A Method for the Perceptual Optimization of Complex Visualizations

Donald House   and   Colin Ware
Visualization Laboratory   Data Visualization Research Lab
Texas A&M University   Center for Coastal and Ocean Mapping
house@viz.tamu.edu   University of New Hampshire
                                     colinw@cisunix.unh.edu

Abstract

A common problem in visualization applications is the display of one surface overlying another. Unfortunately, it is extremely difficult to do this clearly and effectively. Stereoscopic viewing can help, but in order for us to be able to see both surfaces simultaneously, they must be textured, and the top surface must be made partially transparent. There is also abundant evidence that all textures are not equal in helping to reveal surface shape, but there are no general guidelines describing the best set of textures to be used in this way. What makes the problem difficult to perceptually optimize is that there are a great many variables involved. Both foreground and background textures must be specified in terms of their component colors, texture element shapes, distributions, and sizes. Also to be specified is the degree of transparency for the foreground texture components. Here we report on a novel approach to creating perceptually optimal solutions to complex visualization problems and we apply it to the overlapping surface problem as a test case. Our approach is a three-stage process. In the first stage we create a parameterized method for specifying a foreground and background pair of textures. In the second stage a genetic algorithm is applied to a population of texture pairs using subject judgments as a selection criterion. Over many trials effective texture pairs evolve. The third stage involves characterizing and generalizing the examples of effective textures. We detail this process and present some early results.

Introduction

One of the most difficult problems in data visualization is that of displaying one surface overlaying another. This occurs in medical imaging – e.g. in seeing the shape of tissue overlying a skeleton; in mining applications – e.g. in seeing how an ore body is situated beneath the surface of the earth; in oceanography – e.g. in seeing how surfaces defined by temperature changes relate to the underlying topography of the seabed. The problem is

that with a static picture we can really only see one surface in one place at a time. Even when the top surface is made transparent it can be very difficult to see both surfaces simultaneously since the shape-from-shading information from the two surfaces is confounded and it can be impossible to visually separate.

Two things can help. If one or both surfaces are given a partially transparent texture this can help to define and distinguish them (Interrante et al., 1997). Also additional depth information provided through stereoscopic viewing and motion parallax can make them stand apart. The remaining problem is how to choose the pair of textures so that they both optimally reveal surface shape and do not interfere with one another. This is essentially a problem of perception, but because textures can be arbitrarily complex it is one that is not easy to solve.   A poor choice of textures can be disastrous, but it can take ten or twenty parameters to define a complex texture with a reasonably complex set of texture elements and color components. Then, there is the issue of how the textures should be oriented with respect to the viewpoint and the surface topography. In the face of such complex visualization problems, the reluctance of many visualization researchers to evaluate the effectiveness of display techniques is hardly surprising.

In traditional methods used in vision research for studying how such parameters affect perception, it is normal to vary parameters one at a time and measure how error changes in some simple task. For example, there have been parametric studies of the way we can perceive curved surfaces (e.g. Todd and Akerstrom, 1987; Cumming et al., 1993), but such studies generally look only at one or two variables and it is difficult to infer general rules for complex display problems where the interaction of display parameters is unknown. Carrying out a systematic parametric study of all the variables that may contribute to the perception of one surface overlying another would be prohibitively time consuming.   If variables interact, the number of measurements required is exponential in the number of display parameters.

Because of this problem, the main focus of our effort and the subject if this paper became a new approach to solving visualization problems. The approach we arrived at is shown in block diagram form in Figure 1. In broad outline it consists of the following stages.

Stage 1. Develop a parameterization of the problem. The problem must be represented so that a vector of parameters controls the visual representation of the data. An encoding of this vector becomes the genome for stage 2.

Stage 2. Use a genetic algorithm (Fogel, 1999) where the selection process during evolution is guided by user preferences in order to progressively produce an improved set of solutions (encoded in genomes) to the visualization problem. Since the algorithm is user guided, evaluation is built into the process of producing better solutions.

Stage 3. Characterize the results. The output from such a process can be applied in a number of ways. Ranging from the least to the most general these are: an optimized visualization for a specific problem, a description of the characteristics of an effective visualization for a generalized problem, a formal description of the characteristics of an effective visualization that can be applied automatically or algorithmically, an informal perceptual theory of visual mappings that create an effective visualization, a formal perceptual theory of visual mappings that create an effective visualization.
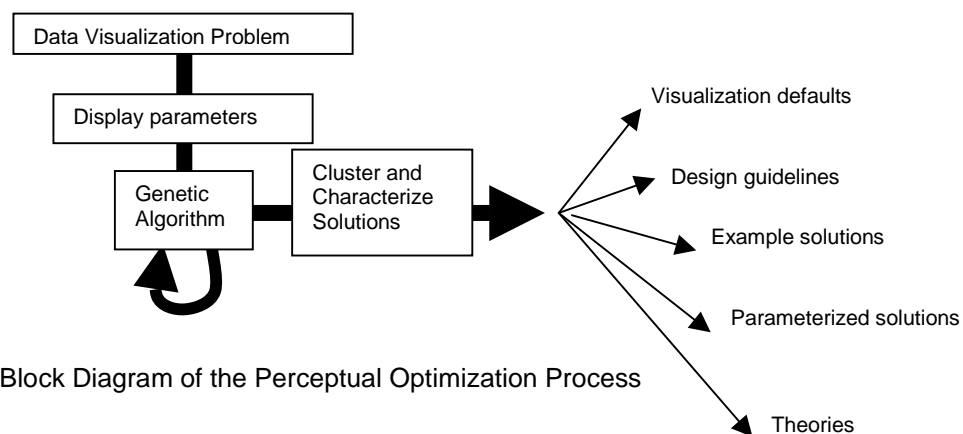
Figure 1. Block Diagram of the Perceptual Optimization Process

The encoding of a visualization problem in the form of a vector of parameters determines a search space within which all solutions derived from this encoding must lie. The dimension of this search space is the number of parameters, which for nearly all

interesting visualization problems is very large. If the parameters take on values over a continuous range of real numbers, this space is infinite, but even if each parameter can take on only a small number of discrete values, the size of the space will be much too large for any kind of exhaustive search. Thus, we can conclude that our methods must of necessity be approximate.

The genetic algorithm approach to searching a high dimensional parameter space is a good choice for our purposes, because it can be organized to provide a search that rapidly explores the space looking for promising regions, and is somewhat resilient in avoiding poor local minima. Most importantly, as has been shown by Dawkins (1986) and Sims (1991), the structure of a genetic algorithm provides a convenient for visualization and user feedback. The algorithm continuously maintains a relatively small population of potential solutions called a generation. Since each solution encodes a visualization, it is straightforward to produce the visualization for each member of a generation and request that a human subject provide an evaluation or score, rating the solution's quality. Each rated solution can be used to guide the progress of the algorithm and also stored for later analysis. The process is capable of quickly producing a large set of generally improving pre-evaluated visualization solutions.

The characterization of results is as yet the hardest to elucidate in the form of a distinct process. However, some things are clear. If the goal is to obtain a specific solution to a specific visualization problem it will probably be satisfactory for a single expert user to develop that solution by interacting with the genetic algorithm and simply stopping when a satisfactory solution is obtained. If, however, the goal is to obtain a set of guidelines for obtaining a good solution to a class of problems then it will probably be necessary for a number of subjects to develop a large set of evaluated solutions, which are later analyzed. As was pointed out by Marks et al. (1997), a logical approach to the analysis phase is to begin by grouping the solutions into *clusters,* all of which share similar parameterizations. A cluster consisting of a number of solutions that share high ratings becomes a candidate for further analysis. One characterization of a good solution, then, would simply be a description of the salient features of a highly rated cluster. The

analysis could proceed by collecting a number of such characterizations and looking for shared features, which would potentially lead to a set of rules or procedures to follow in developing successful visualizations for a class of problems.

Before going on to elaborate our methodology for attacking the specific problem of layered surfaces, we first review some of the relevant prior work into the depiction of surface shape.

Perceptual Issues

Surfaces in nature are generally textured. Gibson (1986) took the position that surface texture is an essential property of a surface. A non-textured surface, he said, is merely a patch of light. Texturing surfaces is especially important when they are viewed stereoscopically. This becomes obvious if we consider that a uniform non-textured polygon contains no *internal* stereoscopic information about the surface that it represents. Under uniform lighting conditions, such a surface also contains no orientation information. When a polygon is textured, every texture element provides stereoscopic depth information, relative to neighboring points. A study by Norman et al. (1995) showed that texture, stereoscopic viewing, shape from shading and motion parallax all contribute to the perception of surface orientation for smoothly curving surfaces. However, they used random speckle textures and there is evidence that oriented elongated texture elements may be more effective in revealing shape.

Contours that are drawn on a shaded surface can drastically alter the perceived shape of a surface (Ramachandran 1988). Figure 2 has shaded bands that are added to provide internal contour information. The sequence of gray values is exactly the same on the left as on the right. Thus the two rectangular areas contain exactly the same shading but different internal contours. The combination of contour information with shading information is convincing in both cases but the surface shapes that are perceived are very different. This tells us that shape from shading information is inherently ambiguous; it can be interpreted in different ways depending on the contours.
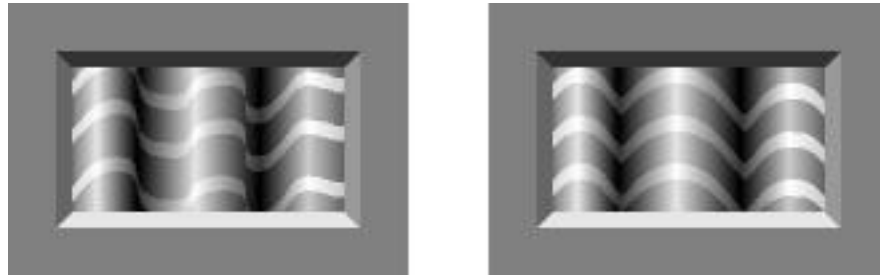
Figure 2. The left to right sequence of gray values in the left pattern is the same as that in the right. The only difference is the contour information. This interacts with the shading information to produce the perception of two very differently shaped surfaces. (From Ware, 1999)

One of the most common ways of representing surfaces is through the contour map. A contour map is a plan view representation of a surface with iso-height contours usually spaced at regular intervals. Conceptually, each contour can be thought of as the line of intersection of a horizontal plane with a particular value in a scalar height field . Although reading contour maps is a skill that takes practice and experience, contour maps should not necessarily be regarded entirely as arbitrary graphical conventions. Contours can be thought of as a special kind of texture that follows the form of a surface.

To see one surface through another, the top surface must be at least partially transparent Watanabe and Cavanaugh (1996) developed the concept of "laciness" to describe when a see-through texture will be readily distinguished from another lying behind it. They provided some interesting examples showing that a poor choice of texture pairs will perceptually fuse into a single pattern, whereas other combinations will be clearly distinct. However, they provided no general guideline that might be applied in visualization algorithms.

Stereoscopic viewing
It seems extremely likely that different textures will be effective with stereosopic viewing in comparison with non-stereoscopic viewing and certain orientations of texture elements will be more effective than others. Rogers and Cagnello (1989) have shown that we are most sensitive to curvature with vertically contoured surfaces viewed in stereo.

Structure from motion.

When objects move, or we move through the environment, the shapes that appear on the retina change correspondingly. The brain can use these moving patterns to infer spatial layout. When surfaces are in motion we can perceive their shape more accurately. For certain visualization tasks, structure-from-motion can be more effective than stereoscopic viewing (Ware and Franck, 1996).

Non-photorealistic rendering

Artists sometimes use short brush or pen strokes call *hache* lines in order to represent a three-dimensional surface shape in drawings and engraving, especially for medical and scientific illustrations. When drawing, the artist may vary the width and spacing of each stroke, but most importantly the artist specifies stroke direction with respect to the surface. Henry Pitz (1957) stated the following general rules for artists

> *horizontal lines make objects appear wider*
> *vertical lines make objects appear taller*
> *lines following the contour of the surface emphasize the surface shape*
> *random lines destroy the surface shape*

The purpose of *non-photorealistic rendering* has been to create pleasing and informative illustrations, or sometimes aesthetically pleasing effects. For example consider recent work to mimic impressionist painting or copperplate engraving (Lansdown and Schofield, 1995; Winkelbach and Salesin, 1996), but other work has been aimed at scientific visualization. Saito and Takahashi (1990) investigated various ways of enhancing map renderings using contours, shape from shading and enhanced silhouettes. However, little of this work has had any evaluation of the perceptual effectiveness of the solutions. One exception is Interrante, et. al. (1997) who examined subjects' abilities to judge the distance between two surfaces. They created experimental see-through textures designed to reveal one curved surface lying above another. On the top surface, contour lines were drawn to be at right angles to the surface in regions of maximum curvature. The results showed that texture pattern had little influence on the judgment of spatial separation. But this is this is not surprising since stereoscopic depth only relies on

disparities that will be present in most textures. The task of judging the shape of a surface is much more likely to be influenced by the texture pattern that is used.

The Problem of Layered Surfaces

The problem that lead us to develop our methodology involving the genetic algorithm is specifically that of being able to view two overlapping or layered surfaces, while maintaining the ability to understand both the shapes of the two surfaces and their spatial relationships to each other. It is quite difficult to imagine a satisfactory approach to this problem involving a single static image, as our ability to differentiate overlapping surfaces comes largely from a combination of stereo and motion cues together with surface texture. It was clear to us from the start that any exploration of the problem would start with a visualization that made use of both stereo and motion and then explored the effects of applying different textures to the two overlapping surfaces. However, what was not at all clear was how to select and then vary the textures. Exactly what characteristics of the textures would we manipulate and how would we account for the visual interactions between textures on the lower and upper surfaces? The problem was clearly too large for a standard study providing for the manipulation of only a few variables. Once we realized that we would need to be able to vary a large number of texture parameters in order to adequately explore the problem, the general outline of our new approach began to form.

The parameterized visualization search space

Our first step was to develop a parameterized texture space that would allow us to test a number of texture attributes that we thought might bear on the layered surface problem. The overall attributes of the texture that seemed most important to us were: 1) orientation, 2) foreground transparency, 3) density of pattern, 4) regularity of pattern (i.e. structured vs. random), and 5) softness (i.e. soft vs. hard edges), and 6) background color. The attributes of individual texture elements that seemed most important to us were: 1) transparency, 2) size, 3) linearity (i.e. long vs. short strokes vs. dots), 4) orientation, and 5) color.

In our method, each texture tile is structured as a set of three lattices. A complete texture across a surface is tiled from a single base tile using a standard texture-mapping approach. A lattice simply divides the tile into a uniform square grid. The center of each grid cell becomes the origin of a local coordinate system within which is drawn a lattice element. The elements available in our method are oriented line segments and filled dots. For example, Figure 4 shows a tile divided into a 6-by-6 lattice with a dot centered at the origin of each cell. Parameters affecting the global characteristics of the tile control the overall orientation of the tiling on the surface, the color and opacity of the tile background, and the kernel width of a gaussian filter applied to the final texture, which affects the overall softness of the texture (a width of 1 indicates no filtering, giving a hard-edged look, while a high width produces a soft-edged look). Each tile is composed or layered from three lattices. The texture elements on the bottom layer are dots, and those of the top two layers are line segments. Parameters affecting the appearance of each lattice control the number of grid cells both horizontally and vertically, the color and opacity of the lattice element, the orientation and position of the center of each lattice element in its cell, the randomness of position and orientation and position of each element, the length and thickness of line segments or radius of dots, and the probability that an individual element will be drawn in its cell. The size, position and orientation parameters of an element are relative to the size and position of the cell within which it is drawn. The choice of two layers of lines and one of dots assured us that each texture would have the potential of having multiple linear elements along with point-like elements. This does not mean that all textures have all three elements. The presence of transparency, size and probability parameters makes it possible that an individual lattice does not get drawn at all.
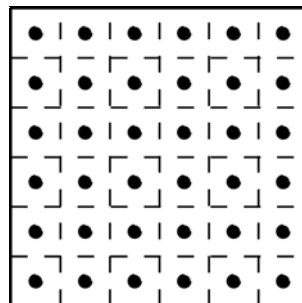

Figure 3. A 6x6 Lattice with Dots

This structure allowed us to parameterize a single texture in a vector of 61 elements. 7 parameters control overall appearance, and 18 parameters control each of the three lattices. Thus, a complete solution to the visualization problem (i.e. textures for both the bottom and the top surfaces) is parameterized in a vector of 122 elements.

The genetic algorithm

The genetic algorithm as we implemented it is shown in Figure 4. In the figure, a Generation is an array of genotypes, where each genotype is an array of genes, and each gene encodes a parameter of the visualization. For our experiments we use a generation size of 40 genotypes. Our genotypes each contain 122 genes, and each gene is encoded as an unsigned integer number. A Phenotype is the expression of a genotype; i.e. it is one complete visualization solution. In our parameterization, the expression of each gene is either a floating point or integer number controlling an aspect of the texture pair. For example, the genes that control tile orientation are each expressed as a floating point number from –90 to 90 degrees. An Evaluation is an array of "fitness" scores, with entries corresponding to genotype entries in the Generation array. In our system, each evaluation is an integer score ranging from 0 (meaning that the corresponding phenotype is completely unsuitable) to 9 (meaning excellent).

The algorithm starts with a generation of random genotypes. It then iteratively searches for better solutions by repeating the following process: 1) display the visualization represented by each genotype and obtain user evaluations, 2) record evaluated genotypes in a history file, 3) select pairs from the generation to "breed" probabilistically based on the user evaluations (such that higher rated genotypes are more likely to be selected to breed), 4) breed two offspring into the new generation from each selected pair from the current generation, 5) make the new generation the current generation, and 6) probabilistically mutate the new offspring. In the case where the algorithm is being restarted, the most recent generation and evaluation pairs are loaded from the history file and the first iteration of the algorithm starts with the selection of pairs for breeding.

```
Generation G, V of size N;                              G current, V next generation, N even
Evaluation E of size N;
Phenotype P;

if restarting from a previous session then
    (G, E) = LoadFromHistoryFile();                     restart from last saved generation
    goto restart;                                       and begin by breeding
else
    RandomlyInitialize(G)                               start anew with random generation
endif

loop

    for each genotype Gi in G do
        P = Phenotype(Gi);                              extract visualization from the genotype
        Ei = UserEvaluation(Display(P));                display and get user evaluation
    endfor

    SaveEvaluatedGenotypesToFile(G, E);                 save evaluated generation for analysis

   restart:
    for k = 1 to N step 2 do                            breed to create a new generation
        (i, j) = SelectBreedingPair(G, E);              selection probabilistic based on eval
        (Vk, Vk+1) = CrossoverBreed(Gi, Gj);            a breeding pair produces 2 offspring
    endfor

    G = V;                                              make new generation the current one

    for each genotype Gi in G do
        Mutate(Gi);                                     genes mutate with low probability

until UserRequestsExit;                                 keep going until user is tired
```

Figure 4. Overview of the Genetic Algorithm

In the algorithm as implemented, breeding is done from a pair of genotypes using a two-point crossover method, with the constraint that the two points are kept equidistant on a circular scale. This works as follows: 1) the genotype of child 1 is initialized by copying the genotype from parent 1 and the genotype of child 2 from parent 2, 2) a crossover division point (i.e. an index into the genotype array) is randomly selected, 3) genes to the right of the division point are copied from parent 2 to child 1 and from parent 1 to child 2. The copying (or crossover) of genes from the opposite parent is done in circular fashion, wrapping back to the start of the genotype, so that each child has exactly half of its genes from parent 1 and half from parent 2.

The mutation approach is somewhat non-standard but quite simple. Each gene in each genome is probabilistically selected to be replaced with a random gene value (instead of using the common approach of bit inversion). The probability of selection is typically kept quite low so that mutation simply becomes a way for the algorithm to "probe" the search space in areas around current solutions. In our current experiments we are using a probability of 0.05.

Our experiments showed that it is possible to evaluate one generation of 40 genotypes in about 5 minutes, so that 12 or more generations can be evaluated in one hour. Using our parameterization we usually converged to a reasonable set of phenotypes in about 20 generations. After this point, the population of a generation becomes somewhat uniform so it is more fruitful to restart the genetic algorithm to look for new solutions, rather than to continue with the current gene pool.

The clustering algorithm

Having run the genetic algorithm to a good stopping point, the evaluated genomes must be clustered so that solutions can be characterized. The process that we used proceeds as follows. First we load the entire history file and discard all genotypes with evaluations below a preset threshold. In our recent work we have been using a threshold of 8 on our scale from 0 to 9. We then convert each genotype into its corresponding phenotype and normalize using an affine transformation to convert the expression of each gene in the phenotype to a fractional floating point number in the range 0 to 1. For example, the expression of a gene encoding orientation of the texture is in the range -90 to 90 degrees so it is converted to a fraction from 0 to 1, where 0 corresponds with –90 degrees and 1 corresponds with 90 degrees. This provides a uniform set of coordinates with which to measure in the 122-dimensional space of solutions and constrains all solutions to fall within the unit hypercube in this space. Thus a convenient "yardstick" by which to make Euclidean measurements in this space is the length of the main diagonal of this hypercube (i.e. the square root of the dimension of the space) since this is the maximum distance in the space.

After this preprocessing, clustering proceeds using a thresholded minimum distance criterion for cluster membership. The process is initialized by clustering each solution in a pair with the solution it is nearest to (all measurements are Euclidean). The process proceeds upwards by merging clusters, always using the minimum distance criterion for cluster membership. If the minimum distance between a member of one cluster and a member of another is greater than a specified threshold the clusters are not merged. Using this approach, it is possible that the shape of the cluster in solution space is elongated, meaning that the maximum distance between cluster members can be large. Nevertheless, this approach has produced good results. In our experiments we have been using a distance threshold of 0.3 units of the hypercube diagonal, and discarding all clusters of size less than 5. For most of our experimental subjects we have found that in a final set of genetic algorithm solutions this usually produces one large cluster with 10 or more solutions and another smaller cluster with 5 or more solutions.

After clustering is completed, a median phenotype is computed for each cluster by finding the median value for the expression of each gene in the cluster. This median phenotype is taken as a representative for the cluster. It is converted back into the corresponding genotype and stored together with the individual genotypes in the cluster in a cluster output file.

The Experimental Process
The experimental process that we have used proceeds as follows. The subject is given a subject id number that is encoded into an otherwise empty history file. For the experiment, the subject wears stereo glasses. The experiment begins when the subject runs the experiment software giving the history file as parameter. The subject is then presented with two layered test surfaces, each with a nominal texture pattern that is known to allow the subject to clearly see both surfaces and their shapes. The same test surfaces were used for all trials of the experiment. The top surface has a sinusoidal wave and a distinct semispherical "hill". The bottom surface is flat with hilly perturbations composed from Gabor functions (Ware, 2000). The display is presented in stereo and is also rocked about an imaginary vertical axis perceptually positioned at the screen surface. Thus the subject has both stereo and motion cues available. When the subjects are

satisfied that they understand the two surfaces and how they are positioned relative to each other they press the space key and the genetic algorithm proceeds. When each phenotype (texture set) is presented, the user must respond by pressing the digit key on the keyboard corresponding with their evaluation of their ability to see the shapes of both surfaces and the relationships between the surfaces. When all the members of one generation of genotypes has been evaluated, the user is asked to press the space key to continue or 'q' to quit. If the user elects to continue, the genetic algorithm proceeds through another generation. If the user elects to quit, they may restart again at any time from where they left off by running the experiment software giving the assigned name of the history file as parameter.

When the subject has completed a session, clustering is run on the history file and the results written to a cluster output file. This file is then used as input to a viewer program, which allows the experimenter to view the phenotypes in each cluster as well as the median phenotype. At this point in the development of our methodology, analysis of results is done simply by visual evaluation. However, since all of the genotypes are available in each cluster together with their evaluations and since it is easy to generate the corresponding phenotype it is possible to do a more quantitative analysis of results, including doing image processing on the phenotype displays.

Results
We have obtained results from 15 separate trials of the genetic algorithm. These were obtained from 5 subjects, with each subject completing 3 trials. Clustering was run on all history files with the threshold parameters described above.

Figure 5 shows the phenotypes of the medians of two clusters from the same trial with the same subject. The left image of each pair shows the upper surface and the right image shows the lower surface. The two solutions are distinctly different although they share some similarities. The textures shown on the upper surfaces are a bit misleading, since in this printed form it is impossible to tell how transparent the different parts of the texture are. Nevertheless, it is possible to see many important characteristics of the textures.

Figure 6, on the other hand, shows distinctly different results from 4 different trials, each with a different subject, demonstrating the diversity of solutions that can be obtained.

For all of the subjects the process was successful in producing good solutions to the problem of displaying overlapping surfaces. The randomly generated texture pairs at the start of the process were almost all very poor solutions, making it almost impossible to see one surface over another. Initially improvement was rapid, but after about 20 generations the evolutionary process yielded a set of solutions that were improving only slowly, but where most of the texture pairs were at least reasonable solutions and where good solutions were common. We found that this took approximately two hours per subject.

It is our observation that the kind and degree of transparency of the top surface is a critical variable. Most of the highly rated solutions have top surface textures consisting of opaque, or nearly opaque, texture elements separated by completely transparent regions. Of the 15 cluster medians we have examined, 12 have these large fully transparent areas. For more than half of these we estimate that 70% of the top texture area is fully transparent, and for the rest the transparent region is at least 50%. There are only three solutions that do not have fully transparent areas. Of these, two have a kind of milky translucent surface scattered with small texture elements.

Many of the textures have components that differ greatly in size between the foreground and the background. An example would be a solution with large foreground texture elements overlaying small background texture elements. Where there are distinct stripes in both the foreground and background, or elongated oriented elements, these are never parallel, but always at least 45 degrees apart in orientation.

We are unable to make any generalizations about the colors of the solutions. In fact we suspect that color is not a particularly important factor. It may therefore be a useful "free

parameter" that in a visualization problem can be used to display data attributes other than the shape of the surfaces.

Conclusion

In summary, we have begun to attack the problem of discovering a coherent approach to determining how to produce texture sets for texturing layered surfaces so that both surfaces are still maximally visible to human subjects. The complexity of attempting to do this made it impractical to use standard psychophysical experimental methodologies and necessitated the development of a new approach to developing solutions to complex visualization problems. Although the results of the study of texture sets are still preliminary, we feel that the methodology for attacking the problem is of real interest to the field and can have wide application.

The process we have developed appears to produce good solutions to a complex visualization problem. However, it does not automatically produce solutions that are necessarily the simplest and most elegant. This is unsurprising, given the nature of the process and the selection criterion we used, but we would like to develop a process for abstracting the key elements of an effective design into simpler, cleaner solution.

We have a number of other ideas we would like to try in order to refine the approach. The first would be to look for cross-subject clusters of solutions. As yet we have looked at only single subject clusters. Another idea is to put in controls to allow the subject to do a local exploration of the solution space about a particular solution. The results of this exploration could be fed back to the genotype. This approach would capitalize on the genetic algorithm's strength – rapidly exploring a high dimensional space – while also capitalizing on the subject's capacity to do visual "gradient following" to locally optimize a solution. Finally, we want to look at more quantitative methods for analyzing the characteristics of successful solutions. This would include direct analysis of the numeric gene expressions in the phenotype as well as image analysis on the display of the phenotype to reveal such global characteristics as average opacity, and spatial frequency spectra.
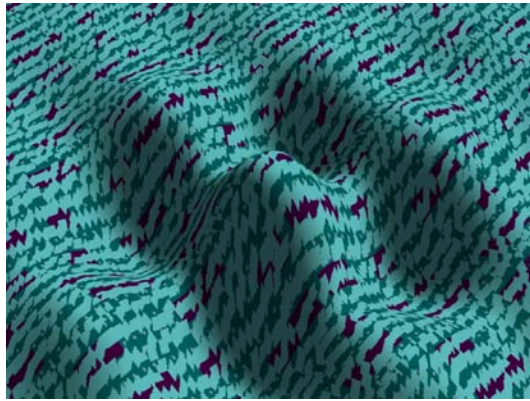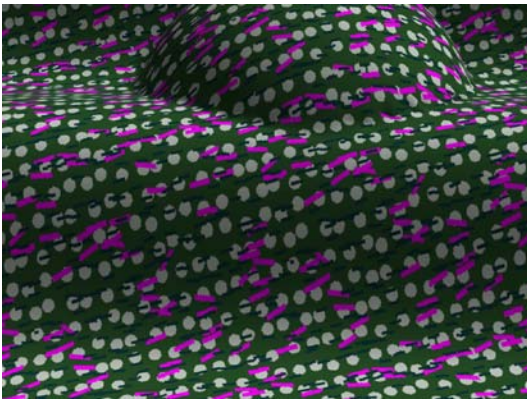
upper surface   lower surface
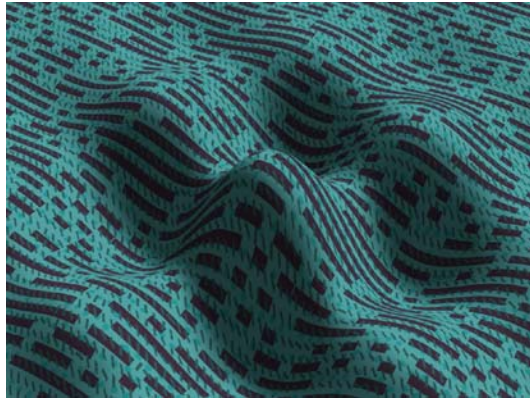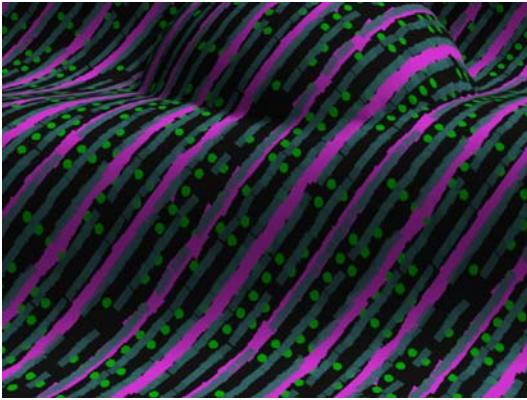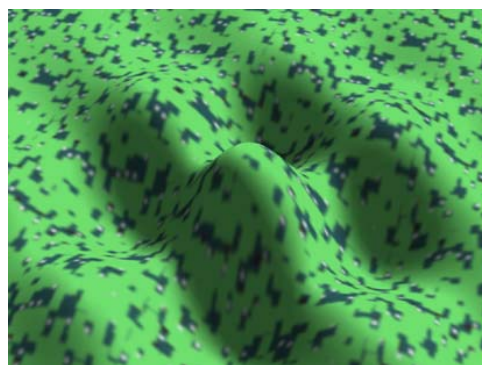

subject 1, trial 1, cluster 1
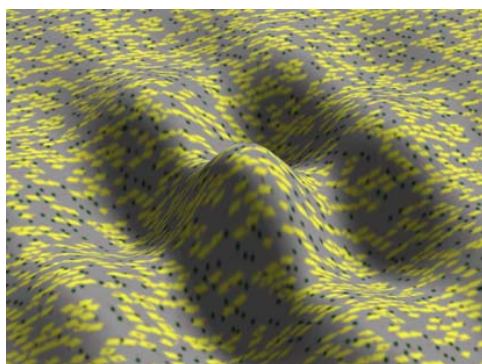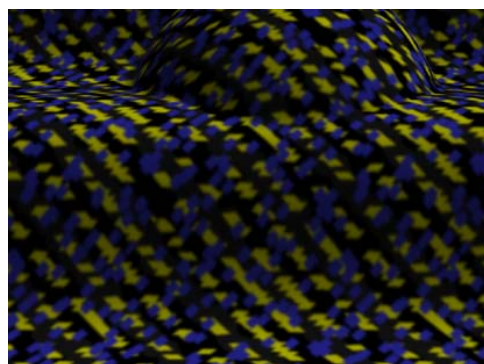

subject 1, trial 1, cluster 2

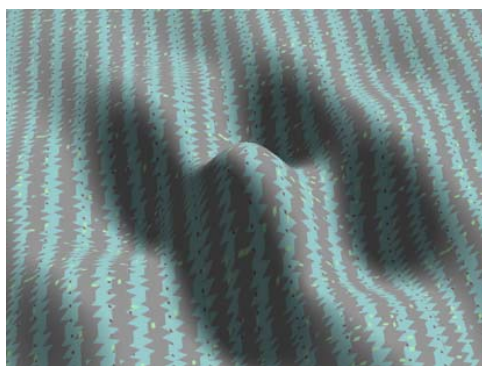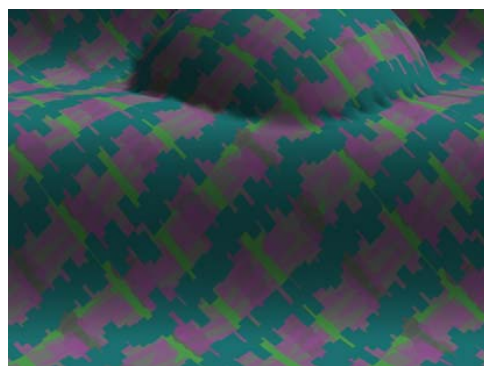Figure 5. Two Clusters From the Same Trial

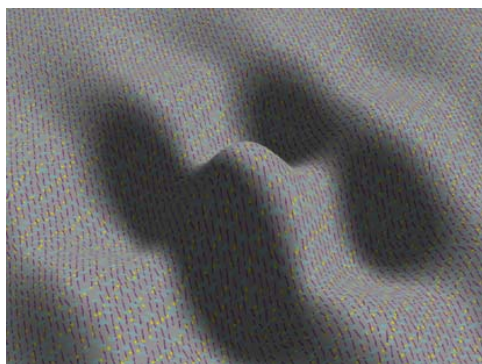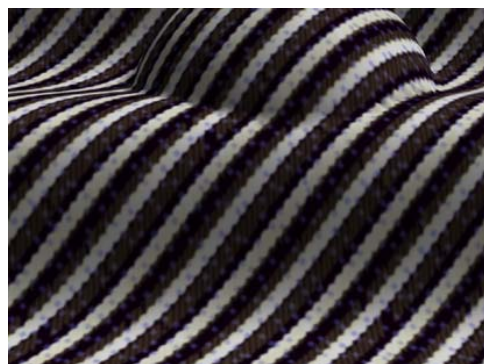upper surface   lower surface

subject 2, trial 3, cluster 1

subject 3, trial 2, cluster 1

subject 4, trial 3, cluster 1

subject 5, trial 1, cluster 1

Figure 6. A Texture Sampling

References

Cumming, B.G. Johnston E.B. and Parker, A.J. (1993) Effects of Different Texture Cues on Curved Surfaces Viewed Stereoscopically, Vision Research, **33**(5/6): 827-838.

Dawkins, R. (1986) The Blind Watchmaker, Harlow Logman.

Fogel, D. B. (1999) Evolutionary Computation: Toward a New Philosophy of Machine Intelligence. IEEE Press, Piscataway, NJ, 2nd edition.

Gibson, J.J. (1986) The ecological approach to visual perception. Lawrence Erlbaum Associates, Hillsdale, NJ.

Interrante, V., Fuchs, H., and Pizer, S.M. (1997) Conveying shape of smoothly curving transparent surfaces via texture. IEEE Trans. On Visualization and Computer Graphics 3(2) 98-117.

Norman, J.F., Todd, J.T. and Phillips, F. (1995) The perception of surface orientation from multiple sources of optical information. Perception and Psychophysics, 57(5) 629-636.

Marks, J., Andalman, B., Beardsley, P.A., Freeman, W., Gibson, S., Hodgins, J., Kang, T. (1997) Design galleries: a general approach to setting parameters for computer graphics and animation, Computer Graphics Proceedings, 389-400.

Pitz, H.C. (1957) Ink drawing techniques. Watson-Guptill Publications, New York.

Ramachandran, V. (1988) Perceived shape from shading. Scientific American, August, 76-780.

Rogers, B. and Cagnello, R. (1989) Disparity crvature and the perception of three-dimensional surfaces. Nature 339, May, 137-139.

Saito, T. and Takahashi, T. (1990) Comprehensible rendering of 3D shapes, Computer Graphics 24, 197-206.

Sims, K. (1991) Artificial Evolution for Computer Graphics, Computer Graphics 25, 319-328.

Todd, J.T and Akerstrom, R. (1987) Perception of Three-Dimensional Form from Patterns of Optical Texture", Journal of Experimental Psychology: Human Perception and Performance, 13(2): 242-255,

Watanabe T. and Cavanaugh, P. (1966) Texture laciness: The texture equivalent of transparency. Perception 25, 293-303.

Ware, C. (2000) Information Visualization Perception for Design, Morgan Kauffman, San Francisco,

Winkelbach G., Salesin D.H. (1994): Computer-generated pen-and-ink illustrations, Computer Graphics 28 (4), pp. 91-100