Knowledge Vector Representation of Three Dimensional Convex Polyhedrons and Reconstruction of Medical Images using Knowledge Vector

Shilpa Rani (shilpachoudhary2020@gmail.com)
Lovely Professional University, Hyderabad
Kamlesh Lakhwani
Lovely Professional University Faculty of Technology and Sciences
Sandeep Kumar
pentagram research center hyderabad

Research Article

Keywords: Construction, reconstruction, 3D medical images, syntactic grammar, knowledge vector.

Posted Date: June 2nd, 2021

DOI: https://doi.org/10.21203/rs.3.rs-380191/v1

Knowledge Vector Representation of Three Dimensional Convex Polyhedrons and Reconstruction of Medical Images using Knowledge Vector

Shilpa Rani[1], Kamlesh Lakhwani[2], Sandeep Kumar[3]
[1]*Research Scholar, Lovely Professional University, Punjab, India*
[2]*Associate Professor, CSE Department, Lovely Professional University, Punjab, India*
[3]*Post Doc Fellow, Pentagram Research Centre, Hyderabad, India*

[1] shilpachoudhary2020@gmail.com, [2]kamlesh.20980@lpu.co.in, [3]er.sandeepsahratia@gmail.com

Abstract

Three-dimensional image construction and reconstruction play an important role in various applications of the real world in the field of computer vision. In the last three decades, researchers are continually working in this area because construction and reconstruction is an important approach in medical imaging. Reconstruction of the 3D image allows us to find the lesion information of the patients which could offer a new and accurate approach for the diagnosis of the disease and it adds a clinical value. Considering this, we proposed novel approaches for the construction and reconstruction of the image. First, the novel construction algorithm is used to extract the features from an image using syntactic pattern recognition. The proposed algorithm is able to extract in-depth features in all possible directions and planes and also able to represent the 3D image into a textual form. These features vector is nothing but a string that consists of direction and length information in syntactic form. For the identification of syntactic grammar, a real 3D clay model was made and identified the different possible patterns in the image. According to the domain knowledge, in a 3D image, a pixel could be present in 26 possible directions and we incorporated all possible directions in the proposed algorithm. In the same way, for the reconstruction of the image novel algorithm is proposed. In this algorithm, the knowledge vector has been taken as an input and the algorithm is able to reconstruct a 3D image. Reconstruction allows us to explore the internal details of the 3D images such as the size, shape, and structure of the object which could take us one step ahead in the field of medical image processing. Performances of the proposed algorithms are evaluated on five medical image dataset and the datasets are collected from Pentagram research institute, Hyderabad and results are outperformed in real-time. The accuracy of the proposed method is 94.78% and the average execution time is 6.76 seconds which is better than state of art methods.

Keywords: Construction, reconstruction, 3D medical images, syntactic grammar, knowledge vector.

1. Introduction

Pattern recognition encloses two primary tasks: description & classification. Pattern recognition system generates the description of the object which is present in the image and then classifies the object according to the description. There are two different approaches for implementing pattern recognition: statistical and structural. These two techniques use a different method for description and classification purposes. Statistical approach [1-3] use decision-theoretic concepts for the discrimination between the objects which belongs to different class based on their quantitative features. Structural pattern recognition approach [4-6] sometimes referred as Syntactic pattern recognition approach because it uses syntactic grammars concept for the discrimination between objects which belongs to different class based upon their shape, size or structural features. Some researchers combined both structural and statistical approach for pattern recognition. The statistical approach is well explored by many researchers but structural pattern recognition still doesn't explore adequately because it requires domain knowledge for the description and classification purpose. Therefore, Structural pattern recognition techniques are domain-specific. In this paper, we proposed a novel approach basis on the Syntactic approach which represents the three-dimensional convex polyhedrons in textual form. This textual representation consists of the direction and length code of the objects which are present in the image. The textual representation is referred to as the knowledge vector. Further, this knowledge vector is considered as input and a novel algorithm is written in such a way that reconstructs a Three Dimensional image using the knowledge vector. This is the First time in the field of image processing where a Three Dimensional image can be reconstructed using textual information and it could be a great contribution in the field of structural pattern recognition as well as it adds a new value in the field of image processing. As we discussed already that structural pattern recognition techniques are domain-specific so that we considered only medical images for our research. Many researchers are working on the reconstruction of a 3D image because the minimally invasive procedure is getting popular nowadays. After all, it provides better accuracy and minimum recovery time. In the above-mentioned method, the surgical navigation system will always play an important and critical role. It helps the doctors to focus on a particular

position, edge, or outline of the image. But Most of the surgical navigation systems are based on two-dimensional medical images but very less development in the field of Three-dimensional medical images. Therefore we developed a platform for the reconstruction of the Three Dimensional images. This System takes the series of 2D/ 3D medical images as an input and converts these series of images into textual form first and then in the second stage this textual information/knowledge vector would be taken as an input and reconstruction of the 3D image would be done in this stage. It displays the human organs on the computer screen which gives convenience to the doctors for the diagnosis of the disease. This Three Dimensional reconstruction gives the facility to doctors to focus at a particular point and find the nature of the lesion and their surrounding tissues, rotation of the image, traverse the image from inside, zoom in and zoom out, thus it helps the doctors to diagnose the diseases and its severity and improve the accuracy and decrease the diagnosis time.

1.1 Motivation and Objectives

The primary motivation behind this research is to develop a novel algorithm which converts an image into a textual form using a syntactic pattern recognition technique and the secondary objective of this paper is to reconstruct a three-dimensional image using a knowledge vector. The motivation behind this research is to explore the new possibility of structural pattern recognition and this research can give a new direction to understand the three-dimensional images in textual form.

This paper is divided into 4 sections. The second section covers the motivation and objective of the research. Previous related work is discussed after the introduction. Third, the fourth and fifth section covers the proposed methodology and evaluation parameters and results. While conclusion and future work are discussed in the last section.

2. Related Work

Yinsheng Li and Guang-Hong Chen [7] introduced DIM which is used to identify the inconsistency of the acquired projected data taken from different viewpoints. Further, the input data is divided into a subset according to the DIM value at each viewpoint angle and then the SMART-RECON algorithm is applied to the datasets for the reconstruction of the images. Practical utilization of the proposed method is tested on vivo human subject data and

performance is better than existing methods. Aurelien Bustin et al. [8] proposed a method that combined the statistical and geometric 3D features. The main aim of the proposed method is to make a single isotropic volume by combining the multiple 3D volumetric data which is acquired from different orientations and recovery of sharp edges of the image and thin anatomical structure. These two aims are achieved using the Augmented Lagrangian scheme. The performance of the proposed method is evaluated on numerical simulations and the MRI datasets. Benjamin Hou et al. [9] proposed a method in which intensity information of 2D image slices is used for the registration of the image and further CNN is used for transformation of the image and for the learning of the regression function which maps the 2D slices into three-dimensional atlas spaces. Performance of the algorithm is evaluated on MRI dataset, fetal imagery with synthetic motion, and real MRI data and achieved average spatial prediction error of 7mm. This method is a computationally better solution for 2D and 3D registration and it gives good results in real-time scenarios. Daniel p. Benalcazar et al. [10] built a 3D model by adding the depth information in 2D iris images. First, the smartphone camera captured the various 2D images from different angles and different lighting conditions, and then structure from motion algorithm is applied for reconstruction of the point cloud three-dimensional model. Finally, the best fitting Three Dimensional mesh model is obtained with the help of the screened Poisson surface reconstruction method. The performance of the proposed method is evaluated on different ten subjects and the model contained approx 11000 three-dimensional points. The performance of the proposed 3D iris scanning method is better than the existing methods. Xuehang Zheng, Saiprasad Ravishankar [11] proposed a new PWLS based reconstruction method and the quality of the reconstructed 3D images is better than the existing PWLS-EP based method. This proposed method improved the quality of the image and image resolution and it's more efficient in terms of computational speed. Chen Zhang[12] proposed a method for volumetric surface reconstruction and assumed that every scene consists of a cuboids structure. The proposed method having reconstruction fidelity even if the object consists of a heavy curved and concave structure and the method is able to preserve the smoothness and cleanliness of the surface. This method is suitable for 3D printing of the images. Yuanhao Guo et al. [13] proposed a two-phase three-dimensional reconstruction method for light microscopy axis view images. In the first phase, an improved 3D volumetric representation is defined and in the 2nd phase, 3D reconstruction is achieved by searching the optimal surface over the confidence mapping. The

performance of the proposed method is evaluated on three different datasets and the approach can represent 3D shape in a precise manner. This method can be used to determine the shape of the objects. Lu Ding et al. [14] proposed a method for optoacoustic image systems which is helpful for finding the shape and size of the image. These methods have significant improvement in the spatial resolution, a contrast to noise ratio, and quality of the reconstructed image is reported better than existing techniques. BIN LI et al. [15] proposed a 3D-ReConstnet network which used a residual network for the extraction of the features from the 2D image. If objects are occluded in the image then the Gaussian probability distribution method is used for the learning purpose. The performance of the proposed method is evaluated on ShapeNet and Pix3D dataset and the results are comparatively better. Volumetric representation is an important approach to three-dimensional reconstruction. The main goal of volumetric representation is estimating a convex hull in three-dimensional spaces from each view [16-19]. In the same way, the space carving algorithm is to recover the shape of the 3D objects by removing the voxels which are not visible in the particular view [20-21]. But these methods require the proper segmentation and it is not always possible in some scenarios. There are many well designed three-dimensional reconstruction algorithms are available. Some algorithms are only designed for the reconstruction of peculiar and transparent objects [22]. These methods outperform on a macroscopic scale because to collect the surface-related information they setup the various lightning while capturing the images [23-24]. In recent research [25] authors have presented a semantic reconstruction which is a combination of a data term and regularization constraints and results are outperformed on public datasets [26]. Now a day's researchers are focusing on deep learning-based methods for improving the matching quality[27-28] and for the reconstruction of 3D images but these methods require a large amount of training and testing data which are not available in our case.

After study various papers on 3D reconstruction, we conclude that volumetric based approaches are giving more promising results and properly addressing the challenges of 3D reconstruction. This is also true that in some cases the shape of the 2D/ 3D images is not accurate as expected. Therefore we developed a novel approach that is working based on 3D convex polyhedrons. The 3D image consists of voxels and each voxel itself represents a cube. So a 3D image is an array of cubes. Here we are considering the structuring element of size 3X3X3 so it consists of 27 neighborhood structures and this 3X3X3 size of the structuring element is used for processing of

a 3D image. The whole 3D image is scanned using the structuring element and an algorithm is developed in such a way that it converts the image into a textual form which consists of direction code and length code and it is called a knowledge vector. This knowledge vector preserves the shape of the objects which is present in the image. Further, the same knowledge vector is used as an input for the reconstruction of the original 3D image. The detail of the proposed method is given in the further section.

3. Proposed Methodology

The proposed methodology is divided into three parts and the flow chart of the proposed method is shown in Fig. 1. The detailed explanation is discussed further:

I. Knowledge acquisition
II. Feature Extraction
III. Reconstruction of 3D images

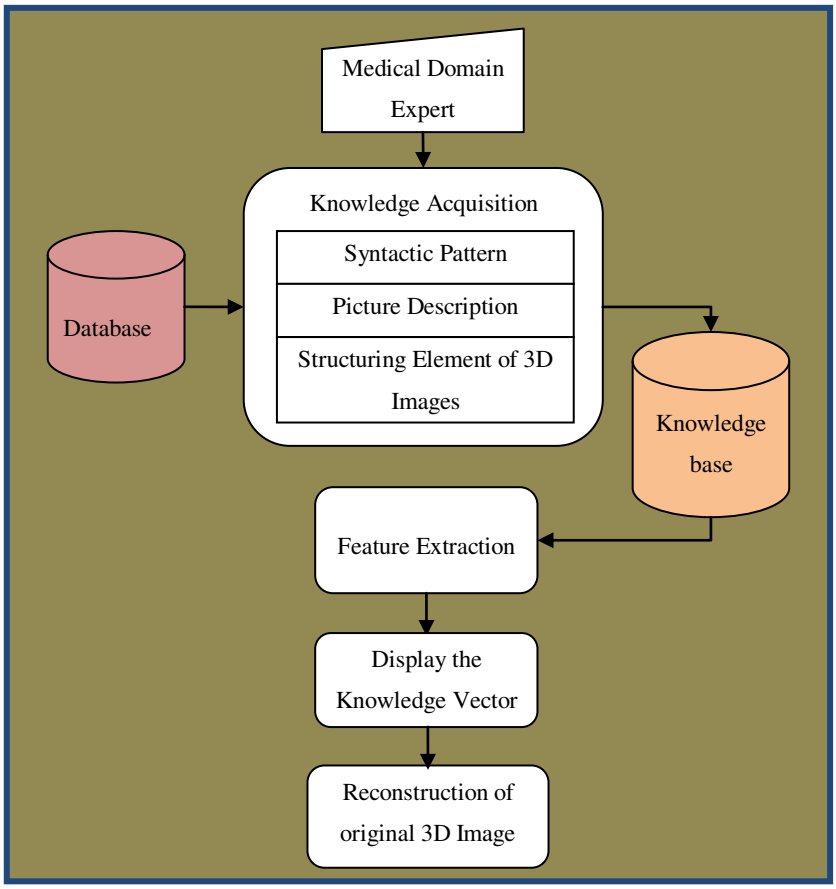


Fig 1: Flow chart of the proposed methodology

3.1 Knowledge Acquisition

Knowledge acquisition is a method on which domain knowledge is acquired for extracting the features of a specific application or domain. This is the first step of structural pattern recognition based methods. Generally, Structural approach uses syntactic grammars concept for the discrimination between objects which belongs to different class based upon their shape, size, or structural features. Structural pattern recognition techniques are domain-specific. In this paper, we proposed a novel approach basis on the Syntactic approach which represents the three-dimensional convex polyhedrons in textual form. This textual representation consists of the direction and length code of the objects which are present in the image. The textual representation is referred to as the knowledge vector. Further, this knowledge vector is considered as input and a novel algorithm is written in such a way that reconstructs a three-dimensional image using the knowledge vector. Further part knowledge acquisition steps are explained in detail.

a. Syntactic Pattern recognition

The new and less explored approach of pattern recognition is Syntactic pattern recognition which consists of the formal language theory concepts. Syntactic pattern recognition term is a synonym of the linguistic, grammatical, and structural pattern recognition system. The formal language theory was originated by Noam Chomsky in the 1950s with the development of a mathematical model of grammar. The concept is described as follows:

Alphabet is a finite set of symbols.

A word is any finite string that consists of symbols from the alphabets. For example a valid word of alphabets {a, b} is {a, b, ab, ba, bb, aa….}. A word with no symbol is an empty word that is denoted byΛ. A language that consists of a set of words over a finite set of alphabets. Every natural language follows some grammar rules. The grammar of the formal language consists of four tuples G = {P_N, P_T, P_r, R}

Here

P_N= Set of variables or terminals.

P_T= Set of constants or non-terminals.

P_r= set of production rules.

R_s=Root symbol or start symbol.

P= Union of P_N and P_T

Here P_N and $P_{T\ are}$ disjoint set and R always belongs to either P_N or P_T

Set of empty words denoted by P* which is also called free Monoid.

P^+ is a set of sentences of V^*- Λ which is called a free semigroup.

The language which is generated by these tuples G is called L(G). It should satisfy two conditions: Each string should only consist of terminals. Every string should derive from the root R_s by applying the suitable production rules. Production rules consist of expressions in the form of $X \rightarrow Y$. It means string X can be replaced by string Y.

Where

X = String in P^+

Y=String in P*

In the proposed methodology, the set of terminals or constants are denoted by:

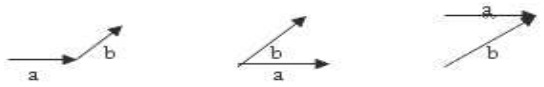
P_T={ R, DR, D, DL, L, UL, U, UR, B, BR, BDR, BD, BDL, BL, BUL, BU, BUR, F, FR, FDR, FD, FDL, FL, FUL, FU, FUR}. And root R can be changed according to the object position. Here these symbols represent the direction in which a pixel could be present.

Here R, D, L, U, B, and F are elementary symbols, and DR, DL, UL, UR, BR, BDR, BD, BDL, BL, BUL, BU, BUR, FR, FDR, FD, FDL, FL, FUL, FU, FUR are composite symbols of the alphabet P_T. The set of production rules are generated through the Normal algorithm which was introduced by A.A. Markov. The Normal algorithm is used to recognize the angle of change in a particular direction during tracking of the contour. This is done with the help of the look-ahead tracing (LAT) method.

b. Picture Description Language (PDL)

Picture description language is an application of semantic/linguistic concepts of pattern recognition. The pattern can be represented in the form of string grammar and this string grammar can be obtained using a simple juxtaposition of a string. Juxtaposition is nothing but keeping the two objects together without losing their identities. This can be done with concatenation also but it involves some spatial arrangements and there is a possibility of losing the identity or losing some information related to the objects. Juxtaposition is easy and reliable because it only involves the head and tail position. Based on this permissible

form of juxtaposition and because each primitive is abstracted as a directed line segment, it is evident that the structure of PDL are directed graphs and also that these structures can be handled by string grammars. The rules of a juxtaposition of PDL are as follows:



Blank primitives must be used for generating disjoint structures. A null point primitive has an identical head and tail. The mechanics of PDL could be used to obtain the contour/ wireframe of a pattern. To illustrate this mechanism, consider the following PDL grammar:

G= $\{P_N, P_T, P_r, R\}$

Where

$P_N = \{S, A_1, A_2, A_3, A_4, A_5, A_6, A_7\}$

P_T={ R, DR, D, DL, L, UL, U, UR, B, BR, BDR, BD, BDL, BL, BUL, BU, BUR, F, FR, FDR, FD, FDL, FL, FUL, FU, FUR}

A set of production rules are represented below:

P_r=

R_s	→	R+P1
P1	→	DR+P2
P2	→	D+P3
P3	→	DL+P4
P4	→	L+P5
P5	→	UL+P6
P6	→	U+P7
P7	→	UR+P8
P8	→	B+P9
P9	→	BR+P10
P10	→	BDR+P11
P11	→	BD+P12
P12	→	BDL+P13

P13	→	BL+P14
P14	→	BUL+P15
P15	→	BU+P16
P16	→	BUR+P17
P17	→	F+P18
P18	→	FR+P19
P19	→	FDR+P20
P20	→	FD+P21
P21	→	FDL+P22
P22	→	FL+P23
P23	→	FUL+P24
P24	→	FU+P25
P26	→	FUR

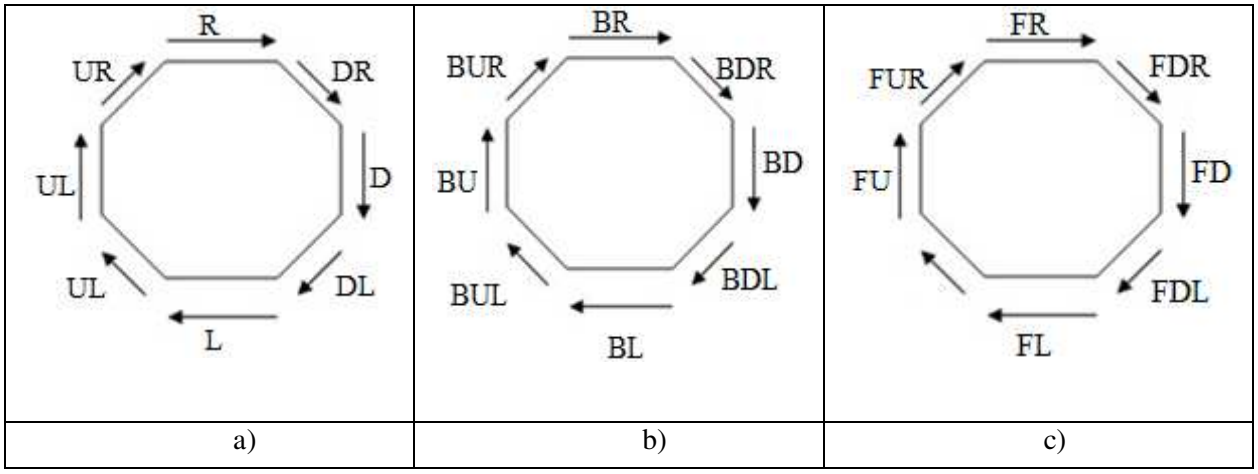
The result obtained by applying all the productions is

mR+(DR+(D+(DL+(L+(UL+(U+(UR+(B+

(BR+ (BDR+ (BD+ (BDL+ (BL+ (BUL+ (BU+ (BUR+ (F+

(FR+ (FDR+ (FD+ (FDL+ (FL+(FUL+ (FU+FUR))))))))))))))))))))))))

The above production system creates an octagon as represented in the below figure:



a)	b)	c)

Fig 2: Pictorial representation of a) Center Plane b) Back Plane and c) Front Plane with corresponding pixel directions.

The directions of the pixels are defined with the help of production rules. Suppose an alphabet A is given then A* could be constructed over A which consists of all the possible words containing A and null word also. So any subset of A* is a formal language. According to this theory, any digital image is consists of a set of patterns and the same digital image can be represented as a language that consists of a finite array of vertex/ vertices. As mentioned, the PDL is a language of representing the digital image into the regular array of vertices.

c. Structuring elements of 3D images

The size and shape of a 2D/ 3D processed image always depend on the structuring elements which have been chosen for processing the image. In the case of a 2D image, if the window size/ neighborhood structure is 3X3 then a maximum of 16 convex polygons can be present. For a 3D image if the window size/ neighborhood structure is 3X3X3 then a maximum of 256 convex polyhedrons can be present. In the same way, we can find the number of concave polygons and polyhedrons. These polygons and polyhedrons are called a structuring element which can be used for the processing of 2D and 3D images. The number of structuring elements always depends on the size of the neighborhood. This concept was introduced by Rajan(1990) and Jirawit Lerdsinmongkol (2008). Suppose an image with dimension (m x n x d) where m is the width of the image, n is the

height of the image and d is the depth of the dataset. The size of the structuring element is 3X3X3 so there will be a total of 27 neighborhoods including central pixel. In the same way, we can have different structuring elements which are represented in Fig. 3. In our research, we used a neighborhood structure of size 3X3X3 so we have 256 possible structuring elements. A detailed description of 3D convex polyhedrons is given below:

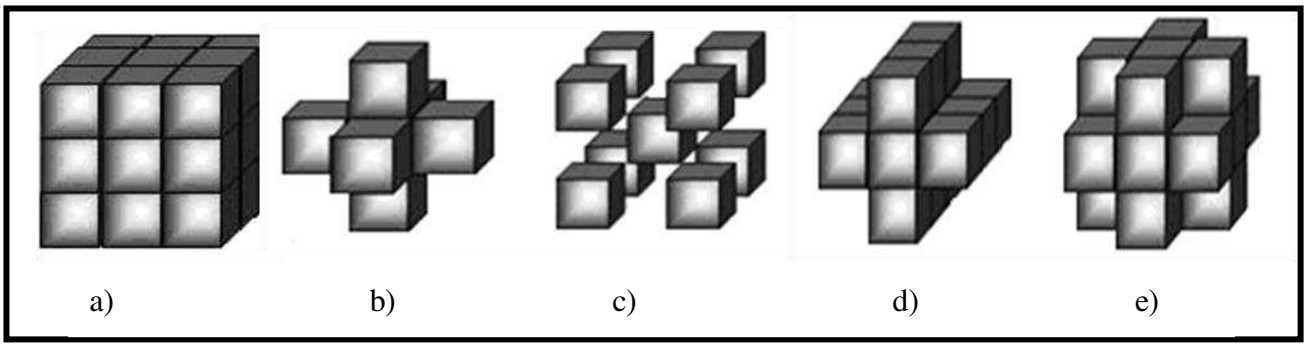


Fig 3 a] 27-Neighbourhood [b] 7-Neighbourhood [c] 9-Neighbourhood [d] 15-Neighbourhood [e] 19-Neighbourhood Structuring elements.

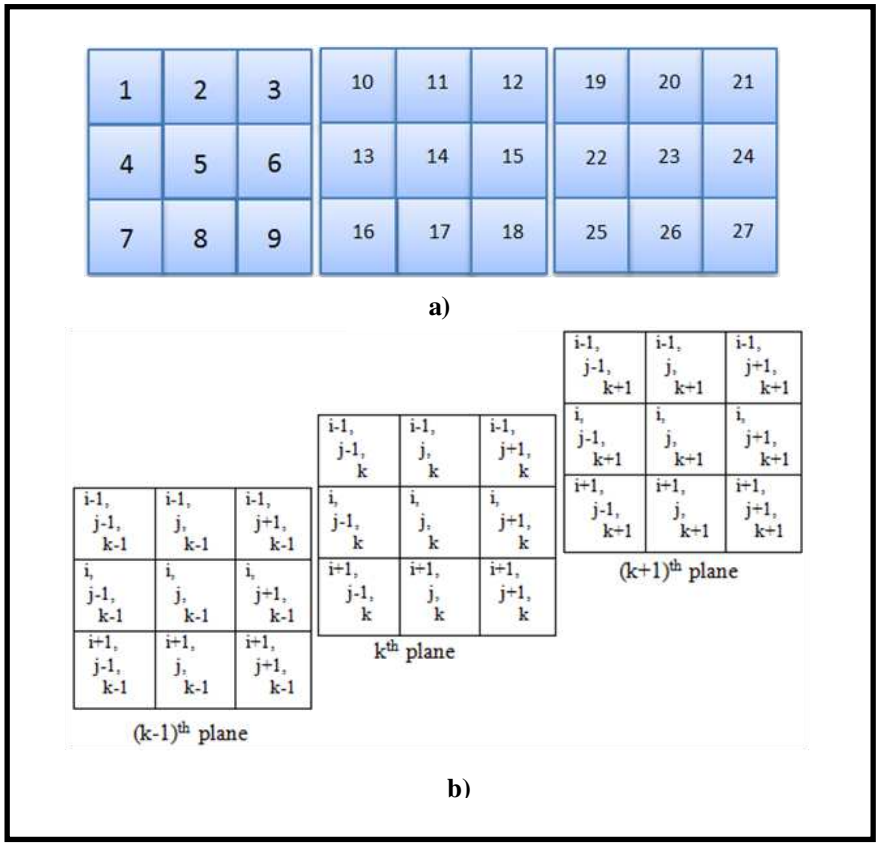


Fig 4. a) 27-neighborhood structure of 3X3X3 window b) Representation of Co-ordinates

Three-dimensional rectangular convex polyhedrons

The idea of constructing 3D convex polyhedrons is the same as 2D convex polygons. As we discussed earlier, a 3D image consists of voxels and each voxel itself represents a cube. So a 3D image is an array of cubes. Here we are considering the structuring element of size 3X3X3 so it consists of 27 neighborhood structures. The smallest size of possible convex polyhedrons consists of 7 neighborhoods. The labeling of the 3X3X3 array and their possible coordinates are represented further in Fig. 4 and three different planes are considered where the k^{th} plane is the central plane and (k-1) and (k+1) is the front and rear plane.

In a 3X3X3 size of array total of 8 corner pixels so there is a possibility of any corner pixel would not be available or any two corner pixels would not be available etc. Therefore, there are 256 different possible ways of constructing the convex polyhedrons w. r. t central pixel 14. The formula is given below:

$$\sum_{k=0}^{8} {}^{8}C_{K}$$

$$={}^{8}C_0+{}^{8}C_1+{}^{8}C_2+{}^{8}C_3+{}^{8}C_4+{}^{8}C_5+{}^{8}C_6+{}^{8}C_7+{}^{8}C_8=256$$

Here, these convex polyhedrons are divided into 9 categories:

- When all the corner pixels are present then the number of possible convex polyhedrons will be ${}^{8}C_0$ which is equivalent to 1.
- When one of the corner pixels is not available then the number of possible convex polyhedrons will be ${}^{8}C_1$ which is equivalent to 8.
- When two of the corner pixels are not available then the number of possible convex polyhedrons will be ${}^{8}C_2$ which is equivalent to 28.
- When three of the corner pixels are not available then the number of possible convex polyhedrons will be ${}^{8}C_3$ which is equivalent to 56.
- When all the corner pixels are not available then the number of possible convex polyhedrons will be ${}^{8}C_4$ which is equivalent to 70.
- When all the corner pixels are not available then the number of possible convex polyhedrons will be ${}^{8}C_5$ which is equivalent to 56.

- When all the corner pixels are not available then the number of possible convex polyhedrons will be ${}^{8}C_{6}$ which is equivalent to 28.
- When all the corner pixels are not available then the number of possible convex polyhedrons will be ${}^{8}C_{7}$ which is equivalent to 8.

When all the corner pixels are not available then the number of possible convex polyhedrons will be ${}^{8}C_{8}$ which is equivalent to 1. Further, all the possible 256 convex polyhedrons are represented in Table 1 and here A, B, C, D, E, F, G, H and I represent the Possible eliminated pixels, and group P, Q, R, S, T, U, V, W, and X represents the possibility of convex polyhedron after removing the pixels.

Table 1: Representation of a possible combination of convex polyhedrons

Group P contains 1 possible combination			
$A=\{\}$ $P_a=\{1,3,7,9,19,21,25,27\}$			
Group Q contains 8 possible combinations			
$B_a=\{1\}$ $Q_1=\{3,7,9,19,21,25,27\}$	$B_b=\{3\}$ $Q_3=\{1,7,9,19,21,25,27\}$	$B_c=\{7\}$ $Q_7=\{1,3,9,19,21,25,27\}$	$B_d=\{9\}$ $Q_9=\{1,3,7,19,21,25,27\}$
$B_e=\{19\}$ $Q_e=\{1,3,7,9,21,25,27\}$	$B_f=\{21\}$ $Q_f=\{1,3,7,9,19,25,27\}$	$B_g=\{25\}$ $Q_a=\{1,3,7,9,19,21,27\}$	$B_h=\{27\}$ $Q_a=\{1,3,7,9,19,21,25\}$
Group R contains 28 combinations			
$C_a=\{1,3\}$ $R_{1,3}=\{7,9,19,21,25,27\}$	$C_b=\{1,7\}$ $R_{1,7}=\{3,9,19,21,25,27\}$	$C_c=\{1,9\}$ $R_{1,9}=\{3,7,19,21,25,27\}$	$C_d=\{1,19\}$ $R_{1,19}=\{3,7,9,21,25,27\}$
$C_e=\{1,21\}$ $R_{1,21}=\{3,7,9,19,25,27\}$	$C_f=\{1,25\}$ $R_{1,25}=\{3,7,9,19,21,27\}$	$C_g=\{1,27\}$ $R_{1,27}=\{3,7,9,19,21,25\}$	$C_h=\{3,7\}$ $R_{3,7}=\{1,9,19,21,25,27\}$
$C_i=\{3,9\}$ $R_{3,9}=\{1,7,19,21,25,27\}$	$C_j=\{3,19\}$ $R_{3,19}=\{1,7,9,21,25,27\}$	$C_k=\{3,21\}$ $R_{3,21}=\{1,7,9,19,25,27\}$	$C_l=\{3,25\}$ $R_{3,25}=\{1,7,9,19,21,27\}$
$C_m=\{3,27\}$ $R_{3,27}=\{1,7,9,19,21,25\}$	$C_n=\{7,9\}$ $R_{7,9}=\{1,3,19,21,25,27\}$	$C_0=\{7,19\}$ $R_{7,19}=\{1,3,9,21,25,27\}$	$C_p=\{7,21\}$ $R_{7,21}=\{1,3,9,19,25,27\}$
$C_q=\{7,25\}$ $R_{7,25}=\{1,3,9,19,21,27\}$	$C_r=\{7,27\}$ $R_{7,27}=\{1,3,9,19,21,25\}$	$C_s=\{9,19\}$ $R_{9,19}=\{1,3,7,21,25,27\}$	$C_t=\{9,21\}$ $R_{9,21}=\{1,3,7,19,25,27\}$
$C_u=\{9,25\}$ $R_{9,25}=\{1,3,7,19,21,27\}$	$C_v=\{9,27\}$ $R_{9,27}=\{1,3,7,19,21,25\}$	$C_w=\{19,21\}$ $R_{19,21}=\{1,3,7,9,25,27\}$	$C_x=\{19,25\}$ $R_{19,25}=\{1,3,7,9,21,27\}$
$C_y=\{19,27\}$	$C_z=\{21,25\}$	$C_{za}=\{21,27\}$	$C_{zb}=\{25,27\}$

$R_{19,27}$={1,3,7,9,21,25}	$R_{21,25}$={1,3,7,9,19,27}	$R_{21,27}$={1,3,7,9,19,25}	$C_{25,27}$={1,3,7,9,19,21}
Group S contains 56 combinations			
D_a={1,3,7} $S_{1,3,7}$={9,19,21,25,27}	D_b={1,3,9} $S_{1,3,9}$={7,19,21,25,27}	D_c={1,3,19} $S_{1,3,19}$={7,9,21,25,27}	D_d={1,3,21} $S_{1,3,21}$={7,9,19,25,27}
D_e={1,3,25} $S_{1,3,25}$={7,9,19,21,27}	D_f={1,3,27} $S_{1,3,27}$={7,9,19,21,25}	D_g={1,7,9} $S_{1,7,9}$={3,19,21,25,27}	D_h={1,7,19} $S_{1,7,19}$={3,9,21,25,27}
D_i={1,7,21} $S_{1,7,21}$={3,9,19,25,27}	D_j={1,7,25} $S_{1,7,25}$={3,9,19,21,27}	D_k={1,7,27} $S_{1,7,27}$={3,9,19,21,25}	D_l={1,9,19} $S_{1,9,19}$={3,7,21,25,27}
D_m={1,9,21} $S_{1,9,21}$={3,7,19,25,27}	D_n={1,9,25} $S_{1,9,25}$={3,7,19,21,27}	D_o={1,9,27} $S_{1,9,27}$={3,7,19,21,25}	D_p={1,19,21} $S_{1,19,21}$={3,7,9,25,27}
D_q={1,19,25} $S_{1,19,25}$={3,7,9,21,27}	D_r={1,19,27} $S_{1,19,27}$={3,7,9,21,25}	D_s={1,21,25} $S_{1,21,25}$={3,7,9,19,27}	D_t={1,21,27} $S_{1,21,27}$={3,7,9,19,25}
D_u={1,25,27} $S_{1,25,27}$={3,7,9,19,21}	D_v={3,7,9} $S_{3,7,9}$={1,19,21,25,27}	D_w={3,7,19} $S_{3,7,19}$={1,9,21,25,27}	D_x={3,7,21} $S_{3,7,21}$={1,9,19,25,27}
D_y={3,7,25} $S_{3,7,25}$={1,9,19,21,27}	D_z={3,7,27} $S_{3,7,27}$={1,9,19,21,25}	D_{za}={3,9,19} $S_{3,9,19}$={1,7,21,25,27}	D_{zb}={3,9,21} $S_{3,9,21}$={1,7,19,25,27}
D_{zc}={3,9,25} $S_{3,9,25}$={1,7,19,21,27}	D_{zd}={3,9,27} $S_{3,9,27}$={1,7,19,21,25}	D_{ze}={3,19,21} $S_{3,19,21}$={1,7,9,25,27}	D_{zf}={3,19,25} $S_{3,19,25}$={1,7,9,21,27}
D_{zg}={3,19,27} $S_{3,19,27}$={1,7,9,21,25}	D_{zh}={3,21,25} $S_{3,21,25}$={1,7,9,19,27}	D_{zi}={3,21,27} $S_{3,21,27}$={1,7,9,19,25}	D_{zj}={3,25,27} $S_{3,25,27}$={1,7,9,19,21}
D_{zk}={7,9,19} $S_{7,9,19}$={1,3,21,25,27}	D_{zl}={7,9,21} $S_{7,9,21}$={1,3,19,25,27}	D_{zm}={7,9,25} $S_{7,9,25}$={1,3,19,21,27}	D_{zn}={7,9,27} $S_{7,9,27}$={1,3,19,21,25}
D_{zo}={7,19,21} $S_{7,19,21}$={1,3,9,25,27}	D_{zp}={7,19,25} $S_{7,19,25}$={1,3,9,21,27}	D_{zq}={7,19,27} $S_{7,19,27}$={1,3,9,21,25}	D_{zr}={7,21,25} $S_{7,21,25}$={1,3,9,19,27}
D_{zs}={7,21,27} $S_{7,21,27}$={1,3,9,19,25}	D_{zt}={7,25,27} $S_{7,25,27}$={1,3,9,19,21}	D_{zu}={9,19,21} $S_{9,19,21}$={1,3,7,25,27}	D_{zv}={9,19,25} $S_{9,19,25}$={1,3,7,21,27}
D_{zw}={9,19,27} $S_{9,19,27}$={1,3,7,21,25}	D_{zx}={9,21,25} $S_{9,21,25}$={1,3,7,19,27}	D_{zy}={9,21,27} $S_{9,21,27}$={1,3,7,19,25}	D_{zz}={9,25,27} $S_{9,25,27}$={1,3,7,19,21}
D_{ya}={19,21,25} $S_{19,21,25}$={1,3,7,9,27}	D_{yb}={19,21,27} $S_{19,21,27}$={1,3,7,9,25}	D_{yc}={19,25,27} $S_{19,25,27}$={1,3,7,9,21}	D_{yd}={21,25,27} $S_{21,25,27}$={1,3,7,9,19}
Group T contains 70 combinations			
E_a={1,3,7,9} $T_{1,3,7,9}$={19,21,25,27}	E_b={1,3,7,19} $T_{1,3,7,19}$={9,21,25,27}	E_c={1,3,7,9} $T_{1,3,7,21}$={9,19,25,27}	E_d={1,3,7,25} $T_{1,3,7,25}$={9,19,21,27}
E_f={1,3,7,27}	E_g={1,3,9,19}	E_h={1,3,9,21}	E_i={1,3,9,25}

$T_{1,3,7,27}$={9,19,21,25}	$T_{1,3,9,19}$={7,21,25,27}	$T_{1,3,9,21}$={7,19,25,27}	$T_{1,3,9,25}$={7,19,21,27}
E_k={1,3,9,27} $T_{1,3,9,27}$={7,19,21,25}	E_l={1,3,19,21} $T_{1,3,19,21}$={7,9,25,27}	E_m={1,3,19,25} $T_{1,3,19,25}$={7,9,21,27}	E_n={1,3,19,27} $T_{1,3,19,27}$={7,9,21,25}
E_o={1,3,21,25} $T_{1,3,21,25}$={7,9,19,27}	E_p={1,3,21,27} $T_{1,3,21,27}$={7,9,19,25}	E_q={1,3,25,27} $T_{1,3,25,27}$={7,9,19,21}	E_r={1,7,9,19} $T_{1,7,9,19}$={3,21,25,27}
E_s={1,7,9,21} $T_{1,7,9,21}$={3,19,25,27}	E_t={1,7,9,25} $T_{1,7,9,25}$={3,19,21,27}	E_u={1,7,9,27} $T_{1,7,9,27}$={3,19,21,25}	E_v={1,3,7,9} $T_{1,7,19,21}$={3,9,25,27}
E_w={1,7,19,25} $T_{1,7,19,25}$={3,9,21,27}	E_x={1,7,19,27} $T_{1,7,19,27}$={3,9,21,25}	E_y={1,7,21,25} $T_{1,7,21,25}$={3,9,19,27}	E_z={1,7,21,27} $T_{1,7,21,27}$={3,9,19,25}
E_{za}={1,7,25,27} $T_{1,7,25,27}$={3,9,19,21}	E_{zb}={1,9,19,21} $T_{1,9,19,21}$={3,7,25,27}	E_{zc}={1,9,19,25} $T_{1,9,19,25}$={3,7,21,27}	E_{zd}={1,9,19,27} $T_{1,9,19,27}$={3,7,21,25}
E_{ze}={1,9,21,25} $T_{1,9,21,25}$={3,7,19,27}	E_{zf}={1,9,21,27} $T_{1,9,21,27}$={3,7,19,25}	E_{zg}={1,9,25,27} $T_{1,9,25,27}$={3,7,19,21}	E_{zh}={1,19,25,27} $T_{1,19,21,25}$={3,7,9,27}
E_{zi}={1,19,21,27} $T_{1,19,21,27}$={3,7,9,25}	E_{zj}={1,19,25,27} $T_{1,19,25,27}$={3,7,9,21}	E_{zk}={1,21,25,27} $T_{1,21,25,27}$={3,7,9,19}	E_{zl}={1,21,25,27} $T_{3,7,9,19}$={1,21,25,27}
E_{zm}={3,7,9,21} $T_{3,7,9,21}$={1,19,25,27}	E_{zn}={3,7,9,25} $T_{3,7,9,25}$={1,19,21,27}	E_{zo}={3,7,9,27} $T_{3,7,9,27}$={1,19,21,25}	E_{zp}={3,7,19,21} $T_{3,7,19,21}$={1,9,25,27}
E_{zq}={3,7,19,25} $T_{3,7,19,25}$={1,9,21,27}	E_{zr}={3,7,19,27} $T_{3,7,19,27}$={1,9,21,25}	E_{zs}={3,7,21,25} $T_{3,7,21,25}$={1,9,19,27}	E_{zt}={3,7,21,27} $T_{3,7,21,27}$={1,9,19,25}
E_{zu}={3,7,25,27} $T_{3,7,25,27}$={1,9,19,21}	E_{zv}={3,9,19,21} $T_{3,9,19,21}$={1,7,25,27}	E_{zw}={3,9,19,25} $T_{3,9,19,25}$={1,7,21,27}	E_{zx}={3,9,19,27} $T_{3,9,19,27}$={1,7,21,25}
E_{zy}={3,9,21,25} $T_{3,9,21,25}$={1,7,19,27}	E_{zz}={3,9,21,27} $T_{3,9,21,27}$={1,7,19,25}	E_{ya}={3,9,25,27} $T_{3,9,25,27}$={1,7,19,21}	E_{yb}={3,19,21,25} $T_{3,19,21,25}$={1,7,9,27}
E_{yc}={3,19,21,27} $T_{3,19,21,27}$={1,7,9,25}	E_{yd}={3,19,25,27} $T_{3,19,25,27}$={1,7,9,21}	E_{ye}={3,21,25,27} $T_{3,21,25,27}$={1,7,9,19}	E_{yf}={7,9,19,21} $T_{7,9,19,21}$={1,3,25,27}
E_{yg}={7,9,19,25} $T_{7,9,19,25}$={1,3,21,27}	E_{yh}={7,9,19,27} $T_{7,9,19,27}$={1,3,21,25}	E_{yi}={7,9,21,25} $T_{7,9,21,25}$={1,3,19,27}	E_{yj}={7,9,21,27} $T_{7,9,21,27}$={1,3,19,25}
E_{yk}={7,9,25,27} $T_{7,9,25,27}$={1,3,19,21}	E_{yl}={7,19,21,25} $T_{7,19,21,25}$={1,3,9,27}	E_{ym}={7,19,21,27} $T_{7,19,21,27}$={1,3,9,25}	E_{yn}={7,19,25,27} $T_{7,19,25,27}$={1,3,9,21}
E_{yo}={7,21,25,27} $T_{7,21,25,27}$={1,3,9,19}	E_{yp}={9,19,21,25} $T_{9,19,21,25}$={1,3,7,27}	E_{yq}={9,19,21,27} $T_{9,19,21,27}$={1,3,7,25}	E_{yr}={9,19,25,27} $T_{9,19,25,27}$={1,3,7,21}
E_{ys}={9,21,25,27} $T_{9,21,25,27}$={1,3,7,19}	E_{yt}={19,21,25,27} $T_{19,21,25,27}$={1,3,7,9}		
Group U contains 56 combinations			
F_a={1,3,7,9,19} $U_{1,3,7,9,19}$={21,25,27}	F_b={1,3,7,9,21} $U_{1,3,7,9,21}$={19,25,27}	F_c={1,3,7,9,25} $U_{1,3,7,9,25}$={19,21,27}	F_d={1,3,7,9,27} $U_{1,3,7,9,27}$={19,21,25}

F_e={1,3,7,19,21} $U_{1,3,7,19,21}$={9,25,27}	F_f={1,3,7,19,25} $U_{1,3,7,19,25}$={9,21,27}	F_g={1,3,7,19,27} $U_{1,3,7,19,27}$={9,21,25}	F_h={1,3,7,21,25} $U_{1,3,7,21,25}$={9,19,7}
F_i={1,3,7,21,27} $U_{1,3,7,21,27}$={9,19,25}	F_j={1,3,7,25,27} $U_{1,3,7,25,27}$={9,19,21}	F_k={1,3,9,19,21} $U_{1,3,9,19,21}$={7,25,27}	F_l={1,3,9,19,25} $U_{1,3,9,19,25}$={7,21,27}
F_m={1,3,9,19,27} $U_{1,3,9,19,27}$={7,21,25}	F_n={1,3,9,21,25} $U_{1,3,9,21,25}$={7,19,27}	F_o={1,3,9,21,27} $U_{1,3,9,21,27}$={7,19,25}	F_p={1,3,9,25,27} $U_{1,3,9,25,27}$={7,19,21}
F_q={1,3,19,21,25} $U_{1,3,19,21,25}$={7,9,27}	F_r={1,3,19,21,27} $U_{1,3,19,21,27}$={7,9,25}	F_s={1,3,19,25,27} $U_{1,3,19,25,27}$={7,9,21}	F_t={1,3,21,25,27} $U_{1,3,21,25,27}$={7,9,19}
F_u={1,7,9,19,21} $U_{1,7,9,19,21}$={3,25,27}	F_v={1,7,9,19,25} $U_{1,7,9,19,25}$={3,21,27}	F_w={1,7,9,19,27} $U_{1,7,9,19,27}$={3,21,25}	F_x={1,7,9,19,27} $U_{1,7,9,21,25}$={3,19,27}
F_y={1,7,9,21,27} $U_{1,7,9,21,27}$={3,19,25}	F_z={1,7,9,25,27} $U_{1,7,9,25,27}$={3,19,21}	F_{za}={1,7,19,21,25} $U_{1,7,19,21,25}$={3,9,27}	F_{zb}={1,7,19,21,27} $U_{1,7,19,21,27}$={3,9,25}
F_{zc}={1,7,19,25,27} $U_{1,7,19,25,27}$={3,9,21}	F_{zd}={1,7,21,25,27} $U_{1,7,21,25,27}$={3,9,19}	F_{ze}={1,9,19,21,25} $U_{1,9,19,21,25}$={3,7,27}	F_{zf}={1,9,19,21,27} $U_{1,9,19,21,27}$={3,7,25}
F_{zg}={1,9,19,25,27} $U_{1,9,19,25,27}$={3,7,21}	F_{zh}={1,9,21,25,27} $U_{1,9,21,25,27}$={3,7,19}	F_{zi}={1,19,21,25,27} $U_{1,19,21,25,27}$={3,7,9}	F_{zj}={3,7,9,19,21} $U_{3,7,9,19,21}$={1,25,27}
F_{zk}={3,7,9,19,25} $U_{3,7,9,19,25}$={1,21,27}	F_{zl}={3,7,9,19,27} $U_{3,7,9,19,27}$={1,21,25}	F_{zm}={3,7,9,21,25} $U_{3,7,9,21,25}$={1,19,27}	F_{zn}={3,7,9,21,27} $U_{3,7,9,21,27}$={1,19,25}
F_{zo}={3,7,9,25,27} $U_{3,7,9,25,27}$={1,19,21}	F_{zp}={3,7,19,21,25} $U_{3,7,19,21,25}$={1,9,27}	F_{zq}={3,7,19,21,27} $U_{3,7,19,21,27}$={1,9,25}	F_{zr}={3,7,19,25,27} $U_{3,7,19,25,27}$={1,9,21}
F_{zs}={3,7,21,25,27} $U_{3,7,21,25,27}$={1,9,19}	F_{zt}={3,9,19,21,25} $U_{3,9,19,21,25}$={1,7,27}	F_{zu}={3,9,19,21,2} $U_{3,9,19,21,27}$={1,7,25}	F_{zv}={3,9,19,25,27} $F_{3,9,19,25,27}$={1,7,21}
F_{zw}={3,9,21,25,27} $U_{3,9,21,25,27}$={1,7,19}	F_{zx}={3,19,21,25,27} $U_{3,19,21,25,27}$={1,7,9}	F_{zy}={7,9,19,21,25} $U_{7,9,19,21,25}$={1,3,27}	F_{zz}={7,9,19,21,27} $U_{7,9,19,21,27}$={1,3,25}
F_{ya}={7,9,19,25,27} $U_{7,9,19,25,27}$={1,3,21}	F_{yb}={7,9,21,25,27} $U_{7,9,21,25,27}$={1,3,19}	F_{yc}={7,19,21,25,27} $U_{7,19,21,25,27}$={1,3,9}	F_{yd}={9,19,21,25,27} $U_{9,19,21,25,27}$={1,3,7}
Group V contains 28 combinations			
F_a={1,3,7,9,19,21} $V_{1,3,7,9,19,21}$={25,27}	F_b={1,3,7,9,19,25} $V_{1,3,7,9,19,25}$={21,27}	F_c={1,3,7,9,19,27} $V_{1,3,7,9,19,27}$={21,25}	F_d={1,3,7,9,21,25} $V_{1,3,7,9,21,25}$={19,27}
F_e={1,3,7,9,21,27} $V_{1,3,7,9,21,27}$={19,25}	F_f={1,3,7,9,25,27} $V_{1,3,7,9,25,27}$={19,21}	F_g={1,3,7,19,21,25} $V_{1,3,7,19,21,25}$={9,27}	F_h={1,3,7,9,21,27} $V_{1,3,7,19,21,27}$={9,25}
F_i={1,3,7,19,25,27} $V_{1,3,7,19,25,27}$={9,21}	F_j={1,3,7,21,25,27} $V_{1,3,7,21,25,27}$={9,19}	F_k={1,3,9,19,21,25} $V_{1,3,9,19,21,25}$={7,27}	F_l={1,3,9,19,21,27} $V_{1,3,9,19,21,27}$={7,25}
F_m={1,3,9,19,25,27} $V_{1,3,9,19,25,27}$={7,21}	F_n={1,3,9,21,25,27} $V_{1,3,9,21,25,27}$={7,19}	F_o={1,3,19,21,25,27} $V_{1,3,19,21,25,27}$={7,9}	F_p={1,7,9,19,21,25} $V_{1,7,9,19,21,25}$={3,27}
F_q={1,7,9,19,21,27} $V_{1,7,9,19,21,27}$={3,25}	F_r={1,7,9,19,25,27} $V_{1,7,9,19,25,27}$={3,21}	F_s={1,7,9,21,25,27} $V_{1,7,9,21,25,27}$={3,19}	F_t={1,7,19,21,25,27} $V_{1,7,19,21,25,27}$={3,9}
F_u={1,9,19,21,25,27}	F_v={3,7,9,19,21,25}	F_w={1,3,9,19,25,27}	F_x={3,7,9,19,25,27}

$V_{1,9,19,21,25,27}$={3,7}	$V_{3,7,9,19,21,25}$={1,27}	$V_{3,7,9,19,21,27}$={1,25}	$V_{3,7,9,19,25,27}$={1,21}
F_y={3,7,9,21,25,27} $V_{3,7,9,21,25,27}$={1,19}	F_z={3,7,19,21,25,27} $V_{3,7,19,21,25,27}$={1,9}	F_{za}={3,9,19,21,25,27} $V_{3,9,19,21,25,27}$={1,7}	F_{zb}={7,9,19,21,25,27} $V_{7,9,19,21,25,27}$={1,3}
Group W Contains 8 combinations			
G_a={1,3,7,9,19,21,25} $W_{1,3,7,9,19,21,25}$={27}	G_b={1,3,7,9,19,21,27} $W_{1,3,7,9,19,21,27}$={25}	G_c={1,3,7,9,19,25,27} $W_{1,3,7,9,19,25,27}$={21}	G_d={1,3,7,9,21,25,27} $W_{1,3,7,9,21,25,27}$={19}
G_e={1,3,7,19,21,25,27} $W_{1,3,7,19,21,25,27}$={9}	G_f={1,3,9,19,21,25,27} $W_{1,3,9,19,21,25,27}$={7}	G_g={1,7,9,19,21,25,27} $W_{1,7,9,19,21,25,27}$={3}	G_h={3,7,9,19,21,25,27} $W_{3,7,9,19,21,25,27}$={1}
Group X 1 combination			
H_a={1,3,7,9,19,21,25,27 **X** $_{1,3,7,9,19,21,25,27}$={ }			

The visualization of 256 convex polyhedrons which are listed in the above table is shown in Fig. 5.

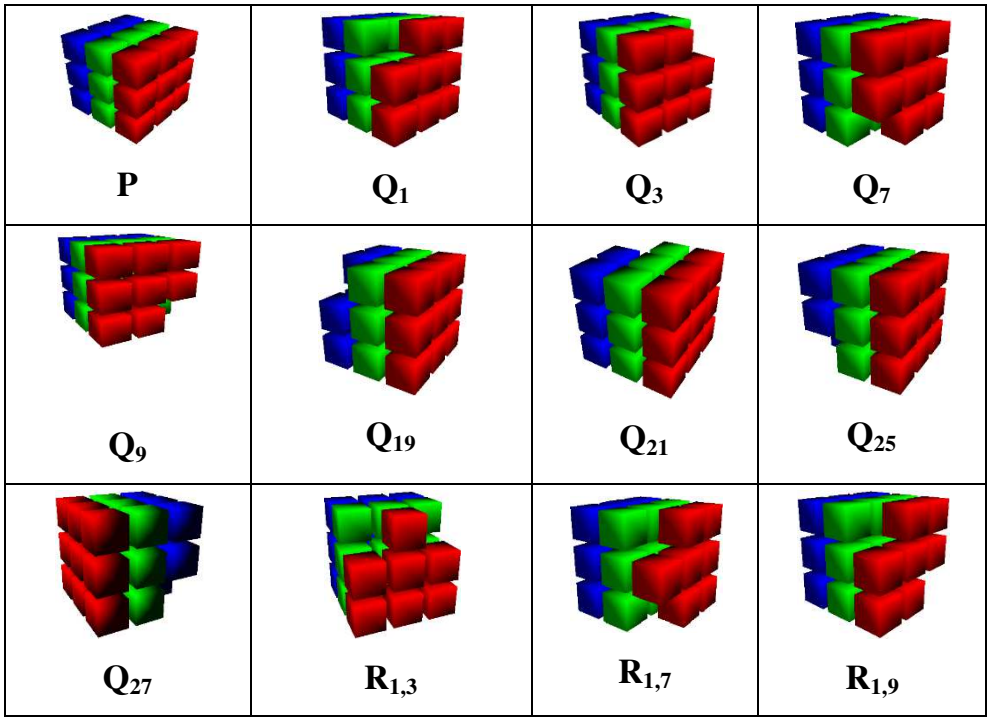


Fig. 5. Three Dimensional visualizations of convex polyhedrons

3.2 Feature Extraction

Extracting the features from the image is an important step in object recognition. In the proposed method we used picture description language for representation of the features. The image is traced pixel by pixel which starts from pixel position (1,1,1) to find the initial foreground pixel

(x_i, $y_{i,}$ z_i) with the intensity value 1. This will identify the initial point of the first component present in the image. The proposed algorithm is written in such a way that it will find the next neighbor of the current pixel in all the preferred direction. Preference will be generally given to the right side of the pixel (from the current pixel) These 26 directions are already represented in Figure 2. The naming conventions of the preferred direction are: Right(R), Down Right (DR), Down (D), Down Left (DL), Left (L), Up left (UL), Up (U), and Up Right (UR), Back(B), Back Right(BR), Back Down Right (BDR), Back Down (BD), Back Down Left (BDL), Back Left (BL), Back-Up left (BUL), Back-Up (BU), and Back-Up Right (BUR), Front(F), Front Right(FR), Front Down Right (FDR), Front Down (FD), Front Down Left (FDL), Front Left (FL), Front Up left (FUL), Front Up (FU), and Front Up Right (FUR). In this method, the previously recognized direction has given priority compared to other preferred directions. The tracing of each pixel will continue until the algorithm will not find any connected neighbor pixel to the current position (x, y) and will remove the pixel which is already traced. If there is no neighbor pixel then the algorithm will display the knowledge vector of that component and again algorithm will continue until it will not trace all the components which are present in the image. If all the components are traced then the final knowledge vector will be displayed. This knowledge vector is nothing but a feature vector that consists of the direction and length code of each component and this knowledge vector will give the information that some objects are present in the image.

Algorithm for creating the knowledge vector from the given image.

Input: Given Image

Empty loader

Output: Knowledge vector

```
Begin
Initialize
    Dimension of the image
i,j,k=dim
length = []
    direction = []
arr= []
loader=[]
      image=[]
     arr2=[]
     arr_i=[]
     arr_j=[]
     arr_k=[]
Assign the dimension values to the variable dim.
#Create an empty container of the same size.
Create a 3D array (img) of the same size as the
dimension variable and initialize it as zero.
Give the path of the dataset and read the images
from the dataset and store the value in array im.
  for k in range(dim):
   for i in range(dim):
      for j in  range(dim):
          if im[i][j][k]!=0
             then assign  img[m][i+1][j+1]==255
```

```
 store_value  function(i, j, k)
    append
    i value in arr_i
    j value in  arr_j
    k value in  arr_k
Traversing function(x, y, z)
     Initialize the x, y, z value to  i, j, k
     Initialize the img array as 0
     Call Store_value (i, j, k)
     Initialize the loader as [0 0]
 While (image[k, i, j]!=255)
#Right direction
   if img(k, i, j+1)==255:
    if R\□ is available in loader  then
     image [k, i, j+1] = 0
     j=j+1
     call store(i, j, k)
     Loader[1]=loader[1]+1
     else:
     append the direction at loader[0]
     append the length with {}\□ at loader [1]
     clear the loader
     append   R\□
     append 0 to the loader
 #Down right direction
    else if img(k, i+1, j+1)==255:
     if DR\□ is available in loader  then
     image [k, i, j+1] = 0
     j=j+1
     call store(i, j, k)
     Loader[1]=loader[1]+1
     else:
     append the direction at loader[0]
     append the length with {}\□ at loader [1]
     clear the loader
     append   DR\□
     append 0 to the loader
 #Down direction
    else if img(k, i+1, j)==255:

     if D\□ is available in loader  then
     image [k, i, j+1] = 0
     j=j+1
     call store(i, j, k)
     Loader[1]=loader[1]+1
     else:
     append the direction at loader[0]
     append the length with {}\□ at loader [1]
     clear the loader
     append   D\□
     append 0 to the loader
#Down left direction
   else if img(k, i+1, j-1)==255:
     if DL\□ is available in loader  then
     image [k, i, j+1] = 0
     j=j+1
     call store(i, j, k)
     Loader[1]=loader[1]+1
     else:
     append the direction at loader[0]
     append the length with {}\□ at loader [1]
     clear the loader
     append   DL\□
     append 0 to the loader
#Left direction
    else if img(k, i, j-1)==255:
     if L\□ is available in loader  then
     image [k, i, j+1] = 0
     j=j+1
     call store(i, j, k)
     Loader[1]=loader[1]+1
     else:
     append the direction at loader[0]
     append the length with {}\□ at loader [1]
     clear the loader
     append   L\□
     append 0 to the loader
 #up left direction
    else if img(k, i-1, j-1)==255:
     if UL\□ is available in loader  then
     image [k, i, j+1] = 0
     j=j+1
     call store(i, j, k)
     Loader[1]=loader[1]+1
     else:
     append the direction at loader[0]
     append the length with {}\□ at loader [1]
     clear the loader
     append   UL\□
     append 0 to the loader
 #Up direction
    else if img(k, i-1, j)==255:
     if U\□ is available in loader  then
     image [k, i, j+1] = 0
     j=j+1
     call store(i, j, k)
     Loader[1]=loader[1]+1
     else:
```

```
        append the direction at loader[0]
        append the length with {}\□ at loader [1]
        clear the loader
        append   U\□
        append 0 to the loader
#Up Right direction
    else if img(k, i-1, j+1)==255:
        if UR\□ is available in loader  then
        image [k, i, j+1] = 0
        j=j+1
        call store(i, j, k)
        Loader[1]=loader[1]+1
        else:
        append the direction at loader[0]
        append the length with {}\□ at loader [1]
        clear the loader
        append   UR\□
        append 0 to the loader
#Back direction
    else if img(k+1, i, j)==255:
        if B\□ is available in loader  then
        image [k, i, j+1] = 0
        j=j+1
        call store(i, j, k)
        Loader[1]=loader[1]+1
        else:
        append the direction at loader[0]
        append the length with {}\□ at loader [1]
        clear the loader
        append   B\□
        append 0 to the loader
 #Back right direction
    else if img(k+1, i, j+1)==255:
        if BR\□ is available in loader  then
        image [k, i, j+1] = 0
        j=j+1
        call store(i, j, k)
        Loader[1]=loader[1]+1
        else:
        append the direction at loader[0]
        append the length with {}\□ at loader [1]
        clear the loader
        append   U\□
        append 0 to the loader
 #Back down right direction
    else if img(k+1, i+1, j+1)==255:
        if BDR\□ is available in loader  then
        image [k, i, j+1] = 0
        j=j+1
        call store(i, j, k)
        Loader[1]=loader[1]+1
        else:
        append the direction at loader[0]
        append the length with {}\□ at loader [1]
        clear the loader
        append   BDR\□
        append 0 to the loader
# Back down direction
   else if img(k+1, i+1, j)==255:
        if BD\□ is available in loader  then
        image [k, i, j+1] = 0
        j=j+1
        call store(i, j, k)
        Loader[1]=loader[1]+1
        else:
        append the direction at loader[0]
        append the length with {}\□ at loader [1]
        clear the loader
        append   BD\□
        append 0 to the loader
# Back down left direction
   else if img(k+1, i+1, j-1)==255:
        if BDL\□ is available in loader  then
        image [k, i, j+1] = 0
        j=j+1
        call store(i, j, k)
        Loader[1]=loader[1]+1
        else:
        append the direction at loader[0]
        append the length with {}\□ at loader [1]
        clear the loader
        append   BDL\□
        append 0 to the loader
# Back left direction
   else if img(k+1, i, j-1)==255:
        if BL\□ is available in loader  then
        image [k, i, j+1] = 0
        j=j+1
        call store(i, j, k)
        Loader[1]=loader[1]+1
        else:
        append the direction at loader[0]
        append the length with {}\□ at loader [1]
        clear the loader
        append   BL\□
        append 0 to the loader
# Back up left direction
   else if img(k+1, i-1, j-1)==255:
```

```
        if BUL\□ is available in loader  then
        image [k, i, j+1] = 0
        j=j+1
        call store(i, j, k)
        Loader[1]=loader[1]+1
        else:
        append the direction at loader[0]
        append the length with {}\□ at loader [1]
        clear the loader
        append   BUL\□
        append 0 to the loader
# Back up direction
    else if img(k+1, i-1, j)==255:
        if BU\□ is available in loader  then
        image [k, i, j+1] = 0
        j=j+1
        call store(i, j, k)
        Loader[1]=loader[1]+1
        else:
        append the direction at loader[0]
        append the length with {}\□ at loader [1]
        clear the loader
        append   BU\□
        append 0 to the loader
# Back up right direction
    else if img(k+1, i-1, j+1)==255:
        if BUR\□ is available in loader  then
        image [k, i, j+1] = 0
        j=j+1
        call store(i, j, k)
        Loader[1]=loader[1]+1
        else:
        append the direction at loader[0]
        append the length with {}\□ at loader [1]
        clear the loader
        append   BUR\□
        append 0 to the loader
# Forward direction
    else if img(k-1, i, j)==255:
        if F\□ is available in loader  then
        image [k, i, j+1] = 0
        j=j+1
        call store(i, j, k)
        Loader[1]=loader[1]+1
        else:
        append the direction at loader[0]
        append the length with {}\□ at loader [1]
        clear the loader
        append   F\□
        append 0 to the loader
# Forward Right direction
    else if img(k-1, i, j+1)==255:
        if FR\□ is available in loader  then
        image [k, i, j+1] = 0
        j=j+1
        call store(i, j, k)
        Loader[1]=loader[1]+1
        else:
        append the direction at loader[0]
        append the length with {}\□ at loader [1]
        clear the loader
        append   FR\□
        append 0 to the loader
#Forward down right  direction
    else if img(k-1, i+1, j+1)==255:
        if FDR\□ is available in loader  then
        image [k, i, j+1] = 0
        j=j+1
        call store(i, j, k)
        Loader[1]=loader[1]+1
        else:
        append the direction at loader[0]
        append the length with {}\□ at loader [1]
        clear the loader
        append   FDR\□
        append 0 to the loader
# Forward down direction
    else if img(k-1, i+1, j)==255:
        if FD\□ is available in loader  then
        image [k, i, j+1] = 0
        j=j+1
        call store(i, j, k)
        Loader[1]=loader[1]+1
        else:
        append the direction at loader[0]
        append the length with {}\□ at loader [1]
        clear the loader
        append   FD\□
        append 0 to the loader
# Forward down left direction
    else if img(k-1, i+1, j-1)==255:
        if FDL\□ is available in loader  then
        image [k, i, j+1] = 0
        j=j+1
        call store(i, j, k)
        Loader[1]=loader[1]+1
        else:
        append the direction at loader[0]
```

```
        append the length with {}\□ at loader [1]
        clear the loader
        append   FDL\□
        append 0 to the loader
# Forward left direction
    else if img(k-1, i, j-1)==255:
        if FL\□ is available in loader  then
        image [k, i, j+1] = 0
        j=j+1
        call store(i, j, k)
        Loader[1]=loader[1]+1
        else:
        append the direction at loader[0]
        append the length with {}\□ at loader [1]
        clear the loader
        append   FL\□
        append 0 to the loader
# Forward up left direction
    else if img(k-1, i-1, j-1)==255:
        if FUL\□ is available in loader  then
        image [k, i, j+1] = 0
        j=j+1
        call store(i, j, k)
        Loader[1]=loader[1]+1
        else:
        append the direction at loader[0]
        append the length with {}\□ at loader [1]
        clear the loader
        append   FUL\□
        append 0 to the loader
# Forward up direction
    else if img(k-1, i-1, j)==255:
        if FU\□ is available in loader  then
        image [k, i, j+1] = 0
        j=j+1
        call store(i, j, k)
        Loader[1]=loader[1]+1
        else:
        append the direction at loader[0]
        append the length with {}\□ at loader [1]
        clear the loader
        append   FU\□
        append 0 to the loader

# Forward up Right direction
    else if img(k-1, i-1, j+1)==255:
        if FUR\□ is available in loader  then
        image [k, i, j+1] = 0
        j=j+1
        call store(i, j, k)
        Loader[1]=loader[1]+1
        else:
        append the direction at loader[0]
        append the length with {}\□ at loader [1]
        clear the loader
        append   FUR\□
        append 0 to the loader
# Back down left direction
    else if img(k+1, i+1, j-1)==255:
        if BDL\□ is available in loader  then
        image [k, i, j+1] = 0
        j=j+1
        call store(i, j, k)
        Loader[1]=loader[1]+1
        else:
        append the direction at loader[0]
        append the length with {}\□ at loader [1]
        clear the loader
        append   BDL\□
        append 0 to the loader
    else
      Break
   append the direction at loader[0])
append the length with {}\□ at loader [1]
clear the loader
pop up the direction
pop up the length
add the direction to the arr
append "*" to arr
add the length to arr
Append   the i, j, k value in this format
<{},{},{}>
Clear the direction array
   Clear the length array

for k in range (dim):
    for i in range (dim):
        for j in range(dim):
            if img[k][i][j]=255:
              Then append < {}, {}, {}> in arr call
traversing function by passing the value of i, j
and k.  (traversing(i,j,k))
Initialize str=''
for element in arr:
      str=str+ element
open a text file and write the content of str into
the text file
Close the text file.
end
```

The above algorithm is applied on a simple cube and knowledge vector is extracted and it is represented futher:

<1,1,1>R□D□L□U□BU□B□R□D□L□U□*99□99□99□98□1□98□99□99□99□98□<2,1,100><1,100,2>B□*97□<1,100,99><100,1,2>B□*97□<100,1,99><100,100,2>B□*97□<100,100,99>

The part before the * symbol is called direction code (DC) and the following part is called length code (LC).

3.3 Reconstruction of original Three-dimensional Image using Knowledge Vector

In the previous step, the knowledge vector of the image is identified which is nothing but features of the image. In this step, the feature vector is considered as an input and a novel algorithm is written which converts the knowledge vector into the image. Further, the reconstruction algorithm is explained in details:

Algorithm for Reconstruction of the image using knowledge vector:

Input: Text file
Output: Image

Begin
Read a text file and store it in an **str array**
Declare array p, q and r, and s, x, and str1
Read the value of **str** and replace "><" with "---" and store in **str1** array
Split the array values on basis of this "---" and store the result in **x**
for i in the range of(len(x)):
do the following replacement in **x** and again keep the value in **x**
replace ">" with "*"
replace "□<" with "*"
replace "<" with ""
replace "□", with ","
Declare 3 variables with the name Direction, Length, and Start.
For i in range(length(x)):
a=x[i]
Split the value of array **a** on basis of "*" and store in result in **an** again.
If the length of the array is equal to 1 or 3 then:
Assign the value of a[0] to **Start**.
Split the value of array **Start** on basis of "," and store in result in **Start** again.
Assign the value of **Start [0], Start [1],** and **Start [2]** to i, j, and k.
Append the value of j, i, k in p, q, r.
If the length of a is equal to 1
Then print the value of a
else
Assign a[0] value to **Start** Array.
Assign a[1] value to the **Direction** array.
Assign a[2] value to **the Length** array.
Assign a[3] value to the **End** array.
Call **Printer Function** bypassing Direction, Length, and **Start** Parameter.
Display the 3D Image on Screen Using graph plot.
Printer function(Direction, Length, Start):
Split the **Start** array on basis of "," and store the resultant array in **the Start** array.
Split the **Length** array on basis of "," and store the resultant array in **the Length** array.
Split the **Direction** array on basis of "," and store the resultant array in **the Direction** array.
Store the Start [0] value in variable i.

```
    Store the Start [1] value in variable j.
  Store the Start [2] value in variable k.
Now append the value of I, j, k in p, q, r.
Display the value of Length array and Direction
array.
  for m in range(Length(Direction)-1):
     Calculate the Length[m] and store the result
in variable L.
   #Right Direction
       If Direction[m]==”R”
          z=j+L
         while(j<z):
         Increment j value by 1.
        append the value of j, i,  k in p, q, r.
    #Down Direction
      Else If Direction[m]==”D”
           z=i+L
           while(i<z):
           Increment i value by 1.
       Append the value of j, i,  k in p, q, r.
   #Left Direction
     Else If Direction[m]==”L”
          z=j-L
         while (j>z):
         Decrement j value by 1.
        Append the value of j, i, k in p, q, r.
    #up Direction
     Else If Direction[m]==”U”
         z=i-L
         while(i>z):
         Decrement i value by 1.
        Append the value of j, i, k in p, q, r.
   #Down right Direction
        Else If Direction[m]==”DR”
        z=j+L
         while(j<z):
         Increment i & j value by 1.
        Append the value of j, i, k in p, q, r.
   #Down Left Direction
        Else If Direction[m]==”DL”
        z=i+L
        while(i<z):
        Increment i value by 1.
        Decrement j value by 1.
       Append the value of j, i, k in p, q, r.
    #up Left Direction
        Else If Direction[m]==”UL”
        z=j-L
       while(j>z):
       Decrement i & j value by 1.
       Append the value of j, i, k in p, q, r.
```

```
    #up right Direction
       Else If Direction[m]==”UR”
        z=i-L
        while(i>z):
        Increment j value by 1.
        Decrement i value by 1.
       Append the value of j, i, k in p, q, r.
  #Back Direction
       Else If Direction[m]==”B”
        z=k+L
        while(k<z):
        Increment k value by 1.
       Append the value of j, i, k in p, q, r.
    #Back right Direction
       Else If Direction[m]==”BR”
        z=j+L
        while(j<z):
        Increment j  and k value by 1.
        Append the value of j, i, k in p, q, r.
  #Back down Direction
        Else If Direction[m]==”BD”
        z=i+L
        while(i<z):
        Increment i and k value by 1.
       Append the value of j, i, k in p, q, r.
  #Back Left Direction
        Else If Direction[m]==”BL”
         z=j-L
        while(j>z):
        Increment k value by 1.
        Decrement j value by 1
       Append the value of j, i, k in p, q, r.
  #Back up Direction
        Else If Direction[m]==”BU”
         z=i-L
        while(i>z):
       Increment k value by 1.
       Decrement i value by 1
       Append the value of j, i, k in p, q, r.
#Back down right Direction
        Else If Direction[m]==”BDR”
         z=j+L
         while(j<z):
         Increment i, j and k value by 1.
         Decrement i value by 1
        Append the value of j, i, k in p, q, r.
#Back down left Direction
        Else If Direction[m]==”BDL”
        z=i+L
        while(i<z):
        Increment i and k value by 1.
```

```
        Decrement j value by 1
        Append the value of j, i, k in p, q, r.
  #Back up left Direction
        Else If Direction[m]==”BUL”
        z=j-L
        while(j>z):
        Increment k value by 1.
        Decrement i and j value by 1
        Append the value of j, i, k in p, q, r.
  #Back up Right Direction
        Else If Direction[m]==”BUR”
         z=i-L
         while(i>z):
         Increment j and k value by 1.
         Decrement i value by 1.
         Append the value of j, i, k in p, q, r.
  #Forward Direction
        Else If Direction[m]==”F”
         z=k-L
      while(k>z):
      Decrement k value by 1
      Append the value of j, i, k in p, q, r.
#Forward Right Direction
      Else If Direction[m]==”FR”
       z=j+L
       while(j<z):
       Increment j value by 1.
       Decrement k value by 1
      Append the value of j, i, k in p, q, r.
 #Forward Down Direction
       Else If Direction[m]==”FD”
       z=i+L
       while(i<z):
       Increment i value by 1.
       Decrement k value by 1
      Append the value of j, i, k in p, q, r.
#Forward left Direction
      Else If Direction[m]==”FL”
       z=j-L
      while(j>z):
```

```
        Decrement j and k value by 1
      Append the value of j, i, k in p, q, r.
 #Forward up Direction
       Else If Direction[m]==”FU”
        z=i-L
        while(i>z):
       Decrement i and k value by 1
       Append the value of j, i, k in p, q, r.
  #Forward down right Direction
       Else If Direction[m]==”FDR”
       z=j+L
       while(j<z):
       Increment i and j value by 1.
      Decrement k value by 1
      Append the value of j, i, k in p, q, r.
 #Forward down left Direction
      Else If Direction[m]==”FDL”
      z=i+L
      while(i<z):
     Increment i value by 1.
     Decrement j and k value by 1
     Append the value of j, i, k in p, q, r.
#Forward up left Direction
    Else If Direction[m]==”FUL”
      z=j-L
      while(j>z):
     Decrement i ,j and k value by 1
     Append the value of j, i, k in p, q, r.
 #Forward up right Direction
     Else If Direction[m]==”FUR”
      z=i-L
     while(i>z):
     Increment j value by 1.
     Decrement i and k value by 1
    Append the value of j, i, k in p, q, r.
END
```

4. Experiments setup and its parameters

Two algorithms are implemented for the proposed work and both algorithms are implemented in python using the Anaconda tool. The experiments are done on a laptop with 16GB RAM, 4GB NVIDIA Graphics card, i7 processors, and windows operating system.

4.1 Datasets

The performance of the proposed approach is evaluated on five datasets and all five datasets are collected from Pentagram research institute, Hyderabad. All five datasets are medical images and a detailed description is given further:

Table II: Description of datasets

Sr. No.	Dataset Name	Details of MRI Image	Remarks
1.	Toutatix	No. of Images = 256 Width = 256 Height = 256 Depth = 256 Max Gray level: 256	Set of 2-D slices
2.	Cerebrix	No. of Images = 43 Width = 256 Height = 256 Depth = 43 Max Gray level: 256	Set of 2-D slices
3.	Cetautomatix	No of Images = 256 Width = 256 Height = 256 Depth = 256 Max Gray level:114	A single file containing 3-D Data
4.	MRI	No of Images = 256 Width = 256 Height = 256 Depth = 256 Max Max Gray level: 181	A single file containing 3-D Data
5.	Cell Farct	No of Images = 256 Width = 256 Height = 256 Depth = 256 Max Gray level: 256	A single file containing 3-D Data

4.2 Evaluation Parameters

To evaluate the performance of the proposed methodology some standard image processing evaluation parameters are considered. The First parameter is Accuracy which means how

accurately the algorithm can find the feature vector of the image and at the same time how accurately the algorithm can reconstruct the 3D image using knowledge vector.

The other most common parameters of evaluating the quality of image reconstruction are Peak signal to noise ratio (PSNR), Mean squared error (MSE), Structure similarity index(SSIM). These parameters work on the pixel-wise comparison between the original image and the reconstructed image. PSNR (based on the MSE metric) is a ratio between the maximum possible power of a signal and the power of corrupting noise that affects the fidelity of its representation. The power of corrupting noise is measured by MSE and SSIM is considered as the mean and variance of the image intensities. PSNR is defined as:

$$PSNR = 10.log_{10}\frac{MAX_I^2}{MSE} \quad (1)$$

Where

MAX_I is the maximum pixel value of the image and

$$MSE(f,g) = \frac{1}{mn}\sum_{i}^{m-1}\sum_{j}^{n-1}(f(i,j) - g(i,j))^2 \quad (2)$$

The other parameter is the execution time. The execution time is nothing but the time which is required for the reconstruction of the original image using a knowledge vector. To calculate the execution time following formula is used.

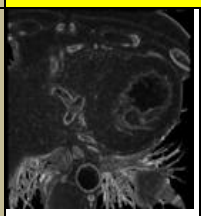
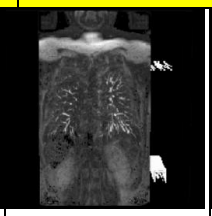
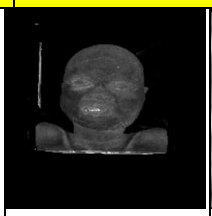
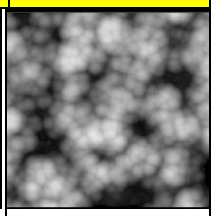
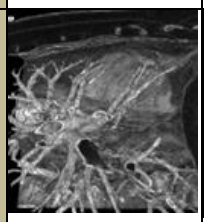
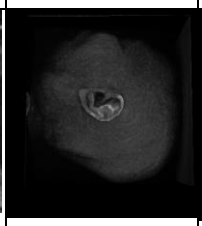
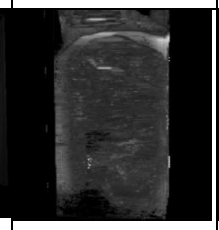
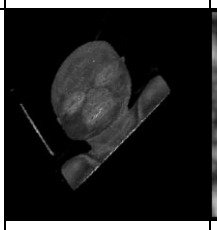
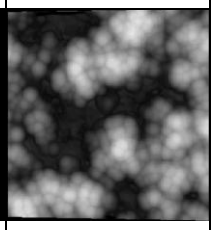
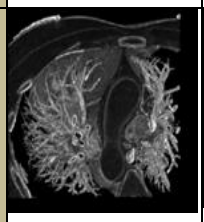
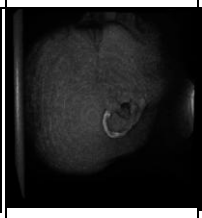
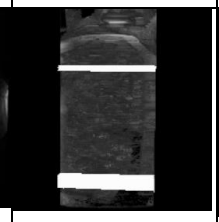
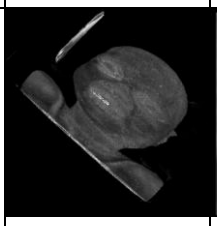
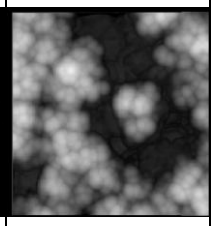
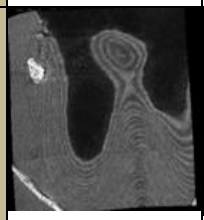
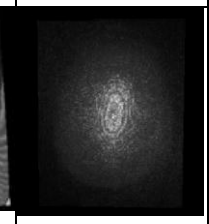
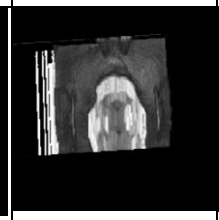
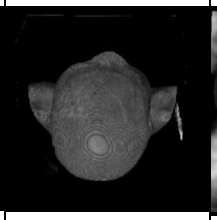
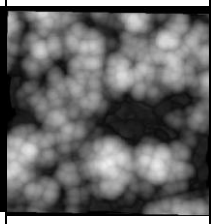
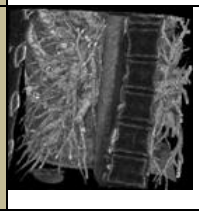
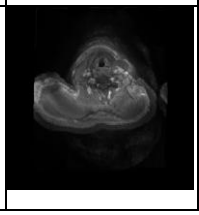
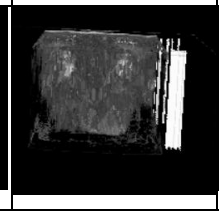
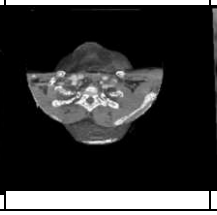
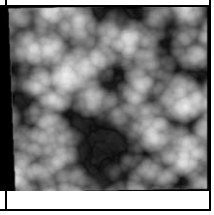
Execution time=1 / (Performance of the Processor) (3)

5. Results

The performance evaluation of any algorithm always depends on the input dataset. Five different datasets are used for the evaluation of the algorithm and different parameters are chosen for the evaluation purpose. The quality of the reconstructed image is represented in the table and it is visible in some images that reconstructed images are more clear and having sharp edges because the first algorithm works based on convex polyhedrons and our algorithm can extract the features in such a way that it is not affecting the quality of the image while reconstruction. PSNR and SSIM are calculated on all five datasets and it is represented in table IV. The other evaluation parameters are execution time and Accuracy and it is represented in table V and figure 6. The

accuracy of the proposed method is 94.78% and the average execution time is 6.76 seconds which is better than state of art methods.

Table III. Reconstruction of Image on different datasets.

	Reconstructed Image				
	Toutatix	**Cerebrix**	**Cetautomatix**	**MRI**	**Cell Farct**
Front View					
Left View					
Right View					
Top View					
Bottom View					

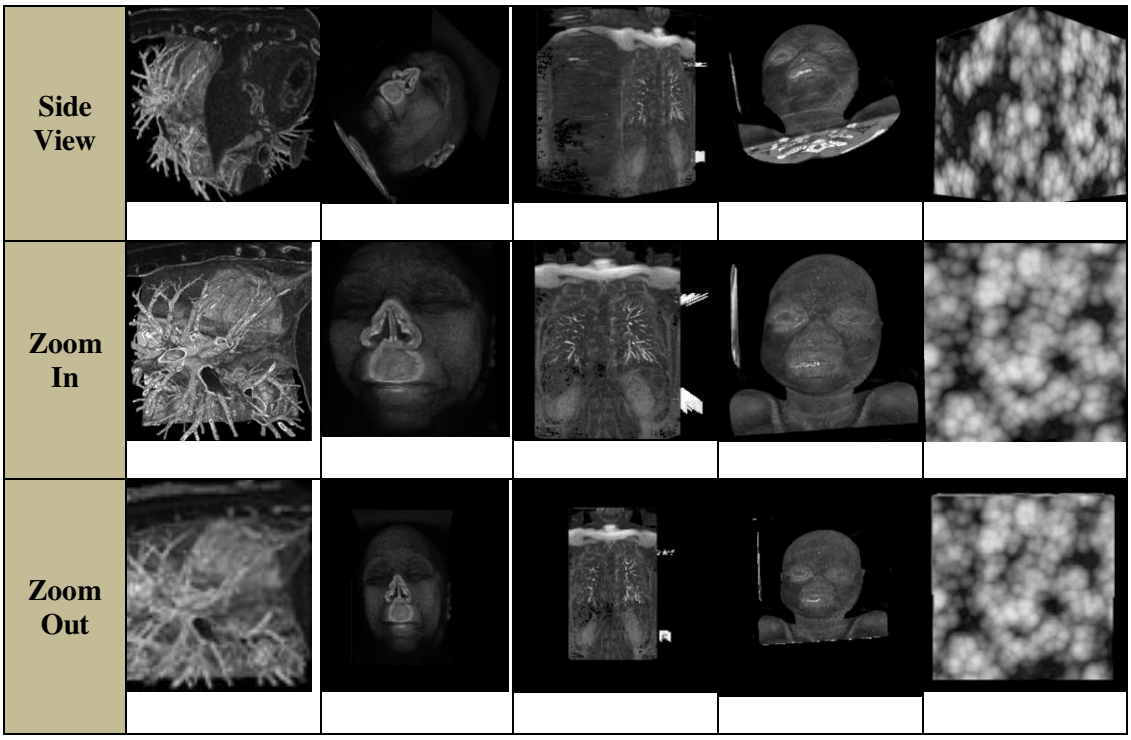


Table IV. PSNR and SSIM value on different medical datasets.

Dataset	PSNR	SSIM
Toutatix	**32.59±1.20**	**0.93±0.01**
Cerebrix	**31.67±1.47**	**0.91±0.03**
Cetautomatix	**29.86±1.20**	**0.87±0.03**
MRI	**32.57±1.20**	**0.90±0.01**
Cell Farct	**31.26±1.80**	**0.89±0.01**

Table V. Comparative analysis between proposed work and other state of art methods in terms of accuracy and execution time.

	Accuracy	Execution Time
Traditional Method [29]	62.84%	7.12
Baijiang Fan [30]	90.52%	12.51
3D-AlexNet [31]	92.11%	-
Inception-V4 [31]	88.01%	-
ResNet 50 [31]	85.8%	-

Proposed Work	94.78%	6.76

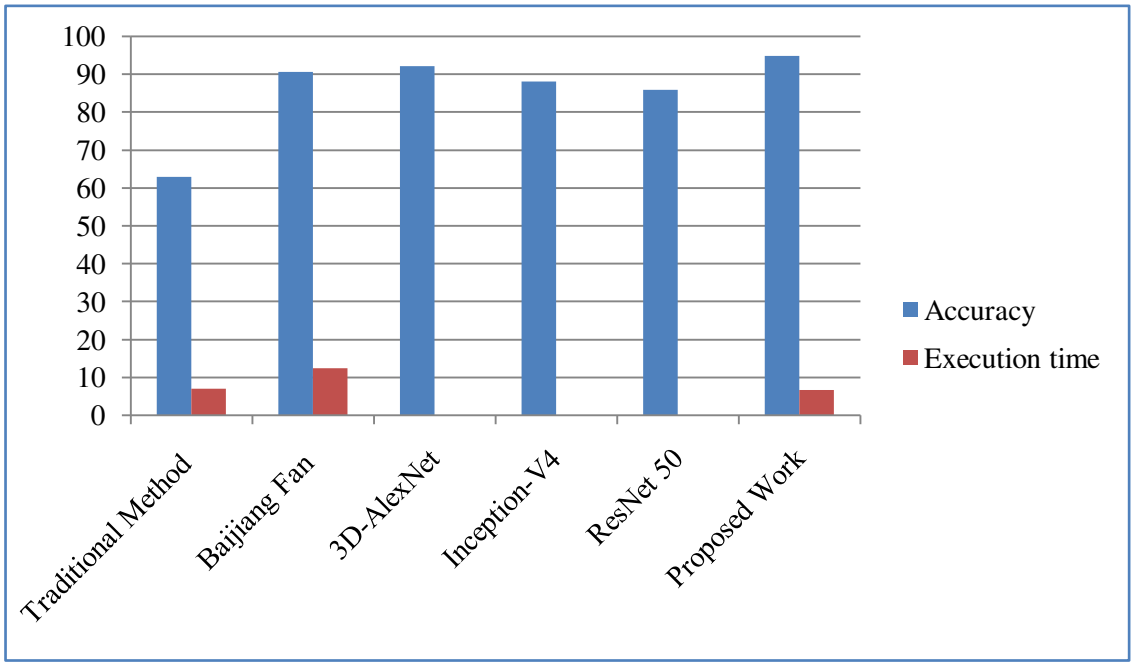


Fig 6. Comparative analysis of accuracy and execution time

6. Conclusion and Future work

In this paper, we have presented a syntactic pattern recognition based approach which can represent the image into textual form and a novel algorithm is proposed for the reconstruction of three-dimensional images. Reconstruction allows us to explore the internal details of the 3D images such as the size, shape, and structure of the object which could take us one step ahead in the field of medical image processing. GUI is prepared for the proper visualization of the image and the GUI is having features of rotating the image in all possible direction and zoom in and zoom out feature is also added in GUI so that internal details of the image can be visualized properly. Performances of the proposed algorithms are evaluated on a medical image dataset and results are outperformed in real-time. The accuracy and execution time results of the proposed methodology are 94.78 % and 6.76 sec respectively which are better than the existing state of art methodology. This approach is better than other existing approaches because there are very few steps of reconstruction of a 3D image which makes this approach unique.

To speed up the reconstruction process, the more advanced architecture such as GPU (Graphical Processing Unit) can be used in the future. The classification task of a structural pattern recognition system is difficult to implement because the syntactic pattern recognition embody the precise criteria which discriminate among groups and, therefore, they are by their very nature domain- and application-specific. In the future, we will try to extend the paper by implementing classification techniques for object recognition.

Acknowledgment

The constant support of Prof. Dr. E. G. Rajan, President, Pentagram Research, and Director, Avatar MedVision US LLC NC USA has resulted to bring the results of the research paper. The authors express their sincere thanks to Mr. Pawan Rao, a Data Scientist in the Pentagram Research Centre, and Hyderabad for his valuable suggestion during the preparation of this research paper.

Declarations

Funding

Not applicable

Conflict of interest

On behalf of all authors, the corresponding author states that there is no conflict of interest.

Availability of data and material

Data and other related material is available.

Code availability

Implementation of algorithm is available.

References

[1] Richard O. Duda, Peter E. Hart, and David E. Stork. Pattern Classification. Wiley, New York, second edition, 2001.

[2] Keinosuke Fukunaga. Introduction to Statistical Pattern Recognition. Academic Press, Boston, second edition, 1990.

[3] Anil K. Jain, Robert P. W. Duin, and Jianchang Mao. Statistical Pattern Recognition: A Review. IEEE Transactions on Pattern Analysis and Machine Intelligence, 22 (1): 4 –37, January 2000.

[4] K. S. Fu. Syntactic Pattern Recognition and Applications. Prentice-Hall, Englewood Cliffs, New Jersey, 1982.

[5] Rafael C. Gonzalez and Michael G. Thomason. Syntactic Pattern Recognition: An Introduction. Addison-Wesley, Reading, Massachusetts, 1978.

[6] T. Pavlidis. Structural Pattern Recognition. Springer-Verlag, Berlin, 1977.

[7] Li, Yinsheng, and Guang-Hong Chen. "An Empirical Data Inconsistency Metric (DIM) Driven CT Image Reconstruction Method." IEEE Transactions on Medical Imaging 38, no. 2 (2018): 337-348.

[8] Bustin, Aurelien, Damien Voilliot, Anne Menini, Jacques Felblinger, Christian de Chillou, Darius Burschka, Laurent Bonnemains, and Freddy Odille. "Isotropic reconstruction of MR images using 3D patch-based self-similarity learning." IEEE Transactions on Medical Imaging, vol. 37, no. 8, pp. 1932-1942, 2018.

[9] Hou, Benjamin, Bishesh Khanal, Amir Alansary, Steven McDonagh, Alice Davidson, Mary Rutherford, Jo V. Hajnal, Daniel Rueckert, Ben Glocker, and Bernhard Kainz. "3-D reconstruction in canonical coordinate space from arbitrarily oriented 2-D images." IEEE transactions on medical imaging, vol. 37, no. 8, pp. 1737-1750, 2018.

[10] Benalcazar, Daniel P., Diego Bastias, Claudio A. Perez, and Kevin W. Bowyer, "A 3D Iris Scanner From Multiple 2D Visible Light Images," IEEE Access 7 (2019): 61461-61472.

[11] Zheng, Xuehang, Saiprasad Ravishankar, Yong Long, and Jeffrey A. Fessler. "PWLS-ULTRA: An efficient clustering and learning-based approach for low-dose 3D CT image reconstruction," IEEE transactions on medical imaging 37, no. 6 (2018): 1498-1510.

[12] Zhang, Chen. "CuFusion2: Accurate and Denoised Volumetric 3D Object Reconstruction Using Depth Cameras." IEEE Access 7 (2019): 49882-49893.

[13] Guo, Yuanhao, Yunpeng Zhang, and Fons J. Verbeek. "A two-phase 3-d reconstruction approach for light microscopy axial-view imaging." IEEE Journal of Selected Topics in Signal Processing 11, no. 7 (2017): 1034-1046.

[14] Ding, Lu, Xosé Luís Deán-Ben, and Daniel Razansky. "Efficient 3-D model-based reconstruction scheme for arbitrary optoacoustic acquisition geometries." IEEE Transactions on Medical Imaging 36, no. 9 (2017): 1858-1867.

[15] Li, Bin, Yonghan Zhang, Bo Zhao, and Hongyao Shao. "3D-ReConstnet: A Single-View 3D-Object Point Cloud Reconstruction Network." IEEE Access 8 (2020): 83782-83790.

[16] A. Fitzgibbon, G. Cross, and A. Zisserman, "Automatic 3d model construction for turn-table sequences," in European Workshop on 3D Structure from Multiple Images of Large-Scale Environments. Springer, 1998, pp. 155– 170.

[17] J. Franco and E. Boyer, "Exact polyhedral visual hulls," in British Machine Vision Conference, vol. 1, 2003, pp. 329–338.

[18] S. Lazebnik, Y. Furukawa, and J. Ponce, "Projective visual hulls," International Journal of Computer Vision, vol. 74, no. 2, pp. 137–165, 2007.

[19] A. Djelouah, J. Franco, E. Boyer, F. Le Clerc, and P. Perez, "N-tuple color segmentation for multi-view ´ silhouette extraction," in European Conference on Computer Vision. Springer, 2012, pp. 818–831.

[20] K. Kutulakos and S. Seitz, "A theory of shape by space carving," International Journal of Computer Vision, vol. 38, no. 3, pp. 199–218, 2000.

[21] Y. Furukawa and J. Ponce, "Carved visual hulls for image-based modeling," International Journal of Computer Vision, vol. 81, no. 1, pp. 53–67, 2009.

[22] I. Ihrke, K. Kutulakos, H. Lensch, M. Magnor, and W. Heidrich, "Transparent and specular object reconstruction," in Computer Graphics Forum, vol. 29, no. 8. Wiley Online Library, 2010, pp. 2400–2426. [17] J. Ye, Y. Ji, F. Li, and J. Yu, "Angular domain reconstruction of dynamic 3d fluid surfaces," in IEEE Conference on Computer Vision and Pattern Recognition. IEEE, 2012, pp. 310–317.

[23] D. Liu, X. Chen, and Y. Yang, "Frequency-based 3d reconstruction of transparent and specular objects," in IEEE Conference on Computer Vision and Pattern Recognition, 2014, pp. 660–667.

[24] Y. Qian, M. Gong, and Y. Yang, "3d reconstruction of transparent objects with position-normal consistency," in IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 4369–4377.

[25] N. Savinov, C. Hne, L. Ladicky, and M. Pollefeys, “Semantic 3d reconstruction with continuous regularization and ray potentials using a visibility consistency constraint.” IEEE, 2016.

[26] S. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski, “A comparison and evaluation of multi-view stereo reconstruction algorithms,” in IEEE Conference on Computer Vision and Pattern Recognition, vol. 1. IEEE, 2006, pp. 519–528.

[27] X. Han, T. Leung, Y. Jia, R. Sukthankar, and A. Berg, “Matchnet: unifying feature and metric learning for patch-based matching,” in IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 3279–3286.

[28] J. Zbontar and Y. LeCun, “Stereo matching by training a convolutional neural network to compare image patches,” Journal of Machine Learning Research, vol. 17, pp. 1–32, 2016.

[29] K Thiruvenkadam, P Nagaraja. "Brain tumor segmentation of MRI brain images through FCM clustering and seeded region growing technique." International Journal of Applied Engineering Research, 2015, 10(76):427-43

[30] Fan, Baijiang, Yunbo Rao, Wei Liu, Qifei Wang, and Huaiyu Wen. "Region-based growing algorithm for 3D reconstruction from MRI images." In *2017 2nd International Conference on Image, Vision, and Computing (ICIVC)*, pp. 521-525. IEEE, 2017.

[31] Chen, Jun, Zhechao Wan, Jiacheng Zhang, Wenhua Li, Yanbing Chen, Yuebing Li, and Yue Duan. "Medical image segmentation and reconstruction of prostate tumor based on 3D AlexNet." Computer Methods and Programs in Biomedicine (2020): 105878.