Report

Reeb graph based segmentation of articulated components of 3D digital objects

Bachelor of Technology
in
Computer Science and Engineering

Submitted by:

Name	Enrollment No.
Priyesh Kumar	16114051
Rishikesh Chaudhary	16114054

Under the guidance of:
Dr. Ranita Biswas

Department of Computer Science and Engineering
Indian Institute of Technology Roorkee
Roorkee-Uttarakhand

Abstract

This project is an implementation of segmentation of the articulated components of digital objects by a fast and efficient algorithm presented in paper [1]. Given a 3D object in the form of a triangulated mesh, the approach involves voxelization of given object followed by construction of weighted Reeb graphs along each of yz-, zx-, and xy-planes. Then segmentation of Reeb graphs is carried out using the algorithm proposed in the paper where the concept of exponential averaging for dynamic thresholding ensures natural segmentation. The segmentation is carried out in a framework of 3D grid for efficient computation and storage. The segmented reeb graph corresponding to the three planes are subsequently related to each other and original given object so as to transform it into a segmented object.
Apart from segmentation of given object, voxelization and decompostion of given object into Reeb Graph along three coordinate axes has also been implemented in this project as a preparatory step for segmentation.

Contents

1 Introduction

Three-dimensional segmentation of digital objects is a well-studied concept and has been approached from various perspectives. Its widespread applications in shape analysis and understanding of peripheral topology of digital objects have contributed to its importance in various fields including 3D shape retrieval, texture matching, skeleton extraction, etc. Apart from geometric segmentation, several research works based on semantic-oriented segmentation have been reported. Graph-theoretic analysis of 3D objects is a well-explored area of shape representation that is characterized by a structural description of the object based on its geometric and topological properties. Topological variation of manifolds, particularly for articulated objects, may be suitably encoded by Reeb graphs as it has a one-dimensional graph structure and is invariant to both global and local transformation. In this project, we have implemented the algorithm proposed in [1] paper to obtain natural segmentation, which is fast, robust, rotationally invariant, pose-invariant, and free from skeletonization.

Implementing the proposed segmentation algorithm required some preprocessing. The first step was to voxelize the given 3D object. Our implementation of voxelization requires an input parameter representing the size of voxels which in turn determines the accuracy of segmentation. We rendered various voxelized object to make sure our implementation was correct. Next step was to decompose the voxelized object into Reeb Graphs along the three coordinate planes. Then segmentation of these Reeb Graphs was done using the proposed algorithm. Finally the three segmented Reeb graphs were related topologically to determine the final segmentation of the object into its articulated components. Natural segmentation with a relatively high degree of accuracy was achieved since the algorithm sets the threshold of segmentation dynamically using exponential averaging. This segmentation is achieved based on an input parameter which determines the relative weight-age to be given to the part of object under consideration with respect to the current component so as to determine if it can be a part of current component or requires a new component. The value of this parameter ranges from 0 to 1.

We tested the implemented algorithm with various 3D objects at different values of input parameter and different scales of voxelization and analyzed the result.

1.1 Background Research

We started with some background research in the topic. First of all we went through the basics of computational topology to get an idea of the problem

statement and relevant concepts like object topology, manifolds, orientability, voxelization, topological and quotient spaces, homeomorphism, Reeb Graph and segmentation itself. Then we went through [1] paper thoroughly to get a road map of how the project should be broken down.

Then our first objective was to get an idea about representation of 3D object in the form of triangular mesh and its processing to extract the necessary information from it. We also learned how to handle .obj files and how to add color codes to different section of given object using a .mtl file so as to represent the articulated components of the object with different colors later on.

Moving on to the next objective, we did some research about surface and solid voxelization and various 3D voxelization algorithms available. We have used [2] triangle-box intersection algorithm for voxelization due to its speed and simplicity. It uses separating axis theorem to determine intersection between a triangle and a cube in 3D space. This algorithm is very robust and is often used for collision detection.

After that we did a through study of reeb graph and how it is represented effectively. We also went through some proposed approaches to make reeb graph. Here we used the voxelized object for the construction of reeb graph along the three coordinate planes.

We decided to implement this project in java language due to its object oriented structure. So, we also brushed up our java concepts. We learned new methods for file handling and file IO and also went through java collections which helped us a lot during the coding phase.

2 Implementation

This project is implemented in an Object oriented fashion and contains various classes and functional modules.

2.1 Language

We have used **java** language

2.2 Classes:

There are 6 classes in total. Following are the names of those classes :-
1.) Vertex

2.) Triangle
3.) Ugc
4.) UgcIndex
5.) Node
6.) NodeIndex

2.2.1 Vertex

This is the class used to represent a point in 3D space. All the vertices are stored in an list called vertices which is a property of Isothetic Cover.

2.2.2 Triangle

This class represents a triangle in 3D space by storing the indices of its three vertices. All the triangles are also stored in an list called faces which is a property of Isothetic Cover.

2.2.3 Ugc

This class represents a unit grid cube. Isothetic Cover has a 3D array called grid which stores all the Ugcs around the object. Each Ugc stores indices of all the triangles which intersect it in an list called T. It also stores the Ugcindex of all the Ugcs which are intersected by the triangles that intersects it in an list U.

2.2.4 UgcIndex

This class stores the three indices required to represent an Ugc in the 3D array grid.

2.2.5 Node

This class represents the nodes of Reeb Graph. They are stored in a 2D list which is a property of Reeb Graph. It stores index of all the Ugcs represented by this node in an list called

S. It also stores the NodeIndex of all the neighbouring nodes in plane above and below in lists nextNeighbours and previousNeighbours respectively. It also has a property compId which will be determined by the segmentation algorithm and will be used in articulation.

2.2.6 NodeIndex

This class stores the two indices required to represent a node in the 2D list of nodes.

3 Functional modules

These modules actually does all the processing and contains all the algorithms used.

3.1 Functions

Following are the different functional modules used in this project

3.1.1 ObjectReader

This module reads the vertex and faces information from the object file. Function checkObj checks the extension of the file to make sure it is an object file. It has two functions readVertexData and readFaceData which scans through each line of the object file and returns an list of vertices and faces respectively. Those two methods use helper functions readVertex and readFace that scans through current line and stores the data in a Vertex and Trianlge object assuming that the object file contains the information of a triangular mesh i.e. all the faces are triangles.

Time complexity: O(v + f).
Space complexity: O(v + f).

3.1.2 OverlapTester

The function triboxOverlap in this module checks if a particular triangle overlaps with a given Ugc. It uses separating axis theorem to determine overlapping and takes centre of Ugc, half size of Ugc and three vertices of triangle. The algorithm can be found here [2].

Time complexity: O(1).
Space complexity: O(1).

3.1.3 IsotheticCover

It stores an list of vertices and faces computed using ObjectReader. It also stores a 3D array of Ugcs. Finally it contains a parameter g which represents the size of a Ugc. Constructor takes the parameter g and an object file as input. It uses initializeGrid function to find size of the 3D array (determined by the extent of the 3D object) along with its minimum coordinates and stores them. Then it calls make3dIsotheticCover function which sets the overlapping information in all the Ugcs. For each face it calls addTriangle function which determines a bounding box around that face using minimum and maximum x, y and z coordinates among all vertex coordinates. Then it uses the overlap tester to test the overlapping of this triangle with each Ugc in the bounding box and store the information accordingly. For each triangular face, we will look at all the Ugcs' intersected by the triangle and store the index of those Ugcs' in an Array List temp while adding the index of this triangle to the set T of all these Ugcs'. Now all the Ugcs' in the temp are added to each other's list U. This completes all the entries due to a particular

face. While adding that information we ensure that no duplicate entries are made in any set.

Ti- list of triangles (ti) intersecting the ith Ugc. (Topological space X).

Ui- list of Ugcs'(ui) intersected by triangles in Ti. (Topological space W).

We assume that all the Ugc which are intersected by triangles to be object occupied since we are doing surface voxelization and assuming the triangular mesh in the object file to be the surface of the object not the entire volume of object.

Time complexity: $O(f * (bbox * |T| + |temp|^2 + |temp| * |U|) + v)$
where

bbox represents the size of bounding box of the cube.
|**T**|− represents the number of triangle intersecting an Ugc.
|**temp**|−represents the number of Ugcs a triangle intersects.
|**U**|−represents the number of Ugcs intersected by triangles intersecting a particular Ugc. Thus $|U| = O(|T| * |temp|)$.
f- represents the number of faces in the object.
v- represents the number of vertices.

Thus,

Time complexity:$O(f * |T| * (|temp|^2 + bbox) + v)$.
Space complexity:$O(v + f + sizeofgrid * (|T| + |U|))$.

Where size of grid is the number of voxels the bounding box of entire 3D object contains. It depends on the extent of coordinates of 3D object and the size of a unit cube (g).
Thus, **size of cube** $= O(|rangeofxcoordinates| * |rangeofy - coordinates| * |rangeofz - coordinates|/g)$.

For Ugc's whose sizes are comparable to the size of the triangle (i.e. g is not much smaller than side of triangle), quantities $|T|, |temp|, |U|$, bbox may be treated as a constant w.r.t faces and vertices which is generally large.
Hence,
Time Complexity: O (f + v).
Space Complexity: O (f + v + size of grid).

3.1.4 RenderIsotheticCover

This module is just for checking if our Isothetic Cover is working fine. It scans through all the Ugcs in the grid and renders the object occupied ones without using any extra space.

Time Complexity - O (size of grid)
Space Complexity - O (1)

3.1.5 ReebGraph

This module constructs the Reeb Graph of the 3D object along x, y and z axis. It contains a 2D list called nodes which will contain the nodes of the Reeb Graph at each level. It also has properties called levels, numberOfNodes and axis which represent the number of levels, number of nodes and the axis along which the Reeb Graph is to be made respectively all of which are initialized by the a parameterized constructor that takes Isothetic cover and axis code as input. (X 0, Y 1, Z 2) First of all, separate components at each level are identified. This is done by iterating through all the Ugcs' at a level and each time an object occupied Ugc is encountered, a new partition/node is made followed by a breadth first traversal to find other neighbouring (connected) object occupied Ugc which are stored in list S a

part of this node, marking them as visited. Finally we add that component to the 2D list called nodes. Each of the components represents a node of the Reeb Graph.
The second step is to connect all the nodes at different levels which have some Ugc touching each other. It is assumed that only the Ugcs of adjacent levels can touch each other. Here we simply check all the Ugc's in a node to check if it touches any other Ugc of upper or lower level. An Ugc is assumed to touch another Ugc if they have a face in common. For every Ugc it touches, two nodes are added as neighbours of each other after checking if one already contains the other. This checking is done by checking the list of neighbours of both nodes to see if it already contains the other.

Time Complexity:
$O(sizeofgrid*sizeofplanarlevel*|nextNeighbours|*|previousNeighbours|)$.

Space Complexity:
$O(sizeofplanarlevel + numberofobjectoccupiedugc's)$.

3.1.6 Segment3D

This module implements the proposed algorithm for segmentation. Apart from that it contains another function which compiles the list of leaf nodes in each Reeb Graph. It stores the list a 2D list which contains object occupied Ugcs of each component along the three coordinate axes named mX, mY and mZ.

Following is the pseudo code for this module:

Algorithm Segment3D (A,G):
01. for each coordinate plane m

02. Gm ←Reeb graph representing quotient space C ‖ m-plane
03. Lm ← leaf nodes of Gm
04. Segment-Graph(Gm, Lm)

Procedure:

Segment-Graph(Gm, Lm):
01. count ← 1
02. **for** each leaf node ϵ Lm
03. **if** visited[v] = 0
04. Enqueue(Q,v)
05. v1 ← v
06. **while** Q is not empty
07. v ← Dequeue(Q)
08. **if**$|\omega[v] - \zeta * \omega| < \lambda * \omega[v1]$
09. compid[v] ← count
10. **for** each neighbor v' of v
11. **if** visited[v'] = 0
12. Enqueue(Q ,v')
13. visited[v] ← 1
14. v1 ← v
15. **else**
16. Lm ← Lm ∪ v
17. count ←count + 1

Time Complexity: O(number of object occupied Ugc's+number of nodes in Reeb Graphs along three axes + number of edges in Reeb Graph along three axes).
Space Complexity: O (number of object occupied Ugcs + space taken by three Reeb Graph).

3.1.7 ArticulatedComponents

This module populates the final components by combining the segments along the three axes generated in last module and stores them in a 2D arraylist called comp. It uses the three components array along the three axes mX, mYand mZ created in the last module. All object occupied Ugcs which have same components along the three axes belong to one component.

Time Complexity: $O((numberofobjectoccupiedUgc's)^3)$.
Space Complexity: $O(numberofobjectoccupiedUgc's)$.

3.1.8 ArticulatedObjects

It creates the new object file with different articulated components in different colours. First of all it writes all the vertices into the object file. The color codes are given using an mtl file. Then it writes all the faces in each Ugc of a particular component together so that it contains same colour. It generates 9 such file in a folder named articulated directory with the accuracy index 0.1, 0.2, 0.3, , 0.9.

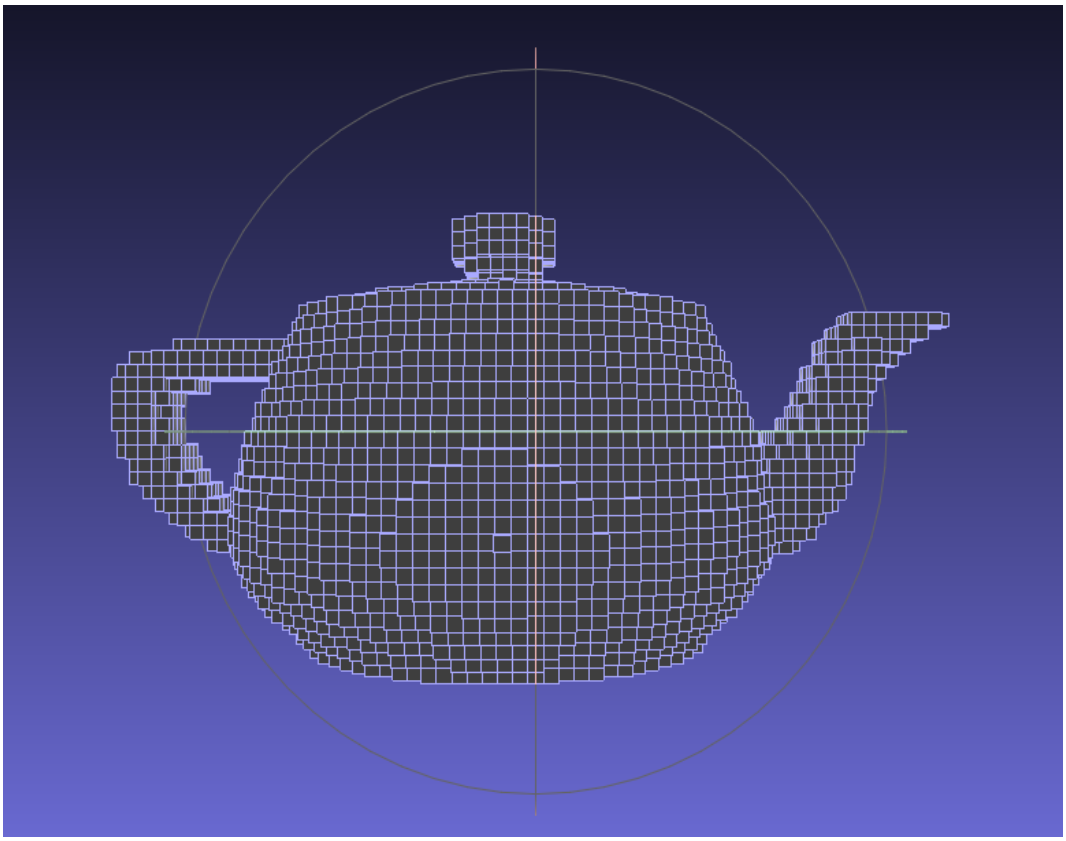
Time Complexity:$O(v + f + numberofobjectoccupiedUgc's)$
Space Complexity: $O(sizeofIsotheticCover+sizeofsegment3D+ sizeofArticulatedComponents + f)$.

3.2 Source code

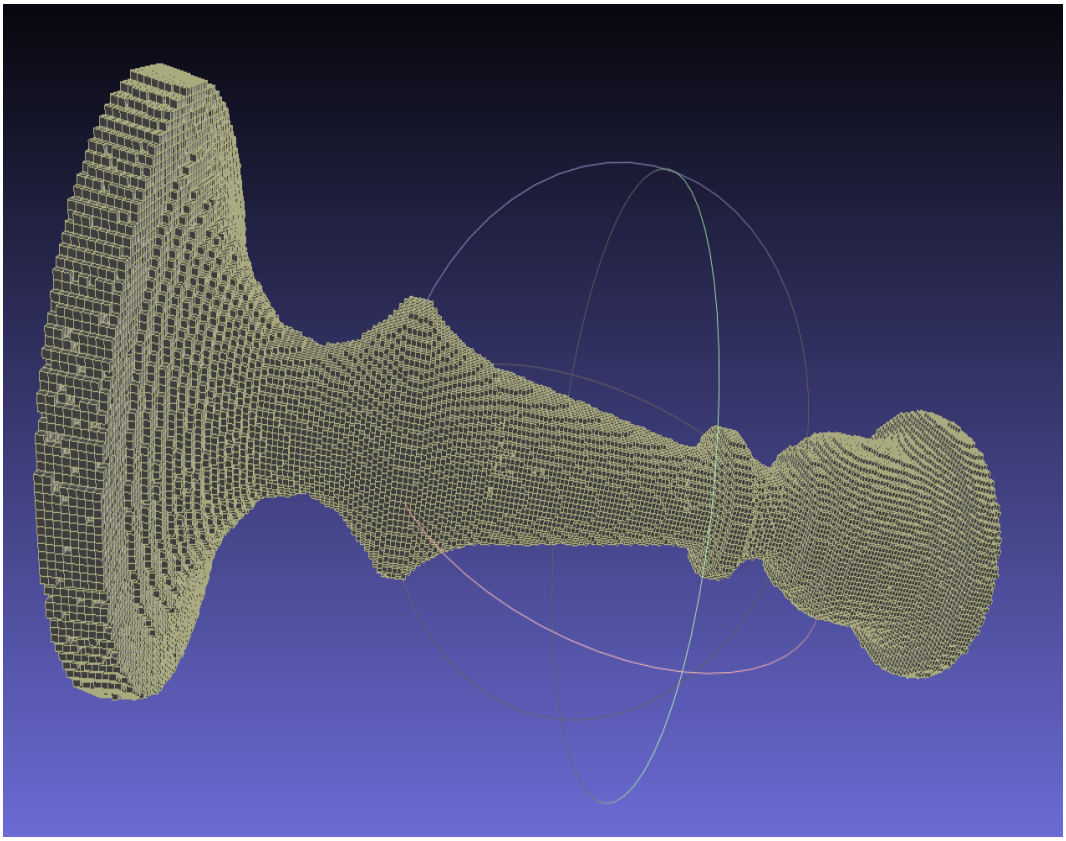
The link to our source code is
https://github.com/16priyesh/3D_Object_Segmentation

4 Results

4.1 Voxelized objects



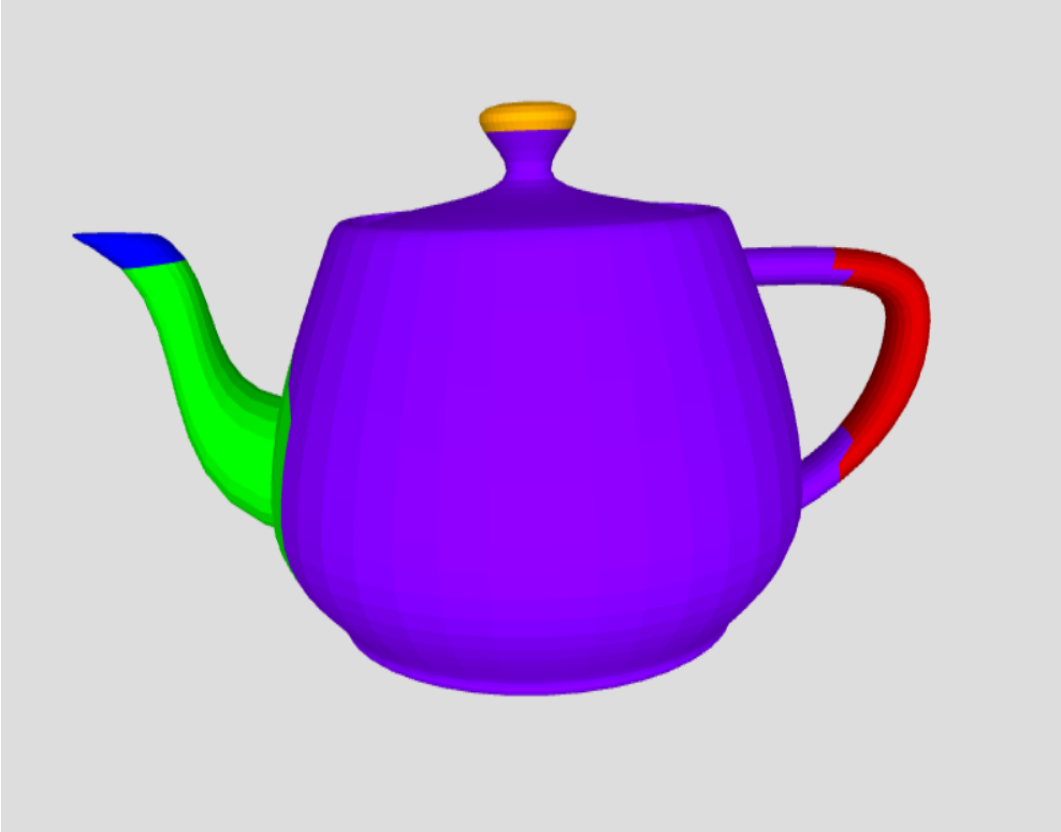
voxelized kettle



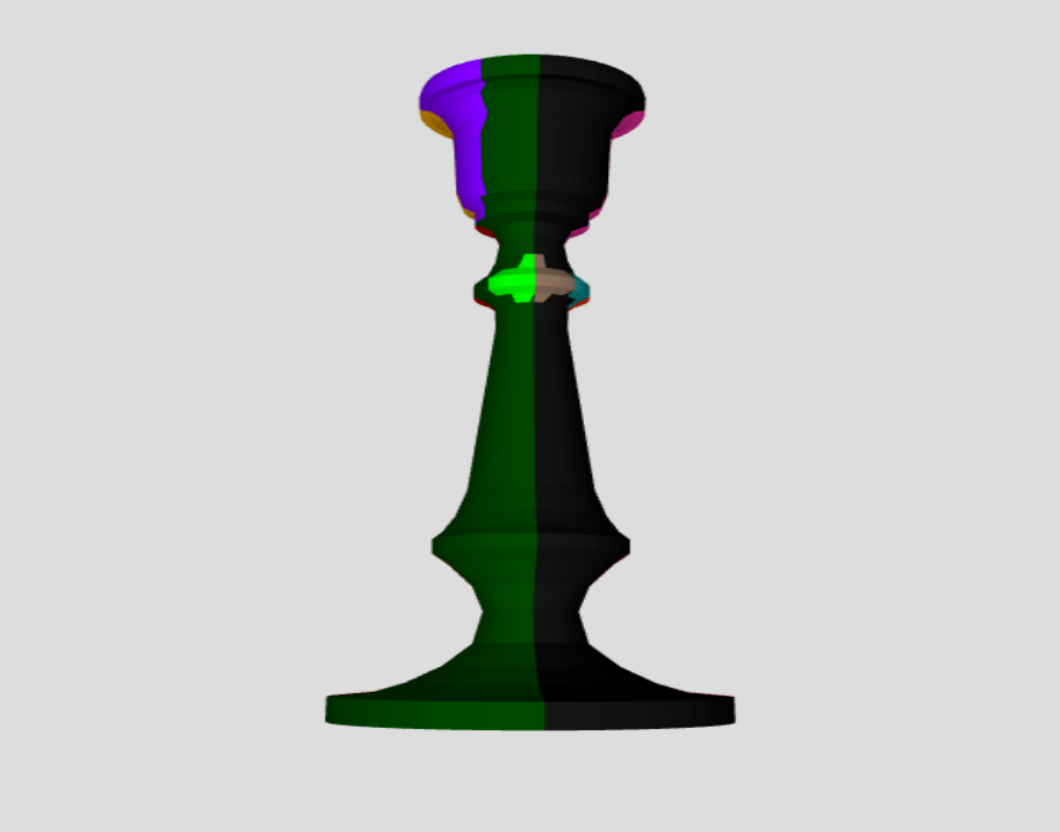
voxelized canstick

4.2 Articulated objects

Articulated Teddy bear



Articulated kettle



Articulated canstick

5 Conclusion

The result of this segmentation algorithm depends on two input values. First one being the scale of voxelization which represents the size of the Ugc's to be used. Lower the scale higher the time required to segment the object. Second input value is the parameter which determines the relative weight-age to be given to the part of object under consideration with respect to the current component so as to determine if it can be a part of current component or requires a new component. The value of this parameter ranges from 0 to 1.

After implementing and testing this segmentation algorithm, we have concluded that natural segmentation is achieved mostly around parameter value 0.4 to 0.6 and the optimal scale for segmentation is around 5% to 15% of object dimensions. Apart from that, higher the details in the input 3D object higher will be the number of components identified. In such cases, resulting object may look a bit over-articulated rather than naturally segmented. Lowering the value of the input parameter and increasing the scale of voxelization a little may lower the number of components identified in most cases. This algorithm works best for objects with lesser details.

This segmentation algorithm is rotationally and translationally invariant i.e. the results remain the same even if the input 3D object is rotated or translated. Resulting objects were moslty pose-invariant in terms of input object. We tested pose-invariance on a 3d horse object and resulting segments were mostly same for all poses of horse. Also, the result is free from skeletonization i.e. the object quality is not reduced at all.

We also checked the voxelization module used for pre-processing of input object extensively. It was fast and accurate.

Acknowledgement

The success and final outcome of this project required a lot of guidance and assistance from many people and I am extremely privileged to have got this all along the completion of my project. All that we have done is only due to such supervision and assistance and we would not forget to thank them.

We respect and thank Dr. Ranita Biswas, for providing us an opportunity to do the project work in IIT ROORKEE and giving us all support and guidance which helped us to complete the project. we are extremely thankful to her for providing such support and guidance.

1.)Priyesh kumar
2.)Rishikesh chaudhary

Dec 2017 to Dec 2017

Indian Institute of Technology Roorkee

Bibliography

1.) https://www.sciencedirect.com/science/article/pii/S0304397515010105
2.) https://dl.acm.org/citation.cfm?id=1198747
3.) Computational Topology An Introduction
4.) http://paulbourke.net/dataformats/obj/
5.) https://www.cs.cmu.edu/~mbz/personal/graphics/obj.html
6.) http://paulbourke.net/dataformats/mtl/
7.) voxel-rendering-techniques