Fast Volume Rotation using Binary Shear-Warp Factorization

Balázs Csébfalvi

Department of Control Engineering and Information Technology,
Technical University of Budapest,
Budapest, Műegyetem rkp. 11, H-1111, HUNGARY
cseb@seeger.fsz.bme.hu

Abstract. This paper presents a fast volume rotation technique based on binary shear-warp factorization. Unlike many acceleration algorithms this method does not trade image quality for speed and does not require any specialized hardware either. In order to skip precisely the empty regions along the rays to be evaluated a binary volume is generated indicating the locations of the transparent cells. This mask is rotated by an incremental binary shear transformation, executing bitwise boolean operations on integers storing the bits of the binary volume. The ray casting is accelerated using the transformed mask and an appropriate lookup-table technique for finding the first non-transparent cell along each ray.

1 Introduction

Direct volume rendering is a flexible but computationally expensive technique for visualization of 3D density arrays. Because of the huge number of voxels to be processed the recent software-only acceleration methods are not fast enough for interactive applications. Real-time frame rates can be achieved using large multi-processor systems [1][4][10][13], but they are not widely used because of their high costs. In the last two decades several accelerated volume-rendering techniques have been proposed, which exploit the coherence of the data set.

Early methods use hierarchical data structures like pyramids, octrees or K-d trees [2][6][11] to quickly traverse the transparent regions decreasing the number of samples to be evaluated. Hierarchical data structures are used in homogeneity acceleration techniques as well [2][14], which apply a simplified approximate evaluation for the homogeneous regions.

Recent algorithms like distance transformation based methods [15][16] or Lacroute's shear-warp factorization projection [5] concentrate on a more precise skipping of empty ray segments instead of approximated evaluation of homogeneous regions. The main advantage of these techniques over hierarchical methods is the applied encoding scheme, since the information about the empty cells, is available with the same indices used for the volume data. On the other hand there is no additional computational cost for handling a hierarchical data structure during the ray traversal. Applying these acceleration methods the rendering time can be reduced significantly, but the frame rates are still far from interactivity. Although there are surface oriented algorithms which provide real-time rotation, their limitation is the lack of the opacity manipulation [17][18].

2 The Algorithm

The software-only acceleration technique presented in this paper can also be considered as a surface oriented algorithm since interactive frame rates can be achieved if the opacity function is binary or near binary (the non-zero opacities are near one). On the other hand unlike the previous iso-surface methods it supports the high quality transfer function based rendering as well. In a practical application the surface rendering can be used as a fast preview, where the user can set the appropriate viewing angle interactively and afterwards the final image is rendered using the alpha-blending evaluation according to the selected transfer function.

2.1 Preprocessing

The input data of a direct volume rendering process is a spatial density function $f : R^3 \rightarrow R$ sampled at regular grid points, yielding a volume $V : Z^3 \rightarrow R$ of size $X \times Y \times Z$, where

$$V_{i,j,k} = f(x_i, y_j, z_k).$$

In the classification process, according to the density values different attributes like opacity or color are assigned to each voxel. The opacity function maps the volume V onto a classified volume $C : Z^3 \rightarrow [0,1]$ of the same size. In order to handle the empty cells efficiently many acceleration algorithms create a binary volume assigning zero to the transparent and one to the non-transparent cells, where a cell is considered transparent if all of its eight corner voxels have zero opacities. In our method the definition of a cell depends on the principal direction of the viewpoint. Without loss of generality, we assume that the principal direction is the *z-axis*. In this case, the proposed algorithm resamples the volume only in planes $z = z_k$, where k = *0,1,2, ... Z-1*. The density samples on these planes are computed from the densities of the four closest voxels using bilinear interpolation. The opacity of the sample is non-zero if at least one of the four voxels is opaque, thus the binary volume $B : Z^3 \rightarrow \{0,1\}$ of size *(X-1)* $\times$ *(Y-1)* $\times$ Z is defined the following way:

$$B_{i,j,k} = \begin{cases} 1 \text{ if } C_{i,j,k} > 0 \text{ or} \\ \quad C_{i+1,j,k} > 0 \text{ or} \\ \quad C_{i,j+1,k} > 0 \text{ or} \\ \quad C_{i+1,j+1,k} > 0 \\ 0 \text{ otherwise.} \end{cases}$$

2.2 Ray Casting

The binary volume B can be stored in an integer array, where an integer represents a segment of a bit row parallel to the *z-axis* In the special case, when the viewing direction is exactly the *z-axis* the volume can be rendered very efficiently by parallel ray casting, since the problem of finding the first non-transparent voxel hit by a ray can be reduced to the problem of finding the first non-zero bit inside an integer. The optimal solution to this problem is the direct addressing of a lookup table by the given integer which contains the position of the first non-zero bit for all the possible combinations. For a typical integer size like 32 bits it would require the allocation of a 2^{32} byte array which cannot be used in practice. Instead of this, the lookup table can store the offset of the first non-zero bit inside one byte for each byte combination and it can be addressed by the first non-zero byte of the given integer. Assuming that the most significant bit is the nearest one to the viewer the first non-zero byte can be determined by binary search. The following routine provides the position of the first non-zero bit, where the size of an integer is supposed to be 32 bits:

```
int Depth(int segment) {
  if(segment < 0x00010000) {
    if(segment < 0x00000100) return 24 + lut[segment];
    else return 16 + lut[segment >> 8];
  } else {
    if(segment < 0x01000000) return 8 + lut[segment >> 16];
    else return lut[segment >> 24];
  }
}
```

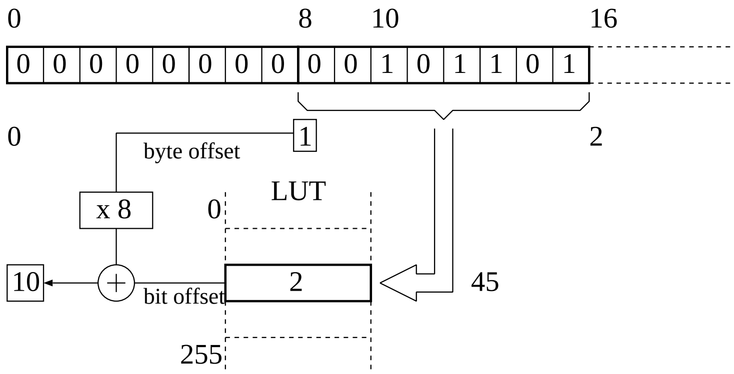

Fig. 1. An example for a LUT entry.

Usually one integer is not enough for storing a complete row of the binary volume B, thus the segments of the rows stored in integers have to be checked sequentially and the routine *Depth* is called only for the first non-zero integer.

2.3 Shear-Warp Factorization

The previous method works only for a special case but it can be extended to viewing directions, where the principal component is the coordinate z using binary shear-warp factorization. This transformation effectively moves the bits of the binary volume perpendicularly to the viewing direction. In an interactive volume rendering application the volume is required to be rotated continuously by little difference angles, in order to perceive the topology of the surfaces much better then in a static image. If the difference angle is small enough then there is no slice in the binary volume which has to be shifted by more than one bit. In this case, one shear operation can be performed very efficiently, since just bitwise operations need to be executed on neighboring integers. That is the reason why our method shears the binary volume B incrementally, applying a technique similar to the method proposed by Cohen-Or and Fleishman [1]. They used their so called incremental alignment algorithm in order to reduce the communication overhead in a large multi-processor architecture supporting shearing of volumes. Since some bits can be shifted out of the integer array storing the binary volume B, it has to be extended by $Z/2$ rows filled with zero values along the *x-axis* and along the *y-axis* as well in both directions. This extended array is defined as: *int mask*[*depth*][*height*][*width*], where $depth = (Z + 31)\ div\ 32$, $height = Y + Z$ and $width = X + Z$. The following routine demonstrates, how to execute one shear step in the left direction, processing 32 voxels in each step of the internal loop:

```
void ShearLeft() {
  int i, j, k;
  for(k = 0; k < depth/2; k++) {
    for(j = 0; j < height; j++) {
      for(i = 0; i < width-1; i++)
        mask[k][j][i] = mask[k][j][i] & shift_x[k] |
                        mask[k][j][i+1] & ~shift_x[k];
      mask[k][j][width-1] &= ~shift_x[k];
    }
  for(k = depth/2; k < depth; k++) {
    for(j = 0; j < height; j++) {
      for(i = width-1; i > 0; i--)
        mask[k][j][i] = mask[k][j][i] & shift_x[k] |
                        mask[k][j][i-1] & ~shift_x[k];
      mask[k][j][0] &= ~shift_x[k];
    }
}
```

For the sake of clarity this routine is not optimized, but it can be improved introducing local pointer variables in order to avoid unnecessary array addressing, and on the other hand only that part of the extended mask needs to be sheared which contains the non-zero bits representing the non-transparent cells. The integer array *shift_x* is defined as: *int shift_x*[*depth*] and it stores a binary vector of size Z indicating those z positions where the corresponding slices have to be shifted in the given shearing phase. There is also such an array denoted by *shift_y* for the y direction. In order to determine the offsets of the slices in different shearing phases, another two arrays are introduced for storing the real translations along the x-*axis* and the y-*axis* and they are defined as: *double trans_x*[*Z*] and *double trans_y*[*Z*] respectively. These translation arrays contain the x and y coordinates of those points, where the $z = z_k$ planes intersect the 3D line connecting the point p_0(*trans_x*[0],*trans_y*[0],0) with point p_1(*trans_x*[*Z-1*],*trans_y*[*Z-1*],*Z-1*). Initially, this line aligns to the z-*axis*, thus the translation arrays contain zeros. Before executing a binary shear operation the *shift* vectors are evaluated in advance according to the rotation direction. For example, when a clockwise rotation around the y-*axis* is needed, the point p_0 is translated by one along the x-*axis* into negative direction and p_1 is translated as well, but into positive direction. After this, the intersection points of the line connecting the new p_0 and p_1 and the planes $z = z_k$ are computed anew and the coordinates are stored in the translation arrays. The new binary shift vectors are determined according to these translation values. For example, the new *shift_x* array can be computed using the following routine:

```
void ComputeNewShiftX()
  {
    double x0 = trans_x[0] -= 1.0, x1 = trans_x[Z-1] += 1.0;
    int bit = 0x80000000, l = Z - 1;
    for(int z = 0; z < Z; z++) {
      double x = (x0 * (l - z) + x1 * z) / l;
      if(floor(x) != floor(trans_x[z])) shift_x[z/32] |= bit >> (z % 32);
      else shift_x[z/32] |= ~(bit >> (z % 32);
      trans_x[z] = x;
    }
  }
```

If the floors of the previous and the new translation values are not the same then the corresponding bit is set to one in the *shift_x* array, indicating that the associated slice needs to be shifted in the next binary shear operation. Since the difference between the old and new translations is the greatest in the plane $z = z_0$ (or $z = z_{Z-1}$) the difference cannot be greater than one in the intermediate z points, thus there is no slice which needs to be shifted with more than one bit.

2.4 Resampling

Using the transformed binary volume an intermediate image of size *width* $\times$ *height* is generated casting the rays from the grid points. Due to the shear transformation the *Depth* routine can be applied in the general case as well, since the segments of the rows perpendicular to the temporary image plane are stored in integers. The position of the first non-zero bit in a row determines the index i of the $z = z_i$ plane, where the first opaque sample is located along the corresponding ray. The accurate location of this sample is computed taken into account the exact translation values at the given depth z. In order to calculate the density in this sample point location, bilinear interpolation is used for the four closest voxels. Since the opacity of the sample is not necessarily one so the ray has to be traced further and evaluated according to the transfer function. If the volume contains internal empty regions (like a human skull) it makes sense to use the binary volume for evaluating the rest of the samples. First, the integer representing a row of bits is copied into a temporary variable, and whenever a non-zero bit has been processed it is set to zero, thus the routine *Depth* can be used again for finding the z position of the next non-transparent cell. Having the intermediate image generated it has to be warped in order to produce the final image, which is the parallel projection of the volume.

3 Extensions

The presented method can be extended to arbitrary viewing directions since a binary volume can be created for each principal direction. Applying an appropriate scaling for the slices perpendicular to the principal component, the algorithm can be used for perspective projection as well. The next two subsections describe two further improvements which could be useful in a practical implementation.

3.1 Rotation of Large Data Sets

Due to the incremental shear transformation the effective speed of the rotation could be low, especially processing larger volumes (256^3). Since most of the time is used for rendering, a possible solution to this problem is to render the volume after a couple of incremental shears. Increasing the size of the data set the ratio of the rendering and shearing times is getting lower, thus this strategy is not the best one. Another alternative is the introduction of super cells, which are square regions in the slices perpendicular to the principal direction. In the binary volume, one is assigned to the corresponding super cell if at least one voxel in it is opaque. Increasing the size of the super cell the shear transformation becomes faster but the rendering process slows down since the routine *Depth* does not necessarily return the exact depth only a lower bound, so the rays have to be traced further until having found a non-transparent sample. In the next section the performance is analyzed investigating the optimal cell size for data sets of different sizes.

3.2 Adaptive Thresholding

The primary limitation of the presented technique is that the volume has to be classified in advance in order to generate the binary volume, and afterwards the opacity function cannot be modified in a flexible way. Supposing that, the user wants to operate with a fixed number of transfer functions an appropriate density encoding scheme can be used to allow rapid switching between them. Each transfer function has a lower density threshold, where below this threshold zero opacity is assigned to the given sample. Assume that we want to use only three transfer functions. The lower density thresholds divide the density domain into intervals I_0, I_1, I_2, I_3 sorted in ascending order by their borders with increasing index. To each interval the two bit binary format of the corresponding index is assigned as a unique code. The code of a cell is defined as the code of the interval which contains the highest corner voxel density. The cell codes are stored in an integer array which is similar to the *mask* array but it contains two bits for each cell instead of one. This array can be sheared as well, but the bits of a code should always be moved at the same time in order to avoid the cutting of the codes. In the rendering phase the routine *Depth* must use the appropriate lookup table depending on the bit pattern to be searched for. Whenever the user changes the transfer function the variable *lut* has to be set to the address of the corresponding lookup table. This encoding scheme allows rapid access to the first non-transparent cell along the viewing ray independently from the selected transfer function. Let us take an example from the medical imaging practice, where only four materials (air, fat soft tissue, and bone) can be separated according to the *Hounsfield densities*[3][9]. In this case it makes sense to divide the density domain according to the lower density threshold of fat, soft tissue, and bone respectively. For example, having selected a transfer function which assigns zero opacities to the samples below the lower threshold of the soft tissue, only the codes 10 and 11 will be searched for in the binary volume, precisely skipping the transparent cells. In this case, the corresponding lookup table contains the bit offset of the first 10 or 11 pattern inside the given byte. The presented density encoding scheme does not effect significantly the performance and allows fast switching between the predefined transfer functions.

4 Implementation

The proposed fast rotation technique has been implemented in C++ and it has been tested on an SGI Indy workstation. Table 1 summarizes the running time measurements for a CT scan of a human head and Table 2 contains the test results for a higher resolution volume of the same data. The applied transfer function assigns high opacities to the voxels representing the bone thus rays terminate right after reaching the boundary of the skull (Figure 2).

cell size	shearing time	rendering time	frame rate
1	0.019 sec	0.114 sec	6.87 Hz
2	0.005 sec	0.107 sec	8.64 Hz
3	0.002 sec	0.118 sec	8.11 Hz
4	0.001 sec	0.156 sec	6.26 Hz

Table 1. Test results for the CT head of size $128 \times 128 \times 113$.

Note that, the optimal super cell size is not necessarily the one which the highest frame rate belongs to, since with larger super cell size the effective rotation speed is higher. In order to rotate the volume continuously the cell size

cell size	shearing time	rendering time	frame rate
1	0.160 sec	0.590 sec	1.21 Hz
2	0.040 sec	0.492 sec	1.81 Hz
3	0.017 sec	0.535 sec	1.78 Hz
4	0.009 sec	0.709 sec	1.37 Hz

Table 2. Test results for the CT head of size $256 \times 256 \times 225$.

must be set small and higher rotation speed can be achieved by setting larger cell size producing approximately the same frame rates.

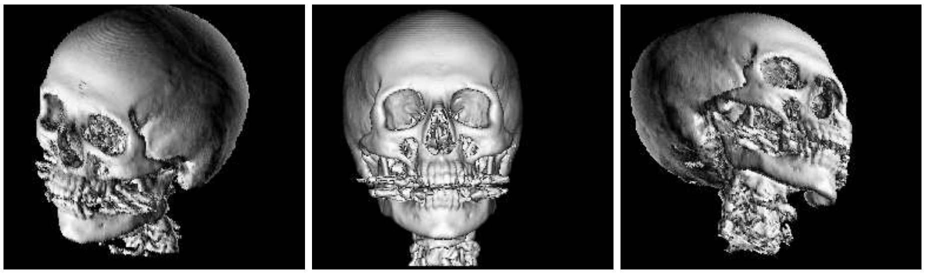

Fig. 2. Interactive rotation using fast iso-surface rendering.

Using transfer functions which assign low opacity values to different tissues the rendering time increases drastically, since after skipping the empty regions the alpha-blending evaluation of the semi-transparent segments is computationally very expensive. Although setting larger super cell size higher rotation speed can be achieved in the fast previewing phase the high quality rendering slows down since the binary volume contains less precise information about the transparent cells. Table 3 and Table 4 show the average rendering times for the low and high resolution data sets respectively using three different transfer functions as demonstrated in Figure 3.

cell size	transfer function A	transfer function B	transfer function C
1	0.36 sec	0.19 sec	0.18 sec
2	0.39 sec	0.21 sec	0.21 sec
3	0.43 sec	0.25 sec	0.24 sec
4	0.51 sec	0.32 sec	0.31 sec

Table 3. Rendering times for the volume of size $128 \times 128 \times 113$.

Fig. 3. Alpha-blending rendering using different transfer functions.

cell size	transfer function A	transfer function B	transfer function C
1	1.97 sec	1.02 sec	0.99 sec
2	1.93 sec	1.01 sec	0.96 sec
3	2.08 sec	1.09 sec	1.08 sec
4	2.44 sec	1.45 sec	1.46 sec

Table 4. Rendering times for the volume of size $256 \times 256 \times 225$.

5 Conclusion

In this paper a fast volume rotation technique has been presented which provides interactive frame rates without using any specialized hardware support. Real-time rotation can be achieved using binary or near binary opacity function, when rays terminate right after reaching an opaque surface. In this sense the proposed technique is a surface oriented algorithm but unlike other interactive iso-surface methods it significantly speeds up the classical transfer function based ray casting. Because of the precise skipping of empty regions it is approximately as efficient as the classical shear-warp algorithm based on run-length encoding. In a practical implementation it can be applied as a fast previewer rendering the iso-surface defined by the lower density threshold of the selected transfer function, where the viewing direction can be set interactively, and the final image is rendered using the alpha-blending evaluation. The effective rotation speed can be increased by setting a larger super cell size and applying the proposed adaptive thresholding extension the user can switch rapidly between predefined transfer functions.

Acknowledgements

This work has been funded by V^{is}M^{ed}. Please refer to `http://www.vismed.at` for further information on this project. This paper has also been supported by the National Scientific Research Fund (OTKA ref.No.: F 015884) and the Austrian-Hungarian Action Fund (ref.No.: 29ö4 and 32öu9). The CT scan was obtained

from the Chapel Hill Volume Rendering Test Dataset. The data was taken on the General Electric CT scanner and provided courtesy of North Carolina Memorial Hospital.

References

1. Daniel Cohen-Or and Shachar Fleishman. An incremental alignment algorithm for parallel volume rendering. *Computer Graphics Forum (EUROGRAPHICS '95 Proceedings)*, pages 123–133, 1995.
2. John Denskin and Pat Hanrahan. Fast algorithms for volume ray tracing. *Workshop on Volume Visualization*, pages 91–98, 1992.
3. R.A. Drebin, L. Carpenter and P. Hanrahan. Volume rendering. *Computer Graphics (SIGGRAPH '88 Proceedings)*, 22:65–74, 1988.
4. Jürgen Hesser, Reinhard Männer, Günter Knittel, Wolfgang Strasser, Hanspeter Pfister and Arie Kaufman. Three architectures for volume rendering. *Computer Graphics Forum (EUROGRAPHICS '95 Proceedings)*, pages 111–122, 1995.
5. Philippe Lacroute and Marc Levoy. Fast volume rendering using a shear-warp factorization of the viewing transformation. *Computer Graphics (SIGGRAPH '94 Proceedings)*, pages 451–457, 1994.
6. David Laur and Pat Hanrahan. Hierarchical splatting: A progressive refinement algorithm for volume rendering. *Computer Graphics (SIGGRAPH '91 Proceedings)*, pages 285–288, 1991.
7. Marc Levoy. Display of surfaces from ct data. *IEEE Computer Graphics and Application*, 8:29–37, 1988.
8. Marc Levoy. Efficient ray tracing of volume data. *ATG*, 9(3):245–261, 1990.
9. Derek R. Ney, Elliot K. Fishman, Donna Magid and Marc Levoy. Computed tomography data: Principles and techniques. *IEEE Computer Graphics and Application*, 8, 1988.
10. Peter Schröder and Gordon Stoll. Data parallel volume rendering as line drawing. *Workshop on Volume Visualization*, pages 25–32, 1992.
11. K.R. Subramanian and Donald S. Fussell. Applying space subdivision techniques to volume rendering. *IEEE Visualization '90*, pages 150–159, 1990.
12. L. Szirmay-Kalos (editor). *Theory of Three Dimensional Computer Graphics.* Akadémia Kiadó, Budapest, 1995.
13. Guy Vézina, Peter A. Fletcher and Philip K. Robertson. Volume rendering on the maspar mp-1. *Workshop on Volume Visualization*, pages 3–8, 1992.
14. Jason Freund and Kenneth Sloan. Accelerated volume rendering using homogeneous region encoding. *IEEE Visualization '97*, pages 191–196, 1997.
15. D. Cohen and Z. Shefer. Proximity clouds - an acceleration technique for 3D grid traversal. *TR FC93-01, Ben Gurion University, Israel*, 1993.
16. K. Zuiderveld, A. Koning, Viergever and A. Max. Acceleration of ray casting using 3D distance transformation. *Visualization in Biomedical Computing*, pages 324–335, 1992.
17. Jae-jeong Choi and Yeong Gil Shin. Efficient Image-Based Rendering of Volume Data. *TR http://cglab.snu.ac.kr/ jjchoi/ibr.html, Seoul National University, Korea*, 1998.
18. Björn Gudmundsson and Michael Randén. Incremental generation of projections of CT-volumes. *In First Conf. on Visualization in Biomedical Computing, Atlanta*, 1990.