Visual Hull Algorithm

Visual hull algorithm (or space carving) is a method for reconstructing a geometry of an object from a series of silhouette images taken from known viewpoints. Silhouettes are basically black and white images, masks, where the white pixels constitute the projection of the object, black pixels belong to the background. The algorithm works by finding the largest volume whose projections are consistent with the silhouettes.

The algorithm consists of the following steps:

1. An initial volume is established that is expected to contain the unknown model entirely.
2. All parts of the volume that are determined not to belong to the model are removed. The test is performed by projecting the part of the volume onto the silhouettes and testing whether the projection is on background pixels.
3. The remaining volume is converted to a mesh.

The algorithm described below can be considered *voxel-based*. Space is regarded as a regular grid of small cubes called voxels (analogous to pixels in 2D). Within the initial volume, the voxels that are projected onto background pixels are the ones removed. The remaining collection of voxels constitute the output of step 2.

For efficiency, we introduce the concept of *boxes*. A box is a cube whose edges are parallel to the coordinate axes.

Building the Box Tree

The *initial* volume is given as a box, and is an input of the algorithm. (Thus step 1 is not part of the implementation.) Step 2 is performed by building a tree from this initial box. All nodes of the tree are boxes themselves. The tree is built by performing a traversal and processing each node as follows:

The box is projected onto the silhouette images. If there is at least one image where the projection constitutes only of background pixels, then the box is marked as *dropped*, and is considered not to be part of the object. Otherwise (i.e. when the projection intersects every silhouette) if every projection constitutes entirely of foreground pixels, then the box is considered to be *final* (part of the object). Otherwise we cannot make a definite assessment of the box, we split it into eight subboxes. These subboxes are child nodes of the box, and will be processed in the same way.

The maximum number of subdivisions is a parameter of the algorithm and determines the minimal box size. A minimal box is not split any further, it is marked as final instead.

After processing all the nodes, this step of the algorithm outputs a tree. The tree's root is the initial box, its non-leaf nodes are boxes that have been split, and its leaves are boxes that are either final or dropped. The model is the union of all final boxes.

There are some cases that need special attention:

- A box (or a part of it) might be projected outside the bounds of a silhouette image. The current implementation consideres these "pixels" to belong to the background, virtually assuming that the object is entirely within the view of every camera.
- If a box contains one of the cameras, it has a chance of never being discarded, as the projection of such boxes will occupy the entire image. If it cannot be dropped based on one of the other silhouette images, it will be split, one of the subboxes will once again contain the camera and should be split, and so on, the box is of minimal size, in which case it is kept. This case might result in small individual boxes at the positions of the cameras. To avoid this, minimal boxes that contain a camera are dropped. (The assumption is that the minimal size is small enough so that it is smaller than the distance of the cameras from the object.)

- When a box is projected onto a silhouette image, it is possible that the projection occupies no pixels, not because the box is not in view of the corresponding camera, but because the box is so small that the projection is inbetween neighbouring pixels. The implementation detects this case and handles it by projecting the center point of the box onto the image and considering the closest pixel.

Mesh Generation

Given the box tree, an algorithm quite similar to Marching cubes[1] is performed to obtain a triangulated mesh. Marching cubes works by iterating over every voxel, testing each vertex of a voxel whether it is inside the object, and generating some trangles accordingly.

The same virtual grid is used for mesh generation as in the box tree creation. The marching cubes we consider are all the possible minimal boxes within the grid.

For voxels that are not at the boundary of the model (i.e. all the corners are either inside or outside), no triangles are generated. This means that only those boxes that touch a final box but are not part of a final box themselves will be relevant. Thus the marching cubes we need to consider can be efficiently iterated by traversing the box tree to find dropped boxes, and enumerating all the minimal-sized boxes located at the edge of the dropped box. (If the dropped box is minimal, it will be a marching cube itself.)

Once we have a marching cube, we need to determine which of its 8 vertices lie on the surface of a final box. Relevant final boxes are found, once again, by traversing the box tree. (In fact, split boxes that are disjoint from the marching cube need not be searched.) This results in an efficient implementation of the marching cube algorithm.

One thing to note is that if one of the final boxes lies at the edge of the initial box, then no triangles can be generated for the outside face as there are no marching cubes adjactent to those boundary faces. (All the marching cubes lie within the initial volume.) In other words, mesh generation works correctly only if the initial box is larger than the model itself, and the distance of their surfaces is at least the minimal size.

1. http://en.wikipedia.org/wiki/Marching_cubes