Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

A generalized combinatorial marching hypercubes algorithm

ARTICLE INFO

Article history:
Received May 6, 2022

Manifold approximation, High-dimensional manifolds, Marching Hypercubes, Combinatorial skeleton
2020 MSC: 68U05

ABSTRACT

We present a Generalized Combinatorial Marching Hypercubes (GCMH) algorithm to compute a cell complex approximation of a manifolds of any dimension and co-dimension, that is, manifolds of dimension $n-k$ embedded into an n-dimensional space. The algorithm uses combinatorial and topological methods to avoid the use of expensive lookup tables and hence be efficient in higher dimensions. We illustrate the effectiveness of our algorithm in higher dimensions and compare its performance with a similar algorithm based on a simplicial decomposition of the domain.

1. Introduction

Methods to compute approximations of two- and three-dimensional manifolds (surfaces and volumes) are widely available and used in computer graphics and applied to animation, digital games, molecular modeling, object detection, object tracking, medical imaging, object reconstruction, etc. (see [1, 2, 3, 4, 5, 6, 7]).

Higher-dimensional manifolds are less prevalent, but they are also used in applications such as superstring theory [8]. Methods to compute such manifolds are also available; however, most of these methods are highly inefficient in high dimensions. An efficient algorithm, called Combinatorial Marching Hypercubes (CMH), to compute high dimensional manifolds of dimension $n-1$ embedded into a n-dimensional space was presented in [9].

In this paper, we present a generalization of the method presented in [9] to compute $(n-k)$-dimensional manifolds embedded into a n-dimensional space. More precisely, we present the Generalized Combinatorial Marching Hypercubes (GCMH) algorithm to compute and represent implicitly defined $(n-k)$-dimensional manifolds as a cell complex embedded into a grid of n-dimensional hypercubes. Our implementation of the algorithm can be found at `https://github.com/gknakassima/GCMH`.

An implicitly defined $(n-k)$-dimensional manifold $\mathcal{M}$ is the set of points where a function $F\colon \mathbb{R}^n \to \mathbb{R}^k$ takes on a given value c (see Section 2 for a precise definition). Figure 1 shows an example of the output of our algorithm to represent a Bagel Klein Bottle defined by $F(x,y,z,s,t) = (0,0,0)$, where $F\colon \mathbb{R}^5 \to \mathbb{R}^3$ is given by

$$F(x,y,z,s,t) = \begin{pmatrix} \cos(s)(3+\cos(s/2)\sin(t) - \sin(s/2)\sin(2t)) - x \\ \sin(s)(3+\cos(s/2)\sin(t) - \sin(s/2)\sin(2t)) - y \\ \sin(s/2)\sin(t) + \cos(s/2)\sin(2t) - z \end{pmatrix}.$$

Note that the Bagel Klein Bottle, as defined above, is a non-orientable two-dimensional manifold embedded in $\mathbb{R}^5$.

1.1. Literature review

One of the most renowned algorithms to approximate 2-dimensional manifolds embedded into a three-dimensional

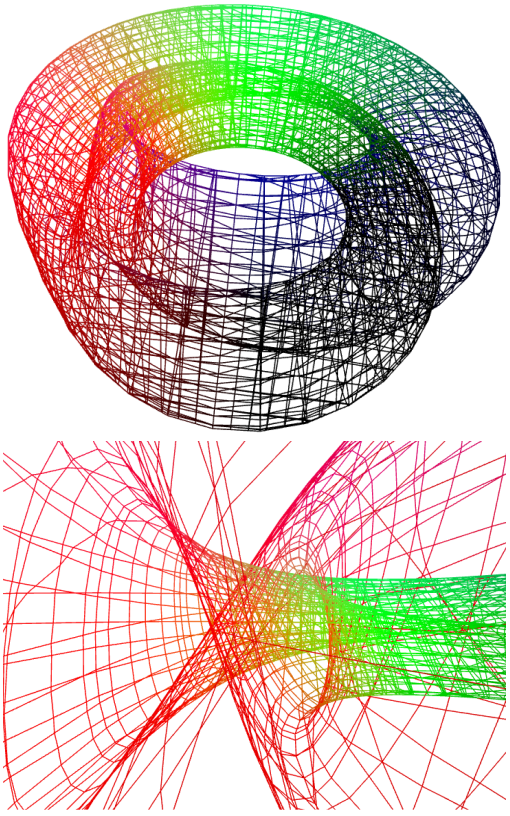


Fig. 1: Top: The Bagel Klein Bottle manifold in $\mathbb{R}^5$; Bottom: a zoom in of the same manifold showing its non-orientability.

space is the Marching Cubes algorithm [10] by Lorensen and Cline. Improvements to this algorithm to deal with ambiguities and improve topological correctness are presented in [11, 12, 13, 14, 15, 16, 17, 18, 19]. A survey of Marching Cubes type algorithms can be found in [20]. Algorithms based on a grid of tetrahedra, rather than a cubical grid, called Marching Tetrahedra are presented in [21, 22, 23, 24].

Generalizations of Marching Cubes to compute manifolds of dimension $n-1$ embedded into an n-dimensional space are presented in [25, 26, 27] for $n = 4$, and in [28, 29] for $n \geq 4$. Generalization of the Marching Tetrahedra algorithm to higher dimensions are presented in [30, 31, 32] and are usually called Marching Simplex algortihms. A continuation algorithm to create simplicial approximations of manifolds in arbitrary dimensions is presented in [33].

The methods above often employ data structures that are very memory intensive and, for this reason, are not efficient in high dimensions. In [9] we presented a memory-efficient algorithm, called Combinatorial Marching Hypercubes (CMH), to compute approximations to manifolds of dimension $n-1$ embedded into an n-dimensional space for arbitrary values of n. See [9] for more details of the relevant literature and for examples in dimensions $n > 4$.

1.2. Contributions and paper organization

In this paper, we present the Generalized Combinatorial Marching Hypercubes (GCMH) algorithm, a generalization of CMH to compute $(n-k)$-dimensional manifolds embedded into an n-dimensional space.

This paper is organized as follows. In Section 2 we present the background material and mathematical definitions used in the paper. In Section 3 we present the details of the GCMH algorithm. In Section 4 we compare the results of our method with the Combinatorial Marching Simplex algorithm from [9]. In Section 5 we present a variation of the GCMH called the Generalized Combinatorial Continuation Hypercube (GCCH) algorithm. Finally, in Section 6 we present results and comparison with other methods, and in Section 7 we present some concluding remarks.

2. Background

In this section we present the definition of an *implicitly defined manifold* and other objects used in the paper. These definitions are from [9].

Definition 2.1. Let $F\colon U \subset \mathbb{R}^n \to \mathbb{R}$ be a C^1 function. We say that a point $c \in \mathbb{R}$ is a *regular value* of F if $\nabla F(x) \neq 0$ for all $x \in F^{-1}(c)$. If $c \in \mathbb{R}$ is not a regular value of F we say that it is a *critical value* of F.

Definition 2.2. A set $\mathcal{M} \subset \mathbb{R}^n$ is called an *implicitly defined* $(n-1)$*-dimensional manifold* if there exists a differentiable function $F\colon \mathbb{R}^n \to \mathbb{R}$ and a regular value c of F such that

$$\mathcal{M} = F^{-1}(c) = \{x \in \mathbb{R}^n \mid F(x) = c\}.$$

The *tangent space* at $p \in \mathcal{M}$ is $T\mathcal{M}_p = \ker(\nabla F(p))$.

We can assume without any loss of generality that $c = 0$, that is, that the manifold is given by

$$\mathcal{M} = F^{-1}(0) = \{x \in \mathbb{R}^n \mid F(x) = 0\}.$$

Definition 2.3 (Transversality)**.** Let $\mathcal{M}$ and $\mathcal{N}$ be differentiable manifolds of dimensions m and n, respectively, in $\mathbb{R}^k$ with $max\{m, n\} \leq k$. Given $p \in \mathcal{M} \cap \mathcal{N}$ we say that $\mathcal{M}$ and $\mathcal{N}$ are *transverse at p* if $T\mathcal{M}_p \oplus T\mathcal{N}_p = \mathbb{R}^k$, where $T\mathcal{M}_p$ and $T\mathcal{N}_p$ are the tangent spaces to $\mathcal{M}$ and $\mathcal{N}$, respectively, at p. We say that $\mathcal{M}$ and $\mathcal{N}$ are *transverse* if they are transverse at every point $p \in \mathcal{M} \cap \mathcal{N}$ or if $\mathcal{M} \cap \mathcal{N} = \emptyset$.

Definition 2.4 (Simplex)**.** The *simplex* generated by the points $v_0, \ldots, v_m \in \mathbb{R}^n$ is the set

$$\sigma = \left\{ v \in \mathbb{R}^n \mid v = \sum_{i=0}^{m} \lambda_i v_i, \text{ with } \lambda_i \geq 0 \text{ and } \sum_{i=0}^{m} \lambda_i = 1 \right\},$$

and it is denoted by $\sigma = [v_0, \ldots, v_m]$. The *dimension* of σ is defined as $\dim(\sigma) = \dim(\text{span}\{v_1 - v_0, \ldots, v_m - v_0\})$.

A simplex of dimension 0 is also referred to as a vertex and a simplex of dimension a 1 is also referred to as an edge. Notice that an edge may be represented by multiple collinear vertices.

Definition 2.5 (Face)**.** Let $\sigma = [u_0, \ldots, u_{k_1}]$ be a k-simplex and $\tau = [v_0, \ldots, v_{k_2}]$ be an m-simplex with $\{v_0, \ldots, v_{k_2}\} \subset \{u_0, \ldots, u_{k_1}\}$. Then τ is a *m-face* (or simply a *face*) of σ if $\tau = \sigma$ or if the following conditions are satisfied:

1. $m < k$;
2. If η is an m-simplex with $\tau \subset \eta$, then $\eta = \tau$.

If τ is a face of σ with $\tau \neq \sigma$ we say that τ is a *proper face* of σ.

Definition 2.6 (Adjacent Simplices)**.** Two simplices σ_1 and σ_2 are said to be *adjacent* if $\sigma_1 \cap \sigma_2$ is a common face to both σ_1 and σ_2.

Definition 2.7 (Incident Simplices)**.** A simplex τ is said to be *incident* to a simplex σ if τ is a proper face of σ.

Our algorithm produces a collection of topological cells, as described in [9], to represent the computed manifold approximation.

Definition 2.8 (Topological Cells)**.** We define a *topological cell* as follows: A topological cell of dimension 0 or 1 is a convex affine cell of dimension 0 or 1, respectively. A topological cell of dimension k is defined as a list of its vertices and a list of its lower dimensional faces represented as topological cells.

A topological cell of dimension k is referred to as a *k-cell* or simply a *cell*. We also refer to 0-cells and 1-cells as *vertices* and *edges*, respectively.

Definition 2.9 (Hypercube)**.** The set $\mathcal{I}^n = \prod_{i=1}^{n} [a_i, b_i] \subset \mathbb{R}^n$ with $a_i < b_i \in \mathbb{R}$ is called an *n-dimensional hypercube* (or simply a hypercube). The $(n-1)$-*faces* of $\mathcal{I}^n$ are the sets

$$\mathcal{I}_i^{n-1} = \{x \in C \mid x_i = a_i\} \quad \text{and} \quad \mathcal{I}_{n+i}^{n-1} = \{x \in C \mid x_i = b_i\}$$

for each $i = 1, \ldots, n$. The faces of dimension less than $n-1$ can be obtained by the intersection of two or more faces of higher dimension, that is

$\mathcal{I}_{i,j}^{n-2} = \mathcal{I}_i^{n-1} \cap \mathcal{I}_j^{n-1}$, for $j \neq n-i$ (since opposite faces have an empty intersection);

$\mathcal{I}_{i,j,k}^{n-3} = \mathcal{I}_i^{n-1} \cap \mathcal{I}_j^{n-1} \cap \mathcal{I}_k^{n-1}$, for $j \neq n-i, k \neq n-i, k \neq n-j$.

The pattern above continues until the edges ($\mathcal{I}^1$) of the hypercube. Each edge is the intersection of $n-1$ faces of dimension $n-2$.

3. The Generalized Combinatorial Marching Hypercubes Algorithm

This section describes the Generalized Combinatorial Marching Hypercubes algorithm (GCMH). It computes an approximation of an $(n-k)$-dimensional implicit manifold $\mathcal{M} = F^{-1}(0)$, with $F\colon \mathbb{R}^n \to \mathbb{R}^k$. The main difference from the CMH algorithm in [9] is the application of a simplex decomposition process to both find the vertices of the manifold and solve possible ambiguities, as described in the following subsections.

3.1. Algorithm Description

Here, we present the main steps of the GCMH algorithm.

Input. The input of GCMH is:

- A C^1 function $F\colon \mathbb{R}^n \to \mathbb{R}^k$.
- A domain hypercube $D = \prod_{i=1}^{n} [a_i, b_i] \subset \mathbb{R}^n$.

- A grid size tuple $(k_1, \ldots, k_n) \in \mathbb{Z}_+^n$.

We subdivide each interval $[a_i, b_i]$ uniformly into k_i intervals; this way we divide D into a uniform grid of smaller hypercubes. We shall refer to it as the *domain grid*. Each $(n-k)$-cell will be incident to two grid hypercubes or one grid hypercube and the boundary.

Output. The output of GCMH is an approximation $\mathcal{M}_A$ for $\mathcal{M} = F^{-1}(0)$, represented by:

- The coordinates of the vertices of $\mathcal{M}_A$.
- A set of $(n-k)$-cells and their r-faces for $0 \leq r \leq n-k-1$.

Each $(n-k)$-cell in $\mathcal{M}_A$ will be a subset of a hypercube of the domain grid, and the algorithm does not create pairs of adjacent $(n-k)$-cells within the same hypercube. Thus, two adjacent $(n-k)$-cells in $\mathcal{M}_A$ necessarily belong to two different adjacent hypercubes.

Given a hypercube $\mathcal{H}$ of the domain grid we denote by $\mathcal{M}_\mathcal{H}$ the cells of $\mathcal{M}_A$ contained in $\mathcal{H}$, that is, $\mathcal{M}_\mathcal{H} = \mathcal{M}_A \cap \mathcal{H}$.

The main steps of the CMH algorithm are:

1) Add a perturbation to each vertex v_i in the domain grid by adding a random n-dimensional vector ϵ_i to v_i.
2) For each hypercube $\mathcal{H}$ in the domain grid do:
   2.1) Compute the vertices and edges of $\mathcal{M}_\mathcal{H}$ and their incidence and adjacency relations. This computation results into a graph $G_\mathcal{H}$.
   2.2) For each connected component of $G_\mathcal{H}$, create an $(n-k)$-cell of $\mathcal{M}_\mathcal{H}$ and compute its r-faces using combinatorial techniques for $2 \leq r \leq n-k-1$.

We add a perturbation to displace any eventual vertices of $\mathcal{H}$ that are directly on the manifold, which could lead to errors. This step is further explained in [9].

The steps 2.1) and 2.2) of the CMH algorithm are detailed in Sections 3.1.1 and 3.1.2, respectively.

3.1.1. Vertices and Edges Computation

Vertices of the approximation $\mathcal{M}_A$ are computed using the simplex decomposition algorithm $\mathcal{CFK}$ [34, 35, 36] applied to each k-face. It is worth noting that, while this calculation is carried out with a simplex decomposition algorithm, information about the simplices is not needed for the combinatorial skeleton, which greatly reduces processing time after the computation of vertices and edges.

For a given hypercube $\mathcal{H}$ of the domain grid, we break each of its k-faces f_k into $k!$ simplices of dimension k, using the $\mathcal{CFK}$ triangulation. Then, a vertex of $\mathcal{M}_\mathcal{H} = \mathcal{M}_A \cap \mathcal{H}$ may be created in each of those k-simplices by computing the linear interpolation of the vertices v_i incident to the given k-simplex, weighted by $F(v_i)$; that is, $v = \sum_{i=0}^{k} \lambda_i v_i$, where λ_i, $i = 1, ..., k$, satisfy

$$\sum_{i=0}^{k} \lambda_i F(v_i) = 0, \quad \sum_{i=0}^{k} \lambda_i = 1.$$

If all $\lambda_i \geq 0$, a new vertex of $\mathcal{M}_\mathcal{H}$ will be created in that k-simplex; otherwise, no vertex is created.

To compute the edges of $\mathcal{M}_\mathcal{H}$, the algorithm loops through each $(k+1)$-face f_{k+1} connecting the manifold vertices created on the k-faces incident to f_{k+1}. For each f_{k+1}, three cases may occur:

- No vertices are found: The algorithm does not create any vertex or edge on this face.
- Two vertices are found: The algorithm simply connects both with an edge of $\mathcal{M}_\mathcal{H}$.
- More than two vertices are found: In this case there is more than one way of connecting the vertices with edges, leading to an ambiguity.

These ambiguities are solved with an algorithm similar to the Marching Simplex (see Figure 2) and is illustrated in Figure 3. Given a $(k+1)$-face with more than two vertices (Figure 3a), we break f_{k+1} into $(k+1)$-simplices (Figure 3b). For all k-simplices incident to each $(k+1)$-simplex we calculate the manifold vertices using the linear interpolation described above (Figure 3c). This recalculates the vertices in the k-faces incident to f_{k+1}, as

well as manifold vertices in the interior of f_{k+1}. Then a manifold vertex in a k-face is chosen, and successively connected with the vertices in the interior of f_{k+1} until it connects with another vertex of $\mathcal{M}_{\mathcal{H}}$ in a k-face (Figure 3d). Those two vertices are connected by a single edge of $\mathcal{M}_{\mathcal{H}}$, and a new vertex in a k-face is chosen again. This process continues until all vertices of $\mathcal{M}_{\mathcal{H}}$ in f_{k+1} are exhausted (Figure 3e). After the disambiguation process, only the edges in $\mathcal{M}_{\mathcal{H}}$ are saved; all other data from the internal $(k+1)$-simplices is discarded.

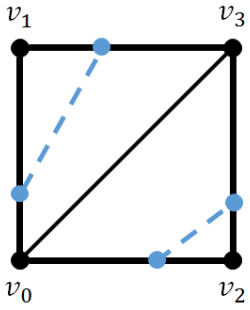


Fig. 2: Disambiguation for a 2-face using the $\mathcal{CFK}$ triangulation

3.1.2. *Combinatorial Skeleton*

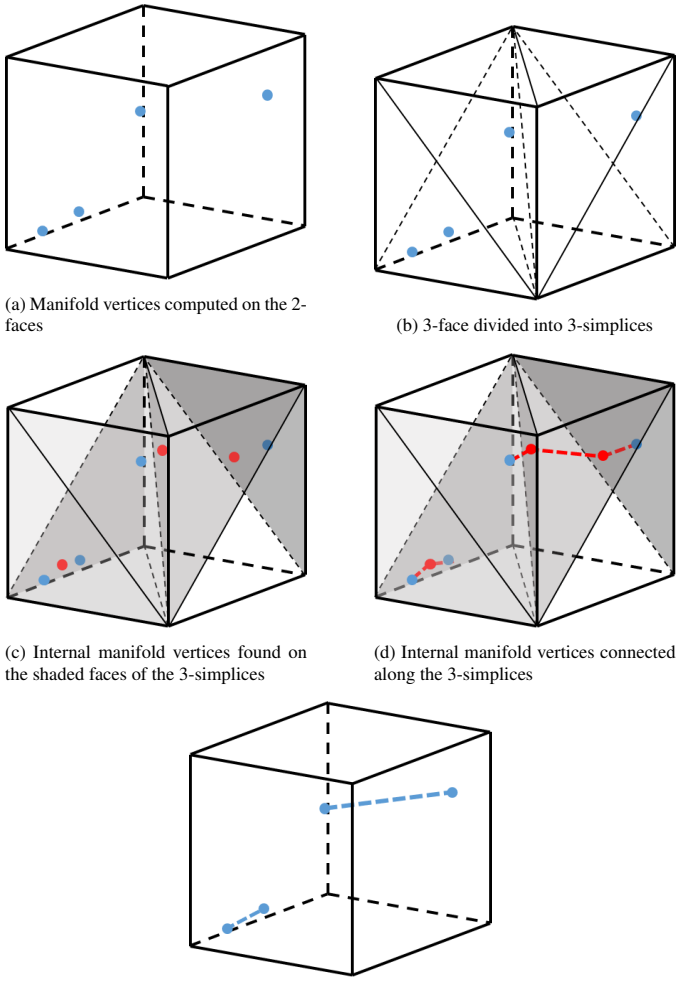
Here we summarize the main definitions and ideas for the construction of the *Combinatorial Skeleton*, which is a structure used to compute higher-dimensional faces of the approximation $\mathcal{M}_A$. Further details can be found in [9].

Definition 3.1 (Face labeling)**.** We represent the faces $\mathcal{I}_i^{n-1}$ and $\mathcal{I}_{n+i}^{n-1}$ of the n-dimensional hypercube $\mathcal{I}^n$ by the integers i and $n+i$, that is, we label $\ell(\mathcal{I}_i^{n-1}) = i$ and $\ell(\mathcal{I}_{n+i}^{n-1}) = n+i$.

The faces of dimensions lower than $n-1$ are labeled accordingly as $\ell(\mathcal{I}_{i,j}^{n-2}) = \{i, j\}$, $\ell(\mathcal{I}_{i,j,k}^{n-3}) = \{i, j, k\}$, etc. The pattern continues until the edges $\mathcal{I}^1$ of $\mathcal{I}^n$.

Due to the construction of the r-faces and the properties of hypercubes, one can prove the following [9]:

- Each $(n-k)$-cell C in $\mathcal{M}_{\mathcal{H}}$ is indeed a cell as defined in Definition 2.8;
- We can represent the incidence relations of C using the labels of $\mathcal{H}$ by labeling each r-face of C with the labels of the corresponding $(r+1)$-face of $\mathcal{H}$.



(a) Manifold vertices computed on the 2-faces

(b) 3-face divided into 3-simplices

(c) Internal manifold vertices found on the shaded faces of the 3-simplices

(d) Internal manifold vertices connected along the 3-simplices

(e) Final manifold edges created and stored in $\mathcal{M}_{\mathcal{H}}$

Fig. 3: Disambiguation process in a 3-face using the $\mathcal{CFK}$ triangulation

These facts are used to build the combinatorial skeleton. Let $\mathcal{L}_r$ be the set of labels of the r-faces of C, for $r = 0, \ldots, n-2$, and set $\mathcal{L} = \bigcup_{r=0}^{n-2} \mathcal{L}_r$. We also define V as the set of manifold vertices of $\mathcal{M}_{\mathcal{H}}$.

The combinatorial skeleton is built as follows: The vertices and edges of $\mathcal{M}_{\mathcal{H}}$ obtained from Section 3.1.1 form a graph G, with possibly more than one connected component. For each component G' of G, we apply the following algorithm:

function CombinatorialSkeleton(n, V, C, $\mathcal{H}$, $\mathcal{L}$)
  for all vertices $v \in V$ of G' on a k-face f of $\mathcal{H}$ **do**
    add v to C
    $\ell(v) \leftarrow \ell(e)$
    add $\ell(v)$ to $\mathcal{L}_0$
  for r $\leftarrow$ 2 to $n-2$ **do**
    for all $(r+1)$-faces $f_{\mathcal{H}}$ of $\mathcal{H}$ **do**
      if $\exists \ell(x) \in \mathcal{L}_{r-1}$ such that $\ell(f_{\mathcal{H}}) \subset \ell(x)$ **then**
        create a new r-face f_C
        add f_C to C
        $\ell(f_C) \leftarrow \ell(f_{\mathcal{H}})$
        add $\ell(f_C)$ to $\mathcal{L}_r$
        for all $(r-1)$-faces f'_C of C **do**
          if $\ell(f_C) \subset \ell(f'_C)$ **then**
            Add f'_C as a face of f_C in C
  return C

3.2. Consistency and Correctness

This section addresses the consistency of the manifold approximation $\mathcal{M}_A$ generated by the GCMH algorithm, as described below.

Definition 3.2 (Consistency)**.** Let $\mathcal{H}_1$ and $\mathcal{H}_2$ be two adjacent hypercubes of the domain grid that share a common $(n-1)$-face f and let $\mathcal{M}_f = \mathcal{M}_{\mathcal{H}_1} \cap \mathcal{M}_{\mathcal{H}_2}$. We say that $\mathcal{M}_f$ is *consistent* if $\mathcal{M}_{\mathcal{H}_1} \cap f = \mathcal{M}_{\mathcal{H}_2} \cap f$. In this case we also say that each cell of $\mathcal{M}_f$ is consistent.

In other words, the approximation is consistent if it generates the same vertices, edges, and higher-dimensional cells on shared $(n-1)$-faces of adjacent hypercubes. In order for the algorithm to be correct, it is sufficient that the approximation is consistent for all pairs of adjacent hypercubes in the domain grid.

Since the construction of the combinatorial skeleton is the same as in [9], the proofs of the consistency results can mostly be adapted from there with minor changes. The only proof which needs a more significant modification is vertex consistency because the vertex generation uses a different process. Thus, for higher-dimensional consistency, we will only state those results and refer to [9] for details of their proofs.

Lemma 3.1 (Vertex consistency)**.** *The vertices generated by the CMH algorithm on $\mathcal{M}_{\mathcal{H}_1} \cap f$ and $\mathcal{M}_{\mathcal{H}_2} \cap f$ are the same.*

Proof. Since the vertices of the $(n-1)$-face f are common to $\mathcal{H}_1$ and $\mathcal{H}_2$, the values of the function F at the vertices of f are the same on $\mathcal{H}_1$ and $\mathcal{H}_2$, that is, when f is viewed as a face of $\mathcal{H}_1$ and as a face of $\mathcal{H}_2$. Moreover, the consistency of the $\mathcal{CFK}$ triangulation ensures that the simplices which $\mathcal{M}_{\mathcal{H}_1} \cap f$ and $\mathcal{M}_{\mathcal{H}_2} \cap f$ are decomposed in having the same vertices. Therefore the vertices of $\mathcal{M}_f$, which are obtained by linear interpolation on the values of F at the vertices of the simplices in f, are the same on $\mathcal{M}_{\mathcal{H}_1}$ and $\mathcal{M}_{\mathcal{H}_2}$. □

Lemma 3.2 (Edge consistency)**.** *The edges generated by the GCMH algorithm on $\mathcal{M}_{\mathcal{H}_1} \cap f$ and $\mathcal{M}_{\mathcal{H}_2} \cap f$ are the same.*

Lemma 3.3 (Higher dimensional consistency)**.** *The k-cells generated by the GCMH algorithm on $\mathcal{M}_{\mathcal{H}_1} \cap f$ and $\mathcal{M}_{\mathcal{H}_2} \cap f$ are the same for $2 \le r \le n-2$.*

The lemmas above prove the following theorem:

Theorem 3.4 (GCMH algorithm consistency)**.** *The set of cells $\mathcal{M}_f = \mathcal{M}_{\mathcal{H}_1} \cap \mathcal{M}_{\mathcal{H}_2}$ generated by the GCMH algorithm is consistent for any pair $\mathcal{H}_1$ and $\mathcal{H}_2$ of adjacent hypercubes of the domain grid.*

3.3. Combinatorial methods for grid generation

In order to represent hypercubes and their vertices and generate grid elements, we use combinatorial techniques based on the enumeration of discrete Cartesian products. Those methods are presented in [9] with almost no modifications; we, therefore, refer to [9] for details on these combinatorial methods.

4. Comparison to Combinatorial Marching Simplex

In order to study the performance of the GCMH algorithm, we implemented the Combinatorial Marching Simplex (CMS) method described in [9]. This method is a triangulation method that also computes an approximation $\mathcal{M}_A$ of a manifold implicitly defined by $\mathcal{M} = F^{-1}(0)$, with $F\colon \mathbb{R}^n \to \mathbb{R}^k$, and also uses a combinatorial structure. Thus, we will be able to compare the performance of our method with a simplicial algorithm.

The CMS algorithm is broadly described as follows:

1. Subdivide a regular and finite grid of hypercubes of a compact subspace of $\mathbb{R}^n$ into simplices using the $\mathcal{CFK}$ triangulation. The resulting subdivision is the domain grid.
2. For each simplex $\mathcal{S}$ of each hypercube $\mathcal{H}$ in the domain do:
   - 2..1 Compute the vertices of $\mathcal{M}_{\mathcal{S}} = \mathcal{S} \cap \mathcal{M}_A$ that are on the edges of $\mathcal{S}$. This is done with linear interpolation of the vertices of each face of the simplex, similar to the computation of vertices in the GCMH algorithm (Section 3.1.1).
   - 2..2 Define the edges of $\mathcal{M}_{\mathcal{S}}$ that are on the 2-faces of $\mathcal{S}$. Contrary to GCMH there is an unique way to this.
   - 2..3 Build the Combinatorial Skeleton to get the k-faces of $\mathcal{M}_{\mathcal{S}}$ for $k > 2$. The details of this step are analogous to GCMH.

It is worth remembering that, unlike the GCMH algorithm, CMS does require that the data from every simplex is stored in order to build the approximation $\mathcal{M}_A$. Since the $\mathcal{CFK}$ triangulation generates $k!$ simplices for every k-dimensional hypercube and the algorithm generates a comparable number of vertices, this leads to an increase in the number and in the processing time of the edges and higher-dimensional cells of $\mathcal{M}_A$. This will be made clearer in Section 6.

5. The Generalized Combinatorial Continuation Hypercube extension

Similar to [9], here we propose an extension to the GCMH algorithm called the Generalized Combinatorial Continuation Hypercube (GCCH), mainly based on the work of Allgower and Georg [37]. The aim is to improve the time efficiency of the GCMH algorithm by checking for vertices only in the hypercubes adjacent to the ones that are transversal to the manifold and therefore have a chance to be transversal themselves.

5.1. Starting Point

Given a user-input starting point x_0, we first find a starting hypercube that contains it. Since the hypercubes are defined by intervals along each dimension, the starting hypercube is found by comparing the coordinates of x_0 with those intervals.

5.2. List of Hypercubes

After finding the initial hypercube $\mathcal{H}_0$ containing the initial point x_0, we put all of its adjacent hypercubes into a list L_{tbp} of hypercubes to be processed, and we process $\mathcal{H}_0$ using the GCMH algorithm. We then create a list L_p of already processed hypercubes, and add $\mathcal{H}_0$ to it. After that, the algorithm proceeds as follows:

While L_{tbp} is not empty do:

1. Remove an hypercube $\mathcal{H}_i$ from L_{tbp}.
2. Find all hypercubes adjacent to $\mathcal{H}_i$ that are not in L_p and add them to L_{tbp}.
3. Process $\mathcal{H}_i$ like in the GCMH algorithm.
4. Add $\mathcal{H}_i$ to L_p.

When $\mathcal{M}$ has more than one connected component, some components might not be computed by the steps above. In order to reach those components, the user can execute the algorithm again with another starting point on a different connected component.

6. Results

This section describes some experimental results to show the empirical effectiveness and efficiency of the GCMH algorithm and the GCCH extension.

We used a computer with a 2.3 GHz 8-Core Intel Core i9 Processor and 64 GB 2667 MHz DDR4 memory, and the software Matlab 2021b for all the computations.

In order to show the efficiency of the GCMH and GCCH we ran an experiment based on the complex cosine function: $\mathcal{M} = F^{-1}(0)$, where $F : \mathbb{C}^2 \to \mathbb{C}^1$ is given by $F(z, w) = z - \cos(w)$. For the purpose of our algorithm, F is treated as a function from $\mathbb{R}^4$ to $\mathbb{R}^2$. We compared the results using GCMH, GCCH, CMS, and CCS. The experiment shows that (see discussion in Section 6.1):

- The processing time of GCMH is significantly lower than the processing time of CMS. The same applies to GCCH when compared to CCS.

div	CMS	GCMH	CCS	GCCH
5	3.83	2.16	0.69	0.54
10	55.54	34.20	2.25	1.54
20	2048.63	589.38	8.34	5.54
40	12362.45	7927.68	38.30	22.35
80			236.79	87.22
160			2837.64	396.12
320			41355.25	1912.98

Table 1: The processing time, in seconds, of CMS, GCMH, CCS, GCCH to approximate the complex cosine function $z = \cos(w)$ for each number *div* of divisions per dimension.

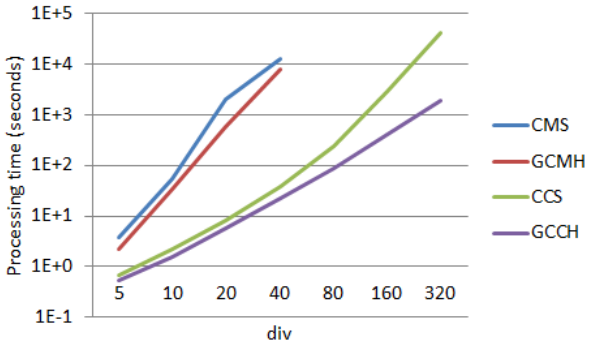


Fig. 4: The processing time, in logarithmic scale, of CMS, GCMH, CCS and GCMH to approximate the complex cosine function $z = \cos(w)$ as a function of the number *div* of subdivisions.

- The continuation method significantly increases the processing time of GCCH and CCS.
- Although the output size of GCCH is smaller than the output size of CCS, the output approximation is smoother when using GCCH.

A second experiment was made to test the efficiency of GCMH and GCCH in higher dimensions. For that purpose we used the 3-torus (or 3D torus) embedded in $\mathbb{R}^6$, given by $\mathcal{M} = S^1 \times S^1 \times S^1$. More specifically, $F(x) = 0$ where $F\colon \mathbb{R}^6 \to \mathbb{R}^3$ is given by

$$F(x_1, x_2, x_3, x_4, x_5, x_6) = \begin{pmatrix} x_1^2 + x_2^2 - \frac{1}{4} \\ x_3^2 + x_4^2 - 1 \\ x_5^2 + x_6^2 - 4 \end{pmatrix}.$$

6.1. Comparing CMS, GCMH, CCS and GCCH

In order to show the advantages of the GCMH algorithm over the CMS and the advantages of the continuation method for both GCCH and CCS, we measured the processing time needed to approximate the complex cosine function. For each run the number of hypercubes of the domain grid is set by a number *div* of divisions per dimension, that is, the total number of hypercubes is div^4.

The results can be seen on Table 1 and Figure 4. As expected, the continuation method significantly improves both algorithms. Also, the processing time using the GCMH algorithm is significantly lower than the CMS.

We also measured the output size, as the number of $(n-1)$-cells, of GCCH and CCS. Note that the continuation method does not change the output of the approximation when compared to GCMH and CMS. The results can be seen on Table 2. As expected, the output of GCCH is smaller.

div	CCS	GCCH
5	1162	148
10	4378	612
20	17258	2112
40	68354	8336

Table 2: The output size (number of $(n-1)$-cells) of the approximation of $z = \cos(w)$ using CCS and GCCH.

Figures 5(a) and 5(b) show projections of the approximation of the complex cosine function using GCCH and CCS, respectively. Figures 5(c) and 5(d) show a zoom in of a portion of both projections. Although the output using GCCH is less dense, it is smoother. We indicate the smoothness of the results in Figures 5(c) and 5(d) by a series of red segments. The consecutive red segments represent "curve breaks". A higher density of curve breaks indicates better smoothness.

6.2. Results in higher dimension

In order to show the effectiveness of GCMH and GCCH in higher dimensions, we measured the processing time needed to approximate the 3-torus embedded in $\mathbb{R}^6$ (see Figure 7). For each computation, the number of hypercubes of the domain grid was set by a number *div* of divisions per dimension, that is, the total number of hypercubes is given by div^6. The results can be seen on Table 3 and Figure 6.

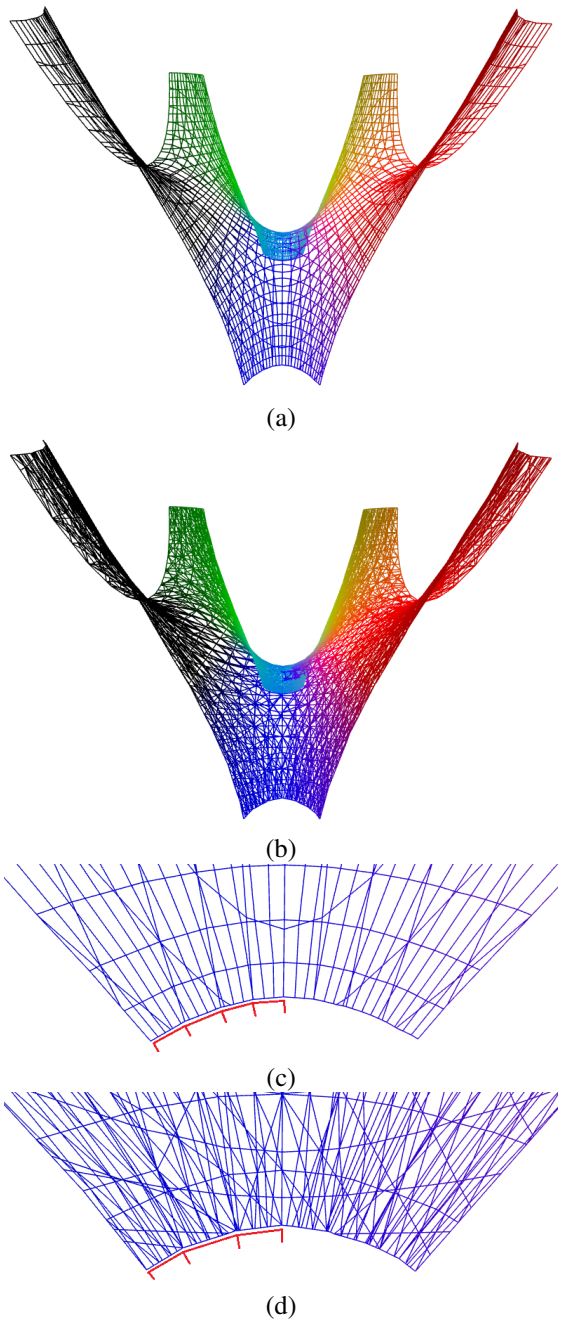


Fig. 5: Projections of the complex cosine function $z = cos(w)$ using (a) GCCH with $div = 40$ and (b) CCS with $div = 20$. In (c) and (d) we show a zoom in of a portion of the manifolds in (a) and (b), respectively.

div	GCMH	GCCH	*div*	GCMH	GCCH
2	93.61	93.07	5	22686.60	2946.30
3	1055.67	734.47	6	67749.18	7126.96
4	5910.59	846.46	7	172271.80	13545.33

Table 3: The processing time, in seconds, of GCMH and GCCH to approximate the 3-torus for each number *div* of divisions per dimension.

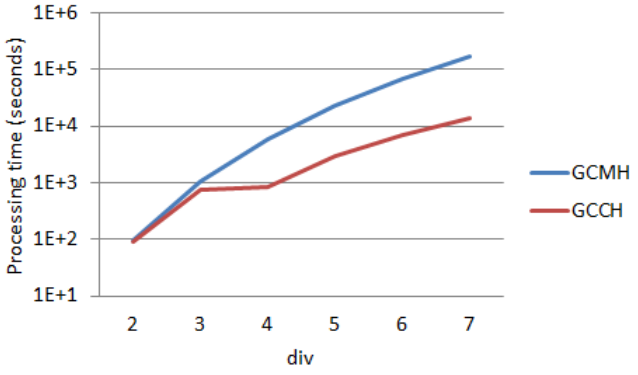


Fig. 6: The processing time of GCMH and GCMH to approximate the 3-torus, in logarithmic scale.

7. Final remarks

This paper proposes a generalization of the Combinatorial Marching Hypercubes algorithm [9] that extends the renowned Marching Cubes algorithm to approximate manifolds of any dimension. In [9] the manifold is required to have co-dimension 1 while in the generalization proposed in this paper this requirement is eliminated.

The proposed implementation does not rely on lookup tables or other expensive memory structure that grows exponentially with dimension. Our algorithm uses combinatorial and topological techniques to build the manifold approximation by a cell complex. Each hypercube can be processed independently, making the proposed algorithm highly parallelizable.

We show the effectiveness of our method in higher dimensions by approximating classical manifolds from the literature and we also show that the algorithm outperforms a similar algorithm based on a simplicial decomposition of the domain grid.

References

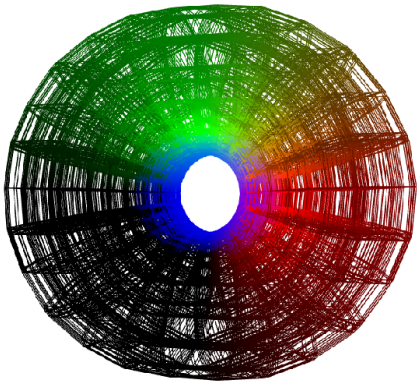
[1] Botsch, M, Kobbelt, L, Pauly, M, Alliez, P, Lévy, B. Polygon mesh processing. CRC press; 2010.

[2] Bloomenthal, J, Bajaj, C, Blinn, J, Wyvill, B, Cani, MP, Rockwood, A, et al. Introduction to implicit surfaces. Morgan Kaufmann; 1997.

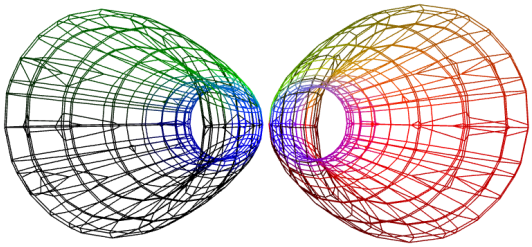
[3] Gomes, A, Voiculescu, I, Jorge, J, Wyvill, B, Galbraith, C. Implicit curves and surfaces: mathematics, data structures and algorithms. Springer Science & Business Media; 2009.

[4] Markl, M, Frydrychowicz, A, Kozerke, S, Hope, M, Wieben, O. 4d flow mri. Journal of Magnetic Resonance Imaging 2012;36(5):1015–1036.

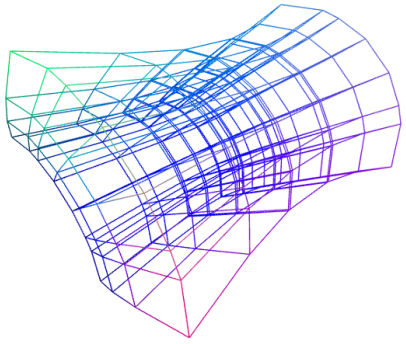
[5] Pan, T, Lee, TY, Rietzel, E, Chen, GT. 4d-ct imaging of a volume influenced by respiratory motion on multi-slice ct. Medical physics 2004;31(2):333–340.



(a) A projection of $\mathcal{M}$ with $div = 8$.



(b) A projection of $\mathcal{M} \cap \{x_6 = 0\}$ with $div = 8$



(c) A projection of $\mathcal{M} \cap \mathbb{R}^6_+$ with $div = 5$

Fig. 7: Projections of the approximations computed by the GCCH algorithm of the 3-torus $\mathcal{M} = S^1 \times S^1 \times S^1$ embedded in $\mathbb{R}^6$.

[6] Gonçalves, LF, Lee, W, Chaiworapongsa, T, Espinoza, J, Schoen, ML, Falkensammer, P, et al. Four-dimensional ultrasonography of the fetal heart with spatiotemporal image correlation. American journal of obstetrics and gynecology 2003;189(6):1792–1802.

[7] Sharf, A, Alcantara, DA, Lewiner, T, Greif, C, Sheffer, A, Amenta, N, et al. Space-time surface reconstruction using incompressible flow. ACM Transactions on Graphics (TOG) 2008;27(5):1–10.

[8] Choquet-Bruhat, Y, DeWitt-Morette, C, de Witt, C, Bleick, MD, Dillard-Bleick, M. Analysis. Gulf Professional Publishing; 1982.

[9] Castelo, A, Moutinho Bueno, L, Gameiro, M. A combinatorial marching hypercubes algorithm. Computers & Graphics 2022;102:67–77. URL: https://www.sciencedirect.com/science/article/pii/S0097849321002405. doi:https://doi.org/10.1016/j.cag.2021.10.023.

[10] Lorensen, WE, Cline, HE. Marching cubes: A high resolution 3d surface construction algorithm. ACM siggraph computer graphics 1987;21(4):163–169.

[11] Nielson, GM, Hamann, B. The asymptotic decider: resolving the ambiguity in marching cubes. In: Proceedings of the 2nd conference on Visualization'91. IEEE Computer Society Press; 1991, p. 83–91.

[12] Chernyaev, E. Marching cubes 33: Construction of topologically correct isosurfaces. Tech. Rep.; 1995.

[13] Custodio, L, Etiene, T, Pesco, S, Silva, C. Practical considerations on marching cubes 33 topological correctness. Computers & graphics 2013;37(7):840–850.

[14] Lewiner, T, Lopes, H, Vieira, AW, Tavares, G. Efficient implementation of marching cubes' cases with topological guarantees. Journal of graphics tools 2003;8(2):1–15.

[15] Plantinga, S, Vegter, G. Isotopic approximation of implicit curves and surfaces. In: Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing. 2004, p. 245–254.

[16] Dyken, C, Ziegler, G, Theobalt, C, Seidel, HP. High-speed marching cubes using histopyramids. In: Computer Graphics Forum; vol. 27. Blackwell Publishing Ltd Oxford, UK; 2008, p. 2028–2039.

[17] Dietrich, CA, Scheidegger, CE, Schreiner, J, Comba, JL, Nedel, LP, Silva, CT. Edge transformations for improving mesh quality of marching cubes. IEEE Transactions on Visualization and Computer Graphics 2008;15(1):150–159.

[18] Liao, Y, Donne, S, Geiger, A. Deep marching cubes: Learning explicit surface representations. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2018, p. 2916–2925.

[19] Raman, S, Wenger, R. Quality isosurface mesh generation using an extended marching cubes lookup table. In: Computer Graphics Forum; vol. 27. Wiley Online Library; 2008, p. 791–798.

[20] Newman, TS, Yi, H. A survey of the marching cubes algorithm. Computers & Graphics 2006;30(5):854–879.

[21] Scarf, H. The approximation of fixed points of a continuous mapping. SIAM Journal on Applied Mathematics 1967;15(5):1328–1343.

[22] Doi, A, Koide, A. An efficient method of triangulating equi-valued surfaces by using tetrahedral cells. IEICE TRANSACTIONS on Information and Systems 1991;74(1):214–224.

[23] Guéziec, A, Hummel, R. Exploiting triangulated surface extraction using tetrahedral decomposition. IEEE Transactions on visualization and computer graphics 1995;1(4):328–342.

[24] Treece, GM, Prager, RW, Gee, AH. Regularised marching tetrahedra: improved iso-surface extraction. Computers & Graphics 1999;23(4):583–598.

[25] Weigle, C, Banks, DC. Extracting iso-valued features in 4-dimensional scalar fields. In: IEEE Symposium on Volume Visualization (Cat. No. 989EX300). IEEE; 1998, p. 103–110.

[26] Roberts, JC, Hill, S. Piecewise linear hypersurfaces using the marching cubes algorithm. In: Visual Data Exploration and Analysis VI; vol. 3643. International Society for Optics and Photonics; 1999, p. 170–181.

[27] Onosato, M, Saito, Y, Tanaka, F, Kawagishi, R. Weaving a four-dimensional mesh model from a series of three-dimensional voxel models. Computer-Aided Design and Applications 2014;11(6):649–658.

[28] Bhaniramka, P, Wenger, R, Crawfis, R. Isosurfacing in higher dimensions. In: Proceedings Visualization 2000. VIS 2000 (Cat. No. 00CH37145). IEEE; 2000, p. 267–273.

[29] Bhaniramka, P, Wenger, R, Crawfis, R. Isosurface construction in any dimension using convex hulls. IEEE Transactions on Visualization and Computer Graphics 2004;10(2):130–141.

[30] Min, C. Simplicial isosurfacing in arbitrary dimension and codimension. Journal of Computational Physics 2003;190(1):295–310.

[31] Castelo, A, Nonato, LG, Siqueira, MF, Minghim, R, Tavares, G. The j1a triangulation: An adaptive triangulation in any dimension. Computers & Graphics 2006;30(5):737–753.

[32] Schwaha, P, Heinzl, R. Marching simplices. In: AIP Conference Proceedings; vol. 1281. American Institute of Physics; 2010, p. 1651–1654.

[33] Brodzik, M. The computation of simplicial approximations of implicitly defined p-dimensional manifolds. Computers & Mathematics with Applications 1998;36(6):93–113.

[34] COXETER, HSM. Discrete groups generated by reflections. Annals of Mathematics 1934;35:588–621.

[35] FREUDENTHAL, H. Simplizalzerlegungen von beschränkter flachheit. Annals of Mathematics 1942;43:580–582.

[36] KUHN, HW. Simplicial approximation of fixed points. Proceedings of National Academy of Sciences of United States of America 1968;61:1238–1242.

[37] Allgower, E, Georg, K. Simplicial and continuation methods for approximating fixed points and solutions to systems of equations. Siam review 1980;22(1):28–85.

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

A generalized combinatorial marching hypercubes algorithm

Antonio Castelo[a,1], Guilherme Nakassima[a,4], Lucas Moutinho Bueno[a,3], Marcio Gameiro[a,b,1]

[a] *Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, São Paulo, Brazil*
[b] *Department of Mathematics, Rutgers, The State University of New Jersey, Piscataway, NJ, 08854, USA*

ARTICLE INFO

Article history:
Received May 9, 2022

Manifold approximation, High-dimensional manifolds, Marching Hypercubes, Combinatorial skeleton
2020 MSC: 68U05

ABSTRACT

We present a Generalized Combinatorial Marching Hypercubes (GCMH) algorithm to compute a cell complex approximation of a manifolds of any dimension and co-dimension, that is, manifolds of dimension $n-k$ embedded into an n-dimensional space. The algorithm uses combinatorial and topological methods to avoid the use of expensive lookup tables and hence be efficient in higher dimensions. We illustrate the effectiveness of our algorithm in higher dimensions and compare its performance with a similar algorithm based on a simplicial decomposition of the domain.

1. Introduction

Methods to compute approximations of two- and three-dimensional manifolds (surfaces and volumes) are widely available and used in computer graphics and applied to animation, digital games, molecular modeling, object detection, object tracking, medical imaging, object reconstruction, etc. (see [1, 2, 3, 4, 5, 6, 7]).

Higher-dimensional manifolds are less prevalent, but they are also used in applications such as superstring theory [8]. Methods to compute such manifolds are also available; however, most of these methods are highly inefficient in high dimensions. An efficient algorithm, called Combinatorial Marching Hypercubes (CMH), to compute high dimensional manifolds of dimension $n-1$ embedded into a n-dimensional space was presented in [9].

In this paper, we present a generalization of the method presented in [9] to compute $(n-k)$-dimensional manifolds embedded into a n-dimensional space. More precisely, we present the Generalized Combinatorial Marching Hypercubes (GCMH) algorithm to compute and represent implicitly defined $(n-k)$-dimensional manifolds as a cell complex embedded into a grid of n-dimensional hypercubes. Our implementation of the algorithm can be found at `https://github.com/gknakassima/GCMH`.

An implicitly defined $(n-k)$-dimensional manifold $\mathcal{M}$ is the set of points where a function $F\colon \mathbb{R}^n \to \mathbb{R}^k$ takes on a given value c (see Section 2 for a precise definition). Figure 1 shows an example of the output of our algorithm to represent a Bagel Klein Bottle defined by $F(x, y, z, s, t) = (0, 0, 0)$, where

*Corresponding author.
[1] castelo@icmc.usp.br
[2] guilherme.nakassima@usp.br
[3] lucasmb86@gmail.com
[4] gameiro@icmc.usp.br