UCLA
COMPUTATIONAL AND APPLIED MATHEMATICS

Integral Equation Preconditioning for the Solution of Poisson's Equation on Geometrically Complex Regions

Christopher R. Anderson
Archie C. Li

June 1997

CAM Report 97-25

Department of Mathematics
University of California, Los Angeles
Los Angeles, CA. 90095-1555

INTEGRAL EQUATION PRECONDITIONING FOR THE SOLUTION OF POISSON'S EQUATION ON GEOMETRICALLY COMPLEX REGIONS

CHRISTOPHER R. ANDERSON * AND ARCHIE C. LI †

Abstract. This paper is concerned with the implementation and investigation of integral equation based solvers as preconditioners for finite difference discretizations of Poisson equations in geometrically complex domains.

The target discretizations are those associated with "cut-out" grids. We discuss such grids and also describe a software structure which enables their rapid construction. Computational results are presented.

1. Introduction. This paper deals with the creation of effective solvers for the solution of linear systems of equations arising from the discretization of Poisson's equation in multiply connected, geometrically complex, domains. The focus is on discretizations associated with "cut-out" grids (grids which result from excluding select points from a uniform grid).

The solvers we describe are iterative procedures which use integral equation solutions (such as those described in [9, 14, 15, 16, 17, 21]) as preconditioners. One may question the need for such an approach; "If one is going to the trouble to implement an integral equation solver, why bother with solving the discrete equations?". The need for such an approach arises in applications in which the solution of the linear system is of primary importance and obtaining the solution of the partial differential equation is a secondary matter. An application where this occurs (and the one which inspired this work) is the implementation of the discrete projection operator associated with the numerical solution of the incompressible Navier-Stokes equations [12]. Our selection of the integral equation procedure as a preconditioner was motivated by its ability to generate solutions for multiply connected domains possessing complex geometry.

With regard to the use of "cut-out" grids to discretize Poisson's equation we are re-visiting an old technique — the particular discretization procedure used is credited to Collatz (1933)[8]. For Poisson's equations, the concept behind the discretization procedure is not complicated; but the actual construction of discrete equations for general geometric configurations using this concept can be. As we will discuss, by combining computer drawing tools with an intermediate software layer which exploits polymorphism (a feature of object oriented languages) the construction of equations can be simplified greatly.

* Department of Mathematics, UCLA, Los Angeles, California 90024. This research was supported in part by Office of Naval Research grant ONR-N00014-92-J-1890 and Air Force Office of Scientific Research Grant AFOSR-F49620-96-1-0327

† Advanced Development Group, Viewlogic Systems Inc., Camarillo, CA 93010 (ali@qdt.com).

In the first section we briefly discuss the constituent components of the complete procedure — the discretization associated with a "cut-out" grid, the iterative method chosen to solve the discrete equations, and the integral equation based preconditioner. In the second section we present numerical results, and in the appendix we discuss details of the integral equation method.

While our solution procedure is developed for discretizations based on "cut-out" grids, the results should be applicable to discretizations associated with other grids (e.g. triangulations or mapped grids). Additionally, there has been active research on discretization procedures for other equations using cut-out grids; e.g. equations for compressible and incompressible flow [2, 4, 5, 7, 18, 25, 26, 27]. The method we describe for constructing equations could be extended to those discretizations as well.

2. Preliminaries.

2.1. The Mathematical Problem. The target problem is the solution of Poisson's equation on a multiply connected, geometrically complex domain. Let Ω be a bounded domain in the plane with a C^2 boundary consisting of M inner contours $\partial\Omega_1 \cdots \partial\Omega_M$, and one bounding contour $\partial\Omega_{M+1}$ ($\partial\Omega = \bigcup_{k=1}^{M+1} \partial\Omega_k$, see Fig. 1). Given

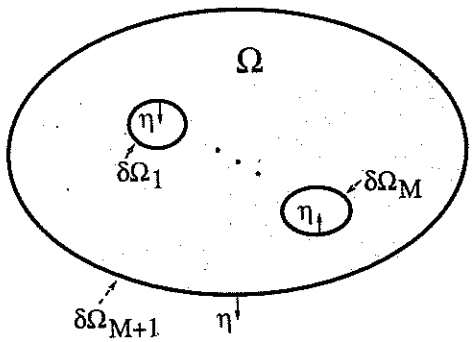

FIG. 1. *A bounded domain.*

boundary data g and forcing function f, we seek the solution to the following equation:

$$\begin{aligned} \Delta u(\boldsymbol{x}) &= f(\boldsymbol{x}), \quad \boldsymbol{x} \in \Omega \\ \lim_{\substack{\boldsymbol{x}\to\boldsymbol{x}_o \\ \boldsymbol{x}\in\Omega}} u(\boldsymbol{x}) &= g(\boldsymbol{x}_o), \quad \boldsymbol{x}_o \in \partial\Omega \end{aligned} \tag{1}$$

In the unbounded case, Ω is the unbounded domain that lies exterior to M contours $\partial\Omega_1 \cdots \partial\Omega_M$ ($\partial\Omega = \bigcup_{k=1}^{M} \partial\Omega_k$, see Fig. 2), and we seek a solution to

$$\begin{aligned} \Delta u(\boldsymbol{x}) &= f(\boldsymbol{x}), \quad \boldsymbol{x} \in \Omega \\ \lim_{\substack{\boldsymbol{x}\to\boldsymbol{x}_o \\ \boldsymbol{x}\in\Omega}} u(\boldsymbol{x}) &= g(\boldsymbol{x}_o), \quad \boldsymbol{x}_o \in \partial\Omega \\ u(\boldsymbol{x}) &= O(1), \quad (\boldsymbol{x} \to \infty). \end{aligned} \tag{2}$$

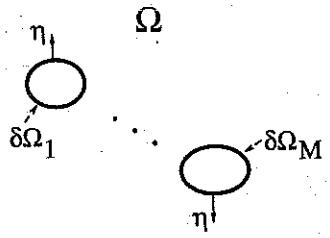

FIG. 2. *An unbounded domain.*

2.2. The Spatial Discretization. Approximate solutions to (1) or (2) are obtained as solutions of a linear system of equations arising from finite difference discretizations. The discretization procedure we used was a "cut-out" grid approach [2, 4, 5, 7, 18, 25, 26, 27]. We selected this discretization procedure because the formulation of the linear system of equations requires little information about the geometry; one need only know if grid points are inside, outside, or on the boundary of the domain and (for points nearest the boundary) the distance of grid points to the boundary along a coordinate axis. Thus, a program can easily be created which automatically constructs a discretization based on information available from minimal geometric descriptions (e.g. descriptions output from a drawing or CAD package).

To form our "cut-out" grid we consider a rectangular region R that contains the domain Ω (for unbounded problems, R contains the portion of Ω that we are interested in). We discretize R with a uniform Cartesian grid, and separate the grid points into three groups: regular, irregular, and boundary points. A regular point is a point whose distance along a coordinate axis to any portion of the domain boundary $\partial\Omega$ is greater than one mesh width. An irregular point is one whose distance to a portion of the boundary is less than or equal to one mesh width but greater than zero, and boundary grid points lie on the boundary (see Fig. 3). Regular and irregular points are further identified as being interior or exterior to the domain. We compute an approximate Poisson solution by discretizing (1) or (2) using the regular and irregular interior grid points. These discrete equations are derived using centered differences and linear interpolation (described as "Procedure B" in [26], and based on ideas presented as far back as [8]): If we introduce the standard five point discrete Laplacian (here, h denotes the mesh width of the Cartesian grid)

$$\Delta_h u(\boldsymbol{x}_{i,j}) \equiv \frac{u(\boldsymbol{x}_{i+1,j}) + u(\boldsymbol{x}_{i-1,j}) + u(\boldsymbol{x}_{i,j+1}) + u(\boldsymbol{x}_{i,j-1}) - 4u(\boldsymbol{x}_{i,j})}{h^2}, \tag{3}$$

then at each regular interior point an equation is given by

$$\Delta_h u(\boldsymbol{x}_{i,j}) = f(\boldsymbol{x}_{i,j}), \quad \boldsymbol{x}_{i,j} \text{ a regular interior point.} \tag{4}$$

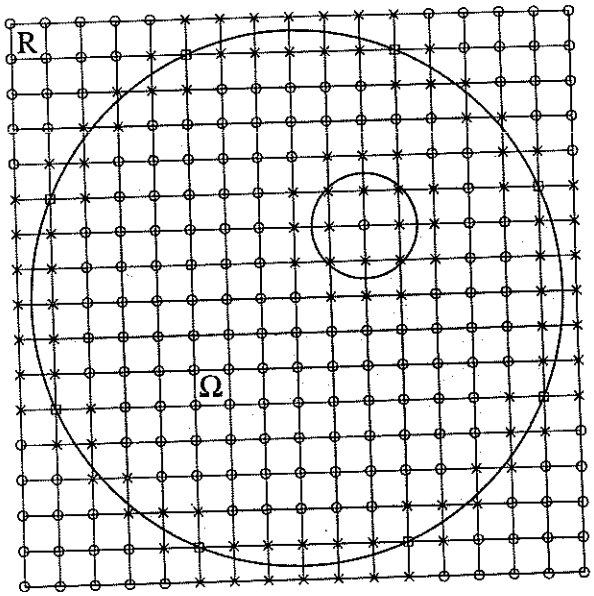

FIG. 3. *A "cut-out" grid: the regular points are marked by circles, irregular points by crosses, and boundary points by squares.*

At each irregular interior point an equation is obtained by enforcing an interpolation condition. Specifically, at an irregular point we specify that the solution value is a linear combination of boundary values and solution values at other nearby points. For example, for an irregular interior point $\boldsymbol{x}_{i,j}$ with a regular interior point $\boldsymbol{x}_{i-1,j}$ one mesh width to its left, and a point on the boundary $\boldsymbol{x}_R$ at a distance d_R to its right (see Fig. 4), a second order Lagrange interpolating polynomial (linear interpolation) can be used to specify an equation at $\boldsymbol{x}_{i,j}$:

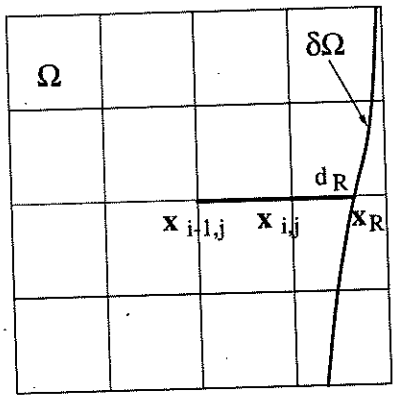

FIG. 4. *Linear interpolation at an irregular interior point.*

$$(5)\quad u(\boldsymbol{x}_{i,j}) - u(\boldsymbol{x}_{i-1,j})\frac{d_R}{h+d_R} = g(\boldsymbol{x}_R)\frac{h}{h+d_R}, \quad \boldsymbol{x}_{i,j} \text{ an irregular interior point.}$$

If this linear interpolation procedure is used, and if the boundary and forcing functions are sufficiently smooth, then the solution of the discrete equations yields values of second order accuracy [26].

This discretization procedure produces a linear system of equations

$$A\vec{x} = \vec{b}, \tag{6}$$

where $\vec{x}$ consists of the solution values at all interior grid points, and $\vec{b}$ involves both the inhomogeneous forcing terms and the boundary values. Due to the interpolation used, the matrix A is usually nonsymmetric.

2.3. Automated Construction of Discrete Equations. As previously remarked, one benefit of using discretizations based upon cut-out grids is that their construction requires a minimal amount of information from the geometry; thus one can create programs which take geometric information output from rather modest drawing tools and automatically construct the required discretizations.

The process we employed going from geometric information to the discretization is described by the functional diagram in Fig. 5.

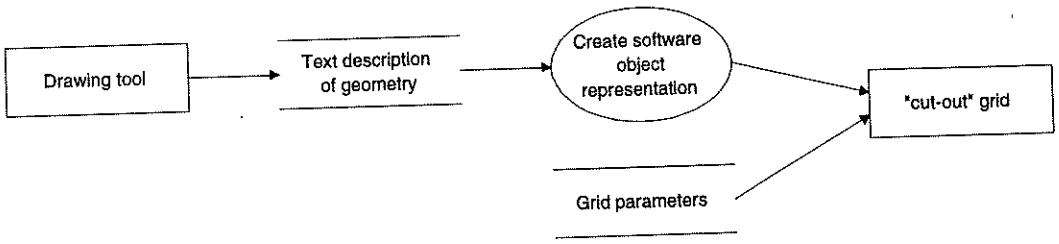

FIG. 5. *Functional depiction of discretization process.*

Key to this process is the introduction of an extra software layer between the drawing tool and the program to create the discretization. In particular, we take a text representation of the drawing and map this to a software representation in which each of the entities that makes up the geometric description is represented as a distinct software object. The program which creates the discretization uses only the functional interface associated with these software objects. Hence, the discretization can be constructed independently from any particular drawing tool output. (To accommodate output from different drawing tools we are just required to construct code which maps the geometric information to the software objects which represent it.)

The class description, using OMT notation [22], associated with the geometric software objects is presented in Fig. 6. (While these classes were implemented as C++ classes, other languages which support class construction could be used).

As indicated in Fig. 6, there is a base class `GeometricEntity` which is used to define a standard interface for all geometric entities. From this base class we derive classes which implement the base class functionality for each particular type of geometric entity. The types created were those which enabled a one-to-one mapping from typical drawing tool output to software objects. Since a "drawing", as output from a drawing tool, is typically a collection of geometric entities; a class `CombinedGeometricEntity` was created to manage collections of their software counterparts.

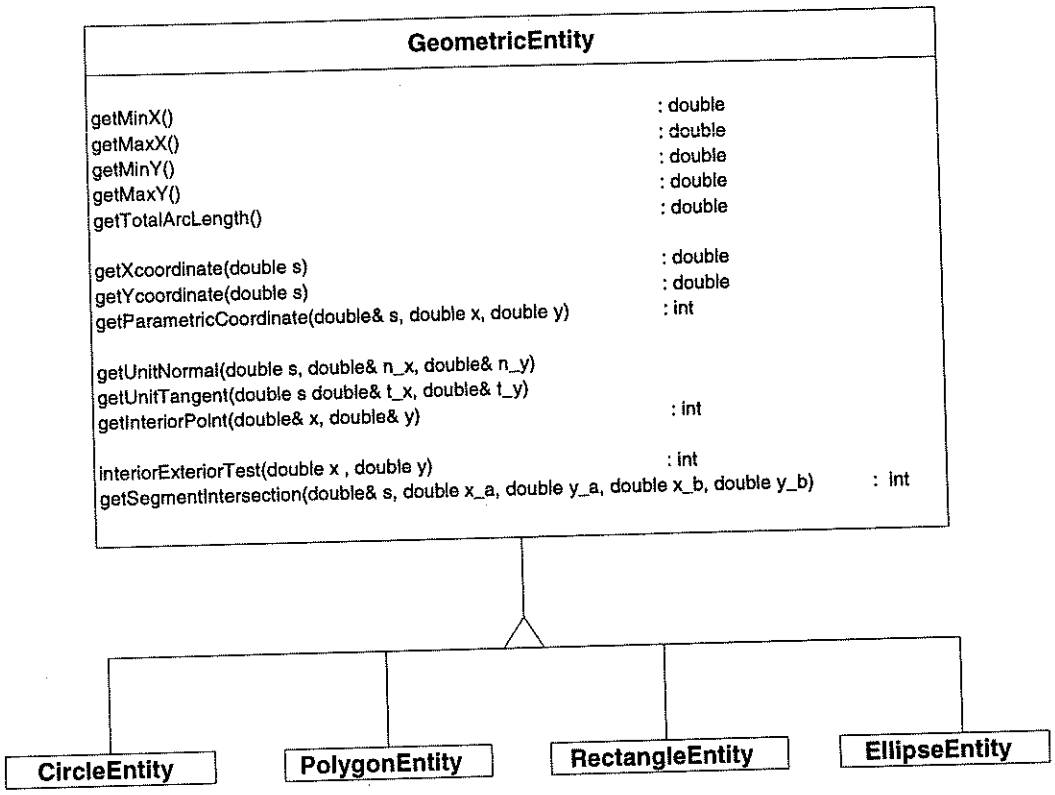

FIG. 6. *Description of classes used to store and access geometric information.*

In the program which creates the discretization, only functionality associated with the base class `GeometricEntity` is used. Thus, this program doesn't require modification if the set of derived classes (i.e. classes implementing particular geometric entity types) is changed or added to. The program will function with any new or changed entity as long as that entity is derived from the base class and implements the base class functionality. This class structure also enables the discretization program to use procedures optimized for particular types of geometric entities. (For example, the interior/exterior test for a circle is much more efficient than that for a general polygon.) This occurs because polymorphism is supported; when a base class method is invoked for a derived class, the derived class' implementation is used.

The success of the intermediate software layer depends upon the functionality associated with the base classes. Ideally, the required functionality should be obtainable with a small number of methods which are easy to implement. (The restriction on the number of methods is desirable because each method must be implemented for all derived classes.) As indicated from the class description, the functionality required to construct "cut-out" grid discretizations and integral equation pre-conditioners can be implemented with a very modest set of methods. It is this latter fact that makes the use of "cut-out" grids attractive; complicated procedures are not required to incorporate geometric information into the construction of a grid and discretizations associated with such a grid.

2.4. Solution Procedure. The discrete equations (4-5) are solved using preconditioned simple iteration. As discussed in the next section, with appropriate preconditioner implementation, more sophisticated iterative procedures are not required. If P is used to denote the preconditioner, and $\vec{r}^n \equiv \vec{b} - A\vec{x}^n$ represents the residual error of the nth iterate, then preconditioned simple iteration can be written as follows:

$$\vec{x}^{n+1} = \vec{x}^n + P\vec{r}^n. \tag{7}$$

The general form of the preconditioner (or approximate inverse) is the solution procedure (and its variants) described in [9, 14, 15, 17, 16, 21], coupled with a relaxation step to improve its efficiency.

The procedure (without the relaxation step) begins by using a Fast Poisson solver to obtain function values that approximately satisfy the Poisson equation at regular interior points. These values do not satisfy the discrete equations at irregular interior points nor do they satisfy the boundary condition; therefore, we correct them by adding function values obtained from the solution of an integral equation. One challenge is to determine the appropriate integral equation problem to supply this correction. This task presents a challenge because we are mixing two types of discretization procedures, finite difference and integral equation discretizations. Additionally, since we are using the solution procedure as a preconditioner we wish to achieve reasonable results without using a highly accurate (and thus more costly) integral equation solution.

The solution component which is obtained with the Fast Poisson solver is constructed to satisfy

$$\begin{aligned} \Delta_h u^{FPS}(\boldsymbol{x}_{i,j}) &= \begin{cases} f(\boldsymbol{x}_{i,j}), & \boldsymbol{x}_{i,j} \text{ is a regular interior point} \\ 0, & \boldsymbol{x}_{i,j} \text{ otherwise.} \end{cases} \\ u^{FPS}(\boldsymbol{x}_{i,j}) &= 0, \quad \boldsymbol{x}_{i,j} \in \partial R. \end{aligned} \tag{8}$$

The correction to $u^{FPS}(\boldsymbol{x}_{i,j})$ that satisfies the correct boundary conditions is a solution of Laplace's equation with boundary conditions $g^{IE}(\boldsymbol{x}_o) \equiv g(\boldsymbol{x}_o) - u^{FPS}(\boldsymbol{x}_o)$:

$$\begin{aligned} \Delta u^{IE}(\boldsymbol{x}) &= 0, \quad \boldsymbol{x} \notin \partial\Omega \\ \lim_{\substack{\boldsymbol{x}\to\boldsymbol{x}_o \\ \boldsymbol{x}\in\Omega}} u^{IE}(\boldsymbol{x}) &= g^{IE}(\boldsymbol{x}_o), \quad \boldsymbol{x}_o \in \partial\Omega \end{aligned} \tag{9}$$

(Note: the correction for the unbounded case is similar, see [17] for details.)

This problem can be solved and evaluated at the regular interior points by using the integral equation approach of Appendix A. The approximate solution to the discrete Poisson problem is formed by combining the Fast Poisson solver solution with the integral correction terms.

$$\tilde{u}(\boldsymbol{x}_{i,j}) \equiv u^{FPS}(\boldsymbol{x}_{i,j}) + u^{IE}(\boldsymbol{x}_{i,j}) \tag{10}$$

Standard truncation error analysis reveals that if $\boldsymbol{x}_{i,j}$ is a regular interior point:

$$\Delta_h \tilde{u}(\boldsymbol{x}_{i,j}) = f(\boldsymbol{x}_{i,j}) - \frac{h^2}{12}(u^{IE}_{xxxx} + u^{IE}_{yyyy}), \tag{11}$$

while if $\boldsymbol{x}_{i,j}$ is an irregular interior point (for convenience we assume that the irregular point is like the one shown in Fig. 4. Alternate cases will have analogous error terms):

$$\tilde{u}(\boldsymbol{x}_{i,j}) - \tilde{u}(\boldsymbol{x}_{i-1,j})\frac{d_R}{h+d_R} = g(\boldsymbol{x}_o)\frac{h}{h+d_R} + \tilde{u}_{xx}\frac{hd_R}{2}. \tag{12}$$

The solution procedure leads to a truncation error that is formally second order; therefore, we expect that it will make a good preconditioner. However, since the accuracy at the irregular points depends on the magnitude of $\tilde{u}_{xx}(\boldsymbol{x})$ (or $\tilde{u}_{yy}(\boldsymbol{x})$), there is a dependence on the smoothness of $u^{FPS}(\boldsymbol{x})$. The smoothness of $u^{FPS}(\boldsymbol{x})$ depends on the discrete forcing values used in (8), and these forcing terms may not be smooth because the specified terms $f(\boldsymbol{x}_{i,j})$ will be the residual errors of the iterative method (which can be highly oscillatory) and because the zero extension used may result in forcing values that are discontinuous across the boundary. To remedy this, we incorporate a relaxation scheme as part of the preconditioning step. A common feature of relaxation schemes is that they result in approximate solutions with smooth errors, even after only a few iterations. Therefore, we apply our approximate Poisson solver to the smooth error equation resulting from the relaxation step, and then combine these terms to form the approximate solution.

That is, we first apply a few iterations of a relaxation scheme (point Jacobi).

$v^0(\boldsymbol{x}_{i,j}) = 0$, $\boldsymbol{x}_{i,j}$ an interior point

`for`$(n = 0 \ldots 3)$

`if` ($\boldsymbol{x}_{i,j}$ a regular interior point)

$$v^{n+1}(\boldsymbol{x}_{i,j}) = \frac{v^n(\boldsymbol{x}_{i-1,j}) + v^n(\boldsymbol{x}_{i+1,j}) + v^n(\boldsymbol{x}_{i,j-1})}{4} + \frac{v^n(\boldsymbol{x}_{i,j+1}) - h^2 f(\boldsymbol{x}_{i,j})}{4}; \tag{13}$$

`if` ($\boldsymbol{x}_{i,j}$ an irregular interior point)

$$v^{n+1}(\boldsymbol{x}_{i,j}) = v^n(\boldsymbol{x}_{i-1,j})\frac{d_R}{h+d_R} + g(\boldsymbol{x}_R)\frac{h}{h+d_R}; \tag{14}$$

`end`

After the relaxation step, we compute the residual error: If $\boldsymbol{x}_{i,j}$ is a regular interior point

$$e(\boldsymbol{x}_{i,j}) \equiv f(\boldsymbol{x}_{i,j}) - \Delta_h v^4(\boldsymbol{x}_{i,j}), \tag{15}$$

and if $\boldsymbol{x}_{i,j}$ is an irregular interior point

$$e(\boldsymbol{x}_{i,j}) \equiv g(\boldsymbol{x}_R)\frac{h}{h+d_R} - (v^4(\boldsymbol{x}_{i,j}) - v^4(\boldsymbol{x}_{i-1,j})\frac{d_R}{h+d_R}). \tag{16}$$

Next, the Fast Poisson solver is applied where the forcing consists of the residual error of the relaxation iterate with zero extension.

$$\Delta_h u^{FPS}(\boldsymbol{x}_{i,j}) = \begin{cases} e(\boldsymbol{x}_{i,j}), & \boldsymbol{x}_{i,j} \text{ an interior point} \\ 0, & \boldsymbol{x}_{i,j} \text{ otherwise.} \end{cases} \tag{17}$$
$$u^{FPS}(\boldsymbol{x}_{i,j}) = 0, \quad \boldsymbol{x}_{i,j} \in \partial R.$$

Then, the integral equation approach is used to solve the correcting Laplace problem. (Now, $g^{IE}(\boldsymbol{x}_o) \equiv -u^{FPS}(\boldsymbol{x}_o)$)

$$\Delta u^{IE}(\boldsymbol{x}) = 0, \quad \boldsymbol{x} \notin \partial\Omega \tag{18}$$
$$\lim_{\substack{\boldsymbol{x}\to\boldsymbol{x}_o \\ \boldsymbol{x}\in\Omega}} u^{IE}(\boldsymbol{x}) = g^{IE}(\boldsymbol{x}_o), \quad \boldsymbol{x}_o \in \partial\Omega$$

Finally, the three terms are combined to form the approximate solution.

$$\tilde{\tilde{u}}(\boldsymbol{x}_{i,j}) \equiv v^4(\boldsymbol{x}_{i,j}) + u^{FPS}(\boldsymbol{x}_{i,j}) + u^{IE}(\boldsymbol{x}_{i,j}) \tag{19}$$

A truncation error analysis shows that at regular interior points:

$$\Delta_h \tilde{\tilde{u}}(\boldsymbol{x}_{i,j}) = f(\boldsymbol{x}_{i,j}) - \frac{h^2}{12}(u^{IE}_{xxxx} + u^{IE}_{yyyy}), \tag{20}$$

while at irregular interior points (after making use of (14))

$$\tilde{\tilde{u}}(\boldsymbol{x}_{i,j}) - \tilde{\tilde{u}}(\boldsymbol{x}_{i-1,j})\frac{d_R}{h+d_R} = \tag{21}$$
$$g(\boldsymbol{x}_o)\frac{h}{h+d_R} + (u^{IE}_{xx} + u^{FPS}_{xx})\frac{hd_R}{2} + (v^3(\boldsymbol{x}_{i,j}) - v^4(\boldsymbol{x}_{i,j})).$$

The combined solution procedure (13-19) comprises the preconditioner for the iterative solver. We expect that due to the smoother forcing values used in (17), u^{FPS} will have smaller second derivatives; therefore, $\tilde{\tilde{u}}$ should satisfy the discrete equations better than $\tilde{u}$, hence the addition of smoothing to the solution procedure should result in a better preconditioner.

3. Computational Results. The iterative procedure described above has been implemented, and in this section we evaluate its effectiveness on two bounded domains. For all domains and discretizations considered, we apply forcing values $f(x,y) = 12x^2 + 6y^2$ and boundary values $g(x,y) = x^4 + 1/2y^4$.

Example 1: We first consider the domain (with smooth, C^2 boundary) depicted in Fig. 7. For an 80x80 grid, we use simple iteration to solve the discrete equations within a relative residual error of 10^{-10}. We apply the integral equation preconditioner both with and without the relaxation step, and vary the number of boundary points used to solve the integral equation. The resulting iteration counts are given in Table 1. We observe that the addition of the relaxation step increases the effectiveness of the preconditioner (as expected), and that the number of iterations needed to achieve

our tolerance is quite low (5-7 iterations for $\frac{\|A\vec{x}-\vec{b}\|}{\|\vec{b}\|} < 10^{-10}$). Furthermore, we see that the number of boundary points used in the integral equation step can be significantly reduced while maintaining the effectiveness of the preconditioner. This illustrates that integral equation preconditioning can be efficient since relatively few points are needed to solve the integral equation.

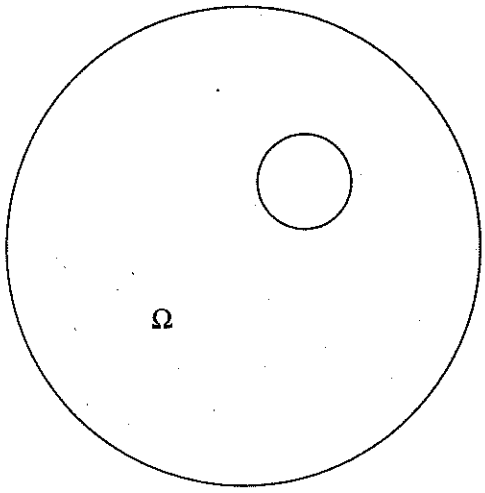

FIG. 7. *A domain with a smooth boundary.*

Example 2: Our second example compares the effectiveness of the preconditioner for two different iterative solvers (simple iteration and FGMRES [23]) and for different grid refinements. Starting with the same smooth domain (Fig. 7), we formulate the discrete equations for four grid refinements. In each case we solve the discrete equations up to a tolerance of 10^{-10}. Both iterative solvers are preconditioned using the integral equation procedure with relaxation, and the results are listed in Table 2. We see that with this preconditioner, simple iteration is just as effective a solver as FGMRES, and this allows us to solve the discrete equations using less memory and fewer computations. This example also demonstrates that the convergence of the preconditioned iterative methods is independent of the grid refinement. This is expected since the preconditioner is based on a solution procedure for the underlying equation.

Example 3: In this example, we test our method on a domain with corners (Fig. 8). This geometry represents the cross section of three traces in an integrated circuit chip with deposited layers and undercutting. In this situation, the Poisson solver can be used to extract electrical parameters such as the capacitance and inductance matrices. We formulate the discrete equations for a 40x40 grid, and apply the integral equation preconditioner with and without relaxation. Since we no longer have a C^2 boundary, we do not meet the smoothness assumptions that our preconditioner requires. In fact, for this problem in which sharp corners are present, the effectiveness of the integral equation solver as a preconditioner deteriorates. One finds an increase in the required number of iterations, an increase which is not reduced by improving the accuracy of the integral equation solution component. This problem occurs because of the large discrepancy which exists between integral equation solutions and finite difference solutions

TABLE 1

Iteration count: different boundary points used to solve integral equation, 80x80 grid (smooth boundary), and stopping criterion $\frac{\|A\vec{x}-\vec{b}\|}{\|\vec{b}\|} < 10^{-10}$.

# of Boundary pts per object	Preconditioner	
	add relaxation	no relaxation
30	7	22
40	6	11
80	5	9
160	5	8
320	5	8

TABLE 2

Iteration count: 80 boundary points(per object) used to solve integral equation, different grids (smooth boundary), and stopping criterion $\frac{\|A\vec{x}-\vec{b}\|}{\|\vec{b}\|} < 10^{-10}$.

Grid	Iterative Method	
	simple iteration	FGMRES
20x20	5	6
40x40	5	6
80x80	5	6
160x160	6	7

for domains with corners. (The integral equation technique more rapidly captures the singularities of the solution). To remedy this, we fitted a periodic cubic spline to the boundary and passed this smoother boundary to the integral equation component. The results are presented in Table 3. With these adjustments, we see essentially the same behavior (few iterations and boundary points required) as for the smooth domain, and we conclude that integral equation preconditioning can be effective for domains with corners as well.

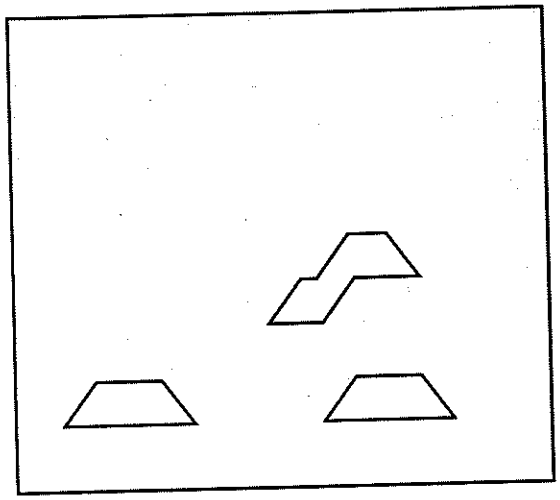

FIG. 8. *The cross section of three traces on an IC chip with depositing and undercutting.*

TABLE 3

Iteration count: different boundary points used to solve integral equation, 40x40 grid (boundary with several corners), and stopping criterion $\frac{\|A\vec{x}-\vec{b}\|}{\|\vec{b}\|} < 10^{-10}$.

# of Boundary pts per object	Preconditioner	
	add relaxation	no relaxation
15	13	INF
20	8	19
30	6	11
40	5	10
60	5	7
80	5	7

4. Conclusion. In this paper we've shown that integral equation solvers can be used as effective preconditioners for equations arising from spatial discretizations of Poisson's equation. In fact, they are so effective as preconditioners that simple iteration can be used; more sophisticated iterative procedures like GMRES [24] are not required. However, the difference in discretization procedures leads to large residuals near the boundaries; and we found that the addition of a relaxation step is an effective mechanism for alleviating this problem. Additionally, the use of a relaxation step allows one to coarsen the discretization of the integral equation without significantly increasing the number of iterations.

Another aspect of this paper is the use of a "cut-out" grid discretization. We've found that with the addition of an intermediate software layer which exploits polymorphism, the task of constructing the equations can be greatly simplified. Our construction method works particularly well with "cut-out" grid discretizations because only modest functionality of the intermediate software layer is required. The key primitive functions being a test if a point is inside or outside a given domain and the determination of the intersection point of a segment with an object boundary.

Both aspects of this paper have applications to other equations; in particular their use in the context of solving the incompressible Navier-Stokes equation is discussed in [12]. While we have concentrated on two dimensional problems, in principle, the ideas apply to three dimensional problems as well.

Acknowledgment. The authors would like to thank Dr. Anita Mayo for her generous assistance with the rapid integral equation evaluation techniques used in this paper.

A. Integral equation details:. The first step in constructing the solution of (18) is the formulation of an appropriate integral equation, and for this we use the results of [9, 19]. Given one bounding contour and M inner contours (where M>0), a solution is sought in the following form (here η is the outward pointing normal, as shown in Fig. 1):

$$u^{IE}(\boldsymbol{x}) = \int_{\partial\Omega} \phi(\boldsymbol{y}) \frac{\partial}{\partial\eta(\boldsymbol{y})} \left(\frac{1}{2\pi} \log \mid \boldsymbol{x} - \boldsymbol{y} \mid\right) ds(\boldsymbol{y}) + \sum_{k=1}^{M} A_k \log \mid \boldsymbol{x} - \boldsymbol{z_k} \mid . \tag{22}$$

We add M constraints to specify the M log coefficients:

$$\int_{\partial\Omega_k} \phi(\boldsymbol{y})\, ds(\boldsymbol{y}) \quad = 0, \quad k = 1, \ldots, M \quad . \tag{23}$$

Applying the boundary conditions leads to a uniquely solvable integral equation [19].

$$\begin{aligned} \frac{1}{2}\phi(\boldsymbol{x}_o) + \quad & \int_{\partial\Omega} \phi(\boldsymbol{y}) \tfrac{\partial}{\partial\eta(\boldsymbol{y})} \left(\tfrac{1}{2\pi} \log \mid \boldsymbol{x}_o - \boldsymbol{y} \mid\right) ds(\boldsymbol{y}) + \\ & \textstyle\sum_{k=1}^{M} A_k \log \mid \boldsymbol{x}_o - \boldsymbol{z_k} \mid = g^{IE}(\boldsymbol{x}_o) \\ & \textstyle\int_{\partial\Omega_k} \phi(\boldsymbol{y})\, ds(\boldsymbol{y}) = 0, \quad k = 1, \ldots, M \end{aligned} \tag{24}$$

The equations for the unbounded case (Fig. 2) are similar, see [9, 19] for details.

The integral equation is solved numerically using the Nyström method [11, 20] (in engineering terms, this amounts to a collocation approach where delta functions are used to represent the unknown charge density ϕ). We discretize this integral equation using the Trapezoidal rule (because of it's simplicity and spectral accuracy when used with closed smooth contours). If we sample n_k boundary points on the kth contour ($\{\boldsymbol{x}_i^k\}$, $i = 1, \ldots, n_k$), then the discretized integral can be written as a simple sum. (Here h_i represents the average arclength of the two boundary intervals that have $\boldsymbol{x}_i^k$ as an endpoint.)

$$\begin{aligned} &\int_{\partial\Omega_k} \phi(\boldsymbol{y}) \frac{\partial}{\partial\eta(\boldsymbol{y})}\left(\frac{1}{2\pi}\log \mid \boldsymbol{x}_o - \boldsymbol{y} \mid\right) ds(\boldsymbol{y}) \\ &\approx \sum_{i=1}^{n_k} \phi(\boldsymbol{x}_i^k) \frac{\partial}{\partial\eta(\boldsymbol{x}_i^k)}\left(\frac{1}{2\pi}\log \mid \boldsymbol{x}_o - \boldsymbol{x}_i^k \mid\right) h_i \end{aligned} \tag{25}$$

Next, we enforce the integral equation at each of the sampled boundary points and apply the quadrature rule. When the integration point coincides with the evaluation point ($\boldsymbol{x}_o = \boldsymbol{x}_i^k$) the kernel has a well defined limit. (Here $\kappa(\boldsymbol{x}_o)$ is the curvature of the contour at $\boldsymbol{x}_o$)

$$\lim_{\substack{\boldsymbol{x}\to\boldsymbol{x}_o \\ \boldsymbol{x}\in\partial\Omega_k}} \frac{\partial}{\partial\eta(\boldsymbol{x})}\left(\frac{1}{2\pi}\log \mid \boldsymbol{x}_o - \boldsymbol{x} \mid\right) = \frac{1}{4\pi}\kappa(\boldsymbol{x}_o) \tag{26}$$

These approximations reduce the integral equation (and constraints) to a finite dimensional matrix equation which can be solved for the log coefficients and the charge densities at the sampled boundary points.

$$\begin{pmatrix} \frac{1}{2}\boldsymbol{I} + \boldsymbol{D}_{cntr} & \boldsymbol{L}_{cntr} \\ \boldsymbol{D}_{con} & \boldsymbol{L}_{con} \end{pmatrix} \begin{bmatrix} \vec{\phi} \\ \vec{A} \end{bmatrix} = \begin{pmatrix} g^{\vec{IE}} \\ \vec{0} \end{pmatrix} \tag{27}$$

where

$$\vec{\phi} = \begin{bmatrix} \phi(\boldsymbol{x}_1^1) \\ \vdots \\ \phi(\boldsymbol{x}_{n_M}^M) \end{bmatrix}, \vec{A} = \begin{bmatrix} A_1 \\ \vdots \\ A_M \end{bmatrix}, g^{\vec{IE}} = \begin{bmatrix} g^{IE}(\boldsymbol{x}_1^1) \\ \vdots \\ g^{IE}(\boldsymbol{x}_{n_M}^M) \end{bmatrix}. \tag{28}$$

$\boldsymbol{D}_{cntr}$ represents the discrete contribution of the double layer potentials, $\boldsymbol{L}_{cntr}$ the effects of the log terms, $\boldsymbol{D}_{con}$ holds the discrete density constraints, $\boldsymbol{L}_{con}$ has the constraints on the log terms (a zero matrix for the case of a bounded domain), and $\boldsymbol{I}$ is the identity matrix.

The linear systems (27) associated with the integral equation correction are solved using Gaussian elimination. This direct matrix solver was employed for simplicity of development and because, for the test problems, the total time of the Gaussian elimination procedure was a small fraction of the total computing time. (Hence, increasing it's efficiency would have little impact). For problems with a large number of sub-domains the operation count of direct Gaussian elimination is highly unfavorable and procedures such as the Fast Multipole Method (FMM) [6, 9, 10, 21] should be used.

A.1. Evaluation of integral representation:. After solving for the charge densities and log coefficients, the function given by (22) must be evaluated at the nodes of a Cartesian "cut-out" grid. The simplest approach is to apply a quadrature method to (22) and evaluate the resulting finite sum; however, this procedure is computationally expensive since this sum must be evaluated for each interior grid point. One way of accelerating the evaluation process is to apply the FMM, which can be used to evaluate our integral representation at a collection of points in an asymptotically optimal way. However, because of the large asymptotic constant involved, using the FMM can still be fairly expensive. Therefore we choose to use a method [3, 14, 15, 16] that relies on a standard fast Poisson solver to do the bulk of the computations. As reported in [17], this approach is (in practice) faster than using the FMM.

The key idea in this method is to construct a discrete forcing function and discrete boundary conditions so that the solution of

$$\begin{aligned} \Delta_h u^{IE}(\boldsymbol{x}_{i,j}) &= f^d_{ij},\ \boldsymbol{x}_{i,j} \in R \\ u^{IE}(\boldsymbol{x}_{i,j}) &= g^d_{ij},\ \boldsymbol{x}_{i,j} \in \partial R \end{aligned} \tag{29}$$

provides the desired function values at the nodes of the rectangular Cartesian grid. (In our procedure we take R to be the rectangular domain used in the construction of the Cartesian "cut-out" grid.) Efficiency is obtained through the use of a Fast Poisson solver (e.g. we used HWSCRT from FISHPAK [1]) and the use of computationally inexpensive procedures to construct the requisite discrete boundary and forcing functions.

The boundary values, g^d_{ij}, are obtained by applying the trapezoid rule to (22). This is computationally acceptable because it is only done for those points that lie on ∂R. (Multipole expansions can be used to make this computation more efficient.)

For the construction of the forcing terms, f^d_{ij}, one notes that the Laplacian of the function (22) is identically zero (both log sources and double layer potentials are harmonic) away from the boundary, so the discrete Laplacian at points away from the boundary will be approximately zero. In particular, at the regular points a standard truncation error analysis yields the following result:

$$\begin{aligned} \Delta_h u^{IE} &= \Delta u^{IE} - \frac{h^2}{12}(u^{IE}_{xxxx} + u^{IE}_{yyyy}) \\ &= 0 - \frac{h^2}{12}(u^{IE}_{xxxx} + u^{IE}_{yyyy}). \end{aligned} \tag{30}$$

If the fourth derivatives of the function are bounded, zero is a second order approximation to the discrete Laplacian. The function (22) is the sum of a double layer potential and isolated log sources. Under rather mild assumptions concerning the contours and charge densities, double layer potentials have bounded fourth derivatives and so one is justified in approximating the contribution to the discrete Laplacian from that component by zero. Therefore, the double layer potential contributions to the discrete Laplacian only have to be calculated at the irregular grid points. The log terms do not have globally bounded fourth derivatives, and the calculation of their contribution requires separate treatment (which will be discussed below).

As discussed in [14], at irregular points the task of creating an accurate discrete Laplacian of a double layer potential requires accounting for jumps in the solution values which occur across the boundary of the domain. If the east, west, south, north, and center stencil points are denoted by $\boldsymbol{x}_e, \boldsymbol{x}_w, \boldsymbol{x}_s, \boldsymbol{x}_n$, and $\boldsymbol{x}_c$ respectively, we decompose the discrete Laplacian into four components.

$$\Delta_h u^{IE}(\boldsymbol{x}_c) = \frac{u^{IE}(\boldsymbol{x}_n) - u^{IE}(\boldsymbol{x}_c)}{h^2} + \frac{u^{IE}(\boldsymbol{x}_s) - u^{IE}(\boldsymbol{x}_c)}{h^2} + \frac{u^{IE}(\boldsymbol{x}_e) - u^{IE}(\boldsymbol{x}_c)}{h^2} + \frac{u^{IE}(\boldsymbol{x}_w) - u^{IE}(\boldsymbol{x}_c)}{h^2} \tag{31}$$

If none of the stencil arms intersect the boundary, then the standard error series analysis produces (30). At an irregular point, $\boldsymbol{x}_{i,j}$, the discrete Laplacian stencil will intersect the boundary along one or more of its stencil arms, and thus the exact Laplacian will not be an accurate approximation to the discrete Laplacian. In order to improve the approximation, a careful Taylor series analysis (one which accounts for the jump in solution values across the interface) is constructed to determine how to compensate for errors introduced by these jumps.

Specifically, when considering a stencil arm that intersects the boundary, we will refer to the two grid points that comprise the stencil arm as $\boldsymbol{x}_c$ and $\boldsymbol{x}_{nbr}$ (where $\boldsymbol{x}_c$ still refers to the center point, while $\boldsymbol{x}_{nbr}$ will represent any of the four remaining stencil points ($\boldsymbol{x}_e, \boldsymbol{x}_w, \boldsymbol{x}_s$, or $\boldsymbol{x}_n$)). ξ will denote the axis direction along $\boldsymbol{x}_c$ and $\boldsymbol{x}_{nbr}$ (i.e. x for horizontal stencil arms, or y for vertical arms), and $\boldsymbol{x}_*$ represents the boundary intersection point (see Fig. 9). We introduce the notation $[\ ]_o$ to represent the

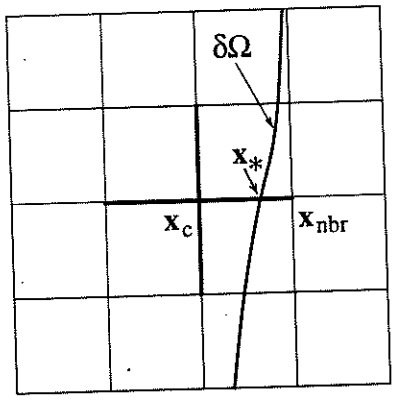

FIG. 9. *Generic description of a stencil arm intersecting the boundary. (here $\xi = x$ and $\boldsymbol{x}_{nbr} = \boldsymbol{x}_e$)*

jump across the boundary from the side containing $\boldsymbol{x}_{nbr}$ to the side containing $\boldsymbol{x}_c$ (i.e. $[u^{IE}(\boldsymbol{x}_*)]_o \equiv u^{IE}(\boldsymbol{x}_* + \epsilon(\boldsymbol{x}_{nbr} - \boldsymbol{x}_*)) - u^{IE}(\boldsymbol{x}_* - \epsilon(\boldsymbol{x}_{nbr} - \boldsymbol{x}_*)); \epsilon > 0, \epsilon \ll 1$).

With this notation, the contributions of the boundary intersection to the discrete Laplacian at $\boldsymbol{x}_c$ can be given as follows (the derivation follows from the procedure presented in [14]):

$$u^{IE}(\boldsymbol{x}_{nbr}) - u^{IE}(\boldsymbol{x}_c) = \tag{32}$$

$$[u^{IE}(\boldsymbol{x}_*)]_o + (\boldsymbol{x}_{nbr} - \boldsymbol{x}_*)[u^{IE}_\xi(\boldsymbol{x}_*)]_o + \frac{(\boldsymbol{x}_{nbr} - \boldsymbol{x}_*)^2}{2!}[u^{IE}_{\xi\xi}(\boldsymbol{x}_*)]_o +$$
$$(\boldsymbol{x}_{nbr} - \boldsymbol{x}_c)u^{IE}_\xi(\boldsymbol{x}_c) + \frac{(\boldsymbol{x}_{nbr} - \boldsymbol{x}_c)^2}{2!}u^{IE}_{\xi\xi}(\boldsymbol{x}_c) + O(\boldsymbol{x}_{nbr} - \boldsymbol{x}_c)^3.$$

This formula can be applied to all four stencil arms (with $\xi = x$ and $\boldsymbol{x}_{nbr} = \boldsymbol{x}_e$ or $\boldsymbol{x}_w$, or $\xi = y$ and $\boldsymbol{x}_{nbr} = \boldsymbol{x}_n$ or $\boldsymbol{x}_s$). When we substitute (32) into (31), we obtain a first order approximation to the discrete Laplacian given in terms of the jump values of the solution and the jumps in its first and second partial derivatives. For a double layer potential, these jump terms can be accurately computed directly from the charge density and its derivatives. Following the analysis presented in [14], we collect the needed jump equations. If we assume that the boundary is parameterized by a parameter s, then the boundary intersection point can be written as $\boldsymbol{x}_* = \boldsymbol{x}_*(s) = (x_*(s), y_*(s))$ and the charge density at that point as $\phi_* \equiv \phi(\boldsymbol{x}_*(s))$. Furthermore, we introduce another jump notation [ ] to represent the jump across the boundary from a point just outside the domain to a point just inside the domain (i.e. $[u^{IE}(\boldsymbol{x}_*)] \equiv u^{IE}(\boldsymbol{x}_*(s) + \epsilon\hat{\boldsymbol{\eta}}(s)) - u^{IE}(\boldsymbol{x}_*(s) - \epsilon\hat{\boldsymbol{\eta}}(s))$, where $\epsilon > 0, \epsilon \ll 1$, and $\hat{\boldsymbol{\eta}}$ points out of the domain). In this notation, the parameterized jump terms are given by the following formulas:

$$\begin{aligned}
[u^{IE}] &= -\phi_* \\
[u^{IE}_x] &= -\frac{\dot{x}_*\dot{\phi}_*}{\dot{x}_*^2 + \dot{y}_*^2} \\
[u^{IE}_y] &= -\frac{\dot{y}_*\dot{\phi}_*}{\dot{x}_*^2 + \dot{y}_*^2} \\
[u^{IE}_{xx}] &= -\frac{(\dot{x}_*^4 - \dot{y}_*^4)\ddot{\phi}_* + (\ddot{x}_*(-\dot{x}_*^3 + 3\dot{x}_*\dot{y}_*^2) + \ddot{y}_*(\dot{y}_*^3 - 3\dot{x}_*^2\dot{y}_*))\dot{\phi}_*}{(\dot{x}_*^2 + \dot{y}_*^2)^3} \\
[u^{IE}_{yy}] &= -[u^{IE}_{xx}].
\end{aligned} \tag{33}$$

In order to relate the [ ] jump definition (from exterior to interior) to the $[\,]_o$ notation (from the neighbor side to the center side), we check to see whether the center point is interior or exterior to the domain.

$$\begin{aligned}
\boldsymbol{x}_c \in \Omega &\Rightarrow [\,]_o = [\,] \\
\boldsymbol{x}_c \in \Omega^c &\Rightarrow [\,]_o = -[\,]
\end{aligned} \tag{34}$$

By using (32-34), we can approximately compute the discrete forcing terms at the irregular points without having to do any solution evaluations at all. This increases the speed of this approach since only local information is used (we avoid summing over all boundary points), and furthermore, this approach does not lose accuracy for grid points near the boundary (as direct summation approaches tend to).

In the computation of the discrete Laplacian of the function component associated with log terms, there are no boundary intersections to interfere with the Taylor series analysis, and no jump terms are needed. However, the derivatives of log sources are unbounded as you approach the source point, so zero is not an accurate approximation

to the discrete Laplacian for points near the log source. The discrete Laplacian is therefore explicitly computed for points which are within a radius of $d \propto h^{1/4}$ about the log source, and set to be zero outside of this radius. (For a point outside this radius, zero is a first order approximation to the discrete Laplacian)

Therefore, for both the log terms and the double layer potential, we can approximate the discrete Laplacian at all grid points by only doing some local calculations near the boundary and the log sources. Once the discrete forcing terms $f^d_{i,j}$ and the boundary values $g^d_{i,j}$ are known, a standard fast Poisson solver will rapidly produce the solution values at all of the Cartesian grid points. This approach produces a second order approximation to $u^{IE}(\boldsymbol{x}_{i,j})$.

REFERENCES

[1] J. Adams, P. Swarztrauber, and R. Sweet. Fishpak - a package of Fortran subprograms for the solution of separable elliptic partial differential equations. National Center for Atmospheric Research, Boulder Co., 1980.

[2] A. S. Almgren, J. B. Bell, P. Colella, and T. Marthaler. A cartesian grid projection method for the incompressible Euler equations in complex geometries. *SIAM J. Sci. Comput.*, (to appear).

[3] C. R. Anderson. A method of local corrections for computing the velocity field due to a distribution of vortex blobs. *J. Comput. Phys.*, 62:111–123, 1986.

[4] S. A. Bayyuk, K. G. Powell, and B. van Leer. An algorithm for the simulation of 2-D unsteady inviscid flows around arbitrarily moving and deforming bodies of arbitrary geometry. Technical Report 93-3391, AIAA, July 1993.

[5] M. J. Berger and R. J. LeVeque. An adaptive Cartesian mesh algorithm for the Euler equations in arbitrary geometries. Technical Report 89-1930, AIAA, 1989.

[6] J. Carrier, L. Greengard, and V. Rokhlin. A fast adaptive multipole algorithm for particle simulations. *SIAM. J. Sci. Stat. Comput.*, 9:669–686, 1988.

[7] W. J. Coirier and K. G. Powell. Solution-adaptive Cartesian cell approach for visous and invisid flows. *AIAA J.*, 34:938–945, 1996.

[8] L. Collatz. Berkungen zur fehlerabschätzung für das differenzenverfahren bein partiellen differentialgleichungen. *Z. Agnew. Math. Mech.*, 13:56–57;576,612,960, 1933.

[9] A. Greenbaum, L. Greengard, and G. B. McFadden. Laplace's equation and the Dirichlet-Neumann map in multiply connected domains. *J. Comput. Phys.*, 105:267–278, 1993.

[10] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *J. Comput. Phys.*, 73:325–348, 1987.

[11] R. Kress. *Linear Integral Equations.* Springer-Verlag, Berlin, 1989.

[12] A. Li. Incompressible navier-stokes flow about multiple moving bodies. Technical Report CAM 96-53, UCLA, 1996.

[13] A. Mayo. Personal communication.

[14] A. Mayo. The fast solution of Poisson's and the biharmonic equations on irregular regions. *SIAM J. Numer. Anal.*, 21:285–299, 1984.

[15] A. Mayo. Fast high order accurate solution of Laplace's equation on irregular regions. *SIAM J. Sci. Stat. Comput.*, 6:144–156, 1985.

[16] A. Mayo. The rapid evaluation of volume integrals of potential theory on general regions. *J. Comput. Phys.*, 100:236–245, 1992.

[17] A. McKenney, L. Greengard, and A. Mayo. A fast Poisson solver for complex geometries. *J. Comput. Phys.*, 118:348–355, 1995.

[18] J. E. Melton, M. J. Berger, M. J. Aftosmis, and M. D. Wong. 3D applications of a Cartesian grid Euler method. Technical Report 95-0853, AIAA, 1995.

[19] S. G. Mikhlin. *Integral Equations.* Pergammon Press, London, 1957.

[20] E. J. Nyström. Über die praktische auflösung von integralgleichungen mit anwendungen auf randwertaufgaben. *Acta. Math.*, 54:185–204, 1930.

[21] V. Rokhlin. Rapid solution of integral equations of classical potential theory. *J. Comput. Phys.*, 60:187–207, 1985.

[22] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modeling and Design.* Prentice Hall, Englewood Cliffs, 1991.

[23] Y. Saad. A flexible inner-outer preconditioned GMRES algorithm. *SIAM. J. Sci. Comput.*, 14:461–469, 1993.

[24] Y. Saad and M. H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM. J. Sci. Stat. Comput.*, 7:856–869, 1986.

[25] E. Y. Tau. A second-order projection method for the incompressible Navier-Stokes equations in arbitrary domains. *J. Comput. Phys.*, 115:147–152, 1994.

[26] D. M. Young and R. T. Gregory. *A Survey of Numerical Mathematics, Vol 2.* Addison-Wesley Publishing Company, Inc., Philippines, 1973.

[27] D. Zeeuw and K. G. Powell. An adaptively refined Cartesian mesh solver for the Euler equations. *J. Comput. Phys.*, 104:56–68, 1993.