B^2LIC: an algorithm for mapping two scalar values on texture-based representations of vector fields

A. Sanna B. Montrucchio P. Montuschi

Dipartimento di Automatica e Informatica,
Politecnico di Torino
corso Duca degli Abruzzi 24,
10129 Torino (Italy)
email: {sanna,montru,montuschi}@polito.it

ABSTRACT

Visualization of vector data produced from application areas such as computational fluid dynamics (CFD), environmental sciences, and material engineering is a challenging task. Texture-based methods reveal to be effective, versatile, and suitable for a large spectrum of applications since they allow to obtain high resolution output textures where direction, orientation, and magnitude of the flow can be displayed. In this paper we present a new method called B^2LIC, which allows both to characterize and visualize interesting structures in the flow and to map two additional scalar values in output textures by bumps, depressions, and shadows, leaving colors for further information mapping. B^2LIC is the natural and direct evolution and the improvement of the BLIC (Bumped LIC) algorithm, which is able to map just one scalar value by the bump mapping technique. Some examples show how the proposed method can effectively allow to map two additional scalar values such as: temperature, vorticity, pressure, and so on, adding new and additional representation capabilities to dense texture-based visualization methodologies.

Keywords: Scientific visualization, multivariate visualization techniques, texture-based methods, LIC.

1 INTRODUCTION

Vector field visualization is fundamental for a large spectrum of disciplines where the data obtained by experimental observations and theoretical elaboration need to be graphically displayed.

Texture-based methods were introduced to solve the problem of traditional approaches that use too many pixels in order to visualize a vector (for instance arrow plot). Indeed, while a scalar can be visualized by varying the color of a pixel, the use of a segment (or an arrow) to visualize a vector may need much more pixels. In addition this can lead to a loss of resolution and of level of detail, thus possibly preventing, in some cases, an effective comprehension of the vector field under analysis. The elegance and the effectiveness of texture-based techniques were the starting point for new and interest-

ing research in the visualization of vector fields. Although several works have been published in the literature (among all [Wijk91] [Cabra93] [Stall95] [Wegen97a], and [Wegen97b]) some issues must be still tackled. In particular, research is involved in improving the quality of the output textures, in order to allow a better understanding of the vector field under analysis. Moreover, in a large spectrum of applications it is required to map in the output texture more information than direction, orientation and magnitude of the flow; for instance, CFD applications may require to map several scalar values like: temperature, pressure, vorticity, and so on.
This last problem can be partially tackled using colors by dye advection [Shen96], but multivariate visualization is still an open problem. The proposed technique improves and extends the BLIC algorithm presented in [Sanna00a]; BLIC employs the bump mapping technique in order to add a scalar value to a texture, generated by a traditional texture-based visualization method such as LIC. The scalar parameter is represented on the resulting texture as bumps and depressions, according to the value to be mapped. The B^{2}LIC algorithm tackles the limitation of BLIC to be able to map just one scalar value using the second free parameter present in the bump mapping algorithm: the azimuth, that is the position of the light. By varying the position of the light that produces the shadows, we can change the appearance of bumps and depressions, thus showing the second scalar value.
The paper is organized as follows. Section 2 reviews the main texture-based techniques and the BLIC algorithm; Section 3 presents goals and basic idea of the new method, while the B^{2}LIC algorithm is described in details in Section 4. Finally, examples, results and remarks are shown in Section 5.

2 BACKGROUND

2.1 Texture based visualization techniques

Texture-based methods attempt to reproduce techniques known from experimental flow visualization such as the observation of randomly dispersed particles or dye injection patterns. The common goal is to produce high resolution images revealing the flow field characteristics: direction, orientation, magnitude, and so on.
Van Wijk [Wijk91] proposed to convolve a random (white noise) texture along a straight segment whose orientation is parallel to the direction of the flow. This method (called spot noise) was then extended by bending spot noise, filtering the image to cut low frequency components, and using graphics hardware methods, also on grids with irregular cell sizes (De Leeuw and Van Wijk [Delee95]).
Cabral and Leedom [Cabra93] introduced the Line Integral Convolution (LIC) algorithm, which locally filters a white noise input texture along a path of vectors tangent to the field, denoted as 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, or streamlines, 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 doing a re-parameterization of $\sigma(u)$ in terms of the arc-length s, we can calculate the line integral convolution (LIC) for a pixel located at $x_0 = \sigma(s_0)$:

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

where $T(x)$ is an input white noise texture, $k(s)$ is the filter kernel (normalized to unity), and the filter length is $2L$.
Stalling and Hege [Stall95] improved the speed of LIC (fastLIC) more than ten times, by observing that the LIC value computed for one pixel can be re-used,

with small modifications, by its neighbor pixels; in this way, the computation becomes streamline oriented and not pixel oriented as in the conventional LIC.
Zöckler et al. [Zockl97] showed a parallel implementation of fastLIC which is able to run in real-time on particular parallel architectures.
Wegenkittl et al. [Wegen97a] introduced OLIC (Oriented Line Integral Convolution) and then Wegenkittl and Gröller [Wegen97b] FROLIC (Fast Rendering OLIC). OLIC simulates the use of drops of ink smeared to the underlying vector field. The algorithm speed can be improved by positioning small and overlapping disks (FROLIC) in order to simulate the convolution. Besides direction, the length of the pixel traces shows vector orientation and local magnitude of the field. However, OLIC and FROLIC employ sparse textures, therefore, small details of the field may be lost in the visualization. Furlike textures are used in [Khous99], where the technique is based on a non stationary two dimensional AutoRegressive synthesis (2D AR). The texture generator allows local control of orientation and length of the synthesized texture (the orientation and length of filaments). This texture model is then used to represent 2D vector fields. Orientation, length, density and color attributes of furlike textures can be employed to visualize local orientation and magnitude of a 2D vector field.
A survey on texture-based methods can be found in [Sanna00b].

2.2 The BLIC algorithm

The proposed work extends and improves the algorithm called BLIC presented in [Sanna00a], which is here briefly reviewed for sake of completeness. BLIC can add a scalar value to a texture produced by a method such as LIC with the use of the bump mapping technique proposed by James Blinn [Blinn78] in 1978. Bump mapping simulates bumps and wrinkles in a surface without the need for geometric modifications to the model. The surface normal of a given surface is perturbed according to a bump map (also called altitude map), and the perturbed normal is used instead of the original one when shadows are computed using the Lambertian technique; this approach provides the appearance of bumps and depressions in the surface. Given a point on a surface parameterized by the function $\mathbf{O}(\mathbf{u},\mathbf{v})$, the normal $\mathbf{n}$ at that point is computed by:

$$\mathbf{n} = \mathbf{Q_u} \otimes \mathbf{Q_v} \qquad (2)$$

where $\mathbf{Q_u}$ and $\mathbf{Q_v}$ are the partial derivatives in the parameter directions u, and v, and $\otimes$ denotes the outer product. A new displaced point can be defined by adding some amount along the normal at that point:

$$\mathbf{Q}'(\mathbf{u},\mathbf{v}) = \mathbf{Q}(\mathbf{u},\mathbf{v}) + P(u,v)\frac{\mathbf{n}}{\mid \mathbf{n} \mid} \qquad (3)$$

where $P(u,v)$ is a perturbation function. The new perturbed normal can be computed as: $n' = Q'_u \otimes Q'_v$. Under the assumption of P small, $\mathbf{n}'$ can be reduced to:

$$\mathbf{n}' = \mathbf{n} + \frac{P_u(\mathbf{n} \otimes \mathbf{Q_v})}{\mid \mathbf{n} \mid} + \frac{P_v(\mathbf{Q_u} \otimes \mathbf{n})}{\mid \mathbf{n} \mid} \qquad (4)$$

The value of the new (perturbed) normal is based both on the original normal and on the perturbation function, which can be defined mathematically or by a two dimensional lookup table.
BLIC considers the scalar value to be mapped on the texture in order to produce the bump map. In a first step a dense texture is computed by a traditional technique such as LIC; then, the bump mapping process is performed by using as input values both the texture and the altitude map related to the scalar value; the algorithm is shown in Fig. 1, while Fig. 2, Fig. 3, and Fig. 4 show an example of BLIC application. Fig. 4 shows the result

and it can be noticed as some vortices appear bumped and other depressed in the surface according to the bump map; therefore, a better characterization of the vortical structures has been obtained and an addition scalar value has been mapped.

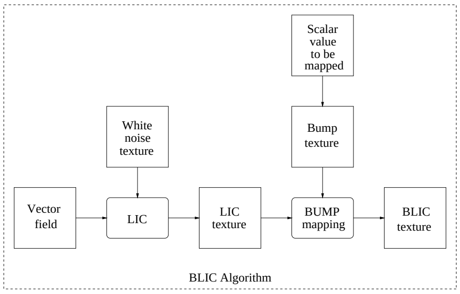

Figure 1: The BLIC algorithm.

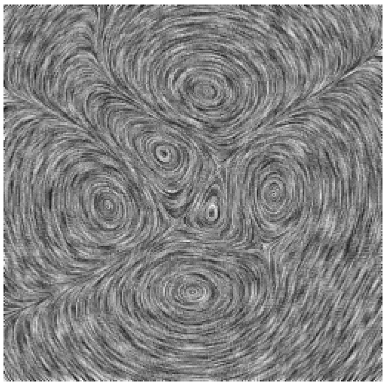

Figure 2: A texture produced by LIC.

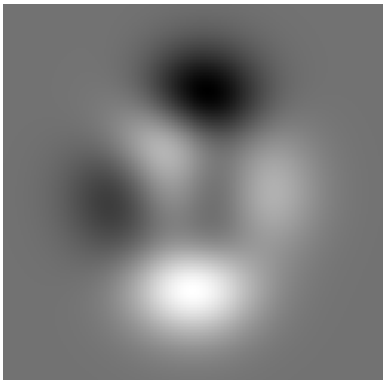

Figure 3: The bump map.

3 GOALS AND BASIC IDEAS

BLIC can show the vector field and one scalar value (without using colors). Now

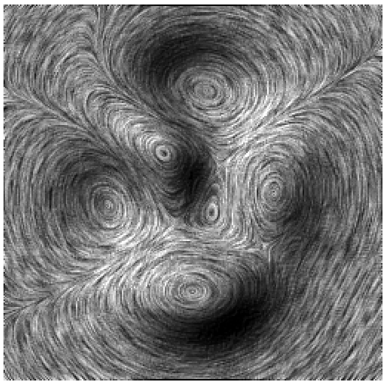

Figure 4: Result obtained by BLIC.

we study the possibility of showing another scalar value, always without colors (that can be used for a further scalar value).

In the bump mapping algorithm we can control many free parameters; the depth of bumps or depressions, and the position of the light, with elevation and azimuth see Fig. 5 (we can assume that the light is at infinite distance).

In the BLIC algorithm the depth has been used to map a scalar on the vector field, while elevation and azimuth are fixed.

The basic idea is now to use azimuth and/or elevation to map a further scalar value. Note that elevation and depth produce a similar effect on the final image, while an azimuth variation can strongly change the aspect of the result.

A simple example of the effects of changing azimuth in different parts of the image is reported in Fig. 6. Each quarter of the image has been computed for different azimuth values, which results in different shadows of the four bumps.

The main problem is how to map the scalar value on the azimuth coordinate; in fact, the additional scalar value visualized by means of the azimuth coordinate can be viewed only where bumps or depressions in the BLIC image there exist (bumps and depressions depends on the first scalar). This problem will be addressed in the next Section.

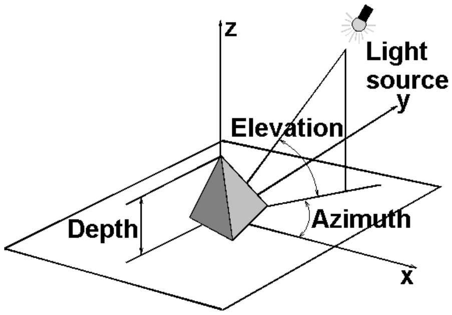

Figure 5: Bump map parameters.

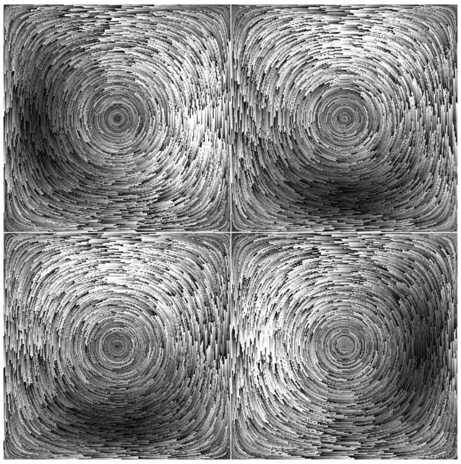

Figure 6: Different values of azimuth for each quarter of the example.

4 THE ALGORITHM

The schema of the B^2LIC algorithm is reported in Fig. 7. The problem addressed in Section 3 is solved by verifying that a bump or a depression exists where the second scalar is used to change the azimuth value. In particular, we use a simple recursive quadtree-like (see [Samet84] for a comprehensive survey on quadtrees) algorithm to verify the presence of a bump or a depression.

At first we sign bumps and depressions thresholding the image of the first scalar; this produces a B/W image where black pixels denotes bumps or depressions. The thresholded image is then divided in squares, each one with a number of black pixels able to identify a bump or a depression. In our implementation we have set the condition that each square must contain at least $(di/16)^2$, where di is the image dimension (without loss of generality we can suppose power of 2 square images). Both the threshold values and the number of black pixels have been experimentally tuned. The final image is then created by computing the mean azimuth for each square (using the second additional scalar value). Note that the azimuth can change only between 0 and π, since for larger values of the azimuth bumps become depressions and vice versa.

5 EXAMPLES, RESULTS AND REMARKS

In this Section the results of the proposed algorithm are shown by means of two examples.
The first example (the sinusoidal example) shows a *sinxsiny* vector field (see Fig. 8 for a texture based representation). The two scalar values added are in Fig. 9 and in Fig. 10. The BLIC result (with only the first scalar value added) is in Fig. 11, while the B^2LIC result is in Fig. 12. The white lines in Fig. 12 have been added only to show the domain of the different subdivisions. The impact of mapping the second scalar value can be noticed in the central part of the resulting image; comparing BLIC (Fig. 11) and B^2LIC (Fig. 12) it can be seen how shadows are differently casted on bumps and depressions. Arrows in Fig. 12 show areas where major differences between BLIC and B^2LIC can be noticed; looking at Fig. 10 it can be seen that areas pointed out by arrows in Fig. 12 correspond to the zones where the second scalar value varies (for the other parts of the image the second scalar is constant and hence no differences can be noticed between BLIC and B^2LIC).
The second example (the exponential example) is based on the vector field of Fig. 13, but inverting the scalar values in re-

```
// B2LIC
image_dimension di;
main() {
 // declare variables
 vector_field  v;
 scalar_1 s1;
 // s1 strongly correlated with v
 scalar_2 s2;
 // s2 loosely correlated with v

 // produce LIC-like image
 v_LIC = LIC-like applied on v;

 // sign bumps and depressions in black
 s1_t = threshold s1 between 25 and 230;

 // apply quadtree algorithm to divide
 // square, and retain square in memory
 divide(s1_t, di);

 // produce final image
 for each square previously stored
 {
  value = pixelize the whole square,
          but in s2;
  a = value * 180 / 255;
  apply BLIC on the square using v_LIC,
          s1 with azimuth a
 }
}

divide(image, n){
 if n == di / 8
 {
   store square; return;// max 3 steps
 }

 if for each of four sub squares
  (value = pixelize whole sub square)
    <(((n/2)^2-(di/16)^2)/((n/2)^2)*255)
  // at least (di/16)^2 black pixels
  // are required in each sub-square
 {
   for each of sub-squares of quadtree
     divide(square, n/2);
 }
 else store square; }
```

Figure 7: Pseudo-code for B^2LIC.

spect of the first example, that is the first scalar in Fig. 10, while the second is in Fig. 9. The B^2LIC result is in this case very similar to the BLIC one (reported in Fig. 14). In fact the second scalar value changes too quickly in comparison with the first one; the B^2LIC algorithm avoids errors in the visualization, but the information shown is very poor. The carrier frequency is too low.

Using B^2LIC direction, orientation and magnitude of the vector field are shown, plus two scalar values, and one further scalar value can be added using the H component of the HSV color coding method. As said before, not all scalar values can be added with good results. In fact the first scalar value should be strongly correlated with the vector field, while the second scalar can be loosely correlated, but it must change slowly. For the former (strong correlation) there are usually no problems, since in many cases vorticity or other scalar values are tightly coupled with the underlying vector field. And even for the latter it is often not difficult to find a slowly varying value. An eventually quickly varying scalar can be mapped using colors, that retain full resolution, even with B^2LIC, leaving the second position to another value.

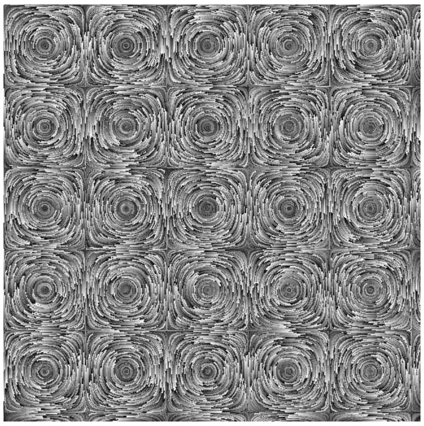

Figure 8: Vector field (texture-based method) of the sinusoidal example.

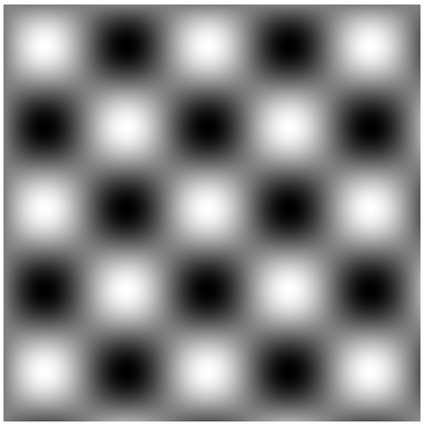

Figure 9: First scalar in the sinusoidal example.

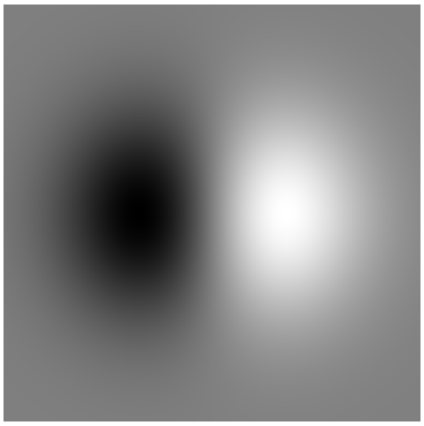

Figure 10: Second scalar in the sinusoidal example.

6 CONCLUSION

This paper presents a new texture-based method to display vector fields and correlated scalar values, that is a common requirement in scientific visualization. The bump mapping method is used to show two additional scalar values, leaving colors for the visualization of a third scalar value. The whole process can be done in hardware with usual graphics cards. The technique seems to be effective, and a real gain in multivariate visualization is perceived, even if not all scalar values can be represented well.
We thank Dr. Marco Tarini of the University of Pisa for useful discussions.

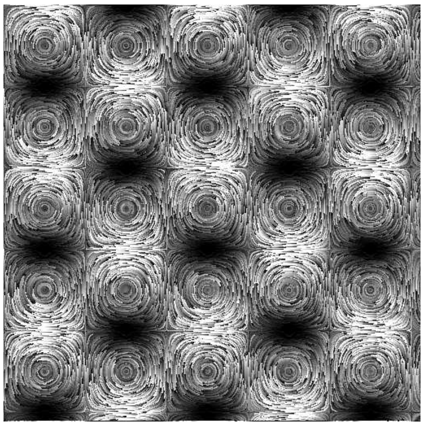

Figure 11: BLIC algorithm on the sinusoidal example.

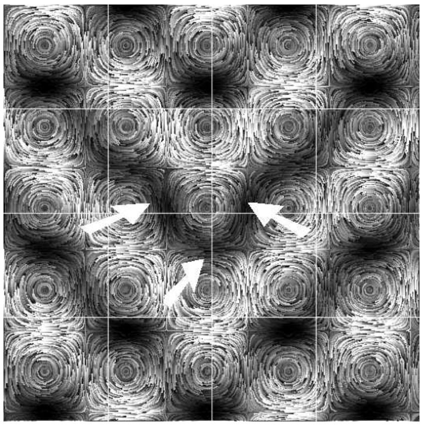

Figure 12: B^2LIC algorithm on the sinusoidal example.

REFERENCES

[Wijk91] van Wijk, J.J.: *Spot noise-texture synthesis for data visualization*, ACM Computer Graphics (Proc. of SIGGRAPH '91), Vol. 25, pp. 309-318, 1991.

[Cabra93] Cabral, B. and Leedom, L.C.: *Imaging Vector Fields Using Line Integral Convolution*, ACM Computer Graphics (Proc. of SIGGRAPH '93), Vol. 27, pp. 262-270, 1993.

[Stall95] Stalling, D. and Hege, H.C.: *Fast and resolution independent line integral convolution*, ACM Computer Graphics (Proc. of SIGGRAPH '95), pp. 249-256, 1995.

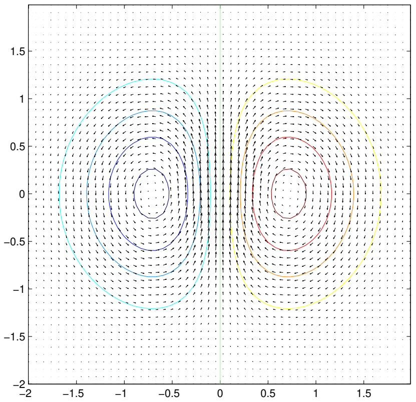

Figure 13: Vector field of the exponential example.

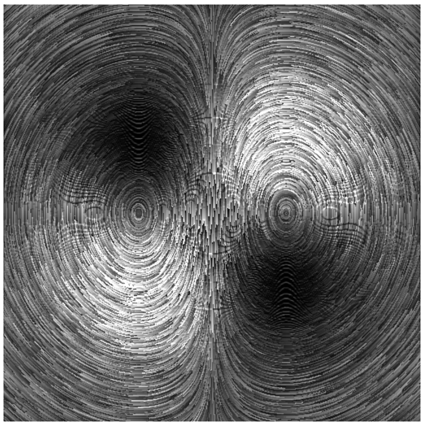

Figure 14: BLIC algorithm on the exponential example.

[Wegen97a] Wegenkittl, R., Gröller, E. and Purgathofer, W.: *Animating Flowfields: Rendering of Oriented Line Integral Convolution*, In *IEEE* Visualization'97 Proceedings, pp. 15-21, 1997.

[Wegen97b] Wegenkittl, R. and Gröller E.: *Fast oriented line integral convolution for vector field visualization via the internet*, In *IEEE* Visualization'97 Proceedings, pp. 309-316, 1997.

[Shen96] Shen, H.W., Johnson, C.R., and Ma, K.L.: *Visualizing vector fields using line integral convolution and dye advection*, Proceedings of the ACM Symposium on Volume Visualization'96, pp. 63-70, 102, 1996.

[Sanna00a] Sanna, A. and Montrucchio, B.: *Adding a scalar value to 2D vector field visualization: the BLIC (Bumped LIC)*, Eurographics'2000 Short Presentations Proceedings, pp. 119-124, 2000.

[Delee95] De Leeuw, W.C. and van Wijk, J.J.: *Enhanced spot noise for vector field visualization*, In *IEEE* Visualization'95 Proceedings, pp. 233-239, 1995.

[Zockl97] Zöckler, M., Stalling, D. and Hege, H.C.: *Parallel line integral convolution*, Parallel Computing, Vol. 23, pp. 975-989, 1997.

[Khous99] Khousas, L., Odet C. and Friboulet, D.: *2D Vector Field Visualization Using Furlike Texture*, In Proceedings of the Joint Eurographics and *IEEE* TCVG Symposium on Visualization, pp. 35-44, 1999.

[Sanna00b] Sanna, A. Montrucchio, B. and Montuschi, P.: *A survey on visualization of vector fields by texture-based methods*, Recebt Res. Devel. Pattern Rec., 1, pp. 13-27, 2000.

[Blinn78] Blinn, J.: *Simulation of wrinkled surfaces*, ACM Computer Graphics (Proc. of SIGGRAPH '78), Vol. 12, pp. 286-292, 1978.

[Samet84] Samet, H.: *The Quadtree and Related Hierarchical Data Structures*, ACM Computer Surv. Vol. 16(2), pp. 187-260, 1984.