CV Project-3

多媒體工程所
宋秉一
9957513

Contents

CV Project-3 1
Distance Map 3
Octree-based visual hull 4
    Project cubes to images 4
    Octree cube nodes 5
Marching cubes 6
Texture mapping 8
    Nearest Camera method 8
    Ray-Tracing Mesh Intersection Mapping 8
    Multiple Camera Coloring 10
Program 11
References 11

Figures
    Figure 1.The distance map form binary image using chessboard kernel function. 3
    Figure 2.The Octree node structure 5
    Figure 3.Octree-cubes with initial size 2 and 3 levels Octree. 6
    Figure 4.Cube vertices and edge positions. 7
    Figure 5.Marching cubes result with initial size 4 and 7 levels Octree. 7
    Figure 6.Texture mapping using nearest camera with initial size 2 and 5 levels Octree. 8
    Figure 7.Ray-tracing texture mapping model with initial size 5 and 5 levels Octree 10
    Figure 8.Marching cubes with multiple camera texture, initial size 4 and 7 levels Octree. 11
Tables
    Table 1.Node type decision table 5

Table 2.Octree node status table 6
Table 3.A sample of vertices binary code 6
Table 4.A sample of marching cubes index. 7

Distance Map

To evaluate the distance to the object surface, we construct distance maps using chessboard kernel function. Define the bounding box of chessboard kernel by two points $\mathrm{P}_1(x_1, y_1)$ and $\mathrm{P}_2(x_2, y_2)$, $\mathrm{d}(\mathrm{P}_1, \mathrm{P}_2)$ is the distance value between P_1 and P_2.

$$\mathrm{d}(\mathrm{P}_1, \mathrm{P}_2) = \max\{|x_1 - x_2|, |y_1 - y_2|\}$$

The desired distance to the mask border is considered as the distance from the surface of object to its inside space. There shows an algorithm of the distance transform with chessboard kernel function.

```
1.  While
2.      For all pixel on the mask image
3.          Find the minimal value from chessboard kernel each pixel
4.          Replace the pixel value with (min value + initial mask value)
5.      End
6.      If the map is the same with previous map, break out while loop.
7.  End while
```

Our project use the distance map to determine whether the projected 2D points are inside the object or not. The inside pixels in distance map shown as positive values and the outside pixel as negative values. By reversing the image mask, we could get the inner distance map and outer distance map.

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	1	1	1	0	0
0	0	1	1	1	0	0
0	0	1	1	1	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

Image mask

-2	-2	-2	-2	-2	-2	-2
-2	-1	-1	-1	-1	-1	-2
-2	-1	1	1	1	-1	-2
-2	-1	1	2	1	-1	-2
-2	-1	1	1	1	-1	-2
-2	-1	-1	-1	-1	-1	-2
-2	-2	-2	-2	-2	-2	-2

Distance map

Figure 1.The distance map form binary image using chessboard kernel function.

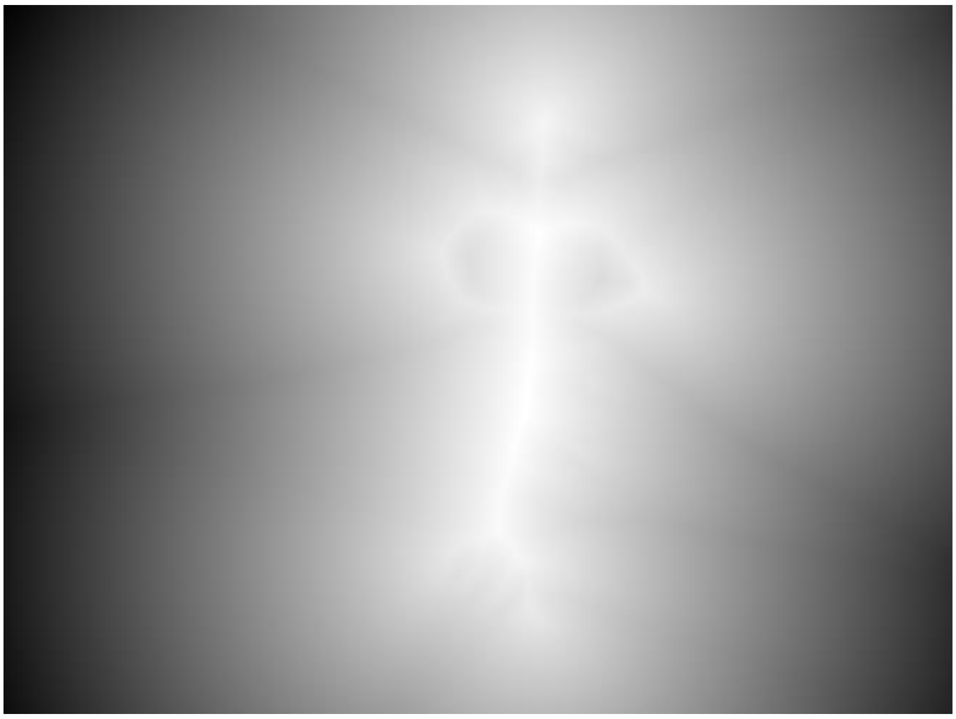


Figure 2.Distance map of image 1.

Octree-based visual hull

Project cubes to images

The main idea of visual hull is a 3D intersection volume method which project 3D cubes to each distance map of different camera views, and check whether the cubes are exist or not. The cube in 3D space is defined by 9 vertices (a cube center and 8 vertices).

With the given projection matrix of each camera view, the 3D points of each cube could be projected on to the 2D image plane and check the existence.

$$\begin{bmatrix} sx \\ sy \\ s \end{bmatrix} = P_{3\times 4} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

There are 3 types of cubes. The black node means the inner cube and the white node means the outer cube. Both of the two node types are not shown as the reconstruction result. That means the program will skip saving these nodes when computing. The desired node type is the gray node, which is considered as the object surface.

We apply some simple rule to determine the node type. Define the projection radius R as the maximum distance from projected center to vertices, the projected cube center c, and the cube vertices as v_i.

$$\mathrm{R} = \max\{\mathrm{d}(\mathrm{c}, \mathrm{v}_i), i = 1,2 \ldots 8\}$$

$\mathrm{DistMap(c)} > 0$	$\mathrm{R} > \mathrm{DistMap(c)}$	Gray node
	$\mathrm{R} < \mathrm{DistMap(c)}$	Black node
$\mathrm{DistMap(c)} < 0$	$\mathrm{R} > \mathrm{DistMap(c)}$	Gray node
	$\mathrm{R} < \mathrm{DistMap(c)}$	White node

Table 1.Node type decision table

Octree cube nodes

The original visual hull method is very time-consuming which the computation complexity of cube projection is a trade-off between model details and the time complexity. The more detail model must needs more small cubes to present, and the white or black nodes are also been checked with the same time of gray nodes.

The Octree is a space carving data structure in 3D space which we could apply for acceleration. Similar with Quadtree, the Octree partition the subspace into half size in each dimension. The subspace is composed by the 8 same size cubes in the given space.

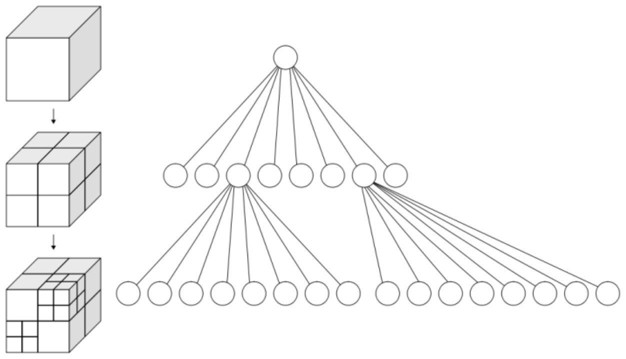


Figure 3.The Octree node structure

If there is any distance map provides a white node case, the cube is absolutely a white node and no need to compute the subspace of this cube. The black node only shows when all projection results are black node case and skip computing the subspace of cube. The other cases are treated as the gray node cases which are applied the same work in the subspace of the cubes until reach the maximum Octree level.

Any map	White node
All map	Black node
Else cases	Gray node

Table 2.Octree node status table

During the time of Octree construction, we also need to record the vertices position in each gray node by using the binary code each vertex. We define the vertex position with 1 when inside cases and 0 for outside cases. The composed binary code is by applying logic AND operation of corresponding vertex in different camera view.

	V8	V7	V6	V5	V4	V3	V2	V1
image1	0	1	0	0	1	0	1	1
image2	0	1	0	1	0	0	1	1
image3	0	1	0	1	1	0	0	1
AND	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**1**

Table 3.A sample of vertices binary code

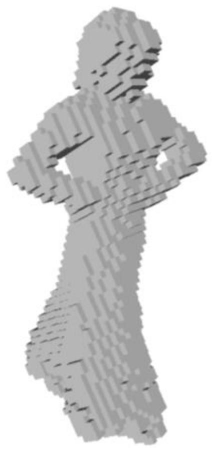


Figure 4.Octree-cubes with initial size 2 and 3 levels Octree.

Marching cubes

The Octree perform an approximate model of target object composed by cubes which are independent of each other and have no topological relations. But for the 3D model rendering, we even can’t see the inside face of cubes. In order to get a more accurate model and a smaller model file, we need to merge the topological faces with triangle mesh.

The marching cube divides a single cube into at most 5 triangles by the corresponded lookup table. The vertices and edges positions are shown in Figure 5 which we use to find the index of lookup table (see Table 4).

	V8	V7	V6	V5	V4	V3	V2	V1	Index
image1	0	1	0	0	1	0	1	1	N/A
image2	0	1	0	1	0	0	1	1	N/A
image3	0	1	0	1	1	0	0	1	N/A
AND	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**65**

Table 4.A sample of marching cubes index.

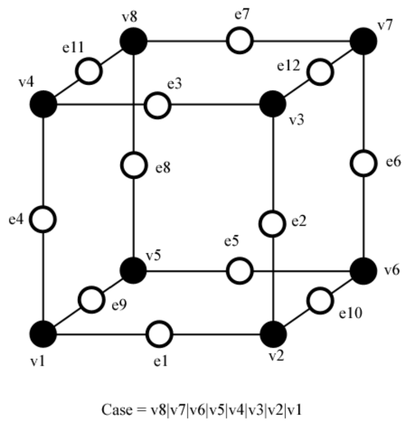


Figure 5.Cube vertices and edge positions.

Once we have the marching cubes model, we almost get a more accurate estimation of target object. But the triangle vertices are duplicated because the independency of each cube. In order to refine a correct topological relation of triangle mesh, we do unique operation on all vertices.

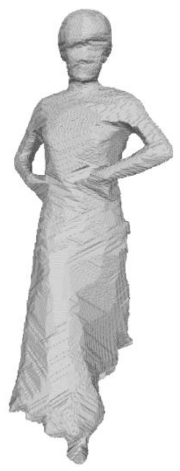


Figure 6.Marching cubes result with initial size 4 and 7 levels Octree.

Texture mapping

We have tried three mapping technics to get the vertex color. The model texture can't be retrieve from only one image. Instead, the main idea is project the 3D triangle vertices in to different camera images and maps the color to vertices.

Nearest Camera method

The simplest and fastest way to do texture mapping is to find the nearest camera. In the common case, the nearest camera should have captured the vertex color in image.

$$\text{Vertex Color} = \min_{color}\{d(v, c_i), i = 1,2 \dots N\}$$

Because of the occlusion of model, the nearest camera method will fail at the occlusion position.

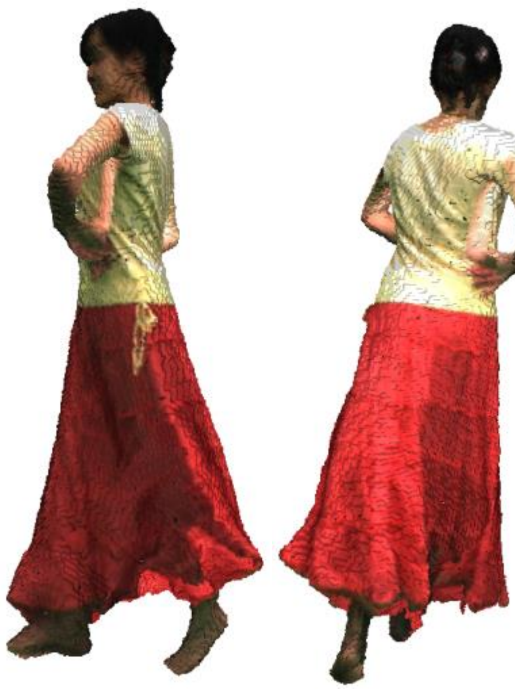
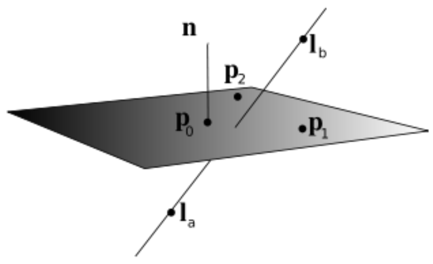


Figure 7.Texture mapping using nearest camera with initial size 2 and 5 levels Octree.

Ray-Tracing Mesh Intersection Mapping

The way to solve the occlusion problem we tried is using the ray-tracing method. If the ray from vertex to nearest camera is intersected by any mesh, we change camera for another ray check.

Define the target vertex $\vec{l}_a$, camera center $\vec{l}_b$ and triangle mesh vertices P_0, P_1, P_2.



The intersection point on the plane of triangle mesh

$$\vec{l}_a + t(\vec{l}_b - \vec{l}_a), t \in [0,1]$$

The intersection point can be span by $(P_1 - P_0)$ and $(P_2 - P_0)$ vectors.

$$\vec{l}_a + t\overrightarrow{l_b l_a} = P_0 + u(P_1 - P_0) + v(P_2 - P_0)$$

Solve the unknown scalar t, u, v

$$\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \begin{bmatrix} x_a - x_b & x_1 - x_0 & x_0 - x_0 \\ y_a - y_b & y_1 - y_0 & y_2 - y_0 \\ z_a - z_b & z_1 - z_0 & z_2 - z_0 \end{bmatrix}^{-1} \begin{bmatrix} x_a - x_0 \\ y_a - y_0 \\ z_a - z_0 \end{bmatrix}$$

If the intersection point is in triangle mesh

$$u, v \in [0,1], (u + v) \le 1$$

The ray-tracing method is depending on the accuracy of model. But the model we constructed is just an approximation result, so the correct ray may hit other mesh. This case result a wrong projection to other camera image or even has no result texture.

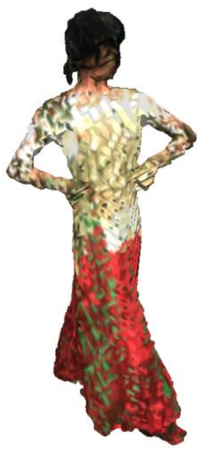


Figure 8.Ray-tracing texture mapping model with initial size 5 and 5 levels Octree.

Multiple Camera Coloring

By pervious methods, we have a new idea to find the vertex texture by considering multiple camera images at once.

We do the nearest camera method and check multiple camera images using nearest camera as center camera. The given camera images is a ring view of target object, so we could attach the neighbor camera by changing the camera index.

Get the nearest camera index I, the multiple camera indexes I_s.

$$I = \min_i \{d(v, c_i), i = 1,2 \ldots N\}$$

$$I_s = I + s, where\ s = [-3, +3]$$

Fix the index bound by the ring views.

$$\begin{cases} I_s = N + I_s & if\ I_s < 1 \\ I_s = I_s - N & if\ I_s > N \end{cases}$$

There may some projections that have wrong position which we can determine by the given silhouette. The wrong projections are outside of the silhouette mask which the pixel values are 0. Remove the wrong projections color and compute the correct color mean.

The desired vertex texture is considered be the nearest one to the color mean.

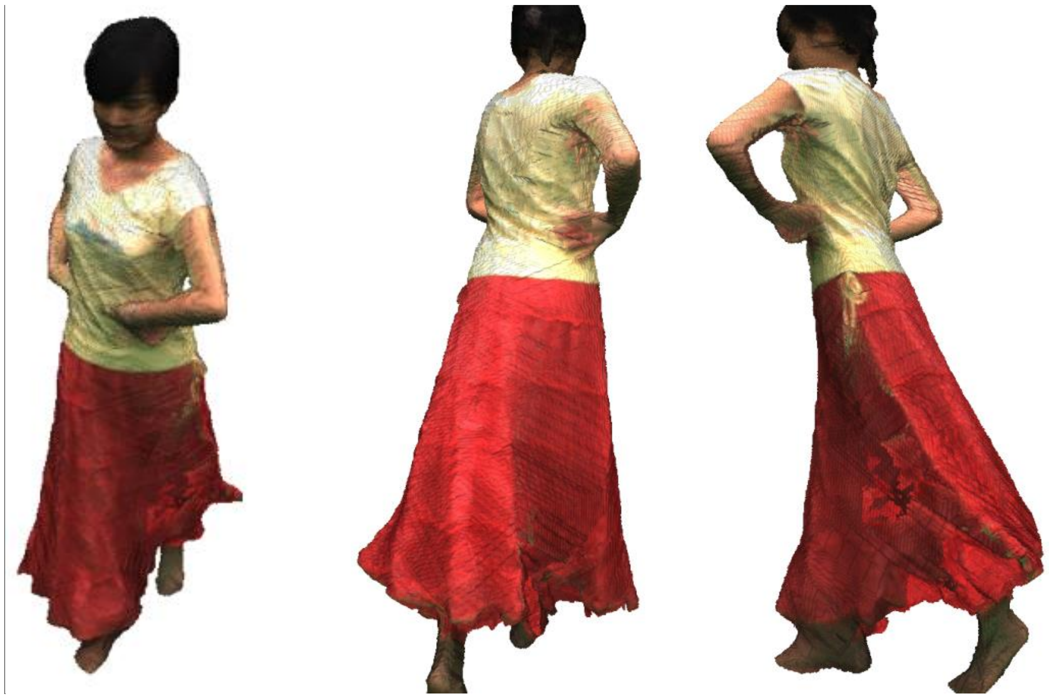


Figure 9.Marching cubes with multiple camera texture, initial size 4 and 7 levels Octree.

Program

Our program is divided by 3 parts. And can start with the main.m.

1. main_loader.m: load images and projection matrices and compute the distance maps
2. main_projection.m: get the Octree structure.
3. main_mesh.m: marching cubes.

The final result is saved as mesh.ply

References

[1]. H. L. Chou and Z. Chen, "Fast octree construction endowed with an error bound controlled subdivision scheme", Journal of Information Science and Engineering, vol. 22, no. 3, pp. 641-657 (2006).

[2]. Wen-Chao Chen, Hong-Long Chou and Zen Chen, A quality controllable multi-view object reconstruction method for 3D imaging systems, Journal of Visual Communication and Image Representation 21 (5–6) (2010), pp. 427–441.

[3]. http://en.wikipedia.org/wiki/Line-plane_intersection

[4]. http://users.polytech.unice.fr/~lingrand/MarchingCubes/accueil.html