New Algorithm To Compute Volume Of 3D Volumetric Cardiac Model With Multivariable Calculus And Binary Indexed Tree

Nguyen Le Quoc Bao & Le Tuan Hy

Abstract

In the burgeoning field of medical imaging, precise computation of 3D volumes holds significant importance for tasks such as automatic segmentation and reconstruction of internal organs. Addressing the need for efficient algorithms in this domain, we propose a novel approach leveraging multivariable calculus and the binary indexed tree data structure for calculating the volume of 3D cardiac models, This algorithm, designed to minimize computational costs without compromising accuracy, is a pivotal component of our larger research project titled "Integration of Deep Learning into volumetric cardiovascular dissection and reconstruction in simulated 3D space for medical practice". Through rigorous mathematical modeling and algorithmic design, we demonstrate the effectiveness of our approach, paving the way for improved accuracy and efficiency in volumetric analysis of cardiac structures.

I. Introduction

In the contemporary landscape of medical imaging, the conversion of tomographic data into precise three-dimensional (3D) models stands as a burgeoning trend of paramount importance. This evolution necessitates robust post-processing methodologies to ensure the meticulous measurement and analysis of these intricate 3D structures with utmost accuracy and efficiency. Of particular significance is the quantification of various parameters within the cardiovascular system, including but not limited to the diameter, area, and volume of critical structures such as the aortic duct. These metrics serve as pivotal indicators for pathologies such as hypertrophy or stenosis, which pose significant risks to patient well-being. Consequently, the accurate assessment and preoperative planning facilitated by such analyses substantially enhance the clinical efficacy and safety of surgical interventions. Notwithstanding the strides made in tomographic reconstruction software, the precise volumetric measurement of 3D cardiac models remains an enduring challenge, underscoring the persistent absence of a definitive solution within the current landscape of medical imaging technologies. While existing software proficiently translates medical tomographic data into comprehensive 3D representations, the capability to perform volumetric analysis remains conspicuously absent, thereby warranting further advancements in this critical domain.

II. Methodology

1. Integral of Surface Function

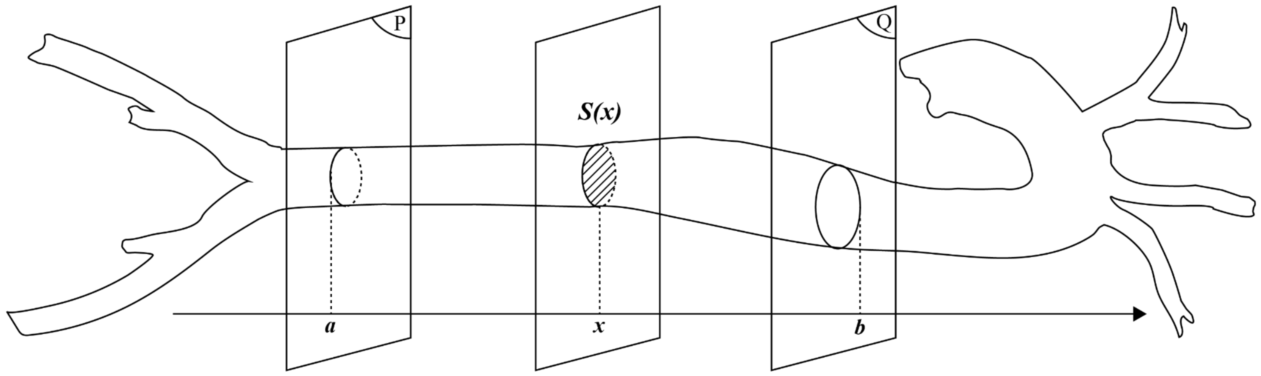


Figure 1: Idea of calculating the volume of the aortic duct by integrating the surface function

We can determine whether the arterial duct is hypertrophied by comparing its volume to standard parameters. We intersect the aorta with two planes *(P),(Q)* using a method. The limits *a,b* on the *Ox* axis of an object are *(x=a, x=b, a<b)*. A plane perpendicular to the *Ox* axis at point *x* and *(a ≤ x ≤ b)* intersects the object with a cross-sectional area of *S(x)* and the equation *S(x)* is continuous on the interval [*a;b*]. Thus, the volume of the upper heart chamber is calculated by the following integral formula:

$$V = \int_a^b S(x)dx$$

The shapes which have fixed shapes such as cube, tetrahedron, prism,... , have predetermined *S(x)* function. However, *S(x)* in this case is not constant. For a set of slice images, if we have *n* slices, then for each *i-th* slice, we have S_i as the area of that part. This is equivalent to having an equation *S(i)* with *i* as the variable. Approximating a function *S(x)* is not straightforward. We can use the Lagrange interpolation method. However, this approach has many limitations. Firstly, the approximation is only accurate if we divide the part into *n* slices, and $n \rightarrow +\infty$ ensures accuracy. This is practically infeasible because the number of slices n is usually not large enough ($300 \rightarrow 512$) and is limited. Secondly, approximating a function is also a complex task, making this algorithm time-consuming. Thus, the approach to the problem with the above method is not ideal. However, the application of calculus and integration has great potential in measuring the volume of objects, specifically with double and triple integrals.

2. Multivariable Calculus

Calculus for multivariable functions focuses on solving problems related to equations in the form of $z = f(x,y)$ in 3D space. This branch of mathematics has numerous practical applications, thereby presenting new potentials for direct application to our three-dimensional space-related research topics. Double and triple integrals are primarily applied in measuring the volume of an object.

2.1 Double Integral

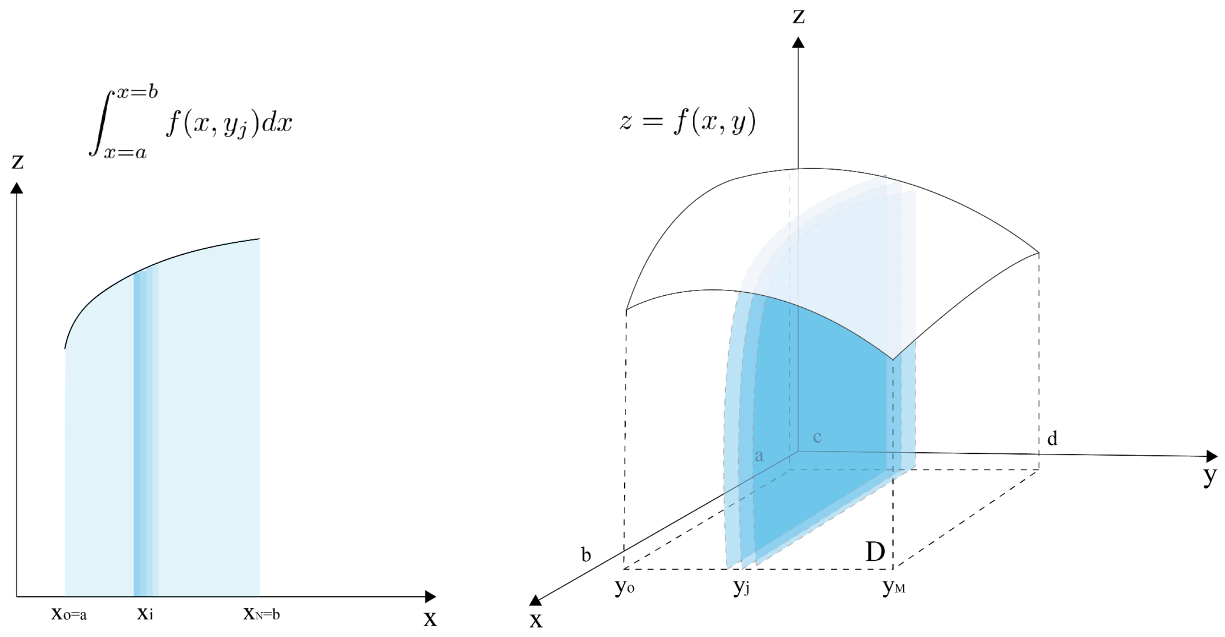


Figure 2: Idea of double integration problem to calculate the volume under a curved surface

For a single integral problem (left figure): the area under a curved graph $f(x)$ is $S = \int_a^b f(x)\,dx$ with dx representing the infinitesimal distance between x_{i+1} and x_i tends towards zero, denoted as $dx \to 0$. For the double integral problem, upon closer inspection, it is essentially an aggregation of multiple single integral problems. Initially, we partition the curved surface $z = f(x,y)$ into surfaces S_j, S_{j+1}, ⋯ , S_M corresponding to each y_j, y_{j+1}, ⋯ , y_M where each surface S_j with fixed y_j represents a single integral problem. Ultimately, we sum up these surfaces S_j to calculate the volume V, which is the integral of S. Thus, we have the formula:

$$D = \begin{cases} a \le x \le b \\ c \le y \le d \end{cases}$$

$$S = \int_a^b f(x,y)dx$$

$$V = \int_c^d \left(\int_a^b f(x,y)dx \right) dy = \int_c^d dy \int_a^b f(x,y)dx$$

The prerequisite for performing double integrals is knowing the equation $z = f(x,y)$. However, in practice, we only have $(x_i, y_i, z_i) \mid i \longrightarrow n$. Hence, we resort to brute force approximation as follows:

$$\begin{cases} \Delta x_i = x_{i+1} - x_i \\ \Delta y_i = y_{i+1} - y_i \end{cases}$$

$$S_j = \lim_{\Delta x_i \to 0} \Sigma_{i=a}^{i=b} z_{ji} \Delta x_i$$

$$V = \lim_{\Delta y_j \to 0} \Sigma_{j=c}^{j=d} S_j \Delta y_j$$

$$\rightarrow V = \lim_{\Delta x_i \to 0} \Sigma_{j=c}^{j=d} (\Sigma_{i=a}^{i=b} z_{ji} \Delta x_i) \Delta y_j$$

2.2 Practical Application

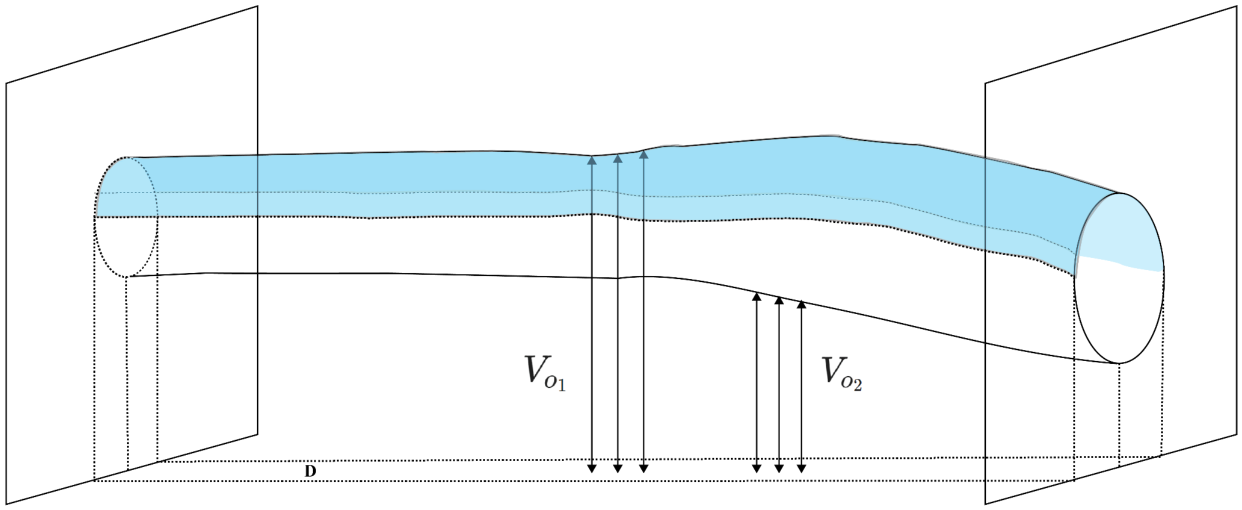


Figure 3: Calculation of partial aortic arch volume with double product

When calculating the volume of a segment (e.g., the aortic arch), we divide this vessel into two parts: the upper half $(o1)$ and the lower half $(o2)$, so the domain D remains the same. The volume of the inner part will be $V_{o1} - V_{o2}$:

$$V_o = V_{o_1} - V_{o_2}$$

$$\rightarrow V_o = \lim_{\substack{\Delta x_i \rightarrow 0 \\ \Delta y_j \rightarrow 0}} (\Sigma_{j=c}^{j=d} (\Sigma_{i=a}^{i=b} z_{ij}^{o_1} \Delta x_i) \Delta y_j) - \Sigma_{j=c}^{j=d} (\Sigma_{i=a}^{i=b} z_{ij}^{o_2} \Delta x_i) \Delta y_j)$$

$$\rightarrow V_o = \lim_{\substack{\Delta x_i \rightarrow 0 \\ \Delta y_j \rightarrow 0}} \Sigma_{j=c}^{j=d} (\Sigma_{i=a}^{i=b} (z_{ij}^{o1} - z_{ij}^{o_2}) \Delta x_i) \Delta y_j$$

With this approach, it is necessary to find a midsection to divide the object into two parts and only use the voxels located on the outer boundary. However, with the double integral method as described above, when a specialist/surgeon performs surgical operations or modifies the segmentation result leading to changes in the 3D reconstruction result, we must rerun the volume calculation algorithm from scratch. If we denote K as the total number of times the cutting or updating operation is performed, the algorithm's complexity will be $O\ (K \times M \times N \times S)$, where S represents the complexity of performing auxiliary operations such as finding the cutting path and determining the outer boundary voxels. For this reason, we have transitioned to researching triple integrals and applying different data structures to address the aforementioned issue.

2.3 Triple Integral

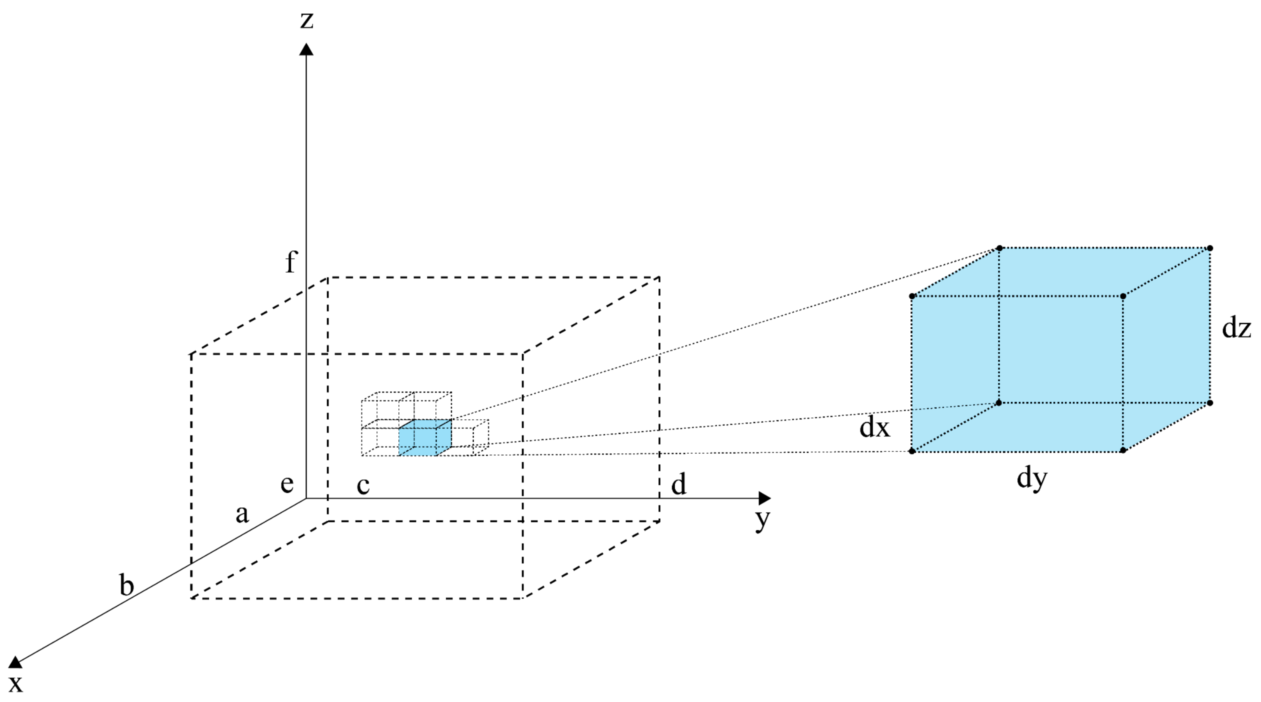


Figure 4: Calculating the volume of an object with triple integration

The concept of triple integration originates from dividing the object volume O into n very small rectangular prisms or cubes O_i with lengths, widths, and heights denoted as dx,dy,dz respectively. Approximating the volume of the original object is straightforward by summing the volumes of all O_i with $1 \leq i \leq n$ and as $n \to +\infty$. Based on the illustration and concept, we have the formula:

$$D = \begin{cases} a \leq x \leq b \\ c \leq y \leq d \\ e \leq z \leq f \end{cases}$$

$$V = \iiint_D dz\, dy\, dx = \int_a^b \int_c^d \int_e^f dz\, dy\, dx$$

Alternatively, in programming, we employ the following brute force formula:

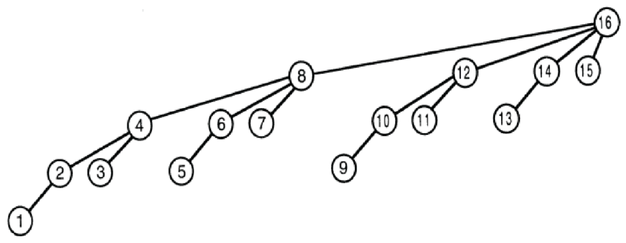
$$V = \lim_{n \to +\infty} \Sigma_{i=1}^{n} \Delta z_i \Delta y_i \Delta x_i$$

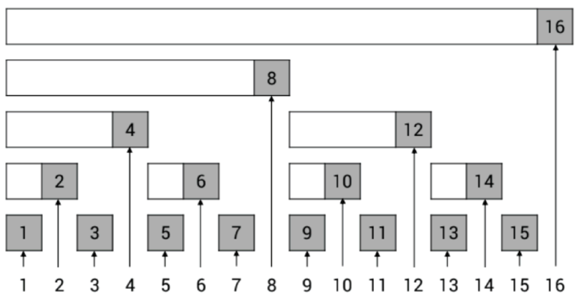
In reconstructing a 3D medical image slice, an image file organizes a volume with three dimensions (N, M, P) where N is the width, M is the length, and P is the height. Therefore, regardless of whether the 3D model undergoes surgery or the segmentation result changes, it still lies within the space region $N \times M \times P$. Thus, if we can manage this space region effectively and store which voxels lie within the volume of the object efficiently, performing two queries — querying within the region and updating within the region—becomes straightforward. This problem leads us to the Fenwick Tree data structure, also known as the Binary Indexed Tree.

3 Binary Indexed Tree

3.1 Principal Operations

The Fenwick Tree is a widely used data structure in competitive programming, introduced in the research paper "A new data structure for cumulative frequency tables" (Peter M. Fenwick, 1994). The Fenwick Tree exhibits the following characteristics: querying the result of a subproblem $[L,R]$ with complexity $O(logN)$, updating the value for one (or a segment) with complexity $O(logN)$, low memory usage $O(N)$, and fast processing speed (due to bitwise operations). In detail, with each update operation, the last bit is always shifted up at least 1 time, leading to the maximum of $\log N$ times of bit shifts.





Fenwick Tree (Binary Indexed Tree), root is node 16 (Fenwick, 1944)

In the Fenwick Tree, the element i in the tree stores the result $F[i]$ of the subproblem containing 2^k elements starting/ending at position i, where k is the lowest set bit in the binary representation of i. Typically, we use a prefix Fenwick tree. Looking at the representation of the tree above, we can easily see that nodes with odd indices manage only themselves, while nodes with index id will have a parent with index $id + 2^k$. Finding the value of 2^k is simply done by performing the bitwise operation: $2^k = -id$ & id.

The Fenwick Tree, or Binary Indexed Tree (BIT), allows us to perform operations such as updating a single element, querying a range, updating a range, and querying an element. The range query operation is similar to removing both ends (surgical operation) without losing the information of the removed part to support the functionality of undoing. The query for the sum of a subarray (u,v) is as follows with complexity $O(logN)$:

$$\begin{aligned}\mathrm{sumRange}(u,v) &= \mathrm{sumRange}(1,v) - \mathrm{sumRange}(1,u-1)\\ &= \mathrm{getSum}(v) - \mathrm{getsum}(u-1)\end{aligned}$$

3.2 Binary Indexed Tree In 3D Space

This is a novel application of binary search trees applied to three-dimensional arrays. Similar to 1D and 2D binary trees, we employ the principle of inclusion and exclusion to query the sum of elements over a 3D space (excluding surrounding parts, akin to cutting but without data loss) and perform updates on a 3D space (altering the segmentation result) with minimal complexity of $O(\log N \times \log M \times \log P)$, where (N, M, P) represent the dimensions of the slice image.

Let V be the 3D array managing the entire spatial region during 3D reconstruction. For our current dataset, V has a size of $(N, M, K) = (512, 600, 600)$. Thus, a cubic block entirely within an object has a volume of 1 (pixel)^3. Note that there is always the ratio between pixel to *mm* specified in the volume data file for unit matching operation. Hence, there are different

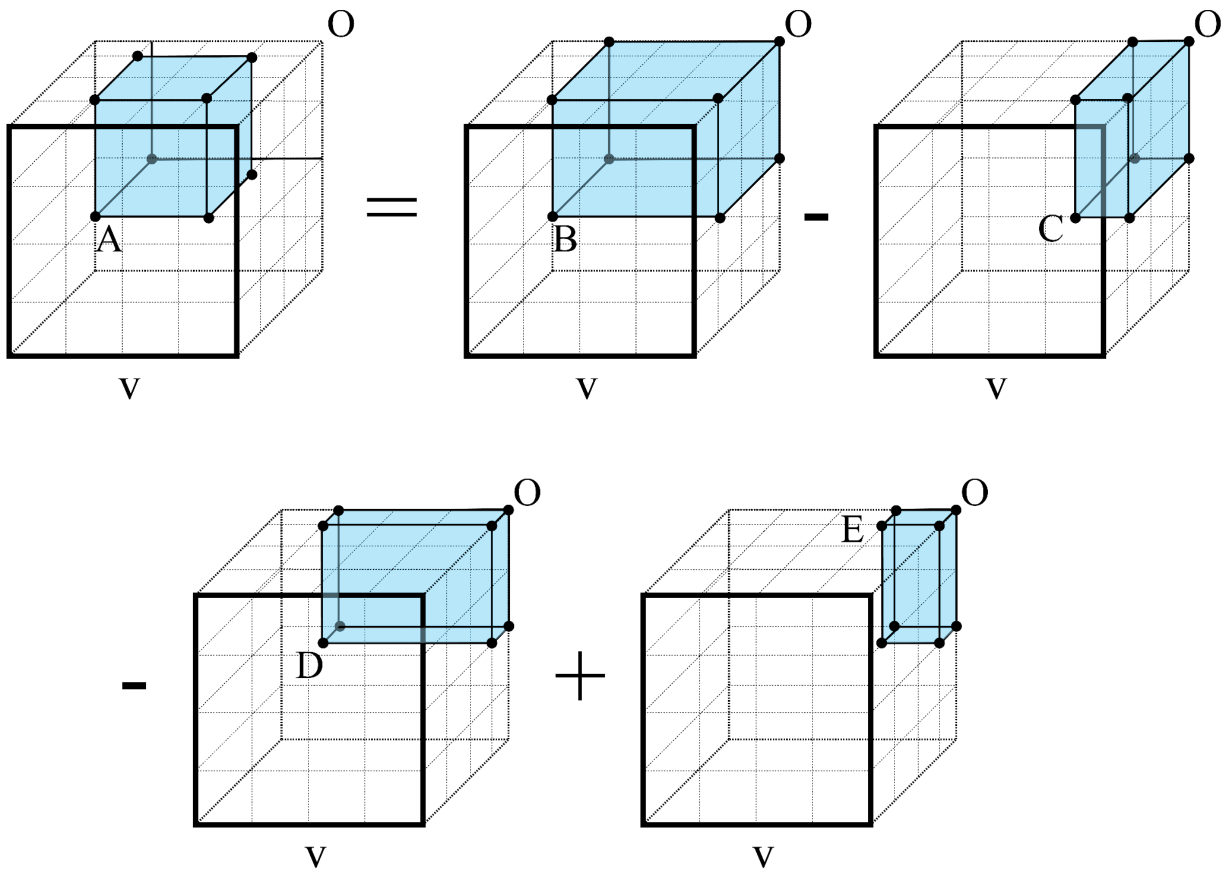
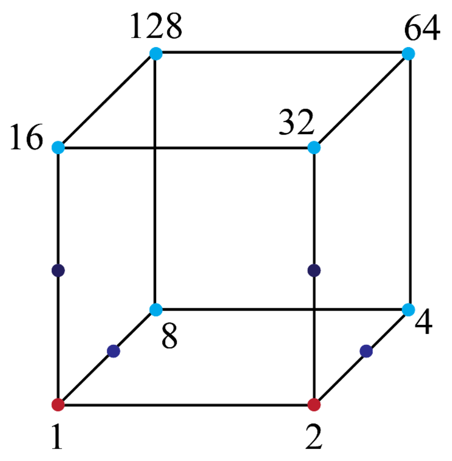


Figure 5: How to query a region in a 3D array with a binary index tree

values for a voxel when the object only partially intersects it. This relates to the marching cubes 3D reconstruction algorithm we are using to reconstruct the 3D object.

5. 3D Reconstruction Algorithm Marching Cubes

Lorensen & Cline (1987) pioneered the revolutionary method, algorithm Marching Cubes, focusing on reconstructing the mesh on the surface of objects. As the object passes through some vertices within a cubic block, to illustrate, let's consider the Marching Squares algorithm: let A be a point where the object passes through (upper object) and B be a point where the object does not pass through (lower object), then the point C between points A and B is likely to be an intersection point with the object. Similarly, with the Marching Cubes algorithm in 3D space. If a vertex has two cases of passing through or not, there are up to $2^8 = 256$ possible cases.



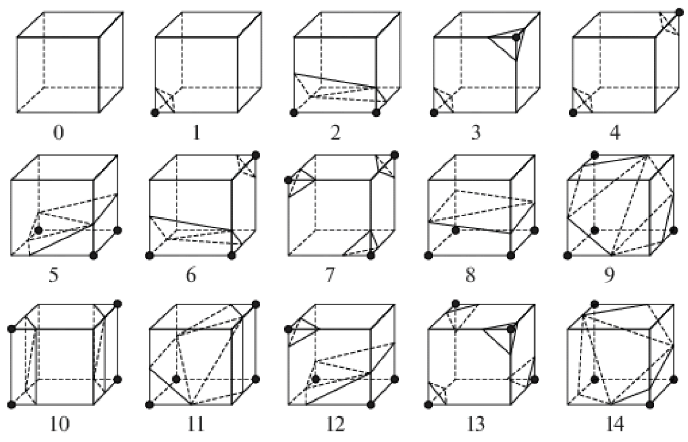


Figure 6: Description of 15 cases of the Marching Cubes algorithm

However, using rotational symmetry in space reduces it to only 15 cases. Thus, we have 15 cases of volume for a cubic block plus the case of volume equal to 1 when the block is entirely within the object. Therefore, when this mesh structure changes, we only need to use the region update function with the Binary Indexed Tree and query the new total volume with low complexity in a short time. Since the Marching Cubes algorithm already iterates through the entire 3D cubic space to reconstruct the 3D structure, we utilize these loops to simultaneously initialize the Fenwick Tree (Binary Indexed Tree).

References

Fenwick, P. M. (1994). A new data structure for cumulative frequency tables. Software: Practice and experience, 24(3), 327-336.

Lorensen, W. E., & Cline, H. E. (1998). Marching cubes: A high resolution 3D surface construction algorithm. In *Seminal graphics: pioneering efforts that shaped the field* (pp. 347-353).

Liu, Y. S., Yi, J., Zhang, H., Zheng, G. Q., & Paul, J. C. (2010). Surface area estimation of digitized 3D objects using quasi-Monte Carlo methods. *Pattern Recognition*, *43*(11), 3900-3909.