NISTIR 6925

Calculation of Obstructed View Factors by Adaptive Integration

George N. Walton

National Institute of Standards and Technology
Technology Administration, U.S. Department of Commerce

NISTIR 6925

Calculation of Obstructed View Factors by Adaptive Integration

George Walton
Indoor Air Quality and Ventilation
Building Fire and Research Laboratory
National Institute of Standards and Technology
Mail Stop 8633
Gaithersburg, MD 20899-8633

November 2002

U.S. DEPARTMENT OF COMMERCE
Donald L. Evans, Secretary
TECHNOLOGY ADMINISTRATION
Phillip J. Bond, Under Secretary of Commerce for Technology
NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
Arden L. Bement, Jr., Director

Abstract

This report describes the use of adaptive integration for the calculation of view factors between simple convex polygons with obstructions. The accuracy of the view factor calculation is controlled by a convergence factor. The adaptive integration method is compared with two other common methods implemented in a modern computer program and found to have significant advantages in accuracy and even advantages in computational speed in some cases.

Keywords: computer algorithm, shape factor, thermal radiation, view factor

TABLE OF CONTENTS

Background

Thermal radiation is typically as important as convection in the overall heat balance in buildings. Radiant processes include the distribution of long wavelength radiation within rooms and between the building envelope and the environment plus the distribution of short wavelength (solar) radiation onto the building envelope and into rooms. The mathematical theory of thermal radiation is well established, but it can lead to complex code and/or very slow calculations. The following discussion will consider only the case of radiant interchange between diffusely reflecting and absorbing surfaces with no absorption in the intervening air. This condition is handled well by the net radiation exchange method described by Hottel and Sarofim (1967, chapter 3). It starts with the calculation of view factors. The problem with view factors is not that they are inherently difficult to compute, but that the calculation time increases exponentially with the number of surfaces involved. Algorithms that are effective for a small number of surfaces may require a hopelessly long computation time for many surfaces.

Consider a situation involving N surfaces. Since each surface may potentially interact with every other surface, there are N^2 interactions, or view factors. Even simplifications, such as the reciprocity relation (section 2.1) and the fact that a flat surface cannot view itself, reduce the number of view factors only to N(N-1)/2, which is still of order (N^2). If it is known that some, but not which, surfaces can obstruct (or "shade" or "occlude") the views between surface pairs, it is necessary to check N-2 surfaces as possible obstructing surfaces for each view factor. This gives N(N-1)(N-2)/2 obstruction checks, that is, **O**(N^3). In addition, the procedures for computing obstructed view factors are slower and less accurate than those for unobstructed view factors.

Most American energy analysis programs are currently using the radiation calculation methods that were developed two or more decades ago and have been essentially unchanged since. Those methods were very limited in their capabilities because of the lack of good methods for computing obstructed view factors. Since then many methods have been developed by many organizations including NIST: "Algorithms for Calculating Radiation View Factors Between Plane Convex Polygons with Obstructions" (Walton, 1986); and "Computer Programs for Simulation of Lighting/HVAC Interaction" (Walton, 1993). A recent extension of the NIST methods looks very promising in terms of speed and accuracy for the class of problems that occur in the simulation of heat transfer in buildings. That extension – the use of adaptive integration – is described and tested in this report.

Algorithms for Unobstructed View Factors

Before discussing the calculation of obstructed view factors, it is necessary to review methods for the calculation of unobstructed view factors.

Double Area Integration **(2AI)**

The fundamental expression for a view factor between isothermal, black-body, diffusely emitting and reflecting surfaces is

$$F_{1\rightarrow 2} = \frac{1}{\pi A_1} \int_{A_1} \int_{A_2} \frac{\cos(g_1)\cdot\cos(g_2)}{r^2} dA_2\, dA_1 \qquad (1)$$

where A_1 and A_2 are the areas of surfaces 1 and 2, g_1 and g_2 are the angles between the unit normals $\vec{n}_1$ and $\vec{n}_2$ to surface differential elements dA_1 and dA_2 and the vector, ***r***, between those differential elements, and r is the length of that vector. This nomenclature is illustrated in Figure 1. The integration over both surfaces leads to the name "double area integration" (2AI).

Double Line Integration **(2LI)**

Stokes' theorem can be used to convert the area integrals in equation (1) to line integrals:

$$F_{1\to 2} = \frac{1}{2\pi A_1} \oint_{C_1} \oint_{C_2} \ln(r)\, d\vec{v}_1 \cdot d\vec{v}_2 \qquad (2)$$

where C$_1$ and C$_2$ are the boundary contours of the surfaces and *r* is distance between $d\vec{v}_1$ and $d\vec{v}_2$, which are vector differential elements on the two contours. These elements are also shown on Figure 1. This is method is called "double line integration" (2LI) because of the two line integrals.

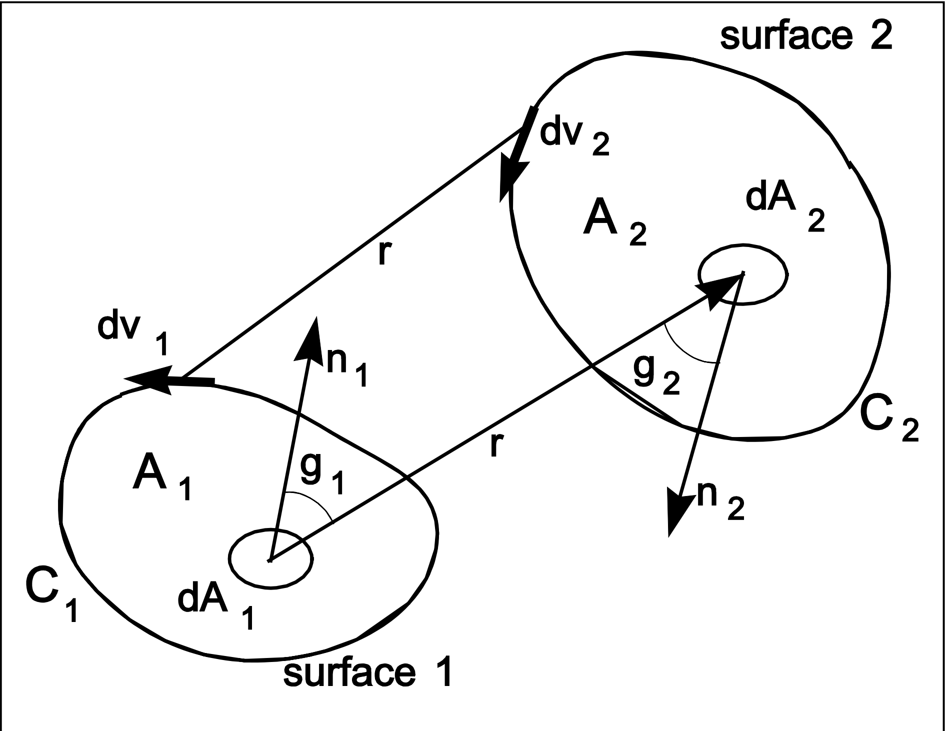


Figure 1. Geometry and Nomenclature for **2AI** and **2LI** Calculations

Numerical Calculations for Plane Polygons

Equation (1) can be expressed in terms of simple vector operations as follows. Since $\vec{r}\cdot\vec{n}_1 = r\cos(g_1)$, $\cos(g_1) = \vec{r}\cdot\vec{n}_1 / r$ and, similarly, $\cos(g_2) = -\vec{r}\cdot\vec{n}_2 / r$, where the minus sign is necessary because ***r*** ($\equiv \vec{r}_{1\to 2}$) is pointing toward surface 2 rather than away. The expression $\dfrac{\cos(g_1)\cos(g_2)}{r^2}$ can be replaced by $\dfrac{-(\vec{r}\cdot\vec{n}_1)(\vec{r}\cdot\vec{n}_2)}{(\vec{r}\cdot\vec{r})^2}$, which involves only simple arithmetic operations -- the actual values of r, g$_1$, and g$_2$ are never computed. Equation (1) can be integrated numerically by dividing both surfaces into small finite subsurfaces as also implied in Figure 1.

$$F_{1\to 2} \approx \frac{-1}{\pi A_1} \sum_i \sum_j \frac{(\vec{r}\cdot\vec{n}_1)(\vec{r}\cdot\vec{n}_2)}{(\vec{r}\cdot\vec{r})^2} \Delta A_i \, \Delta A_j \tag{3}$$

If, for example, we restrict the surface geometry to flat rectangles ($\vec{n}_1$ and $\vec{n}_2$ are then constant for each surface), and each surface is divided into N^2 subsurfaces (i.e., N divisions along each edge), the expression inside the summation will be evaluated N^4 times to compute the view factor, i.e., **2AI** is an order (O)N^4 algorithm. Both the calculation time and the accuracy of the computed view factor should increase with increasing N.

The ln(r) term in equation (2) can be simplified by the identity $\ln(r) = \ln(\vec{r}\cdot\vec{r})/2$. The dot product of the contour vector elements, $d\vec{v}_1 \cdot d\vec{v}_2$, equals $\Delta v_1 \, \Delta v_2 \cos(\Phi)$, where Δv_j is the length of $d\vec{v}_j$ and Φ is the angle between the two vector elements. When evaluating this term between any two polygon edges, p and q, the angle Φ_{pq} is constant. Equation (2) can then be approximated for numerical integration by

$$F_{1\to 2} \approx \frac{1}{4\pi A_1} \sum_{p=1}^{E_1} \sum_{q=1}^{E_2} \cos(\Phi_{pq}) \sum_{i=1}^{N_1} \sum_{j=1}^{N_2} \ln(\vec{r}\cdot\vec{r}) \Delta v_i \, \Delta v_j \tag{4}$$

where E_1 and E_2 are the number of edges of polygons 1 and 2 which are each divided into N_1 and N_2 short vectors. Again considering rectangles gives $E_1 = E_2 = 4$ and letting $N_1 = N_2 = N$, **2LI** is an **O**($16N^2$) algorithm. Whenever the surfaces are oriented so that two edges are perpendicular, $\Phi_{pq} = 0$ for that pair of edges allowing the numerical integration to be skipped for that pair of edges.

Three other methods are available for computing view factors between polygons.

Single Area Integration (**1AI**)

Hottel and Sarofim (1967, p 48) give a formula for a view factor from an infinitesimal area, dA_1, to polygon A_2. Integrating his formula over polygon A_1 gives:

$$F_{1\to 2} = \frac{1}{2\pi A_1} \int_{A_2} \sum_{i=1}^{E_2} \vec{g}_i \cdot \vec{n}_1 \tag{5}$$

where $\vec{n}_1$ is the unit normal to polygon A_1 and $\vec{g}_i$ is a vector whose magnitude is equal to the angle subtended by an edge of polygon A_2 and facing outward from the plane passing through that edge and the point at dA_1. This geometry is shown in Figure 2.

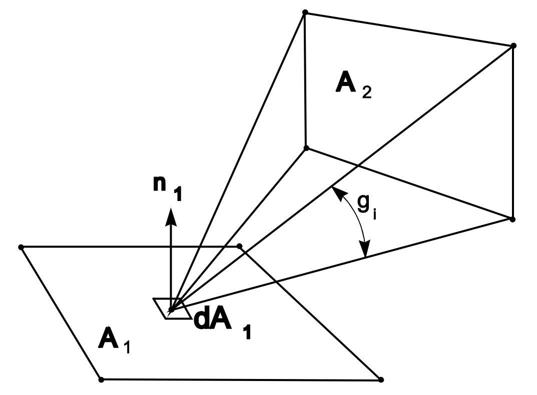


Figure 2. Single Area Integration

Equation (5) can be expressed in terms of simple vector operations as follows. Consider *a* and *b* to be two consecutive vertices defining an edge of polygon 2 and *p* the point representing dA_1. Let $\vec{b}$ be the vector from *p* to *b* and $\vec{a}$ the vector from *p* to *a*. Also let $\vec{c} = \vec{a}\times\vec{b}$, $e = |\vec{c}|$, and $d = \vec{a}\cdot\vec{b}$. Then the direction of g is given by $\vec{c}/e$ and the magnitude by tan(g) = e/d.

Substituting into equation (5) gives

$$F_{1\to 2} = \frac{1}{2\pi A_1} \int_{A_2} \sum_{i=1}^{E_1} \frac{\vec{c}_i \cdot \vec{n}_1}{e_i} \tan^{-1}\left(\frac{e_i}{d_i}\right) \tag{6}$$

If either d or e equal 0, equation (6) is undefined. e may be 0 only if a = 0, b = 0, g = 0, or g = π. None of these conditions should occur as long as dA_1 is above A_2. d will equal 0 only if a = 0, b = 0, or g = π/2. It is certainly possible to have g ≈ π/2 in which case it is best to replace $g = \tan^{-1}(e/d)$ with $g = \pi/2 - \tan^{-1}(d/e)$ which should be defined for all reasonable geometries. The $\tan^{-1}$ function is used in the evaluation of g because it is less sensitive to round-off errors than the $\sin^{-1}$ or $\cos^{-1}$ functions at certain angles. In benchmark tests it has also been faster. Replacing the integral by a summation over finite areas gives

$$F_{1\to 2} \approx \frac{1}{2\pi A_1} \sum_{j=1}^{N^2} \sum_{i=1}^{E_2} \left(\frac{\vec{c}_i \cdot \vec{n}_1}{e_i} \left[\frac{\pi}{2} - \tan^{-1}\left(\frac{d_i}{e_i}\right) \right] \right) \Delta A_j \tag{7}$$

For a pair of rectangles this is an **O**($4N^2$) algorithm.

Single Line Integration (**1LI**)

Mitalas and Stephenson [1966] present a method where one of the contour integrals in equation (2) has been solved analytically:

$$F_{1\to 2} = \frac{1}{2\pi A_1} \sum_{p=1}^{E_1} \sum_{q=1}^{E_2} \cos(\Phi_{pq}) \oint \left(t \cdot \cos(h)\ln(t) + s \cdot \cos(g)\ln(s) + u \cdot f - r \right) d\vec{v}_1 \tag{8}$$

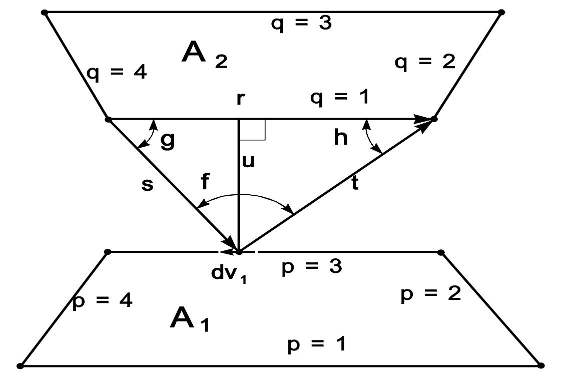


Figure 3. Single Line Integration

where vectors *s*, *t*, and u and angles f, g, and h are functions of the location of $d\vec{v}_1$ on the edges of surface A_1 as shown in Figure 3. For two rectangles this is an **O**(16N) algorithm with very complex calculations for each pair of edges. The details for simplifying the geometric calculations will not be presented. See Walton (1993).

Analytic Line Integration (**0LI**)

Schröder and Hanrahan [1993] developed analytic expressions for the double line integrals for any pair of polygon edges. These expressions depend on the geometry of the edges being relatively simple for parallel edges and very complex for skewed edges. Their report is available on-line at http://www.multires.caltech.edu/pubs/ffpaper.pdf. There is no numerical integration between any of the pairs of edges.

View3D

All five methods listed above have been implemented for testing in a computer program called View3D. The analytic line integration method was adapted from another program, Chaparral, to be described later. **0LI** is included for the following performance calculations but is not used in View3D in the later benchmark comparison tests.

Computational Performance: Speed

Figures 4 and 5 show the times to compute 100,000 view factors (on an 866 MHz Pentium-based computer) using different numbers of surface and edge divisions for the different algorithms for different surface geometries. In Figure 4 the geometry is two squares on the opposite sides of a cube which has 8 pairs of parallel edges for the line integral methods. The curves in this figure show the order of the different algorithms: **2AI - O**(N^4), **2LI - O**($8N^2$), **1AI - O**($4N^2$), and **1LI - O**(8N). The **0LI** solution requires only 2.5 s for this configuration. This happens to be about the time for all four approximate algorithms with N = 4.

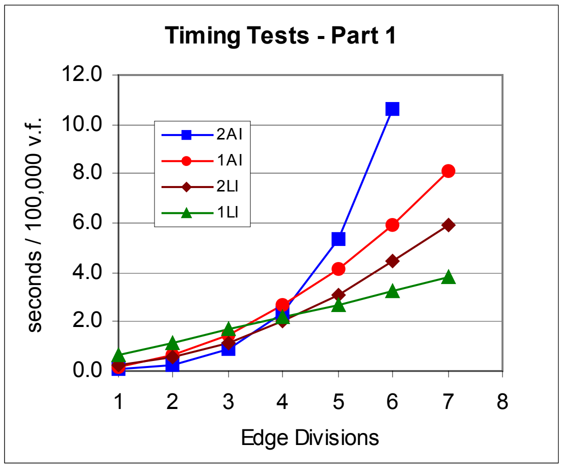


Figure 4. Timing Tests – Part 1

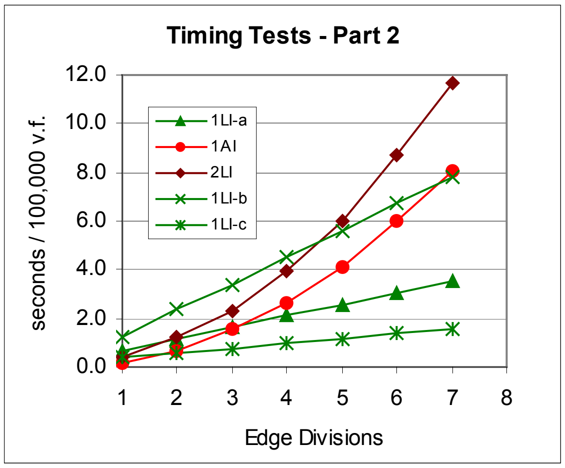


Figure 5. Timing Tests – Part 2

Figure 5 shows results for some other surface geometries as follows:

curve	description	time for **0LI**
1AI	two squares with no parallel edges	
2LI	two squares with no parallel edges	
1LI-a	two squares meeting to form an angle	28 s
1LI-b	two squares with no parallel edges	218 s
1LI-c	two perpendicular squares meeting at a common edge	1.5s

The time for the **0LI** method increases dramatically when there are non-parallel edges due to the numerical calculation of some special functions required for those cases.

Computational Performance: Accuracy

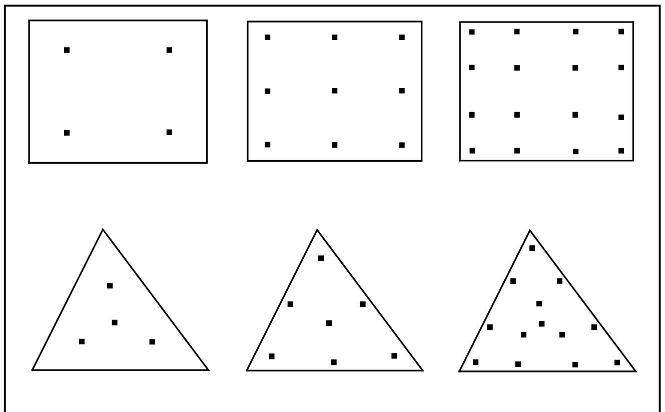


Figure 6. Points for Gaussian Area Integration

Earlier work (Walton, 1986) found that the numerical integrations of the line integrals could be much more accurate if Gaussian integration were used. In Gaussian integration, or 'quadrature', the function is evaluated at specially selected points instead of uniformly distributed points. The process is described in most introductory numerical analysis texts. Such uneven spacing can also be used in evaluating area integrals. Figure 6 shows the positions of such points for rectangles and triangles. The Gaussian integration points and weights for rectangles and parallelograms are obtained by applying 2-, 3-, and 4-point 1-D forms in both directions parallel to the edges (Press et. al., 1992, p 163). The Gaussian coefficients for triangles are more complicated (Moan, 1974). In Figure 7 the line and area integrals have been evaluated using uniformly spaced points (rectangular integration), while in Figure 8 Gaussian integration has been used for computing the view factor between surfaces on opposite sides of a cube. Note the different vertical scales.

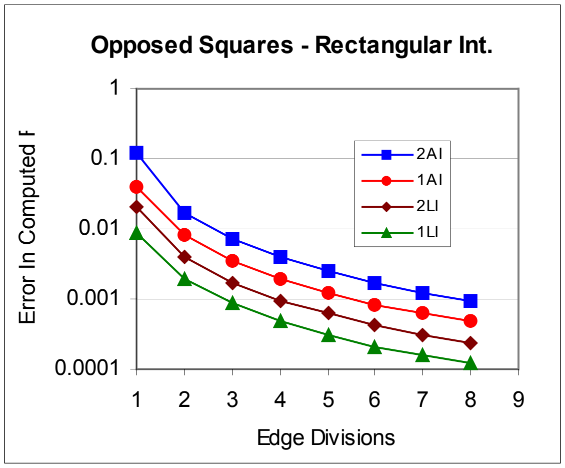


Figure 7.

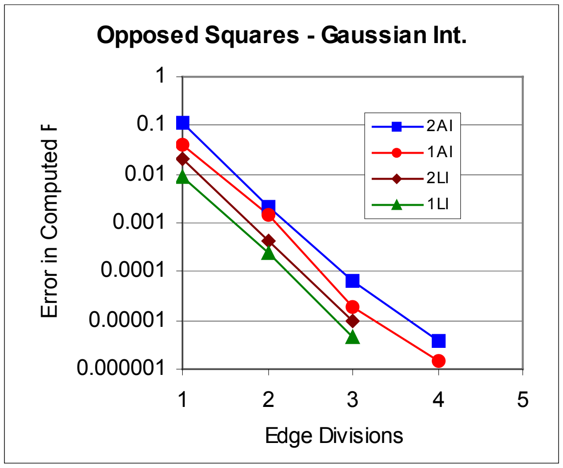


Figure 8.

The superiority of Gaussian integration is dramatic. The difference from the analytic value (0.199825) is less than 10^{-6} at N=4 for the **1LI**, **2LI**, and **1AI** methods and N=5 for the **2AI** method. Such accuracy is not achieved with N=10 using rectangular integration.

Figures 9 and 10 show the results for view factors between adjacent surfaces on a cube.

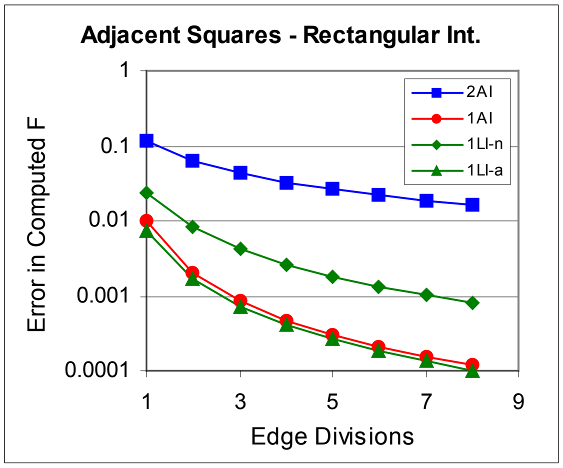


Figure 9. Adjacent Squares – Rectangular Int.

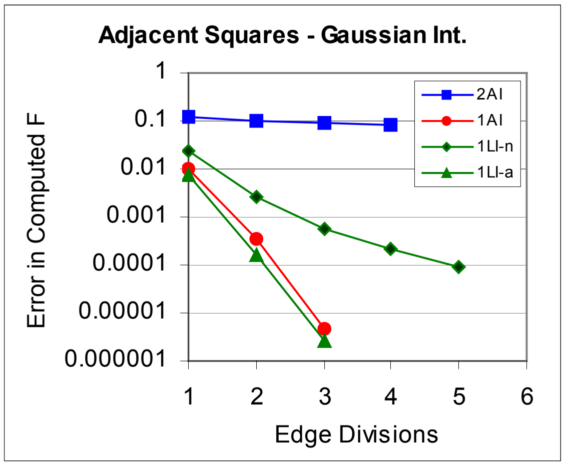


Figure 10. Adjacent Squares – Gaussian Int.

The **2LI** is not used in these tests because *r* goes to zero at some integration points along the edge shared by the adjacent surfaces. **1LI-n** uses numeric integration along the common edge while **1LI-a** uses an analytic formula and is significantly more accurate. For this geometry Gaussian integration works very well for the **1LI-a** and **1AI** methods producing errors less than 10^{-6} for N = 4. The **2AI** method is least accurate and Gaussian integration is worse than rectangular integration.

A large number of other tests indicate that the numerical integration methods require many divisions when the surfaces are relatively close, but they are quite accurate with very few divisions when the surfaces are relatively far apart. In the NIST computer program, View3D, 'relative separation' is computed as the distance between the two surface centroids divided by the sum of radii enclosing both surfaces. Relative separations greater than three quickly produce very accurate view factors.

It can be reasonably argued that the analytic view factor algorithm should be used for all cases where there are no view obstructions between the two surfaces. On the other hand, when the surfaces are widely separated, and especially when some pairs of edges are not parallel, some time can be saved by numerical integration with little loss of accuracy.

It is important to use the appropriate number of divisions in the numerical integration – too few will produce an insufficiently accurate view factor and too many will waste computation time. This can be handled by a form of adaptive integration – beginning with one edge division [*k*=1], compute successive values of the view factor with increasing numbers of divisions until the difference between two successive values is less than some specified tolerance, ε.

$$\left| AF^{[k+1]} - AF^{[k]} \right| < \varepsilon \, A_{\min} \tag{9}$$

This should be done only under conditions where the maximum value of *k* is expected to be small, *i.e.*, when the surfaces are relatively far apart. The cost in terms of additional complexity in the program to determine when to use numerical integration would have to be recovered over the life of the program.

Algorithms for Obstructed View Factors

When a third surface is added to the problem it may partially or completely block the view between the first two surfaces, or it may have no effect. Calculation of the partially obstructed view factor between two surfaces can be done by modifications to the area integration methods.

Double Area Integration with Blockage

Equation (3) can be modified to account for obstructions by the addition of a single term:

$$F_{1\to2} \approx \frac{-1}{\pi A_1} \sum_i \sum_j \frac{(\vec{r}\cdot\vec{n}_1)(\vec{r}\cdot\vec{n}_2)}{(\vec{r}\cdot\vec{r})^2} b_{i,j}\, \Delta A_i\, \Delta A_j \tag{10}$$

where the blockage factor, $b_{i,j}$, is zero if the ray connecting ΔA_i to ΔA_j is blocked by the third surface and it is one if the ray is not blocked. These two conditions are shown in Figure 11 where the ray between dA_1 and dA_2 is not blocked while the ray between dA_1 and dA_S is blocked. Multiple obstructions are easily handled by checking each ray against each obstruction until a blockage is found or all obstructions have been tested.

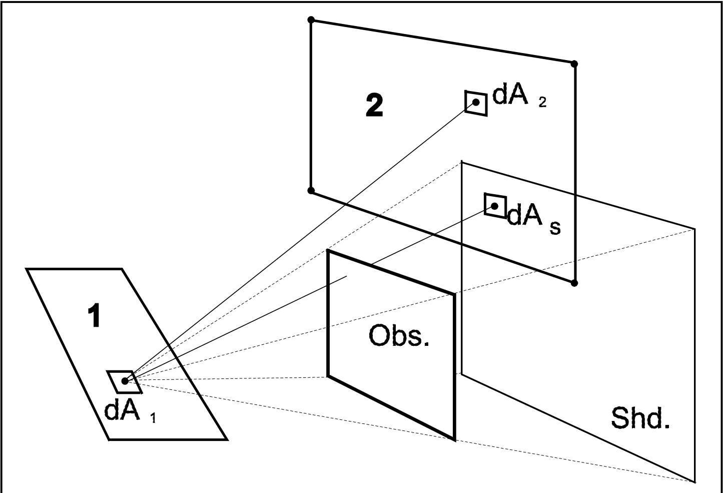


Figure 11. Partially Obstructed View

Single Area Integration with Projection

An alternate method is to project a 'shadow' of the obstruction onto the plane of the second surface. Figure 11 shows the shadow of the obstruction over a corner of surface 2. $F_{1\to2}$ is then computed by equation (7) where the inner summation is around the edges of the unshaded portions of surface 2. The position of the shadow changes with each different integration/projection point on surface 1. When there are multiple obstructions there are multiple shadows. The calculation details for processing surface and shadow polygons are given in the appendix. The method used requires that the polygons be convex which thus requires that more complex surfaces be decomposed into simpler convex polygons. The current polygon processing method provides better worst-case performance than previous methods that subtracted the view to the shadow(s) from the unobstructed view. View3D uses this method to compute obstructed view factors.

Adaptive Integration for Obstructed View Factors

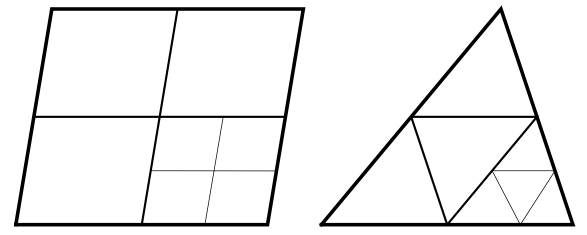


Figure 12. Adaptive Division of Polygons

Adaptive integration is used to control the number of points used for the **1AI** integration. Two view factors are computed for parallelograms using the 9- and 16-point forms shown in Figure 6 (or the 7- and 13-point forms for triangles.) If equation (9) is satisfied for those two values the result of the higher order integration is accepted. Otherwise, the surface is divided into four congruent subsurfaces as shown in Figure 12, and the view factors are computed from each of those subsurfaces. This process is repeated recursively (as shown by the subdivision of the lower right subsurfaces in Figure 12) until equation (9) is satisfied or a limit to the number of recursions is reached. The subsurface view factors are then summed to complete the $F_{1\to2}$ view factor.

This method of surface subdivision automatically provides greater refinement over the portions of the surface where it is needed. General forms for higher order integration points for triangles are not readily available. More general convex polygons are first divided into triangles.

Eliminating Potential Obstructions

In an N surface problem there are potentially N-2 view obstructions for every pair of surfaces. It is therefore very important to reduce the number of potential obstructions as quickly as possible. View3D begins by creating a list of potential view obstructing surfaces that excludes those surfaces that can never be obstructions, such as the surfaces that form a box-shaped enclosure. A surface cannot be an obstructing surface if all other surfaces are on or in front of the plane of the surface. This test, and several later tests, must determine the relationship between the vertices defining one polygonal surface and the plane containing another surface. The distance of a vertex in front of (+) or behind (-) a plane is given by

$$d = v_x s_x + v_y s_y + v_z s_z - s_w \tag{11}$$

where (v_x, v_y, v_z) are the coordinates of the vertex and (s_x, s_y, s_z, s_w) are properties of the plane: s_x, s_y, and s_z are the direction cosines of the vector normal to the plane and $s_w = s_x p_x + s_y p_y + s_z p_z$ where $\vec{p}\ (= \overrightarrow{0p})$ is for any point on the plane.

The elements in the lower half of the matrix of view factors $F_{i,j}$ are computed by row from i = 1 to i = N and within each row from j = 1 to j = i-1. ($F_{i,i}$ = 0 for flat surfaces.) When the elements of row i are computed, a reduced list of obstructing surfaces is created that excludes those surfaces that are completely behind surface i.

The view between a pair of surfaces, i and j, may be obstructed by their own positions and orientations. It may be that surface i is entirely behind surface j or that surface j is entirely behind i giving $F_{i,j}$ = 0. It is also possible that a surface is only partially behind the plane of the other surface, in which case it is necessary to remove, or 'clip', the portion of the one surface which lies behind the other before continuing with the view factor calculation. Clipping may increase the number of vertices by one.

A series of tests are then made to further reduce the list of potential obstructions. The first removes any surface where the sphere enclosing the surface lies outside the cylinder enclosing surfaces i and j.

This test uses a simple vector cross product calculation. When surfaces *i* and *j* have significantly different radii, it is advantageous to use a slightly more complex test based upon a cone rather than a cylinder. This test is most useful when the surfaces have similar sizes as will often occur in the critical large N case. The second test involves determining the minimal box containing both surfaces *i* and *j*. If a surface is entirely outside that box, it cannot be an obstructing surface. This test works best when surfaces are aligned along the axes of the Cartesian coordinate system. There are three surface orientation relationships where surface *k* cannot obstruct the view from between surfaces *i* and *j*: (1) *k* entirely behind *j* (j cannot see *k*), (2) *i* and *j* entirely in front of *k*, and (3) *i* and *j* entirely behind *k*.

If these tests have removed all surfaces from the list of potential obstructions, $F_{i,j}$ will be computed by one of the algorithms for unobstructed view factors. Otherwise, one more test will be made to determine whether the shadow of the obstructing surfaces will be projected from *i* or from *j*. This test is based on the observation (see below) that more accurate view factors are usually computed by projecting the obstruction toward the nearest surface. After the direction of projection has been chosen, any obstructions that can see only the source surface are deleted from the list. $F_{1,2}$ is then computed by adaptive single area integration with obstruction projection.

Analytic Test

Shapiro (1983) presents a configuration of surfaces that has an analytic solution for an obstructed view factor. The test consists of two directly opposed unit squares (surfaces 1 and 2) with unit separation and a pair of back-to-back 0.5 x 0.5 squares (surfaces 3 and 4) parallel to the unit squares, centered on a line between the centers of the unit squares, and 3/4 of for the distance from surface 1 to surface 2. Only surface 1 is visible from surface 3, and only 2 from 4. Figure 13 shows the squares from the side to highlight the fact that every ray drawn from surface 1 through the obstruction will intercept surface 2.

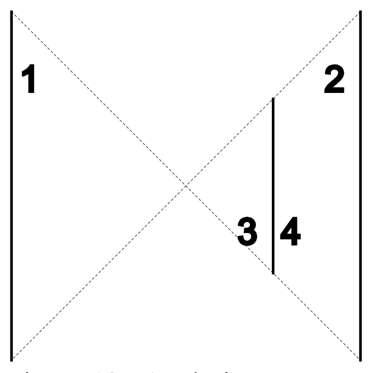


Figure 13. Analytic Test

Analytic solutions for unobstructed views:

$F_{3,1} = 0.33681717$; $F_{1,3} = 0.084204294$; $F_{4,2} = 0.79445272$; $F_{2,4} = 0.19861318$

$F^*_{1,2} = F^*_{2,1} = 0.19982490$ (if 3 and 4 are not present).

Therefore, the analytic solution for the obstructed view is

$F_{1,2} = F^*_{1,2} - F_{1,3} = 0.11562061$

The results of View3D calculations in Table 1 consist of the adaptive integration tolerance, ε from equation (9), the computed view factor, the error compared to the analytic value, and the number of projection points used. The first two rows are for shadow projections from surface 1. The 25 points indicate that the solution was done with one 9-point and one 16-point integration – surface 1 was not subdivided. The 125 points in the second row indicate that surface 1 was subdivided one time to achieve a remarkably accurate answer. The last five rows in the table are for shadow projections from surface 2, i.e., compute $F_{2,1}$ and use reciprocity to determine $F_{1,2}$. In this case many of the projected shadows will only partially

ε	$F_{1,2}$	error	points
10^{-3}	0.11563653	0.00015924	25
10^{-4}	0.11562055	-0.00000006	125
10^{-3}	0.11473675	-0.00088386	525
10^{-4}	0.11526465	-0.00035596	925
10^{-5}	0.11553235	-0.00008826	2925
10^{-6}	0.11560305	-0.00001756	8125
10^{-7}	0.11561626	-0.00000435	18525

Table 1. Results for Analytic Test

overlap surface 1. Many more evaluations are being performed, but the error is properly decreasing as ε decreases.

The results of evaluating the same case with double area integration and view blockage (equation 10) are shown in Table 2 for comparison. These results were obtained with the Chaparral program described below. Sampling points were distributed both uniformly and randomly (Monte Carlo) on surfaces 1 and 2 with the number of points determined by the input 'mc_nsamples'. These results exhibit the same slow convergence to a solution shown in Figure 7 for the calculation of an unobstructed view factor.

sampling: mc_nsamples	uniform error	random error
25	0.001691	-0.003032
100	0.011172	0.017821
400	0.003847	0.001563
2500	0.003289	-0.007019
10000	0.000414	-0.000892
40000	0.000111	-0.000775
250000	0.000315	-0.000253
1000000	0.000154	-0.000271

Table 2. 2AI Results for Analytic Test

Problem Cases

When obstructing surfaces are close to both surfaces 1 and 2, as in Figure 14, there is no optimum direction of projection. The solution will require many adaptive steps to approach an accurate answer. There is a need for an estimate of accuracy when analytic solutions are not available, which is generally the case for partially obstructed views. The following discussion will use enclosures for test cases. In an enclosure the sum of the view factors from any surface should be one. The summations will be called "rowsums". The maximum value of $\left|\left(\sum_{j=1}^{N} F_{i,j}\right)-1\right|$ will be used as a figure of merit - the "rowsum error". It is not necessarily a true representation of accuracy because positive and negative errors for individual view factors may offset, but it is statistically difficult for errors to cancel across many rows in the view factor matrix.

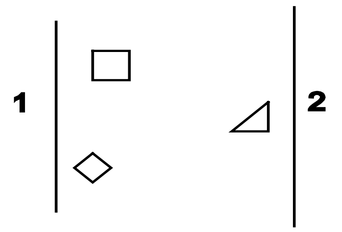


Fig. 14. Problem Case 1

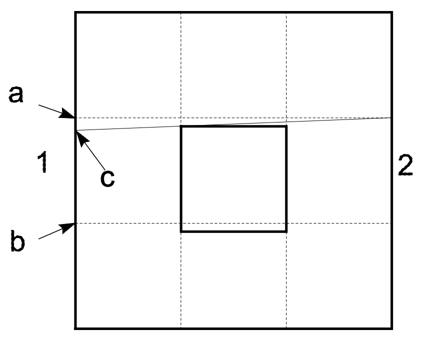


Figure 15. Problem Case 2

A more serious case is shown in Figure 15 which is a cross-sectional representation of a cubic enclosure with 9 unit squares on each face of a cubic obstruction also composed of unit squares near its center giving a total of 60 surfaces. A small portion of surface 1, which extends from **a** to **b**, can view a portion of surface 2 through the space above the obstructing cube. Surface 2 can be viewed only from the area between points **a** and **c**. If there are no integration points in that area, adaptive subdivision will not occur in that area and $F_{1,2}$ will be computed as 0. Gaussian integration helps on this problem because it tends to force the integration points toward the edge of the surface as was shown in Figure 6. The coordinates of the outer points for 4×4 integration are at 0.06943 times the width of the surface from its edge, while uniform subdivision would put those points at 0.125 from the edge.

Table 3 shows the results for $F_{1,2}$ with different shifts of the obstruction and convergence factors, ε. When the obstruction is centered in the enclosure ($\Delta z = 0$), smaller ε values give rapidly decreasing rowsums. For Δz shifts between 0.01 and 0.04 the rowsums are not converging to zero because the values of $F_{1,2}$ are being computed as zero instead of the more accurate values shown in the final column of Table 3. When the shift reaches 0.05 the rowsums are again decreasing well because the outer-most integration points on surface 1 can see surface 2. The values of $F_{1,2}$ shown in the final column were computed by forcing additional polygon subdivisions until very accurate rowsums were achieved. These values indicate the absolute errors in $F_{1,2}$ are not very large. Relative error is not a meaningful term when $F_{1,2}$ is incorrectly computed to be zero.

Shift	convergence factors ε				$F_{1,2}$
Δz	1.0e-4	1.0e-5	1.0e-6	1.0e-7	
0.00	0.000151	0.000024	0.000006	0.000001	0.000000
0.01	0.000152	0.000030	0.000028	0.000028	0.000006
0.02	0.000185	0.000114	0.000112	0.000060	0.000021
0.03	0.000258	0.000257	0.000255	0.000062	0.000046
0.04	0.000205	0.000107	0.000069	0.000075	0.000082
0.05	0.000240	0.000096	0.000014	0.000006	0.000128

Table 3. Case 2 Figure of Merit Values: $\left| \left(\sum_{j=1}^{N} F_{i,j} \right) - 1 \right|_{\max}$

These problem cases would also be difficult for most other obstructed view factor procedures.

Benchmark Tests

The next step in evaluating View3D is to compare it against another program. The Chaparral program from Sandia National Laboratory (Glass, 2001) was chosen because of its recent development. It has two fundamentally different methods to compute view factors for 3D geometries: "adaptive" and "hemicube". The adaptive method computes view factors one pair at a time using a collection of algorithms that adapt to the geometry of the problem. It includes testing for obstructions between surfaces pairs and then the **0LI** method for unobstructed views or **2AI** with blockage for the obstructed views. The hemicube method calculates the view factors from one surface to all other surfaces and proceeds row-by-row through the view factor matrix. The hemicube method is analogous to Nusselt's hemisphere method but uses the hemicube for computational efficiency. It was developed (Cohen et al. 1985) to use the radiosity method for the computer graphic display of scenes composed of large numbers of surfaces.

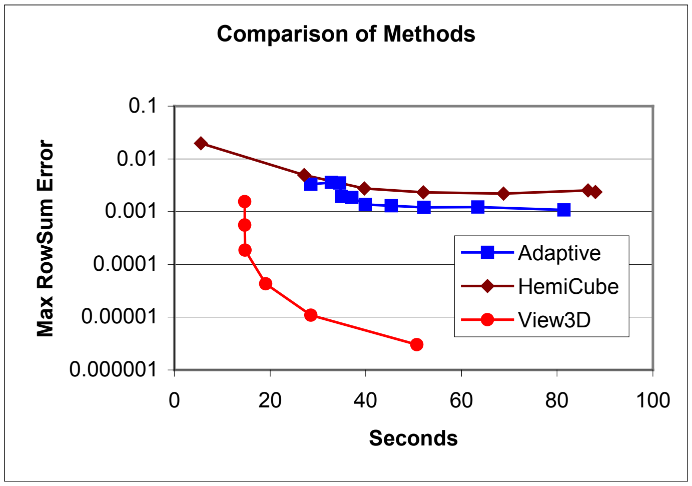


Figure 16. Comparison of Methods

Figure 16 compares the three methods for a simple test case consisting of a 4×4×4 cube made up of unit squares centered in a 10×10×10 cube also composed of unit squares for a total of 696 square surfaces. For the hemicube method the number of surface subdivisions for close surfaces was found to be the most important one of several input parameters and was varied from 1 to 7. For the adaptive method the number of uniformly spaced samples on each surface varied from 3×3 to 40×40. For View3D the integration tolerance was varied from 10^{-2} to 10^{-7}.

For this case the primary advantage of View3D is the much smaller rowsums achieved without excessive computation times.

It is also important to determine how the different methods respond to the size of the problem. A series of similar test cases was created consisting of a small cube made up of unit squares centered in a large cube also composed of unit squares. Table 4 summarizes the six test cases indicating the total number of surfaces, the number of surfaces on the small obstructing cube, and the percentage of surfaces that are potential view obstructions – none of the surfaces of the enclosing cube can obstruct a view.

name	description	total srf.	obstr. srf.	% obstr.
BB52	2x2x2 box in 5x5x5 box	174	24	13.8
BB73	3x3x3 box in 7x7x7 box	348	54	15.5
BB104	4x4x4 box in 10x10x10 box	696	96	13.8
BB156	6x6x6 box in 15x15x15 box	1566	216	13.8
BB208	8x8x8 box in 20x20x20 box	2784	384	13.8
BB2510	10x10x10 box in 25x25x25 box	4350	600	13.8

Table 4. Test Cases of Increasing Size

The accuracy comparisons shown in Figure 17 for the range of test cases are consistent with Figure 16. The adaptive method was used with 25 and 100 samples from each surface and the hemicube method used 2 and 4 surface subdivisions. The accuracy of the hemicube method first gets better as the separation between surfaces increases then gets worse as aliasing errors increase in the grid on each face of the hemicube. The accuracy of the adaptive method increases as the average relative separation between surfaces increases with the number of surfaces. The rowsum error for View3D is below 10^{-3} for an integration tolerance of 10^{-4}. "View3D-O" will be described later.

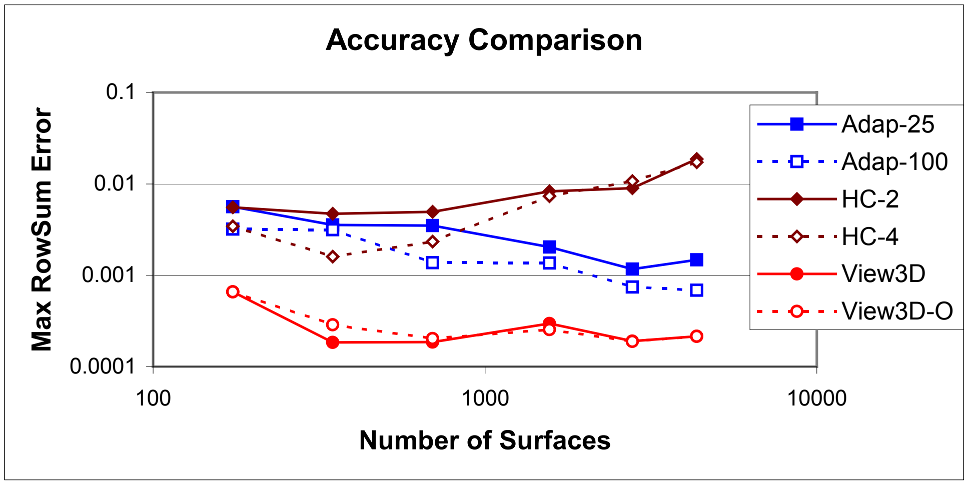


Figure 17. Accuracy Comparison

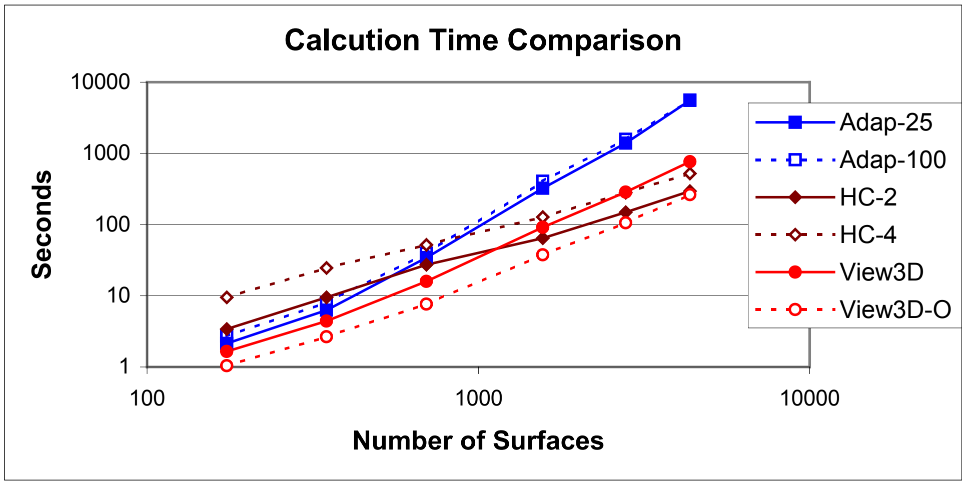


Figure 18. Calculation Time Comparison

The comparison of calculation times is shown in Figure 18 and Table 5.

case	Nsrf	View3D	View3d-O	Adap-25	Adap-100	HC-2	HC-4
BB52	174	1.65	1.04	2.14	2.69	3.41	9.49
BB73	348	4.40	2.64	6.26	8.02	9.45	24.55
BB104	696	15.98	7.63	34.05	39.60	27.14	52.02
BB156	1566	92.00	37.40	323.23	407.93	64.70	127.26
BB208	2784	285.06	105.79	1403.24	1575.70	149.18	280.18
BB2510	4350	762.47	264.08	5534.41	5598.19	296.49	524.10

Table 5. Computation Times Benchmark

Notable trends include:

- The rapidly increasing computation times for Chaparral's adaptive method – going from 2 s to 5600 s as the number of surfaces increases from 174 to 4350.
- The slower increase for View3D that reaches 762 s for the largest case.
- The different trend for the hemicube method that takes longer than the other two methods for cases with a small number of surfaces but a much shorter time for cases with many surfaces.

Other tests indicate that most of the calculation time in the adaptive method was being used to determine which surfaces were obstructions. The initialization used by View3D to eliminate the surfaces that can never obstruct, in this case those forming the outer cube, seems especially helpful. Chaparral tests all surfaces as potential obstructions for every surface pair. This is probably the main reason it is much slower than View3D for the cases with many surfaces.

View3D has an option to group obstruction surfaces as subsurfaces of a single surface – in these cases the unit squares of the obstructing cubes can be grouped into six surfaces that represent the faces of the cubes. In case BB2510 the 600 unit squares are reduced to six squares for the purpose of evaluating view shadows. This reduces the computation time from 762 s to 264 s. Therefore, about two-thirds of the 762 s was spent processing obstructions. Results for View3D grouping the obstructions are labeled "View3D-O" in Figure 18 and Table 5. This method of surface grouping is optimal for the tests shown so the calculation times should not be considered typical. The adaptive method includes a grouping method, binary space partitioning, which is more general.

The hemicube method is both slower and less accurate than View3D for the tests involving small numbers of surfaces. Such small numbers would be representative of rooms and their furnishings in building energy analysis. Hemicube becomes faster than View3D at about 1000 surfaces although accuracy still suffers. It is becoming less accurate because of aliasing errors on the hemicube grid. This problem could be improved by using a finer grid but at the cost of greater execution time.

Conclusions and Possible Future Developments

This report has demonstrated that adaptive integration (as implemented in View3D) can compute view factors more accurately than ray tracing or hemicube methods (as implemented in Chaparral) and faster than the ray tracing method. Methods that use a predetermined integration method or number of integration subdivisions will tend to either compute some view factors less accurately than desired or waste time computing others too accurately. Adaptive integration is especially applicable when the surfaces are relatively close – a condition that occurs in rooms with odd shapes and/or furnishings. View3D has a speed advantage for configurations with a relatively small number of surfaces – again typical of rooms. Therefore, the methods in View3D have notable advantages for computing room view factors for building energy analysis programs. The rapid elimination of potential view obstructions in the view between any pair of surfaces is very important for problems involving large numbers of surfaces, and will probably be the main emphasis of future developments.

The computational performance of View3D for relatively large numbers of surfaces, on the order of two thousand, indicate that it could be the basis for modeling the more challenging problem of the radiant environment around buildings as well. With slight modification the shadow projection and overlap method can be used to model the shadowing of direct sunlight. View factors can also model the incidence of diffuse light on the building. This includes the light reflected by the ground and adjacent buildings with those surfaces in sunlight or shadow and, by dividing the sky into many patches, the effect of shadowing surfaces on anisotropic light. View factors can also model the long-wavelength radiant interchange between the building and its surroundings.

Ray-tracing methods automatically allow the modeling of non-ideal surfaces – those that are not diffusely emitting and reflecting. Hottel and Sarofim (1967, chapter 5) describes a method of reflected images that can be used to compute view factors for specular and partially specular surfaces. This method is probably better than ray tracing if only a small fraction of the total number of surfaces are specular. Adding such a reflection algorithm would permit modeling of specular reflections from adjacent glass covered buildings.

There are some methods used in Chaparral that could be advantageously used in View3D and *vice versa*. Chaparral could benefit from the method of automatically determining which surfaces can never obstruct views and it might benefit from placing the ray tracing points according to 2D Gaussian integration. View3D could use the analytic algorithm for unobstructed views and, for cases involving large numbers of surfaces, a method like binary space partitioning which permits the elimination of groups of surfaces in the determination of the potential view obstructions.

Acknowledgement

The U.S. Department of Energy, Office of Building Technologies, supported this work under Interagency Agreement No. DE-A101-9CE21042.

References

Cohen, M.F. and D.P. Greenberg. 1985. "The Hemi-Cube -- A Radiosity Solution for Complex Environments," *ACM SIGGRAPH*, Vol 19, No 3.

Emery, A.F., O. Johansson, M. Lobo, A. Abrous. 1991. "A Comparative Study of methods for Computing the Diffuse Radiation Viewfactors for Complex Structures," *ASME Journal of Heat Transfer*, 113: 413-422.

Glass, M.W. 2001. "CHAPARRAL v2.x: A Library for Solving Enclosure Radiation Heat Transfer Problems" (draft), Sandia National Laboratories.

Hottel, H.C. and A.F. Sarofim. 1967. *Radiative Transfer*, McGraw-Hill, New York NY.

Mitalas, G.P. and D.G. Stephenson. 1966. "FORTRAN IV Programs to Calculate Radiant Interchange Factors," National Research Council of Canada, Division of Building Research, Ottawa, Canada, BDR-25.

Moan, T. 1974. "Experiences with Orthogonal Polynomials and 'Best' Numerical Integration Formulas on a Triangle; with Particular Reference to Finite Element Approximations," *Zeitschrift Fuer Angewandte Mathematik und Mechanik*, 54:501-508.

Newman, W.M., and R.F. Sproull. 1973. *Principles of Interactive Computer Graphics*, McGraw-Hill.

Pavlidis, T. 1982. *Algorithms for Graphics and Image Processing*, Computer Science Press, Rockville MD.

Press, W.H., S.A. Teukolsky, W.T. Vetterling and B.P. Flannery. 1992. *Numerical Recipes in C: the Art of Scientific Computing, Second Edition*, Cambridge University Press.

Schroder, P. and P. Hanrahan, 1993. "A Closed Form Expression for the Form Factor Between Two Polygons", Department of Computer Science, Princeton University, Technical Report CS-404-93.

Shapiro, A.B. 1985. "Computer Implementation, Accuracy, and Timing of Radiation View Factor Algorithms," *ASME Journal of Heat Transfer*, 107: 730-732.

Walton, G.N. 1978. "The Application of Homogeneous Coordinates to Shadowing Calculations," *ASHRAE Transactions*, Vol 84, Part I.

Walton, G.N., 1986. "Algorithms for Calculating Radiation View Factors Between Plane Convex Polygons With Obstructions," National Bureau of Standards, NBSIR 86-3463 (1987 - shortened report in *Fundamentals and Applications of Radiation Heat Transfer*, HTD-Vol.72, American Society of Mechanical Engineers).

Walton, G.N. 1993. "Computer Programs for Simulation of Lighting/HVAC Interactions," National Institute of Standards and Technology, NISTIR 5322.

Appendix: Numerical Processing of Convex Polygons

The processing of geometric data in the view factor algorithms is based on conventional vector calculations which are familiar to most engineers and on homogeneous coordinate (HC) techniques which have found extensive application in computer graphics but are unfamiliar to most engineers. This appendix reviews the fundamental properties of two-dimensional HC and describes a method for processing overlapping convex polygons.

Homogeneous Coordinates

HC are described in most textbooks about the mathematics of computer graphics, such as [Newman and Sproull, 1973; Pavlidis, 1982]. Points and lines in HC are represented by a single form which allows simple vector operations between those forms. A point (X, Y) is represented by a three element vector (x, y, w) where x = wX, y = wY, and w is any real number except zero. A line is also represented by a three element vector (a, b, c). The directed line (a, b, c) from point (x_1, y_1, w_1) to point (x_2, y_2, w_2) is given by

$$\begin{aligned}(a,b,c) &= (x_1,y_1,w_1)\times(x_2,y_2,w_2)\\ &= (y_1w_2 - y_2w_1,\ w_1x_2 - w_2x_1,\ x_1y_2 - x_2y_1)\end{aligned} \tag{12}$$

The sequence in the cross product is a convention to determine sign. The condition that a point (x, y, w) lie on a line (a, b, c) is that

$$(a,b,c)\cdot(x,y,w) = ax + by + cw = 0 \tag{13}$$

Normalize a point by dividing its coordinates by w. Then if

$$(a,b,c)\cdot\left(\frac{x}{w},\frac{y}{w},1\right) > 0 \tag{14}$$

the point (x, y, 1) lies to the left of the line, and if it is less than zero, the point lies to the right of the line (as viewed from the starting point of the line). The intercept (x, y, w) of line (a_1, b_1, c_1) and line (a_2, b_2, c_2) is given by

$$(x,y,w) = (a_1,b_1,c_1)\times(a_2,b_2,c_2) \tag{15}$$

The use of HC as outlined above provides a consistent method and notation for defining points and lines, for determining intercepts, and for determining whether a point lies to the left, to the right, or on a line. Normalization provides a means of transforming between HC and Cartesian coordinates. Thus, if (X, Y) is the Cartesian coordinate pair of a point, its HC description is (X, Y, 1). Similarly, the HC point (x, y, w) can be transformed to the Cartesian point (x/w, y/w).

The area, A, of a plane polygon consisting of n sequential vertices (x_1, y_1), (x_2, y_2), ..., (x_n, y_n) is given by

$$A = \tfrac{1}{2}(x_1y_2 + x_2y_3 + \cdots + x_{n-1}y_n + x_ny_1 - y_1x_2 - y_2x_3 - \cdots - y_{n-1}x_n - y_nx_1) \tag{16}$$

If the HC of the vertices have all been computed with the same value of w (for example w = 1), then area is expressed more simply in terms of the c coordinates of the edges as

$$A = \frac{1}{2w^2} \sum_{i=1}^{n} c_i \tag{17}$$

The area is positive if the vertices are in counter-clockwise sequence and negative if clockwise.

Problems may occur during computation because of the critical importance of the value zero in determining the relationship of a point to a line in equation (3). Computer arithmetic with 'real' numbers is subject to round-off error, so the tests against zero in equations (2) and (3) may fail. To prevent this, the tests are done against some small number, typically 10^{-5} times the area of the surfaces involved in the calculation.

Numerical Processing of Convex Polygons

The numerical processing of polygons to determine the portions that do or do not overlap will be shown by example. Figure A1 shows two polygons that partially overlap. Polygon A consists of vertices a-b-c-d in clockwise sequence and polygon B consists of vertices e-f-g-h-i. These are convex polygons -- the internal angle at every vertex is less than 180°. Convexity is necessary for the simple mathematical operations used to determine the part of B that overlaps A and the part of B that is outside A. This is done by successively testing the vertices of B against each edge of A.

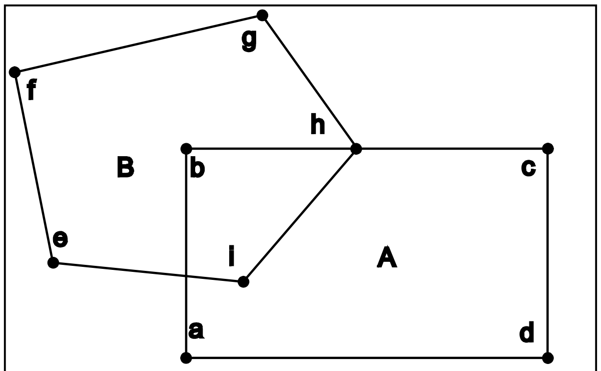


Figure A1 Two Overlapping Polygons

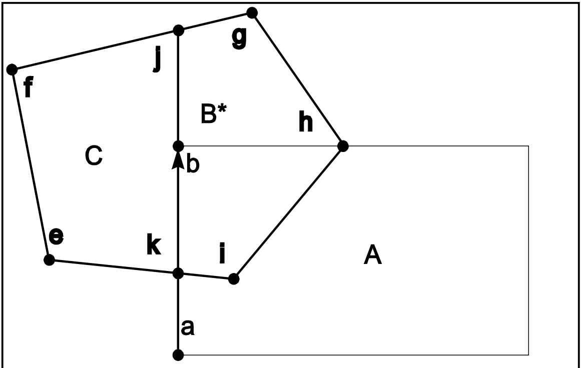
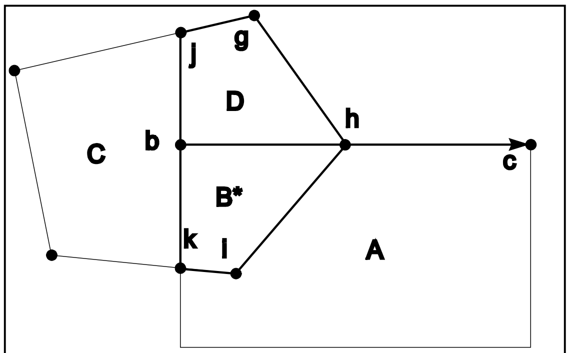


Figure A2 Test against Side a-b

Figure A2 shows the first test against edge a-b. Application of equation (3) will show that vertices e and f are to the left of edge a-b (extended) and vertex g is to the right. Since f and g are on opposite sides of edge a-b, that edge must intercept edge f-g. That intercept is computed by equation (4) and labeled j in the figure. Vertices h and i are also to the right of edge a-b. Vertices i and e are on opposites sides of the edge so another intercept at K is computed. The portion of polygon B that lies to the left of edge a-b (polygon C: vertices e-f-j-k) cannot overlap polygon A. The portion to the right of edge a-b (polygon B*: vertices j-g-h-i-k) may overlap depending on the results of tests against other edges. C and B* are convex polygons.

Polygon B* is then tested against edge b-c of polygon A as shown in Figure A3. Vertices j and g are to the left of edge b-c; vertex h is the edge. Vertices i and k are to the right of edge b-c. Vertices k and j are on opposites sides of the edge so another intercept is computed. It is identical to vertex b on polygon A.



Figuare A3 Test against Side b-c

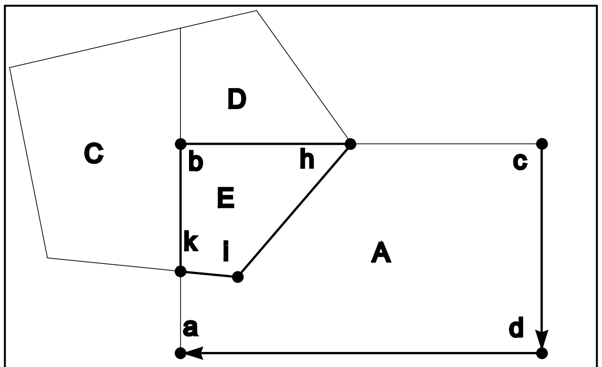


Figure A4 Tests against Sides c-d and d-a

Polygon D is another part of B that is entirely outside polygon A being composed of all vertices (j-g-h-b) to the left of or on edge b-c. A new B* consisting of vertices (h-i-k-b) on or right of edge b-c remains to be processed against the remaining edges of polygon A.

Tests against edges c-d and d-a in Figure A4 show that vertices h-i-k-b are all to the right of both edges, so polygon E (h-i-k-b) is the inter-section of polygons A and B.

If at any time during the tests against an edge of polygon A, it is found that all vertices of B are left of the edge, then A and B do not overlap. If all vertices of B are on or to the right of all edges of A, then B is entirely within A. If the area of the overlap is equal to the area of A, then A is within B. Whenever a polygon split into parts to the left and right of an edge it is possible that one of the new polygons may have one more vertex than the original. The maximum number of vertices in any single polygon will tend to remain small because all polygons are convex. Vertices are stored in linked list data structures that maintain the order of the vertices as the new polygons are created. Polygons are also stored in linked list data structures to allow the generation of an indefinite numbers of surfaces.

The most common alternative method for evaluating non-overlapping polygon areas divides the base polygon into many small areas, determines which of those areas lie within the overlap, and adds up the areas of those not within the overlap. That method can be faster that the method presented above if the number of small areas is not large, but that leads to relatively inaccurate answers. For example, if a square is divided into N^2 small areas the accuracy of the solution tend to be proportional to N, but the calculation time will be proportional to N^2. High accuracy becomes computationally expensive. This method provides very accurate answers with execution time dependent only of the total number of polygons being processed.

This method is used in View3D to evaluate the portion of a polygon that can be seen from an integration point after the view obstructing surfaces have been projected onto the plane of the polygon. It has improved the worst-case performance of the original method which computed only overlapping portion of two polygons (Walton, 1978). The obstructed view factor was computed by subtracting the view factors to the shadows of the obstruction polygons from the unobstructed view factor. When obstruction shadows overlapped the view factor to that overlap had to be added back into the total view factor. Multiple overlaps could become quite complicated. This original method was developed for processing shadows of sunlight to assess solar heat gains in buildings. The current method should also be used for that purpose because of its improved worst-case performance.