

Local Feature Relevance for Efficient Navigation  
and Visualization of Large Data Sets

By

Peng Xiang

Bachelor of Mechanical Engineering

Nanjing University of Aeronautics  
and Astronautics

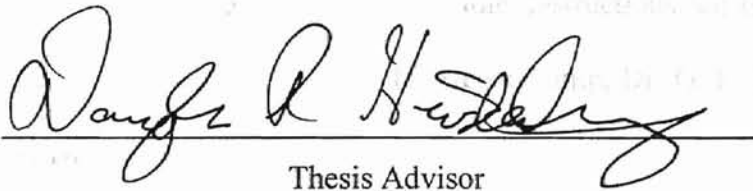
Nanjing, China

1994

Submitted to the Faculty of the  
Graduate College of the  
Oklahoma State University  
In partial fulfillment of  
the requirement for  
the Degree of  
MASTER OF SCIENCE  
August 2002

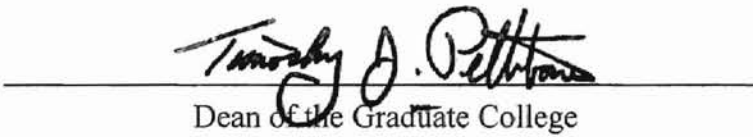
Local Feature Relevance for Efficient Navigation  
and Visualization of Large Data Sets

Thesis Approved:

  
Thesis Advisor





  
Dean of the Graduate College

## TABLE OF CONTENTS

1 36 1

2 2017-2018 Visualization as Representation of Data Explanation 2

## ACKNOWLEDGMENTS

I would like to express my sincere thanks to my major advisor, Dr. Douglas Heisterkamp. His careful reviewing for my thesis and constructive advice are very important for the completion of the thesis. Thanks go to my committee members, Dr. G. E. Hedrick and Dr. John P. Chandler. Their precious instructions are guarantee for the success of this thesis. Thank you, Dr. Douglas Heisterkamp, Dr. G. E. Hedrick and Dr. John P. Chandler. Their help is really appreciated.

Finally, I also want to thank my family. Their encouragement keeps me away from laziness and depression. I also thank their love, without which I would give up long time ago. My thanks also extend to my friends for their support and understanding.

1. Introduction.....31

2. A survey of some visualization Representation of Data Exploration.....2

    2.1 Turn-Key.....3

    2.2 Data-Flow.....3

    2.3 Spreadsheet-like Interface.....4

    2.4 Design Gallery.....4

    2.5 Image Graph.....5

3. Background.....5

    3.1 Distance Metric.....6

    3.2 Content-Based Image Retrieve (CBIR).....6

    3.3 Relevance Feedback (RF).....6

    3.4 Covariance Matrix.....7

4. Local Feature Relevance for Efficient Navigation and Visualization of Large Data Sets.....8

    4.1 Design Analysis.....8

    4.2 Specific Methods.....9

5. Performance Experiments.....13

    5.1 Experimental Framework.....13

    5.2 Results and Discussion.....16

6. Summary.....17

References.....21

Appendix A Example code.....22

Portion 1.: Code for System Calculation.....22

Portion 2.: Code for mapping process .....30

Appendix B Glossary.....41

## LIST OF TABLES

Table 1. Framework for system background calculation -----	14
Table 2. Classes for mapping process -----	15
Table 3. Driver Programs -----	15
Table 4. Summary of the experiment -----	17

## LIST OF FIGURES

Figure 1. Flow chart of implementation process -----	11
Figure 2. A selection-based dispersion heuristic -----	12
Figure 3. User-interface map -----	16
Figure 4. Practical operation 1-----	18
Figure 5. Practical operation 2-----	19
Figure 6. Practical operation 2-----	19
Figure 7. Practical operation 2-----	20

## 1. Introduction

Both efficient algorithms and intuitive user interfaces are pretty useful to gain insight from large, scientific data sets via visualization. A simple definition of (scientific) visualization is the merging of data with the display of geometric objects through computer graphics [5]. Visualization is important for its convenience and intuition in analyzing large data sets. T.J. Jankun-Kelly and Kwan-Liu Ma already explored the intuitive user interface that is a spreadsheet-like interface [18] to present and navigate through the visualization of the data. That means the user can modify or navigate parameters by a user interface. Design Gallery [8] and image graph [12], which adopt different ways to navigate parameters by an intuitive interface. A different approach from above all is provided in this thesis. We apply a weighting updating method to regenerate image. Adapting the input vector parameters based on the user's feedback in the original input space is the general idea of the method. The key point is that all of the navigations are performed by system platform, which is hidden from user, and the only thing needed is feedback from the user by clicking relevant images and irrelevant images. This system includes several key elements: *input vector* is a list of parameters that control the generation of the output graphic via a mapping process. The *distance metric* on the space of input vectors approximates the perceptual similarity of the output spaces. The *dispersion* of graphics is to present the user dispersed graphics through a perceptually reasonable arrangement method that makes use of the distance metric. The end user need only recognize and select appealing graphics from gallery.

A brief introduction of visualization of data will be given in section 2, and we are going to give some background knowledge for our approach in section 3. The specific approach at the algorithmic level will be presented in section 4. Section 5 introduces the

concrete experiment in detail. Section 6 is the summary of the thesis, which includes main contribution of our work and future work. Example code is presented in appendix A, and glossary is in appendix B.

## 2. A survey of some visualization Representation of Data Exploration

First of all, we should know what the purpose of visualization of data [6] is.

- a. Harness perceptual capabilities of human visual system to extract information from data sets.
- b. Look for structure, features, patterns, trends, anomalies, relationships.
- c. Provide a qualitative overview of large, complex data sets.
- d. Assist in identifying region(s) of interest and appropriate parameters for more focused quantitative analysis.

Secondly, we would like to introduce some common terms for you.

- *Visualization* - the graphical (as opposed to textual or verbal) communication of information (e.g. data, documents, structure).
- *Interaction* - a fundamental component to visualization which permits user-specified modifications to the visualization parameters.
- *Data model* - representation of data, may include structure, attributes, relationships, behavior, and semantics as well as repository for data values themselves.
- *Graphical attributes* - user-controllable aspects of the image generation process, including position, size, color, shape, orientation, speed, texture, and transparency of graphical entities.



- *Mapping* - associating data values and attributes to graphical entities and attributes.
- *Rendering* - creating and displaying an image.
- *Field* - a grid of data points, may be uniform/nonuniform.
- *Scalar* - a single numeric data value.
- *Vector* - a list of values associated with a single datum.

## 2.1 Turn-Key

In traditional turn-key visualization, a user iteratively changes parameter values directly in order to search for the desired result, see [18].

This trial and error process is inefficient and does not communicate context that directs a user toward the goal. Once an acceptable visualization result is obtained, only the final parameter settings and images are available to be recorded and shared with collaborators; all previous results are lost.

## 2.2 Data-Flow

Data-flow interface represent the data exploration process by a directed network of connected components. These components act upon the data sets or output of other states to produce their final result(s). Each component in the network represents an operation or transformation on the output of the previous step. Components can set parameter values for subsequent visualization techniques, see [18].

Data-flow interfaces have a better state display than traditional turn-key interface through the data-flow graph, and this flow graph can be shared with collaborators to

communicate the process needed to generate the final result, but one weakness of the approach is that it does not indicate the history of the visualization process.

### **2.3 Spreadsheet-like Interface**

The spreadsheet-like interface [18] represents a two-dimensional window into a multidimensional visualization parameter space. Data is explored by navigating this space via 2D spreadsheet interface. The visualization parameter space is presented to the user in a manner that identifies which parameters correspond to which visualized result. Operations defined on this space can be applied which generate new results. Combined with a general-purpose interpreter, these functions can be utilized to generate the desired results. Like the Design Gallery and image graph, Spreadsheet-like interface consider visualization exploration a process of examining a multidimensional space of parameter values.

By visually organizing the data exploration process while providing tools to build upon and share this process, spreadsheet-like interface makes visualization more efficient and effective, but it complicates matters as it represents a multidimensional space. It is difficult to force a 2D view on a multidimensional space.

### **2.4 Design Gallery**

The Design Galleries system [8] considers data exploration a process of exploring a multidimensional space of visualization parameters. The results a user desires exist within this space. It is the system's job to aid in the discovery of the parameters that correspond to the images. After a preprocessing rendering stage, the system displays a 3D

representation of the design space. A user then navigates this space to find the desired images.

By replacing a trial and error approach with a structured navigation of parameter values, the system allows a more efficient exploration. Screen space becomes limited as the number of different parameter settings increases. In addition, it is pre-computed, global, and fixed sampling of visualization parameters.

## 2.5 Image Graph

The image graph system [12] follows a similar structured approach. Unlike the DG system, image graphs are built dynamically instead of during a preprocessing stage; it avoids preprocessing in favor of adding newly rendered images to an image graph. An image graph is a graph representation of the visualization process that distinctly displays the relationship between generated images via glyph edges. The graph is used to explore the space of visualization parameters. As more visualization is added, the graph structures itself so that related images are clustered together. A user can manipulate this structure as desired. Operations upon the edges and nodes in the graph can be used to generate further result with the result itself. The user can take advantage of the information in image graphs to understand how certain parameter changes affect visualization result. Users can also share image graphs to streamline the process of collaborative visualization.

The interface is limited by its display and manipulation of a single data set at a time. This prevents cross data set comparisons or operations.

## 3. Background

### 3.1 Distance Metric [1], or combined version [2]. In the weighting updating method,

The set of input attributes, for which we want to make a prediction about the resulting output attributes, is called the query, or query point. The first step is to define what is meant by similarity. We have to define a distance metric that tells how close two points are. The distance between two points (between their input attributes) in a scaled Euclidean distance metric is defined by:

$$\text{dist}(\mathbf{q}, \mathbf{r}) = \left( (\mathbf{q} - \mathbf{r})^T \mathbf{A} (\mathbf{q} - \mathbf{r}) \right)^{1/2} \quad (1)$$

where  $\mathbf{A}$  is an identity matrix (a diagonal matrix composed of ones) and  $\mathbf{q}$  and  $\mathbf{r}$  refer to vectors of input attributes. Other distance metrics include Manhattan distance, and Mahalanobis distance [2]. Scaled Euclidean distance works well for most cases.

### 3.2 Content-Based Image Retrieve (CBIR)

Typically, the content of an image can be characterized by a variety of visual properties known as features. The key issue in Content-based image retrieval [14, 15, 16, 20] is how to match two images according to these computational extracted features. It is common to compare images by color, texture, and shape.

### 3.3 Relevance Feedback (RF)

Relevance feedback is a technique to learn the user's subjective perception of similarity between images. It is the modification of the mapping process to improve accuracy by incorporating information obtained from prior relevance judgment. RF has been used in CBIR, but has not been used explicitly in visualization of data sets. There are two basic types of relevance feedback: (1) weighting updating [10, 11, 17, 19], and

(2) query point moving [1], or combined version [3]. In the weighting updating method, the weights (parameters) associated with the similarity measure are updated based on the feedback. In the query point moving, the query  $\mathbf{Z}$  is modified based on the feedback.

In this thesis, we consider the combined approach. In probabilistic feature relevance learning (PRFL) [9], retrieved images with relevance feedback are used to compute local feature relevance. The relative relevance can be used as a weighting scheme for a weighted k-nearest neighbor search (KNN):

$$w_i(\mathbf{Z}) = \exp(Tr_i(\mathbf{Z})) / \sum_{j=1}^t \exp(Tr_j(\mathbf{Z})) \quad (2)$$

where  $t$  is the number of samples with RF,  $r_i$  is a measure of feature relevance at query  $\mathbf{Z}$  and  $T$  is a parameter that can be chosen to maximize (minimize) the influence of  $r_i$  on  $w_i$ . For further details, see [9]. Then we can get diagonal weighting matrix such as

$$\mathbf{B} = \begin{pmatrix} w_1 & & 0 \\ & \ddots & \\ 0 & & w_n \end{pmatrix} \quad (3)$$

which provides parameter on  $dist_{A,B}^2$  which stands for the squared distance between image A and B.

$$dist^2(\mathbf{q}, \mathbf{r}) = (\mathbf{q} - \mathbf{r})^T \mathbf{B}(\mathbf{q} - \mathbf{r})$$

where  $\mathbf{q}$  and  $\mathbf{r}$  are vectors corresponding to image A and B.

### 3.4 Covariance Matrix

Covariance matrix is represented as equation 5, and we can use it to calculate Mahalanobis distance  $dist_M^2(\mathbf{x})$  [2] of  $\mathbf{x}$  from the mean vector  $\mathbf{m}_x$ . Equation 6 shows the specific method.

$$C_x = \frac{1}{n} (\mathbf{x} - \mathbf{m}_x)(\mathbf{x} - \mathbf{m}_x)^T \quad (5)$$

$$dist_M^2(\mathbf{x}) = (\mathbf{x} - \mathbf{m}_x)^T C_x^{-1} (\mathbf{x} - \mathbf{m}_x) \quad (6)$$

This matrix provides us with a way to measure distance that is invariant to linear transformations of the data.

## 4. Local Feature Relevance for Efficient Navigation and Visualization of Large Data Sets

### 4.1 Design Analysis

This approach is to generate images rather than retrieve images from a large database. The whole process is a cycle that is depicted in the flow chart (Figure 1). Input is *input vectors*  $\mathbf{Q}$  depicted in chart 1 of Figure 1. A **modified dispersion** algorithm is used in dispersing images, because it is impossible to present all images to user. This algorithm is modified based on DG dispersion algorithm. There are some differences between them (See Figure 2 in detail). Since it is not necessary for user to be involved in navigating parameters so that it doesn't require higher level of user's expertise, we design a simple **GUI** (Graphical User Interface) for user to select his/her desired images (See Figure 3 in detail). RF is explicitly used in visualization of data sets by this approach, which applies weighting updating method in calculating weighting matrix. The central idea behind the weighting updating method is simple and intuitive. Since an  $N$  dimensional feature vector represents each image, we can view it as a point in an  $N$  dimensional space. Then, the basic idea is to enhance the importance of those dimensions of a feature that help in retrieving the relevant images and reduce the importance of those dimensions that hinder this process [7]. A simple algorithm based on this idea was

described in the ImageRover system [17]. **Query point moving** method is necessarily used in this approach to direct user to the goal. In the beginning, we independently do sampling for each dimension on modified space by standard normal distribution, after that, covariant normal distribution should also be tried in sampling on modified space so that we can compare their effects. A structure such as trainingSet is designed to save the user's feedback (relevant or irrelevant images' information) during every stage. We need to remember all selections of user so that user can reuse them and keep the history so that user can go back to previous stage to reselect. Our code is designed to handle multi-dimensional vectors such as hundreds or more parameters of vectors.

#### 4.2 Specific Methods

The whole process is described as following flow chart (see Figure 1). Let's explain chart by chart.

The **modified dispersion** algorithm is adapted from DG dispersion algorithm and is outlined in Figure 2. The dispersion method is used to find a set of input vectors that map to a well-distributed set of output vectors in DG, whereas we want to find a well-distributed set of input vectors themselves. We disperse according to distance of input vectors, and then it is necessary for us to modify the dispersion algorithm so that it can be used in our approach. We calculate similarity (distance) between input vectors instead of similarity of output vectors. The detail of modification is introduced in Figure 2.

We construct a simple GUI rendered with GL and implement mouse function and keyboard function to interact with user. Then take advantage of PRFL to calculate weighting matrix. Project relevant vectors and irrelevant vectors  $R$  in new space so as to get modified space.

$$D = B^{1/2} R = AR \quad (7)$$

First of all we need to solve mean value and deviation of the vectors,  $m_i \in R$ ,

$$\mu = \sum_{i=1}^n m_i / n \quad (8)$$

$$\sigma = \sqrt{\sum_{i=1}^n (m_i - \mu)^2 / n} \quad (9)$$

$$n = |m| \quad (10)$$

Then we tried to choose samples independently by standard normal distribution for each dimension of a vector on this space. Covariance matrix method that takes all impacts of all parameters of a vector into account is also tried. So a new set of vectors  $G$  is generated on the modified space by sampling, then we disperse 20 vectors by modified DG dispersion method and query point moving method. At last, project back to original input space after that.

$$O = \left( B^{1/2} \right)^{-1} T = A^{-1} T \quad (11)$$

**Query point moving** method is also used in this approach. We use Rocchio equation [4] whose parameters  $\alpha, \beta, \gamma$  can be adjusted to get new query value.

$$Z_0 = m_0 \quad (12)$$

$$Z_{i+1} = \alpha \cdot Z_i + \beta \cdot m_{i+1} + \gamma \cdot n_{i+1} \quad (13)$$

where  $i \geq 0$  and  $m_i$  is vector of mean values for positive feedback, and  $n_i$  is vector of mean values for negative feedback. Since we saved all relevant and irrelevant input vectors in memory, an optional function is designed to decide if take the previous



relevant vectors or irrelevant vectors into account. Currently we display ten of the twenty images by modified DG dispersion algorithm and other ten images by query point moving algorithm. First, get new query point based on previous query point, and then we sample ten points by standard normal distribution.

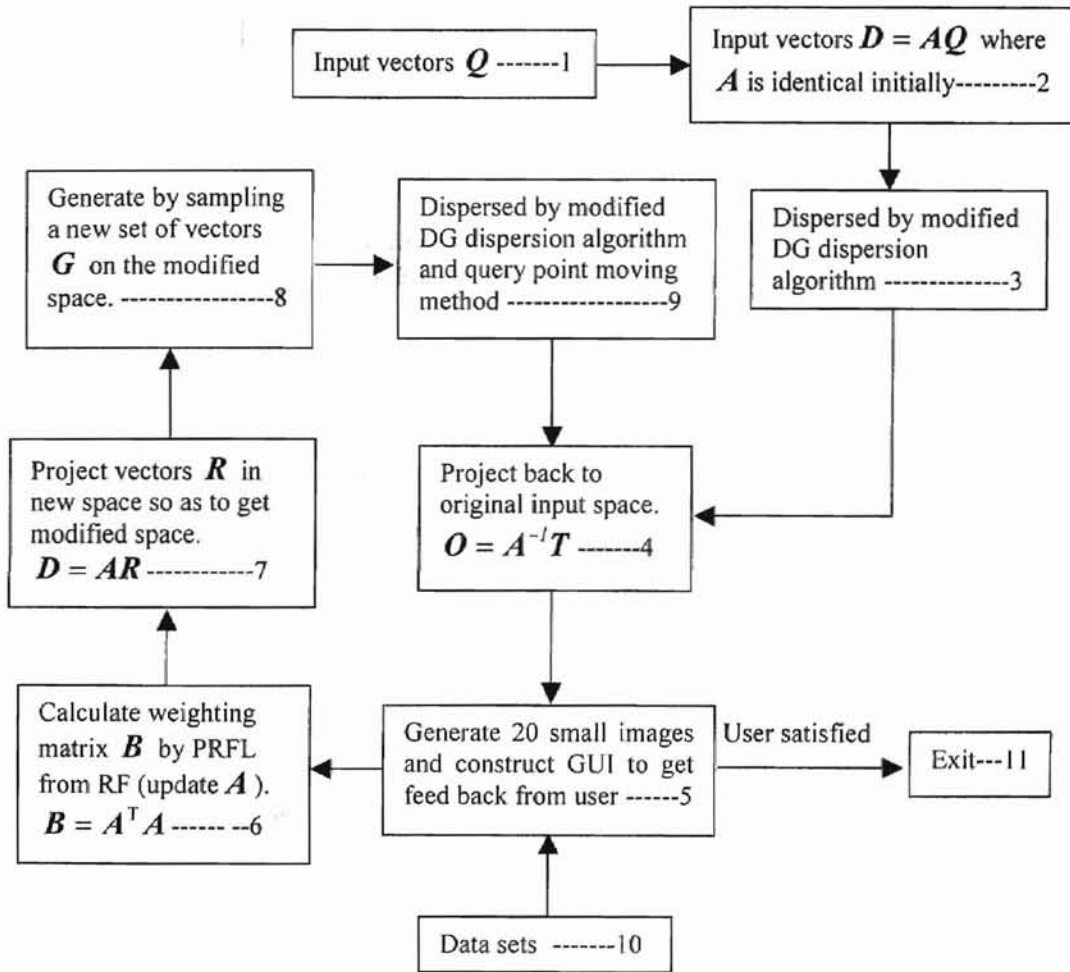


Figure 1: Flow chart of implementation process

Input:  
L, a set of input vectors.  
 $n < |L|$ , the size of the selected subset.

Output:  
 $I \subset L$ , a set of  $n$  dispersed vectors.

Procedure:

```
Selection_disperse(L, n)
{
  I ← L1;
  for i ← 2 to n do
  {
    p_score ← -∞;
    for each q ∈ L do
    {
      q_score ← ∞;
      for each r ∈ I do
      {
        if dist (q,r) < q_score then
          q_score ← dist (q,r);
      }
      if q_score > p_score then
      {
        p_score ← q_score;
        P ← q;
      }
    }
  }
  I ← I ∪ {P}
  L ← L \ {P}
}
```

Notes:

L1 denotes the random input vector in the set L.  
\ denotes set difference.  
dist (q,r) returns the value computed by equation 1.

Figure 2. A selection-based dispersion heuristic (This is Modified version, which is cited from DG [8]. The first difference, it begins at L1 rather than 0. The second, image\_diff based on output vectors is replaced by dist depending on dissimilarity of input vectors.)

### 5.1 Experimental Framework

Current implementation is in C++, using OpenGL [13]. The whole procedure is divided into two portions, one is mapping process, and another is background calculation. First of all, select an appropriate subset of input vectors over the original input space. Then, render with GL in back buffer, and copy back buffer into block of memory. Of course, different input vectors create different visualizations, that is, they correspond to different mapping processes. We are going to try rendering circle and teapot in our experiment. Of course, the code is designed to be suited for multi-dimensional vectors. After rendering, the final twenty images are going to be presented to user in reasonable dispersion by the user interface. Because we don't want user adapt parameters by himself, the only thing needs to do is to select the images that they want, we just need a simple user interface like  $|I| = 20$ , which is arranged by  $5 \times 4$  small windows (see Figure3). It needs to be pointed out that ten of images are generated by modified dispersion algorithm and others are produced by query point moving method. Basically, we define VECTOR in whose element's type is double to act as a *vector*, and will be always taken as basic element in both portions. There expect are five classes in system calculation portion (see table 1 in details). Mapping process includes two classes and driver program (see table 2 and 3 in detail). The vector consist of four parameters in rendering circle, the first one is hue, the second and third one are saturation and brightness, the last one is radius of circle. Meanwhile, the vector is ten parameters vector in rendering teapot, they are hue, saturation, brightness (the three are for material of teapot), size of teapot, light location on x axis, light location on y axis, light location on z

axis, the angel rotation about x axis, the angel rotation about y axis, the angel rotation about z axis.

Class Name	Methods' Name	Behavior
dispersion	Selection_disperse	Disperse vectors in reasonable way with modified DG algorithm.
	dist	Calculating distance by equation 1.
Und	mean	Calculate mean values of a set of vectors, equation 8, 10.
	deviation	Calculate standard deviation of a set of vectors, equation 9, 10.
	selection	Independently choose some samples on each dimension by standard normal distribution.
transform	M	A set of vector project to modified space by multiplying weighting matrix. This method is used to implement right hand of equation 7.
	T	A set of vectors also needs to be projected back to original input space. This method is to implement right hand of the equation 11.
RF	WM	Get diagonal weighting matrix by equation 2.
bg	calc	<ol style="list-style-type: none"> <li>1. Retrieve the vectors based on relevant images.</li> <li>2. Calculating weighting matrix depended on the relevant vectors, equation 2, 3.</li> <li>3. Project to the modified space.</li> <li>4. Solve mean and deviation values of the relevant vectors and mean values of the irrelevant vectors as well as out.</li> <li>5. Sampling on the modified space (two ways), equation 5 and 6.</li> <li>6. Project them back to original input space.</li> <li>7. Disperse ten of twenty input vectors by modified DG dispersion algorithm and other ten input vectors are selected by query point moving algorithm, equation 12, 13.</li> </ol>

Table 1: Framework for system background calculation.

Class Name	Methods' Name	Behavior
colorconv	HSVtoRGB	Convert from Hue-Saturation-Value color space to Red-Green-Blue color space.
Canvas	setWindow	Set parameters of the window
	setViewport	Set parameters of view port
	getWindowAspect	Get values of window aspect
	lineTo	Draw a line to a destined point
	moveTo	Move to a destined point
	turn	Turn a angle based on original angle
	turnTo	Turn to a destined angle
	forward	Move to or line to a destined point
ngon	Draw a polygon of n edges	

Table 2: Classes for mapping process.

Driver Program's Name	Functions' Name	Behavior
circle	myInit	Clear back ground color and set window size
	display	Display circles and construct GUI
	mouse	Click left button to choose relevant images, go to next stage or back to previous stage. Click middle button to zoom out, click right button to zoom back in.
	keyboard	Click Esc key to exit
	getFeedback	Get information from user's operations
teapot	myInit	Clear back ground color and set window size
	display	Display teapots and construct GUI
	mouse	Click left button to choose relevant images, go to next stage or back to previous stage. Click middle button to zoom out, click right button to zoom back in.
	keyboard	Click Esc key to exit

Table 3: Driver Programs

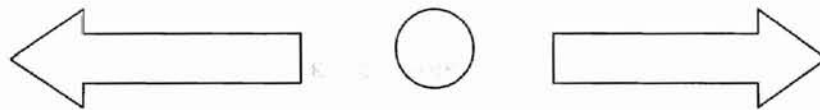



Figure 3: User-interface map

Twenty small images will be shown separately on these twenty small windows. When user click middle key on the target, the full-size image of the target will be shown on the screen. User can go back to old stages when he/she click on the “back” mark ⇐, he/she can also go to next stage when he/she click on the “next” mark ⇒ after chose at least one relevant image, user can also take the previous selections into account by clicking the middle circle mark, of course it is an optional function.

## 5.2 Results and Discussion

We present user 10 images with distance between two of them is as large as possible by our **modified dispersion** algorithm, which makes user have as many as possible choices, and present other 10 images by query point moving method. We tried two algorithms in sampling on modified space, one is to choose sample independently on

each dimension by standard normal distribution, the other take advantage of covariance matrix. Both of them work well in our experiments. Query point moving method, which directs user toward his/her goal, is applied in our experiment, and it is verified to be effective. On other hand, user can go back old stage to select again if current images are not satisfied by the simple **UGI**. The approach is more flexible and the history of each stage is saved in the memory so that they can be retrieved later. There are two resulting examples showed in Figure 4 and 5.

Application	Circle	Teapot
Input vector	Radius, hue	Material, size, angle, light position

Table 4: Summary of the experiment

## 6. Summary

We apply a relevant feedback mechanism for navigation and visualization of large data sets. In this approach, a PRFL measure is defined over the user's feedback and the probabilistic method is applied in sampling in the modified space. The **modified dispersion** algorithm is verified to be useful in spreading images, and **query point moving** method work well too to direct user to the goal. The standard normal distribution method used in independently choosing samples on each dimension also works well in our experiment. We adapt input vector parameters based on user's feedback, meanwhile save more useful information into training set so as to approach to user's idea quickly. By navigating data in the background, our approach makes visualization simpler. The **GUI** used in our experiment is easy to operate and easy to understand by user. We are going to keep exploring more reasonable algorithms in section of system calculation so as to make navigation of the large data sets more efficient in the future.

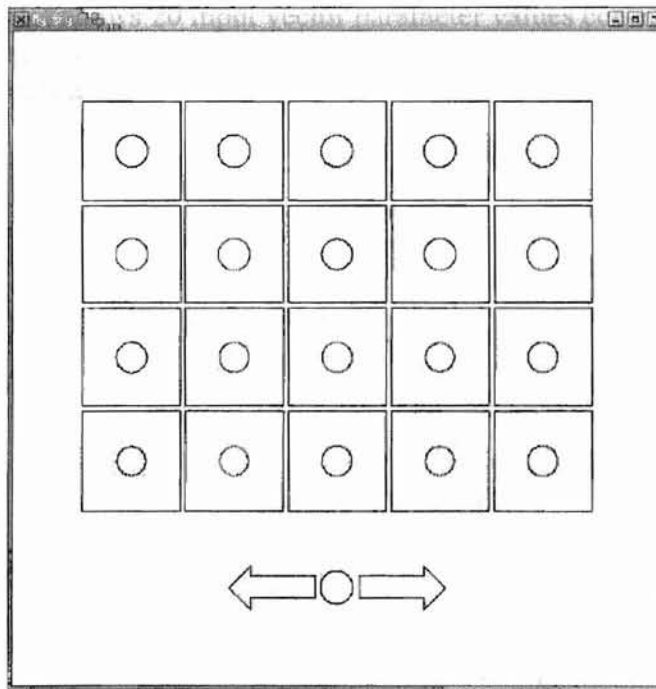
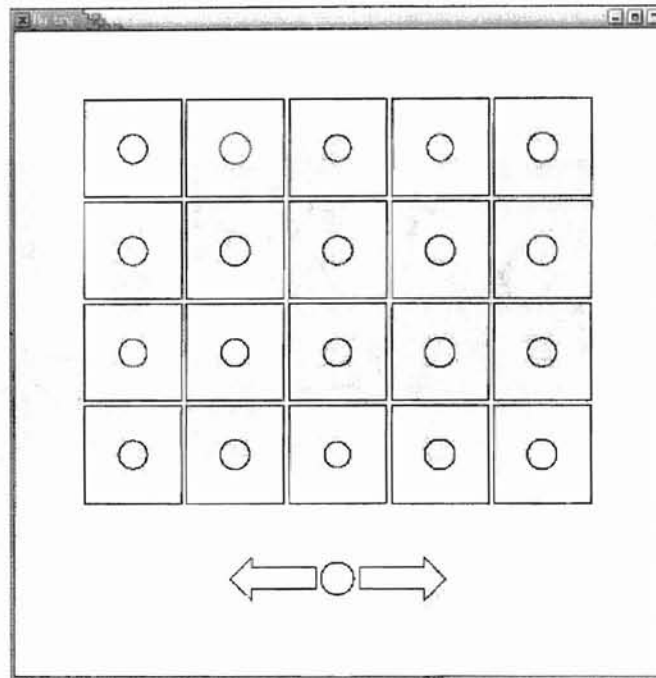


Figure 4. Practical operation 1. The top picture is a scenario for rendering circles and user Choose four out of them (row 1, column 5; row 2, column 4; row 3, column 5; row 4, column 5, from left to right, top to bottom), the bottom picture is the result based on these choices.



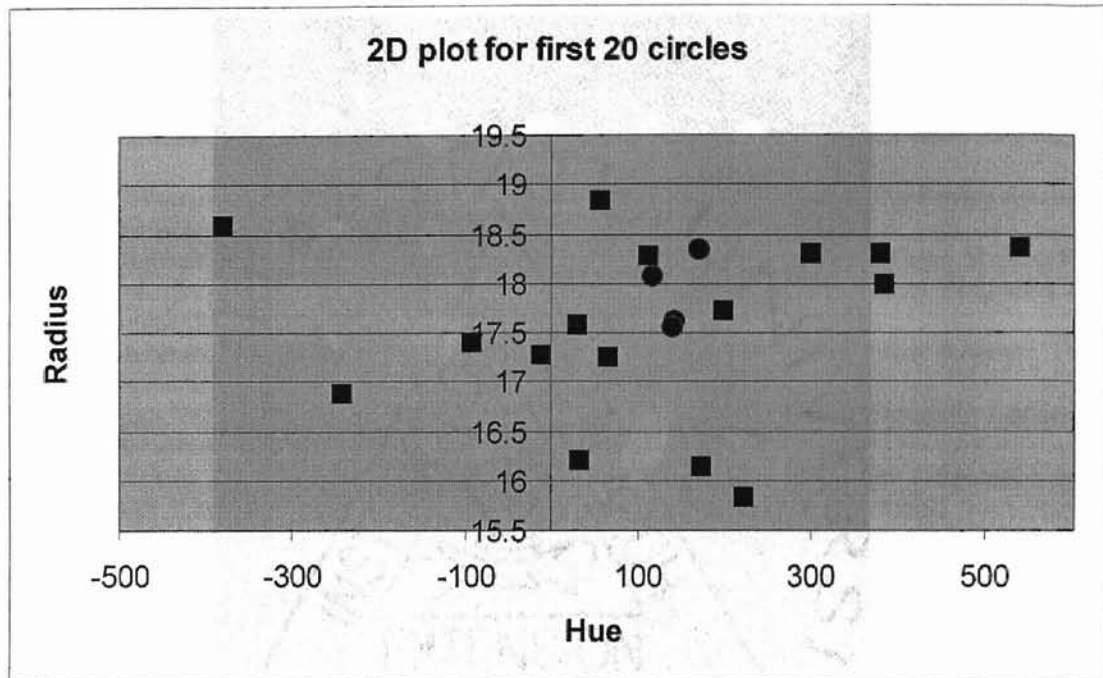


Figure 5. This plot shows 20 input vector parameter values corresponding to top 20 images in figure 4. The four points (marked as asterisks) stand for input vectors selected by user.

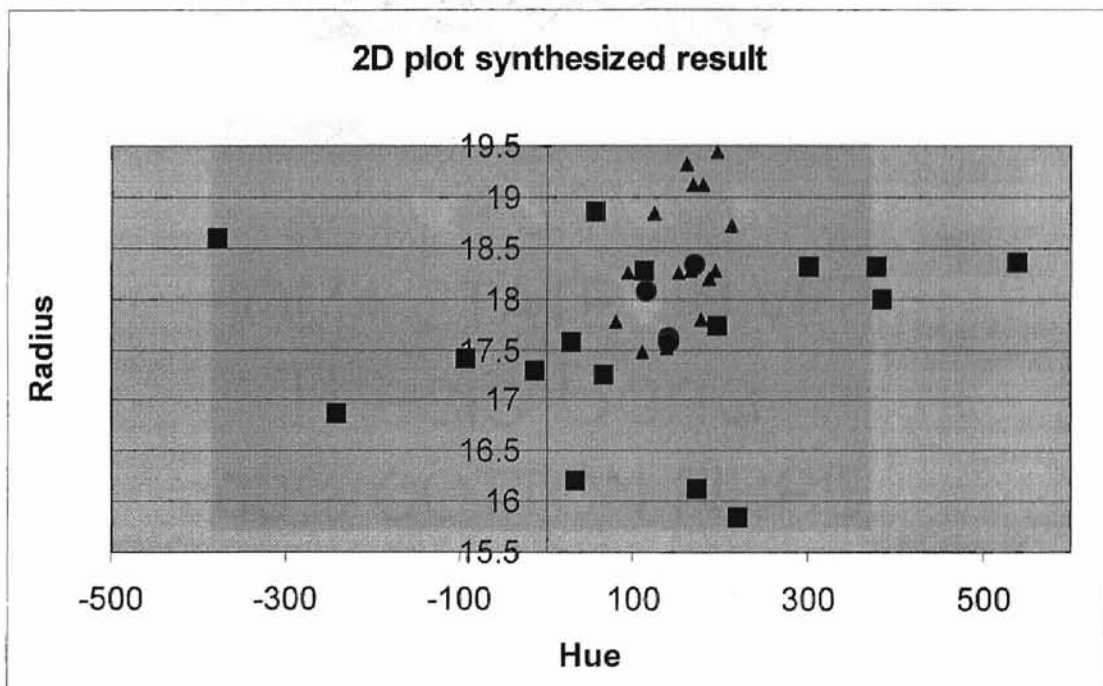
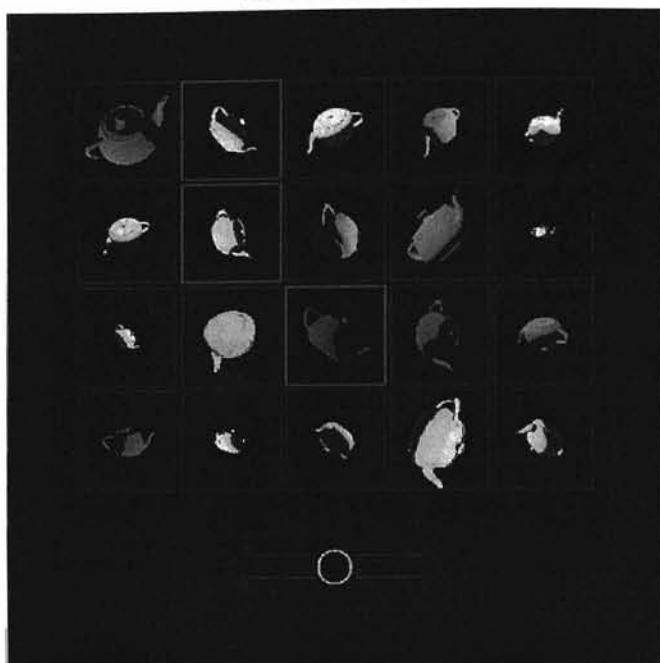


Figure 6. This plot shows 40 input vector parameter values corresponding to top and bottom 20 images in figure 4. The 20 input vectors marked as triangles are 20 input vector parameter values corresponding to bottom 20 images in figure 4.

SELECTING 3



Number of Images: 25  
Number of Selected: 3  
Total: 25  
Final Image: 3

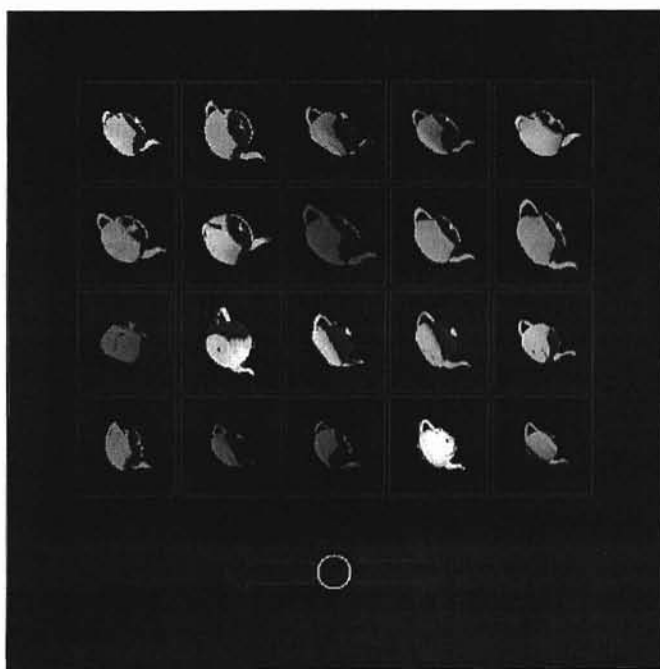


Figure 7. Practical operation 2 (The top picture is a scenario for rendering teapots and user chose three out of them (row 1, column 2; row 2, column 2; row 3, column 3, from left to right, top to bottom), the bottom picture is the result based on these choices).

## REFERENCE

- [1] C. Meihac and C. Nastar, "Relevance Feedback and Category Search in Image Database" *Proceeding IEEE International Conference, Multimedia Computer & System*, Florence, 1999, Pages 512-517, Volume 1.
- [2] Charles W. Therrien, "Decision Estimation and Classification — An Introduction to Pattern Related Topics" John Wiley & Sons, New York, 1989.
- [3] Douglas R. Heisterkamp, Jing Peng, H. K. Dai, "Feature Relevance Learning with Query Shifting for Content-based Image" *Proceedings International Conference on Pattern Recognition*, September 3-7, 2000, Barcelona, Spain, Pages 250-253, Volume 4.
- [4] David A. Grossman, Ophir Frieder, "Information Retrieval Algorithms and Heuristics" Kluwer Academic Publisher, Boston, 1998.
- [5] Edward Angel, "Interactive Computer Graphics — A Top-down Approach with OpenGL" Addison-Wesley, Massachusetts, 2000.
- [6] Georges Grinstein, "Introduction to Visualization: Vis '96 Tutorial #2" The MITRE Corporation and Univ. of Mass., Lowell, The Institute for Visualization and Perception Research, 1996.
- [7] Hong-jiang Zhang, Zheng Chen, Wen-Yin Liu and Mingjing Li, "Relevance Feedback in Content-based Image Search" *12th International Conference on New Information Technology (NIT)*, Beijing, May 29-31, 2000.
- [8] J. Marks, B. Andalman, P.A. Beardsley, W. Freeman, S. Gibson, J. Hodgins, T. Kang, B. Mirtich, H. Pfister, W. Ruml, K. Ryall, J. Seims, and S. Shieber, "Design Galleries: A General Approach to Setting Parameters for Computer Graphics and Animation" *Proceedings Of SIGGRAPH97*, Los Angeles, Aug. 1997, Pages 389-400.
- [9] J. Peng, B. Bhanu, and S. Qing, "Probabilistic Feature Relevance Learning for Content-based Image Retrieval" *Computer Vision and Image Understanding, Special Issue on Content Based Access of Image*, July/August 1999, Pages 150-164, Volume 75.
- [10] J. Peng, B. Bhanu, and S. Qing, "Learning Feature Relevance and Similarity Metrics in Image Databases" *IEEE Workshop on Content-based Access of Image and Video Libraries*, Santa Barbara, CA. USA 1998, Pages 14-18.
- [11] J.R. Bach, C. Fuller, and et al., "The Visage Image Search Engine: An Open Frame Work for Image Management" *Proceeding Of SPIE Storage and Retrieval for Image and Video Database*, 1995, Pages 76-87, Volume 2670.
- [12] K. L. Ma, "Image Graphs-A Novel Approach to Visual Data Exploration." *Proceeding IEEE Visualization 1999*, Oct. 1999, Pages 81-88.
- [13] OpenGL Architecture Review Board, Mason Woo, Jackie Neider and Tom Davis, "OpenGL™ Programming Guide: The Official Guide to Learning OpenGL, Version 1.2" Addison Wesley, Reading, Massachusetts, 1999.
- [14] N. Vasconcelos and A. Lippman, "A Probabilistic Architecture for Content-based Image Retrieval" *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, South Carolina, 2000, Pages 216-221, Volume 1.
- [15] N. Vasconcelos and M. Kunt, "Content-based Retrieval from Image Databases: Current Solutions and Future Directions" *Proceedings of International Conference on Image Processing*, Thessaloniki, Greece, 2001, Pages 6-9, Volume 3.
- [16] R. O. Stehling, M. A. Nascimento, and A. X. Falcao, "An Adaptive and Efficient Clustering-based Approach for Content-based Retrieval in Image Databases" *Technical Report 01-03, Dept. of Computer Sciences*, Univ. of Alberta, March 2001.
- [17] S. Sclaroff, L. Taycher, and M. La Casica, "Imagerover: A Content-based Image Browser for the World Wide Web" *IEEE Workshop on Content-based Access of Image and Video Libraries*, June 1997, Pages 2-9.
- [18] T.J. Jankun-Kelly and Kwan-Liu Ma, "Visualization Exploration and Encapsulation via A Spreadsheet-like Interface" *IEEE transaction on visualization and computer graphics*, July-Sept 2001, Pages 275-287, Volume 7, Issue 3.
- [19] Yong Rui, et al, "Relevance Feedback: A Power Tool for Interactive Content-based Image Retrieval" *IEEE Transactions Circuits and Video Technology*, 1995, Pages 644-655, Volume 8, Issue 5.
- [20] Y. Rui, T. Huang, and S. Mehrotra, "Content-based Image Retrieval with Relevance Feedback in

## Appendix A

### Example code

#### Portion 1.: Code for System Calculation.

```
/*
*****
Class dispersion is used to implement modified DG dispersion
algorithm.
*****
*/
class dispersion
{
private:
    const static double infinite = 2147483648.;
    unsigned int count, mark;
    double p_score, q_score, d;
    VECTOR p;
public:
    dispersion() {}
    void Selection_disperse(unsigned int size, unsigned int m, unsigned int n,
                           vector<VECTOR>L, vector<VECTOR>&I);
    double dist(VECTOR a, VECTOR b, unsigned int size);
};

double dispersion::dist(VECTOR a, VECTOR b, unsigned int size)
{
    double sum=0.0;
    for(unsigned int i=0; i<size; i++)
    {
        sum+=pow((a[i]-b[i]),2);
    }
    return sqrt(sum);
}

void dispersion::Selection_disperse(unsigned int size, unsigned int m, unsigned
int n,
vector<VECTOR>L/*in*/, vector<VECTOR>&I/*out*/)
{
    I[0]=L[0];
    count = 1;
    L.erase(L.begin());
    --m;
    for(unsigned int k=1; k<n; k++)
    {
        p_score = - infinite;
        for(unsigned int l=0; l<m; l++)
        {
            q_score = infinite;
            for(unsigned int t=0; t<count; t++)
            {
                d = dist(L[l], I[t], size);
                if(d<q_score)
                    q_score = d;
            }
        }
    }
}
```

```

        if(q_score>p_score)
        {
            p_score = q_score;
            p = L[l];
            mark = l;
        }
    }
    I[k] = p;
    ++count;
    L.erase(L.begin()+mark);
    --m;
}

/*
*****
We define a class Und, which is used to implement uniform
distribution.
*****
*/
class Und
{
public:
    Und(){}
    void mean(VECTOR &m,vector<VECTOR> MI);
    void deviation(VECTOR &d,vector<VECTOR> MI, VECTOR m);
    void selection(vector<VECTOR>&s,unsigned int num, VECTOR m, VECTOR d);

    Urand draw;
    double rand;
};

void Und::mean(VECTOR &m/*out*/,vector<VECTOR> MI/*in*/)
{
    double temp;
    unsigned int n = MI.size();
    unsigned int dim = MI[0].size();
    for(unsigned int j=0;j<dim;j++)
    {
        temp = 0.;
        for(unsigned int i=0;i<n;i++)
            temp += MI[i][j];
        m[j] = temp/n;
    }
}

/*
*****
Following method is to calculate deviations for each dimension of
vector based on a set of vectors.
*****
*/

void Und::deviation(VECTOR&d/*out*/,vector<VECTOR>MI/*in*/,VECTOR m/*in*/)
{
    double temp;
    unsigned int n = MI.size();
    unsigned int dim = m.size();
    for(unsigned int j=0;j<dim;j++)
    {
        temp = 0.;
        for(unsigned int i=0;i<n;i++)

```

```

        temp += pow((MI[i][j] - m[j]),2);
        temp = sqrt(temp/n);
        d[j] = temp;
    }
}

/*
*****
Chose some sampling points on each dimension of vector by uniform
distribution.
*****
*/
void Und::selection(vector<VECTOR> &s,unsigned int num, VECTOR m, VECTOR d)
{
    unsigned int dim = m.size();
    for(unsigned int i=0;i<num;i++)
    {
        for(unsigned int j=0;j<dim;j++)
        {
            //s[i][j] = ((double)rand()/RAND_MAX)*6. - 3.)*d[j] + m[j];
            s[i][j] = snorm()*d[j] + m[j];
        }
    }
}

```

```

/*
*****

```

(STANDARD-) N O R M A L DISTRIBUTION

```

*****
*****

```

FOR DETAILS SEE:

AHRENS, J.H. AND DIETER, U.  
EXTENSIONS OF FORSYTHE'S METHOD FOR RANDOM  
SAMPLING FROM THE NORMAL DISTRIBUTION.  
MATH. COMPUT., 27,124 (OCT. 1973), 927 - 937.

ALL STATEMENT NUMBERS CORRESPOND TO THE STEPS OF ALGORITHM 'FL'  
(M=5) IN THE ABOVE PAPER (SLIGHTLY MODIFIED IMPLEMENTATION)

Modified by Barry W. Brown, Feb 3, 1988 to use RANF instead of  
SUNIF. The argument IR thus goes away.

```

*****
THE DEFINITIONS OF THE CONSTANTS A(K), D(K), T(K) AND  

H(K) ARE ACCORDING TO THE ABOVEMENTIONED ARTICLE

```

```

*/
float snorm(void)

class transform
{
public:
    transform(){}
    void M(VECTOR&result, vector<VECTOR> B, VECTOR R)
    {
        for(unsigned int i=0;i<R.size();i++)
            result[i] = B[i][i]*R[i];
    }
}

```

```

void T(VECTOR&result, vector<VECTOR> B, VECTOR G)
{
    for(unsigned int i=0;i<G.size();i++)
        result[i] = (1.0/B[i][i])*G[i];
}
};

/*
*****
void genmn(float *parm,float *x,float *work)
    GENERate Multivariate Normal random deviate
        Arguments
parm --> Parameters needed to generate multivariate normal
    deviates (MEANV and Cholesky decomposition of
    COVM). Set by a previous call to SETGMN.
    1 : 1          - size of deviate, P
    2 : P + 1      - mean vector
    P+2 : P*(P+3)/2 + 1 - upper half of cholesky
        decomposition of cov matrix
x    <-- Vector deviate generated.
work <--> Scratch array
        Method
    1) Generate P independent standard normal deviates -  $E_i \sim N(0,1)$ 
    2) Using Cholesky decomposition find A s.t.  $\text{trans}(A)*A = \text{COVM}$ 
    3)  $\text{trans}(A)E + \text{MEANV} \sim N(\text{MEANV}, \text{COVM})$ 
*****
*/
void genmn(float *parm,float *x,float *work)

/*
*****
void setgmn(float *meanv,float *covm,long p,float *parm)
    SET Generate Multivariate Normal random deviate Function
    Places P, MEANV, and the Cholesky factorization of COVM
    in GENMN.
        Arguments
meanv --> Mean vector of multivariate normal distribution.
covm  <--> (Input) Covariance matrix of the multivariate
            normal distribution
            (Output) Destroyed on output
p     --> Dimension of the normal, or length of MEANV.
parm  <-- Array of parameters needed to generate multivariate norma
            deviates (P, MEANV and Cholesky decomposition of
            COVM).
            1 : 1          - P
            2 : P + 1      - MEANV
            P+2 : P*(P+3)/2 + 1 - Cholesky decomposition of COVM
            Needed dimension is (p*(p+3)/2 + 1)
*****
*/
void setgmn(float *meanv,float *covm,long p,float *parm)

/*
*****
Class bg is the most important portion of the implementation, which is
responsible for all tasks of calculation in the background that hidden
from user. We define tainttype as a pair of Boolean value and a VECTOR,
which is going to be used as the type of element of trainingSet. That
means the label (relevant or irrelevant) always associated with the
vector. we know if it is relevant or irrelevant when we get a input
vector.
*****

```

```

void T(VECTOR&result, vector<VECTOR> B, VECTOR G)
{
    for(unsigned int i=0;i<G.size();i++)
        result[i] = (1.0/B[i][i])*G[i];
}
};

/*
*****
void genmn(float *parm,float *x,float *work)
    GENERATE Multivariate Normal random deviate
        Arguments
parm --> Parameters needed to generate multivariate normal
        deviates (MEANV and Cholesky decomposition of
        COVM). Set by a previous call to SETGMN.
        1 : 1                - size of deviate, P
        2 : P + 1           - mean vector
        P+2 : P*(P+3)/2 + 1 - upper half of cholesky
                        decomposition of cov matrix
x    <-- Vector deviate generated.
work <--> Scratch array
        Method
1) Generate P independent standard normal deviates -  $E_i \sim N(0,1)$ 
2) Using Cholesky decomposition find A s.t.  $\text{trans}(A)*A = \text{COVM}$ 
3)  $\text{trans}(A)E + \text{MEANV} \sim N(\text{MEANV}, \text{COVM})$ 
*****
*/
void genmn(float *parm,float *x,float *work)

/*
*****
void setgmn(float *meanv,float *covm,long p,float *parm)
    SET Generate Multivariate Normal random deviate Function
    Places P, MEANV, and the Cholesky factorization of COVM
    in GENMN.
        Arguments
meanv --> Mean vector of multivariate normal distribution.
covm  <--> (Input) Covariance matrix of the multivariate
            normal distribution
            (Output) Destroyed on output
p     --> Dimension of the normal, or length of MEANV.
parm  <-- Array of parameters needed to generate multivariate normal
            deviates (P, MEANV and Cholesky decomposition of
            COVM).
            1 : 1                - P
            2 : P + 1           - MEANV
            P+2 : P*(P+3)/2 + 1 - Cholesky decomposition of COVM
            Needed dimension is  $(p*(p+3)/2 + 1)$ 
*****
*/
void setgmn(float *meanv,float *covm,long p,float *parm)

/*
*****
Class bg is the most important portion of the implementation, which is
responsible for all tasks of calculation in the background that hidden
from user. We define tainttype as a pair of Boolean value and a VECTOR,
which is going to be used as the type of element of trainingSet. That
means the label (relevant or irrelevant) always associated with the
vector. we know if it is relevant or irrelevant when we get a input
vector.
*****

```



```

*/
typedef vector<double> VECTOR;
typedef pair<bool, VECTOR> traintype;

#define Relevant true
#define Irrelevant false

class bg
{
public:
    vector<VECTOR> L,I;
    vector<unsigned int> rel;//number of relevant images for each stage
    unsigned int stage;
    vector<vector<traintype> > trainingSet;
    bool previous;

    bg(unsigned int mm, unsigned int nn, unsigned int number,
        unsigned int no_of_features);
    void calc();
    ~bg(){L.clear();I.clear();rel.clear();trainingSet.clear();
        query.clear();PMEAN.clear();NMEAN.clear();DEV.clear();}

private:
    vector<VECTOR> query,PMEAN,NMEAN,DEV;
    unsigned int m;
    unsigned int n;
    unsigned int q;
    unsigned int num;
    unsigned int size;

    unsigned int i,j,k,l;
    double alfa,beta,gama,rand;

    dispersion disp;
    transform trans;
    Und u;

};

/*
*****
We initially select some vectors from the set L, then disperse into
the set I, and we always get input vectors from the set I to generate
20 images.
*****
*/
//Background calculations
#include "bg.h"

bg::bg(unsigned int mm, unsigned int nn, unsigned int number,
    unsigned int no_of_features)//constructor
{
    m = mm; //original input vectors
    n = nn; //number of images for every stage
    num = number; //number of selected vectors
    size = no_of_features; //size of vector
    stage = 0;
    alfa = -.6;
    beta = 1.2;
    gama = .5;
    q = 10; //number of images chose by query point moving

```

```

previous = false;

query.resize(100); //query point
PMEAN.resize(100); //mean value of relevant input vectors
NMEAN.resize(100); //mean value of irrelevant input vectors
DEV.resize(100); //deviation value
rel.resize(100);
trainingSet.resize(100);
for(i=0;i<100;i++)
{
    rel[i]=0;
    query[i].resize(size);
    PMEAN[i].resize(size);
    NMEAN[i].resize(size);
    DEV[i].resize(size);
    trainingSet[i].resize(n); //twenty small images
}

L.resize(m);
I.resize(n);

for(i=0;i<m;i++)
{
    if(i<n) I[i].resize(size);
    L[i].resize(size);
}

for(i=0;i<m;i++)
{
    for(j=0;j<size;j++)
    {
        cin>>L[i][j];
    }
}

disp.Selection_disperse(size,m,n,L,I);
}

/*
*****
1. Retrieve the vectors based on relevant images.
2. Calculating weighting matrix depended on the relevant vectors.
3. Project to the modified space.
4. Solve mean and deviation values of the relevant vectors and
mean values of irrelevant vectors as well as out.
5. Sampling on the modified space (two ways).
6. Project them back to original input space.
7. Disperse ten of twenty input vectors by modified DG dispersion
algorithm and other ten input vectors are selected by query
point moving algorithm.
*****
*/
//sampling on modified space and project back to original input space
//sampling on modified space and project back to original input space
void bg::calc()
{
    vector<VECTOR> B,R,IR,M;

    /*****initialization*****/
    B.resize(size); //weighting matrix
    for(i=0;i<size;i++)

```

```

    B[i].resize(size);

R.resize(100*n);//relevant input vectors
IR.resize(100*n);//irrelevant input vectors
for(i=0;i<100*n;i++){R[i].resize(size);IR[i].resize(size);}

M.resize(num);//sampling points on modified space
for(i=0;i<num;i++)M[i].resize(size);
/*****/

j=0;
k=0;
if(previous)
{
    for(l=0;l<=stage;l++)
    {
        for(i=0;i<n;i++)
        {
            if(trainingSet[l][i].first == Relevant)
            {
                R[j]=trainingSet[l][i].second;
                j++;
            }
            else
            {
                IR[k]=trainingSet[l][i].second;
                k++;
            }
        }
    }
    previous = false;
}
else
{
    for(i=0;i<n;i++)
    {
        if(trainingSet[stage][i].first == Relevant)
        {
            R[j]=trainingSet[stage][i].second;
            j++;
        }
        else
        {
            IR[k]=trainingSet[stage][i].second;
            k++;
        }
    }
}

unsigned int rn = j;//get the number of relevant images
unsigned int irn = k;//get the number of irrelevant images

//get weighting matrix
for(i=0;i<size;i++)
    for(j=0;j<size;j++)
    {
        if(i==j)B[i][j] = 1.;
        else B[i][j] = 0.0;
    }

//Modified space
for(i=0;i<rn;i++) trans.M(R[i],B,R[i]);
for(i=0;i<irn;i++) trans.M(IR[i],B,IR[i]);

```

```

u.mean(PMEAN[stage],R,rn);
u.mean(NMEAN[stage],IR,irn);
u.deviation(DEV[stage],R,PMEAN[stage],rn);

cout<<"The number of relevant images: "<<rn<<endl;
cout<<"The number of irrelevant images: "<<irn<<endl;

for(i=0;i<size;i++)
    cout<<"Mean["<<i<<"]="<<PMEAN[stage][i]
        <<" Deviation["<<i<<"]="<<DEV[stage][i]<<endl;

if(stage==0)query[stage]=PMEAN[stage]; //query point moving
else
    for(i=0;i<size;i++)
        query[stage][i]=alfa*query[stage-1][i]
            + beta*PMEAN[stage][i]+gama*NMEAN[stage][i];

//for(i=0;i<size;i++)
    //cout<<"Query["<<i<<"]="<<query[stage][i]<<endl;

//sampling on modified space
/*
*****
Way 1: Select sample point independently by standard normal
distribution.
*****
*/
u.selection(M,num,PMEAN[stage],DEV[stage]);

/*
*****
way 2: Select sample point by taking covariance matrix into
account.
*****
*/
float *CM=new float[size*size]; //covariance matrix
for(i=0;i<size;i++)
{
    for(j=0;j<size;j++)
    {
        CM[i*size+j] = 0.;
        for(k=0;k<=rn;k++)
            CM[i*size+j]+=(R[k][i]-PMEAN[stage][i])*(R[k][j]-PMEAN[stage][j]);
        CM[i*size+j] /= rn;
    }
}

float * meanv = new float[size];
for(i=0;i<size;i++)meanv[i] = PMEAN[stage][i];

float * parm = new float[int(size*(size+3)/2. + 2.);]
//setgmn(meanv,CM,size,parm);

float * x = new float[size];
float * work = new float[size*size];

for(i=0;i<num;i++)
{
    //genmn(parm,x,work);
    //for(j=0;j<size;j++) M[i][j] = x[j];

```

```

}

/*
 *select some input vectors by modified DG dispersion algorithm
 */
disp.Selection_disperse(size,num,n,M,I);

/*
 *select some input vectors by query point moving
 */
for(i=n-q;i<n;i++)
{
    for(j=0;j<size;j++)
        I[i][j]=query[stage][j] + snorm()*DEV[stage][j];
}

/*
 *project back to original input space
 */
for(i=0;i<n;i++)
    trans.T(I[i],B,I[i]);

B.clear();
R.clear();
IR.clear();
M.clear();

delete []meanv;
delete []CM;
delete []parm;
delete []x;
delete []work;
}

```

## Portion 2.: Code for mapping process.

```

/*
*****
Following mapping process is to render circles that has an input vector
of four parameters and construct a GUI that is used to interact with
user. The first parameter is hue, the secon one is saturation, and the
third is brightness, the last one is radius of circle. In the beginning
of the program, we set up some basic parameters.
*****
 */
#include "bg.h"
#include "graphics2d.h"
#include "colorconv.h"

#define Relevant true
#define Irrelevant false

static unsigned int WindowSize = 600;
static unsigned int level;
static double zoomfactor = .15;
static const int left = -47, bottom = -28, space = 19, win_wid = 18;
static bool back = false, next = true,click = false, zoomout = false, middle;
static GLubyte* gallery[100][20];
static unsigned int i,j;
static double R,G,B;

```

```

//initialization:
//input 30 vectors, output 20 vectors,
//select 400 vectors on modified space, size of vector is 4
bg b(30,20,400,4);
colorconv cc;
Canvas cvs(WindowSize,WindowSize,"My try");

void myInit()
{
    glClearColor(1.,1.,1.,0.);
    cvs.setWindow(-60.,60.,-60.,60.);
}

void drawCircle(Point2 center, float radius)
{
    const int numVerts = 5000;
    cvs.ngon(numVerts, center.x, center.y, radius);
}

void display()
{
    if(next)
    {
        b.rel[b.stage]=0;//initialize the number of relevant images

        glClear(GL_COLOR_BUFFER_BIT);

        float p1 = -60*(1.-zoomfactor), p2 = -60*(1.-zoomfactor);

        Point2 c(p1,p2);

        for(i=0;i<20;i++)
        {
            if(b.I[i][0]>360.)b.I[i][0]=360.;
            //uper limit of h = hue --- range[0.0,360.0)
            if(b.I[i][0]<0.)b.I[i][0]=0.;
            //lower limit of h = hue --- range [0.0,360.0)
            if(b.I[i][1]>1.)b.I[i][1]=1.;
            //uper limit of s = saturation --- range [0.0,1.0)
            if(b.I[i][1]<0.)b.I[i][1]=0.;
            //lower limit of s = saturation --- range [0.0,1.0)
            if(b.I[i][2]>1.)b.I[i][2]=1.;
            //uper limit of v = brightness --- range [0.0,1.0)
            if(b.I[i][2]<0.)b.I[i][2]=0.;
            //lower limit of v = brightness --- range [0.0,1.0)
            if(b.I[i][3]>60.)b.I[i][3]=60.;//uper limit of radius
            if(b.I[i][3]<0.)b.I[i][3]=0.;//lower limit of radius

            cc.HSVtoRGB(b.I[i][0],b.I[i][1],b.I[i][2],R,G,B);
            b.trainingSet[b.stage][i] = make_pair(Irrelevant,b.I[i]);

            glColor3d(R,G,B);

            drawCircle(c,b.trainingSet[b.stage][i].second[3]*zoomfactor);
        }
    }
}

```

```

        gallery[b.stage][i] = new GLubyte[3*WindowSize*WindowSize];

        glReadPixels(0,0,(int)(WindowSize*zoomfactor),
                    (int)(WindowSize*zoomfactor) GL_RGB,
                    GL_UNSIGNED_BYTE,gallery[b.stage][i]);

        glClear(GL_COLOR_BUFFER_BIT);
    }
}

if(back || next)
{
    for(i=0;i<20;i++)
    {
        level = i/5;
        j = i%5;
        glRasterPos2i((int)(left+j*space),(int)(bottom+level*space));
        glDrawPixels((unsigned)(WindowSize*zoomfactor),
                    (unsigned)(WindowSize*zoomfactor),
                    GL_RGB,GL_UNSIGNED_BYTE,gallery[b.stage][i]);
    }
}

if(click || back || next)
{
    for(i=0;i<20;i++)
    {
        level = i/5;
        j = i%5;

        if(!b.trainingSet[b.stage][i].first) glColor3f(0.,0.,1.);
        else glColor3f(1.,0.,0.);
        //Draw 20 small windows
        cvs.moveTo((int)(left+j*space),(int)(bottom+level*space));
        cvs.lineTo((int)(left+j*space),(int)(bottom+level*space+win_wid));
        cvs.lineTo((int)(left+j*space+win_wid),
                    (int)(bottom+level*space+win_wid));
        cvs.lineTo((int)(left+j*space+win_wid),(int)(bottom+level*space));
        cvs.lineTo((int)(left+j*space),(int)(bottom+level*space));
    }

    glColor3f(0.,0.,1.);
    //Draw "Back" mark
    cvs.moveTo((int)(-20),(int)(-42));
    cvs.lineTo((int)(-16),(int)(-38));
    cvs.lineTo((int)(-16),(int)(-40));
    cvs.lineTo((int)(-4),(int)(-40));
    cvs.lineTo((int)(-4),(int)(-44));
    cvs.lineTo((int)(-16),(int)(-44));
    cvs.lineTo((int)(-16),(int)(-46));
    cvs.lineTo((int)(-20),(int)(-42));

    //Draw "Next" mark
    cvs.moveTo((int)(20),(int)(-42));
    cvs.lineTo((int)(16),(int)(-38));
    cvs.lineTo((int)(16),(int)(-40));
    cvs.lineTo((int)(4),(int)(-40));
}

```

```

cvs.lineTo((int)(4),(int)(-44));
cvs.lineTo((int)(16),(int)(-44));
cvs.lineTo((int)(16),(int)(-46));
cvs.lineTo((int)(20),(int)(-42));

//Draw "circle" mark
if(b.previous && !next && !back) glColor3f(1.,0.,0.);
else glColor3f(0.,0.,1.);
float p1 = 0, p2 = -42.;
Point2 c(p1,p2);
drawCircle(c,3.);

}

glFlush();
glutSwapBuffers();

}

void getFeedback(double x, double y)
{
    if(x>-3 && x<3 && y>-45 && y<-39)//click the circle
    {
        if(b.previous)b.previous = false;
        b.previous = true;//take previous selections into account
        back=false;next=false;click=true;
        glutSwapBuffers();
        glutPostRedisplay();
    }
    else if(x>-20 && x<-4 && y<-40 && y>-44)//click Back
    {
        back=true;next=false;click=false;
        if(b.stage>0)b.stage--;
        glutPostRedisplay();
    }
    else if(x>4 && x<20 && y<-40 && y>-44)//click Next
    {
        back=false;next=true;click=false;
        if(b.rel[b.stage]<=1)glutPostRedisplay();
        else{
            b.calc();
            b.stage++;
            glutPostRedisplay();
        }
    }
}

for(i=0;i<20;i++)
{
    level = i/5;
    j = i%5;
    if(!middle && x>(int)(left+j*space) && x<(int)(left+j*space+win_wid)
    && y>(int)(bottom+level*space) &&
    y<(int)(bottom+level*space+win_wid))//click
    {
        if(!b.trainingSet[b.stage][i].first)
        {
            b.trainingSet[b.stage][i].first = Relevant;
            b.rel[b.stage]++;
        }
        else
        {

```



```

        b.trainingSet[b.stage][i].first = Irrelevant;
        b.rel[b.stage]--;
    }
    back=false;next=false;click=true;
    glutSwapBuffers();
    glutPostRedisplay();
}
else if(middle && x>(int)(left+j*space) &&
        x<(int)(left+j*space+win_wid) &&
        y>(int)(bottom+level*space) &&
        y<(int)(bottom+level*space+win_wid))//zoomout
{
    zoomout = true;
    glClear(GL_COLOR_BUFFER_BIT);
    float p1 = 0, p2 = 0;
    Point2 c(p1,p2);
    cc.HSVtoRGB(b.trainingSet[b.stage][i].second[0],
               b.trainingSet[b.stage][i].second[1],
               b.trainingSet[b.stage][i].second[2],R,G,B);
    glColor3d(R,G,B);
    drawCircle(c,b.trainingSet[b.stage][i].second[3]);
    glutSwapBuffers();
}
}
}

void mouse(int button, int state, int X, int Y)
{
    switch(button)
    {
        case GLUT_LEFT_BUTTON://click
            if(state == GLUT_DOWN && !zoomout)
            {
                middle = false;
                getFeedback((double)((120./WindowSize)*(X-WindowSize/2.)),
                           (double)((120./WindowSize)*(WindowSize/2.-Y)));
            }
            break;
        case GLUT_MIDDLE_BUTTON://zoomout
            if(state == GLUT_DOWN && !zoomout)
            {
                middle = true;
                getFeedback((double)((120./WindowSize)*(X-WindowSize/2.)),
                           (double)((120./WindowSize)*(WindowSize/2.-Y)));
            }
            break;
        case GLUT_RIGHT_BUTTON://zoomin
            if(state == GLUT_DOWN && zoomout)
            {
                zoomout = false;
                glutSwapBuffers();
                glClear(GL_COLOR_BUFFER_BIT);
            }
            break;
        default: break;
    }
}

```

```

    }
}

void keyboard(unsigned char key, int x, int y)
{
    switch(key)
    {
        case 27: exit(0); //Esc
                break;
    }
}

int main(int argc, char **argv)
{
    myInit();
    glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB);
    glutDisplayFunc(display);
    glutMouseFunc(mouse);
    glutKeyboardFunc(keyboard);
    glutMainLoop();

    return -1;
}

/*
*****
Different input vectors have different mapping process, following
mapping process is to render teapot that has an input vector of ten
parameters and construct a GUI that is used to interact with user. The
ten parameters from first one to last one are: hue, saturation,
brightness (the three parameters for material of teapot), size of
teapot, value on x axis, value on y axis, value on z axis (the three
parameters for light position), angel rotated about x axis, angel
rotated about y axis, angel rotated about z axis (the three parameters
for orientation of teapot in the three dimensional space). In the
beginning of the program, we set some necessary parameters.
*****
*/
#include "bg.h"
#include "graphics2d.h"
#include "colorconv.h"

#define Relevant true
#define Irrelevant false

static unsigned int WindowSize = 600;
static unsigned int level;
static double zoomfactor = .15;
static const double left = -4.7, bottom = -2.8, space = 1.9, win_wid = 1.8;
static bool back = false, next = true, click = false, zoomout = false, middle;
static GLubyte* gallery[100][20];
static unsigned int i,j;
static double R,G,B;

//initialization:
//input 30 vectors, output 20 vectors,
//select 400 vectors on modified space, size of vector is 10

```

```

bg b(30,20,400,10);
colorconv cc;
Canvas cvs(WindowSize,WindowSize,"My try");

void myInit()
{
    glClearColor(0.,0.,0.,0.);
}

void drawCircle(Point2 center, float radius)
{
    const int numVerts = 1000;
    cvs.ngon(numVerts, center.x, center.y, radius);
}

void display()
{
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-6.,6.,-6.,6.,-6.,6.);
    glMatrixMode(GL_MODELVIEW);

    if(next)
    {
        b.rel[b.stage]=0;//initialize the number of relevant images

        glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);

        float p1 = -6*(1.-zoomfactor), p2 = -6*(1.-zoomfactor);

        for(i=0;i<20;i++)
        {
            if(b.I[i][0]>360.)b.I[i][0]=360.;
            //uper limit of h = hue --- range [0.0,360.0)
            if(b.I[i][0]<0.)b.I[i][0]=0.;
            //lower limit of h = hue --- range [0.0,360.0)
            if(b.I[i][1]>1.)b.I[i][1]=1.;
            //uper limit of s = saturateion --- range [0.0,1.0)
            if(b.I[i][1]<0.)b.I[i][1]=0.;
            //lower limit of s = saturateion --- range [0.0,1.0)
            if(b.I[i][2]>1.)b.I[i][2]=1.;
            //uper limit of v = brightness --- range [0.0,1.0)
            if(b.I[i][2]<0.)b.I[i][2]=0.;
            //lower limit of v = brightness --- range [0.0,1.0)
            if(b.I[i][3]<0.)b.I[i][3]=0.;//lower limit of teapot's size
            if(b.I[i][3]>4.)b.I[i][3]=4.;//uper limit of teapot's size

            b.I[i][7]-=int(b.I[i][7]/360.)*360;
            b.I[i][8]-=int(b.I[i][8]/360.)*360;
            b.I[i][9]-=int(b.I[i][9]/360.)*360;

            b.trainingSet[b.stage][i] = make_pair(Irrelevant,b.I[i]);

            cc.HSVtoRGB(b.I[i][0],b.I[i][1],b.I[i][2],R,G,B);
            GLfloat mat_specular[] = {R,G,B,1.};
            GLfloat mat_ambient[] = {R,G,B,1.};
        }
    }
}

```

```

//GLfloat mat_diffuse[] = {R,G,B,1.};
//GLfloat mat_shininess[] = {50.};
GLfloat light_position[] = {b.trainingSet[b.stage][i].second[4],
                           b.trainingSet[b.stage][i].second[5],
                           b.trainingSet[b.stage][i].second[6],0.};
GLfloat white_light[] = {1.,1.,1.,1.};
glShadeModel(GL_SMOOTH);
glMaterialfv(GL_FRONT_AND_BACK,GL_SPECULAR,mat_specular);
glMaterialfv(GL_FRONT_AND_BACK,GL_AMBIENT_AND_DIFFUSE,mat_ambient);
//glMaterialfv(GL_FRONT_AND_BACK,GL_DIFFUSE,mat_diffuse);
//glMaterialfv(GL_FRONT_AND_BACK,GL_SPECULAR,mat_shininess);

glLightfv(GL_LIGHT0,GL_POSITION,light_position);
glLightfv(GL_LIGHT0,GL_DIFFUSE,white_light);
glLightfv(GL_LIGHT0,GL_SPECULAR,white_light);

glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
glEnable(GL_DEPTH_TEST);

glPushMatrix();
glTranslated(p1,p2,0);
glRotated(b.trainingSet[b.stage][i].second[7],1,0,0);
glRotated(b.trainingSet[b.stage][i].second[8],0,1,0);
glRotated(b.trainingSet[b.stage][i].second[9],0,0,1);
glutSolidTeapot(b.trainingSet[b.stage][i].second[3]*zoomfactor);
glPopMatrix();

gallery[b.stage][i] = new GLubyte[3*WindowSize*WindowSize];

glReadPixels(0,0,(int)(WindowSize*zoomfactor),
            (int)(WindowSize*zoomfactor),
            GL_RGB,GL_UNSIGNED_BYTE,gallery[b.stage][i]);

glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
}
}
if(back || next)
{
for(i=0;i<20;i++)
{
level = i/5;
j = i%5;
glRasterPos2f(left+j*space,bottom+level*space);
glDrawPixels((unsigned int)(WindowSize*zoomfactor),
            (unsigned int)(WindowSize*zoomfactor),
            GL_RGB,GL_UNSIGNED_BYTE,gallery[b.stage][i]);
}
}
if(click || back || next)
{
glDisable(GL_LIGHTING);
glDisable(GL_LIGHT0);
glDisable(GL_DEPTH_TEST);

for(i=0;i<20;i++)

```

```

{
    level = i/5;
    j = i%5;

    if(!b.trainingSet[b.stage][i].first) glColor3f(0.,0.,1.);
    else glColor3f(1.,0.,0.);
    //Draw 20 small windows
    cvs.moveTo(left+j*space,bottom+level*space);
    cvs.lineTo(left+j*space,bottom+level*space+win_wid);
    cvs.lineTo(left+j*space+win_wid,bottom+level*space+win_wid);
    cvs.lineTo(left+j*space+win_wid,bottom+level*space);
    cvs.lineTo(left+j*space,bottom+level*space);

}

glColor3f(0.,0.,1.);
//Draw "Back" mark
cvs.moveTo(-2.0,-4.2);
cvs.lineTo(-1.6,-3.8);
cvs.lineTo(-1.6,-4.0);
cvs.lineTo(-.4,-4.0);
cvs.lineTo(-.4,-4.4);
cvs.lineTo(-1.6,-4.4);
cvs.lineTo(-1.6,-4.6);
cvs.lineTo(-2.0,-4.2);

//Draw "Next" mark
cvs.moveTo(2.0,-4.2);
cvs.lineTo(1.6,-3.8);
cvs.lineTo(1.6,-4.0);
cvs.lineTo(.4,-4.0);
cvs.lineTo(.4,-4.4);
cvs.lineTo(1.6,-4.4);
cvs.lineTo(1.6,-4.6);
cvs.lineTo(2.0,-4.2);

//Draw "Stop" mark
if(b.previous && !next && !back) glColor3f(1.,0.,0.);
else glColor3f(1.,1.,0.);
float p1 = 0, p2 = -4.2;
Point2 c(p1,p2);
drawCircle(c,.3);

}

glFlush();
glutSwapBuffers();

}

void getFeedback(double x, double y)
{
    if(x>-.3 && x<.3 && y>-4.5 && y<-3.9)//click the circle
    {
        if(b.previous)b.previous = false;
        else b.previous = true;//take previous selections into account
        back=false;next=false;click=true;
        glutSwapBuffers();
        glutPostRedisplay();
    }
    else if(x>-2.0 && x<-.4 && y<-4.0 && y>-4.4)//click Back

```

```

{
    back=true;next=false;click=false;
    if(b.stage>0)b.stage--;
    glutPostRedisplay();
}
else if(x>.4 && x<2.0 && y<-4.0 && y>-4.4)//click Next
{
    back=false;next=true;click=false;
    if(b.rel[b.stage]<=1)glutPostRedisplay();
    else{
        b.calc();
        b.stage++;
        glutPostRedisplay();
    }
}
}

for(i=0;i<20;i++)
{
    level = i/5;
    j = i%5;
    if(!middle && x>(left+j*space) && x<(left+j*space+win_wid) &&
        y>(bottom+level*space) && y<(bottom+level*space+win_wid))
        //click small windows
    {
        if(!b.trainingSet[b.stage][i].first)
        {
            b.trainingSet[b.stage][i].first = Relevant;
            b.rel[b.stage]++;
        }
        else
        {
            b.trainingSet[b.stage][i].first = Irrelevant;
            b.rel[b.stage]--;
        }
        back=false;next=false;click=true;
        glutSwapBuffers();
        glutPostRedisplay();
    }
    else if(middle && x>(left+j*space) && x<(left+j*space+win_wid) &&
        y>(bottom+level*space) && y<(bottom+level*space+win_wid))//zoomout
    {
        zoomout = true;
        glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);

        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        glOrtho(-6.,6.,-6.,6.,-6.,6.);
        glMatrixMode(GL_MODELVIEW);

        cc.HSVtoRGB(b.I[i][0],b.I[i][1],b.I[i][2],R,G,B);
        GLfloat mat_specular[] = {R,G,B,1.};
        GLfloat mat_ambient[] = {R,G,B,1.};
        GLfloat light_position[] = {b.trainingSet[b.stage][i].second[4],
            b.trainingSet[b.stage][i].second[5],
            b.trainingSet[b.stage][i].second[6],
            0.};

        GLfloat white_light[] = {1.,1.,1.,1.};
        glShadeModel(GL_SMOOTH);
        glMaterialfv(GL_FRONT_AND_BACK,GL_SPECULAR,mat_specular);
        glMaterialfv(GL_FRONT_AND_BACK,GL_AMBIENT_AND_DIFFUSE,mat_ambient);
        glLightfv(GL_LIGHT0,GL_POSITION,light_position);
        glLightfv(GL_LIGHT0,GL_DIFFUSE,white_light);
    }
}

```

```

glLightfv(GL_LIGHT0, GL_SPECULAR, white_light);
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
glEnable(GL_DEPTH_TEST);

glPushMatrix();
glRotated(b.trainingSet[b.stage][i].second[7], 1, 0, 0);
glRotated(b.trainingSet[b.stage][i].second[8], 0, 1, 0);
glRotated(b.trainingSet[b.stage][i].second[9], 0, 0, 1);
glutSolidTeapot(b.trainingSet[b.stage][i].second[3]);
glPopMatrix();

glutSwapBuffers();
}
}
}

void mouse(int button, int state, int X, int Y)
{
    switch(button)
    {
        case GLUT_LEFT_BUTTON://click
            if(state == GLUT_DOWN && !zoomout)
            {
                middle = false;
                getFeedback((double)((12./WindowSize)*(X-WindowSize/2.)),
                    (double)((12./WindowSize)*(WindowSize/2.-Y)));
            }
            break;
        case GLUT_MIDDLE_BUTTON://zoomout
            if(state == GLUT_DOWN && !zoomout)
            {
                middle = true;
                getFeedback((double)((12./WindowSize)*(X-WindowSize/2.)),
                    (double)((12./WindowSize)*(WindowSize/2.-Y)));
            }
            break;
        case GLUT_RIGHT_BUTTON://zoomin
            if(state == GLUT_DOWN && zoomout)
            {
                zoomout = false;
                cvs.setWindow(-6., 6., -6., 6.);
                //chage back to original coordinate system
                glutSwapBuffers();
                glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
            }
            break;
        default: break;
    }
}

void keyboard(unsigned char key, int x, int y)
{

```

```

switch(key)
{
    case 27: exit(0); //Esc
            break;
}
}

int main(int argc, char **argv)
{
    myInit();
    glutInitDisplayMode (GLUT_DOUBLE|GLUT_RGB|GLUT_DEPTH);
    glutDisplayFunc (display);
    glutMouseFunc (mouse);
    glutKeyboardFunc (keyboard);
    glutMainLoop();

    return -1;
}

```

## Appendix B

### Glossary

DG: Design Gallery.

CBIR: Content Based Image Retrieve.

RF: Relevance Feedback.

GL: OpenGL

GUI: Graphical User Interface

trainingSet: Training Set.

Relevant: Relevant images.

Irrelevant: Irrelevant images.

Back Buffer: One of the buffers in two buffers' rendering system.



VITA 2

Peng Xiang

Candidate for the Degree of

Master of Science

Thesis: LOCAL FEATURE RELEVANCE FOR EFFICIENT NAVIGATION AND  
VISUALIZATION OF LARGE DATA SETS

Major Field: Computer Science

Biographical:

Personal Data: Born in Yaan, Shi Chuan Province, China, On June 6, 1971.

Education: Received Bachelor of Science degree in Mechanical Engineering from  
Nanjing University of Aeronautics and Astronautics, China in 1994.

Completed the requirements for the Master of Science degree with a major in  
Computer Science at Oklahoma State University in August 2002.

Experience: Raised in a factory in Yaan City; employed by Chuan Jiang Machinery  
Company as an engineer; employed by Xin Ke Magnetic Products Company as  
a supervisor; employed by Hongguan Technology Lt. Co. as an engineer;  
Oklahoma State University, Department of Computer Science, 2001 to present.