SCAN LINE GRAPHICS GENERATION ON THE MASSIVELY PARALLEL PROCESSOR

John E. Dorband

NASA/Goddard Space Flight Center/635
Greenbelt, MD 20771

ABSTRACT

This paper describes how we have implemented a scan line graphics generation algorithm on the Massively Parallel Processor (MPP). Pixels are compute in parallel and their results are applied to the Z buffer in large groups. To perform pixel value calculations, facilitate load balancing across the processors and apply the results to the Z buffer efficiently in parallel requires special virtual routing (sort computation[1,2]) techniques developed by the author especially for use on single-instruction multiple-data (SIMD) architectures.

Keywords: Graphics, scanline, Z-buffer, sorting, sort computation, SIMD, massively parallel, MPP, load balancing.

INTRODUCTION

A scan line graphics generation algorithm basically determines the brightness of pixels in a simulated 3-D scene a scan line at a time. The brightness value of a pixel is based on the surface brightness of simulated polygon which would be seen through the pixel. Triangles are the polygons used here. Only the triangle nearest the pixel on the simulated viewing screen will be seen through the pixel. Therefore a Z buffer is setup to accumulate the values of the closest polygons to the viewing screen. It is actually not necessary to only process one scan line at a time. On the MPP, a subset of triangles at a time are processed for all scan lines that these triangles cover. This is done by projecting each triangle onto the viewing screen and determining which scan lines it covers. Then the pixels of each scan line that the triangle covers is determined. This results in pixels of different values and distances from the viewing screen which are loaded into the Z buffer. When all triangles are processed the Z buffer can be displayed as an image.

To efficiently compute pixel values in parallel an efficient load balancing method was developed so as many processors as possible could be kept busy. This is of importance when greater parallelism can be realize by duplicating data into more processors. This is made complex when it is determined that data in certain processors is of no more computational use, randomly leaving processors without work to do. Therefore the data must be moved in such a way that it is known that when the data is slid to new processors it will not be written over useful data. This movement or compression is done by sorting. Although efficiency of processor usage is of primary interest here, efficiency of data movement is also of importance. Therefore the inefficiencies in the use of sorting are also considered. This has prompted the modification of the sorts used. This involves a preprocessing (scout) step which determines how much of the sort is necessary to provide sufficient contiguous space to duplicate the data. Once this has been determined a sort is used to compress the data which can be terminated early based on the information derived by the scout step. This then allows one the ability to reasonably efficiently keep as many processors as possible busy.

PROJECTION CALCULATION

The projection calculation converts the three coordinates of the three corners of a triangle in a 3 dimensional viewing space into two coordinates on the viewing screen and a range from the view point. Given the coordinates of the triangle ($X_1, Y_1, Z_1, X_2, Y_2, Z_2, X_3, Y_3, Z_3$), the coordinates of the view point (X_v, Y_v, Z_v), and the projected coordinates ($X'_1, Y'_1, R_1, X'_2, Y'_2, R_2, X'_3, Y'_3, R_3$) the following equations do the conversion from 3-D coordinates to 2-D projected coordinates. The first set of equations rotates the triangles in space so that the viewing axis lines up with the Z axis. Thus the view point will lie along the Z axis.

$$X''_i = X_i * x_v + Z_i * z_v, \quad Y''_i = Y_i,$$

$$Z''_i = X_i * z_v - Z_i * x_v,$$

$$X''_v = 0, \quad Y''_v = Y_v, \text{ and } Z''_v = \sqrt{Y_v^2 + Z_v^2},$$

where x_v, y_v, and z_v are normalized values of X_v, Y_v, and Z_v.

$$X'''_i = X''_i, \quad Y'''_i = Y''_i * y''_v + Z''_i * z''_v,$$

$$Z'''_i = Y''_i * z''_v - Z''_i * y''_v,$$

where y''_v and z''_v are normalized values of Y''_v, and Z''_v.

Thus the rotated coordinates of a triangle is X'''_1, Y'''_1, Z'''_1, X'''_2, Y'''_2, Z'''_2, X'''_3, Y'''_3, and Z'''_3. The rotated triangles are projected on to the screen which is the distance R_s from the view point. The following equations give the values for X', Y', and R' for each corner of a triangle.

$$R_v=\sqrt{X_v^2+Y_v^2+Z_v^2}\ ,\ \ X'=X'''\frac{R_s}{R_v},\ \ Y'=Y'''\frac{R_s}{R_v},$$

$$\text{and}\ \ R'=\sqrt{(X'''-X_v)^2+(Y'''-Y_v)^2+(Z'''-Z_v)^2}\ .$$

A brightness value (B) is also calculated for each triangle. The actual means of calculating it is not important, only that it exists and must be included with the rest of the information for each triangle.

SCAN LINE DETERMINATION

Once the projection calculations have been performed each triangle will be described by an X and Y coordinate and a range for each corner and a brightness for the entire triangle. This information will make up a triangle description record. These records will be duplicated so that there exists one copy of a triangle's description record for each scan line that intersects the triangle's projection onto the screen.

Assume that scan lines are parallel to the X axis. Then the corners of a triangle with the largest and smallest Y values define the range of scan lines that the triangle intersects. By recursively dividing this range in half and making records corresponding to the two halves, we will eventually have a record for each scan line in the range. The difficulty arises when this has to be done in parallel, especially when it is done on a large array of processors, like the MPP. The number of scan lines that a triangle overlaps is not the same for all triangles. This means that the rate of creation of new records is uneven across the processors and some sort of load balancing must be performed if one is to efficiently utilize large arrays of processors.

LOAD BALANCING

Load balancing consists of redistributing records across the processors when some processors contain more than one record. This is caused by creating more records in one area of the array of processors than in others. One can do this by moving all the records to one end of the array of processors, only one record per processor. Any left over records, if all processors have at least one record, can be saved in a stack. There several means by which the records can be moved(compressed) to one end of the array, but we have found that parallel bitonic sort is very efficient at doing this on the MPP. So the use of sort to load balance is what will be discussed here.

Actually the records are sorted to one end of the array so that there are two records per processor. Therefore if only half of the processors have any record in them, then half must have none.

The final step of the load balancing is to move one record from each processor that has two to a processor that has none by sliding them halfway across the array. This means that a complete sort has to be done and the data moved halfway across the array. Though the sort is efficient, there is no sense in doing a complete one if one doesn't have to.

Therefore, a scouting step was developed to determine how much of the sort needs to be performed so that records can be simply moved to empty processors. Simply implies moving one record form each processor that has two to a processor that has none by moving them all the same number of processors away form their original processor.

For an incomplete sort to be useful at least the following condition must be true. That for every group of processors, at least half of the processors must be empty. These groups must contain the same number of processors and all records within each group must be compressed to the same side of the array of processors of the group. The scout routine determines the shortest sort necessary to meet these conditions by performing a sort on a set of flags that represent where the records exist within the array. The difference from the sort being that after every merge step it checks to see if the required conditions have been meet.

Scan line determination is merely duplication of records, modification so that they represent different ranges of scan lines, and redistribution of records (load balancing). This is repeated until each record represents only on scan line.

PIXEL DETERMINATION

At this point each record represents a triangle and one scan line that it intersects. The range along the scan line which represents the part of the scan line that is covered by the triangle is determined. Then in the same way that scan lines ranges were reduced to individual scan lines, so pixel ranges are reduced to individual pixels. Analogous to scan line determination, pixel determination involves duplication of records, modification so that they represent different ranges of pixels, and redistribution of records (load balancing). Thus, each record will represent a triangle and a pixel that it covers. From each of these records a pixel record is created that contains the pixels location on the screen, the bright of the triangle, and the distance to the triangle as seen through the pixel.

Z BUFFERING

Many of the pixel records will represent the same pixel, but with different range and brightness and range values. The Z buffer is merely a collection of the records for which duplicate pixel records are eliminated. They are eliminated based on there range value. Only the pixel record with the smallest range is kept for each pixel. This is done using a sort computation function, sort minimum, which will flag the minimum range record for each pixel during the sort. All unflagged records can be mark as deleted.

IMAGE ASSEMBLY

The records in the Z buffer are then used to form a final image. Techniques for assembling data points into an image were developed previously in the process of developing algorithms of point plotting and raytracing on the MPP[3].

Since there may not be a Z buffer record for every pixel in the image, a template image must be created. This consists of a group of pixel records that contain a record for every pixel in the image. Image assembly is a two step operation, pixel value distribution and image organization. Both of these operation can be done with sort computation functions. Pixel value distribution is done with sort distribution. Z buffer records are flagged as containing valid data and image template records are not. Sort distribution copies data from Z buffer records to image template records. This however leaves Z buffer records interspersed with image template records. Thus the image can not be displayed in this form as is. Since image records are flagged as belonging to the image template and Z buffer records are not, the records can be sort with the image flag as the major key. This will separate the Z buffer records from the template records. At the same time the pixel location can be used as the minor key, which will order the pixels so that they can be displayed as a raster scan image.

CONCLUSION

This technique is in use on the MPP, which is a 2-D grid of 128 by 128 processors. We are generating 3-D renderings of elevation data. The data consists of a 512 by 512 grid of points which is converted into 524,288 triangles (see Color Plate II, p. 694). These triangles take from 45 seconds to 75 seconds to render, which is from 6 to 12 thousand triangles a second. Currently we are working on more efficient means of data movement and organization to increase its speed.

REFERENCES

1 Dorband, John E., *Sort Computation*, Frontiers 88 Conference Proceedings, September 1988.

2 Dorband, John E., *Sort Computation and Conservative Image Registration*, Ph.D. thesis, Pennsylvania State Univ., December 1985.

3 Dorband, John E., *3-D Graphic Generation on the MPP*, Proceedings of the 2nd International Conference on Supercomputing, Vol. II, pg 305-309, 1987.