# An Efficient and Topological Correct Polygonisation Algorithm for Finite Element Data Sets

Burkhard Wünsche[1] and Jenny Zheng Lin[2]
Division for Biomedical Imaging & Visualization, Department of Computer Science,
University of Auckland, Auckland, New Zealand
[1]burkhard@cs.auckland.ac.nz, [2]jennylin_nz@yahoo.com

## Abstract

Implicit defined surfaces of scalar fields (isosurfaces) are a common entity in biomedical, scientific and engineering science. Polygonising an isosurface permits hardware assisted rendering and simplified geometric operations such as surface analysis and area computation. This paper introduces a novel algorithm for computing isosurfaces for scalar fields defined over (potentially curvilinear) finite elements which are common in numerical simulations and physically-based modelling. The advantages of the method are demonstrated by visualising the myocardial strain in a healthy and a diseased heart.

**Keywords**: polygonisation methods, isosurfaces, Marching Cubes, curvilinear elements, finite elements

## 1 Introduction

Implicit defined surfaces of scalar fields (isosurfaces) are common in biomedicine and other sciences. Isosurfaces impart knowledge about the overall distribution of a scalar field and can be used to extract anatomical structures from medical imaging data. The *c-isosurface* of a scalar field $s$ is defined as all points $\mathbf{x}$ for which $s(\mathbf{x}) = c$.

Numerous algorithms (so-called *polygonisation methods*) have been proposed for the efficient computation of isosurfaces (e.g., [1, 2, 3, 4]). Interactive display rates and reduced storage requirements can be achieved by utilising adaptive methods [5], mesh reduction techniques [6] and multi-resolution meshes [7, 8]. A survey and analysis of polygonisation methods and optimisation techniques to achieve faster computation and rendering of isosurfaces is found in [9].

The Marching-Cube algorithm [10] has been one of the earliest and most popular methods. The algorithm requires as input a regular grid of sampled field values and "marches" through the volume cell-by-cell. Each grid cell has eight sample values at its corners. The method constructs a tessellation by computing for each cell the intersection points of the cell's edges with the isosurface and by connecting these intersection points with triangles obtained by a table look-up. The look-up table contains all configurations which fulfill the assumption that the isosurface intersects a cell's edge at most once. Since there are eight vertices in each cubic cell and two values, *positive* and *negative*, there are $2^8 = 256$ ways the surface can intersect the cube. Lorensen and Cline use symmetries to reduce the number of patterns to 15 which are shown in figure 1[1].

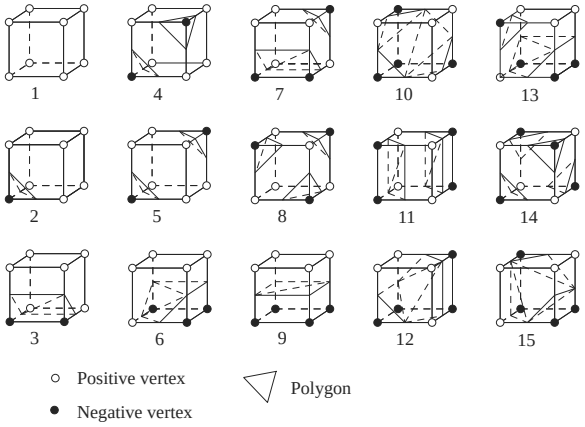


Figure 1: Triangulated cubes.

The main disadvantage of the algorithm is that some patterns in figure 1 are topologically ambiguous as noted by van Gelder and Wilhelms [12]. This may produce a surface with a hole as pointed out by Düurst [13] (see figure 2). The literature offers various solutions to the ambiguity problem [14, 15, 16, 17, 18]. Also in some applications the topology of a biomedical structure is known in advance and specialised polygonisation algorithms can be employed to take this into consideration [19].

In this paper we present a modification of the Marching Cubes algorithm which can be applied directly to curvi-

[1]The cases 12 and 15 are reflective with respect to the xy-plane. This leaves 14 topologically distinct patterns (22 without inversed patterns) [11].

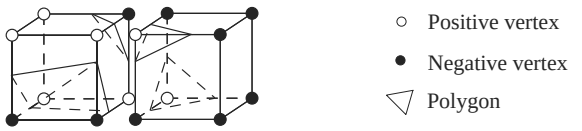


Figure 2: A hole in the polygonisation because of a face ambiguity.

linear finite element domains without requiring that the field is sampled over the bounding box enclosing the domain. The resulting algorithm is more precise and efficient and can take into account the complexity of the model domain. Furthermore we suggest an efficient table look-up scheme to resolve ambiguous cube configurations.

## 2 Finite-Element Geometry

The geometry of a *finite element* model is described by a set of nodes and a set of elements, which have these nodes as vertices. The nodal coordinates are interpolated over an element using *interpolation functions*. Curvilinear elements can be defined by specifying nodal derivatives.

As an example of a finite element consider the cubic Hermite-linear Lagrange element in two dimensions shown in figure 3 (b). We first specify a parent element, shown in part (a) of the figure, which is a square in $\xi$-parameter space. The coordinates $\xi_i$ ($0 \leq \xi_1, \xi_2 \leq 1$) are called the element or *material coordinates*. The value of some variable $u$ (e.g., temperature) at the material coordinates $\xi$ is then specified by interpolating the variables $u_i$ linearly in the given parameter direction. In our example we assume that additionally derivatives in $\xi_1$-direction $\left(\frac{\partial u}{\partial \xi_1}\right)_i$ $(i = 1,\ldots,4)$ are specified at the element nodes. In this case a cubic Hermite interpolation is performed in that direction.

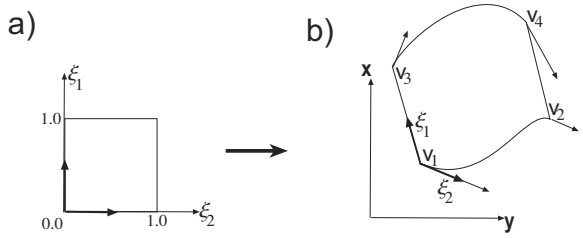


Figure 3: A cubic Hermite-linear Lagrange finite element.

The cubic Hermite-linear Lagrange interpolation of $u$ over the entire 2D parameter space is then defined by the tensor products of the interpolation functions in each parameter direction:

$$
\begin{aligned}
u(\xi_1,\xi_2) = \; & H_1^0(\xi_1)L_1(\xi_2)u_1 + H_2^0(\xi_1)L_1(\xi_2)u_2 \\
+ \; & H_1^0(\xi_1)L_2(\xi_2)u_3 + H_2^0(\xi_1)L_2(\xi_2)u_4 \\
+ \; & H_1^1(\xi_1)L_1(\xi_2)\left(\frac{\partial u}{\partial \xi_1}\right)_1 + H_2^1(\xi_1)L_1(\xi_2)\left(\frac{\partial u}{\partial \xi_1}\right)_2 \\
+ \; & H_1^1(\xi_1)L_2(\xi_2)\left(\frac{\partial u}{\partial \xi_1}\right)_3 + H_2^1(\xi_1)L_2(\xi_2)\left(\frac{\partial u}{\partial \xi_1}\right)_4
\end{aligned}
$$

where

$$L_1(\xi) = 1 - \xi, \;\text{ and }\; L_2(\xi) = \xi$$

are the one-dimensional linear Lagrange basis functions, and

$$
\begin{aligned}
H_1^0(\xi) &= 1 - 3\xi^2 + 2\xi^3, & H_1^1(\xi) &= \xi(\xi-1)^2 \\
H_2^0(\xi) &= \xi^2(3 - 2\xi), & H_2^1(\xi) &= \xi^2(\xi - 1)
\end{aligned}
$$

are the one-dimensional cubic Hermite basis functions.

In general we can express the interpolation of a variable as

$$u(\xi) = \sum_{i,k} u_i^k \phi_i^k(\xi) \tag{1}$$

where $u_i^k$ are scalar field values and their partial derivatives (if any) at each node and $\phi_i^k$ are appropriate interpolation functions. Analogously we can compute for a point in material coordinates $\xi$ the corresponding world coordinates $\mathbf{x}$ by using the isoparametric map

$$\mathbf{x}(\xi) = \sum_{i,k} \mathbf{x}_i^k \phi_i^k(\xi) \tag{2}$$

where $\mathbf{x}_i^k$ are the nodal coordinates and the coordinate curve tangents (if any) at the nodes.

## 3 A Polygonisation Algorithm for Curvilinear Finite Element Data

We have developed a polygonisation algorithm which computes an isosurface in material space. The algorithm divides the cubic parent element of each (potentially curvilinear) finite element into a regular grid of $(n+1)^3$ sample values which form $n^3$ cubes in material space. The algorithm determines how the surface intersects a cube, then moves to the next cube. The isosurface intersection is determined by the sign of the scalar field at the cube's vertices. Each edge with vertex values of different sign is assumed to intersect the isosurface once. The intersection point in material coordinates is approximated by linearly interpolating the scalar field values between the vertices.

The surface normals of the isosurface are given by the field's gradient function if it is defined and if its use is appropriate. Otherwise the normals are determined by first precomputing the material coordinate gradients for

all grid points using finite differences. For each isosurface intersection the $\xi$-derivative of the scalar field $s$ at that point is then approximated by linearly interpolating the gradients at the grid vertices. Finally the surface normal is given by the gradient in world coordinates which is

$$\nabla s = \begin{pmatrix} \frac{\partial s}{\partial x_1} \\ \frac{\partial s}{\partial x_2} \\ \frac{\partial s}{\partial x_3} \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{3} \frac{\partial s}{\partial \xi_i} \frac{\partial \xi_i}{\partial x_1} \\ \sum_{i=1}^{3} \frac{\partial s}{\partial \xi_i} \frac{\partial \xi_i}{\partial x_2} \\ \sum_{i=1}^{3} \frac{\partial s}{\partial \xi_i} \frac{\partial \xi_i}{\partial x_3} \end{pmatrix} = \nabla_\xi s \, \mathbf{J}^{-1}$$

where $\mathbf{J}^{-1} = \frac{\partial \xi_i}{\partial x_j}$ is the inverse of the Jacobian of the isoparametric mapping (equation 2) and $\nabla_\xi s$ is the gradient of $s$ with respect to the material coordinates.

## 3.1 Resolving Face Ambiguities

Some configurations, such as number 4 in figure 1, contain *ambiguous faces* for which the edge intersection points can be connected in two different ways shown in figure 4. Nielson and Hamann [14] achieve a disambiguation by determining the topology of the bilinear interpolant over a face from the intersection of its asymptotes. Mackerras shows that this test can be replaced by sorting the four intersection points along one coordinate [20]. The first pair and the last pair of the sorted points are connected.

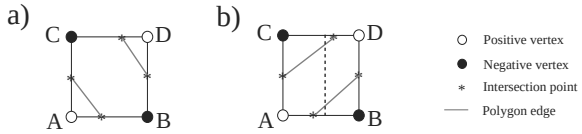


Figure 4: An ambiguous face.

The justification for this result is shown in figure 4. Assumed the connection in (b) is correct then the scalar field along the dotted line varies from negative to positive to negative. However, it can be shown easily that the bivariate interpolant varies linearly parallel to any coordinate axis so that this connection of intersection points represents an impossible topology.

We can derive from the above observation an alternative test which determines the correct edges within a face by comparing the products *AD* and *BC* which can be done without computing the intersection points. However, since the computation of the intersection points is necessary for creating the triangles inside a cell and since the intersection points are hashed for efficiency (see next subsection) it is most efficient to resolve face ambiguities by comparing the material coordinates of the intersection points. Since our application subdivides the material space we compute and sort the intersection points along one $\xi$-coordinate and then compute their world coordinates for use in the triangle creation step.

The number of possible topologically different triangulations for a pattern from figure 1 depends on its number of ambiguous faces. As an example consider the pattern 4, where only the front face is ambiguous. The two possible triangulations are shown in figure 5.

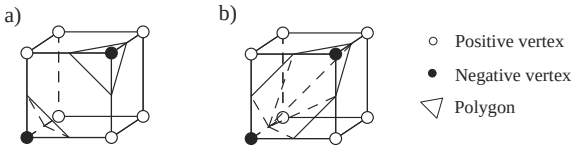


Figure 5: Two topologically different triangulations for an ambiguous pattern.

In general a configuration with $k$ ambiguous faces has $2^k$ topologically different triangulations. Our implementation precomputes a table consisting of all topologically different triangulations using the algorithm shown in figure 6 which results in a table with 656 entries.

The table is indexed by first using an 8-bit index for the main configuration and then using a $k$-bit index for the subconfiguration where $k$ is the number of ambiguous faces of that configuration.

## 3.2 Vertex and Normal Hashing

Vertices are shared by up to eight cubes and edges by up to four cubes. Furthermore the edge intersection points are also required for resolving face ambiguities. In order to prevent recalculations of values we use three hash tables. One hash table stores the $\xi$-coordinates of a grid point. A second hash table stores the $\xi$-coordinate gradient at grid vertices. This is only necessary if the normal at an intersection point can not be computed from the gradient of the interpolated field. A third hash table stores the edge intersection points and the corresponding normals in world coordinates.

Since a face is shared by two cells one could consider using another hash table to prevent recomputation of the correct edge connection for a face. However, comparing the intersection point's $\xi$-coordinates is faster than computing the hash function for a cube's face and an additional hash table is therefore unnecessary.

## 3.3 Summary of the Algorithm

The algorithm can be summarized as follows

1. Precalculate a look-up table with all topologically different triangulations
2. Subdivide the material space into cubic cells.
3. Calculate an 8-bit index for each cube from the sign of the eight scalar field values at the cube vertices.

```
For config = 0 to 255
  ambiguousFaces = set of all ambiguous faces computed from bit pattern of config
  intersectingEdges = set of all intersecting edges computed from bit pattern of config
  For subConfig = 0 to 2^numAmbiguousFaces
    unusedEdges = intersectingEdges
    LUTable[config][subConfig].polygons = []
    While unusedEdges ≠ ∅
      currentIntersectingEdge = remove any edge from unusedEdges
      Initialise new outputPolygon
      firstIntersectingEdge = currentIntersectingEdge
      repeat
        outputPolygon.add(currentIntersectingEdge)
        Choose face to right of currentIntersectingEdge (if going negative→positive)
        if face is not ambiguous or if subConfig bit
              corresponding to current face is 0              // case (a) in figure 4
            currentIntersectingEdge = first intersecting edge clockwise around face
                                      from currentIntersectingEdge
        else                                                  // case (b) in figure 4
            currentIntersectingEdge = first intersecting edge anti-clockwise around face
                                      from currentIntersectingEdge
      until currentIntersectingEdge == firstIntersectingEdge
      LUTable[config][subConfig].polygons += outputPolygon
```

Figure 6: Algorithm for computing a table with all topologically different triangulations.

4. If the configuration has $k$ ambiguous faces compute a $k$ bit sub-index by comparing the $\xi$-coordinates of the intersection points as explained in subsection 3.1.

5. Using the index, look up the list of edges forming triangles from the precalculated table.

6. Using the scalar field values at each edge vertex linearly interpolate the isosurface intersection in material coordinates and compute its world coordinates.

7. Compute the normal at an edge vertex using the field's gradient function if possible. Otherwise compute the gradient in material coordinates at the grid points using finite differences, linearly interpolate the gradients and transform the result to world coordinates.

Performing the isosurface computation in material space has the advantage that scalar field values can be computed directly without performing a multi-dimensional Newton method or resampling the data. The resulting isosurface lies smoothly inside the finite element, i.e., there are few bits of the isosurface sticking out of the model boundaries and there are no erroneous results due to sample values which lie outside the model boundary and for which the scalar field is undefined. Furthermore the method is more efficient since only the actual model domain is subdivided rather than its bounding box in world coordinates. Finally the computation in material space is often more precise. For example, if tricubic elements are used then the linear interpolation used to compute the intersection points of the cube's edges with the isosurface will yield the exact result. In contrast the computation in world coordinates is exact only if the elements are cuboidal and if the sample grid is aligned with each element.

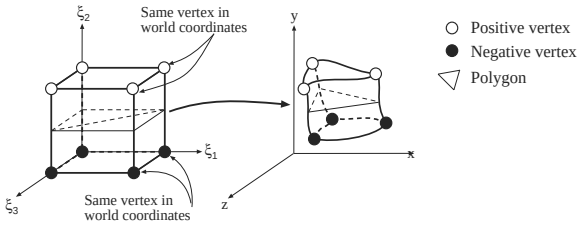


Figure 7: Isosurface within a degenerate finite element approximated with one Marching Cubes cell.

It is interesting to note that our polygonisation algorithm is stable even if degenerate finite elements are used. Figure 7 gives an example of an element which has two pairs of vertices with the same world coordinates. If we approximate the finite element with a single MC cell then we obtain two triangles (configuration 9) where one of the triangles is a line since the two edge intersections on the right face have the same world coordinates. Our algorithm automatically removes such degenerate triangles since they are not rendered and since the polygonised isosurface might be

used as input to a postprocessing step such as a mesh reduction algorithm.

## 4 Results

We use our algorithm to visualise the normal strains inside the myocardium of a healthy and a diseased left ventricle. The model for reconstructing the 3D motion and strain of the left ventricle from tagged MR images has been developed by Young et al. based on a finite element model of the left ventricle [21, 22]. The model consists of 16 finite elements with its geometry being interpolated in the radial direction using linear Lagrange basis functions and in the circumferential and longitudinal directions using cubic Hermite basis functions. The strain field is represented by 10x10x6 sample points per element with 10 sample points each in the circumferential and longitudinal directions and 6 sample points in the radial direction.

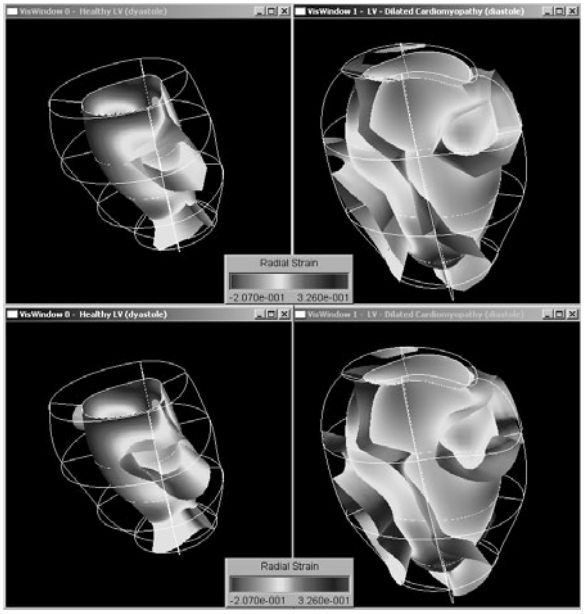


Figure 8: The myocardial radial strain in the left ventricle of a healthy (left) and diseased (right) heart visualised using colour mapping and the 0-isosurface.

Figure 8 shows the myocardial radial strain in the left ventricle of a healthy (left) and diseased (right) heart visualised using colour mapping and the 0-isosurface. The isosurface separates regions of the myocardium contracting and stretching in the radial direction. The images on the left of the figure show clearly that the healthy left ventricle expands in the radial direction (wall thickening). The only exceptions are three small cylindrical regions at the apex and the septal and lateral wall. For the diseased heart wall thickening is observed in the basal-lateral wall, the basal-septal wall and in parts of the anterior and inferior wall. The rest of the myocardium shows an abnormal deformation. Analysing additionally the distributions of the normal strains in circumferential and longitudinal direction shows that the diseased ventricle does not contract evenly but rather performs a shape change [23].

The images in the top row of figure 8 were created using just 3x3x3 cells per finite element. It can be seen that this is sufficient to obtain a close approximation of the isosurface. Using the 10x10x6 sample points which define the strain field (bottom row) gives a virtually smooth polygonisation.

We have computed different isosurfaces for different models and found that ambiguous configurations represent less than 5% of all intersected cells. The added complexity of the algorithm and the slight overhead caused by computing the indices for the expanded look-up table are more than compensated for by the fact that we obtain a topologically correct hole-free polygonisation. Note that these two properties are particularly important in biomedical sciences since holes could easily be interpreted as anatomical of physiological abnormalities.

## 5 Conclusion

We have introduced a novel algorithm for computing isosurfaces for scalar fields defined over curvilinear finite elements. The algorithm is fast, topologically correct, gives a nearly precise fit of the isosurface with respect to the model geometry and is stable for degenerate finite elements. Ambiguous cube configurations are resolved using an efficient table look-up scheme. An algorithm for creating this table automatically was suggested. The advantages of the algorithm were demonstrated by visualising the myocardial strain in a healthy and a diseased left ventricle.

## 6 Acknowledgements

We would like to thank Alistair A. Young from the Department of Physiology of the University of Auckland, Auckland, New Zealand, for valuable discussions and for providing us with the models of the left ventricle. Our thanks also goes to Dr. Richard White of the Cleveland Clinic, Cleveland, Ohio, USA, who kindly provided tagged MRI data of a left ventricle diagnosed with dilated cardiomyopathy.

## References

[1] Geoff Wyvill, Craig McPheeters, and Brian Wyvill. Animating soft objects. *The Visual Computer*, 2(4):235 – 242, August 1986.

[2] Jules Bloomenthal. Polygonization of implicit surfaces. *Computer-Aided Geometric Design*, 5(4):341 – 355, November 1988.

[3] A. Doi and A. Koide. An efficient method of triangulating equi-valued surfaces by using tetrahedral cells. *IEICE Trans. Commun. Elec. Inf. Syst.*, E-74(1):214 – 224, January 1991.

[4] Jane Wilhelms and Allan van Gelder. Octrees for faster isosurface generation. *ACM Transactions on Graphics*, 11(3):201 – 227, July 1992.

[5] Jules Bloomenthal and Brian Wyvill. Interactive techniques for implicit modeling. *Computer Graphics*, 24(2):109 – 116, March 1990. Special Issue on 1990 Symposium on Interactive 3D Graphics.

[6] Alan D. Kalvin and Russell H. Taylor. Superfaces: Polygonal mesh simplification with bounded error. *IEEE Computer Graphics and Applications*, 16(3):64–77, May 1996.

[7] Hugues Hoppe. View-dependent refinement of progressive meshes. In *SIGGRAPH '97 Conference Proceedings*, Annual Conference Series, pages 189–198. ACM SIGGRAPH, Addison-Wesley Publication Company Inc., August 1997. Held in Los Angeles, California, August 03-08, 1997, URL: `http://www.research.microsoft.com/research/graphics/hoppe`.

[8] Michael Lounsbery, Tony D. DeRose, and Joe Warren. Multiresolution analysis for surfaces of arbitrary topological type. *ACM Transactions on Graphics*, 16(1):34 – 73, January 1997.

[9] Burkhard C. Wünsche. A survey and analysis of common polygonization methods & optimization techniques. *Machine Graphics & Vision*, 6(4):451 – 486, 1997.

[10] W. Lorensen and H. Cline. Marching cubes: A high resolution 3D surface construction algorithm. *Computer Graphics*, 21(4):163 – 169, July 1987. Proceedings of SIGGRAPH '87.

[11] S. Lobregt, P. W. Verbeek, and F. C. A. Groen. Three-dimensional skeletonization: Principle and algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-2(1):75 – 77, January 1980.

[12] Allen van Gelder and Jane Wilhelms. Topological considerations in isosurface generation. *ACM Transactions on Graphics*, 13(4):337 – 375, October 1994.

[13] Martin J. Düurst. Additional reference to "marching cubes". *Computer Graphics*, 22(2):72, April 1988. Letter.

[14] Gregory M. Nielson and Bernd Hamann. The asymptotic decider: Resolving the ambiguity in marching cubes. In Gregory M. Nielson and Larry Rosenblum, editors, *Proceedings of Visualization '91*, pages 83 – 91, Los Alamitos, California, October 1991. IEEE, Computer Society Press.

[15] C. Montani, R. Scateni, and R. Scopigno. A modified look-up table for implicit disambiguation of marching cubes. *The Visual Computer*, 10(6):353–355, July 1994.

[16] Jacques-Olivier Lachaud and Annick Montanvert. Continuous analogs of digital boundaries: A topological approach to iso-surfaces. *Graphical models*, 62(3):129–164, May 2000.

[17] Gregory M. Nielson. On marching cubes. *IEEE Transactions on Visualization and Computer Graphics*, 9(3):283–297, July 2003.

[18] Adriano Lopes and Ken Brodlie. Improving the robustness and accuracy of the marching cubes algorithm for isosurfacing. *IEEE Transactions on Visualization and Computer Graphics*, 9(1):16 – 29, January 2003.

[19] Stephan Bischoff and Leif Kobbelt. Isosurface reconstruction with topology control. In *Proceedings of the 10th Pacific Conference on Computer Graphics and Applications (Pacific Graphics '02)*, pages 246–255. IEEE CS Press, October 2002. URL: `http://www-i8.informatik.rwth-aachen.de/publications/downloads/topo.pdf`.

[20] Paul Mackerras. A fast parallel marching-cubes implementation on the Fujitsu AP1000. Technical report, Department of Computer Science, Australien National University, August 1992. URL: `http://cs.anu.edu.au/techreport/1992/TR-CS-10.ps.gz`.

[21] Alistair A. Young and Leon Axel. Three-dimensional motion and deformation of the heart wall: Estimation with spatial modulation of magnetization - a model-based approach. *Radiology*, 185:241–247, 1992.

[22] Alistair A. Young, Dara L. Kraitchman, Lawrence Dougherty, and Leon Axel. Tracking and finite element analysis of stripe deformation in magnetic resonance tagging. *IEEE Transactions on Medical Imaging*, 14(3):413 – 421, September 1995.

[23] Burkhard C. Wünsche and Alistair A. Young. The visualization and measurement of left ventricular deformation using finite element models. *Journal of Visual Languages and Computing - Special Issue on Biomedical Visualization for Bioinformatics*, 14(4):299–326, August 2003.

# Bi-Quadratic Interpolation of Intensity for Fast Shading of Three Dimensional Objects

*Ekta Walia and †Chandan Singh

## Abstract

Researchers in the field of Computer Graphics are often confronted with the trade off between visual realism and computational cost. So far, Phong and Gouraud shading have been treated as well established methods and attempts have been made to improve visual realism or to reduce computational cost or both. These methods use linear interpolation to compute the normals or intensity, respectively, at each point on the surface. However, it has been proved that no surface would yield proper distribution of illumination generated by the traditional Phong shading. Attempts have been made to improve the defects of linear interpolation used in Phong shading. One such attempt is the use of biquadratic normal vector interpolation. In this paper we have proposed an algorithm to achieve the visual realism of this method and at the same time we have reduced the cost of shading.

**Keywords:** Phong Shading, Linear Interpolation, Quadratic Interpolation, Biquadratic Interpolation, Bezier Triangle

## 1. Background:

In increasing order of visual realism, there are three well-known shading methods. The simplest shading model for a polygon mesh is Constant Shading or Flat Shading [1]. It uses illumination model once to determine a single intensity value that is then used to shade an entire polygon. This shading method does not produce the variations in shade across the polygon that should actually occur for visual realism. However, in this method, no interpolation takes place for variation in shade. This is a very fast method.

Gouraud shading [2], also called intensity interpolation shading or color interpolation shading, eliminates the intensity discontinuities across the adjacent polygons. Although a fairly fast method of shading, it suffers from Mach Band effect and fails to capture detailed lighting characteristics. All the subsequent works are a variation of such interpolative shading.

Phong Shading [3], also known as normal-vector interpolation shading, interpolates the surface normal vector rather than the interpolation of intensity. The interpolation occurs across a polygon span on a scan line, between starting and ending normals for the span. These normals are interpolated along polygon edges from the vertex normals. These interpolated normals are then used in intensity calculations. Phong shading yields substantial improvements over Gouraud shading when an illumination model with specular reflection is used. With this method, however the cost of shading is increased since the interpolated normal are used to interpolate intensity over the surface of a polygon.

## 2. Existing Methods:

### 2.1 Attempts to improve Visual realism:

Many researchers have contributed a lot to improve the visual realism proposing various interpolation techniques.

Overveld and Wyvill [4] proposed a quadratic normal vector interpolation algorithm to replace the traditional linear interpolation. Their algorithm is an extension of Phong shading in which quadratic interpolation of normals is done. It overcomes the inappropriateness of traditional linear interpolation when a surface approximated by polygons has

* Lecturer, Department of Computer Science, Technical Teachers Training Institute, Chandigarh, India
wekta@yahoo.com
† Professor&Head, Department of Computer Science &Engg, Punjabi University, Patiala, India
chandan@pbi.ac.in