Joint EUROGRAPHICS - IEEE TCVG Symposium on Visualization (2002)
D. Ebert, P. Brunet, I. Navazo (Editors)

Adding a scalar value to texture-based vector field representations by local contrast analysis

A. Sanna, C. Zunino, B. Montrucchio and P. Montuschi

Dipartimento di Automatica e Informatica, Politecnico di Torino, corso Duca degli Abruzzi 24, I-10129 Torino (Italy)

Abstract

Several algorithms can effectively represent vector fields by texture-based representations, visualizing at most all information on the field: direction, orientation, and local magnitude. An open problem still remains the mapping on textures of adjunctive information such as temperature, pressure, and so on, without using colors. This article addresses this issue by proposing a technique to add a scalar value denoting streamlines by means of different levels of contrast. Both streamline starting tones and the range of tones depend on the scalar value to be mapped; in this way, areas visualized by different contrast levels are represented. Two examples show the effectiveness of the proposed technique.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Viewing Algorithms

1. Introduction

Visualizing vector fields is one of the most important tasks in scientific visualization. Several techniques are known in the literature: vector plot, particle tracing, stream surfaces, volume rendering, and so on. One of the main drawbacks of these methodologies is to often provide a rather coarse spatial resolution; in this way, small vector field features could be lost in the visualization.

Texture-based methods overcome this problem, only for 2D-data sets, by providing flow representations up to the pixel resolution. Curves having tangent vectors coincident to the vector field (streamlines) are computed in order to produce textured representations of the flow. Direction, orientation, and local magnitude can be denoted in very effective and efficient ways.

On the other hand, the multivariate visualization can be a problem for this kind of techniques. Several applications require to map a set of scalar values besides vector field structure (for instance, pressure, temperature, vorticity, ...), but it is very difficult as pixel tones are computed to denote direction, orientation, and magnitude of the flow.

Colors are often used to add an "adjunctive coordinate" but this could be not sufficient. This paper addresses this issue by proposing a new, innovative, and efficient strategy based on the local contrast analysis. The sensitivity of the human eye to different contrast levels [1] is used to add a scalar value in the textures. Low contrast levels will denote low values of scalar, while highly contrasted areas will represent high scalar values. Practical examples show how this approach can improve the user capability to investigate and detect vector field features, thus leaving colors to map further information.

Section 2 reviews main texture-based techniques and Section 3 presents goals and the basic idea behind the proposed work. The algorithm is described in the details in Section 4 and two examples show the effectiveness of using local contrast to add a scalar value to textures in Section 5.

2. Background

Texture-based methods can improve spatial resolution up to the pixel limit (dense textures). One of the first algorithm, called spot noise, was proposed by Van Wijk [2]; spot noise convolves a random texture along a straight segment whose orientation is parallel to the flow direction. This method was then extended by bending spot noise, filtering the image to cut low frequency components, and using graphics hardware methods [3]. Cabral and Leedom [4] presented the Line Integral Convolution (LIC) algorithm which soon became one of the most popular techniques to represent vector fields by textures. LIC locally filters a white noise input texture a-

long a streamline. Given a steady vector field defined by a map $v : \Re^2 \rightarrow \Re^2, x \longmapsto v(x)$, its directional structure can be shown by the integral curves, where an integral curve is a path $\sigma(u)$ having tangent vectors coincident to the vector field (that is $\frac{d}{du}\sigma(u) = v(\sigma(u))$). By a re-parameterization of $\sigma(u)$ in terms of the arc-length s, it is possible to compute the line integral convolution (LIC) for a pixel located at $x_0 = \sigma(s_0)$:

$$I(x_0) = \int_{s_0-L}^{s_0+L} k(s-s_0)T(\sigma(s))ds. \qquad (1)$$

where $T(x)$ is an input white noise texture, $k(s)$ is the filter kernel (normalized to unity), and the filter length is $2L$.

Forssell [5] extended the LIC to curvilinear grid surfaces, by doing calculations in computational space on a regular cartesian grid, and then displaying results on curvilinear grids in physical space. Stalling and Hege [6] reduced the computational time of LIC by more than ten times proposing fastLIC. The LIC value computed for one pixel can be re-used, with small modifications, from its neighbor pixels. In this way, the computation is streamline oriented and not pixel oriented.

A fast implementation of LIC-like algorithms can be also found in [7] and [8]; in [7] every field line is drawn using a different gray value (the same value along the whole streamline). However, because of the constant gray value, this method cannot show orientation and magnitude of the field and it is not suitable for animation. On the other hand, in [8] the TOSL (Thick Oriented Streamline) algorithm was proposed, which is able to depict all vector field characteristics and computational times are up to three times faster than fastLIC. TOSL may suffer of the macro-structure problem since streamline starting tones are randomly assigned and it is possible that neighboring pixels have very similar gray tones thus reducing the capability to distinguish streamlines.

Visualization of dense and oriented flow fields is also performed by Jobard and Lefer [9]. The minimization of the number of the streamlines is performed by an evenly-spacing algorithm which is able to produce a good quality image. However, the use of this algorithm slows down the speed in comparison to fastLIC.

Bi-dimensional LIC images can be animated to show the orientation of the field besides the simple direction. This can be done by changing shape and location of the filter kernel k over time. To avoid the need for animation, Wegenkittl et al. [10] introduced OLIC (Oriented Line Integral Convolution) and then FROLIC [11] (Fast Rendering OLIC). OLIC simulates the use of drops of ink smeared to the underlying vector field. The algorithm can be made faster by positioning small and overlapping disks (FROLIC) in order to simulate the convolution. The length of the pixel traces shows vector orientation and local magnitude of the field. However, OLIC and FROLIC use sparse textures, and therefore, small details of the field may be lost in the visualization. The vector field visualization can be achieved also using furlike textures [12] and the results are similar to FROLIC.

An extended LIC algorithm, called UFLIC (Unsteady Flow LIC), for visualizing vector data in unsteady flow fields has been presented in [13]. The convolution algorithm consists of a time-accurate value depositing scheme and a successive feed-forward method. The value depositing scheme accurately models the flow advection, and the successive feed-forward method maintains the coherence between animation frames.

Although several algorithms can map information of direction, orientation, and local magnitude, adding scalar values such as temperature or pressure can be a problem. Classical methods use colors to denote special vector field characteristics. For instance, in [14] and [15] the users, using LIC, can introduce colored dyes into the vector field to highlight local flow features. The inserted dyes propagate through the flow field highlighting vector field characteristics.

A completely different approach is used in [16] and [17] where texture algorithms are merged both to the bump mapping technique and shadow casting. Bumps and depressions are used in [16] where a bump texture is built according to the scalar value to be mapped and, by a post-processing phase, the bump mapping algorithm is applied on the texture achieved by a classical texture-based technique such as LIC. A depth coordinate was used in [16] in order to map a scalar value. A second scalar can be mapped also considering the azimuth coordinate to cast shadows over bumps and depressions [17]. This method can lead to effective visualizations as long as the second scalar value changes slowly and the first scalar is strongly related to the vector field structure.

3. Goals and Basic Idea

Texture-based methods can visualize vector field characteristics up to the pixel resolution. In a broad spectrum of applications, for instance in CFD (Computational Fluid Dynamics), other scalar values such as temperature, pressure, and so on, should be mapped on the textures. A few work has been done to tackle this important issue and this paper aims to propose a new approach to address multivariate visualization.

It has been proved [1] that the level of contrast strongly affects the capability of human eye/brain system to perceive details. In particular, details are less noticed when displayed in low contrast areas.

The idea behind this work is to use the local contrast level to add information in a texture (that is, a scalar value). Areas denoted by larger scalar values will be characterized by higher contrast levels, and streamline tones will be assigned in order to allow users to easier detect vector field structure. On the other hand, zones where the scalar value to be mapped is low will be represented by means of lowly contrasted streamlines.

The level of contrast has to be sufficiently high to permit a clear identification of streamlines in low contrast areas and maximum for zones associated to high scalar values in order to enhance the user capability to perceive the vector field features.

4. The Algorithm

A C-like pseudo code of the algorithm is shown in Figures 1 and 2; in particular, the procedure allows to determine gray tones assigned to a streamline. The algorithm is divided in two parts. The first phase (Figure 1) calculates a starting tone attempting to maximize the local contrast [18], while the second phase (Figure 2) adapts the starting tone according to values of the scalar encountered along the streamline under analysis.

In the first phase all pixels along the streamline are analyzed and, for each one, an area around the pixel itself is considered (`x_start x_stop y_start y_stop`). This step is performed to collect statistics about pixels of previously computed streamlines which are neighbor to the streamline under analysis.

The variable `n_less` is used to store the number of pixels having gray tones in the range `]0,127]` within a square area with center in `(x,y)` and side 2 pixels (see [18]); in the same way, `n_greater` stores the number of values in the range `[128,255]`. The value `0` denotes pixels not yet computed. On the other hand, `c_less` and `c_greater` are used to store the average value of gray tones in the range `]0,127]` and `[128,255]`, respectively.

The search area is analyzed by two nested cycles and then the average values are computed for gray tones both in the range `]0,127]` and `[128,255]` (if `n_less` and `n_greater` are not null). This phase is repeated for every pixel along the streamline under analysis. In order to enhance the local contrast `128` is added to the average value in the range `]0,127]` while is subtracted for the range `[128,255]`.

If no pixel already computed has been found in the searching area, a random gray tone is assigned to the starting pixel of the streamline, otherwise, an average value is set. Finally, the phase 2 of the algorithm is called. The second phase begins by computing the maximum scalar value along the streamline. This value is necessary to set both a `range` of values allowed for the streamline pixels and a correction parameter (`distance`). The parameter `distance` is added to the starting tone computed in the first step of the algorithm. Scalar values are normalized loading vector field files between 0 and 1, and therefore, larger values of `scalar_value` will produce lower `distance` values, while a lower `scalar_value` will lead the starting tone nearer to 127 (that is, the center in the range of permitted values). Moreover, streamline pixels can assume values only within a range (`tone_begin` and `tone_end`) depending both on `scalar_value` and on the parameter `bandwidth` set by the user. A large value of scalar will lead to use at most all the range denoted by `bandwidth`, on the other hand, a low scalar value forces pixels to be clustered around 127. In this way, the local contrast strongly depends on the scalar value. If streamlines move in field areas where the scalar value is high, the range of permitted gray tones will be large and the local contrast will be high, but if low scalar values are encountered along streamlines, pixel tones are forced to be close to 127 into a small range, thus producing poorly contrasted streamlines. Moreover, the user can control the `bandwidth` parameter which can be used as a minimum filtering band (examples are shown in Section 5), that is, the range `tone_end - tone_begin` can be never smaller than `bandwidth`. Finally, gray tones along a streamline change according to the local velocity magnitude. The constant value `cost_tone` is proportionally added to the previous pixel tone and if the value is greater than `tone_end` (that is, it is out of the permitted range) the gray tone restarts from `tone_begin`.

```
phase1(int d,int x,int y)
{
 c_less = c_greater = 0;
 n_less = n_greater = 0;
 x_start = x - 1; x_stop = x + 1;
 y_start = y - 1; y_stop = y + 1;

 for each pixel of the streamline
  for(i=x_start;i<x_stop;i++)
   for(j=y_start;jy_stop;j++) {
     if(pixel(i,j) in ]0,127])
     {
      n_less++;c_less+=pixel(i,j);
     }
     if(pixel(i,j) in [127,255])
     {
      n_greater++;
      c_greater += pixel(i,j);
     }                          }
 if(n_less != 0)
 c_less = c_less/n_less + 128;
 if(n_greater != 0)
 c_greater=c_greater/n_greater-128;
 if(not found pixels)
   pixel = random_value;
 else
   pixel = ave(c_less,c_greater);

   phase2(pixel,bandwidth);
}
```

Figure 1: *The first phase of the algorithm.*

5. Examples

In the first example, we visualize the flow field corresponding to the natural development of a spatially evolving two-

```
phase2(int tone, int bandwidth)
{
   int scalar_value;
   int distance, range;
   int tone_begin, tone_end;

   scalar_value = 0;
   for each pixel along the streamline
      if (local_scalar > scalar_value)
         scalar_value = local_scalar;

   distance = (127 - tone)*(1-scalar_value);
   tone += distance;
   range = (bandwidth / 2 + ((255 - bandwidth)/2) * scalar_value);
   tone_begin = 127 - range;
   tone_end = 127 + range;

   for each pixel along streamline
   {
      tone += ceil((cost_tone*(cost_tone - modulo)/max_modulo));
      if(tone > tone_end)
         tone = tone_begin;
   }

}
```

Figure 2: *The second phase of the algorithm.*

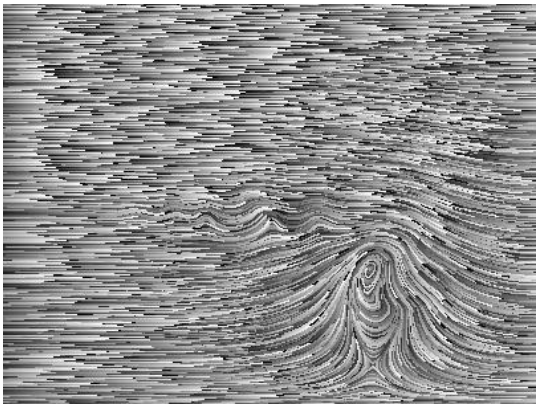

Figure 3: *Two-dimensional laminar mixing layer visualized by TOSL.*

dimensional laminar mixing layer. A mixing layer originates in the merge of two parallel streams, each with a uniform velocity U_1 and U_2, both assumed in the same direction (the upper stream U_1 is faster than U_2). A vorticity develops in the streamwise direction. Figures 3 and 4 show the visualization respectively obtained by TOSL and LIC, without considering the scalar value to be mapped on the texture, while Figure 5 represents a gray tone encoding of the scalar value to be mapped on the texture; dark tones correspond to high values, on the other hand, light tones denote low scalar values.

Figure 4: *Two-dimensional laminar mixing layer visualized by LIC.*

Figure 6 shows the result obtained after the first phase of the algorithm attempting to enhance the local contrast in order to better denote streamlines. Figure 7 shows the situation obtained by setting `bandwidth = 100`. The scalar to be mapped is the local vorticity and it can be noticed as the vortex and the area characterized by higher values are clearly denoted. On the other hand, the two streams are visualized by lower contrast levels and although information concern-

Figure 5: *Gray tone representation of the scalar value.*

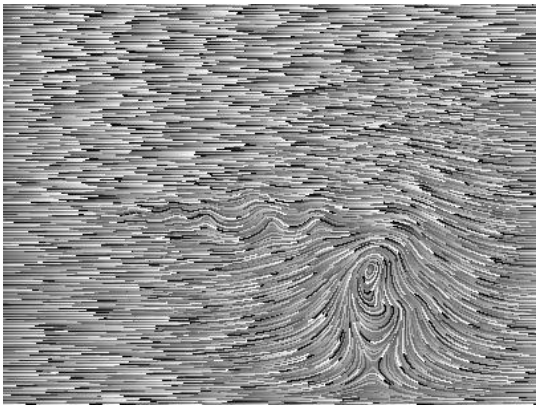

Figure 6: *Two-dimensional laminar mixing layer visualized after the first phase of the algorithm maximizing local contrast.*

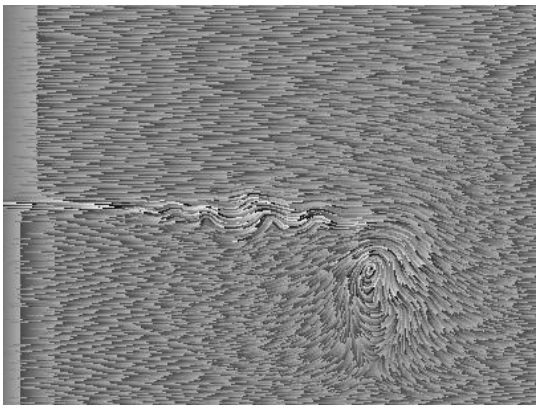

Figure 7: *Two-dimensional laminar mixing layer -* `bandwidth = 100`.

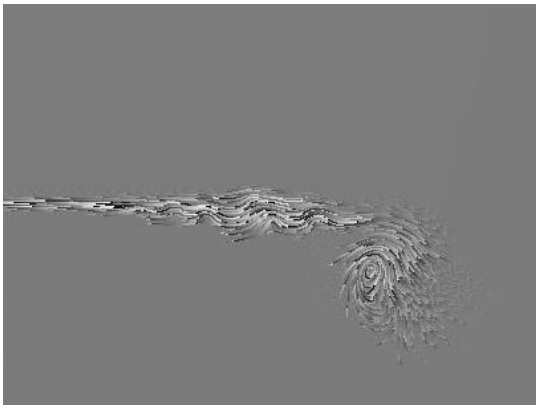

Figure 8: *Two-dimensional laminar mixing layer -* `bandwidth = 0`.

ing the flow field (direction, orientation, local magnitude) can still be detected, user can easier identify interesting areas. The impact of the choice of `bandwidth` can be noticed in Figure 8 where the mixing layer example is visualized by setting `bandwidth` equal to 0. A null contrast is used to depict streamlines where the scalar value is minimum and only the vorticity in the developing is represented (the two parallel streams disappear). The second example represents two vortices. Figures 9, 10, and 11 depict the visualization setting `bandwidth` equal to 140, 50, and 0, respectively. Figure 12 denotes a gray tone encoding of the scalar value in the same way as Figure 5. It can be clearly noticed the impact of the proposed methodology; the left vortex involves lower scalar values and streamlines representing it are visualized by lower contrast levels; moreover, the choice of `bandwidth` affects the representations of the left vortex, and in particular of its center.

6. Conclusion and Remarks

This paper presents a new and effective technique to tackle the multivariate visualization problem using texture-based representations. A local contrast analysis phase allows to assign pixel gray tones along streamlines according to the local value of the scalar to be mapped. Neighboring streamlines will be highly contrasted if they move in areas where the scalar value is high or, otherwise, lowly enhanced. The user has to be aware of this encoding scheme in order to correctly interpret underlying data. The examples show how, although in poorly contrasted areas information on the vector field is clearly visible, highly contrasted streamlines allow to immediately detect zones of particular interest. In order to be fair, it has to be outlined how the proposed method is able to provide a qualitative evaluation of the scalar value mapped on the texture, while a quantitative information could be better obtained by using colors.

Computational overhead due to the contrast analysis

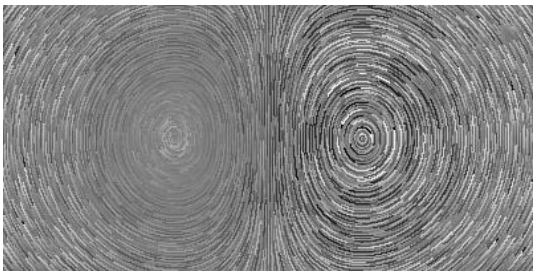

Figure 9: *Double vortex -* `bandwidth = 140`.

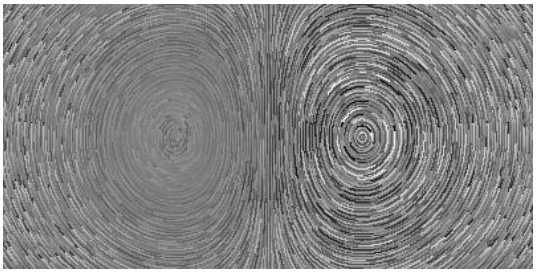

Figure 10: *Double vortex -* `bandwidth = 50`.

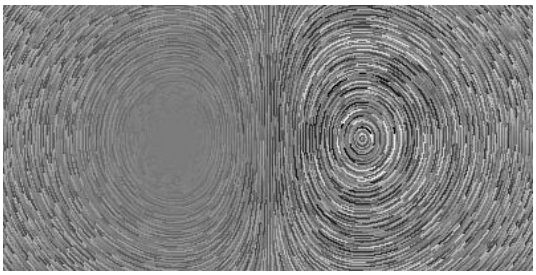

Figure 11: *Double vortex -* `bandwidth = 0`.

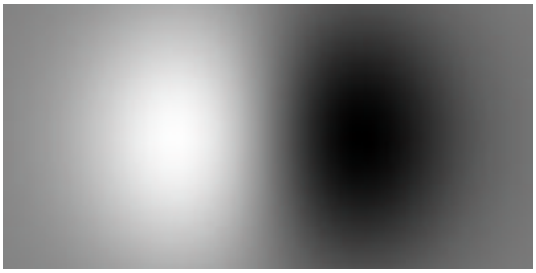

Figure 12: *Gray tone representation of the scalar value.*

phase is negligible in terms of absolute values. In fact, computational time for the first example (resolution of 402x302 pixels) without the proposed methodology is of 0.266*s* versus 0.328*s* necessary using local contrast, while computational time for the second example (resolution of 400x200 pixels) changes from 0.203*s* to 0.250*s*; all times refer to a 800 MHz Pentium III.

It has been shown in Section 5 the impact of the parameter `bandwidth`; users can use `bandwidth` to investigate in-depth flow characteristics by varying the range from 0 to 255, but it could be difficult immediately identifying a value of `bandwidth` suitable for clearly detecting vector field features.

Future work will be aimed to analytically evaluate important issues such as the number of levels of contrasts that can be clearly detected by human perception and the possibility of combining the proposed methodology with other techniques for mapping adjunctive scalar values.

References

1. C. Ware. *Information Visualization: Perception for Design*. Morgan Kaufmann Publishers, 2000. 1, 2
2. J.J. Van Wijk. Spot Noise-Texture Synthesis for Data Visualization. *ACM Computer Graphics (Proc. of SIGGRAPH '91)*, **25**(4):309–318, July 1991. 1
3. W.C. De Leeuw and J.J. Van Wijk. Enhanced Spot Noise for Vector Field Visualization. *Proceedings of IEEE Visualization '95*,:233–239, 1995. 1
4. B. Cabral and L. Leedom. Imaging Vector Fields Using Line Integral Convolution. *ACM Computer Graphics (Proc. of SIGGRAPH '93)*,:263–270, 1993. 1
5. L.K. Forssell. Visualizing Flow over Curvilinear Grid Surfaces Using Line Integral Convolution. *Proceedings of IEEE Visualization '94*,:240–247, 1994. 2
6. D. Stalling and H.C. Hege. Fast and Resolution Independent Line Integral Convolution. *ACM Computer Graphics (Proc. of SIGGRAPH '95)*,:249-256, 1995. 2
7. C.P. Risquet. Visualizing 2D Flows: Integrate and Draw. *Proceedings of 9th Eurographics Workshop on Visualization in Scientific Computing*,:132–142 1998. 2, 2
8. B. Montrucchio, P. Montuschi, A. Sanna and A. Sparavigna. Visualizing Vector Fields: the Thick Oriented Stream-Line Algorithm (TOSL). *Computers & Graphics*, **25**(5):847–855, 2001. 2, 2
9. B. Jobard and W. Lefer. Creating Evenly-Spaced Streamlines of Arbitrary Density. *Proceedings of the eight Eurographics workshop on visualization in scientific computing*,:57–66, 1997. 2
10. R. Wegenkittl, E. Gröller and W. Purgathofer. Animating Flowfields: Rendering of Oriented Line Integral Convolution. *Proceedings of Computer Animation '97*,:15–21, 1997. 2
11. R. Wegenkittl and E. Gröller. Fast Oriented Line Integral Convolution for Vector Field Visualization via the

Internet. *Proceedings of IEEE Visualization '97*,:309–316, 1997. 2

12. L. Khouas, C. Odet and D. Friboulet. 2D Vector Field Visualization Using Furlike Texture. *Proceedings of Data Visualization '99*,:35–44, 1999. 2
13. H.W. Shen and D.L. Kao. UFLIC: a line integral convolution algorithm for visualizing unsteady flows. *Proceedings of IEEE Visualization '97*,:317–322, 1997. 2
14. H.W. Shen, C.R. Johnson and K.L. Ma. Visualizing Vector Fields Using Line Integral Convolution and Dye Advection. *Symposium on Volume Visualization '96*,:63–70 1996. 2
15. H.W. Shen. Using Line Integral Convolution to Visualize Dense Vector Fields. *Computer in Physics*, **11**(5):474–478, 1997. 2
16. A. Sanna and B. Montrucchio. Adding a Scalar Value to 2D Vector Field Visualization: the BLIC (Bumped LIC). *Eurographics'2000 Short Presentations Proceedings*,:119–124, 2000. 2, 2, 2
17. A. Sanna, B. Montrucchio and P. Montuschi. B^2LIC: an Algorithm for Mapping Two Scalar Values on Texture-Based Representations of Vector Fields. *Proceedings of WSCG '01*,:I138–I145, 2001. 2, 2
18. A. Sanna, B. Montrucchio, C. Zunino and P. Montuschi. Enhanced vector field visualization by local contrast analysis. *Proceedings of WSCG '02*,:II389–II396, 2002. 3, 3