Eurographics Conference on Visualization (EuroVis) 2017
B. Kozlíková, T. Schreck, and T. Wischgoll (Editors)

Short Paper

Marching pentatopes for continuous morphing of isosurfaces from four dimensional data in HTML5/WebGL

A. R. Watters[1] [1]Center for Computation Biology, Flatiron Institute of the Simons Foundation, New York, New York, USA

Abstract

Animations which show three dimensional volumes continuously changing over time facilitate the exploration and analysis of complex data sets such as calcium image data of neural activity and phase contrast magnetic resonance imaging of blood flows. This paper explains the marching pentatopes *method for representing the iso-surfaces of a four dimensional data set as a triangulated surface smoothly deforming as time progresses. The morphing triangulations generated by the this method may be rendered using the morph geometry capabilities provided by the* three.js *javascript library for cross platform HTML5/WebGL presentation in standard web browsers [Cab17].*

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—Display algorithms

1. Introduction

Biological data is often four dimensional in raw form. For example phase-contrast magnetic resonance imaging [DBB*15] detects blood flow velocity in three spatial dimensions varying over a sequence of time samples and calcium imaging detects neuron activity in three dimensions varying over time [GKH07]. It is useful to be able to visualize the data directly using morphing 3 dimensional isosurfaces in order to identify characteristics, features, or measurement problems.

This paper presents the *marching pentatopes* method for deriving smoothly morphing triangularized geometries approximating iso-contours of scalar data $f(x,y,z,t)$ within a grid in a four dimensional volume as implemented in the *contourist library* [Wat17] for numeric Python [JOP*]. We present the marching pentatopes method as a generalization of the lower dimensional *marching triangles* method for deriving contour lines and the *marching tetrahedra* method for deriving iso-surfaces. The *contourist* package implements all of these methods using a common object oriented architecture designed to be used in conjunction with the *three.js* javascript library for WebGL [Cab17] which allows the geometric structures generated by *contourist* to be rendered in web browsers in combination with the features provided by *three.js* – such as lighting, shadowing, text rendering, various material implementations and interactive controls [Dir13].

2. Preliminaries

The *contourist* software computes implicit figures for any target value and any grid geometry. This presentation uses a simplified framework without loss of generality for notational convenience. Below we present some terminology and definitions useful in the explanation to follow.

We consider the problem of approximating zero valued iso-contours interpolated within grids limited to coordinates values $N = \{0,1,...n-1\}$ for some fixed positive integer n where the grid N^d has dimension $d \in \{2,3,4\}$. The contour computation seeks to interpolate a fixed function $f : N^d \to \Re$ approximating a hypothetical point set $\{\mathbf{p} \in \Re^d | f(\mathbf{p}) = 0\}$ of the zeros of f within the limits of the grid N^d.

We say a point $\mathbf{p} \in N^d$ is a *positive* if $f(\mathbf{p}) \geq 0$ or we say $\mathbf{p}$ is a *negative* if $f(\mathbf{p}) < 0$. A set of grid points $S \subset N^d$ *crosses zero* if S contains at least one negative point and at least one positive point. If $\{\mathbf{p},\mathbf{p}'\} \subset N$ cross zero the *interpolated zero* $\mathbf{p} \circ \mathbf{p}'$ is defined as $\mathbf{p} \circ \mathbf{p}' = \frac{\mathbf{f(p)p'-f(p')p}}{\mathbf{f(p)-f(p')}} \in \Re^d$.

Two grid points $\mathbf{p},\mathbf{p}' \in N^d$ are *adjacent* if $\max(|p_i - p'_i|) = 1$. A sequence of grid points $\mathbf{p_0},\mathbf{p_1},..,\mathbf{p_m} \in N^d$ where each $\mathbf{p_i}$ is adjacent to $\mathbf{p_{i+1}}$ define a *grid path* as the union of the line segments connecting each $\mathbf{p_i}$ to $\mathbf{p_{i+1}}$.

For any finite set of points $S = \{\mathbf{p_0},\ldots,\mathbf{p_m}\} \subset N^d$ we define the *convex closure* of S, $\mathcal{C}(S)$, to be the set of points generated by $\sum \alpha_i p_i$ for any set $\alpha_i \in [0,1]$ where $\sum \alpha_i = 1$. If the points $S = \{\mathbf{p_0},\ldots,\mathbf{p_m}\}$ are linearly independent we say that the convex closure $C(S)$ defines a *simplex of dimension* $m-1$.

The *voxel vertices* $P(\mathbf{p})$ for a grid point $\mathbf{p} \in N^d$ are the set of grid points including $\mathbf{p}$ and also including all $\mathbf{p}' \in N^d$ adjacent to $\mathbf{p}$ where all $p'_i \geq p_i$. The *voxel* $V(\mathbf{p}) = \mathcal{C}(P(\mathbf{p}))$ is the convex closure of the voxel vertices $P(\mathbf{p})$ – a square if $d = 2$, a cube if $d = 3$, or a tesseract if $d = 4$.

A *contour* separating a positive seed point $\mathbf{p} \in N^d$ and a nega-

DOI: 10.2312/eurovisshort.20171125

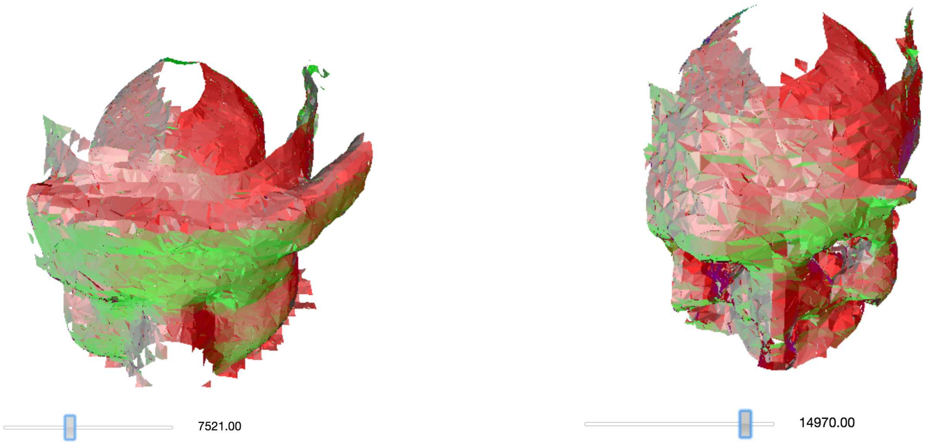

Figure 1: *Smoothly interpolated CT-Scan data from the Stanford volume data archive [Lev00] sampled using local variance at different level sets..*

tive seed point $\mathbf{p}' \in N^d$ is a set of linearly independent point sets $C = \{c_0, .., c_m\}$ where each $c_i \subset \Re^d$ of size d represents a simplex of dimension $d-1$ such that every grid path connecting $\mathbf{p}$ and $\mathbf{p}'$ intersects the simplex $\mathcal{C}(c_i)$ for some c_i. Additionally for each c_i there must be a grid point $\mathbf{p} \in N^d$ where each point in $\mathbf{q} \in c_i$ is an interpolation of a pair of points from $P(\mathbf{p})$ that cross zero. More precisely for each $\mathbf{q} \in c_i$ there are voxel vertices $\mathbf{v}, \mathbf{v}' \in P(\mathbf{p})$ crossing zero where $\mathbf{q} = \mathbf{v} \circ \mathbf{v}'$. Note that this definition of the *contour* concept does not define a unique contour for each pair of seed points – it is possible for seed point pairs $\mathbf{p}$ and $\mathbf{p}'$ to have several contours that separate them.

The main contribution of this paper is to introduce *marching pentatopes*, a method for finding four dimensional morphing contours. In the process we describe a unified algorithm structure for finding contours for $d = 2$, $d = 3$, and $d = 4$:

In the 2 dimensional case the algorithm is a variant of the *marching triangles* contour algorithm which is a simplification of the marching squares contour algorithm [HSIW96, Map03]. In the 3 dimensional case the algorithm is a variant of the *marching tetrahedra* contour algorithm which is a simplification of the *marching cubes contour method* [WM97, LC87]. In the 4 dimensional case the algorithm, *marching pentatopes*, could be viewed as a simplification of *marching hypercubes* [RH99, BWC04]. The marching pentatopes algorithm has two special cases for dividing the four dimensional pentatope into tetrahedra where the marching hypercubes algorithm provides 74 special cases for dividing the tesseract (hypercube) into tetrahedra.

All method variants presented here only examine voxels adjacent to the contour and voxels lying on a line segment between "seed points". For this reason the algorithm need not examine the vast majority of the grid points in many cases where the grid may be large or the calculation or retrieval of the $f(\mathbf{p})$ values may be expensive.

2.1. Other related work

Weigle and Banks [WB96] present a mesh generation approach that can be used for any number of dimensions which is similar to the approach presented here. Their method introduces midpoints in the interpolation step (step 4 below). The use of midpoints results in as many as 4 triangles dividing each crossing tetrahedron in three dimensions and as many as 5 tetrahedra dividing each pentatope in four dimensions. By contrast the methods described below are more parsimonious – producing at most 2 triangles for each crossing tetrahedron in three dimensions and at most 3 tetrahedra for each crossing pentatope in four dimensions. Other methods for interpolating four dimensional fields require special purpose GPU shader programs and are not easily converted into morphing triangularized geometries as implemented by libraries such as *three.js* [BSL*13]. It is not straightforward to add additional functionality to the tightly coupled internal shader logic implemented in *three.js*.

3. Marching methods for approximating implicit figures

All of the marching methods described here have a similar outline. All basic features of the method are illustrated for the simplist case of Marching Triangles in Figure 2.

3.1. Inputs and Outputs

Inputs: A grid N^d, a function f and a pair of seed points $\mathbf{p}, \mathbf{p}' \in N^d$ where $\mathbf{p}$ is negative and $\mathbf{p}'$ is positive.

Outputs: A contour C separating $\mathbf{p}$ from $\mathbf{p}'$ in N^d and a post-processed structure derived from C suitable for rendering.

3.2. Method outline

1: Locate initial crossing voxels. Use binary search between $\mathbf{p}$ and $\mathbf{p}'$ to find a grid point $\mathbf{p_0}$ where $V(\mathbf{p_0})$ crosses zero.

2: Find all adjacent crossing voxels. Find the smallest set of grid points $G = \{\mathbf{p_i}\}$ such that $\mathbf{p_0} \in G$ and for any $\mathbf{p}, \mathbf{q} \in N^d$ if $\mathbf{p} \in G$ and $\mathbf{q}$ is adjacent to $\mathbf{p}$ and $V(\mathbf{q})$ crosses 0, then $\mathbf{q} \in G$.

3: Tile the crossing voxels into crossing simplices of dimension d. For example for $d = 3$ each crossing cube is tiled into six tetrahedra. Collect the generated simplices that cross zero into the set T.

4: Separate positive from negative vertices of simplices in T. For each simplex $s \in T$ find simplices of dimension $d-1$ with vertices that interpolate the vertices if s separating the positive from the negative vertices in s. The generated collection C of simplices of dimension $d-1$ is the desired contour.

5: Post processing. Translate the contour C into a structure suitable for rendering.

Steps 1 and 2 are applications of the well known binary search [Ben75] and transitive closure [War75] techniques. Please see the *contourist* source code for additional details of the implementation. The steps 3, 4, and 5 vary between the cases $d = 2$, $d = 3$, and $d = 4$, with each case discussed in its own subsection below.

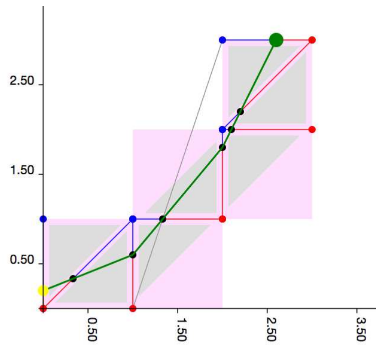

Figure 2: *Summary diagram for the marching triangles method with dimension $d = 2$ and grid size $n = 4$. Here the contour approximates $f(x,y) = y - 0.4x^2 - 0.2 = 0$ in the grid $\{0,1,2,3\}^2 = N^d$. The seed line segment between the seed points $(1,0)$ and $(2,3)$ is in dark gray (Step 1). Crossing squares are in pink (Step 2). Crossing triangles tiling the squares are in light gray (Step 3). Positive vertices are in blue. Negative vertices are in red. Line segment interpolation points are in black. Green line segments joining interpolation points separate positive from negative vertices in the crossing triangles (Step 4). Contour approximation line segments assemble into a path starting at the large yellow point and ending at the large green point (Step 5).*

3.3. Marching triangles specialized steps

This section sketches the specialized steps 3, 4, and 5 for the marching triangles method where dimension $d = 2$.

Marching triangles step 3: tile crossing squares as 2 triangles. We divide each crossing square $V(\mathbf{p})$ into two triangles (simplices of dimension 2). Of the generated triangles we select those that cross zero as the set T.

Marching triangles step 4: Separate positive and negative vertices in crossing triangles with interpolated line segments. For each crossing triangle of the tile set T we find one line segment between interpolated points lying on crossing edges. We collect the interpolated line segments generated as the contour C.

Marching triangles step 5: Assembling contour paths from line segments. Finally in order to render the contour curve in an efficient and appropriate manner using graphics libraries we must join connecting line segments into continuous paths.

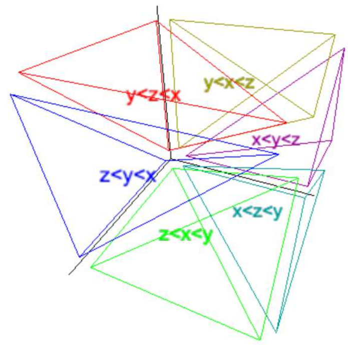

Figure 3: *Crossing cubes are tiled into six tetrahedra in marching tetrahedra method step 3. Each of the tetrahedra is associated with one of the permutations of "xyz".*

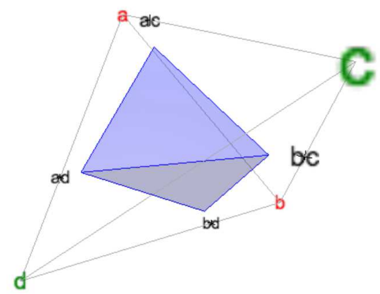

Figure 4: *For a tetrahedron with two positive vertices $\{\mathbf{a}, \mathbf{b}\}$ and two negative vertices $\{\mathbf{c}, \mathbf{d}\}$ separate positive and negative vertices using two triangles with vertices at $\{\mathbf{a} \circ \mathbf{d}, \mathbf{b} \circ \mathbf{c}, \mathbf{a} \circ \mathbf{c}\}$ and $\{\mathbf{a} \circ \mathbf{d}, \mathbf{b} \circ \mathbf{c}, \mathbf{b} \circ \mathbf{d}\}$ in marching tetrahedra step 4.*

3.4. Marching tetrahedra specialized steps

This section explains the specialized steps 3, 4, and 5 for the marching tetrahedra method where dimension $d = 3$.

Marching tetrahedra step 3: tile crossing cubes using 6 tetrahedra. We divide each crossing cube $V(\mathbf{p})$ into six tetrahedra by assigning to each of the six permutations of "xyz" a tetrahedron where the dimension quantity order is defined by the permutation. For example for the permutation "zyx" we associate the tetrahedron

$$\{(p_0 + x, p_1 + y, p_2 + z) \mid x, y, z \in [0,1] \text{ and } z \leq y \leq x\}$$

The cube tiling is illustrated in Figure 3. The tetrahedral tiles that cross zero are collected as the tile set T.

Marching tetrahedra step 4: Separate positive and negative vertices in each crossing tetrahedron using one or two triangles. For each crossing tetrahedron of the tile set T we determine the positive vertices and the negative vertices of the tetrahedron. For a tetrahedron with one positive vertex (or symmetrically one negative vertex) $\mathbf{a}$ and negative vertices $\mathbf{b},\mathbf{c},\mathbf{d}$, separate $\mathbf{a}$ from the negative points using a triangle with vertices at $\{a \circ b, a \circ c, a \circ d\}$. Otherwise there are two positive and two negative vertices in the crossing tetrahedron: separate the positive and negative vertices using two triangles as shown in Figure 4. The set of triangles generated define the contour triangulation C.

Marching tetrahedra step 5: Orient the triangles of the contour to be counterclockwise when viewed from the "outside". In order to make sure that the surface normals are computed consistently for proper lighting interactions in *three.js* the triangle vertices must be provided in anti-clockwise order when viewed from the "outside" of the volume [Muk12]. Please consult the *contourist* source code for the implementation details of this operation.

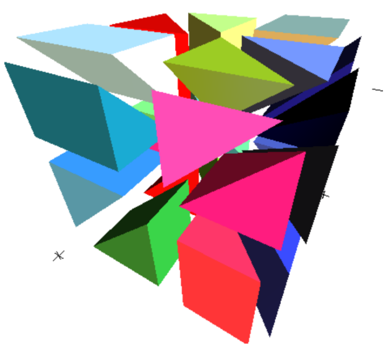

Figure 5: *The 24 pentatopes tiling a tesseract intersected with the hyperplane $t = 0.4$ from marching pentatopes step 3.*

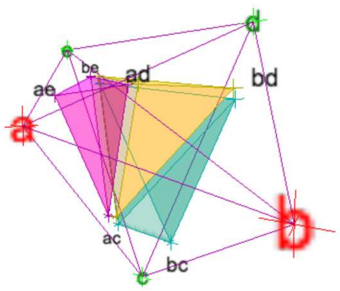

Figure 6: *For a pentatope with two positive vertices $\{\mathbf{a},\mathbf{b}\}$, here projected into 3 dimensions, separate the positive vertices from the negative vertices using three tetrahedra with vertices $\{\mathbf{a}\circ\mathbf{e},\mathbf{a}\circ\mathbf{c},\mathbf{a}\circ\mathbf{d},\mathbf{b}\circ\mathbf{e}\}$, $\{\mathbf{b}\circ\mathbf{d},\mathbf{a}\circ\mathbf{c},\mathbf{a}\circ\mathbf{d},\mathbf{b}\circ\mathbf{e}\}$, $\{\mathbf{b}\circ\mathbf{c},\mathbf{a}\circ\mathbf{c},\mathbf{b}\circ\mathbf{d},\mathbf{b}\circ\mathbf{e}\}$. in marching pentatopes step 4.*

3.5. Marching pentatopes specialized steps

This section explains the specialized steps 3, 4, and 5 for the marching pentatopes method where dimension $d = 4$.

Marching pentatopes step 3: tile crossing tesseracts using 24 pentatopes. We divide a crossing tesseract $V(\mathbf{p})$ into 24 pentatopes by assigning to each of the 24 permutations of "xyzt" a pentatope where the dimension quantity order is defined by the permutation, illustrated in Figure 5. For example for the permutation "zytx" we associate the pentatope

$$\{(p_0 + x, p_1 + y, p_2 + z, p_3 + t) \mid x,y,z,t \in [0,1] \text{ and } z \le y \le t \le x\}$$

The tiling pentatopes which cross zero define the tile set T.

Marching pentatopes step 4: Separate positive and negative vertices in each crossing pentatope using one or three tetrahedra. For each crossing pentatope of the tile set T we separate the positive vertices and the negative vertices of the pentatope using interpolated tetrahedra. For a pentatope with one positive vertex (or symmetrically one negative vertex) $\mathbf{a}$ and negative vertices $\mathbf{b},\mathbf{c},\mathbf{d},\mathbf{e}$ separate $\mathbf{a}$ from the negative vertices using one tetrahedron with vertices $\{\mathbf{a}\circ\mathbf{b},\mathbf{a}\circ\mathbf{c},\mathbf{a}\circ\mathbf{d},\mathbf{a}\circ\mathbf{e}\}$. Otherwise there are two positive and three negative vertices (or symmetrically two negative and three positive vertices) in the crossing pentatope. In that case separate the positive and negative vertices using three tetrahedra as shown in Figure 6. The tetrahedra generated define the contour C.

Marching pentatopes step 5: Convert the contour tetrahedra into morphing triangles. The *three.js* library implements morphing using "morphing triangles" where each vertex of the triangularization is associated with a three dimensional start position at a start time and a three dimensional end position at an end time. Translate C into morphing triangles using the following procedure for each tetrahedron in $\tau \in C$

Sort the t values of the vertices of τ with up to 4 unique values $t_0, t_1, \ldots$ For each t interval between the vertex t values t_i, t_{i+1} compute the intersection τ_i of τ with the hyperplane at the midpoint $t = (t_i + t_{i+1})/2$. The intersection either forms a triangle or a tetrahedron in three dimensions and each of the vertices of the intersection lies on a 4 dimensional line segment between two vertices of τ. If the intersection forms a triangle then generate a single morphing triangle using the vertices of τ that correspond to the vertices of τ_i as the start and end positions of the morph. If the intersection τ_i forms a tetrahedron then generate two morphing triangles using the vertices of τ that correspond to the vertices of τ_i as the start and end positions of the morph in a manner similar to Figure 4. The resulting set of morphing triangles are suitable for rendering using the three.js library, which uses WebGL and the GPU to render the morphs when available.

Above the choice of slicing in the t dimension is arbitrary and may be replaced with x, y or z as desired.

Acknowledgements: Tarmo Äijö pointed out many issues, errors, and possible improvements in this paper and other members of the Bonneau Laboratory for systems biology at New York University offered helpful comments and suggestions.

References

[Ben75] BENTLEY J. L.: Multidimensional binary search trees used for associative searching. *Commun. ACM 18*, 9 (Sept. 1975), 509–517. URL: http://doi.acm.org/10.1145/361002.361007, doi:10.1145/361002.361007. 3

[BSL*13] BORDIGNON A. L., SÁ L., LOPES H., PESCO S., DE FIGUEIREDO L. H.: Technical section: Point-based rendering of implicit surfaces in r4. *Comput. Graph. 37*, 7 (Nov. 2013), 873–884. URL: http://dx.doi.org/10.1016/j.cag.2013.06.005, doi:10.1016/j.cag.2013.06.005. 2

[BWC04] BHANIRAMKA P., WENGER R., CRAWFIS R.: Isosurface construction in any dimension using convex hulls. *IEEE Transactions on Visualization and Computer Graphics 10*, 2 (Mar. 2004), 130–141. URL: http://dx.doi.org/10.1109/TVCG.2004.1260765, doi:10.1109/TVCG.2004.1260765. 2

[Cab17] CABELLO C.: three.js. https://github.com/mrdoob/three.js, 2017. 1

[DBB*15] DYVERFELDT P., BISSELL M., BARKER A. J., BOLGER A. F., CARLHÄLL C.-J., EBBERS T., FRANCIOS C. J., FRYDRYCHOWICZ A., GEIGER J., GIESE D., HOPE M. D., KILNER P. J., KOZERKE S., MYERSON S., NEUBAUER S., WIEBEN O., MARKL M.: 4d flow cardiovascular magnetic resonance consensus statement. *Journal of Cardiovascular Magnetic Resonance 17*, 1 (2015), 72. URL: http://dx.doi.org/10.1186/s12968-015-0174-5, doi:10.1186/s12968-015-0174-5. 1

[Dir13] DIRKSEN J.: *Learning Three.js: The JavaScript 3D Library for WebGL*. Community experience distilled. Packt Publishing, 2013. URL: https://books.google.com/books?id=6TVeAQAAQBAJ. 1

[GKH07] GÖBEL W., KAMPA B. M., HELMCHEN F.: Imaging cellular network dynamics in three dimensions using fast 3D laser scanning. *Nature methods 4*, 1 (Jan. 2007), 73–79. URL: http://dx.doi.org/10.1038/nmeth989, doi:10.1038/nmeth989. 1

[HSIW96] HILTON A., STODDART A. J., ILLINGWORTH J., WINDEATT T.: Marching triangles: range image fusion for complex object modelling. In *Proceedings of the International Conference on Image Processing* (Sept. 1996), vol. 1, pp. 381–384. 2

[JOP*] JONES E., OLIPHANT T., PETERSON P., ET AL.: SciPy: Open source scientific tools for Python, 2001–. [Online; accessed 2017-02-17]. URL: http://www.scipy.org/. 1

[LC87] LORENSEN W. E., CLINE H. E.: Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH Comput. Graph. 21*, 4 (Aug. 1987), 163–169. URL: http://doi.acm.org/10.1145/37402.37422, doi:10.1145/37402.37422. 2

[Lev00] LEVOY M.: The stanford volume data archive. https://graphics.stanford.edu/data/voldata/, 2000. 2

[Map03] MAPLE C.: Geometric design and space planning using the marching squares and marching cube algorithms. In *2003 International Conference on Geometric Modeling and Graphics, 2003. Proceedings* (July 2003), pp. 90–95. doi:10.1109/GMAG.2003.1219671. 2

[Muk12] MUKUNDAN R.: *Advanced Methods in Computer Graphics: With examples in OpenGL*. Springer London, 2012. URL: https://books.google.com/books?id=CmfSj9z3gasC. 4

[RH99] ROBERTS J. C., HILL S.: Piecewise Linear Hypersurfaces using the Marching Cubes Algorithm. In *Visual Data Exploration and Analysis VI, Proceedings of SPIE,* (January 1999), Erbacher R. F., Pang A., (Eds.), vol. 3643, IS&T and SPIE, pp. 182–196. URL: http://www.cs.kent.ac.uk/pubs/1999/700. 2

[War75] WARREN JR. H. S.: A modification of warshall's algorithm for the transitive closure of binary relations. *Commun. ACM 18*, 4 (Apr. 1975), 218–220. URL: http://doi.acm.org/10.1145/360715.360746, doi:10.1145/360715.360746. 3

[Wat17] WATTERS A.: contourist. https://github.com/AaronWatters/contourist, 2017. 1

[WB96] WEIGLE C., BANKS D. C.: Complex-valued contour meshing. In *Proceedings of the 7th Conference on Visualization '96* (Los Alamitos, CA, USA, 1996), VIS '96, IEEE Computer Society Press, pp. 173–ff. URL: http://dl.acm.org/citation.cfm?id=244979.245051. 2

[WM97] WEHLE M., MÜLLER H.: Visualization of implicit surfaces using adaptive tetrahedrizations. *Dagstuhl '97 - Scientific Visualization Conference 00*, undefined (1997), 243. doi:doi.ieeecomputersociety.org/10.1109/DAGSTUHL.1997.10005. 2