A Comparative Analysis of Iso-surface Generation Techniques for 3D Scalar Field Visualization

Shubhashree Sahoo
Department of Computer Science and Engineering
Teegala Krishna Reddy Engineering College, India

Abstract:
A simple and efficient method for the extraction of high-quality continuous isosurfaces from the volumetric datasets is implemented and tested. The present method proceeds in two steps. In the first step, a continuous interpolant of the dataset were determined and second step involves the extraction of isosurface geometry by sampling the points on Marching Cubes triangles and projecting them onto the isosurface defined by the interpolant. The algorithms are studied and implemented using Matlab. The results using a synthetic datasets and discussion of practical considerations are presented. The importance of present method is that the proposed implementation could be applied to any arbitrary data.

Keywords: Volumetric visualization, Iso-surface generation, Marching Cubes, MATLAB environment.

1. INTRODUCTION

Recent progressment in computing hardware has eased the job of researchers and practitioners to model and simulate a variety of physical phenomena in a faster and efficient manner [1]. The output of such simulations and measurements is a dataset, which is massive in size and complex in nature. The eminent challenge is extraction of essential information from such a huge dataset. Scientific data visualization offers techniques and tools to gain information and insight from dataset by transforming them into visuals suitable for human comprehension [2]. Scalar visualization techniques furnish algorithms suitable to process and visualize the scalar data. Isosurfaces are the basic tools in data visualization that extracts absolute surfaces representing the features of a scalar field. Isosurfaces are commonly displayed using computer graphics. Iso-surface generation is a technique to visualize the 3D volume data using intermediate polygonal based rendering [3]. It mainly contrives a surface made up of tiny triangles to narrate the areas having similar or constant properties. There are three cell based high-resolution 3D surface construction isosurface algorithms namely Marching Cubes, Dividing Cubes and Dual Contouring. Among these algorithms, the Marching Cubes algorithm is popular one. The Marching Cubes algorithms generate polygonal representation of surfaces having constant properties from 3D volume data [4]. A facile method for the isosurface construction from 3D volume dataset has been adopted. The continuous interpolation scheme has been defined and isosurface generation process is addressed by first computing the triangles of a Marching Cubes surface. Inside the generated triangles, the points of interest are sampled and projected onto the necessary isosurface using continuous interpolant and surface rendering method principles. A software framework for Marching Cubes has also been designed and developed for isosurface construction in MATLAB environment. The present method requires only a moderated implementation effort and could be easily integrated into existing applications.

2. RELATED WORK

There are assorted approaches to the difficulty of 3D isosurface generation [5]. One of the elderly approaches involves construction of surface contours and its interconnection. However, presence of more than one contour in a slice causes ambiguity. The Mayo Clinic has used a different approach that displays the density volume rather than the surface and produced a conventional shadow graph that could be viewed from arbitrary angles. These surface construction techniques are discarded due to their inefficiency in extracting essential information from the original data. Lorensen and Cline introduced the Marching Cubes algorithm for the isosurface visualization and the approach is greatly popularized [4]. Authors have used a case-driven approach that involved triangle generation to approximate the isosurface of interest. The relative simplicity and elegance of Marching Cubes algorithms make it a desirable technique for applying to multifarious datasets [6]. There are innumerable variations and improvements of the original Marching Cubes algorithms. Himish Carr has used Dividing Cubes algorithm that eliminates the scan conversion step used when rendering surfaces extracted by the Marching Cubes algorithms. The Dual Contouring method is proposed by Ju et. al. and provides a uniform approach for extracting isosurfaces. The major advantages include simplicity to implement and production of sharp features, which is not realized by other implicit surface meshing algorithms.

3. ISOSURFACE GENERATION

Isosurface generation is a technique that visualizes the 3D volume datasets using intermediate polygonal based representation. It principally constructs a surface made up of tiny triangles to narrate areas having similar or constant properties. Isosurfaces are rendered by a simple polygonal model that could be drawn on the screen very rapidly [7]. Isosurfaces are used to extract essential information from
volumetric datasets obtained in the field of medical imaging, pharmacology chemistry, geophysics, computational geometry and meteorology. There are three well known cell based iso-

surface generating algorithms such as Marching Cubes, Dividing Cubes and Dual Contouring [3].

3.1 Marching Cubes Algorithm
Marching Cubes is a rudimentary algorithm in the field of isosurface polygonization and widely accepted by the visualization community for its simplicity and robustness. The results from Magnetic Resonance (MC), Computed Tomography (CT) and Single-Photon Emission Computed Tomography (SPECT) had proven the quality and functionality of Marching Cubes [8]. These algorithms employ a divide and conquer approach that involves splitting of the volume into a grid of regular cells. Marching Cubes algorithm operates on two basic steps such as detection of active cells (cells intersected by the isosurface) and provoking the triangles inside each active cell. The algorithm determines the way that surface intersects the cube and subsequently moves or marches to the next cube [9].

To find the surface intersection in a cube succeeding steps has to be followed.

Step 1: Cube’s vertex is assigned 1, when data value at the vertex exceeds the value of surface that is being constructed and such vertices are inside the surface.

Step 2: Cube’s vertex is assigned 0, when data value falls behind the surface value and such vertices are outside the surface.

Step 3: When one vertex is inside the surface (one) and the other outside the surface (zero), then the surface intersects the cube edges.

Each cube contains eight vertices and two states such as inside and outside. Therefore, there are 2^8 = 256 ways that a surface could intersect the cube. The two essential symmetries of the cube reduce the problem from 256 cases to 14 unique patterns [4].

Step 4: The intersecting edge could be found by interpolating the surface intersection along the edge by using Linear Interpolation method.

Step 5: The gradient vector is the derivative of density function and is given by

$$\nabla f(x, y, z) = \frac{\partial}{\partial x} f(x, y, z)$$

Step 6: The gradient vectors at the surface of interest could be estimated by determining the gradient vectors at cube vertices and linearly interpolating the gradient at the point of intersection.

The gradient vector at cube vertex (i, j, k) could be estimated using central differences along the three coordinate axes by:

$$Gx(i, j, k) = \frac{D[i+1, j, k] - D[i-1, j, k]}{\Delta x}$$

$$Gy(i, j, k) = \frac{D[i, j+1, k] - D[i, j-1, k]}{\Delta y}$$

$$Gz(i, j, k) = \frac{D[i, j, k+1] - D[i, j, k-1]}{\Delta z}$$

Step 7: Finally the Marching Cubes algorithm calculates a unit normal for each triangle vertex that renders the image [10, 11].

3.2 Dividing Cubes Algorithm
Dividing Cubes algorithm was developed to eliminate scan conversion step of conventional polygon based display algorithm. The basic principle involves creation of surface points instead of triangles, association of surface normals with each surface and subdivision of cells as necessary. This algorithm subdivides the voxels into smaller cubes that lie on the object surfaces and projects the calculated intensity for each cube onto the viewing plane creating a gradient shaded representation of the 3D object [12]. The Dividing Cubes algorithm involves the following steps.

Step 1: The 3D volume data input.

Step 2: At first, four consecutive slices were read into the memory. Subsequently, the data are being processed by advancing one slice at a time.

Step 3: Creation of cube that is defined by eight data values from two consecutive slices.

Step 4: Calculation of gradient vector components by taking differences between forward and backward neighbors along each axis at eight-voxel vertices.

Step 5: Cube classification: Interior cube (intensities of each vertex are above surface value), exterior cube (intensities of the vertices falls behind surface value) and the surface intersect the cube.

Step 6: Division of each cube to subcubes followed by its scanning and testing for surface intersections.

Step 7: Gradient vector interpolation at each cube intersecting the surface.

Step 8: Intensity calculation at each surface point.

3.3 Dual Contouring Algorithm
Dual Contouring algorithm is nearly similar to Marching Cubes algorithm. However, the meshing is performed on a dual mesh and needs the scalar functions to furnish gradients or surface normals in addition to function value. The major advantage of this algorithm is to procreate sharp features, which is not realized by the other implicit surface meshing algorithms. The Dual Contouring algorithm commonly works on Hermite data, i.e. the function value at any given point with all the partial derivatives of the function [13]. The Dual contouring algorithm involves the following steps.

Step 1: The region to be meshed is divided into convex overlapping cells.

Step 2: Evaluation of the meshed scalar function \(f(x, y, z)\) at the vertices of corresponding cells.

Step 3: Labeling of each vertex as being either inside or outside. In addition, the cells having a mixture of inside and outside vertices contain a portion of the surface.

Step 4: Generation of a single dual vertex per cell straddling the surface followed by their connection with neighboring dual vertices to produce final mesh.

Step 5: Location of the dual vertex that passes through the edge intersections is calculated by the equation,

$$E(d) = \sum_{i=1}^{n}(d - p_i)N_i^2$$

Where, \(d\)-dual vertex position, \(p_i\)- location of \(i^{th}\) edge intersection and \(N_i\)- normal for \(i^{th}\) intersection.
4. IMPLEMENTATION

A simple and efficient method for the extraction of continuous
isosurfaces is implemented and tested on various datasets. It
involves two crucial steps. In the first step, a continuous
interpolant of the dataset was determined and second step
includes the extraction of isosurface geometry by sampling the
points on Marching Cubes triangles and projecting them onto the
isosurface defined by the interpolant. The isosurface generation
problem is addressed by computing the triangles of a Marching
Cubes surface in each grid. Inside the triangles, the points are
computed at which volume for function value and gradient are
sampled using interpolation scheme, and then projected onto the
isosurface using continuous interpolant and surface rendering
techniques. The resulting points are rendered by employing the
surface rendering algorithm.

4.1 Coding
The Marching Cubes algorithm is implemented using MATLAB
environment. The entire section of code is divided into three
modules such as data module, algorithm module and rendering
module. Figure 1 shows the block diagram of software design.

![Figure 1. Block diagram of software design.](image)

The data module takes care of input data and its processing
suitable for algorithm module. However, the algorithm module
contains the algorithm portion of the software and implements
the Marching Cubes algorithm. The results are exhibited using
graphical tools of MATLAB in the display module. The output
from the algorithm module is in form of triangles, which are
rendered using MATLAB’s rendering system. Two data sets
have been taken for the present experiments. A raw data
‘bucky.raw’ has been downloaded from the publicly available
domain. Subsequently, this raw data is implemented in
MATLAB environment in order to find the pixel value at each
part of the data. The information of raw data is provided in the
Table 1 and its visual imagery is available in public domain.

![Figure 2. Visual imagery of sphere data](image)

A surface rendering algorithm is exploited for visualizing the
resulting point set representing the isosurface. A software
framework for Marching Cubes has also been designed and
developed for isosurface construction in MATLAB
environment. The results using a synthetic datasets and
discussion of practical considerations are presented. The
importance of present method is that the proposed
implementation could be applied to any arbitrary data.

4.2 Experimental Setup
In the current research work, a software framework is designed
using MATLAB that can be easily installed on Windows, UNIX
and Mac OS X. There are minimum system requirements for
each of the operating system. The platform used for
implementing the algorithms is MATLAB with Windows
operating system. MATLAB makes the task easier as it supports
various image formats such as TIF, Bmp, JPEG, PNG, GIF etc.
The MATLAB installer informs the hard disk space requirement
for particular partition.

5. CONCLUSION
The simplicity and generality of the present method make it an
attractive solution for acquiring high quality isosurfaces from
any volumetric data. An efficient implementation of Marching
Cubes algorithm is presented for the reconstruction of bucky
data and equation based sphere data that have been downloaded
from the publicly available domain. The computational aspects
of Marching Cubes algorithms for faster and efficient implementation are highlighted. The practical aspects of the implementation of Marching Cubes algorithms and performance are studied. MATLAB is easy for prototyping and handling volume data.

6. REFERENCES


