6 An Accelerated Rendering Algorithm for Stereoscopic Display

Sheng Fu, Hujun Bao, Qunsheng Peng

State Key Laboratory of CAD & CG, Zhejiang University, Hangzhou, P.R. China

With the development of the scientific visualization and the virtual environment techniques, stereo viewing systems have now been used extensively. In this paper, we present an accelerated rendering algorithm for stereoscopic display. As the difference between the left view and the right view is slight, we generate the right view by a transformation of the left view conforming to the stereo disparity. The problem of visibility change of a few polygons during the transformation is discussed and an efficient algorithm is developed for filling the holes that may arise in the right view after the transformation. This method makes fully use of the coherence between the left view and the right view. Experiments prove its efficiency.

KEYWORDS: stereoscopic display, viewpoint, z-buffer, transformation, parallel, rendering, BSP tree

INTRODUCTION

While most users rarely notice it, nearly all the 3D applications are actually displayed in two dimensions. Although a solid model, for example, can be rotated and shaded, it is frequently displayed as a 2D projection on a flat screen. The depth information of the model is lost.

Stereo viewing makes it easy to interpret and manipulate complex models, from wire-frame to fully rendered solid models. One study found that the use of a stereo display eliminated 75 percent of the errors made by engineers designing sheet metal parts: the systems helped them better visualizing the process [ii]. With the development of the scientific visualization and the virtual environment techniques, stereo viewing systems are now being used more and more extensively.

True depth perception occurs when each eye sees the world from a slightly different perspective. Our minds fuse these two into a single stereoscopic image. A stereo ready system replicates this phenomenon by delivering two slightly different images, one to each eye. Typically the left image and the right image are rendered separately [ii] and the computer needs to process the scene twice. It can be easily found that the difference between the two images is slight and most of the scene are visible to both the left eye and the right eye. If we account for only the view-independent shading, the intensity of a visible object indifferent views should be the same. The difference between the left view and the right view for the same visible point is that it may appear at different

locations. This difference is called the stereo disparity. Obviously, if we can make fully use of the view coherence, the rendering time of stereo display can be greatly reduced.

Adelson took this coherence to accelerate stereoscopic ray-tracing [vii]. The pixels 8A in the left view are re-projected to their inferred position in the right view and cleaning up the image by recalculating only those pixels whose value are unknown or in question. The re-projection method is also applied to generate stereoscopic animation [viii].

Eric Chen, et al. generate the intermediate image by an interpolation of the two adjacent images which depict the same object from slightly different viewpoints [iii, iv, v]. These methods use the camera's position and orientation and the range data of the images to determine a pixel-by-pixel correspondence between images, the intermediate image is generated by interpolating the correspondent pixels.

In virtual environment, the scene must be processed in real time. The major drawback of ray tracing is the large computational cost of calculating ray object intersections. So it can hardly be used in virtual environment. Instead, the hardware z-buffer is always adopted. In this paper, we present an accelerated method based on the view coherence for generating the stereo display. This method renders the left image first using hardware z-buffer. The intensity and the coordinates of the visible points in the left image can be read from the frame buffer. As the transformation between the left view and the right view are available, the pixels in the left image can be transformed into the right image. Therefore the objects visible to both the left eye and the right eye need no longer be rendered twice. A special problem of this approach is how to fill the holes that may arise in the right view after the transformation due to either image expansion or object visibility change. An efficient hole filling algorithm is developed in this paper by localizing the visibility test to a few objects.

One advantage of our method is that it can be run in a parallel mode. As will be shown in later section, the transformation of each line of the left image can be conducted independently. So the left image can be divided into some blocks and each block can be processed parallely. This accelerates the rendering speed.

Section 2 introduces the perspective fundamentals. In section 3 we discuss the difference between the left view and the right view with emphasis on the fast generation of the right image based on view coherence. The algorithm for filling the holes in the right image is presented in section 4. Section 5 demonstrates the whole algorithm and presents some results. Conclusions and future research directions are addressed in the last section.

1.0 PERSPECTIVE FUNDAMENTALS

For viewing purpose, the objects in the scene are transformed into the eye coordinate system whose origin is located at the viewpoint and z axis is coincident with the sight direction. A perspective transformation is then applied to the objects with the viewpoint being moved infinitely far away.

The correspondent formula is (see Fig. 1):

$X_s = (x_{ed}) / z_{ee}$, $Y_s = (y_{ed}) / z_e$, $z_s = z_e$

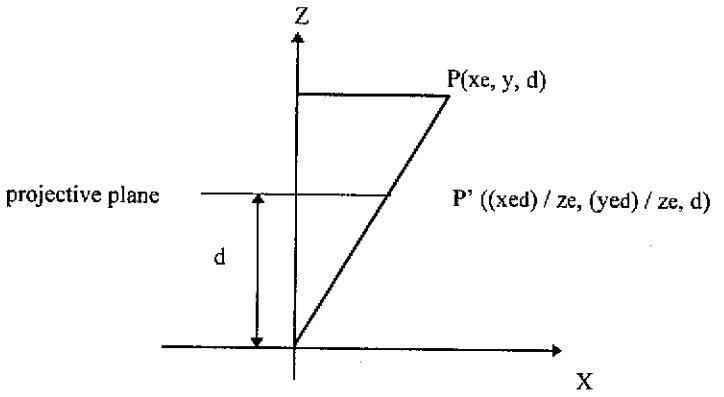

Fig. 1 The projection of point P

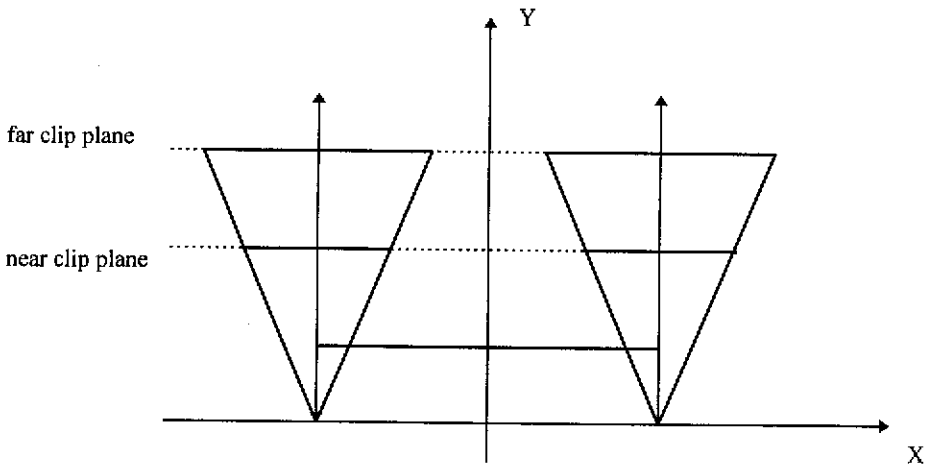

Fig. 1 The top view of the left and the right viewing frustum

To take the advantage of the man's ability to synthesize stereoscopic display, the computer must generate a couple of images, one for each eye, at two slightly different viewpoints. To simplify the calculation while preserving effective and comfortable stereoscopic display, we assume that the sight-lines from the two eyes are parallel, with an offset along horizontal direction (x direction) [ii] (see Fig. 2).

With this assumption, the right-eye coordinate system can be readily set up by a translation of the left-eye coordinate system in x direction. If the coordinates of a point in the left-eye coordinate system are (x_e, y_e, z_e), then the coordinates of the same point in the right-eye coordinate system will be (x_e, y_e, z_e), and its projection point will be ((x_e - a) d / z_e, y_{ed} / z_e, d)).

According to the above discussion, we can transform the pixel in the left image into the right image by adding an offset -ad/z_e in x direction. Since a and d are constant, the offset is only determined by the z coordinate of the visible point at each pixel.

Obviously, the nearer an object is to the eye, the greater its stereo disparity offset will be.

2.0 TRANSFORM THE LEFT IMAGE INTO THE RIGHT IMAGE

According to the visibility status in the left view and the right view, objects in the scene fall into four categories:

- visible to both the left eye and the right eye
- visible to the left eye and invisible to the right eye
- visible to the right eye and invisible to the left eye
- invisible to both the left eye and the right eye

In this paper, we only consider view-independent shading, so visible objects appear in different views at same intensity according to this assumption. Note that most polygons are visible to both the left eye and the right eye, providing great coherence between the left view and the right view. Thus, the pixels in the left image corresponding to the common visible objects in both views can be directly transformed into the right image by an offset of ad/z_e in the x direction.

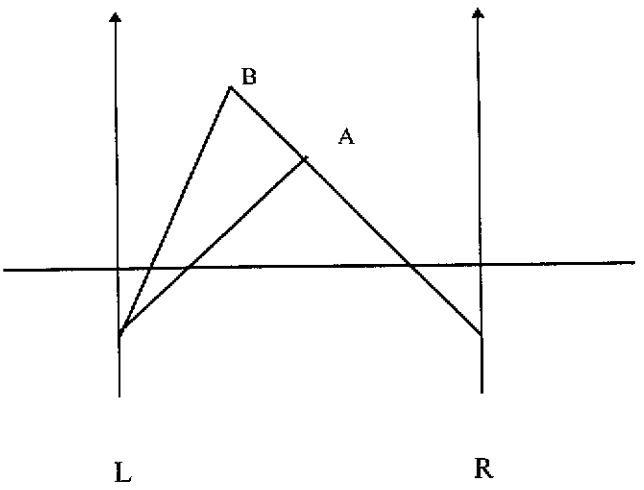

Fig. 3 Point B is visible in the left image and invisible in the right image

It is found that the most costly step in transformation is to calculate the offset for each pixel. It involves one division operation which may be slow. Note that the Z coordinate value preserved in the hardware z-buffer is in a discrete form (in IRIS, it's a 24-bit data). Thus the offset relative to each possible value can be calculated in advance and kept in a look-up table. When the pixels are transformed, we need only fetch the offset value from the table. This saves a lot of time. For polygons which are visible to the left eye and invisible to the right eye, the transformation of the left image may cause different pixels in the left image projected on the same location in the right image. This

is shown in Fig. 3. Nevertheless, since the left image is generated with hardware z-buffer algorithm, the intensity and the z-coordinate of the visible point at each pixel of the left image are available, the z-coordinate information associated with these pixels can be used to resolve visibility after the transformation.

The rendering of polygons which are invisible to the left eye and visible to the right eye warrants special attention. To the left image, point B is invisible. So the information of point B is not reserved and the intensity of point B can't be got from the left image. This is why the holes occur when the left image is transformed into the right image (Fig. 4).

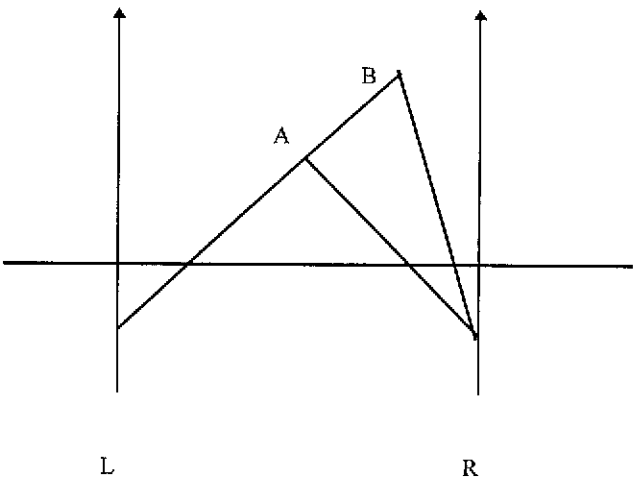

Fig. 4 Point B is invisible in the left image and visible in the right image

Note that holes may also arise by local image expansion. When we zoom in a polygon, the visible area of the polygon will be expanded and the holes occur. We will discuss how to distinguish and fill the two kinds of holes in the next section. The holes must be recorded during the transformation of the left image. For each sacnline on the right image, we search for the hole segments that may exist after the transformation. A hole segment is recorded by the position of its start pixel Xstart and end pixel Xend, the number of pixels on the segment, the y coordinate of the current scanline, as well as a pointer to the next hole segment of the same hole. This is shown in Fig. 5. Adjacent hole segments between consecutive scanlines are merged together to form large homogeneous hole regions.

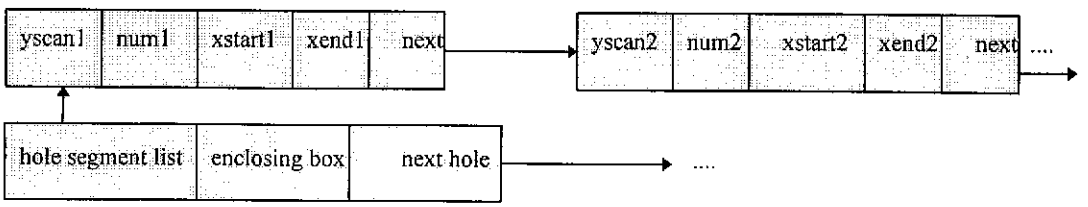

Fig. 5 The data structure of a hole and a hole segment

To facilitate the merging operation, an active hole list is maintained. If a hole is just modified, it is at the top of the list. The organization of the list shortens the searching

time for merging as the holes on the next scanline have great probability to join the holes on the front of the list.

The polygons which are invisible to both the left eye and the right eye are insignificant to both views and are just ignored in the generation of the right image.

3.0 FILLING THE HOLES ON THE RIGHT IMAGE

From above discussion, we know that holes on the right image fall into two classes:

- holes arise due to local image expansion
- holes arise due to visibility change of a few objects

It is a nontrivial problem to distinguish these two kinds of holes on the right image. Previous methods rarely address this problem. It is, however, significant to determine the exact class of the hole. As the holes arise due to local image expansion can be filled by linear interpolation of the intensities of neighboring pixels, it costs less time. On the other hand, filling the second kind of holes is difficult because we have to search the whole scene to find the polygons covering these holes. It is easily found that the neighbouring pixels around the hole of the first class depict the visible points from the same polygon. Nevertheless, the current z-buffer reserves only the depth information of the visible point associated with each pixel. The polygon identifier of each visible point is not available.

In order to solve this problem, we introduce a new buffer - identifier buffer (ID buffer). In this buffer, we record the identifier of the visible polygon at each pixel. The ID buffer is filled while the frame buffer and the z buffer are being processed. A modified z-buffer algorithm can be described as follows:

```
for each pixel do {
  if(polygon[k].z < zbuffer[i][j].z){
    frame-buffer[i][j].R = polygon[k].R;
    frame-buffer[i][j].G = polygon[k].G;
    frame-buffer[i][j].B = polygon[k].B;
    zbuffer[i][j] = polygon[k].z;
    ID-buffer[i][j] = polygon[k].id;
  }
}
```

With this ID buffer, it is easy to distinguish the two kinds of holes. In our system, we assume that all polygons have no holes. Polygons with holes can be partitioned to meet this assumption. So if the neighbouring pixels of a hole on the right image are related to the same polygon, the hole must arise by local image expansion. Beside its advantage for detecting the first class of holes, the ID buffer also helps in searching for the

polygons which can be seen through the second class of holes. Obviously, polygons visible through the hole lie certainly behind the polygons surrounding the hole. If these surrounding polygons are identifiable, we can clip off a lot of polygons that lie in front of them from our consideration.

As the scene may consist of enormous amount of polygons, it is important to provide a quick sort of all polygons according to their distance to the view point, which may move dynamically in the environment. The BSP algorithm, developed by Fuchs, Kedem and Nalor, provides an extremely elegant and simple way to determine visibility priority among polygons in a scene [ix, x]. A modified in order traversal of this tree provides for O(n) back-to-front ordering from an arbitrary viewpoint. Note that when the successive positions of the viewpoint are within the same sub-region formed by binary space partitioning, the order of all polygons in the scene will remain constant.

Nevertheless, not all the polygons lying behind the surrounding polygons are visible through the holes. For each hole, the polygons whose projected area on the screen do not cover the hole can be rejected. To accelerate the searching process, we associate each polygon with an enclosing box and organize all objects in the scene into a hierarchical tree. Each node on the tree has a flag whose initial value is zero. When the projected area of the enclosing box of a tree node is outside the hole, the flag of the node together with all nodes of its sub-tree will be set to a negative value without any further tests. Otherwise, we set the flag of the current node to a positive value and the non terminal nodes of its sub-tree are tested recursively. Finally, the flag of all unvisited terminal nodes are set to a positive value indicating that the projected area of those polygons may cover the hole.

For each hole, we then search the sorted polygon list from the position of the polygons surrounding the hole to the back, the nearest polygon is checked first. If its associated flag is negative, we jump to the next polygon, otherwise, we check the projected area of the enclosing box of the current polygon to see if it can be separated from the enclosing box of the concerned hole. If the test fails, the polygon is projected onto the screen and its projected area is scan converted by adopting a modified scan line algorithm. Each scanline is coincident with one of the hole segment of the hole and only these pixels on the scanline which lie simultaneously inside both the polygon and the hole segment need be scan converted. The intensities of the polygon at those pixels are calculated and the appropriate values are filled into the frame buffer, z-buffer and ID-buffer accordingly. The hole segment is then updated to reflect the current status. If a hole segment is completely covered by the current polygon, it is deleted from the hole

Note that the hole may be partially covered by the current polygon. Before processing the next polygon, the enclosing box of the hole need be updated and the above process continues until the hole is completely filled.

4.0 IMPLEMENTATIONS AND RESULTS

The proposed algorithm was implemented on an IRIS 4D/320 VGX workstation with a pair of stereo view 3D glasses. 3D glasses offer an inexpensive solution to the problem

of presenting 3D, dynamic, color imagery on a single screen. Scene is slightly shifted to the left for the right eye and to the right for the left eye [ii].

The above described algorithm can be sketched out as follows:

```
Build a BSP tree
Set up the object hierarchy
Initialize offset look-up table
for(different viewpoint do){
        rendering the left image using hardware z-buffer
        transform the left image into the right image
        distinguish the two kinds of holes
        fill the holes on the right image
}
```

We used different complex scenes to test the performance of the algorithm. If the scene is very simple, the generation of the right image may be even slower than that of the left image. This is because the hardware z-buffer is implemented very fast. When the scene to be rendered becomes more and more complex, the generation time of the left image is becoming longer than that of the right image. The rendering time of the right image is almost constant (see table below):

Polygon Number	Rendering time of left image (sec.)	Rendering time Right image (sec.)	of Total time
100	0.002	0.052	0.054
3000	0.06	0.057	0.137
4000	0.08	0.06	0.14

CONCLUSIONS

In this paper, we present an accelerated method for fast generation of stereoscopic display. This method makes fully use of the coherence between the left view and the right view. ID buffer is used to distinguish the two kinds of holes and while combining with BSP tree and hierarchical enclosing box technique, it helps us to fill the holes on the right image efficiently. The generation of the right image is almost independent of the scene complexity and the rendering time is proportional to the

image resolution. This method can also run parallely. The future research work should take the view-dependent shading of the scene into account. Presently, our method can only process view-independent shading. But the view-dependent shading (such as specular reflection) will enhance the reality of the image. It is hoped that the progressive refinement approach will fulfill this task.

REFERENCES

[i] Harry Veron, David A, Southard, Jeffrey R. Leger, John L. Conway, "Stereoscopic Displays for Terrain Database Visualization", Sterescopic Displays and Applications, Proc. SPIE 1256, 1990, pp 124-135.

[ii] Thaut Tessman, "Perspective on Stereo", Stereoscopic Displays and Applications. Proc. SPIE 1256, 1990, pp 22-27.

[iii] Shenchang Eric Chen, Lance Williams, "View Interpolation for Image Synthesis", Computer Graphics, 1993, pp 279-288.

[iv] Greene, N. and M.Kass. "Approximating Visibility with Environment Maps", Technical Report 41, 1993, Apple Computer Inc.

[v] Miller, G. and S.E. Chen, "real-time Display of Surroundings Using Environment Maps", Technical Report 42, 1993, Apple Computer Inc.

[vi] Jin Xiaogan, Wan Huagen and Peng Qunsheng, "Accelerating Ray Tracing Through Polygon Projection", Proc. CAD/Graphics, Beijing, 1993, pp 147-150.

[vii] Stephen J. Adelson, Larry F. Hodges, "Stereoscopic ray-tracing", The visual computer, Oct. 1993, pp 127-144.

[viii] Stephen J. Adelson, Larry F. Hodges, "Generating Exact Ray Tracing Animation Frames by Re-projection", IEEE Computer Graphics and Applications, May 1995, pp 43-52.

[ix] Fuchs, H., Kedem, A., and Nalyor, B. "On visible Surface Generation by A Priori Tree Structures", Proc. SIGGRAPH '80. In Computer Graphics, 14:3, July 1980, pp 124-133.

[x] Fuchs, H., Abram, G., and Grant, E. "Near real-time Shaded Display of Rigid Objects", Proc. SIGGRAPH '83, In Computer Graphics, 17:3, July 1983, pp 65-72.