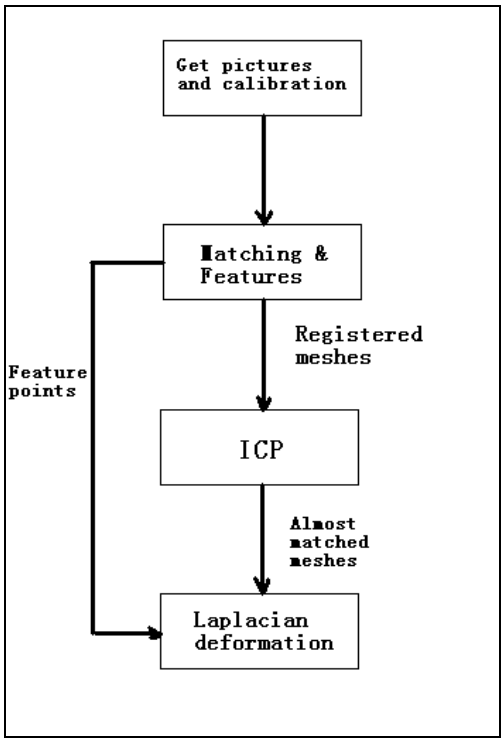
Scanning Real World Objects without Worries
——3D Reconstruction

Feng Li 308262
Kuan Tian 308263

1. Overview

This document is written for the 3D reconstruction part in the course "Scanning real world objects without worries project". The purpose of the project is using current scanning technology to generate depth images of statues, to register the depth images and generate standard computer graphics models from the aligned depth images.

To do the reconstruction work, ICP algorithm and Laplacian mesh deformation are used in this project.



The ICP and Laplacian deformation part is put after the Matching & Features part. ICP part receives registered meshes from the Matching part, which are supposed to be close enough to allow the ICP algorithm work. After the ICP algorithm, it is reasonable to assume that the meshes are already well matched except some detail part.

In case that in some places, the meshes still seem to be divided into two very close layers, which will make the matching cube algorithm fail. We need to do Laplacian deformation to merge the meshes. The Laplacian deformation has also effect on reducing the noise of meshes.

2. ICP Algorithm

2.1 Motivation

ICP is short for Iterative Closest Point. It is usually used to register two point clouds. In this project, we used a modified ICP algorithm to register meshes which are generated from the point clouds after Feature & Match part. The result of ICP is the input data for Laplacian mesh deformation which will improve the matching.

2.2 Modified ICP Algorithm

The crude ICP algorithm can not be used to register range images because it requires that each point on one surface have a corresponding point on the other surface. The scans in this project are overlapping. The range images got from the scans do not satisfy this requirement. So we use a modified ICP algorithm, which, to each vertex on one mesh, try to find the nearest position instead of the closest vertex on the other mesh. Its steps are:

1) Find the nearest position on mesh A to each vertex of mesh B.
2) Eliminate pairs in which either points is a mesh boundary.
3) Find the rigid transformation that minimizes a weighted least-squared distance between the pairs of points.
4) Iterate until convergence.

2.2.1 Find the Nearest Position

There are several ways to find the nearest position. In this project, we use the method below:

Step 1) To each vertex vi on mesh A, find the closest vertex vj on mesh B. (Use KD Tree to reduce the cost.)
Step 2) Form a ball whose center is vi , radius is the distance between vi and vj

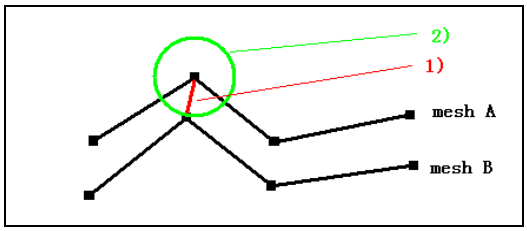


Figure 1: Step 1) and Step 2)

Step 3) Get the faces which are adjacent to vj and intersect with the ball.
Step 4) Project vi to the faces got in step 3.
Step 5) Check whether the projected point is inside the face (including the situation that the point is on the edge or on the vertex).

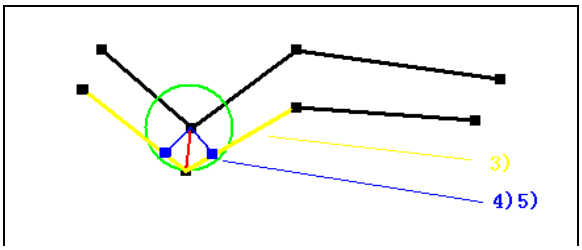


Figure 2: Step 3), Step 4) and Step 5)

Step 6) Among the points got in step 1e), find the closest point to vi.

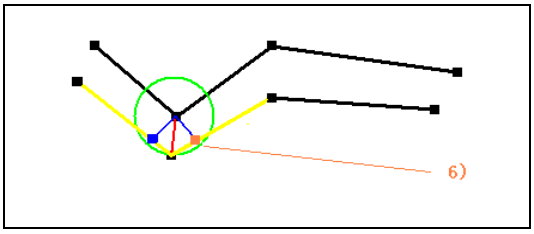


Figure 3: Step 6)

This is not a most optimal method to find the nearest position. Actually, sometimes the position found with this method is not exactly the "nearest" one. In the figure below, we can see such a situation. In this situation, the closest vertex found in step 1 is the blue one, and the final result is the green point, but the right result should be the red point. However, this mistake will not lead to great problems, since we should do the ICP algorithm iteratively. After doing several loops of ICP algorithm, the two meshes will be close enough to avoid such a situation, the problem will automatically disappear.

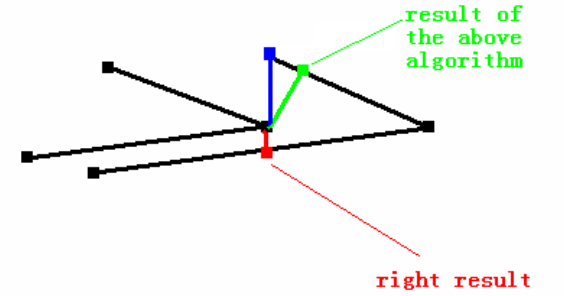


Figure 4: One situation that the method will provide a wrong result.

There are certainly some methods to totally solve such a problem. For example, we can use the normal projection. Instead of finding the closest vertex, we can project the vertex to the other mesh along its normal. Or we can calculate all the faces that intersect with the ball formed in step 2, instead of only finding the faces that are adjacent to the closest vertex, but this will causes some additional computation.

2.2.2 Eliminate Pairs

It is important to keep only the overlapping parts of the range images as the input for the least square fitting, if not, the result of the fitting will be wrong.

In order to fit this requirement, we should eliminate the point pairs which are got from section 2.1. If either point in one pair is on the boundary of a mesh, we should drop this. In the figure below, dotted arrows show matched point pair that one or both points are on the boundary. Such pairs should be dropped, since they tend to drag the two meshes erroneously. (In this case, mesh B will be dragged up and left.)

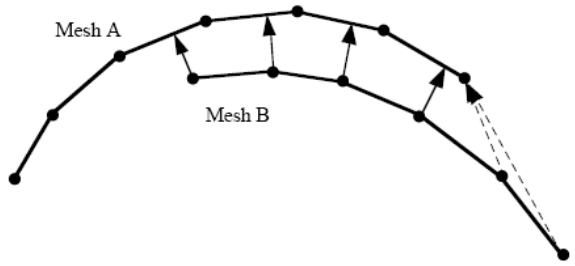


Figure 5

2.2.3 Rigid Transformation

After section 2.2, we get a list of point pairs, based on which we register the meshes. Since we only used rigid transformation here (The non-rigid deformation is introduced in the Laplacian Mesh Deformation.), it is reasonable to choose the least square fitting algorithm, finding the least-squares solution of R (rotate matrix) and T (translation vector), which minimize

$$\sum^{2} = \sum_{i=1}^{N} \left\| p_i' - (Rp_i + T) \right\|^2$$

One method to solve the LS Fitting efficiently is presented in [ICP02]. The method is based on SVD.

Step 1) $p \leftarrow \frac{1}{N}\sum_{i=1}^{N} p_i \quad p' \leftarrow \frac{1}{N}\sum_{i=1}^{N} p_i'$

For each i = 1, 2, …, N

$q_i \leftarrow p_i - p \quad q_i' \leftarrow p_i' - p'$

Step 2) $H \leftarrow \sum_{i=1}^{N} q_i q_i'^{t}$

Step 3) $[S \quad V \quad D] \leftarrow SVD(H)$

Step 4) $X \leftarrow VU^t$

Step 5) if det(X) = 1 then $R \leftarrow X \quad T \leftarrow p' - Rp$

else algorithm failed. (This case usually does not occur.)

2.2.4 Result

For testing the ICP algorithm that has been implemented, we translate one cat mesh to 6 different positions and add Gaussian noise to each point on the meshes.
Figure 6 shows the original translated and noised meshes and the matched meshes after ICP.

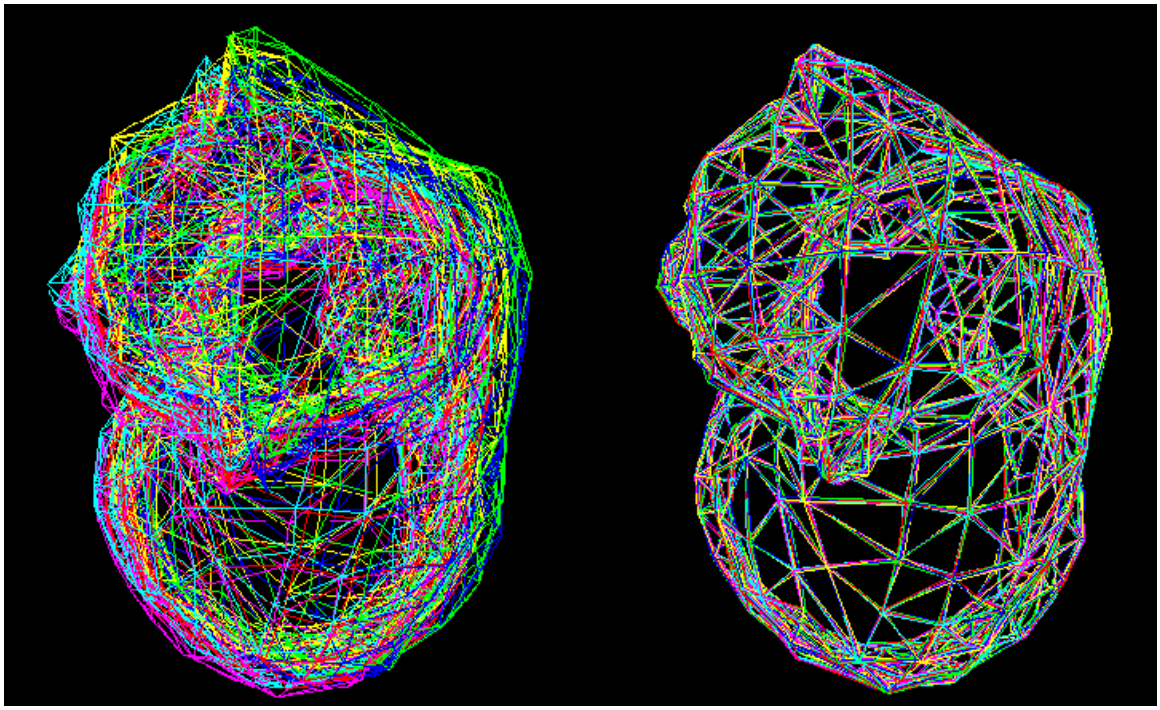


Figure 6: Original 6 translated and noised meshes (left) 6 matched meshes after ICP (right)

Since each point on the mesh has been added Gaussian noise, it is impossible to make all the corresponding points fit perfectly, but in this case the result is already good enough to let the matching cube algorithm work.

In addition, we can have a look at the result after Laplacian deformation in advance. Figure 7 shows the result after Laplacian deformation. Although we can still see some "noise" in this result, the corresponding points have been definitely better fitted.

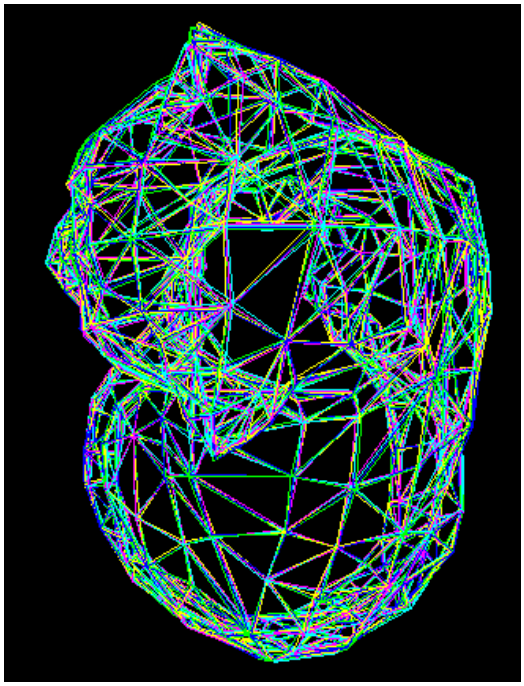


Figure 7: Result after Laplacian deformation

3. Laplacian Mesh Deformation

3.1 Motivation

After Feature Match and ICP algorithm we get a list of well matched meshes. But we still need to do one more step to merge these meshes, so we can put all the meshes together to get the object's profile.

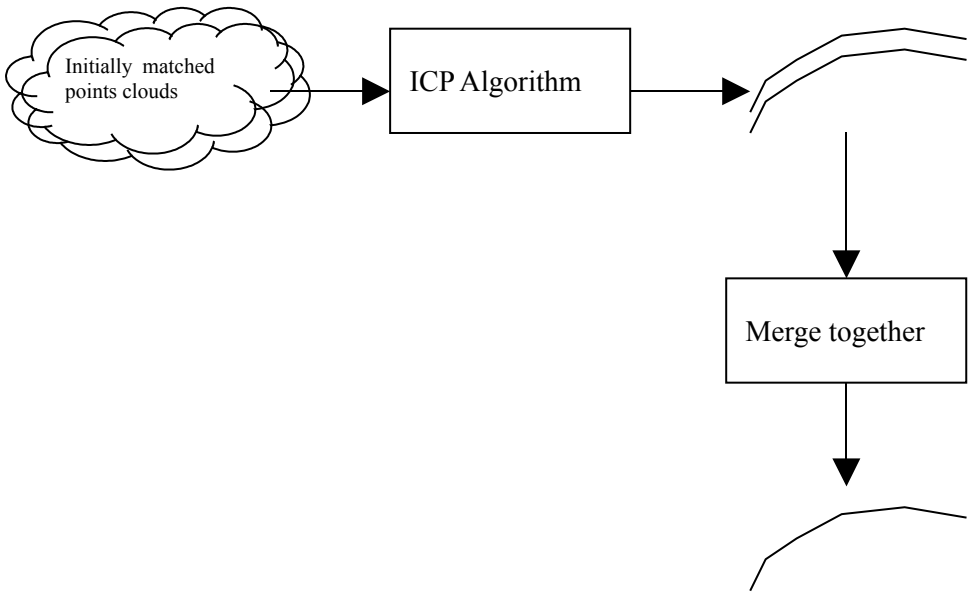


Figure 8

3.2 The way to merge

To merge these meshes together, there are two ways to do it. One way is to join these meshes to generate a big mesh, this needs some operation on the boundaries of the meshes. The other way is the way we use is the project, i.e., Laplacian Deformation, in these way we can get a point cloud of the project and later use some method like Matching Cubes to render the object.

3.3 Short introduction of the first way, called zippered polygon meshes.

Zippering two triangle meshes consists of three steps, each of which we will consider in detail below:

1) Remove overlapping portions of the meshes.
2) Clip one mesh against another.
3) Remove the small triangles introduced during clipping.

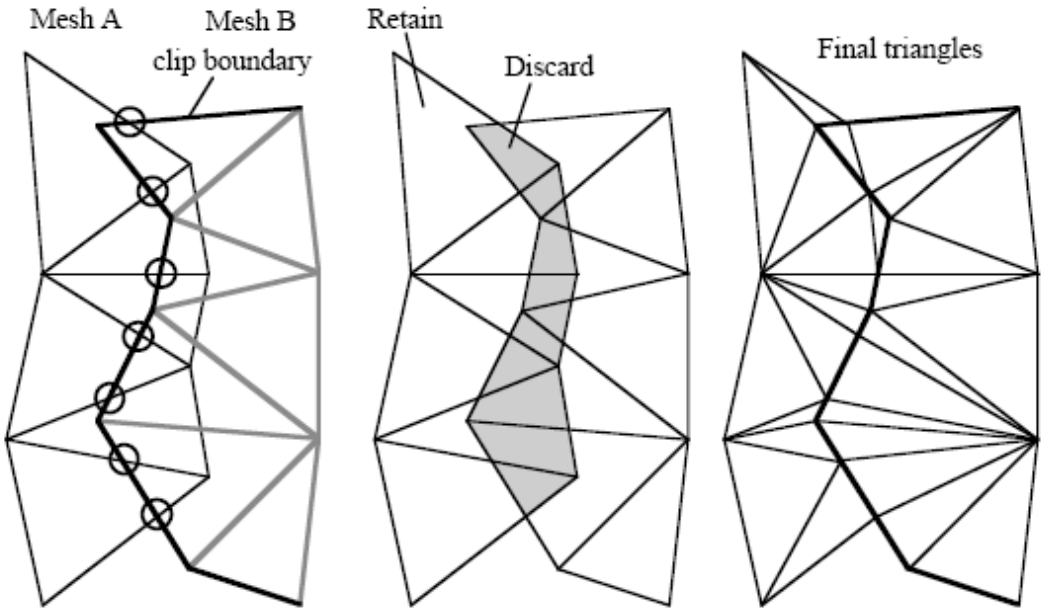


Figure 9:Mesh *A* is clipped against the boundary of mesh *B*. Circles (left) show intersection between edges of *A* and *B*'s boundary. Portions of triangles from *A* are discarded (middle) and then both meshes incorporate the points of intersection (right).

3.4 Laplacian Mesh Deformation

Laplacian mesh Deformation is a method to use differential coordinates and some features point to calculate position coordinates of the mesh. In another word, it is to change the Cartesian coordinates of the vertices but keep connectivity of the mesh

(differential coordinates doesn't change).

3.4.1 Definition of Differential Coordinate

Let M = (V,E,F) be a given triangular mesh with n vertices. V denotes the set of vertices, E denotes the set of edges and F is the set of faces. Each vertex i ∈M is conventionally represented using absolute Cartesian coordinates, denoted by
vi = (xi, yi, zi).

Define:

The differential or d-coordinates of vi is defined as to be the difference between the absolute coordinates of vi and the center of mass of its immediate neighbors in the mesh.(figure.1)

$$\delta_i = (\delta_i^{(x)}, \delta_i^{(y)}, \delta_i^{(z)}) = \mathbf{v}_i - \frac{1}{d_i} \sum_{j \in N(i)} \mathbf{v}_j$$

where N(i) = { j |(i, j)∈E} and di = |N(i)| is the number of immediate neighbors of i (the degree or valence of i).

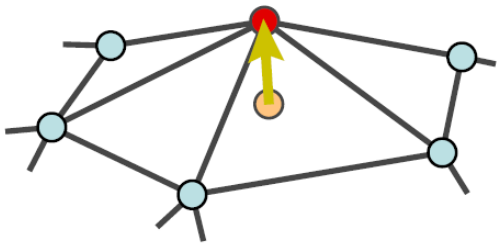


Figure10: definition of differential coordinates, denoted by the yellow vecor.

This operation to calculate differential coordinates from position coordinates can be represented as Matrix form. To do this first we define two matrix A and D.
a. Let A be the adjacency (connectivity) matrix of the mesh:

$$A_{ij} = \begin{cases} 1 & (i,j) \in E \\ 0 & \text{otherwise} \end{cases}$$

b. Let D be the diagonal matrix such that

$D_{ii} = d_i$, di is previously defined (the degree of the vertex i)

Then we can get the matrix L to transform the absolute coordinates to differential coordinates.

$$L = I - D^{-1}A$$

Then is used to calculate d-coordinates $L^*(x,y,z) = \delta^{(x,y,z)}$. (1)

3.4.2 Surface reconstruction from d-coordinates

After we calculate the differential coordinates we have saved the relationship of the vertices, i.e., the shape of the mesh. Now as we said want to reconstruction the mesh from the d-coordinates.

But we can't directly recalculate the positions from d-coordinates we had. Because the matrix L that we use to transform from position to differential coordinates is singular, and therefore the converse expression of equation 1 is undefined.

Here we solve it in another way. we place more than one constraint on spatial locations of M's vertices. Denote by C the set of indices of those vertices whose spatial location is known. We have therefore |C| additional constraints of the form: $v_j = c_j, j \in C$. Then we add the |C| additional constraints as additional rows to equation 1: $L * (x, y, z) = \delta^{(x,y,z)}$.

Then we can get our linear system looks like this:

$$\left(\frac{L}{wI_{m \times m} | 0} \right) * (x, y, z) = \begin{pmatrix} \delta^{(x,y,z)} \\ wc_{1:m} \end{pmatrix} \quad (2)$$

We denote the system matrix in (2) by L'. Note that we treat the positional constraints in the least-squares sense. This way, for each j ∈C we are also able to retain the "smoothness" term, i.e. the equation $Lv_j = \delta_j$. Note that the position constraint may not be fully satisfied by the least-squares solution. The weight w > 0 can be used to tweak the importance of the positional constraints (of course, every constraint may have its own weight, for example, we can set a bigger weight to the feature that is in sharper area of the mesh).

The additional constraints make our linear system over-determined, (more equations than unknowns) and in general no exact solution may exist. However, the system is full-rank and thus has a unique solution in the least-squares sense:

$$(x, y, z) = \arg\min_{(x,y,z)} \left(\left\| L * (x, y, z) - \delta^{(x,y,z)} \right\|^2 + \sum_{j \in C} w^2 \left| x_j - c_j \right|^2 \right)$$

The analytical least-squares solution is expressed in terms of the rectangular (n+m)×n system matrix L' from equation 2:

$$(x, y, z) = (L'^T L')^{-1} L'^T b$$

b is the right-hand side matrix of the linear system in equation 2.

3.4.3 Result

The following picture is the original mesh, which is a cat. The two white points is used as constraint to position of vertex 10 and 20.

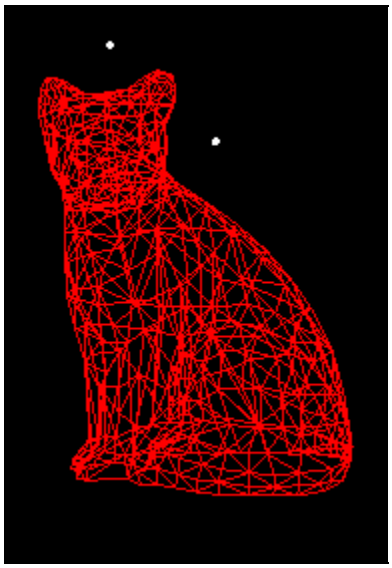


Figure 11

The following picture uses weight as 0.5 to deform the cat mesh.

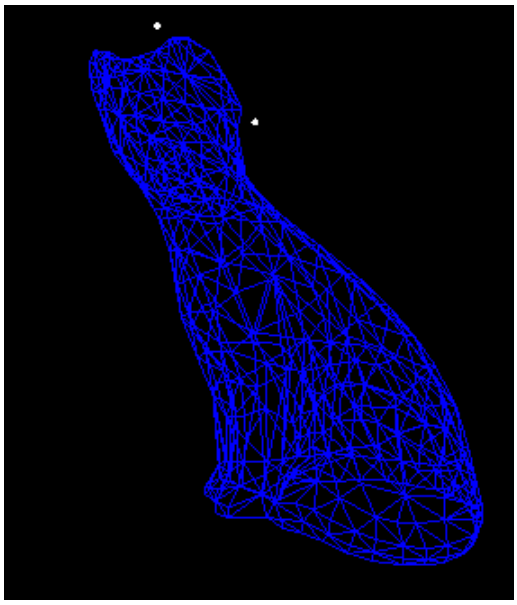


Figure 12

The following picture uses weight as 5 to deform the cat mesh. We can see the difference between these two deformed pictures. The bigger weight deforms more.

And the constraint is more satisfied.

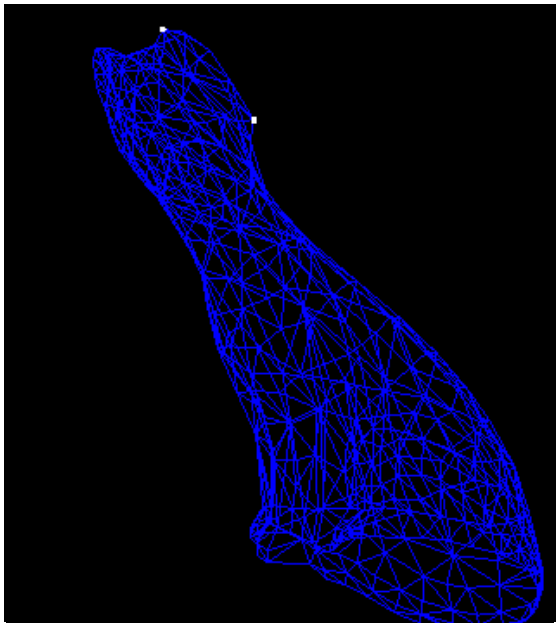


Figure 13

4. Related Work

4.1 KD Tree

A kd-tree (short for k-dimensional tree) is a space-partitioning data structure for organizing points in a k-dimensional space. Levels of the tree are split along successive dimensions at the points. (http://www.nist.gov/dads/HTML/kdtree.html).

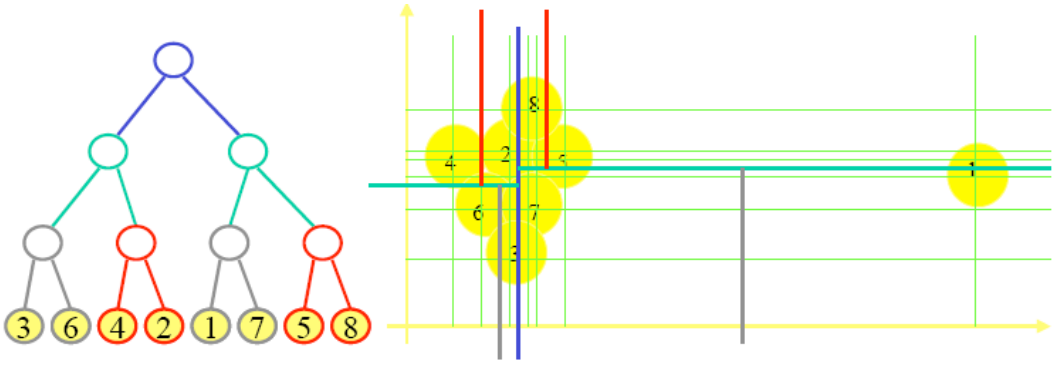


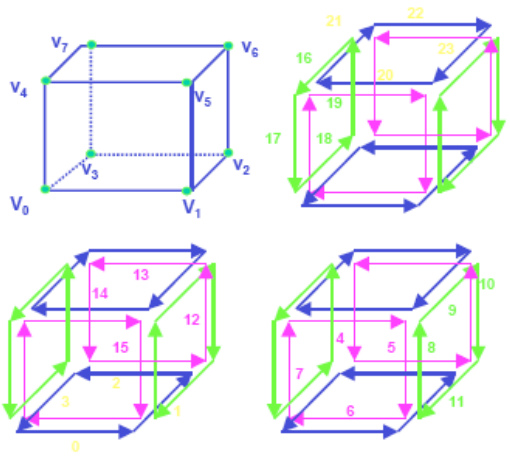
Figure 14: An example of kd-tree

In this project, kd tree is used to reduce the cost of finding the closest vertex on the mesh.

4.2 Half Edge Data Structure

A half-edge data structure is an edge-centered data structure for polyhedron. Each edge is decomposed into two halfedges with opposite orientations. One related face and one related vertex are stored in each half-edge. One next half edge and one opposite half-edge are also stored in each half-edge. For each face and each vertex, one related halfedge is stored.

In this project, half-edge data structure is used to store meshes.



Vertexlist

v	coord			he
0	0.0	0.0	0.0	0
1	1.0	0.0	0.0	1
2	1.0	1.0	0.0	2
3	0.0	1.0	0.0	3
4	0.0	0.0	1.0	4
5	1.0	0.0	1.0	9
6	1.0	1.0	1.0	13

Face

f	e
0	e0
1	e8
2	e5
3	e6
4	e11

Half-Edgelist

he	vstart	next	prev	opp	he	vstart	next	prev	opp
0	0	1	3	6	12	2	13	15	10
1	1	2	0	11	13	6	14	12	22
2	2	3	1	15	14	7	15	13	19
3	3	0	2	18	15	3	12	14	2
4	4	5	7	20	16	7	17	19	21
5	5	6	4	8	17	4	18	16	7
6	1	7	5	0	18	0	19	17	3
7	0	4	6	17	19	3	16	18	14
8	1	9	11	5	20	5	21	23	4
9	5	10	8	23	21	4	22	20	16
10	6	11	9	12	22	7	23	21	13

Figure 15: An example of half-edge data structure

4.3 Matching Cube Algorithm

The marching cubes algorithm is used in volume rendering to construct an polyhedron from a 3D field of values.

The basic principle of the marching cubes algorithm is to subdivide space into a series of small cubes. Then the algorithm tests the corner points of each cube to decide whether the points are inside or outside the polyhedron. After that, polygons can be formed in each cube according to figure 16, the total sum of all the polygon will be a surface that approximates the one that the data set describes.

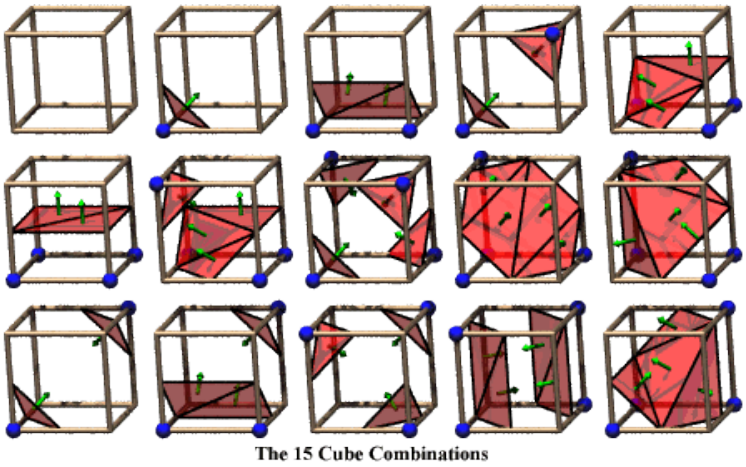


Figure 16: The 15 cube combinations in matching cube algorithm

(http://www.exaflop.org/docs/marchcubes/)

5. Reference

[ICP01] Zippered Polygon Meshes from Range Images, Greg Turk and Marc Levoy, Computer Science Department, Stanford University

[ICP02] Least-Squares Fitting of Two 3-D Point Sets, K.S.Arun, T.S. Huang, and S.D.Blostein

[LAP01] Laplacian Mesh Processing, Olga Sorkine, School of Computer Science, Tel Aviv University, Israel

[LAP02]

[REL01] Folien of the course “Computer Graphics 2”, TU Berlin

[REL02] http://www.exaflop.org/docs/marchcubes/