Two Computer Graphics Systems for the Visualization of Pressure Distributions and Airflow in Wind Tunnel Experimentation

By

Carol T. Luan
and
W. N. Martin

Computer Science Report RM-85-01

April, 1985

Two Computer Graphics Systems for the Visualization of Pressure Distributions and Airflow in Wind Tunnel Experimentation

Carol T. Luan
and
W.N. Martin

Department of Computer Science
Thornton Hall
University of Virginia
Charlottesville, Virginia 22901

RM-85-1

April, 1985

This research was supported in part by NASA Langley Research Center through Grant NAG-1-242 and by the National Science Foundation through Grant ESC-83-07248.

Acknowledgement

We would like to thank Mr. Don Lansing of ACD, NASA Langley Research Center for his guidance and encouragement of Ms. Luan while she worked on the first part of this project at NASA Langley Research Center during the summer of 1984.

Abstract

A method is presented for displaying pressure distributions with planer color contours and general surface meshes. McAllister's algorithm for shape-preserving osculatory quadratic splines is the major approach.

A graphics system is also presented for visualization of solid obstacles immersed in a set of convective translucent particles. Approximation of particle density volumes and Torrence-Sparrow's method for computing surface reflectance are the major approaches.

CONTENTS

Chapter 1. Introduction

The list of computer graphics applications is enormous and growing rapidly. In this paper, two systems in engineering applications are presented. The first system is concerned with the pressure distribution on an airfoil surface in wind-tunnel testing. In an actual wind-tunnel test, the pressure values of a collection of points on the airfoil surface are measured and recorded. Our job is to help with the interpretation of the data obtained, by providing a visualization of the pressure distribution with color graphics.

A second system is concerned with the air flow in wind-tunnel testing. To assist with the observation of this flow, some opaque trace particles are also injected into the flow. If the trace particles are injected in a localized manner, the resulting stream can help visulize the air flow through a selected volume. With a wind-tunnel flow simulator, frames of three-dimensional volumes of air and trace particle densities throughout a sequence of time intervals can be generated. Our job is to provide a visualization of the particle flow through color graphics presentation. Our discussion of this second system will be subdivided into two parts. The first part deals with visualization of particles while the second part deals with visualization of a solid obstacle which can be immersed in the particle flow.

Although both systems are wind-tunnel systems, the first one centers attention on the flow on an airfoil surface, while the second system centers attention on the flow of injected particles in the wind-tunnel.

A graphics package has been written for the first system. This package is to be used by the aeronautic engineers for pressure analysis at NASA Langley, Va. A sumation of algorithms used in this package is covered in Chapter 2.

The remainder of this report discusses the second system. Chapter 3 covers the visualization of particles. Chapter 4 covers the visualization of a solid obstacle. Chapter 5 addresses extending the second system by discussing problems and difficulties that would arise from the alteration of our assumptions for visualization of density particles and an obstacle.

Chapter 2. Display of Pressure Distribution

2.1 Input data

The input to the system is a set of gridded data points. By gridded, we mean that the projection of these data points on an x-y plane is a set of intersection points of orthorgonal lines. See Fig. 2.1.1. Each data item is an ordered triple (x,y,z), where z is the pressure value measured at a point (x,y) on an airfoil. In this way, z can be viewed as the result of a single-valued function, f(x,y). If the original data are scattered, not gridded, a separate program has been written to compute an interpolated sampling with uniform spacing. This program uses a contouring subroutine JSSGRD of DI-3000, a software package by Precision Visuals Inc.. The subroutine is an implementation of an algorithm by Akima [1], which in turn uses the triangulation algorithm of Lawson [2]. The x-y plane is divided into triangular cells, with the vertices determined by projecting groups of three data points onto the plane. A bivariate fifth-degree polynomial in x and y is applied to interpolate points within a triangle. Estimated values of partial derivatives at each data point are used in determining the polynomial. Optional extrapolation may be performed to determine pressure values at points lying outside of the area covered by traiangles, but within the outermost grid area.

Akima's interpolating scheme, our first interpolating mechanism in this system, can be used independent of any other interpolating schemes which will be introduced in the next two sections. But it is not too sucessful in preserving the original shape of data. Therefore, we only apply this method to scattered data for computing an initial course sampling with uniform spacing.

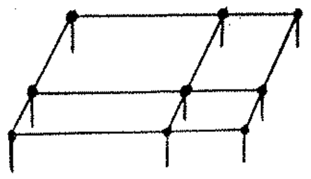


Fig. 2.1.1

2.2 McAllister's algorithm and Coon's bivariate interpolating scheme

McAllister's algorithm [3] [4] is an algorithm for calculating a Shape-Preserving Osculatory Quadratic Spline that preserves the monotonicity and convexity of the data when consistent with the given derivation at the data points. This algorithm is used here to compute the shape-preserving splines passing through each grid line in both the x and y direction, as shown in Fig 2.2.1.

In Fig. 2.2.2, the pressure value for point p* can be interpolated using Coon's Bivariate Interpolating Scheme if pressure values are known at points 1, 2, 3, and 4.

Our second interpolating mechanism is the combination of McAllister's algorithm and Coon's scheme. With this second method, the lattice in Fig. 2.2.2 can be refined into smaller uniformly spaced grids, as shown in Fig 2.2.3.

Without the third interpolating scheme which is to be introduced in the next section, this method alone can produce better results, but at too high a cost. Hence, we apply this second interpolating scheme only to compute an intermediate course sampling, and let the third interpolating scheme do the final refinement.

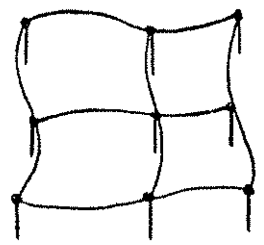


Fig. 2.2.1

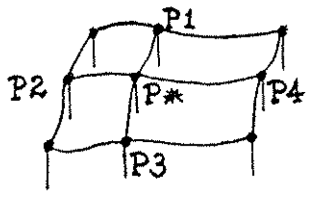


Fig. 2.2.2

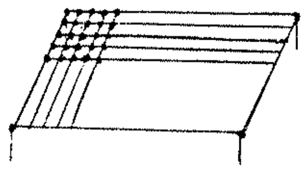


Fig. 2.2.3

2.3 Planer color contour

Upon completion of the second interpolating process as described in Sec. 2.2, if each unit area of the new grid is sufficiently small, it can be used as a display unit by converting its pressure measure to an intensity value. For each different pressure level, a distinctive pseudo-color can be assigned. For instance, red can be used to represent a high pressure level and blue to represent a low pressure level. If each new grid is not small enough to be a display unit, we further refine the course sampling. Since each grid should be relatively small at this point, it can be assumed that its surface is appropriately approximated by a quadric surface. With this underlying assumption, the following bilinear interpolating formula can be used to save computation time and storage.

$Z = C1X + C2Y + C3XY + C4$

The coefficients, C1, C2, C3, C4 are determined by the values of the four corner points, e.g., points 1, 2, 3, 4 in Fig. 2.3.1. With the coefficients known, the pressure value of any point on the surface can be interpolated.

This last interpolating method is very economical in comparison with the first two methods.

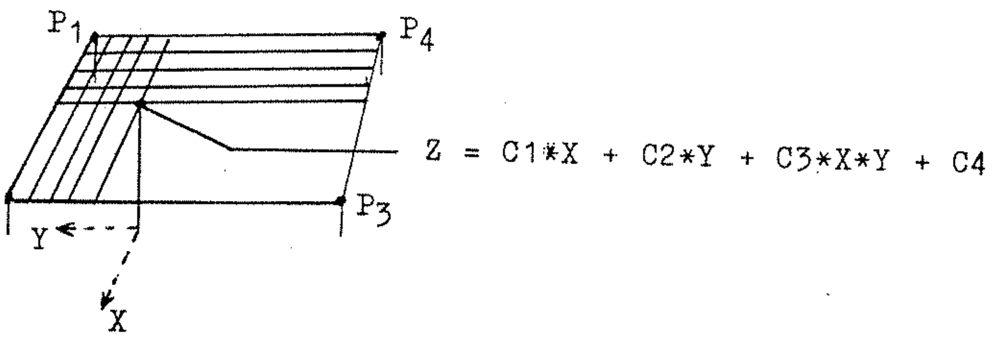


Fig. 2.3.1

With these interpolating schemes the program can now step through each image coordinate, (x,y), calculate the appropriate pressure value, z, and then display the color assigned to this pressure level. The resulting display appears as a planar contour map of the pressure values.

2.4 General surface mesh

An alternative approach is to create a mesh corresponding to the surface of the pressure function. For each grid line segment in the intermediate interpolation a line is drawn corresponding to the orthogonal projection of the grid segment from an arbitrary observation point. These mesh lines should preserve the shape of the original data. By rotating the observer's position, different views can be obtained.

Two mechanisms are used for line drawing, hardware line drawing, and software line drawing. Hardware line drawing is done through the TEKTRONIX 4115 command "DRAW LINE". Because the line drawings are done in hardware, the large data transmission cost is avoided. Therefore, it is relatively fast in comparison with the second line drawing method which is to be introduced next. But precision and flexibility are sacrificed for speed. One has little control over the color of each display unit.

The second mechanism is software line drawing. Lines are drawn point by point using the "RASTER WRITE" commands, for each point along the line, specifying the color. One has total control over the position and color of each point. Bresenham's algorithm [5] is applied here to plot a straight line between two points. Because of high transmission cost, speed is the price paid for flexibility here. On the average, to generate the same picture, using software line drawing mechanism takes four times as long as using hardware line drawing mechanism. Obviously, this proportion is closely related to the average length of lines being drawn.

Terry Jenssen's [6] algorithm for removal of hidden lines is used in our package. The basis of the algorithm is to compare each unique line to be plotted against potential planar surfaces which may hide all or a portion of the line. Each unit grid area is decomposed into a pair of triangles. Triangular area coordinates are then used to determine the intersection point of the line and the triangle. By utilizing area coordinates, no angle determination or trigonometric functions are required.

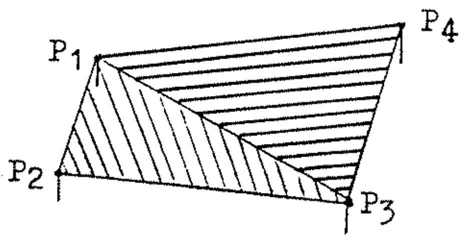


Fig. 2.4.1

While working with Jenssen's algorithm, we encountered a few problems which are not addressed in his paper. His algorithm takes four "corner" points and creates two triangles, as shown in Fig. 2.4.1, with one triangle having vertices P1, P2, P3, and another having vertices P1, P3, P4. His algorithm does not hide a line which lies along the diagonal line connecting P1 and P3. Also, a uniform error tolerance cannot be used throughout the algorithm. This error tolerence problem is the weakest part of Jenssen's algorithm and our program currently requires the user to set various error tolerance for their particular data sets.

Chapter 3. Visualization of Particles

Objects such as clouds, fog, flames, dust, and particles can be represented by densities within a volume unit. James Kajiya and Brian Von Herzen of California Institute of Technology developed light scattering equations [6] to solve the three-dimensional radiative scattering problem associated with objects in the above category. But an enormous amount of computation and resources are necessary in order to apply their new approximation method. As a much more economic alternative, we approximate each density volume by a sequence of unit planes to determine a reflection coefficient for the volume, and then apply geometric optics to the set of unit volumes.

3.1 Theoretical approach

Let A be a finite subset of Z^3, where Z is the set of integers. Given a set of particles contained in A, we can represent it by a density function $D: A \rightarrow R$, where R is the set of reals. Here, for any $(i,j,k) \in A$, $D(i,j,k)$ is the density of particles in the unit cube $C(i,j,k)$ centered at point (i,j,k).

In order to apply geometric optics, the particles in each unit cube are approximated by a sequence of parallel unit planes perpendicular to the incident light. These planes are identical and translucent. The number of planes, N, obtained for cube $C(i,j,k)$, depends linealy on density

D(i,j,k) with a approximation coefficient a:

$$N = N(i,j,k) = [D(i,j,k)/a] \qquad (3.1)$$

where [x] is the floor of x.

For simplicity, we assume that only a first order approximation to the reflected light intensity is considered. By first order approximation, we mean that any light ray which is reflected twice or more is neglected. For example, when the light ray reflected by P_2 reaches P_1, a portion of it is reflected back to P_2. This portion is of order 2 because its intensity is $O(\alpha^2)$ where α is the reflection coefficient of a single plane. If α is small enough, the amount of light bouncing back and forth among the planes can be considered neglectable. See Fig. 3.1.3.

Let I be the incident light ray, the intensity of the light reflected by cube C(i,j,k) is

$$I_r = I * \alpha * \sum_{i=0}^{N-1} (1 - \alpha)^{2i} \qquad (3.2)$$

where N = N(i,j,k) from Eq. (3.1).

For simplicity, we assume that the light transmitted through the cube is

$$I_t = I - I_r \qquad (3.3)$$

See Fig. 3.1.1.

Eq. (3.3) is derived from the assumption that no light is lost in the process, that is, all light is either reflected or transmitted. In reality, this assumption would not sustain since there is always light scattered in other directions. But the amount of scattered light for each unit plane can be viewed as a constant proportion of the incident light.

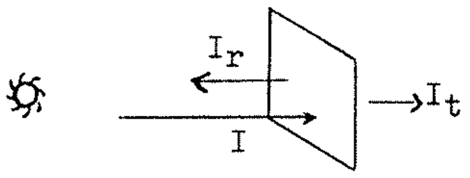


Fig. 3.1.1

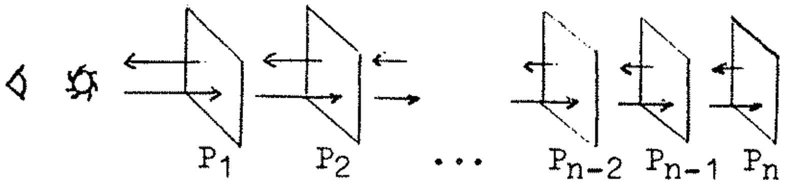


Fig. 3.1.2

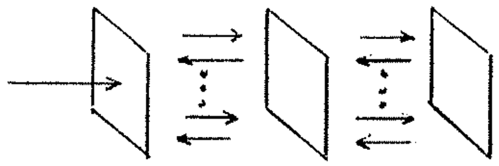


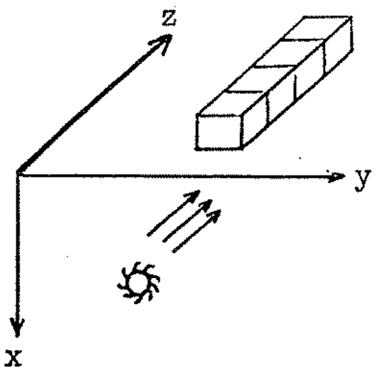
Fig. 3.1.3

In Fig. 3.1.2, the light ray reflected by the plane P. has to go through plane P_{n-1}, P_{n-2}, ..., P_1 before reaching the viewer. This is the reason why the exponent in Eq. (3.2) is 2N instead of N.

It is obvious that the reflection coefficient α depends on the substance of the particles as well as the approximation coefficient a, while the approximation coefficient a depends not only on the substance of the particles, but also on the accuracy desired, and of course, on the capacity of the computing system.

3.2 Implementation

In the actual implementation, we assume that we are working with parallel projection, the light source coincides with the viewpoint, and incident light rays are parallel to z axis. See Fig 3.2.1. Different intensities of grey scale are used to represent light reflected by various particle densities. The orginal light intensity is set to the brightest white color ([255, 255, 255] in Tektronix [red, green, blue] color coordinate system). Our input density data size is 150 X 150 X 150. These data are stored in a file such that D(i,j,k) precedes D(i1,j1,k1) if (k,j,i) is lexicographically less than (k1,j1,i1). The data are projected onto a plane of size 150 X 150. Each display unit on this plane contains 3 X 3 (9) pixels. Thus, we have a viewport of 450 X 450.



Left-handed Coordinate System

Fig. 3.2.1

Color at display unit [i,j] is determined by intensity of light reflected by cubes C[i,j,1], C[i,j,2], ..., and

C[i,j,150]. In the previous section, we described how reflected and transmitted light intensity for each volume unit can be calculated. The light intensity transmitted through C[i,j,1], I_{t1}, becomes the incident light intensity for C[i,j,2]. From I_{t1} and D[i,j,2], we then can determine I_{t2}, and I_{r2}, the reflected and transmitted light intensity for cube C[i,j,2], respectively. And I_{t2} becomes the incident light intensity for C[i,j,3]. This process is repeated throughout the 150 cubes. The final reflected light intensity is the sum of light intensity reflected by each of the 150 cubes,

$$I_r = I_{r1} + I_{r2} + \dots + I_{r150},$$

see Fig. 3.2.2. Tektronix command, "RUNLENGTH WRITE" is used to paint 50 display units at a time.

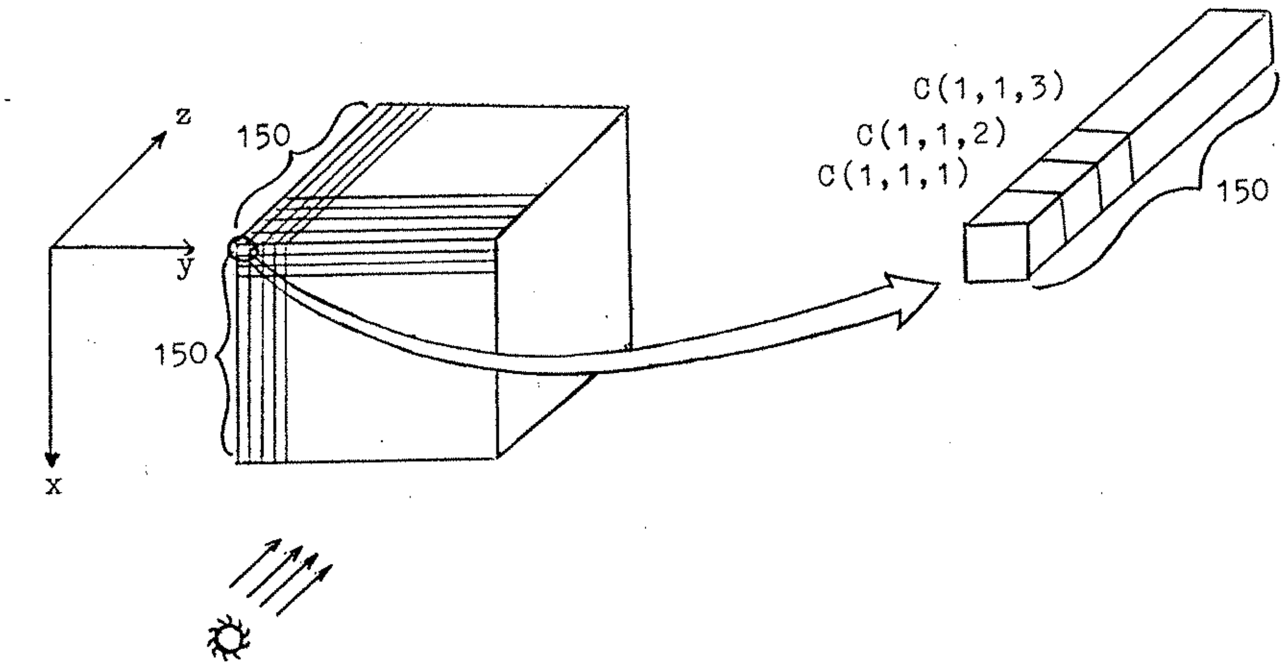


Fig. 3.2.2

The reflection coefficient, α, is set to 0.005. The approximation coefficient a is set to 1.0.

Since our data are "discrete", not "continuous", there exist aliasing problems. But overall, the result that we obtained from this approximation scheme is satisfactory.

Chapter 4. Visulization of a Solid Obstacle

The Torrance-Sparrow model [8,9] is a theoretically-based model of a reflecting surface. To determine the amount of light reflected at a certain point on the surface of a solid obstacle, we apply the Gaussian distribution function in Torrence-Sparrow's model. The surface of the obstacle is reconstructed from input data by constructing local facets on the surface.

Let [x,y,z] be an orthorgonal coordinate system where x, y, z are integers. z-axis is perpendicular to the screen and parallel to the incident light ray. The density of any unit volume C[i,j,k] contained in the obstacle is set to infinity, ∞. The visible parts of the obstacle surface can be characterized by a two-dimensional array S in the following way:

$$S = \{ S(i,j) \ / \ S(i,j) = k \text{ where } k \text{ is the minimum } k \text{ for which } D(i,j,k) = \infty; \text{ or } S(i,j) = \infty \text{ if no } D(i,j,k) = \infty \}$$

See FIg. 4.1.1.

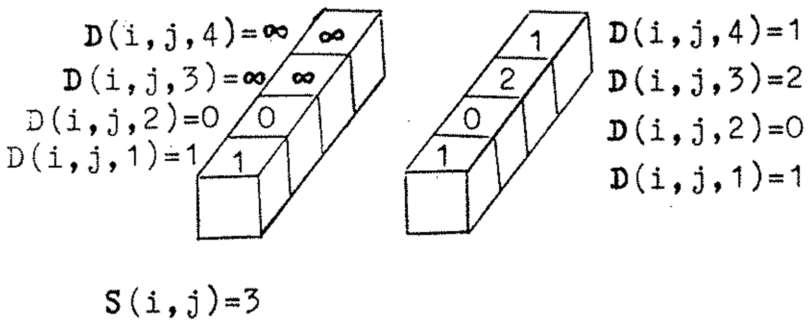


Fig. 4.1.1

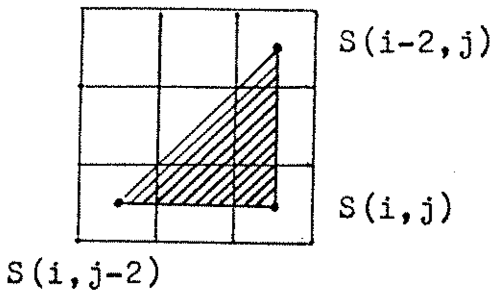


Fig. 4.1.2

The array, S, is then used to reconstruct the surface of the obstacle. By taking three S values, S[i-2,j], S[i,j-2], and S[i,j], as shown in Fig. 4.1.2, a local facet can be determined. If i<3, i.e., this cube is on the top edge of the viewport, S[i-2,j] = ∞; if j<3, i.e., this cube is on the left edge of the viewport, S[i,j-2] = ∞. By repeating the same process for every possible S[i,j], the surface of the obstacle can be reconstructed from these individual facets.

To reduce aliasing problem, we define T[m,n] as the average of S[m-1,n-1], S[m,n-1], S[m-1,n], and S[m,n]. In other words, instead of letting three points determine a surface, we let twelve points determine a surface. See Fig. 4.1.3. In Fig. 4.1.3, four X's determine the value for point 1, four O's determine the value for point 2, and four V's determine the value for point 3. Therefore, the surface is determined by values of twelve points.

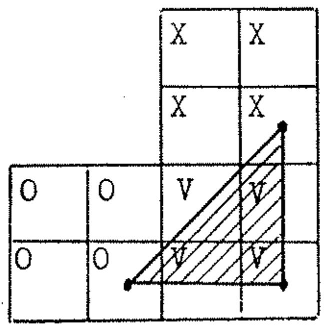


Fig. 4.1.3

Another advantage of this mechanism is that we only have to maintain a maximum of five rows of information (three rows of T and two rows of S) at any given instance.

For every facet, we need to determine its normal vector:

$$\vec{n} = \vec{a} \times \vec{b}$$

See Fig. 4.2.5.

The angle, β, between normal vector $\vec{n}$ and eye line vector $\vec{e}$ is :

$$\beta = \text{Arc Cos } ((\vec{n} \cdot \vec{e}) / (|\vec{n}||\vec{e}|))$$

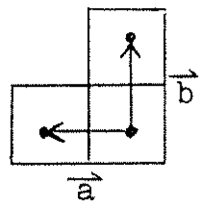


Fig. 4.2.5

Knowing angle β, we can now apply Torrance-Sparrow's method to calculate the amount of light reflected by this local facet. Torrance-Sparrow's method assumes that an imperfect macrosurface is composed of thousands of microscopic facets. Most of them are perpendicular to the normal direction of the macrosurface, some facing other directions. We assume that the facets on the obstacle surface, as computed above, are such imperfect macrosurfaces. The angle r between the normal direction of the macrosurface and the normal direction of each microscopic facet varies according to a Gaussian distribution. See Fig. 4.2.6.

In Fig. 4.2.6, area A with width w1, represents number of microscopic facets whose normal direction forms an angle β with $\vec{n}$. Since $\vec{n}$ is in a three-dimensional space, there are infinitely many vectors which point in different directions and form a angle β with n. w2 is a constant to cut off the invisible range, as shown in Fig. 4.2.7. Reflected light intensity is calculated by:

$$I_\gamma = w1 * w2 * \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{\beta^2}{2\sigma^2}} * I$$

where w1 and w2 are user-defined constants, determined by the reflectivity and smoothness of the surface.

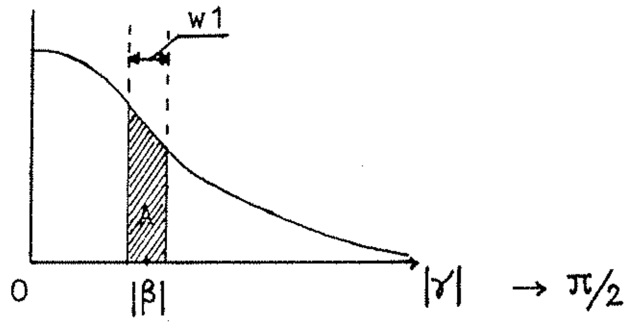


Fig. 4.2.6

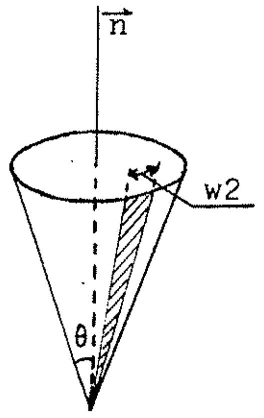


Fig. 4.2.7

With I_γ determined for a volume unit corrsponding to the obstacle surface our system can create an image for a wind-tunnel volume containing both particles and obstacles. The system processes the non-obstacle units as described in

Chapter 3, but when an obstacle unit is encountered the appropriate I_{γ} is added to the display intensity and processing is terminated for that (i,j).

Chapter 5. Discussion on Generalized System

In previous chapters, we made several assumptions for reasons of simplicity and developed methods for the simplified systems. Now, we will further explore possible ways to deal with more general and complicated systems. Problems arising because of increased data size, rotated viewpoint, and separation of light source and viewpoint are discussed. Possible solutions and alternatives are also included.

5.1 Increasing input data size

This is an easy task provided storage is not a problem. All underlying algorithms remain the same. We need to increase size of arrays, and do one or all of the following:

1) increase viewport
2) increase resolution
3) pack data (if resolution has reached maximum)
   Here, each new unit volume contains the average density of more than one cube.

Of course, with the input data size increased, one should expect processing time be proportionally increased.

5.2 Rotating viewpoint and light source

Recall that our input data are arranged in the order that D(i,j,k) is preceeding D(i1,j1,k1) if (k,j,i) is lexicographically less than (k1,j1,i1). See Fig, 5.2.1. This makes it possible for us to process one row at a time, as row 1 and row 2 in Fig. 5.2.1. But if we rotate the light source and viewpoint (i.e., light source and viewpoint no longer on z-axis) and still keep the same data file, we will not be able to take the same approach. There are two possible cases:

1) light source and viewpoint moving together

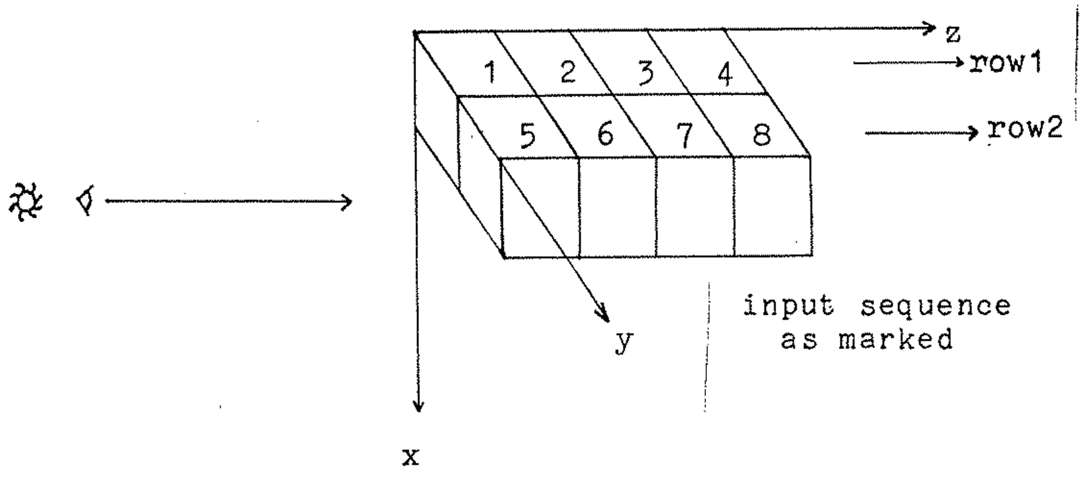


Fig. 5.2.1

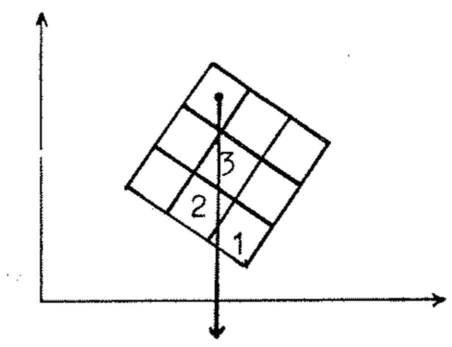


Fig. 5.2.2

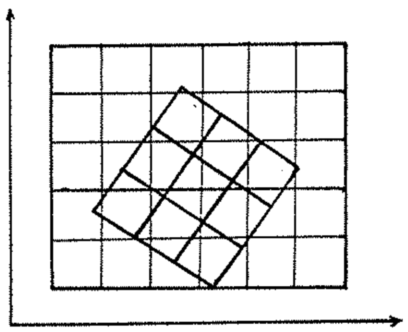


Fig. 5.2.3

A new cube array C2 and a corresponding density array D2 are allocated. Elements in D2 are initialized to zeroes. When a new density value D(i,j,k) is read in, we first perform the rotation on (i,j,k), obtain a new point (Oi,Oj,Ok), and D(i,j,k) is added to D2(Oi,Oj,Ok). We have addition here because two different points may be rotated to be within the same volume.

Note that after the rotaion, C2 is not necessarily on the lattice Z^3. See Fig. 5.2.2. There are several possible ways to solve this problem. For example, in Fig. 5.2.2, we can compute the intersection length of the incident light ray and cube 1, 2, and 3 separately; and with the density values for the three cubes known, we can thus determine the number of translucent unit planes between C(Oi,Oj,Ok) and viewpoint. A second method is to let C2 be on lattice Z^3, and for each rotated cube, find an approximated (Oi,Oj,Ok) to fit in. See Fig. 5.2.3. The approximation mechanism may vary. The advantage of this method is that it is much simpler. But we pay the price on precision.

2) separating light source and viewpoint

For obstacles, our algorithm can be applied without any significant modification. But for particle volumes, we would need a three-dimensional array to keep all the input data in storage. For each cube C(i,j,k), we have to determine the intensity of light ray reaching the cube from the light source using our method described in Sec. 3.1. And by applying Kajiya and Von Herzen's light scattering equation [6] at C(i,j,k), we can compute the intensity of light scattered to direction of viewpoint. Knowing this light intensity, the density of volume units between C(i,j,k) and viewpoint, and the intersection length of light ray and cubes, the intensity of light reflected to viewpoint can be calculated, again, using our method described in Sec. 3.1. See Fig. 5.2.4.

Note that cubes along $\vec{l}$ are approximated into planes perpendicular to $\vec{l}$, and cubics along e are approximated into planes perpendicular to $\vec{e}$. Geometric optics are applied to all cubes but C(i,j,k) where we apply Kajiya and Von Herzen's light scattering equations instead.

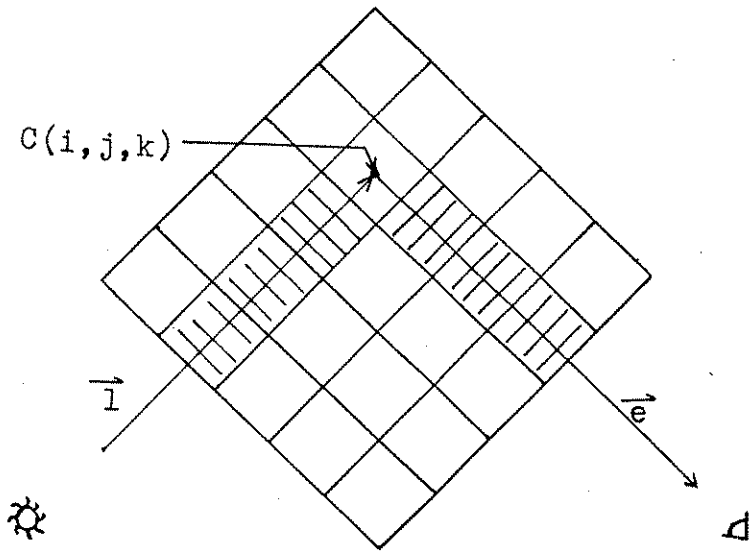


Fig. 5.2.4

5.3 Particle flow throughout a sequence of time intervals

To represent flow of particles, we need only to identify positions of particles at a sequence of time, t1, t2, ...,tn. The simulation or actual tracing of particle flow is done in a separate system which creates the sequence of frames. Each frame contains a three-dimensional volume of density data points. Our program can read these data in through a secondary device and graphically represent the change of states from one time to another. Note that the sequence of frames would take up an enormous amount of storage. For instance, with a frame of 1000 X 1000 X 1000 unit volumes, we need approximately 2000 megabytes to store a single frame. With a movie which displays 16 frames per second, we need a total of 1,920,000 megabytes for only one minute of movie.

Chapter 6. <u>Conclusion</u>

The two systems discussed in this paper are just two examples among thousands of computer graphics systems being developed for engineering applications. One of the major goals of computer graphics is aimed at making an engineer's job easier. The analysis and interpretation effort involved in experiments and simulations can be drastically reduced with the aid of computer graphics, especiallly when a vast amount of data is involved. The two systems we have developed aid in the visualization of wind-tunnel phenomena which were difficult or impossible to visualize in the past. Aircraft and automobile design engineering are two examples of areas that benefit greatly from the use of computer graphics technology. We can expect to see this trend continue as the use of computer graphics as a helpful tool grown in the future. In particular, the second system discribed in this paper is just the first phase in the development of an extensive graphics system for the visualization of airflow in wind-tunnel simulations.

REFERENCES

[1] Akima, Hirosha. "A method of Bivariate Interpolation and Smooth Surface Fitting for Irregularly Distributed Data Points." ACM Transactions on Mathematical Software, Vol. 4, No 2, pp. 148-159, June 1978.

[2] Lawson, C. L. "Software for C1 Surface Interpolation." JPL Publication 77-30, 15 August 1977.

[3] McAllister, D. F., Dodd, S. L., Roulier J. A. "Shape-Preserving Spline Interpolation for Specifying Bivariate Functions on Grids" IEEE Computer Graphics and Applications, September 1983.

[4] McAllister David F., Roulier John A. "An algorithm for Computing a Shape-Preserving Osculatory Quadratic Spline" ACM Transactions on Mathematical Software, Vol 7, No 3, September 1981, Pages 331-347.

[5] Foley James D., Van Dam Andries, Fundamentals of Interactive Computer Graphics, Addison-Wesley, Reading, Massachusetts, 1983.

[6] Janssen Terry L. "A Simple Efficient Hidden Line Algorithm", Computers and Structures, Vol. 17, No 4, pp. 563-571, 1983.

[7] Kajiya, James T., Von Herzen, Brian P. "Ray Tracing Volume Densities", ACM Computer Graphics, Vol 18, No 3, pp 165-175, July 1984.

[8] Torrance, K. E., Sparrow, E. M., "Polarization, Direction Distribution, and Off-Specular Peak Phenomena in Light Reflected from Roughened Surfaces", J. Opt. Soc. Am., 56(7), July 1966, pp. 916-925.

[9] Torrance, K. E., Sparrow, E. M. "Theory for Off-Specular Reflection from Roughened Surfaces", J. Opt. Soc. Am., 57(9), Sept. 1967, pp. 1105-1114.