Dynamic View Selection for Time-Varying Volumes

Guangfeng Ji*
The Ohio State University

Han-Wei Shen†
The Ohio State University

ABSTRACT

Animation is an effective way to show how time-varying phenomena evolve over time. A key issue of generating a good animation is to select ideal views through which the user can perceive the maximum amount of information from the time-varying dataset. In this paper, we first propose an improved view selection method for static data. The method measures the quality of a static view by analyzing the opacity, color and curvature distributions of the corresponding volume rendering images from the given view. Our view selection metric prefers an even opacity distribution with a larger projection area, a larger area of salient features' colors with an even distribution, and more perceived curvatures. We use this static view selection method and a dynamic programming approach to select time-varying views. The time-varying view selection maximizes the information perceived from the time-varying dataset based on the constraints that the time-varying view should show smooth changes of direction and near-constant speed. We also introduce a method that allows the user to generate a smooth transition between any two views in a given time step, with the perceived information maximized as well. By combining the static and dynamic view selection methods, the users are able to generate a time-varying view that shows the maximum amount of information from a time-varying data set.

CR Categories: I.3.6 [Computing Methodologies]: Computer Graphics—Methodology and Techniques; G.1.6 [Mathematics of Computing]: Numerical Analysis—Optimization;

Keywords: dynamic view selection, information entropy, optimization

1 INTRODUCTION

Visualization of time-varying data has been a challenging problem due to the large size and the time varying nature of the underlying datasets. Previously, researchers have proposed various techniques [9, 25, 8, 18, 15] to allow for a better understanding of the time-dependent features and their evolutions through high dimensional projection, feature tracking, and illustration. However, the most general and commonly used method for visualizing time-varying data is still animation, which is created by rendering each static volume data in the time sequence. One problem for producing animations for time-varying data is that the features of interest often evolve over time, with their shapes, positions, and orientations changing continuously. To provide the user with the best visualization of those features in an animation, it is very important to select dynamic views that can follow those features so that a maximum amount of information throughout the time sequence can be perceived. As a time-varying dataset is usually large in size and time-consuming to render, selecting views by hand can be a daunting task if it is simply done by trial-and-error. To ensure that the large scale time-varying dataset can be explored in an efficient and effective way, the process of view selection should be done automatically as much as possible.

In the context of data visualization, researchers have considered ways to automate the process of view selection [19, 3, 21, 22]. However, their focuses had not been on time-varying data, which requires special treatments in order to maximize the amount of information embedded in the whole time sequence. In addition, certain important factors when selecting a good view for static data such as the perceived colors, curvatures, and opacities in the final image were not considered in their algorithms. In this paper, we first present an improved static view selection technique to address some issues that were not previously considered, and then use the new static view selection method and a dynamic-programming optimization approach to find the best time-varying view. The goal of identifying the optimal time-varying view is to maximize the amount of information the user can perceive from the rendering sequence, with constraints on movement of the views to ensure a smooth viewing path. Our static view method measures the quality of a view based on the opacity, color and curvature images generated by a volume rendering technique. The contribution of the paper is as follows:

- An optimization approach that finds the best time-varying view in a polynomial time within a search space of exponential size. The approach also takes into account the constraints of the movement of the views.
- We properly design the probability function for the opacity distribution and incorporate it into the opacity entropy evaluation. Our opacity entropy prefers an image with a large projection area with an even opacity distribution. This technique avoids some problems that can be encountered in [3].
- The color transfer function conveys important information for volume rendering. We explicitly take into account the color information by properly designing a probability function and incorporating it into the color entropy evaluation.
- The curvature of the dataset contains essential geometric information about the dataset. We explicitly take the curvature into account during the static view selection.

In this paper, we assume that all the view points are located on the surface of a viewing sphere. At each view point the user looks at the center of the sphere, where the volume is located. During view selection, the view moves on the sphere, which means the distance between the view and the volume center is fixed. We also assume the viewing and projection parameters are appropriately set up so that the projection of the volume from any view will not fall outside the window.

The organization of the paper is as follows. In section II, we discuss the related work. In section III, we introduce our static view selection method, which includes the evaluation of opacity entropy, color entropy and information from curvatures. We also discuss how to incorporate all the three factors into a utility function. In section IV, we introduce our optimization method to perform time-varying view selection in a polynomial time from an exponential-size search space. We also give a method to select a dynamic path

*e-mail: jig@cse.ohio-state.edu
†e-mail: hwshen@cse.ohio-state.edu

between any two views which also maximizes the perceived information. In section V, we present results to prove the effectiveness of our method.

2 Related Work

The study of view point evaluation can be dated back to 1976, when Koenderink and van Doorn [12, 13] introduced the idea of *aspect graph* to partition the viewing regions surrounding an object. The node of the aspect graph is a stable view, around which the topology of the object projection does not change within a small region. The edge of the aspect graph represents a transition from a stable view to an adjacent one. The aspect graph defines the minimum number of views required to represent all the topologically different projections of the object. After its introduction, aspect graph has been studied intensively in computer vision, where many researchers used aspect graph for object recognition [4, 5, 2].

In computer graphics, several methods have been proposed to locate the optimal views for polygonal meshes. Kamada and Kawai [10] defined a view to be optimal if it minimizes the number of degenerated faces under orthogonal projection. Barral *et al.* [14] extended the idea to cope with perspective projection. In [21, 22], Vazquez *et al.*utilized the concept of information entropy from Information Theory [16] to evaluate the quality of a viewpoint. The relative visibility of each face is defined as its probability, and the optimal view is found by maximizing the probability distribution using the entropy function. Recently, Takahashi *et al.* [19] discussed view selection in the context of volume visualization. They decompose the volume into a set of feature interval volume components, and use the surface-based view point selection method suggested in [21, 22] to find the optimal view for each of the components. Then they calculate the globally optimal view by a compromise between the locally optimal views of all the feature components. In [3], Bordoloi and Shen took a volume rendering approach and proposed that in a good view point, the visibility of a voxel should be proportional to the noteworthiness value of the voxel. The noteworthiness value, or the weight of the voxel can be determined by factors such as the opacity and color of the voxel. They also discussed view similarity and how to partition the view space.

There is a rich literature in computer graphics and animation about dynamic view selection [17, 1, 24, 7, 20]. Applicable techniques range from direct orientation interpolation [17] to complex view planning for complicated 3D scenes. Andujar *et al.* [1] proposed a camera path planning method for walkthrough of complex scene models. Their method is based on identifying the free-space structure of the scene and an entropy-based measurement of the relevance of a viewpoint. Wernert and Hanson [24] discussed the camera path planning based on a personal "guide" that keeps the user oriented in the navigation space which also points to interesting subject area. Hong *et al.* [7] studied how to select camera path to navigate in the human colon. van. Wijk and Nuij [20] introduced an elegant method to generate a smooth animation from one view to the other by zooming and panning. There are two major differences between our work and the previous work. First we deal with the problem of view selection for time-varying data where the underlying phenomena are changing over time. Second our problem involves a different scene setting from the previous work, where our views move on a viewing sphere and look at the center of the sphere. Our goal is to maximize the information perceived from the time-varying data while following the view movement constraints. In [3], Bordoloi and Shen considered the problem of finding a good viewpoint for time-varying dataset. However, their method is to find a static view point throughout the animation so that the user can perceive the maximum summation of conditional entropy from the time series. The conditional entropy is the relative entropy of a datastep based on its previous step. Compared with the method, our method tries to find a dynamic viewing path.

3 Static View Selection

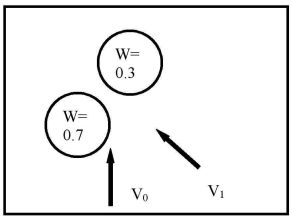


Figure 1: The figure to illustrate that the result from [3] should be improved.

The essential problem any view selection technique tries to solve is to find a good view point through which the users are able to perceive the maximum amount of information from the underlying scene. In the context of volume visualization, Takahashi *et al.* [19] proposed a surface-based view point optimization algorithm where the geometric properties of interval volumes faces are considered. Their method produces good static views for data that can be decomposed into different interval volumes. Bordoloi and Shen [3] took a direct volume rendering approach without the need of intermediate geometry. Their method generates good views in general with the exception of some cases. For example, in figure 1, there are two voxels in the scene, one with a weight of 0.7 and the other 0.3. Since their method prefers views from which the visibility of the voxel is proportional to its weight, the voxel with weight 0.7 has to occlude the other voxel to some degree in order to achieve a higher score for their entropy formula. However, these two voxels are readily visible through some views such as V_1. From this example, we can see that if the visibility of a voxel can be maximized, it does not have to be proportional to its weight. To remedy this problem and consider additional important properties of the data, we propose an image-based view selection method. Our method measures the quality of a static view not only based on its opacity and projection size (which is the primary criterion of some of the previous algorithms), but also explicitly considers the color and curvature distribution of the rendered images. Our motivation comes from the fact that color and curvature convey very important information about the underlying phenomenon in many applications.

3.1 Measurement of Opacity Distribution and Projection Size

Imagine a user is visualizing a volumetric dataset using a volume rendering technique. Some voxels in the volume have higher opacity values, meaning these voxels are more important. Less important voxels are assigned with smaller opacities. Initially the user may choose a view through which many opaque voxels are aligned in the viewing direction and hence more occlusion occurs. In this case, some pixels in the final image will have very high opacity values, while the opacity values at other pixels are low. The user realizes that this is not a good view, so s/he changes to a view where less occlusion occurs in the volume, so that the user can see many voxels more clearly. In this case, the opacity value in the image will be more evenly distributed. Besides this, the user may also generally prefers a rendering image with a larger projection area. From this example, it can be seen that an important factor that contributes to the selection of good views is the distribution of opacity values and the size of the projection area in the resulting image. An image with an even opacity distribution and a large projection area should be more favorable than one with a uneven opacity distribution and/or a

small projection area. A function is desired to reflect the property. The Shannon entropy function [16] can be utilized to perform the measurement.

In Information Theory, the Shannon entropy function is used to measure the amount of information contained in a random sequence of symbols. Suppose the symbols occur in the set $\{a_0, a_1, ..., a_{n-1}\}$ with the occurrence probability $\{p_0, p_1, ..., p_{n-1}\}$, the average information of the sequence, called entropy, is defined as

$$H(x) = -\sum_{i=0}^{n-1} p_i \cdot log_2(p_i) \quad (1)$$

One nice property of the entropy function is that it is a concave function. It only has one local maximum value, which is also the global maximum value. It reaches this maximum value $log_2 n$ when $p_0 = p_1 = ... = p_{n-1} = 1/n$, that is, the distribution of the probability is perfectly even among all the symbols. As the probability moves away from the perfectly even distribution along a straight line in any direction, the probability becomes less and less evenly distributed, and the value of the entropy function will also decrease.

The Shannon entropy function can be utilized to measure the information contained in an opacity image. We now explain how the probability is designed so that the entropy function gives a higher value when the opacity value is more evenly distributed and the projection area is larger, while it gives lower values otherwise. Given an opacity image which contains n pixels with opacity value $\{\alpha_0, \alpha_1, ..., \alpha_{n-1}\}$, we define the probability p_i of the ith pixel as

$$p_i = \frac{\alpha_i}{\sum_{j=0}^{n-1} \alpha_j} \quad (2)$$

The image entropy is calculated by equation 1. Although the entropy is evaluated over all the image pixels, the background pixels actually do not contribute to the entropy. The reason is that the opacity value of any background pixel is 0, so it will not affect the probability and entropy contribution of any foreground pixel. Furthermore, since $0 \cdot log_2 0$ is defined as 0, background pixels will not contribute to the final entropy value of the whole image. Therefore, we can define the image entropy just over the foreground area. The image entropy gets the maximum value when all the foreground pixels occur in the same probability, that is, all the foreground pixels have the same opacity values.

The entropy function also takes into account the size of the projection area, which is the foreground of the image. The reason is that the maximum entropy value of an image is $log_2 f$, where f is the size of the foreground. Therefore, the entropy of an image with a large foreground area and even distribution gets a higher value than one with smaller foreground areas. In summary, our opacity entropy function prefers an image with a large projection area with an even opacity distribution.

3.2 Measurement of Color Distribution

Opacity is just one factor that influences the selection of good views. Another important factor that determines the quality of a view is color. In volume rendering, colors are often assigned to voxels by using a color transfer function. A well-designed color transfer function should highlight salient features by using perceptually attentive colors, and map unimportant voxels to some less attentive colors. The measurement of a view's quality should keep the fidelity of the color transfer function. This means that in the color-mapped volume, even though some colors (the less attentive colors assigned to unimportant voxels, for example) may occur more frequently than some other colors (attentive colors assigned to salient features, for example), the less frequently salient feature colors actually carry more information. Therefore a good volume rendering image should contain more of these colors and thus more information about the salient features. Furthermore, we always want to highlight as many salient features as possible in the limited screen area. If the volume contains multiple salient features, these features should be mapped to the final images equally, i.e, the projected areas for different colors should be as even as possible among all the features. Based on the analysis, it can be seen that a good view should maximize the area of the salient colors while maintaining an even distribution among these colors.

To measure the color distribution of the volume rendering image, we also utilize the Shannon entropy function. The entropy function and the probability evaluation should be designed so that the entropy function gives a higher value for an image with more evenly distributed and larger areas of salient colors, while gives lower values for images with less evenly distributed and/or smaller areas of salient colors. Suppose there are n colors $\{C_0, C_1, ...C_{n-1}\}$, where $C_1, C_2, ...C_{n-1}$ occurs in the color transfer function and C_0 is the background color (actually C_0 can be a spectrum of colors, which includes every pixel of the image which is not perceptually similar to any of $C_1, C_2, ...C_{n-1}$). Given any pixel in the rendered image, we can determine which feature it belongs to by measuring the perceptual color distance between the pixel color and the feature color. If it does not belong to any feature (either the feature it should belong to is highly occluded, or it comes from unimportant voxels), it will be assigned to C_0. Suppose the total window area is T and the color areas of $C_1, C_2, ...C_{n-1}$ are $A_1, A_2, ...A_{n-1}$ respectively. The area for C_0 is then $A_0 = T - \sum_{i=1}^{n-1}(A_i)$. The probability is defined as

$$p_i = \frac{A_i}{T} \quad (3)$$

It is a probability definition since $T = \sum_{i=0}^{n-1}(A_i)$. The color entropy function is defined as in equation 1. We can see that the entropy reaches its maximum value when $A_0 = A_1 = ... = A_{n-1}$, that is, all the color areas are even. Due to the inclusion of A_0, large background area will incur small total salient color area, and thus uneven probability distribution and small entropy value accordingly. Therefore, the entropy function and our probability definition prefer larger total salient color area and more even distribution among all salient colors. It should be noted that the probability definition can lead to a small undesired effect. This happens when we see each of the salient colors and the background with the same area, which reaches the maximum of the entropy. The entropy will get smaller if the area of salient colors is enlarged, and this is undesired. However, this is less likely to happen in practice since the background area for any given view is usually large enough so that the volume rendering images from all the views can be projected into the window. We can also intentionally increase the window size to avoid the problem. Furthermore, even if the error occurs, it can be as large as $log(n) - log(n-1)$, which is a negligible number for a relatively large n.

It is also noteworthy to mention that a perception-based color space should be used during the determination of the feature a pixel belongs to. We choose the CIELUV color model since it provides a perceptually equal color space, i.e., the distance in CIELUV space reflects the perceptual color difference. We also choose a lighting model which involves only ambient and diffuse lighting calculation. Specular lighting is not included since it can alter the color of pixel by the color of the light. The color entropy evaluation works well for a well-designed color transfer function where colors are used to highlight different features. If a color transfer function just simply assigns gray-scale or rainbow colors according to different values, the color entropy may not reflect the feature information contained in the view.

3.3 Measurement of Curvature Information

Opacity and color are two important factors that measure the quality of a view. In addition to opacity and color, there are other properties that also contribute to the information provided in a volume rendering image. One of such properties is the curvature. Low curvatures imply flat areas and high curvatures mean highly irregular surfaces, which often contain more information (If the volume is noisy, a smoothing operation should be performed beforehand). Therefore, it is important to take the curvature information into account during the selection of good views.

One problem of considering curvature information in view selection is how to present the curvatures in a volume rendering image. We achieve this with two steps. First we calculate the curvature at each voxel position of the volume, using the method proposed by Kindlmann *et al.* [11]. When the volume is rendered, the color of a voxel is determined by its curvature. Voxels with high curvature are assigned with high intensity colors, while voxels below a certain low-curvature threshold are assigned with the color $(0,0,0)$. The opacity of the voxel is determined independently, which can be based on its original data value, or some other properties such as the gradient. After the rendering is performed and the image is generated, the intensity of the image reflects the amount of curvature perceived from the visible part of the volume, that is, an image with high intensity means that the user can see many high-curvature voxels from that view.

3.4 The Final Utility Function

Opacity, color and curvature all contribute to the information perceived from a rendering of the volume. We need a function to incorporate all the factors. This utility function [23] u from a view v should have the following basic form:

$$u(v) = (opacity(v) + color(v) + curvature(v))/3 \qquad (4)$$

that is, the utility function should consider contribution from all the factors. One problem with the utility function is that the opacity, color and curvature contributions are not normalized. We should normalize each of the factors into $[0,1]$ before the summation. The maximum value of the entropy function of an image with a projection size of n is $log_2 n$. So if we find the maximum projection size M of the images among all the views, each of the entropies can be normalized by dividing over $log_2 M$. The maximum value of the color entropy is $log_2 n$, where n is the number of colors (see section 3.2). Therefore, the color entropy can be easily normalized by a division over $log_2 n$. The normalization of the curvature contribution can also be easily done by a division over the maximum projection size M, since the maximum intensity of each pixel is 1.

If we possess any prior knowledge of the volume, it is often desirable to give different weights to different factors. The utility function then have the following form:

$$u(v) = \alpha \cdot opacity(v) + \beta \cdot color(v) + \gamma \cdot curvature(v) \qquad (5)$$

where $\alpha + \beta + \gamma = 1$. One scenario is that people often design very sophisticated opacity transfer function, but use a simple gray-scale or rainbow color transfer function. In this case, it is desirable to put more weight into $opacity(v)$ than $color(v)$, since opacity conveys more information. However, in another case where different colors are used to highlight different features in a segmented volume, it is desirable to put large weight to $color(v)$. In practice, we can choose proper weight for every factor based on the characteristic of the data and transfer function and the nature of the application.

4 Dynamic View Selection

In this section, a dynamic view selection algorithm is presented. The goal of dynamic view selection is to allow the user to find a viewing path which shows the maximum amount of information from the time-varying dataset, and the path should show near-constant angular velocity (all the views lie on the surface of a viewing sphere). We formulate this into the following three principles that a good dynamic viewing path should follow:

- The view should move at a near-constant speed.
- The view should not change its direction abruptly.
- The information perceived from the time-varying data should be maximized among all the viewing paths.

In the following subsections, we first discuss the issue of how to select time-varying views that follow the three principles. Then we present a method that allows the user to find a path between any two views in a given timestep that maximizes the perceived information while obeying the other two principles.

4.1 Time-Varying View Selection

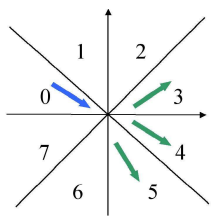
The problem of time-varying view selection is that given a view at $t = 0$, among all the possible paths along which the view can move smoothly to the final timestep at a near-constant angular velocity, find the path that gives the maximum perceived information. If in average a view can move to one of n possible views at the next timestep, and there are total t timesteps, the complexity of the problem can be n^t. This search space is exponentially large. It is impractical to try all these paths and find the optimal one.

To solve the problem more efficiently, we can employ the dynamic programming approach. Let's first consider selecting time-varying views with the first and third principles in mind, that is, we want to find a time-varying view that moves at a near-constant speed, and the information perceived from that path is maximized out of all possible paths. Suppose the camera is moving with speed V, with $V_{min} \le V \le V_{max}$. V_{min} and V_{max} are used to bound the speed of the view so that when V_{min} is close to and V_{max}, the view moves at a near-constant speed. We use $P_{i,j}$ to denote the position of the jth view at $t = i$, and $MaxInfo(P_{i,j})$ is the maximum amount of information perceived from $P_{i,j}$ to some view at the final timestep. The following recursive function holds:

$$MaxInfo(P_{i,j}) = \max_{k=0}^{NumofViews-1}\{u(P_{i,j}) - Cost(P_{i,j}, P_{i+1,k}) + MaxInfo(P_{i+1,k})\}$$

where $u(P_{i,j})$ measures the information perceived at the view $P_{i,j}$. $Cost(P_{i,j}, P_{i+1,k})$ measures the cost to move from $P_{i,j}$ to $P_{i+1,k}$. If the jth view point and the kth view are within $[V_{min}, V_{max}]$, the cost is 0, otherwise the cost is $+\infty$. The equation basically says the maximum amount of information perceived from $P_{i,j}$ to some view point at the final timestep will be equal to the sum of the information perceived at $P_{i,j}$, and the maximum information perceived from $P_{i+1,k}$ to some view at the final time step. $P_{i+1,k}$ represents a view point at $t = i+1$ that can be reached within $[V_{min}, V_{max}]$ distance from $P_{i,j}$. We will consider all the views $P_{i+1,k}$ at timestep $i+1$. The following C-style code performs the calculation of all the $MaxInfo(P_{i,j})$.

```
//Initialization
for (i=0; i<NumofViews; i++)
    MaxInfo[NumofTimeSteps-1, i]=
    u[NumofTimeSteps-1, i];
```



(a)

$$\{\{0,0,0,1,1,1,0,0\},\\ \{0,0,0,0,1,1,1,0\},\\ \{0,0,0,0,0,1,1,1\},\\ \{1,0,0,0,0,0,1,1\},\\ \{1,1,0,0,0,0,0,1\},\\ \{1,1,1,0,0,0,0,0\},\\ \{0,1,1,1,0,0,0,0\},\\ \{0,0,1,1,1,0,0,0\}\}$$

(b)

Figure 2: An example of a partition of a view point's local tangent plane and one of the possible allowed turns encoded in matrix.

```
//Dynamic Programming
for (i=NumofTimesteps-2; i>=0; i--)
    for (j=0;j<NumofViews; j++)
    {
        MaxInfo[i, j]=0;
        for (k=0; k<NumofViews; k++)
        {
            double Info=u[i, j]-Cost(j,k)
            +MaxInfo(i+1, k);
            if (Info>MaxInfo[i, j])
            {
                MaxInfo[i, j]=Info;
                NextViewIndex[i, j]=k;
            }
        }
    }
```

The initial condition is $MaxInfo(P_{n-1,i}) = u(P_{n-1,i})$ for $i \in [0..NumofViews-1]$. The dynamic programming process calculates all the $MaxInfo\{P_{i,j}\}$ backwards in time, according to the recursive function. $NextNodeIndex\{P_{i,j}\}$ records the view index at the next timestep that gives the maximum information from $P_{i,j}$ to some view at the final timestep, and it can be used to recover the time-varying path. The dynamic programming process finishes all the computation in $O(n \cdot v^2)$ time, where n is the number of total timesteps, and v is the number of total views. This process only takes a polynomial time complexity.

The above dynamic programming calculates an optimal path based on the restriction that the view should move with the speed within $[V_{min}, V_{max}]$. But it does not prohibit the view from making sharp turns, which is undesirable when viewing the animation. It is also impossible to use the information stored at *NextViewIndex* to find the optimal path that does not make sharp turns, since *NextViewIndex* only records the optimal paths that move at a near-constant speed. To address this problem, at each view point on the viewing sphere, we partition its local tangent plane into many different regions, and restrict the allowed turns. Figure 2 illustrates a partition of eight regions and a matrix that encodes the allowed turns. We use $MaxInfo(P_{i,j,r})$ to denote the maximum amount of information perceived from $P_{i,j}$ to some view point at the final timestep, and $P_{i,j}$ was entered from region r from its previous view. Then the following recursive function holds:

$$MaxInfo(P_{i,j,r}) = \max_{r=0..NumofRegions-1, k \in Regionr}\{u(P_{i,j}) \\ -Cost(P_{i,j}, P_{i+1,k}) + MaxInfo(P_{i+1,k,s})\}$$

The following C-like code calculates all the $MaxInfo(P_{i,j,r})$:

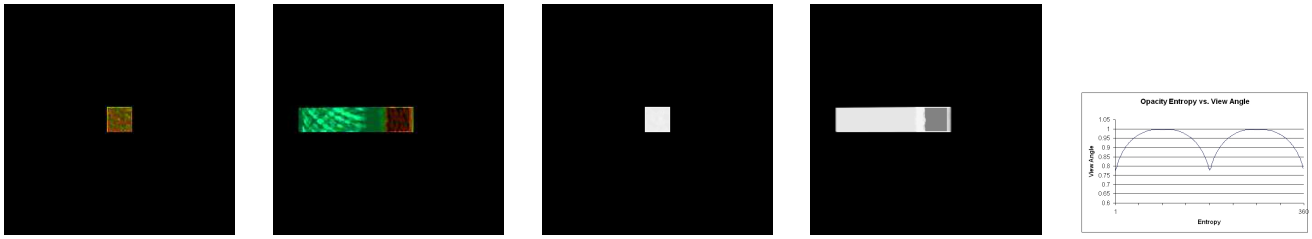
```
\\Initialization
for (i=0; i<NumofViews; i++)
    for (j=0; j<NumofRegions; j++)
        MaxInfo[NumofTimeSteps-1, i, j]=
        u[NumofTimeSteps-1, i];

\\Dynamic Programming
for (i=NumofTimesteps-2; i>=0; i--)
    for (j=0; j<NumofViews; j++)
        for (r=0; r<NumofRegions; r++)
        {
            MaxInfo[i, j, r]=0;
            for (all ks that are in region r)
            {
                int o=FindRegionNum(k, j);
                if (!AllowedTurn[r, o])
                    continue;
                int s=FindRegionNum(j, k);
                double Info=u[i, j]-Cost(j, k)
                +MaxInfo(i+1, k, s);
                if (Info>MaxInfo[i, j, r])
                {
                    MaxInfo[i, j, r]=Info;
                    NextViewIndex[i, j, r]=k;
                    NextRegionIndex[i, j, r]=s;
                }
            }
        }
```

where o is the region number leaving the jth view and s is the region number entering the kth view at the next timestep. o and s can be easily determined based on the the projection to local tangent plane at the jth and kth view respectively. *NextViewIndex* and *NextRegionIndex* record the view and region index at the next timestep that offers the maximum information to some view at the final timestep. These two data structures can be used to recover the path. The dynamic programming process finishes all the computation in $O(n \cdot r \cdot v^2)$ time, where n is the number of timesteps, v is the number of views, and r is the number of regions. This process only takes a polynomial time complexity. After the dynamic programming is done, given the initial view at $t = 0$, the results stored at $MaxInfo$, *NextViewIndex* and *NextRegionIndex* can be used to find the maximum perceived information and the optimal time-varying view associated with the initial view.

4.2 Viewing Path Between any Two Views in a Given Timestep

Another case of dynamic view selection is to find a viewing path between any two viewpoints in a given timestep. This viewing path should also follow the three principles, i.e., moves between these two viewpoints smoothly with a near-constant angular velocity, and maximizes the perceived data information at the same time. This technique can be very useful to showcase a static dataset. When generating an animation, keyframes are usually specified by the user, and intermediate frames are generated by interpolation. If different viewpoints are assigned in the different keyframes, spherical linear interpolation (SLERP) is a common technique to interpolate the intermediate view positions. SLERP does give a viewing path with constant angular velocity, but it does not take the perceived information into consideration. Next we will explain how we maximize the perceived information and take all three principles into consideration.

Given any two views on a viewing sphere, there are an infinite number of paths that connect these two views. One factor in our design of the dynamic path is that it should follow the general direction of the SLERP path, since the SLERP path is the shortest path that connects the two points with constant angular velocity. Therefore, we only allow the view to move at the neighboring views of the SLERP path (as shown in figure 3). We also need to put restriction on the direction of the allowed movement so that the view



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

Figure 4: The figure shows the static view selection results based on opacity entropy for the shockwave dataset. (a) shows the worst view, (b) is the best view, and (c) and (d) are the opacity images for (a) and (b) respectively. (e) plots the change of opacity entropy with respect to different viewing angles where the shockwave is rotated around the vertical axis in a full circle.

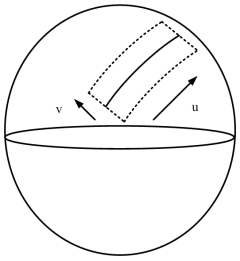


Figure 3: The solid curve is the SLERP path. Our algorithm will consider all the neighbors of the SLERP path that lie within the dotted area. All the neighbors are parameterized by u and v.

will not go back and forth in a circular manner. We achieve this by parameterizing all the neighbors relative to the SLERP path, as illustrated in figure 3. A movement is allowed only if the u parameter is increasing and the v parameter difference is within a threshold. We call these paths *monotonic paths*. We can also enforce the direction change by adopting the local coordinates and the admissible turn matrix in figure 2. When evaluating the quality of different paths, the summation of information should not be used, since some paths can go through more view points than others. One good criterion can be the average information. The pseudo code below illustrates how to use the propagation method similar to the single-source shortest path algorithm to find the optimal path.

```
ActiveSet={Source viewpoint S};
PathLength=0;
PathInfo[S,PathLength]=u(S);

Initialize all other PathInfos to a minimum value;
NextActiveSet=empty;
while(ActiveSet is not empty)
{
    PathLength++;
    for each view V in ActiveSet
        for each neighbor N of V
        if (the movement from V to N is monotonic)
        {
            Path[N, PathLength]=max(Path[N,
            PathLength], u(V)+Path[V, PathLength-1]);
            Put N in NextActiveSet;
        }
        ActiveSet=NextActiveSet;
}

For all the PathInfo[D, n] where D is the destination
    Find the one with the maximum average
    information and it will be the optimal path.
```

Notice the above process only runs on the neighborhood of the SLERP path. It finishes in $O(N^2)$ time, where N is the number of neighbors along the SLERP path.

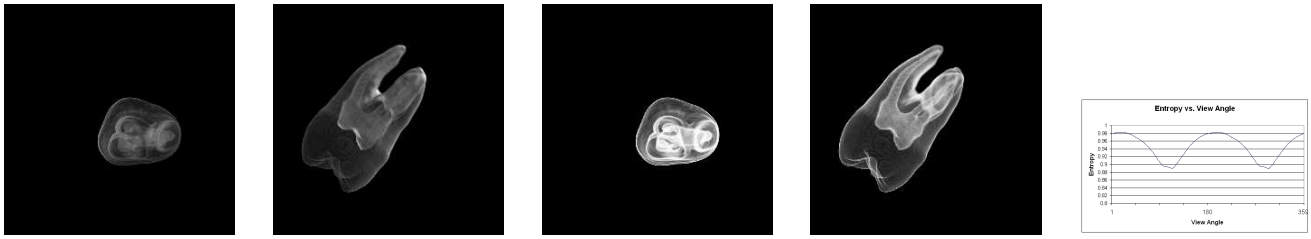
5 Results and Discussion

We have implemented and tested both the static and dynamic view selection algorithms on a Pentium IV 1.4GHz machine with an nVidia GeForce 6800 graphics card. Our view selection algorithms take as input the opacity, color and curvature images rendered from the dataset, which can be generated by any volume rendering technique. In our implementation, we choose a hardware-based volume slicing technique with 3D texture mapping to generate those images. 256 sample views were used for each dataset, and these views are evenly distributed on the viewing sphere.

The test result for the $512 \times 64 \times 64$ shockwave dataset is shown in figure 4. The opacity entropy value is used during the test to show its effectiveness in determining view quality. Figure 4 (a) shows the worst view which has the smallest opacity entropy, and figure 4 (b) shows the best view with the highest opacity entropy. Figure 4 (c) and (d) illustrate the opacity images of the worst and best views respectively. It took 6.92 seconds to compute the opacity entropy values for the 256 views and find the best and worst views, and the size of the opacity image is 256×256. By using the entropy function and the proposed probability function, our opacity entropy evaluation takes both the opacity distribution and the projection area into consideration, and the opacity entropy prefers an image with an even opacity distribution and a larger projection area. To illustrate how the opacity entropy varies according to different viewing angles, the view is rotated along the vertical axis (Y axis) in a complete circle. Figure 4 (e) plots the change of opacity entropy with respect to different views.

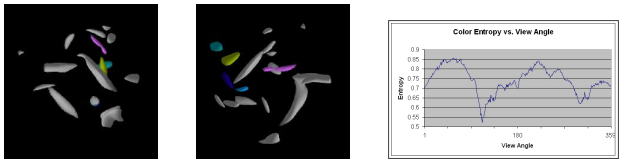
We also used the $128 \times 128 \times 80$ tooth data to test the view selection algorithm based on the opacity entropy, and the result is shown in figure 5. Figure 5 (a) shows the worst view with the smallest opacity entropy, and figure 5 (b) shows the best view with the largest opacity entropy. Figure 5 (c) and (d) are their opacity images. It took 7.18 seconds to compute the opacity entropy values for the 256 views and find the best and worst views, and the size of the opacity image is 256×256. The variation of opacity entropy with respect to different views is also plotted in the Figure 5 (e), where the viewing angle is rotated incrementally around the X axis.

We used the 128^3 vortex dataset to show the effectiveness of the color entropy function. The data set contains many components and we use the color transfer function to highlight components which may go through topological changes in future timesteps. Other



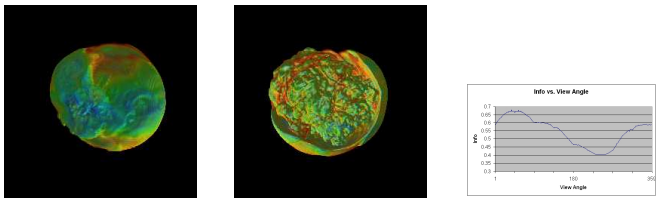
(a) (b) (c) (d) (e)

Figure 5: The figure shows the static view selection results based on opacity entropy for the tooth dataset. (a) shows the worst view, (b) is the best view, and (c) and (d) are the opacity images for (a) and (b) respectively. (e) plots the change of opacity entropy with respect to the viewing angle when the tooth is rotated around the X axis in a full circle.



(a) (b) (c)

Figure 6: The figure shows the static view selection results based on color entropy for the vortex dataset. (a) shows the worst view, (b) is the best view, and (c) plots the change of color entropy with respect to different viewing angles when the vortex is rotated around the Y axis in a full circle.



(a) (b) (c)

Figure 7: The figure shows the dynamic view selection results for the TSI dataset. (a) shows the worst view, (b) is the best view, and (c) plots the change of the final information with respect to the viewing angle when the TSI dataset is rotated around the vertical axis in a full circle.

components are assigned a gray-scale color. Figure 6 (a) shows the worst view with the smallest color entropy, and figure 6 (b) shows the best view. It can been easily seen that figure 6 (b) conveys more information about the topologically important features than figure 6 (a). In figure 6 (a), the total projection area of the highlighted features is small, and the projection area ratio among the highlighted features is very uneven. This leads to a very small color entropy value. In contrast, in figure 6 (b), the highlighted features have a large projection area and an even projection area distribution, and therefore a large value for the color entropy. It took 16.3 seconds to compute the color entropy values for the 256 views and find the best and worst views, and the size of the color image is 256×256. Figure 6 (c) plots the change of color entropy with respect to different views where the viewing angle is rotated incrementally around the Y axis.

Figure 7 gives the view-selection result for the Terascale Supernova Initiative (TSI) dataset. The dataset is to model the core

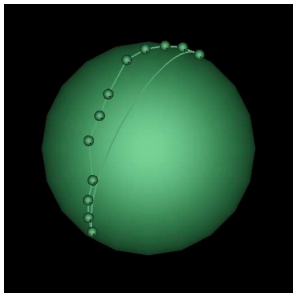


Figure 8: The figure shows two paths which move from one view point to the other. The right path is generated by SLERP interpolation with an average information of 0.51. The left path is generated by our method. The path is smooth and gives an average information of 0.56.

collapse of supernovae and was generated by collaboration among Oak Ridge National Lab and eight universities. In the paper, we visualize the entropy scalar component of the dataset, which is derived from pressure and density scalar values. When exploring the dataset, we used the rainbow color transfer function. In our view selection test, two factors, curvature and opacity, are considered in the calculation of view information. We want to design a utility function which puts more weight for views that show more jagged area. Therefore, in our design, we set the coefficients for curvature and opacity to 0.8 and 0.2 respectively. Figure 7 (a) shows the worst view, and figure 7 (b) is the best view. It is obvious that figure 7 (b) shows more detailed information about the jagged area than Figure 7 (a). It took 18.7 seconds to evaluate the curvature information and opacity entropy for all the 256 views and find the best and worst views, and the size of the image is 256×256. To show how the view utility function varies, figure 7 (c) plots the change of utility value with respect to different views, where the view is rotated incrementally around the vertical (Y) axis.

We also used the TSI dataset to test our dynamic view selection algorithm. The supernova is a very dynamic phenomenon where the features are morphing and rotating rapidly in space. Our previous static view selection shows that from some views very little information about the phenomenon can be perceived. If the view for an animation is fixed, much of the phenomenon would be occluded for many timesteps (see figure 9 (f)-(i)). Recall that the goal of our algorithm is to find a viewing path with the maximum amount of information and also follow the constraint that the camera moves at a near-constant angular velocity. We used our static view selection to calculate the view information of every view point at every timestep and used our dynamic programming algorithm to find the best path. All the timesteps use the same view point set on the

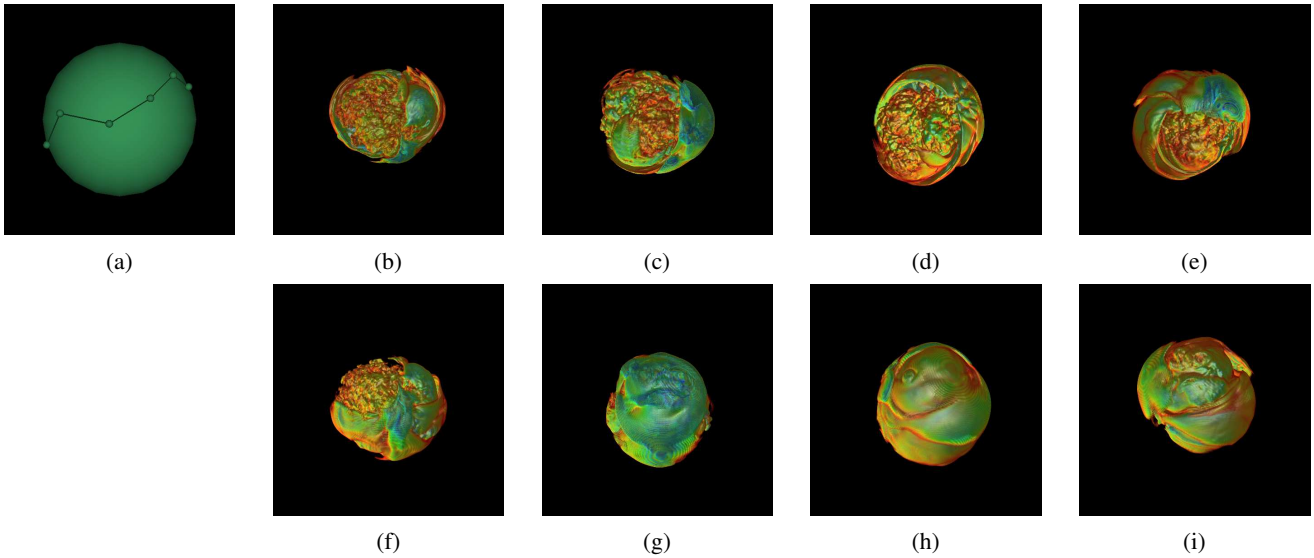


Figure 9: The figure shows the dynamic view selection result for the TSI dataset. (a) shows the path of the time-varying view, which exhibits constant angular velocity. (b)-(e) show four snapshots captured by our time-varying view. (h)-(i) show the image from the original static view at the timestep corresponding to (b)-(e) respectively.

sphere. Figure 9 (a) shows the best path in which viewpoint $P_{0,0}$ moves in time with the speed within (0.9, 1.2) (The radius of the viewing sphere is 1). Although the supernova phenomenon is morphing rapidly, we still perceive a maximum amount of information following our dynamic viewing path. It took 4.31 seconds for the dynamic programming process to find the optimal path. Figure 9 (a) shows part of the path, which demonstrates near-constant angular velocity (the distance in figure 9 (a) is distorted). Furthermore, following the path, the overall information perceived from the time-varying data is maximized. Figure 9 (b)-(e) show four snapshots of the time-varying dataset captured by the time-varying view path, and figure 9 (f)-(i) show the images seen from the original view at the timesteps corresponding to (b)-(e) respectively. The user can apparently see more turbulent side of the phenomenon all the time from the time-varying views.

We also used the TSI dataset to show a viewing path selected from any two views in a given timestep. The TSI dataset at $t = 0$ is used, and figure 8 shows both the SLERP and the optimized paths. It took 0.08 seconds to find the optimized path. The average information perceived by the SLERP path is 0.51, while the optimized path gives 0.56.

6 Conclusion and Future Work

In this paper, we present methods for both static and dynamic view selection. Our static view selection algorithm analyzes opacity, color and curvature images generated from different view points. We properly design the probability functions and use entropy to evaluate opacity and color distributions. Our algorithm also prefers a view which shows high curvature information. Depending on the characteristic of the data set and the opacity and color transfer function, and the nature of the application, we can design different utility functions to assign different weights to the three factors. Based on our static view selection and dynamic programming, our dynamic view selection method maximizes the information perceived from the time-varying dataset following a near-constant angular velocity path. The optimization is achieved in a polynomial time. Our results show the effectiveness of the static and dynamic view selection.

In addition to dynamic view point planning, another important parameter for animation would be lighting design. Gunhold [6] discussed light source placement for static polygonal meshes. We would like to conduct the research for lighting design for time-varying polygonal and volumetric data in our future work.

References

[1] C. Andujar, P. Vazquez, and M. Fairen. Way-finder: Guided tours through complex walkthrough models. *Computer Graphics Forum*, 23(3):488–508, 2004.

[2] T. Arbel and F. Ferrie. Viewpoint selection by navigation through entropy maps. In *Proceeding of International Conference on Computer Vision*, pages 248–254, 1999.

[3] Udeepta D. Bordoloi and Han-Wei Shen. View selection for volume rendering. In *IEEE Visualization Conference 2005*, pages 487–494, 2005.

[4] C.M. Cyr and B.B. Kimia. 3d object recognition using shape similarity-based aspect graph. In *Proceeding of International Conference on Computer Vision*, pages 254–261, 2001.

[5] K.D. Gremban and K.Ikeuchi. Planning multiple observation for object recognition. *International Journal of Computer Vision*, 12(2/3):137–172, 1994.

[6] Stefan Gumhold. Maximum entropy light source placement. In *IEEE Visualization Conference 2002*, pages 275–282, 2002.

[7] Lichan Hong, Shigeru Muraki, Arie Kaufman, Dirk Bartz, and Taosong He. Virtual voyage: Interactive navigation in the human colon. *Computer Graphics*, 31:27–34, 1997.

[8] Guangfeng Ji, Han-Wei Shen, and Rephael Wenger. Volume tracking using higher dimensional isocontouring. In *IEEE Visualization Conference 2003*, pages 209–216, 2003.

[9] Alark Joshi and Penny Rheingans. Illustration-inspired techniques for visualizing time-varying data. In *IEEE Visualization Conference 2005*, pages 86–93, 2005.

[10] T. Kamada and S. Kawai. A simple method for computing general position in displaying three-dimensional objects. *Proceeding of International Conference on Computer Vision*, 41(1):248–254, 1988.

[11] Gordon Kindlmann, Ross Whitaker, Tolga Tasdizen, and Torsten Moller. Curvature-based transfer functions for direct volume rendering: Methods and applications. In *IEEE Visualization Conference 2003*, pages 513–520, 2003.

[12] J.J. Koenderink and A.J. van Doorn. The sigularities of the visual mapping. *Biological Cybernetics*, 24:51–59, 1976.

[13] J.J. Koenderink and A.J. van Doorn. The internal representation of solid shape with respect to vision. *Biological Cybernetics*, 32:211–216, 1979.

[14] G. Dorme P. Barral and D. Plemenos. Scene understanding techniques using a virtual camera. In *Proceeding of Eurographics 2000*, 2000.

[15] R. Samtaney, D. Silver, N. Zabusky, and J. Cao. Visualizing features and tracking their evolution. *IEEE Computer*, 27(7):20–27, 1994.

[16] Claude E. Shannon. A mathematical theory of communication. In *Bell System Technical Journal*, pages 379–423 & 623–656, 1948.

[17] K. Sheomake. Animation with quaternion curves. *Computer Graphics*, 19:245–254, 1985.

[18] Deborah Silver and Xun Wang. Volume tracking. In *IEEE Visualization Conference 1996*, pages 157–164, 1996.

[19] Shigeo Takahashi, Issei Fujishiro, Yuriko Takeshima, and Tomoyuki Nishita. A feature-driven approach to locating optimal viewpoints for volume visualization. In *IEEE Visualization Conference 2005*, pages 495–502, 2005.

[20] Jarke J. van Wijk and Wim A.A. Nuij. Smooth and efficient zooming and panning. In *IEEE Symposium on Information Visualization 2003*, pages 15–23, 2003.

[21] Pere-Pau Vazquez, Miquel Feixas, Mateu Sbert, and Wolfgang Heidrich. Viewpoint selection using viewpoint entropy. In *Vision Modeling and Visualization Conference 2001*, 2001.

[22] Pere-Pau Vazquez, Miquel Feixas, Mateu Sbert, and Wolfgang Heidrich. Automatic view selection using viewpoint entropy and its application to image-based modeling. *Computer Graphics Forum*, 22(4):689–700, 2003.

[23] John von. Neumann and Oskar Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, 1944.

[24] E.A. Wernert and A.J. Hanson. A framework for assisted exploration with collaboration. In *IEEE Visualization Conference 1999*, pages 241–248, 1999.

[25] Jonathan Woodring, Chaoli Wang, and Han-Wei Shen. High dimensional direct rendering of time-varying voulmes. In *IEEE Visualization Conference 2003*, pages 417–424, 2003.