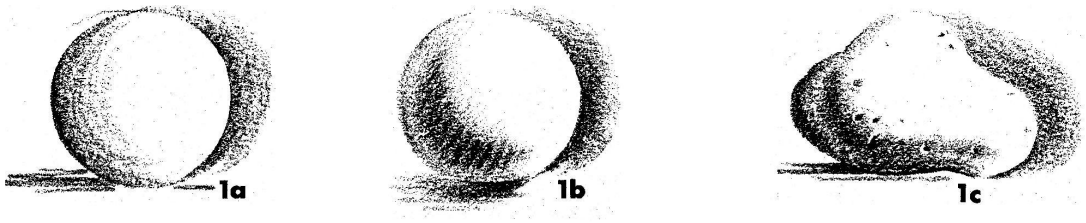
3. Remote Non-Photorealistic Rendering of Isosurfaces

Our work is motivated by the observation that in pen-and-ink or pencil drawing artwork, the artist rarely needs to shade the entire surface to convey its shape. Frequently, empty space is used to depict highly illuminated regions, while darker tones are used for areas that face away from the light. It is also not uncommon for the artist to use only a small number of tones to distinguish different illuminations. Figure 1 shows several sketch examples.

Images generated by the traditional computer graphics techniques, on the other hand, show very different characteristics. When Gouraud shading is used, for example, every pixel in the surface projection area is shaded, and a complete surface geometry is required. For remote isosurface visualization applications, however, transmitting the geometry over the network can become a major bottleneck. This is because the number of triangles computed from the marching cubes algorithm [12] is often quite large. To reduce the size of the geometry, researchers have proposed various approaches such as surface decimation [15] and view-dependent methods [13, 14]. In general, surface decimation is mostly performed at a post-processing step and thus is not suitable for dynamic isosurface exploration. View-dependent methods can reduce the geometry size considerably. However, even the visible portion of the isosurface can be still quite large.

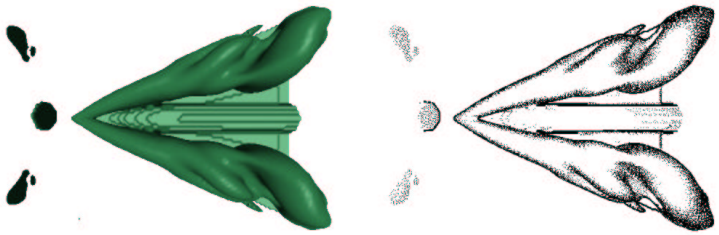
Inspired by the idea of artistic sketching, this paper presents a new approach for visualizing remote isosurfaces aiming to reduce the communication overhead. Instead of transmitting surface geometry, the server computes the essential features such as silhouettes and illuminations from the visible portion of the isosurface, and transmits a compact form of those information to the client. A NPR technique is used by the client to reconstruct and render the surface, and generate images resembling point stippling. The client is allowed to select arbitrary isosurfaces dynamically, and receives interactive feedback from the server. The client can also freely zoom in and out the isosurfaces without incurring any network traffic. When the view changes, the server only needs to provide differential information for the client to update the rendering. Compared to transmitting a complete or visible portion of the isosurface geometry, our algorithm can reduce the network traffic significantly when visualizing remote isosurfaces. Figure 2 compares the Gouraud shading image of an isosurface (Figure 2(a)) with the image generated by our NPR technique (Figure 2(b)).

In the following, we describe our algorithm in detail. We first describe the technique that is used to produce non-photorealistic rendering of isosurfaces. We then discuss the client-server remote visualization algorithm.



(a) (b) (c)

Figure 1: *The examples show that a combination of simple tones and empty space can effectively illustrate the shapes of the objects.* [5]



(a) (b)

Figure 2: *Two images generated using different surface representations and rendering techniques (a) Using the standard Gouraud shading technique (b) Using the proposed technique*

3.1. Non-Photorealistic Rendering of Isosurfaces

Two of the most important visual cues to convey the shape of a surface are silhouettes and shading. Silhouettes are used to depict the surface outlines, while shading provides cues to depict the surface orientations. For a polygonal mesh, silhouettes are edges that are shared by back faces and front faces. For parametric surfaces, silhouettes consist of the points whose normals are perpendicular to the eye direction. That is,

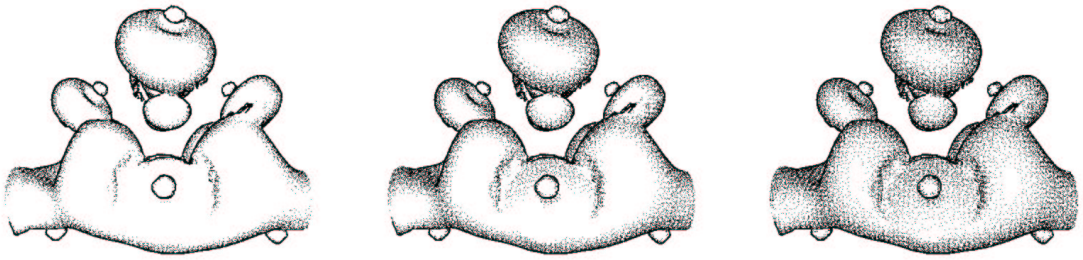
$$n_i.(x_i - C) = 0 \qquad (1)$$

where n_i is the normal of the surface point, x_i is the position of the point, and C is the camera center. To compute silhouettes for an isosurface, finding edges that are shared by front and back faces is difficult. This is because edge-face adjacency information is not readily available for isosurfaces generated by the marching cubes algorithm [12]. We compute silhouettes for an isosurface by first identifying silhouette triangles. This is done by evaluating equation 1 for each of the three vertices of an isosurface triangle and checking their signs. The normal n_i for each vertex is the gradient computed from the volume data. If all three vertices have the same sign, which means all three vertices lie in a front face or a back face, the triangle contains no silhouette. Otherwise, the triangle is a silhouette triangle and the silhouette line can be computed with linear interpolations to find the two end points on the triangle edges that have zero dot product value.

Shading is typically illustrated using different tone colors. Artists often convey various tones with strokes of different densities or by varying the pressure of the pencil point. Unlike most of the NPR techniques which use strokes or precomputed textures to display different tones [16, 17, 18], we use a point-based shading algorithm to avoid sending the triangular meshes over the network.

In our point-based shading algorithm, points with different densities convey different tones over a surface. Tone refers to the amount of visible light reflected toward the observer from a given area on the surface. Thus, the density of points over a given area should be inversely proportional to the illumination intensity of the area, that is, areas that are highly illuminated should display less points while areas in dark should display more points. Furthermore, point density should not change abruptly across the whole surface. It should vary gradually from the maximum density in the darkest area to zero density in the highlight area. Notice any illumination model can be used to calculate the tone over a surface. In our algorithm, we adopt the Phong illumination model.

The following formulas are used to implement the above



(a) (b) (c)

Figure 3: *Hipip images to show the effect of different shading threshold (a) threshold=0.3; (b) threshold=0.6; (c) threshold=0.9*

idea:

$$Coeff = ((Threshold - Tone)/Threshold)^{Power} \quad (2)$$

$$NumPoints = Area \times Coeff \quad (3)$$

where *Tone*, *NumPoints* and *Area* are the tone, number of points and the screen projection size of a given area, respectively. *Power* controls how fast the point density varies across the surface areas with different tones. The bigger *Power* is, the quicker the points drop from the maximum density in the zero tone area to zero density in the highly illuminated area. *Threshold* is a user specified shading threshold. If the tone of a given area is greater than *Threshold*, the area is considered as highly illuminated and therefore no points will be assigned. This corresponds to the empty space in the sketches shown in Figure 1. In our remote visualization algorithm, we use *Threshold* to control the network traffic. When *Threshold* is small, a larger area of surface will receive zero point. For those regions, the server does not need to send any information to the client. This can effectively reduce the amount of information required to represent the surface. Figure 3 shows the effect of different shading thresholds in our point-based rendering algorithm. Figure 4 illustrates the effect of using different values for *Power*.

The point density should also be proportional to the screen projection size of the surface. It should increase when the user zooms in and decrease when zooming out. Figure 5 illustrates how point density varies when the user zooms in and out.

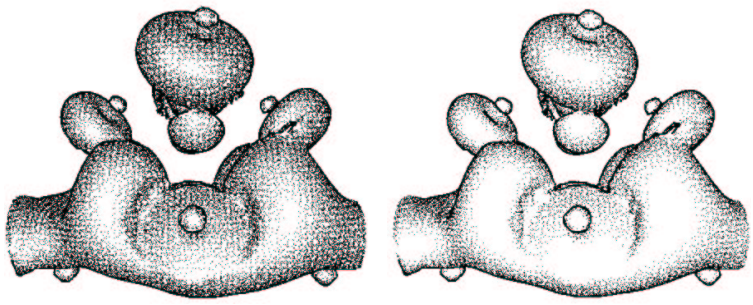
3.2. Remote Visualization of NPR Isosurfaces

In this section, we describe in detail the remote algorithm that implements the above non-photorealistic rendering technique. We first provide an overview of the remote method. We then elaborate the important components of our algorithm in details.

3.2.1. Remote Algorithm Overview

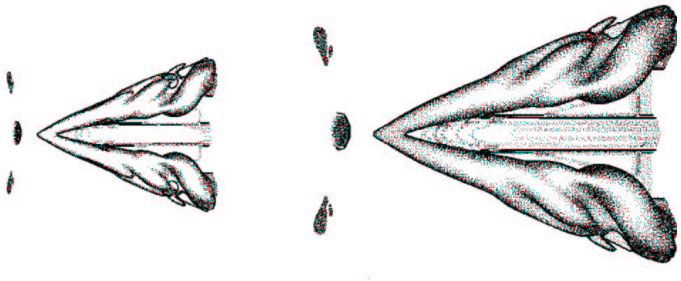
Our assumption is that only the server has a direct access to the data set since it is too large to be transmitted over the network. It is therefore the server's responsibility to provide the client with the information necessary to produce a non-photorealistic rendering of isosurfaces. Specifically, the tasks that the server has to perform include:

- **Extract the isosurfaces**: We assume that the server has enough computation power to extract isosurfaces at an interactive rate. Many efficient algorithms [19, 20, 21] can be used to accomplish this goal. In our implementation, we use a simple Branch-On-Need Octree[21] to compute the isosurfaces.
- **Extract silhouettes**: This operation is performed on a per view basis. In essence, the server will go through each isosurface triangle to evaluate equation 1 to detect triangles that contain silhouettes. We assume that the gradients of the volume are pre-computed, and thus vertex normals of the isosurface triangles can be quickly computed by interpolation.
- **Compute the point density for surface tones**: This is achieved by evaluating equation 2 and equation 3 to determine the number of points to be drawn within each isosurface triangle. If a triangle is highly illuminated and thus does not receive any points, the server can completely discard the triangle and does not send any information about the triangle to the client.
- **Compute the visibility of the surface features**: Visibility test plays a crucial role in our algorithm for two reasons: First, in order to minimize the communication cost, we do not want to send the surface features (silhouettes, surface tones) that are not visible to the client. Second, since the highly illuminated triangles in the visible isosurface are not available, the client cannot perform a com-



(a) (b)

Figure 4: *Hipip images to show the effect of different shading power (a) power=1.0; (b) power=2.0*



(a) (b)

Figure 5: *Delta wing images to show the adaptive change of point density when the user zooms in or out*

plete visibility test using graphics hardware to occlude invisible features. Therefore, the visibility test needs to be performed by the server.

- **View-dependent updates**: If we assume parallel light sources positioned at infinity and ignore the specular illumination component, the $Coeff$ in equation 2 is then view-independent, which implies that once $Coeff$ of a surface is computed, it can be reused from frame to frame when the surface is transformed. This allows us to utilize frame coherence. More specifically, when the view of the client is changed, the server only needs to inform the client the shading information for the newly visible surfaces, and the IDs of those surfaces that become invisible. The client can then quickly update the rendering.

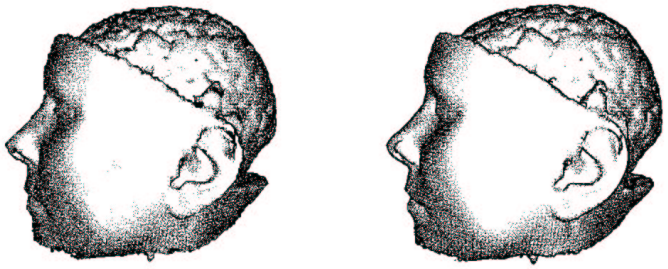
The actual non-photorealistic rendering of isosurfaces is performed by the client. At each view, the server updates the clients with new silhouette and shading information. To render the surfaces, the client needs to have the following information:

- Positions of visible silhouettes
- Positions of visible surfaces
- $Coeff$ for each visible surface

After getting all the necessary information, the client then draws silhouettes, calculates $NumPoints$ inside each visible surface and stochastically generates and renders points for all the surfaces. A brute-force way to provide the client with those information is to have the server send the exact silhouettes and visible surfaces positions and $Coeff$ of each visible surface across the network. However, the message size can be very large since every point position requires three floats (12 bytes for most of the hardware platforms) and so does the $Coeff$ of each surface. In the following, we describe the optimization technique used to address this problem.

3.2.2. Communication Optimization

Shading Points: One way to inform the client of the shading information is to have the server send the precise positions of shading points to the client. But this will incur very high communication overhead. We adopt a better approach, which is to have the client compute the point positions autonomously. To allow this, the client needs to know the po-



(a) (b)

Figure 6: *Brain examples to show that the image generated by approximate isosurface geometry is similar to that by exact isosurface geometry (a) approximate isosurface geometry (b) exact isosurface geometry*

sitions of the triangles that will receive shading points, and their tone coefficients computed by equation 2. For the triangle positions, since the underlying isosurface is produced by the marching cubes algorithm, they can be predicted if the IDs of the voxel cells containing the triangles and the corresponding marching cubes geometry cases are known. To avoid sending the vertex offsets along the isosurface cell edges, we can approximate the triangle vertex positions by the center points of the edges. Based on this idea, in our algorithm the server sends only the isosurface cell ids (4 bytes each), the corresponding marching cubes case numbers (1 byte each) to specify the triangle positions. We quantize the tones of the triangles within each isosurface cell into two bytes in total. Using the cell id, the client can quickly compute the cell location, and then uses the marching cubes case number to perform a quick table lookup to determine the triangle positions. The client then computes the number of points inside each triangle using equation 2 and equation 3, and stochastically generates and renders the points within each triangle.

It is noteworthy that although the positions of the isosurface triangles available to the client are only approximation, we are able to achieve a smooth shading effect because accurate vertex normals are used by the server to evaluate the triangle's tone. We found out that the normal, and thus the illumination, have a much larger effect to the perception of geometry shapes than the actual vertex positions. Figure 6 compares the effects of the NPR rendering using the approximate and the exact isosurface geometry.

Silhouettes: Silhouettes can also consume a large amount of network bandwidth. This is because each silhouette line consists of two end points, and each point requires three floats (12 bytes in most of hardware platforms) worth of data. To reduce the communication cost, the client approximates the silhouette lines by placing a high number of points inside the silhouette triangle. With this approximation, for each silhouette line, the server only needs to pass the cell id that contains the silhouettes (4 bytes), the marching cubes case number (1 byte), and silhouette triangle index (1 byte) to the client. Here the triangle index is a unique index assigned to each of the triangles in each marching cube case.

Bucketization: To further reduce the communication cost, the server sends the silhouettes or point shaded cells in different batches based on the marching cubes case number. That is, the server bucketizes the cells based on the marching cubes case number so that all the cells of the same case are sent at the same time to save one byte per cell. We note that without applying any data compression techniques, the total amount of message is already much smaller compared to sending the triangle vertices or silhouette lines to the client. We anticipate that the message size can be further reduced by applying existing compression techniques.

3.2.3. Visibility Determination

As mentioned previously, visibility plays a crucial role in our algorithm because the server only sends the visible cells that contain silhouettes or shading points to the clients. We have implemented two methods in our algorithm to determine the visibility of isosurfaces:

Progressive Visibility Culling

Progressive visibility culling extracts only the visible part of the isosurfaces and saves the isosurface extraction time. The algorithm performs a progressive visibility culling method similar to [23] on the server to cull away most of the invisible isosurface triangles, and perform a two-pass rendering on the client to perform further visibility culling to ensure the rendering correctness. In this section, we describe the main idea of our algorithm. Implementation details about the progressive visibility culling algorithm can be found in [23].

In our algorithm, an octree data structure is used for a

front-to-back depth order traversal. Hereafter we use the term *isosurface blocks* to refer to the octree leave nodes that contain isosurface patches, and *visible isosurface blocks* to refer to the nodes that are not completely occluded by the isosurface patches extracted from other nodes. The main idea of our algorithm is to first use the octree nodes' bounding boxes to roughly estimate the visibility of the isosurface blocks, and then use the actual isosurface extracted progressively to perform a more accurate culling. Specifically, our isosurface visibility culling algorithm is based on the following two principles:

(1) An isosurface block can be safely culled away if its bounding box is occluded by the already extracted isosurface.

(2) An isosurface block is visible if its bounding box is not occluded by either other isosurface blocks' bounding boxes, or the already extracted isosurface.

Obviously, it is impossible to cull away any isosurface blocks without first extracting some portion of the isosurface patches. To solve this problem, we devise an algorithm to cull away the invisible isosurface blocks in multiple passes. In each pass, the bounding box of each isosurface block that has not been culled is projected onto the screen, and the visibility of the block is decided based on whether the projection area has already been occupied by the previously rendered surface patches or bounding boxes. If the projection area of the bounding box is completely behind the previously extracted surface, we can discard this block according to the first principle mentioned above. If some of the pixels in the bounding box projection area are visible, based on the second principle, this block is a visible block. We should extract the isosurface patches within the block. For the blocks that do not satisfy either conditions, we defer the decision of visibility to the next pass. We repeat the process progressively until no isosurface blocks are left. To speed up the visibility determination, we use graphics hardware on the server machine to perform all the cell projection and rendering for the best performance.

The visibility determination is done at the block level. If a block contains more than one voxel, triangles inside a visible isosurface block can still be invisible. To ensure a correct rendering result, the client needs to performs a two pass rendering. In the first pass, the OpenGL depth test is enabled, and the approximate triangles are rendered. We mask the frame buffer so that the rendering result will not be seen. In the second pass, we use the depth buffer result from the first pass, and render the silhouettes and shading points. In this way, all the points that are occluded by the visible triangles will not be seen in the final image. Since the number of triangles to be rendered by the client is relatively small, our experiments show that the two pass rendering does not slow down the client speed.

Visibility Determination by OpenGL

Visibility can also be determined by using OpenGL back buffer. This is done by having the server extract isosurfaces and render them into the back buffer, with the triangle color encoded uniquely by the triangle ID. The visibility of each triangle can then be retrieved immediately from colors in the back buffer.

During our implementation, we noticed that if the data set is not very large, the overhead of progressive visibility culling can be noticeable. In this case the simple OpenGL visibility determination algorithm performs better.

Shading threshold	0.9	0.7	0.5	0.3	0.1
Silhouette	3556	3556	3556	3556	3556
Shading	5238	3431	1802	1031	380
Empty cells	1522	998	549	314	129
Bits/face	1.72	1.33	0.99	0.82	0.68

Table 1: *A list of message sizes(in bytes) when the shading threshold decreases from 0.9 to 0.1 using the delta wing dataset with isovalue 0.94. The last row shows the average bits per face. The isosurface contains 47781 triangles.*

Shading threshold	0.9	0.7	0.5	0.3	0.1
Silhouette	7767	7767	7767	7767	7767
Shading	12843	10822	6656	3364	1255
Empty cells	3075	2632	1674	855	335
Bits/face	1.03	0.92	0.70	0.52	0.41

Table 2: *A list of message sizes(in bytes) when the shading threshold decreases from 0.9 to 0.1 using the small brain dataset with isovalue 30. The last row shows the average bits per face. The isosurface contains 184580 triangles.*

4. Results and Discussion

We have tested our remote isosurface visualization algorithm using three data sets, a $111 \times 126 \times 51$ delta wing data set, a $128 \times 128 \times 72$ small UNC brain data set, and a $256 \times 256 \times 145$ big UNC brain data set. The server is a 2.0GHz Pentium IV PC with an NVIDIA Quadro2 Pro graphics card and 1.0GB memory, and the client is a 1.4GHz Pentium IV PC with a GeForce4 Ti 4600 graphics card and 768MB memory. The network connection between the client and server is a 10Mbps Ethernet.

Table 1, Table 2 and Table 3 show the effect of different shading thresholds to the network message size for the

Shading threshold	0.9	0.7	0.5	0.3	0.1
Silhouette	18160	18160	18160	18160	18160
Shading	99727	77933	43695	20683	7075
Empty cells	23446	19428	11784	5588	2104
Bits/face	1.01	0.83	0.53	0.32	0.20

Table 3: *A list of message sizes(in bytes) when the shading threshold decreases from 0.9 to 0.1 using the big brain dataset with isovalue 30. The last row shows the average bits per face. The isosurface contains 1116870 triangles.*

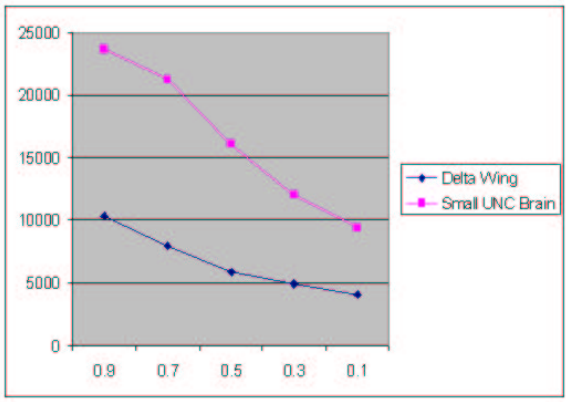


Figure 7: *The effect of shading threshold on the message size(in bytes) using delta wing and small UNC brain datasets*

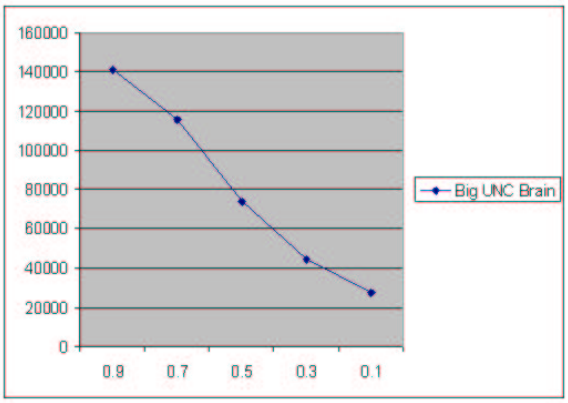


Figure 8: *The effect of shading threshold on the message size(in bytes) using big UNC brain dataset*

Rotation degree	180	60	30	10	1
Silhouette	13246	17235	17506	17843	18636
Shading	111994	107276	98893	82232	68117
Empty cells	52000	46582	34707	22147	14692

Table 4: *This table shows the message size(in bytes) when the isosurface of the big UNC brain rotated with different incremental degree. The slower it rotated, the more the frame coherence it had.*

delta wing, small UNC brain and big UNC brain data sets respectively. Shown in these tables are the size of silhouette message (Silhouette), incremental shading message (Shading) which carries shading information for the newly visible non-empty cells, and incremental empty cell ID message (Empty cells) which informs the client of which cells become invisible. The average bits per face information, which is calculated by the total size of all three types of message divided by the number of triangles contained in the isosurfaces, is also shown. We gathered the results using an automatic script which followed the following procedure. First the isosurfaces were rotated around Y axis by five degrees at each frame. After the isosurfaces had been rotated 360 degrees and returned to the original position, they were then rotated around X axis with the same constant rate and stopped at 360 degrees. The message sizes shown in the tables are the average size for all the frames. Note that we did not perform zooming since it can be processed by the client locally without incurring any network traffic. Figure 7 and Figure 8 plot the effect of different shading thresholds on the message size. As discussed previously, we use the shading threshold to control the illumination. When the shading threshold decreases, the area occupied by the highly illuminated regions (empty space) grows, and thus the message size decreases. The average bits per face also decreases as the shading threshold decreases. It can be seen that less than 1 bit per triangle is needed to transmit the isosurfaces over the network. Figure 9 shows the small UNC brain images using shading threshold 0.7, 0.5 and 0.3 respectively.

Table 4 and Table 5 show the message sizes at different rotation rates. We show the results for 180, 60, 30, 10 and 1 rotation degrees between two consecutive frames. This is to test how the frame coherence can affect the network traffic. It can be seen that the size of silhouette messages almost stays constant, due to its view dependent characteristic while the size of shading and empty-cells messages decreases when more frame coherence exists, i.e., smaller incremental rotation angle is used, since the smaller the rotation angle is, the less previous invisible triangles become visible and visible triangles become invisible.

Our algorithm allows the client to freely zoom-in, zoom-out and translate the transmitted NPR isosurfaces without the need to communicate with the server. Under such cir-

(a) (b) (c)

Figure 9: *Small UNC brain images with different shading threshold (a) threshold=0.7; (b) threshold=0.5; (c) threshold=0.3*

Rotation degree	180	60	30	10	1
Silhouette	7680	7821	7553	7646	8215
Shading	47275	35627	23738	13738	8407
Empty cells	22175	15885	9296	3993	1610

Table 5: *This table shows the message size (in bytes) when the isosurface of the small UNC brain rotated with different incremental degree. The slower it rotated, the more the frame coherence it had.*

cumstances, the client can adaptively adjust the point density according to the screen projection size of each triangle using equation 3. The computation of the point density for each triangle and the point rendering speed are all very fast. For instance, we can achieve more than 40 frames per second (fps) for the big UNC brain, more than 50 fps for the small UNC Brain dataset, and more than 100 fps for the Delta Wing dataset.

In our testing client-server remote visualization environment, we can achieve 2 fps for the big UNC brain data set, 4 fps for the small brain data set, and 5 fps for the data wing data set when the client requests arbitrary isosurfaces. The network delay is still noticeable, although it is much faster than sending the entire visible portion of isosurface triangles.

5. Conclusion and Future Work

This paper presents a non-photorealistic rendering technique for remote visualization of isosurfaces. Unlike the conventional visualization methods which scan convert smooth or flat shaded polygons to render isosurfaces, we propose a point-based rendering technique which can significantly reduces the network traffic between the client and the server. The server extracts important isosurface features such as surface silhouettes and illuminations, and transmits only a minimum amount of information to convey those features to the client. On the client site, based on the information received, a point-based shading system is used to produce non-photorealistic rendering that resembles point stippling of the isosurface. Our algorithm can significantly reduce the communication cost required to visualize remote isosurfaces, and is highly adaptable to the network speed. When the network speed becomes slow, our algorithm can reduce the amount of data to be transmitted by changing the shading threshold without significantly affecting the quality of visualization.

Future work includes investigating the use of various compression techniques to reduce the network message size even further, since network delay is still noticeable in our experiments. We will also investigate combining our NPR technique with mesh simplification algorithms so that higher message reduction rate can be achieved. Finally, we will experiment with other NPR rendering techniques to visualize isosurfaces.

References

1. O. Engel, K. Sommer and T. Ertl. A framework for interactive hardware accelerated remote 3d-visualization. In *Proceedings of Vissym 2000*, 2000.

2. O. Hendin, N. John, and O. Shochet. Medical volume rendering on the www using vrml and java. In *Proceedings of MMVR'97*, 1997.

3. K. Engel and T Ertl. A texture-based volume visualization for multiple users on the world wide web. In *Proceedings of Eurographics Workshop on Virtual Environment 1999*, 1999.

4. L. Freitag and R. Loy. Comparison of remote visualiza-

tion strategies for interactive exploration of large data sets. In *Proceedings of IPDPS 2001*, 2001.

5. J. Hamm. Drawing scenery: Landscapes and seascapes. 1982.
6. D. Stredney. Interactive medical data on demand: A high-performance image-based approach across heterogeneous environments. In *Proceedings of MMVR 2000*. IOS Press, 1996.
7. VRML 97, international specification iso/iec is 14772-1. 1997.
8. D. Ebert and P. Rheingans. Volume illustration: Non-photorealistic rendering of volume models. In *Proceedings of Visualization '2000*, pages 1951–202. IEEE Computer Society Press, Los Alamitos, CA, 2000.
9. A. Lu, C. J. Morris, D. S. Ebert, P. Rheingans and C. Hansen. Non-Photorealistic Volume Rendering Using Stippling Techniques. In *Proceedings of Visualization '2002*, pages 211–218. IEEE Computer Society Press, Boston, MA, 2002
10. V. Interrante. Illustrating surface shape in volume data via principle direction-driven 3d line integral convolution. In *Proceedings of SIGGRAPH 97*, pages 109–116. ACM SIGGRAPH, 1997.
11. R. Kirby, H. Marmanis, and D. Laidlaw. Visualizing multivalued data from 2d incompressible flows using concepts from painting. In *Proceedings of Visualization '2000*, pages 333–340. IEEE Computer Society Press, Los Alamitos, CA, 2000.
12. William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *Computer Graphics*, 21(4):163–169, July 1987.
13. Y. Livnat and C. Hansen. View Dependent Isosurface Extraction. *Proc. of Visualization '98*. pp. 175–180, 1998.
14. S. Parker, P. Shirley, Y. Livnat, C. Hansen and P.-P. Sloan. Interactive Ray Tracing for Isosurface Rendering. *Proc. of Visualization '98*. pp. 233–238, 1998.
15. W. J. Schroeder, J. A. Zarge and W.E. Lorensen. Decimation of triangle meshes. *Computer Graphics* **26**(2): 65–70, 1992.
16. G. Winkenbach and D. H. Salesin. Computer-generated pen-and-ink illustration. In *Proceedings of SIGGRAPH 94*, pages 91–100. ACM SIGGRAPH, 1994.
17. E. Praun, H. Hoppe, M. Webb, and A. Finkelstein. Real-time hatching. In *Proceedings of SIGGRAPH 2001*. ACM SIGGRAPH, 2001.
18. A. Lake, C. Marshall, M. Harris, and M. Blackstein. Stylized rendering techniques for scalable real-time 3d animation. In *Proceedings of NPAR 2000*, pages 13–20, 2000.
19. Paolo Cignoni, Paola Marino, Claudio Montani, Enrico Puppo, and Roberto Scopigno. Speeding up isosurface extraction using interval trees. *IEEE Transactions on Visualization and Computer Graphics*, 3(2):158–170, 1997.
20. Han-Wei Shen, Charles D. Hansen, Yarden Livnat, and Christopher R. Johnson. Isosurfacing in span space with utmost efficiency (ISSUE). In *Proceedings of Visualization '96*, pages 287–294. IEEE Computer Society Press, Los Alamitos, CA, 1996.
21. Jane Wilhelms and Allen Van Gelder. Octrees for faster isosurface generation. *ACM Transactions on Graphics*, 11(3):201–227, 1992.
22. C. Montani, R. Scateni, and R. Scopigno. Discretized marching cubes. In *Proceedings of Visualization '94*, pages 281–287. IEEE Computer Society Press, Los Alamitos, CA, 1994.
23. Jinzhu Gao, Han-Wei Shen, and Antonio Garcia. Parallel view dependent isosurface extraction for large scale data visualization. In *Proceedings of the Tenth SIAM Conference on Parallel Processing for Scientific Computing 2001*. SIAM Activity Group on Supercomputing, 2001.