10/31/95

SANDIA REPORT

SAND95–2049 • UC–700
Unlimited Release
Printed August 1995

CHAPARRAL: A Library for Solving Large Enclosure Radiation Heat Transfer Problems

Micheal W. Glass

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550
for the United States Department of Energy
under Contract DE-AC04-94AL85000

Distribution
UC-700

SAND95-2049
Unlimited Release
Printed August 1995

CHAPARRAL: A Library for Solving Large Enclosure Radiation Heat Transfer Problems

Micheal W. Glass

Manufacturing and Environmental Fluid Dynamics

Sandia National Laboratories

Albuquerque, NM 87185

Abstract

Large, three-dimensional enclosure radiation heat transfer problems place a heavy demand on computing resources such as computational cycles, memory requirements, disk I/O, and disk space usage. This is primarily due to the computational and memory requirements associated with the view factor calculation and subsequent access of the view factor matrix during solution of the radiosity matrix equation. This is a fundamental problem that constrains Sandia's current modeling capabilities. Reducing the computational and memory requirements for calculating and manipulating view factors would enable an analyst to increase the level of detail at which a body could be modeled and would have a major impact on many programs at Sandia such as weapon and transportation safety programs, component survivability programs, energy programs, and material processing programs. CHAPARRAL is a library package written to address these problems and is specifically tailored towards the efficient solution of extremely large three-dimensional enclosure radiation heat transfer problems.

[illegible]

Table of Contents

List of Figures

List of Tables

1. Introduction

Use of the global net radiation method [1] to solve enclosure radiation problems involves the calculation of surface-to-surface view factor pairs. In a standard finite element analysis, the common approach is to use the exposed side of elements as radiating surfaces. To adequately resolve temperature gradients during a finite element thermal analysis, a large number of elements are required in the body being modeled which in turn increases the number of radiating surfaces. Because the number of view factors between the radiating surfaces increases with $O(N^2)$, the calculation of the view factors can easily become a significant, if not dominant, portion of the computation for realistic problems. This is particularly true for steady-state problems or transient problems with a changing geometry or mesh. In many cases, there are problems that we cannot currently solve without sacrificing accuracy (i.e. a less detailed mesh) for reduced computational time.

Depending upon the implementation strategy, efficient solution of large radiation enclosure heat transfer problems typically involves balancing CPU, core memory, and disk I/O requirements for a particular problem. Because there is no set formula for determining how these requirements should be allocated, the experience and intuition of the analyst must be relied upon. CHAPARRAL was written to reduce some of these requirements and allow as much flexibility as possible.

Traditional view factor algorithms from the heat transfer field have proven to be inefficient for very large enclosure radiation problems, particularly when obstructing surfaces are present. However, in the field of computer graphics, new algorithms have been developed for calculating view factors and solving the radiosity problem for a radiosity based global illumination model. One such method for calculating view factors, the hemicube method, has been widely used in the computer graphics field with great success. The hemicube method has been implemented within CHAPARRAL along with more traditional methods for two-dimensional view factor calculations. The hemicube method has demonstrated a tremendous speedup over the traditional double area integration method for three-dimensional geometries. For solution of the radiosity matrix equation, CHAPARRAL includes both a Gauss-Siedel iterative method and a progressive refinement method. For some problems, the progressive refinement method can significantly reduce the number of times the view factors must be accessed, thus speeding up the solution of the matrix equation. Due to the large memory requirements for storage of the view factor matrix, an internal view factor database with compression/decompression algorithms has also been included to reduce the memory requirements and/or disk I/O for storing and accessing the view factors.

2. Problem Overview

2.1 Enclosure Radiation Heat Transfer and the Radiosity Model

The analysis of thermal radiation heat transfer within an enclosure is complicated because the energy transfer mechanism introduces nonlinearities into the problem and requires the specification of radiation view factors. The energy flux leaving a given surface is composed of directly emitted and reflected energy[1]. The reflected energy flux is dependent upon the incident energy flux from the surroundings, which can be expressed in terms of the energy flux leaving all other surfaces. The energy reflected from surface k is given by

$$q_{ok} = \varepsilon_k \sigma T_k^4 + \rho_k q_{ik} \tag{1}$$

where q_{ok} is the energy flux leaving the surface, ε_k is the emissivity, σ is Boltzmann's constant, T_k is the temperature, ρ_k is the reflectivity, and q_{ik} is the energy flux incident upon the surface from its surroundings. The amount of incident energy upon a surface from another surface is a direct function of the surface-to-surface view factor, F_{jk}. The view factor, F_{jk}, can be thought of as the fraction of energy leaving surface k which is incident upon surface j. The incident energy flux, q_{ik}, can be expressed in terms of the energy flux leaving all other surfaces as

$$A_k q_{ik} = \sum_{j=1}^{N} A_j q_{oj} F_{jk} \tag{2}$$

where F_{jk} is known as the view factor between surface k and surface j (also called form factor, configuration factor). The view factor F_{jk} represents the fraction of energy leaving surface j which is directly incident upon surface k. For N surfaces, using the view factor reciprocity relationship gives

$$A_j F_{jk} = A_k F_{kj} \qquad \text{for } j = 1, 2, 3, \ldots\ldots, N \tag{3}$$

so that

$$q_{ik} = \sum_{j=1}^{N} F_{kj} q_{oj} \tag{4}$$

therefore

$$q_{ok} = \varepsilon_k \sigma T_k^4 + \rho_k \sum_{j=1}^{N} F_{kj} q_{oj} \tag{5}$$

Equation (5) can be rewritten as

$$B_k = E_k + \rho_k \sum_{j=1}^{N} F_{kj} B_j \tag{6}$$

where B_k represents the radiosity of surface k and E_k represents the emissivity of surface k. This represents N equations which can be recast into matrix form as

$$\begin{bmatrix} 1-\rho_1 F_{11} & -\rho_1 F_{12} & \bullet & \bullet & \bullet & -\rho_1 F_{1N} \\ -\rho_2 F_{21} & 1-\rho_2 F_{22} & \bullet & \bullet & \bullet & -\rho_2 F_{2N} \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ -\rho_N F_{N1} & -\rho_N F_{N2} & \bullet & \bullet & \bullet & 1-\rho_N F_{NN} \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ \bullet \\ \bullet \\ B_N \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \\ \bullet \\ \bullet \\ E_N \end{bmatrix} \tag{7}$$

or

$$\mathbf{KB} = \mathbf{E} \tag{8}$$

Equation (8) is referred to as the radiosity model or radiosity matrix equation. The view factor between two finite surfaces i and j is given by

$$F_{ij} = \frac{1}{A_i} \int_{A_i} \int_{A_j} \frac{\cos\theta_i \cos\theta_j}{\pi r^2} \delta_{ij} \, dA_j \, dA_i \tag{9}$$

where δ_{ij} is determined by the visibility of dA_j to dA_i. δ_{ij} is equal to one if dA_j is visible to dA_i and is equal to zero otherwise. See Figure 1 for the geometric interpretation of the individual terms.

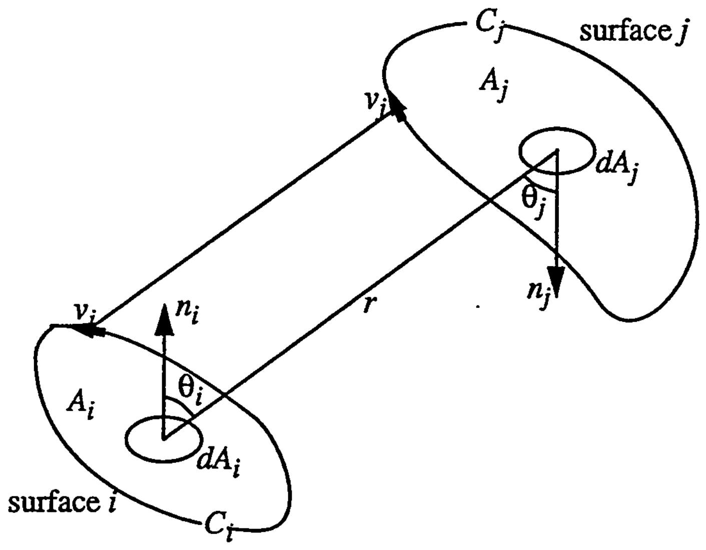


Figure 1 Geometry and nomenclature for calculating view factors between finite areas.

As mentioned earlier, enclosure radiation heat transfer problems usually require computer resources of the following nature: (1) CPU processing to calculate the view factors and solve the radiosity matrix equation; (2) core memory for storage of the view factors; and (3) disk storage and I/O for out-of-core storage of the view factors and accessing the view factors when solving the radiosity matrix equation. Efficiently solving problems of this nature requires balancing these three needs to achieve optimum performance. CHAPARRAL addresses these three areas and provide some improvement in performance. CHAPARRAL consists of three main sections: (1) calculation of the geometric view factors; (2) an internal database for storing and accessing the view factor matrix; and (3) solution of the radiosity matrix equation. In this chapter, the implementation of each of these areas will be discussed in greater detail.

2.2 Calculation of View Factors

Emery et. al. [2] provide an excellent review of the various methods commonly used to calculate view factors. Each has their advantages and disadvantages as detailed in the paper. Therefore, for greater flexibility, multiple algorithms are available within CHAPARRAL for view factor computations. Traditional methods were made available by modifying the FACET [3] package to be a subroutine library and incorporating it into CHAPARRAL. FACET uses traditional methods such as line integration and double area integration and handles 2D planar, 2D axisymmetric, and 3D geometries. An implementation of the hemicube algorithm is available for 3D geometries. By default, FACET is used to calculate the view factors for 2D geometries and the hemicube method is used for 3D geometries. Because the hemicube algorithm is less well known in the heat transfer field as a method to calculate view factors, more detail on this method is provided next.

2.2.1 The Hemi-Cube Method

When Goral et. al. [4] first introduced the radiosity method to the computer graphics field, the view factors were calculated using contour line integration; however, even when applied to simple scenes, this was considered a computationally expensive method. For complex geometries with obstructed views, the double area integration scheme had to be used and the time to calculate the view factors became exorbitantly high. With the strive towards real-time image generation, the first significant contribution from the computer graphics field was in the area of view factor calculation. Specifically, the hemicube method for view factor calculation was developed to decrease the computational effort required for such calculations.

The hemicube method is based upon Nusselt's hemisphere analogy and was first proposed by Cohen et. al. in 1985 [5]. Instead of projecting the surfaces onto a hemisphere, a hemicube is used. Nusselt's analogy shows that any surface which covers the same area on the hemisphere has the same view factor. From this it is evident that any intermediate surface geometry can be used without changing the value of the view factors (see Figure 2). Hence the hemisphere can be replaced with a hemicube.

The use of a hemicube allows for the development of very efficient algorithms for calculating

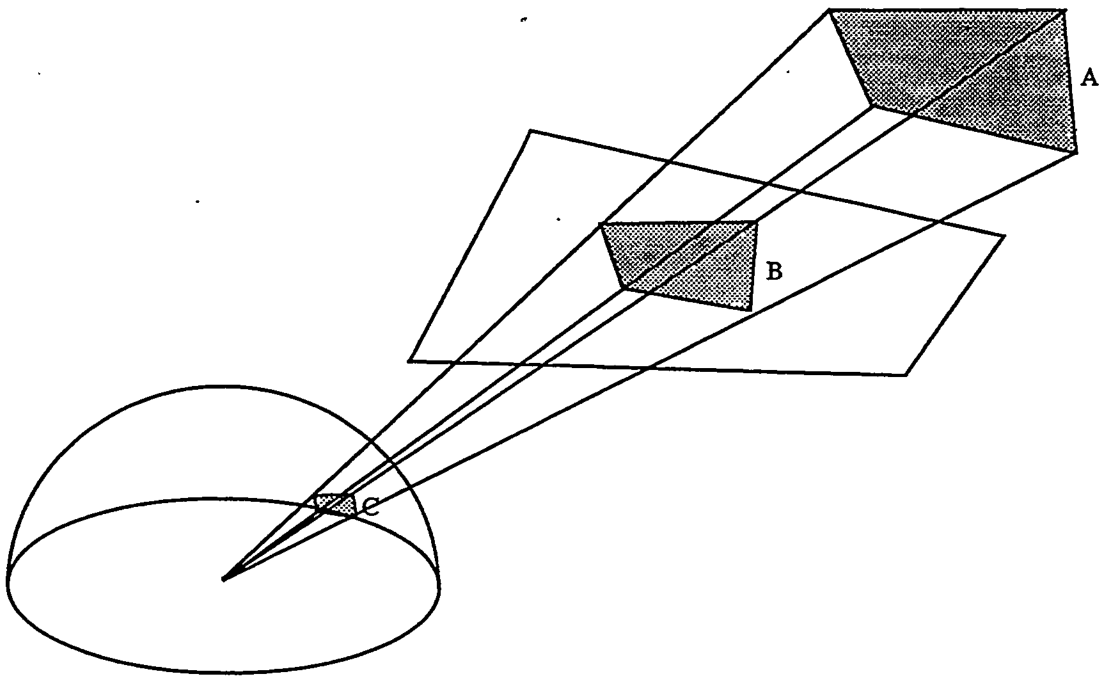


Figure 2 Nusselt's hemishere method.

view factors in a complex environment. The basic algorithm begins by discretizing the surface of the hemicube into a set of N uniform subpatches which will be called pixels. Each pixel defines a particular direction and angle from the receiving patch's centroid. Thus, each pixel contributes a specific delta-view factor value to the overall view factor between two surfaces if the pixel is covered by the projection of the transmitting surface onto the discretized hemicube. See Figure 3 for the geometric interpretation of this method.

The delta-view factors for each pixel, $n=1,2,...,N$ on the hemicube are found from

$$\Delta\text{viewfactor} = \Delta F_n = \frac{\cos\theta_i \cos\theta_j}{\pi r^2}\Delta A_j \tag{10}$$

See Figure 4 for the geometric interpretation of the terms used in Equation (10). The overall view factor is then calculated by

$$F_{ij} = \sum_{n=1}^{N} \Delta F_n \tag{11}$$

For efficiency, the delta-view factor for each pixel is calculated only once and stored in a lookup table for later use. This lookup table actually only contains values for 1/4 of the top face and

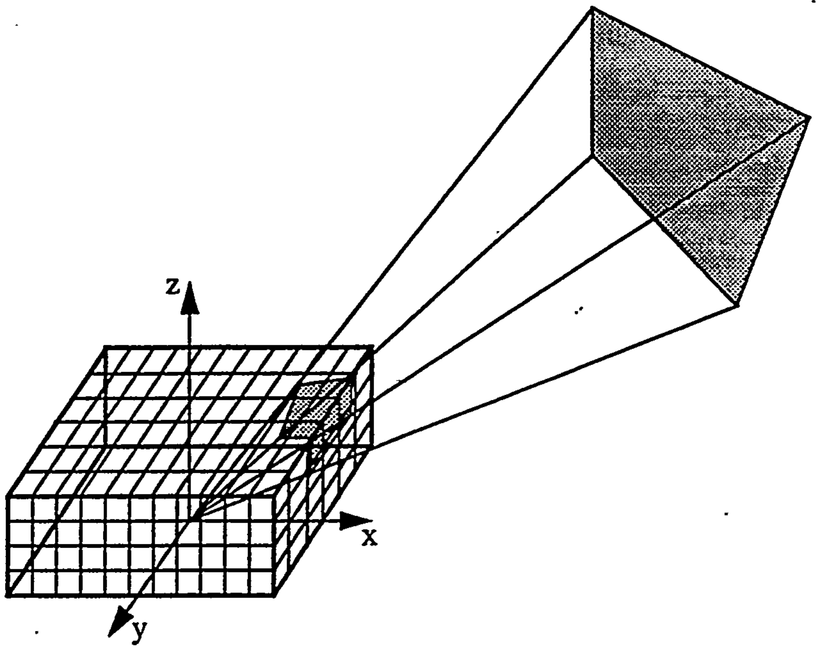
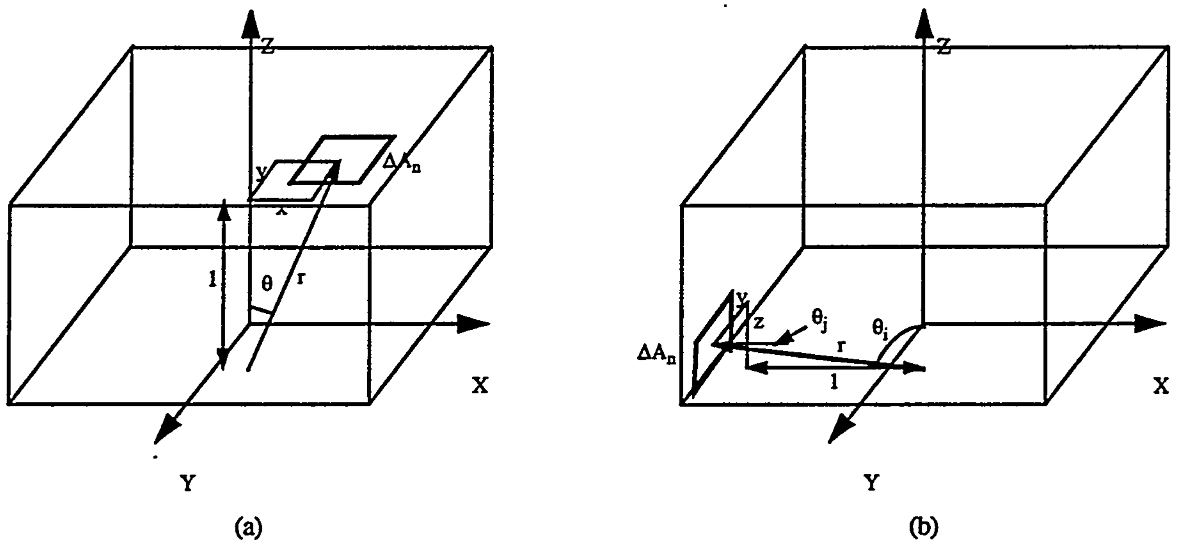


Figure 3 Discretization of the hemicube.



$$r^2 = x^2 + y^2 + 1$$

$$\cos\theta = \cos\theta_i = \cos\theta_j = \frac{1}{r}$$

$$\Delta F_n = \frac{1}{\pi r^2}\Delta A_n$$

$$r^2 = 1 + y^2 + z^2$$

$$\cos\theta_i = \frac{z}{r} \qquad \cos\theta_j = \frac{1}{r}$$

$$\Delta F_n = \frac{z}{\pi r^2}\Delta A_n$$

Figure 4 Derivation of delta-view factors for (a) pixel on top of hemicube and (b) pixel on the side of the hemicube.

1/2 of each side face due to symmetry.

The top face of the hemicube represents a 90° viewing frustum and each side face represents one half of a 90° viewing frustum which is well known in the field of computer graphics. Hence, all the developments for projecting environments within the frustum can be taken advantage of for computational efficiency.

One can see that instead of calculating view factors on a pair by pair basis, the hemicube algorithm allows for an entire row of view factors F_{ij}: $j = 1, 2, 3, \ldots, N$ to be calculated at the same time after projecting the environment onto the hemicube for surface i. Pseudo-code for the hemicube algorithm is given below

```
Loop on the number of enclosures, n
    Initialize the hemicube delta view factors
    Loop on the number of surfaces, i
        Calculate surface areas, centroids, and normals
    End loop
    Loop on the number of surfaces, i
        Initialize view factor row $F_{ij} = 0$ for all j
        Initialize hemicube ID buffer to NULL surface ID
        Initialize hemicube zbuffer to maximum real number
        Place hemicube at surface centroid
        Loop on number of hemicube sides (including top)
            Align view direction with top or side
            Loop on number of surfaces, j
                Project surface j onto the hemicube
                Scan convert and zbuffer surface j projection onto hemicube sides
            End loop
            Sum contribution to $F_{ij}$, $F_{ij} = F_{ij} + \sum \Delta F$ of grid cells with zbuffered ID = j
        End loop
    End loop
End loop
```

2.2.2 Errors Associated with the Hemi-Cube Method

The hemicube algorithm is based upon various assumptions about the environmental geometry which if violated, will produce inaccurate values for the view factors [6]. The three major assumptions of the hemicube algorithm are: (1) proximity - the distance between surfaces is great compared to the effective diameter of the surfaces; (2) visibility - the visibility between any two surfaces does not change; and, (3) aliasing - the true projection of each visible surface onto the hemicube can be accurately accounted for by using a finite resolution hemicube.

The proximity assumption is violated whenever surfaces are close compared to their effective diameters or are adjacent to one another. In such cases the distance between the centroid of one surface to all points on the other surface can vary greatly. Because the view factor dependence on distance is nonlinear, the result is a poor estimate of the view factor. The most common violation of this assumption occurs when surfaces are adjoint. Baum, et. al. [6] used a hybrid scheme which combined the hemicube method with an analytical method when the proximity assumption was violated.

The visibility assumption requires that the term δ_{ij} in Equation (9) remain constant across surface i. Because δ_{ij} is a discontinuous function, the single point evaluation at the centroid of surface i can introduce significant errors to the value of the view factor. This is depicted in Figure 5 where surface 1 has a complete view of surface 2 from its centroid but in fact surface 3 occludes much of surface 2 from surface 1, In such a case the hemicube algorithm will overestimate F_{ij} by using F_{dij} calculated from the centroid of surface 1. To reduce this error, surface 1 could be subdivided into smaller subelements with the hemicube method applied to each subelement and the results combined to produce a more accurate value for the view factor.

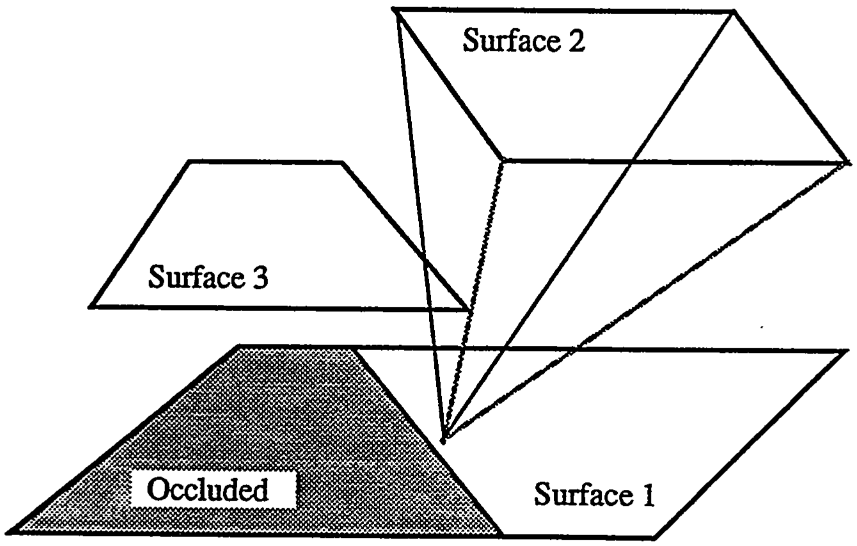


Figure 5 Geometry where the visibility assumption is violated

The aliasing assumption, that surfaces project exactly onto a whole number of hemicube pixels, is similar to aliasing problems associated with graphical image displays. Because of the finite resolution of the hemicube, the projected areas and resultant view factors may be over or under estimated as shown in Figure 6. Aliasing effects can be reduced by increasing the resolution of the hemicube, by using multiple hemicube subsamples and filtering the results, or by using jittered hemicubes.

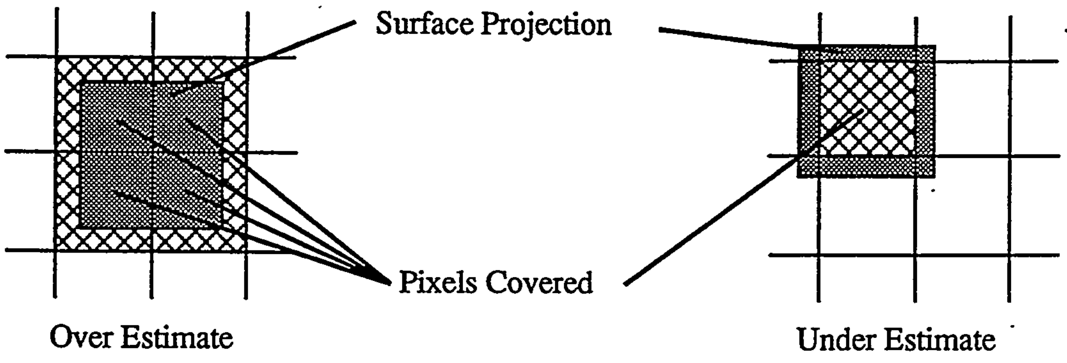


Figure 6 Example of how aliasing can over or under estimate the projected surface area.

In this initial release of CHAPARRAL, only hemicube jittering has been implemented. With jittered hemicubes, each hemicube is randomly rotated about its parent surface's normal vector. This reduces any preferential alignment of surfaces edges along the hemicube's discretized boundaries.

2.3 Internal Data Base Storage of View Factors

When solving radiation heat transfer problems, memory usage can be a major area of concern. The view factor matrix can be either stored in core memory or in a disk file. Depending upon the problem, the view factor matrix may either be sparse or full. If the matrix is stored in full form, one can quickly run into limitations - either core memory limitations or available disk space. For a problem with 5000 participating surfaces, 25 megawords of memory is required to store the entire view factor matrix! From experience, disk I/O (particularly on the Cray or over NFS) is always a bottleneck and should be avoided whenever possible. In CHAPARRAL, disk I/O has been eliminated from the resource balancing equation by allowing the view factors to always be stored in core memory. Though storing the view factors in core memory eliminates the disk I/O bottleneck, one can quickly run into core memory limitations. On Sandia's Cray/YMP, processes are limited to 32 megawords of core memory usage so this limit can be easily exceeded on problems of more than 5000 participating surfaces. On other systems without this memory limit, the memory requirements can still quickly exceed the amount of physical memory which can lead to excessive swapping as virtual memory is accessed. For this reason, when the view factor matrix is stored in memory, CHAPARRAL uses data compression to reduce the required amount of memory. The view factor matrix is stored row-by-row and each row is compressed separately. This requires CPU overhead to perform the data decompression each time the view factors are accessed, which adds a new dimension to balancing CPU and memory requirements for a job, but the memory savings can be substantial.

To implement the data compression, an internal database is used to store the view factor matrix and associated parameters. Access to the database is accomplished through a defined Application Programmer's Interface (API). This allows the internal workings of the database to be transparent to the user and supplies a well defined interface for the data compression/decompression routines. The API is described in further detail in Chapter 3.

Several data compression/decompression schemes are available. These include: (1) no compression; (2) byte runlength encoding (BRLE); (3) word runlength encoding (WRLE); and (4) Lempil-Zev-Walsh encoding (LZW). WRLE compression/decompression is available because some computer architectures access words much than faster bytes. Runlength encoding is probably the lowest level compression scheme available. The user's choice of a data compression schemes will depend upon speed/memory requirements of the particular problem.

2.4 Solution of the Radiosity Matrix Equation

Solving the radiosity matrix equation requires solving a system of linear equations which, depending upon the geometry of the model, may be dense or sparse. Though there are many methods available for this type of problem, some are better suited than others. Direct methods are not very well suited because of the potentially large size of the problem, thus iterative methods have been used almost exclusively. Of the iterative methods available, Gauss-Seidel has probably been the most widely used; and progressive refinement, one of the newer methods, can be advantageous in certain situations. Both of these methods are available in CHAPARRAL and are briefly described below.

2.4.1 Gauss-Seidel Approach

Gauss-Seidel is a variation of the Jacobi method and provides a true relaxation method which usually improves the speed at which the method converges to a solution. This iterative approach converges to a solution by solving the system of equations one row at a time. Physically, this means that the evaluation of the *ith* row of equations provides an estimate of the radiosity of surface *i* based on the current radiosities of all the other surfaces. Hence, the energy leaving surface *i* is determined by gathering in the energy from the rest of the environment. As such, the solution to the radiosity problem is calculated one surface at a time. Pseudo-code for the algorithm is given below.

```
Loop on the number of enclosures
    Loop on the number of surfaces
        Set $B_i$ = starting guess
        End loop
    While not converged
        Loop on number of surfaces, i
            $B_i = E_i - \sum_{j=1, j \neq i}^{n} B_j \frac{K_{ij}}{K_{ii}}$
        End loop
    End while
End loop
```

2.4.2 Progressive Refinement Approach

As an illumination model, the overwhelming cost of the radiosity method is the computation of the view factors. As with thermal radiation codes, this cost is reduced by calculating the view factors once at the beginning of a calculation and storing them for repeated use in the iteration cycle. In such an environment, the storage cost for the view factors are $O(N^2)$. In 1988, Cohen et. al. [7] proposed a progressive refinement method for the radiosity illumination model which computed the view factors on the fly, thus reducing the storage costs to $O(N)$.

For the progressive refinement approach, the problem is formulated so that the contribution made by surface *i* to all the other surfaces is calculated. This represents an iterative column solution to the set of radiosity equations. Physically, this means that the energy leaving surface *i* is shot out into the environment and its effect on all the other surfaces is calculated. Because the hemicube method and basic ray-tracing methods compute the view factors on a row-by-row basis, the reciprocity relationship (Equation 3) must be used to transform a row of view factors into a column of view factors. To converge gracefully and as quickly as possible, the surfaces are presorted according to the energy that each surface has to shoot. Theoretically, the solution is not converged until all the emitted and reflected energy is dispersed throughout the environment. But, the final solution may be approached to within a certain error tolerance very quickly and with only a fraction of the view factors calculated as compared to the full matrix formulation. As an example, Cohen et. al. [7] computed the global illumination of a steel mill scene composed of 30000 surfaces subdivided into 50000 patches. The full matrix formulation would have required the computation of 1.5×10^9 view factors requiring 6 Gbytes of storage space. The progressive refinement method needed only one column of view factors requiring 0.12 Mbytes of storage space. Additionally, the solution converged to an acceptable value after the energy from 2000 surfaces had been shot or less than 5% of the total number of view factors had been calculated!

While the progressive refinement approach to solving the radiosity problem has demonstrated tremendous speedups for image synthesis, its usefulness to heat transfer calculations may not have as big an impact. While the method converges rapidly to an acceptable result for image synthesis, are the error tolerances as tight as they would probably be in an engineering calculation or is the converged solution merely a visually acceptable result? Another reason this approach works so well as an illumination model is the fact that in image synthesis there are usually light sources included which have a much larger amount of energy to shoot than the other surfaces. By shooting the energy from the light sources first, and then from the major surfaces that are impacted with that energy, a very reasonable image can be obtained in a short period of time. But the presence of only a few highly emissive surfaces may not always be the case for heat transfer problems. One has only to look at a metal casting problem as an example where the vast majority of surfaces have emissive powers of the same order. And for transient problems, a breakpoint will exist where it is cheaper to calculate all the view factors first rather than repeatedly calculating a smaller number of view factors for each time step. On the other hand, this approach may prove to be very useful when solving transient problems where the geometry changes with time and hence the view factors will also change and cannot be precomputed for the duration of the calculation. Pseudocode for this method is given below.

```
Loop on the number of enclosures
    Loop on the number of surfaces, i
        set $B_i = E_i$
        set $\Delta B_i = E_i$
    End loop
    While not converged
        Choose surface i, such that $\Delta B_i * A_i$ is largest
        Loop on the number of surfaces, j
            Set $\Delta rad = \Delta B_i * \rho_j F_{ji}$
            Set $\Delta B_j = \Delta B_j + \Delta rad$
            Set $B_j = B_j + \Delta rad$
        End loop
        $\Delta B_i = 0$
    End while
End loop
```

2.5 Performance

To demonstrate the various aspects of CHAPARRAL, a finite element model of a SRAM fire-set casting was used as an example model. This model [8] consists of 3297 elements with 4538 radiating surfaces in 2 enclosures (3495 and 1043 surfaces in the top and bottom enclosures respectively). This model is shown in Figure 7. The three main parts of CHAPARRAL will now be discussed in further detail. All timing values are for a Sun Sparcstation 10/30 workstation.

For flexibility, multiple algorithms are available for view factor computations. The available algorithms include those available in FACET along with an implementation of the hemicube algorithm. By default, FACET is used to calculate the view factors for 2D geometries and the hemicube method is used for 3D geometries. The advantage of using the hemicube method for 3D geometries is illustrated by Table 1. Comparison of the various data compression schemes is shown in Table 2. From Table 2 it can be seen that for this particular test problem, there is no performance penalty for using WRLE compression. Comparison of the two methods for solving the radiosity matrix equation is shown in Table 3.

Table 1: Comparison of FACET with the Hemicube Method

Case Number	**Method Used to Calculate View Factors**	**Time (min) to Calculate View Factors for:**	
		Enclosure 1	**Enclosure 2**
1	Hemicube	19.4	1.8
2	FACET	15.1	2.8
3	FACET	204.1	51.4
4	FACET	292.6	12.3

Case 1: 50x50 hemicube resolution.

Case 2: no blocking surfaces specified, 2 subdivisions per edge.

Case 3: no blocking surfaces specified, auto subdividing on edges.

Case 4: all surfaces specified as potentially blocking, no edge subdivisions.

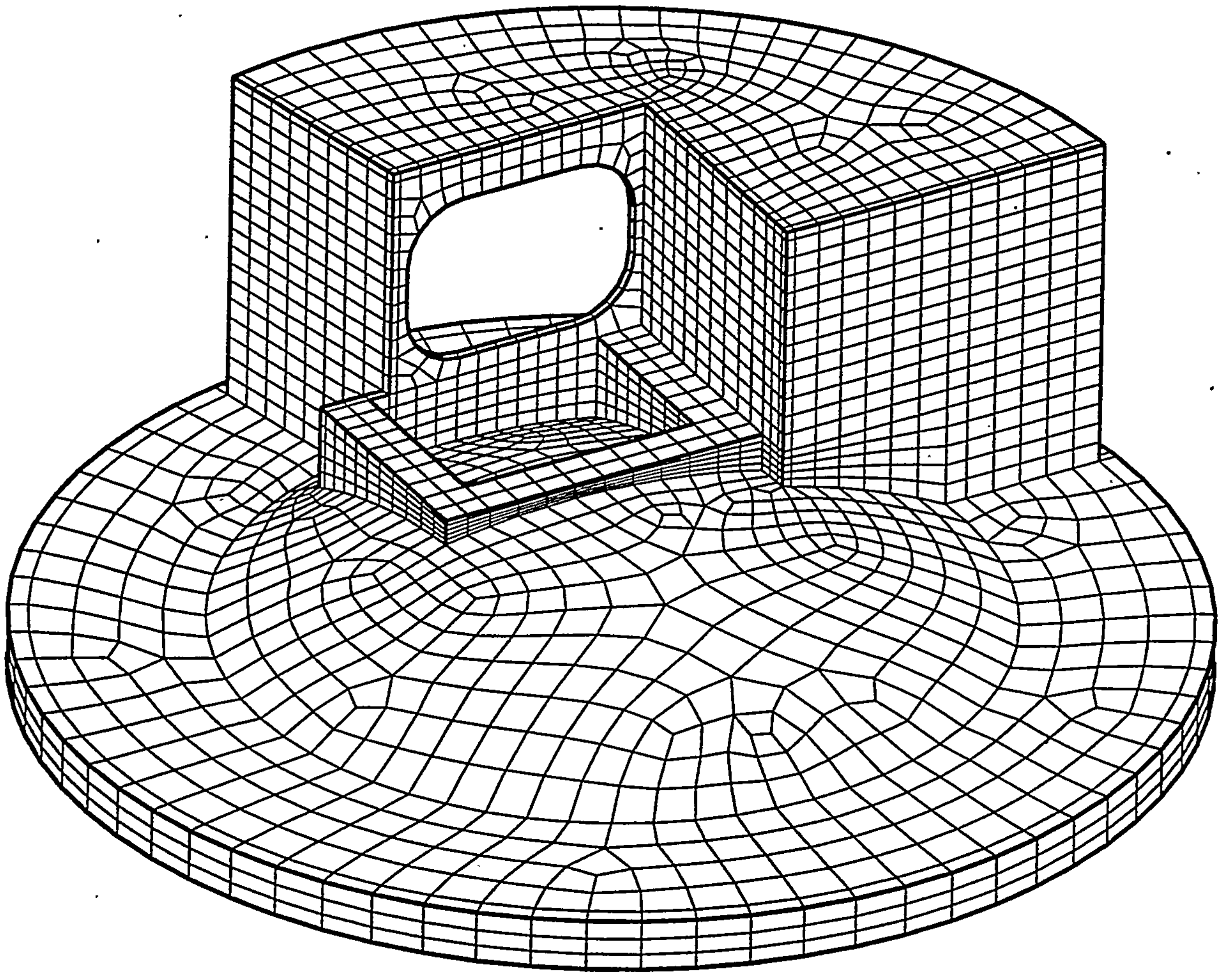


Figure 7 SRAM fireset casting.

Table 2: Comparison of Data Compression Schemes

Algorithm	Total Mb of Storage	Time (sec) to compress all the view factor rows for:		Time (sec) to decompress all the view factor rows for both enclosures
		Enclosure 1	Enclosure 2	
None	51.3	5	0.5	3
WRLE	13.4	5	0.7	3
BRLE	12.2	15	1.8	7
LZW	7.8	41	5.6	25

Table 3: Comparison of Solution Methods for Solving the Radiosity Matrix Equation

Solution Method	Compression Scheme	Time (sec) to Solve the Radiosity Matrix Equation for:	
		Enclosure 1	Enclosure 2
Gauss-Seidel	NONE	107	5.3
	WRLE	100	5.7
	BRLE	150	6.0
	LZW	354	32.1
Progressive Refinement	NONE	69	4.8
	WRLE	70	4.9
	BRLE	80	10.2
	LZW	126	11.9

3. User's Guide

The CHAPARRAL library is written in both C and FORTRAN with both a C and FORTRAN interface. The routines reside in a non-sharable library called libvf.a. Its location will be dependent upon your installation. On the Department 1500 / Building 880 LAN at Sandia National Labs, it resides in the directory `/usr/local/lib` on all the hardware platforms available on the IRN. For internal Sandia use, a standalone program called *chaparral* is available which accepts COYOTE [9] input files and just calculates the view factors. For users running on multiprocessor, shared memory machines, limited support for multiprocessing is available. Currently, the hemicube method is the only portion of the code that has been written to take advantage of a multi-processing architecture. For the Sun Microsystem's Solaris 2.x operating system, a special version of the library, libvfmt.a exists which utilizes multithreading. The multithreading currently uses Solaris threads. However, when POSIX threads become readily available on other multi-processor architectures, the port should be very easy. When running on a Sun Sparccenter 2000 with N processors, the standalone *chaparral* application demonstrates an almost N times speedup.

Informational output from the CHAPARRAL library, as defined by the debug output level, is written to standard output.

3.1 Pseudo Code for Usage of CHAPARRAL

Below is pseudo code outlining the usage of CHAPARRAL. The portions of the pseudo code that are handled by CHAPARRAL are indicated by boldface type.

```
General code initialization
Initialize internal view factor database
If view factor matrix has been previously computed
    Read view factors into database from pre-existing disk file
Else
    Loop on the number of enclosures
        Calculate view factors for this enclosure
    End loop
    Write out view factor database to disk file
End if
.
.
.
Loop on number of enclosures
    Calculate surface fluxes via Gauss-Siedel or
    Calculate surface fluxes via Progressive Refinement
```

End loop

.

.

.

End of code

3.2 C Language Interface

1. Initialize internal view factor database

```
init_vf_dbase (num_enclosures, max_surfaces, storage_format)
```

int	num_enclosures	total number of enclosures
int	max_surfaces	maximum # of surfaces from all the enclosures.
int	format	compression format: 0=no storage, 1=no compression, 2=word run length encoding, 3=byte run length encoding, and 4=LZW encoding

2. Calculate view factors

```
viewfactor (enclosure, nsurfaces, x, y, z,
            connectivity, i_param, r_param)
```

int	ienclosure	current enclosure number
int	nsurfaces	number of surfaces in this enclosure
float	*x	array of x-ccordinates for nodes
float	*y	array of y-coordinates for nodes
float	*z	array of z-coordinates for nodes
int	*iconnectivity	surface element connectivity array
int	*i_param	integer parameter array: [0] dimensionality of problem: 1=2D axisymmetric (FACET), 2=2D planar (FACET), 3=3D (hemicube), -3=3D (FACET) [1] partial enclosure flag: 0=full

```
                                 enclosure, 1=partial enclosure
                             [2] hemicube resolution, must be an
                                 even integer
                             [3] number of nodes i.e. length of x,
                                 y, and z arrays
                             [4] least squares smoothing flag:
                                 0=off, 1=on
                             [5] number of rotations
                             [6] number of x-grid divisions
                             [7] number of y-grid divisions
                             [8] number of z-grid divisions
                             [9] check for blocking: 0=off, 1=on
                             [10] maximum number of surface subdi-
                                 visions
                             [11] debug output level
                             [12] Number of child processes (or
                                 threads) to use with the hemicube
                                 algorithm
    float *r_param           real parameter array:
                             [0] area of enclosure if this is a
                                 partial enclosure problem
```

3. Read the view factors from a pre-existing disk file into the internal database

```
read_vfdbase (filename)

    char  *filename      file name
    int   format         storage format: 0=ASCII, 1=native ma-
                         chine binary, 2=XDR, 3=NETCDF (for-
                         mats 2 and 3 are not yet supported).
```

4. Write the view factor matrix to a disk file

```
write_vfdbase (filename, format)

    char  *filename      file name
    int   format         storage format: 0=ASCII, 1=native ma-
                         chine binary, 2=XDR, 3=NETCDF (for-
                         mats 2 and 3 are not yet supported).
```

5. Solve the radiosity matrix equation

```
radsolve_gauss (enclosure, radq, tsurf, eps, sigma,
                tol, niter, iter)
```

int	enclosure	current enclosure number
float	*radq	array of surface fluxes
float	*tsurf	array of surface temperatures
float	*eps	array of surface emissivities
float	sigma	Stefan-Boltzmann constant
float	tol	convergence tolerance
int	niter	maximum number of iterations
int	iter	if 0, no convergence reached, else the actual number of iterations

```
radsolve_prog(enclosure, radq, tsurf, eps, sigma,
              tol, niter, iter)
```

int	enclosure	current enclosure number
float	*radq	array of surface fluxes
float	*tsurf	array of surface temperatures
float	*eps	array of surface emissivities
float	sigma	Stefan-Boltzmann constant
float	tol	convergence tolerance
int	niter	maximum number of iterations
int	iter	if 0, no convergence reached, else the actual number of iterations

3.3 FORTRAN Language Interface

1. Initialize internal view factor database

```
call INIT_VFDB(num_enclosures, max_surfaces, iformat)
```

integer	num_enclosures	total number of enclosures
integer	max_surfaces	maximum number of surfaces from all the enclosures.
integer	iformat	compression format: 0=no storage, 1=no compression, 2=word run length encoding, 3=byte run length encoding,

and 4=LZW encoding

2. Calculate view factors

```
call VUFACTOR(ienclosure, nsurfaces, x, y, z,
              iconnectivity, i_param, r_param)
```

integer	ienclosure	current enclosure number
integer	nsurfaces	number of surfaces in this enclosure
real	x(*)	array of x-ccordinates for nodes
real	y(*)	array of y-coordinates for nodes
real	z(*)	array of z-coordinates for nodes
integer	iconnectivity(5,*)	surface element connectivity array
integer	i_param(*)	integer parameter array: (1) dimensionality of problem: 1=2D axisymmetric (FACET), 2=2D planar (FACET), 3=3D (hemicube), -3=3D (FACET) (2) partial enclosure flag: 0=full enclosure, 1=partial enclosure (3) hemicube resolution, must be an even integer (4) number of nodes i.e. length of x, y, and z arrays (5) least squares smoothing flag: 0=off, 1=on (6) number of rotations (7) number of x-grid divisions (8) number of y-grid divisions (9) number of z-grid divisions

		(10) check for blocking: 0=off, 1=on
		(11) maximum number of surface subdivisions
		(12) debug level for FACET
		(13 Number of child processes (or threads) to use with the hemicube algorithm
real	r_param(*)	real parameter array:
		(1) area of enclosure if this is a partial enclosure problem

3. Read the view factors from a pre-existing disk file into the internal database

```
call RD_VFDB(filename)
```

character*(*) filename	file name

4. Write the view factor matrix to a disk file

```
call WR_VFDB(filename, iformat)
```

character*(*) filename		file name
integer	iformat	storage format: 0=ASCII, 1=native machine binary, 2=XDR, 3=NETCDF (formats 2 and 3 are not yet supported).

5. Solve the radiosity matrix equation

```
call RADSOL_GAUSS(ienclosure, radq, tsurf, eps,
              sigma, tol, niter, iter)
```

integer	ienclosure	current enclosure number
real	radq(*)	array of surface fluxes
real	tsurf(*)	array of surface temperatures
real	eps(*)	array of surface emissivities
real	sigma	Stefan-Boltzmann constant
real	tol	convergence tolerance
int	niter	maximum number of iterations
int	iter	if 0, no convergence reached, else the actual number of it-

erations

```
call RADSOL_PROG(ienclosure, radq, tsurf, eps,
                 sigma, tol, niter, iter)
```

integer	ienclosure	current enclosure number
real	radq(*)	array of surface fluxes
real	tsurf(*)	array of surface temperatures
real	eps(*)	array of surface emissivities
real	sigma	Stefan-Boltzmann constant
real	tol	convergence tolerance
int	niter	maximum number of iterations
int	iter	if 0, no convergence reached, else the actual number of iterations

4. References

[1] Siegel, R. and Howell, J. R., Thermal Radiation Heat Transfer, McGraw Hill, NY, 2nd Edition, 1981.

[2] Emery, A. F., Johansson, O., Lobo, M., and Abrous, A., *"A Comparative Study of Methods for Computing the Diffuse Radiation Viewfactors for Complex Structures,"* AIAA / ASME / ASCE / ASH 29th SDM Conference, AIAA paper 88-2223, 1988.

[3] Shapiro, A. B., *"FACET - A Radiation View Factor Computer Code for Axisymmetric, 2D Planar, and 3D Geometries with Shadowing,"* Lawrence Livermore Laboratory Technical Report UCID-19887, August 1983.

[4] Goral, C. M., Torrance, K. E., Greenberg, D. P., and Battaile, B., *"Modeling the Interaction of Light Between Diffuse Surfaces,"* Computer Graphics, 18(3):213-222, 1984.

[5] Cohen, M. F. and Greenberg, D. P., *"The Hemi-Cube: A Radiosity Solution for Complex Environments, Computer Graphics,"* 19(3):31-40, 1985.

[6] Baum, D. R., Rushmeier, H. E., and Winget, J. M., *"Improving Radiosity Solutions Through the Use of Analytically Determined Form-Factors,"* Computer Graphics, 23(3):325-334, 1989.

[7] Cohen, M. F., Chen, S. E., Wallace, J. R., and Greenberg, D. P., *"A Progressive Refinement Approach to Fast Radiosity Image Generation, Computer Graphics,"* 22(4):75-84, 1988.

[8] Hogan, R. E., Glass, M. W., Schunk, P. R., and Gartling, D. K., *"Thermal Analysis of the Cooling and Solidification of Investment Castings,"* Proceedings of the 1st International Conf. on Transport Phenomena in Processing, S. I. Guceri, Ed., March 22-26, 1992.

[9] Gartling, D. K. and Hogan, R. E., *"Coyote II - A Finite Element Computer Program for Nonlinear Heat Conduction Problems, Part II - User's Manual,"* SAND94-1179, Sandia National Laboratories, Albuquerque, NM, 1994.

5. Distribution

M. S. Engelman
Fluid Dynamics International
500 Davis Street, Suite 600
Evanston, IL 60201

S. N. Smith
McDonnell Douglas Aerospace
13100 Space Center Blvd
Houston, TX 77059-3556

MS 0841	1500	P. J. Hommert
MS 0833	1503	J. H. Biffle
MS 0828	1504	E. D. Gorham
MS 0827	1511	D. K. Gartling
MS 0827	1511	M. W. Glass (25)
MS 0834	1512	A. C. Ratzel
MS 0834	1512	M. R. Baer
MS 0834	1512	R. J. Gross
MS 0834	1512	M. L. Hobbs
MS 0835	1513	R. D. Skocypec
MS 0835	1513	D. R. Adkins
MS 0835	1513	R. L. Akau
MS 0835	1513	B. L. Bainbridge
MS 0835	1513	E. A. Boucheron
MS 0835	1513	D. Dobranich
MS 0835	1513	R. C. Dykhuizen
MS 0835	1513	S. E. Gianoulakis
MS 0835	1513	L. A. Gritzo
MS 0835	1513	D. M. Hensinger
MS 0835	1513	R. E. Hogan
MS 0835	1513	R. R. Lober
MS 0835	1513	V. F. Nicolette
MS 0835	1513	V. J. Romero
MS 0835	1513	M. P. Sherman
MS 0835	1513	S. R. Tieszen
MS 0826	1514	W. L. Hermina
MS 0825	1515	W. H. Rutledge
MS 0833	1516	C. W. Peterson
MS 0443	1517	H. S. Morgan
MS 0437	1518	R. K. Thomas
MS 9043	8743	M. L. Callabresi
MS 9018	8523-2	Central Tech Files
MS 0899	13414	Technical Library (5)
MS 0619	13416	Technical Publications
MS 0100	7613-2	Document Processing for DOE/OSTI (10)

Org.	Bldg.	Name	Rec'd by	Org.	Bldg.	Name	Rec'd by

Sandia National Laboratories