MODELLING OF THREE-DIMENSIONAL TRANSIENT CONJUGATE CONVECTION-CONDUCTION-RADIATION HEAT TRANSFER PROCESSES AND TURBULENCE IN BUILDING SPACES

STEPHEN EDWARD POTTER

A thesis submitted in partial fulfilment
of the requirements of the University of
Northumbria at Newcastle for the
degree of Doctor of Philosophy

August 1998

ABSTRACT

A survey of the developments in the field of Computational Fluid Dynamics (CFD) is presented, the results of which are used to identify numerical methods capable of solving the equation sets that define the various categories of fluid flow and heat transfer that apply to air movement within buildings. The background to turbulence modelling is discussed together with the treatment of near-wall regions to which turbulence models are inapplicable. A further survey into the application of CFD methods to air movement within buildings is presented together with an appraisal of the success of these studies in terms of realistic modelling. From this survey it is concluded that there is a need to integrate surface radiation heat transfer methods within CFD procedures in order to provide a fully coupled model.

The equation set describing advection, convection and conduction processes together with the k-ε turbulence model are presented and the development of this equation set into the final mathematical model described. Details of the numerical procedure adopted for solution of the equation set are provided together with a general approach to the incorporation of radiation heat transfer within the same solution scheme.

Shortwave and longwave radiation heat transfer processes in buildings are discussed and the geometric requirements for the numerical simulation of radiation process identified. A general numerical method for handling room geometry is presented together with a method for linking building surface and CFD grid geometries. A method for incorporating shortwave solar radiation together with an approximate method for longwave radiation within the CFD solution scheme is detailed, dispensing with the need for an involved iterative approach.

A computer program has been developed from these mathematical models which is capable of solving coupled three-dimensional convection, conduction and radiation heat transfer processes. The program has been applied to a set of test cases for which data sets have been provided by the International Energy Agency. The results of simulation are compared with the measured data.

ACKNOWLEDGEMENTS

I would like to thank Dr. C. P. Underwood of the Department of the Built Environment, University of Northumbria at Newcastle for his continuing practical and intellectual support which was instrumental in the completion of this study.

I would also like to thank Annex 20 of the International Energy Agency for providing measured data.

Finally, thanks to my family, Lois, Liz and Tom for putting up with periods of my absence during the course of the study and preparation of the thesis.

CONTENTS

NOMENCLATURE

A	area of finite volume interface, TDMA coefficient, plane equation coefficient and surface area
a	coefficient of finite-difference equation and solar altitude
B	TDMA coefficient and plane equation coefficient
b	finite difference equation term containing grid point variable independent elements of source term
C	plane equation coefficient
C_P	specific heat at constant pressure
C_1, C_2, C_3, C_D, C_μ	constants of turbulence model
D	conductance and plane equation constant
E	Log-law constant (= 9.793)
F	flow rate and radiation view factor
f	interpolation factor
f_i	friction vector
G	irradiation
G_B	non-dimensional buoyancy parameter
G_K	generation of turbulence energy
g	acceleration due to gravity (= 9.81 m / s^2)
g_i	gravitational vector (0, -g, 0)
I	radiation intensity
J	radiosity
k	turbulence kinetic energy (= ($u'^2 + v'^2 + w'^2$) / 2)
L	characteristic width of calculation domain and length scale
l	mixing length and length scale
P	pressure and shading factor
P_e	Peclet number (= ρ u L / Γ)
Q	shortwave radiative flux
q	longwave radiative flux
q"	heat flux across a surface
r	radius and rotation matrix element
R	universal gas constant
Re	Reynolds number (= U_0 L / ν)
Re_T	turbulence Reynolds number (= k^2 / ν ε)

R_f	flux Richardson number (= - G_B / G_K)
S	general source term
S_c	grid point variable independent element of source term
S_p	grid point variable dependent element of source term
t	translation matrix element
T	temperature
T^*	non-dimensional heat flux temperature
U, V, W	mean velocity components in x,y and z directions
u', v', w'	fluctuating velocity components in x, y, z directions
$\overline{U}_i$	mean velocity components in Cartesian tensor notation
U^*	non-dimensional velocity in wall region (= U / U_τ)
$\hat{U}_i$	pseudo-velocity components in Cartesian tensor notation
U_j^*	guessed velocity components in Cartesian tensor notation
U_τ	friction velocity (= $\sqrt{(\tau_w / \rho)}$)
u_i'	fluctuating velocity components in Cartesian tensor notation
v	specific volume
$-\rho\ \overline{u_i' u_i'}$	reynolds stresses
$-\rho\ \overline{u_i T'}$	turbulent heat fluxes
y^*	non-dimensional distance from wall (= $U_\tau y / \nu$)
x, y, z	directions of co-ordinate axes and Cartesian coordinates
z	solar azimuth
α	thermal diffusion coefficient, relaxation factor, azimuth angle and absorptivity
β	coefficient of cubical expansion and elevation angle
Γ	diffusion coefficient
ΔT	reference temperature difference (= T - T_r)
δ_{ij}	Kronecker delta (= 1 if i = j; otherwise = 0)
$\hat{V}$	velocity scale
ε	dissipation rate of turbulence energy (= $\nu\ \overline{\frac{\partial u_i'}{\partial x_j}\frac{\partial u_i'}{\partial x_j}}$ at high Reynolds No.) and emissivity
ϕ	rotation angle and spherical coordinate angle
κ	Von Karman constant (= 0.4187)
μ	viscosity
ν	kinematic vicosity
ρ	density and reflectivity

θ	rotation angle
$\sum_{j} a_{nb}$	summation of the six neighbouring grid point coefficients
Π	total pressure
σ	Prandtl number
τ	shear stress and transmissivity
ψ	spherical coordinate angle
$d\Omega_n$	elemental solid angle

Subscripts

eff	effective value
max	maximum value
P	value relating to grid node P, point in Cartesian coordinates
r	reference value
t	turbulent value
w	wall value
ϕ	dependent variable
nb	neighbouring coefficients

Superscripts

'	fluctuating component
*	non-dimensional value and value at previous iteration

1.0 INTRODUCTION

1.1 Historical Developments - Building Energy Simulation

Clarke [1] suggests that the evolution of building energy simulation may be perceived in the form of a series of model generations.

a) First Generation

Traditionally, building designers have relied on a series of discrete piecemeal calculations in an attempt to quantify building performance. The individual calculations are analytical and normally involve various simplifying assumptions in order to permit formulation. For example, simple heat loss calculation methods assume one dimensional steady heat flow through building elements, a situation which rarely, if ever, occurs in the real world.

This approach does not attempt to simulate energy and mass flow paths that would occur in real buildings but rather provides the designer with an indication of building performance under peak conditions. The methods are therefore satisfactory for the estimate of peak building loads for the sizing of HVAC (Heating, Ventilating and Air Conditioning) systems but must be considered to provide doubtful accuracy in the assessment of part load building energy performance or the prediction of energy consumption.

b) Second Generation

In the wake of the 1973 oil embargo, initiative was provided to investigate the temporal aspects of energy flow, particularly in the case of long time constant elements (i..e. thermally massive elements).

The methods employed were still analytical and piecemeal in nature. Such methods included the convolution or response factor method, Mackey and Wright, 1944 [2]; Stephenson and Mitalas, 1967 [3] and the thermal admittance procedure, Millbank and Harrington-Lynn, 1974 [4].

Resulting model integrity remained relatively low due to inherent decoupling and the simplifying assumptions applied to the flow paths.

c) Third Generation

As previously indicated, one of the major problems associated with modelling energy flow paths in buildings is the severe limitation imposed on the modelling method through the adoption of simplifying assumptions, which are necessitated in order to force a solution. Simplifying assumptions applied to the partial differential equations describing heat and mass transfer normally include equation linearity and uni-dimensionality, such assumptions strictly being quite unrealistic. In very recent times, the increasing accessibility of relatively powerful computers has enabled the application of advanced numerical methods to the solution of the fundamental defining equation sets for heat and mass transfer in buildings. Hence, severe equation non-linearity and multi-dimensionality could, in principle, be managed.

Using such methods, the building system is considered as a field problem, in which the only true independent variables are the space dimensions and time. All the other quantities (flux exchanges and state variables) are fully coupled and entirely dependent temporally and space-dimensionally throughout the building system. Thus model integrity can be raised substantially so that predictions simulate reality much more closely.

Although the current generation of building energy models incorporate quite advanced treatments for coupled transient conduction and radiation heat transfer processes, advection and convection processes are still treated in a relatively simplistic fashion using empirical relationships.

1.2 Computational Fluid Dynamics and the Prediction of Air Flows in Buildings

A group of numerical methods capable of solving the partial differential equation set that describes three-dimensional laminar and turbulent convection processes has been made available through the field of Computational Fluid Dynamics (CFD). Over the last twenty years, several researchers have applied these methods to air flows in building spaces, some of these studies being described in Chapter 2. Whilst the results of these studies have indicated the considerable potential for CFD methods when applied to buildings, the computations involved assume that the thermal boundary conditions are known in advance.

However, for most practical building spaces, the field problem comprises a conjugate heat transfer process whereby transient conduction through the confining solid elements (floors, walls, partitions, roof, windows, etc.), convection-diffusion in the confined air volumes and shortwave/longwave inter-surface radiation exchange between windows, walls, etc. must all be considered with correct

matching at the fluid-solid interfaces.

1.3 Primary Aims of the Study

The existing state-of-the-art dynamic building thermal models such as ESP [5], BLAST [6] and TARP [7] are capable of modelling thermal boundary conditions to a high degree of accuracy. However, quite severe approximations are normally made in the calculation of convective heat transfer, simplified empirical convection models generally being employed. Conversely, the existing CFD codes which have been applied to building problems generally assume that the thermal boundary conditions may be prescribed in advance.

It is the purpose of this study to:-

a) Demonstrate the requirement and scope for a coupled convection-conduction-radiation model.

b) Identify a potentially accurate method of predicting surface radiation heat transfer processes in buildings which may be incorporated directly within a CFD solution scheme, without the requirement for an involved iterative approach.

c) Develop a fully coupled model capable of predicting turbulent convection, conduction and radiation heat transfer processes in single or multi-zone building spaces.

d) Compare the coupled model predictions with measured data, provided by the International Energy Agency Annex 20.

1.4 Outline of the Thesis

Chapter 2 contains a review of developments in the field of Computational Fluid Dynamics, culminating in general numerical methods capable of solving the equations defining convection and conduction heat transfer processes. A review of turbulence modelling methods is also presented. Chapter 2 concludes with a discussion of the application of CFD to building air flows and building thermal modelling. The mathematical basis of the convection-conduction model is presented in Chapter 3 together with details of the numerical method employed to solve the defining equation set, this chapter also includes a detailed description of the k-ε turbulence model. Chapter 4 reviews shortwave and longwave radiation modelling methodology and details the algorithms employed in

this study for the calculation of radiative heat exchange. A method for incorporating the solution for inter-surface longwave radiation exchange within the same solution scheme developed in Chapter 3 for convection-diffusion processes is also presented. The resulting computer model predictions are then compared with the measured results, provided by Annex 20 of the International Energy Agency, in Chapter 5. Overall conclusions and recommendations for future work are provided in Chapter 6.

2.0 LITERATURE REVIEW

2.1 Computational Fluid Dynamics

Roache [8] suggests that the term Computational Fluid Dynamics (CFD) has arisen in order to distinguish a specific set of numerical solution procedures from the entire spectrum of general numerical methods applied to problems of fluid dynamics. These procedures generally involve the discretisation of the defining partial differential equations which describe the behaviour of various categories of fluid flow (i.e. conservation equations of mass, momentum and energy), through the sub-division of the flow domain into finite volumes, a numerical scheme is then employed to adjust the values of the required variables at each grid point until the various discretised conservation equations are satisfied.

Although many significant contributions were made towards the evolution of CFD from the early part of the century onwards, such as the numerical solution of the partial differential equations for a viscous fluid dynamics problem by Thom in 1933 and Allen and Southwall's residual relaxation solution to the problem of viscous incompressible flow over cylinders in 1955 [5], research interests were first directed in earnest towards the development of such numerical methods during the mid-1960's when digital computers were beginning to appear with adequate processing capabilities to afford the very significant computational effort required.

The majority of the pioneering work was conducted separately by the Mechanical Engineering Department of Imperial College, London (UK) and the Los Alamos Laboratories, New Mexico (US). The early methods employed by the Imperial College group used the stream function-vorticity procedure in which the pressure and velocity terms which appear in the defining equation set are displaced by the two quantities, stream function and vorticity. The method was brought into existence in 1966 by Runchal and Wolfshtein, using a relatively simple turbulence model, an account of the method is detailed in the book by Gosman, Pun, Runchal, Spalding and Wolfshtein [9].

One of the most significant features of the method developed by Runchal and Wolfshtein is the prescribed treatment for the convection terms. Previously, the numerical solution to the equations of fluid dynamics were seen to become quite unstable at high absolute values of Peclet No. ($P_e = \rho u L/\Gamma$), where the flow becomes dominated by convection, the reason for this being the highly non-linear relationship between the transported variable and the transport distance. The conventional finite difference approach employed by preliminary formulations makes the assumption that the convected property at the finite volume grid interface is the average of the downwind and upwind grid point values. The remedy proposed by Runchal and Wolfshtein is to leave the diffusion

terms unchanged but to allow the interface values of the convected properties to take on the upwind grid point values. This scheme is known as the 'upwind scheme' and was originally proposed independently by Courant Isaacson and Rees in 1952 [10]. Various refinements to the way in which the convective terms are handled have been proposed since the upwind scheme came into existence, these include the 'exact' or 'exponential scheme' after Spalding [11], the 'hybrid scheme' again after Spalding[11] and the 'power-law scheme' after Patankar [12].

The stream function-vorticity method was widely used by researchers from the mid-1960's through to the mid-1970's. However, the method was primarily developed for the investigation of two-dimensional elliptical flow phenomena and was not readily extensible to three-dimensional elliptical flows or flows with pressure dependent densities and thus incentive was provided to develop a computational procedure with wider applicability than the stream function-vorticity method.

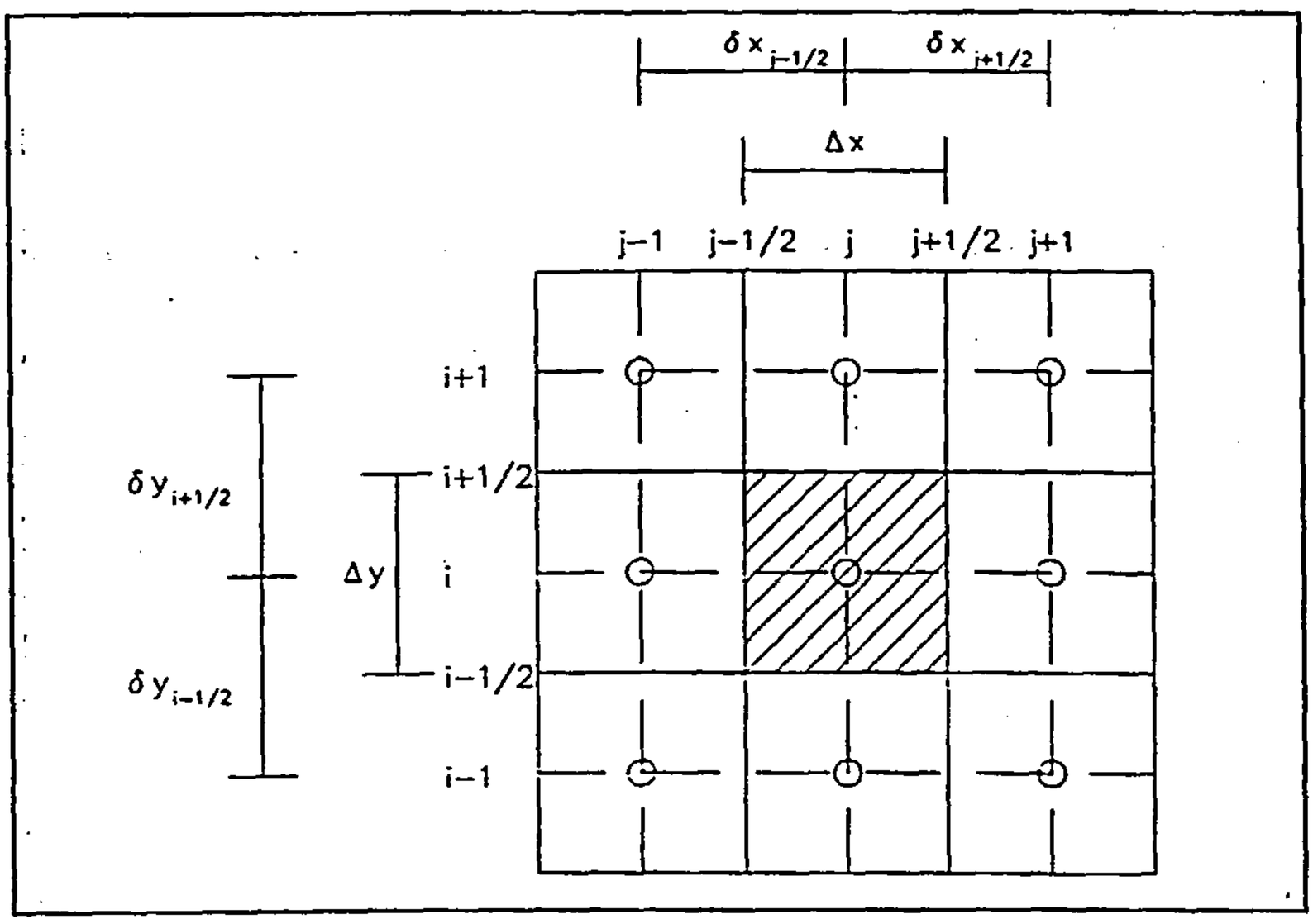


Figure 1 - representation of pressure differentials in the finite volume grid

One of the methods to be most widely used, emerging from the Los Alamos research, concerned an explicit solution to the 'primitive variable' form of the defining equation set, the so-called MAC (Marker-And-Cell) method, developed by Harlow and Welch in 1965 and described by Roache [8].

The 'primitive variable' approach introduces a significant problem to the solution scheme. The finite volume form of the integrated Navier-Stokes equations must contain a representation of the pressure term ($-\partial P/\partial x_i$) integrated over the control volume. With reference to Figure 1, the resulting contribution to the discretised momentum equation may be seen to be $P_{j-1/2} - P_{j+1/2}$, which is the net pressure force exerted on the control volume of unit cross-sectional area. In order to represent

this difference in terms of grid point pressures, a piecewise-linear profile may be adopted for pressure. If the finite volume grid interfaces are chosen to lie midway between the respective grid points, the pressure difference becomes half of the difference between the adjacent grid point pressures. The implication of this is that the momentum equation contains pressure differences between alternative grid points rather than adjacent ones. Besides the problem that the pressure field is being extracted from a coarser grid than the other variables and the subsequent loss in accuracy, there is a much more profound problem in that a quite unrealistic 'zig-zag' pressure field could arise that would behave as a uniform pressure field to the momentum equations. The remedy proposed by Harlow and Welch is to use a secondary finite volume grid for the calculation and storage of the velocity components. In such a displaced or 'staggered' grid, the velocity components are calculated for the points that lie on the interfaces of the finite volume grid cells.

In order to obtain a method that enjoyed the applicability of the 'primitive variable' approach, coupled with the computational economy of the 'implicit' approach adopted by the stream function-vorticity procedure, effort was directed by Imperial College towards the development of a method that combined the two approaches.

The first procedure to appear was the SIVA (SImultaneous-Variable-Adjustment) method. However, an improved method appeared very shortly afterwards and was indeed presented in the same publication as SIVA [13]. This improved method is known as the SIMPLE (Semi-Implicit-Method-for- Pressure-Linked-Equations) procedure and forms the basis of many of the codes in use at present in the UK, such as TEACH and CHAMPION [14]. The SIMPLE procedure has been refined somewhat in order to provide a less arbitrary method of arriving at the so-called 'guessed' pressure field, notably the SIMPLER procedure after Patankar [15] and the SIMPLEST procedure after Spalding [16]. It is the SIMPLER procedure that forms the basis of the numerical procedure described in this section.

2.2 Turbulence Modelling

Turbulence in fluid motion is characterised by fluctuating rotational eddies, the size and frequency of which vary considerably throughout the flow. The large eddies are of the same magnitude as the flow domain and are determined by the flow boundaries, the smaller high frequency eddies being determined by viscous forces. It is the large eddies that extract kinetic energy from the mean flow and pass this energy by vortex interaction (self-stretching of the vortices) between successively smaller eddies, until the energy is dissipated as heat through molecular viscous action at the smallest eddy sizes, this process being known as energy cascade, see Tennekes and Lumley[17]. Additionally,

where buoyancy forces are present, there may be an exchange between mean flow potential energy and turbulent kinetic energy.

Because the large eddy motion interacts with the mean flow motion, it is highly anisotropic. However, at high Reynolds numbers, where the large scale eddies and small scale eddies are spectrally widely separated, the small scale motion loses directional dependence and becomes almost isotropic, this phenomonon is known as local isotropy and becomes highly significant when making turbulence modelling assumptions.

The defining partial differential equations of fluid dynamics, described fully in Section 3.1 are theoretically applicable to both laminar and turbulent flows, assuming that the fluid is Newtonian. In order to accurately represent turbulent flows, the numerical finite volume grid size must be at least comparable in magnitude to the smallest eddy size, this may be estimated from dimensional analysis to be in the order of the Kolmogorov microscale ($(\nu^3/\varepsilon)^4$), typically 10^{-2} - 10^{-3} m, see Tennekes and Lumley [17]. A whole flow domain may extend over many metres when considering air flows in building spaces and thus the numerical finite volume grid would require such a vast number of grid points so as to extend far beyond the capabilities of current digital computers. Fortunately, interest in building air flows is concerned with the instantaneous mean quantities rather than the turbulent structure itself and thus a turbulence model approach may be adopted.

The theory behind a turbulence model is to provide a set of differential equations, associated algebraic equations and constants, which when solved simultaneously with the continuity, Navier-Stokes and energy equations closely simulate the behaviour of real turbulent fluids.

2.2.1 Eddy Viscosity/Diffusivity Models

In turbulent flows, the instantaneous velocity (U) may be considered to comprise a mean component ($\overline{U}$) and a fluctuating component (U'), $U = \overline{U} + U'$. If the velocity components in the partial differential equation set are replaced with this instantaneous velocity relationship, additional terms arise that represent turbulent stresses (so-called Reynolds' stresses) and turbulent heat flux, taking the form $-\rho\,\overline{u'v'}$ and , $-\rho\,\overline{v'T'}$ respectively.

The first major contribution towards the development of turbulence models was by Boussinesq in 1877 [15], who introduced the eddy viscosity concept (turbulent shear stress, $\tau_t = -\rho\,\overline{u'v'} = \mu_t \frac{\partial u}{\partial y}$, where μ_t is the turbulent or eddy viscosity).

Despite the identification of conceptual inconsistencies to the eddy viscosity concept (Rodi [19]), it has been found to work well in practice. This is mainly due to the fact that the eddy viscosity as defined can be determined to a good approximation in many flow situations. Rodi emphasises the proportionality of eddy viscosity to a velocity scale ($\hat{V}$) and a length scale (L) characterising the (large- scale) turbulent motion:-

$$\mu_t \alpha \hat{V}L \qquad \text{Eq.2.1}$$

Using the definition of eddy viscosity, the laminar diffusion coefficient appearing in the defining transport equations may be replaced with an effective diffusion coefficient, being the sum of the laminar and turbulent coefficients.

This leaves the problem of closure, that is establishing a relationship that enables the calculation of the eddy viscosity and provides a closed set of equations.

2.2.1.1 Constant Effective Viscosity Models

Although not strictly correct, the use of a constant effective viscosity/diffusivity is sometimes referred to as a turbulence model. Because local turbulence is not modelled, this approach can only be justified where the flow is not significantly influenced by local turbulence effects. However, the constant effective viscosity approach has been applied with some success, e.g. Whittle [20].

2.2.1.2 Zero-Equation Models

The simplest turbulence models are based on the so-called 'mixing length' hypothesis of Prandtl [21] which allows the eddy viscosity μ_t, to be calculated from the equation:-

$$\mu_t = \rho l^2 \left|\frac{\partial u}{\partial y}\right| \qquad \text{Eq.2.2}$$

where l is the 'mixing length', ρ is the local fluid density, u is the velocity in the main stream direction and y is the distance normal to that direction. However, this proposal to some extent replaces the question of μ_t with that of l, although l varies much less throughout the flow domain than μ_t.

However, models that relate μ_t directly to the mean velocity distribution must assume that there is no transport of turbulence either spatially or temporally and that turbulence is first generated and then

dissipated locally. Thus, for example in building air flow problems, where turbulence is generated by a diffuser and transported by the mean motion to a downstream location, if the mean velocity is uniform at that location, mixing length models would indicate zero turbulence. Additionally, the mixing length hypothesis is difficult to apply to recirculating flows where there is no single direction of dominant velocity gradient. Despite these drawbacks, this method has been widely used by researchers. One application of the method used in conjunction with the stream function-vorticity method is presented by Gosman, Pun et al. [9].

2.2.1.3 One-Equation Transport Models

The problem of lack of universality associated with the mixing length hypothesis was to some extent alleviated with the introduction in 1945 of the Prandtl turbulence-energy model [22]. This model is based on the supposition that the eddy viscosity may be characterised by $\sqrt{k}$ (where k is the kinetic energy of turbulent motion per unit mass = $(\overline{u'^2} + \overline{v'^2} + \overline{w'^2})/2$) and the turbulence length scale l:-

$$\mu_t = C_\mu \rho k^{1/2} l \qquad \text{Eq.2.3}$$

where C_μ is a constant, k being derived from a transport equation:-

$$\frac{Dk}{Dt} = \text{div}(\Gamma \text{ grad } k) + \mu_t \left\{ \left(\frac{\partial u}{\partial y}\right)^2 + \ldots\ldots \right\} - C_D \frac{k^{3/2}}{l} \rho \qquad \text{Eq.2.4}$$

C_D is a constant and l must be a prescribed function, e.g.:-

$$\frac{l}{y} = \lambda f\left(\frac{y}{y_t}\right) \qquad \text{Eq.2.5}$$

see Spalding [23]. The merit of this approach is the ability to model local turbulence effects through the convection and diffusion of k, although there is still the problem of prescribing l. Equation 2.3 is sometimes known as the Kolmogorov-Prandtl expression because it was independently introduced by Kolmogorov in 1942 [24].

2.2.1.4 Two-Equation Transport Models

The Prandtl one-equation model was largely superseded by the emerging family of two equation models, where the k-equation is retained but the prescription for the length scale l is replaced with a

second turbulence quantity and an associated transport equation. The length scale equation taking the following form:-

$$Z = k^m l^n \qquad \text{Eq. 2.6}$$

The predominant two-equation models are the k-kl model described by Rodi and Spalding [25], the k-W model described by Spalding [23] and finally the k-ε model which was originally proposed by Harlow and Nakayama in 1968 [27] and subsequently modified by Launder and Spalding in 1973 [28].

The results of the models are similar and all exhibit the same form:-

$$\frac{\partial Z}{\partial t} + u_i \frac{\partial Z}{\partial x_i} = \frac{\partial}{\partial x_j}\left(\frac{\sqrt{k}L}{\sigma_z}\frac{\partial Z}{\partial x_i}\right) + C_{z1}\frac{Z}{k}P - C_{z2}Z\frac{\sqrt{k}}{L} + S \qquad \text{Eq.2.7}$$

where

P = production by shear = $-\overline{u_i' u_j'}\frac{\partial u_i}{\partial x_j}$

S = length scale variable dependent source term

and σ_z, C_{z1} and C_{z2} are empirical constants

A correction to the source term S is usually required for near-wall situations, however experience (Rodi [19]) has shown that this is not necessary for the ε equation and for this reason, the k-ε model has become the most generally used model.

2.2.1.5 Reynolds Stress Models

A central assumption made in the development of the one-equation and two-equation eddy viscosity models is that of an isotropic viscosity. In some cases, this assumption may be too crude; for example in recirculating flows with a high degree of swirl. This possibility has motivated the development of models in which the transport of constituent Reynolds stresses and scalar fluxes ($-\overline{u_i' u_j'}$ and $-\overline{u_i' \phi}$) are modelled separately. An exact equation for $-\overline{u_i' u_j'}$ first being derived by Chou [29].

However, for general flows, the Reynolds stress model requires the solution of 10 partial differential

equations (6 components of Reynolds stress, 3 components of scalar flux and one equation for scalar fluctuations). The transport equations have therefore been simplified by dependent variable gradient approximations in order to replace differential equations with algebraic equations and thus reduce computational demands. Rodi [30] has assumed that the transport of - $\overline{u_i' u_j'}$ is proportional to the transport of k, the resulting non-constant proportionality factor being - $\overline{u_i' u_j'}$ /k. This approach is known as algebraic stress modelling

A state-of-the-art review of Reynolds stress and algebraic stress models is presented by Launder [31]. Although showing significant potential for the future, these models are as yet largely untested and computationally very expensive and have consequently not yet achieved practical application.

2.2.2 Subgrid Scale Modelling or Large Eddy Simulation (LES)

A quite different approach to turbulence modelling has been adopted by Deardroff [32], whereby the spectral separation of large scale and small scale turbulent motion, exhibited at high Reynolds numbers, is exploited. The large scale motion, which is comparable in magnitude to the flow domain is resolved explicitly, whereas the isotropic small scale motion which cannot be resolved using the same grid size is modelled separately.

However, the limited research to date, indicates that the method, while being theoretically promising is computationally extremely expensive, and has revealed no significant advantage over the considerably more computationally efficient two-equation models.

2.2.3 Summary of Turbulence Models

Although the constant effective viscosity model has been applied with some success [20], the inherent inability to model local turbulence and turbulence transport indicates the possibility of a severe lack of generality to this approach.

The Prandtl mixing-length hypothesis [21] is however capable of local turbulence modelling and has been applied very successfully to a wide range of problems. However, it is unable to model the transport of turbulence and requires the specification of l, which in some cases can prove very difficult indeed.

The introduction of one-equation models such as the Kolmogorov-Prandtl model [22] has enabled the transport of turbulence to be taken into account, however these models still require the

specification of a length scale and thus suffer from the same deficiency as the mixing-length models.

Whilst the LES and Reynolds/algebraic stress methods both show considerable potential for the future they are as yet largely untested and computationally very expensive, whereas the k-ε two-equation model has been widely tested and shown to be reliable and computationally efficient under a wide range of flow conditions (see Launder and Spalding [28]). It is therefore the k-ε model that has been selected as the preferred turbulence model for this study.

2.2.4 Turbulence Modelling and the Near-Wall Region

The k-ε turbulence model is only valid for regions of fully turbulent flow and thus an alternative treatment is required to model fluxes of momentum and heat in near wall regions where the local Reynolds' number of turbulence is so small that the molecular viscous effects dominate over turbulent ones.

The two methods of accounting for these regions in turbulence modelling are firstly the employment of wall functions and secondly the use of low-Reynolds-number turbulence models.

Low Reynolds number modelling requires a high resolution finite volume grid to be associated with the near-wall regions and the adoption of an alternative pair of equations for k and ε which enable molecular viscosities to exert influence as a solid boundary is approached. The wall function method is normally preferred for reasons of computational economy. The usual approach to the wall function method is to assume that the thin fluid layer adjacent to the solid boundary is a zone of local equilibrium where the shear stress and heat flux in that region are uniform and the universal logarithmic law of the wall applies, see Schlichting [34].

This results in the following velocity profile:-

$$\frac{U_p}{U_\tau} = \frac{1}{\kappa} \ln (E\, y_p^*) \qquad \text{Eq.2.8}$$

where U_τ = friction velocity = $\sqrt{(\tau_w / \rho)}$

the log-law constants κ and E are determined from experiment.

When uniform shear stress prevails in the near-wall region and the generation and dissipation of energy are in balance, Ideriah [35] has shown that the following relationship prevails:-

$$\tau_w = \tau_p = \rho C_\mu^{1/2} k \qquad \text{Eq.2.9}$$

thus:-

$$\tau_p = \rho \kappa C_\mu^{1/4} k_p^{1/2} U_p / \ln (E y_p^*) \qquad \text{Eq.2.10}$$

The dissipation rate ε_p near the wall is fixed by the requirement that the length scale varies linearly with distance from the wall, thus:-

$$\varepsilon_p = C_\mu^{3/4} k_p^{1/2} / \kappa y_p \qquad \text{Eq.2.11}$$

Near-wall turbulence energy k_p, is obtained from the direct solution of the k-equation (ref. Section 3.3), diffusion of energy to the wall being set to zero, in the absence of better information (see Launder and Spalding [28]).

A similar analysis for heat flux yields a similar relationship (see Spalding [23]):-

$$\rho U_\tau C_p (T_w - T) / \dot{q}_w^* = \sigma_t (U_p^* + P\{\sigma / \sigma_t\}) \qquad \text{Eq.2.12}$$

where $P\{\sigma / \sigma_t\}$ is a function of the ratio of laminar and turbulent Prandtl numbers, such as that based on the work of Jayatillaka [36] :-

$$P\{\sigma / \sigma_t\} = 9.24 \{ (\sigma / \sigma_t)^{3/4} - 1\} \{1 + 0.28 \exp (-0.007 \sigma / \sigma_t)\} \qquad \text{Eq.2.13}$$

2.3 Comparison of CFD Predictions with Measured Results for Building Air Flows

One of the first attempts to model air flows in a building space using a two-equation model of turbulence was made by Nielsen, 1974 [37] who applied a variation of the Runchal Wolfshtein stream function-vorticity procedure to a two-dimensional representation of a mechanically ventilated room, the Launder and Spalding k-ε model of turbulence was employed with no modification to include buoyancy generation and destruction terms. Very good agreement between predicted and experimental results was achieved.

Zohrabian, Mokhtarzadeh-Dehghan and Reynolds, 1988 [38] applied the SIMPLE algorithm to two-dimensional buoyancy driven flow in a stairwell. They concluded that the k-ε model of turbulence

provided adequate results in terms of flow characteristics for both predicted velocities and heat transfer at the upper compartment of the stairwell when compared with the measured results.

Jedrzejewska-Scibak and Lipinski, 1985 [39] compared the results of a two-dimensional simulation of turbulent buoyant flows in an industrial hall with experimentally measured results and obtained satisfactory correlations. They concluded that despite the achievement of satisfactory correlations, there is a need for some degree of heuristic, experimental input.

However, there have been relatively few studies comparing numerical predictions for three-dimensional non-isothermal recirculating flows with measured results in building spaces. These include works such as that of Hjertager and Magnussen, 1977 [40] who developed a computer code based on the Patankar and Spalding SIMPLE algorithm which they employed to predict temperature and velocity fields in a space with a high side-wall air supply diffuser and adjacent ceiling mounted extract grilles. This study again employed the Launder and Spalding k-ε model of turbulence but without buoyancy modification, good agreement was achieved between predicted and experimental results for isothermal flows but not for buoyant flows.

Sakamoto and Matsuo, 1980 [41] employed a variation of the Harlow and Welch MAC method to examine isothermal flow in a rectangular space with a square ceiling mounted supply diffuser and a low side-wall extract grille, both a two equation model of turbulence and a more advanced 'large eddy' simulation procedure were used. Relatively good agreement was observed between their own experimental measurements of the velocity field and the predicted results, although there was some discrepancy in turbulence quantities. However, these discrepancies were exhibited by both turbulence modelling methods and the author's recommendation was to adopt the more computationally efficient two equation turbulence model.

Gosman et al, 1980 [42] applied the TEACH code in conjunction with the k-ε turbulence model to an isothermal flow in a rectangular room with a square side-wall diffuser at high level and a low-level extract grille situated on the far wall. This code had the facility to incorporate empirical data concerning the inlet flow properties within the finite volume control cells surrounding the inlet, thus dispensing with the requirement for fine mesh resolution in the vicinity of the inlet.

Chen Quingyan and Van Der Kooi, 1987 [43], have applied the PHOENICS [44] code and the CHAMPION [14] code in conjunction with the k-ε turbulence model to a space equipped with both heating and mechanical cooling facilities, both facilities employing forced convection. The test space comprised a rectilinear room with a window complete with venetian blind located in one end wall

and two fan-coil units supplying pre-cooled air positioned beneath this window, warm air being supplied during the heating cycle from a diffuser located in the opposite wall, air extraction effected by two floor level extract grilles sited on the same wall as the warm air diffuser. A particularly interesting feature of this investigation was the simulation of transient response to a step input of heat flux to the venetian blind. Very good correlations were established between predicted and experimental measurements for both temperature and velocity distribution during the cooling cycle. However, some discrepancy was noticed between experimentally measured and predicted temperature distributions for the heating cycle, and from this they concluded that the wall functions for heat flux required improvement.

Murakami et al, 1989 [49] obtained good correlation between experimental measurements and predicted results with the Launder and Spalding k-ε model through the application of a variant of the MAC method to isothermal flow in a rectangular space with obstacles in the flow domain, air being supplied to the space by means of ceiling mounted rectangular diffusers and extracted by low-level side-wall extract grilles.

Kurabuchi, Fang and Grot, 1989 [50] have also used a variant of the MAC method, in the form of their EXACT code, in conjunction with the k-ε turbulence model to investigate isothermal flows in building spaces and have again obtained a good comparison between predicted and experimentally measured results.

Haghighat, Jiang and Wang, 1989 [51] have applied a code based on the SIMPLE algorithm in conjunction with the k-ε turbulence model to a partitioned room with an interconnecting doorway between the two resulting spaces. Air flow being buoyancy driven, induced by a temperature differential between the two end walls, the remaining walls, ceiling and floor being assumed to be well insulated. The conclusions of the study were that the air flow pattern was sensitive to the location of the partition and the size and location of the doorway in the partition. The rate of heat transfer between the two spaces although being sensitive to door height and location was found not to be sensitive to partition location. Good agreement was obtained between measured and simulated Nusselt numbers.

Haghighat et al, 1992 [45] have developed the Concordia code, again based on the SIMPLE algorithm incorporating the Launder and Spalding k-ε turbulence model. They have attempted to validate the code using experimental data obtained for a typical office space. They concluded that, in general, satisfactory agreement was achieved between measured and simulated Nusselt numbers throughout the flow domain.

Ohira and Omari, 1996 [46] applied a code based on a variant of the SIMPLE algorithm employing the k-ε turbulence model to a problem involving the venting of gas water heater flue gases into a centralised apartment block courtyard. They compared the predicted results with measured results and concluded that the CFD predictions were reasonably accurate.

2.4 Application of CFD to Building Thermal Modelling

Chen Quingyan, 1989 [52] has used the PHOENICS [44] code and the CHAMPION [14] code in conjunction with the k-ε turbulence model to predict a number of air flow distributions for various forced convection boundary conditions in typical office type spaces. The resulting velocity distributions are stored in a database and made available to a room energy balance program ACCURACY. The ACCURACY program employs z-transfer function techniques to calculate transient heat conduction through the building fabric, simplified analytical techniques being employed in the treatment of shortwave and longwave radiation processes.

One of the conclusions of the study was that the accuracy of energy prediction for forced convection problems in offices where temperature differentials occur may be improved considerably through the use of CFD techniques.

A central assumption made in the technique adopted by Chen Quingyan is that the velocity field changes only very slightly with time if the supply air flow rate is maintained constant. This assumption allows a number of velocity fields to be calculated in advance for various boundary conditions (inlet mass, internal gains, etc.). The room energy balance program may then select an appropriate velocity field for the given boundary conditions and subsequently solve the energy equation (see Chapter 3) for temperature, calculating the energy equation convection coefficients from the pre-calculated velocities. This assumption will not be valid for flows involving a significant degree of natural convection, where the transient velocity field is being determined by the thermal boundary conditions.

Holmes, Lam, Ruddick and Whittle [53] have coupled a two-dimensional thermal model, R2D2 with a CFD code AIRFLO. The thermal model employs the radiosity concept for the calculation of inter-surface radiation exchange and due to the simple geometry of the building space involved, analytical radiation view factors are adopted for parallel and orthogonal planes. The model calculates transient heat conduction through the building fabric using an explicit finite difference algorithm. The CFD code uses a pressure coupled formulation for the solution of the momentum equations and the

'hybrid' differencing scheme (see Section 3.3.3) for the calculation of convection-diffusion coefficients.

Coupling between the thermal model and the CFD code was achieved through the surface convection coefficients which were derived from recommendations made by the Chartered Institute of Building Services Engineers [54].

The test case for the study comprised a rectilinear office space supplied with heated air by a ceiling mounted air diffuser, the temperature of the supply air being controlled by feedback from a sensor located at the return air grille, the grille itself being located near the floor level.

The thermal model and CFD code were run in both sequential and fully coupled mode. Holmes et al. concluded that the fully coupled simulation resulted in an oscillatory air flow pattern which indicated an instability in the control of the supply air temperature, this characteristic of the problem was not predicted through sequential operation of the codes.

Havet, Blay and Veyrat, 1994 [47] have coupled a thermal model with a CFD code using an iterative procedure in order to investigate air flow and temperature distribution within a large enclosure heated by radiant panels. The method involved the calculation of surface convective heat transfer coefficients using a CFD code followed by the calculation of surface temperatures using the calculated convection coefficients in conjunction with the thermal model. The calculated surface temperatures are then passed back to the CFD code and this process is continued until convergence is achieved. They concluded that the flow pattern was dependent on the location of the radiant panel and that the wall and mean air temperatures were insensitive to panel location.

Gan, 1995 [48] has combined a CFD code based on the SIMPLE algorithm with a radiosity model using an iterative approach. The resulting code was applied to a typical office space under winter heating and summer cooling modes. Gan concluded that the inclusion of a radiation model is necessary in order to obtain a satisfactory evaluation of thermal comfort.

There are commercial codes available including CFX, supplied by AEA Technology Engineering Software; PHOENICS, supplied by Concentration Heat and Momentum Limited and FLOW3D, supplied by Computational Fluid Dynamics Solutions which include facilities for radiation modelling. These facilities are generally based on the flux and zone methods described by Spalding [16] as opposed to a fully coupled model.

2.5 Conclusions

a) A summary of the developments in the field of Computational Fluid Dynamics (CFD) has been presented. It has been demonstrated that these developments have culminated in a set of numerical procedures (SIMPLE, SIMPLER and SIMPLEST) that are capable of solving the equation set defining transient three-dimensional advection, convection and conduction heat transfer processes in buildings.

b) The main characteristics of turbulent flow have been discussed and the interpretation of turbulent flow variables in terms of mean components and fluctuating components has been presented. Various methods of turbulence modelling have been discussed, ranging from the simplest constant effective viscosity models to the very complex Reynolds stress and large eddy simulation techniques. It has been concluded that a good compromise between modelling rigour and computational efficiency is offered by the two-equation family of models, adopting the Boussinesq eddy viscosity concept. The k-ε model is the preferred two-equation model due to the fact that it requires no modification for near-wall flows and that it is the most widely used and tested model.

c) It has been shown that an adequate number of investigations have been conducted into three dimensional turbulent isothermal air flows in building spaces [40][41][42][43][45][46][49][50][51] in order to establish that the Launder and Spalding k-ε turbulence model can provide very good correlations between experimentally measured and predicted velocity fields for this type of problem.

d) It appears that little research has been conducted into the validity of the k-ε turbulence model and wall function method when applied to three-dimensional turbulent buoyant flows in building spaces. The investigation conducted by Hjertager and Magnussen [40] produced poor correlations between predicted and measured results for buoyant flows although the version of the k-ε model employed had no modification to include buoyancy generation and destruction terms. Quingyan and Van Der Kooi [43] concluded from the results of their investigation into buoyant flows that the existing wall function methods require improvement.

e) The limited research conducted into the coupling of building thermal models with CFD codes [47][48][52][53][54] has indicated the potential for improving the range of applicability of independent CFD and thermal codes and the potential for the inclusion of a radiation model [48], although no attempt has been made to produce a universally applicable fully coupled three-dimensional model.

3.0 ADVECTION, CONVECTION AND CONDUCTION PROCESSES

In all practical building spaces, the field problem comprises a conjugate heat transfer process in which conduction in the confining solid elements (floors, walls, partitions, roof, etc.) and convection-diffusion in the confined air volumes must be considered with correct matching at the fluid-solid interfaces.

Adopting a conventional approach to the solution for transient heat conduction in solid regions (finite difference, response factor, etc.) in conjunction with a separate solution scheme for the fluid regions would either lead to severe approximations or an involved iterative procedure for matching the interface conditions.

This section describes the development of a single solution scheme for conjugate convection-conduction heat transfer processes throughout the calculation domain. Additionally, a detailed description of the k-ε turbulence model is included.

Methods for handling air infiltration processes and casual convective heat gains arising from occupants, lighting and equipment are also discussed and a general approach to the inclusion of surface radiative heat exchange is described, the approach being further developed in Chapter 4.

3.1 Mathematical Basis

3.1.1 The Dependent Variable Equation Set

The processes of fluid flow and heat transfer are governed by the principles of conservation of mass, momentum and energy. These principles may be expressed in the form of a series of partial differential equations. The equations are presented using Cartesian-Tensor notation.

The mass conservation or continuity equation takes the following form:-

$$\frac{\partial \rho}{\partial t} + \frac{\partial (\rho u_j)}{\partial x_j} = 0 \qquad \text{Eq.3.1}$$

When considering building air flows, the velocities are normally so low that we may consider the flow to be incompressible in which case the continuity equation reduces to:-

$$\frac{\partial u_j}{\partial x_j} = 0 \qquad \text{Eq.3.2}$$

The general equations describing conservation of momentum are sometimes referred to as the Navier-Stokes equations, derivations of which are described by Schlichting [34], et al. A compact form of the momentum conservation equations may be expressed as follows:-

$$\frac{\partial}{\partial t}(\rho u_i) + \frac{\partial}{\partial x_j}\left\{\rho u_j u_i + p\,\delta_{ij} - \mu\left[\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} - \frac{1}{3} e_{KK}\,\delta_{ij}\right]\right\} = g_i - f_i \qquad \text{Eq.3.3}$$

where

$$e_{ij} = \frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}$$

and therefore

$$e_{KK} = 2\frac{\partial u_K}{\partial x_K} = 2\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z}\right)$$

δ_{ij} is the Kronecker delta:-

$$\begin{cases} \delta_{ij} = 1; & i = j \\ \delta_{ij} = 0; & i \neq 1 \end{cases}$$

Again, assuming incompressible flow, the general momentum conservation equation reduces to:-

$$\frac{\partial}{\partial t}(\rho u_i) + \frac{\partial}{\partial x_j}\left\{\rho u_j u_i + p\,\delta_{ij} - \mu\left[\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right]\right\} = g_i - f_i \qquad \text{Eq.3.4}$$

where g_i = body force and f_i = friction force

Expanding this equation into the three velocity components and assuming negligible friction force:-

$$\frac{\partial}{\partial t}(\rho u) + \frac{\partial}{\partial x_j}(\rho u_j u) = \frac{\partial}{\partial x_j}\left[\mu\left(\frac{\partial u}{\partial x_j} + \frac{\partial u_j}{\partial x}\right)\right] - \frac{\partial p}{\partial x} + g_x \qquad \text{Eq.3.5}$$

$$\frac{\partial}{\partial t}(\rho v) + \frac{\partial}{\partial x_j}(\rho u_j v) = \frac{\partial}{\partial x_j}\left[\mu\left(\frac{\partial v}{\partial x_j} + \frac{\partial u_j}{\partial y}\right)\right] - \frac{\partial p}{\partial y} + g_y \qquad \text{Eq.3.6}$$

$$\frac{\partial}{\partial t}(\rho w) + \frac{\partial}{\partial x_j}(\rho u_j w) = \frac{\partial}{\partial x_j}\left[\mu\left(\frac{\partial w}{\partial x_j} + \frac{\partial u_j}{\partial z}\right)\right] - \frac{\partial p}{\partial z} + g_z \qquad \text{Eq.3.7}$$

The g terms in the momentum equations express the temperature dependent gravitational effect, i.e. buoyancy effect which is manifest in the V-component momentum conservation equation and may be described by the following term:-

$$g(\rho_r - \rho) \qquad \text{Eq.3.8}$$

The Boussinesq approximation is employed, which assumes the instantaneous temperature dependent properties of the fluid (density, molecular viscosity, etc.) to be invariant throughout the calculation domain with the exception of the buoyancy term where the local density is derived from an equation of state:-

$$Pv = RT \qquad \text{Eq.3.9}$$

If specific heat capacity is assumed invariant throughout the flow domain and independent of temperature throughout the range experienced and that kinetic heating and chemical reaction is absent, temperature may be considered a conserved property and the energy equation becomes:-

$$\frac{\partial}{\partial t}(\rho T) + \frac{\partial}{\partial x_j}(\rho u_j T) = \frac{\partial}{\partial x_j}\left(\Gamma_t \frac{\partial T}{\partial x_j}\right) + S_T \qquad \text{Eq.3.10}$$

where S_T = volumetric rate of heat generation

3.1.2 The k-ε Turbulence Model

If the substitution of $\overline{U_i} + U'$ is made for U in the Navier-Stokes equations, the resulting time-averaged form of the equation results:-

$$\frac{\partial}{\partial t}(\overline{\rho}\ \overline{u_i}) + \frac{\partial}{\partial x_j}\left\{\rho\ \overline{u_j u_i} + \overline{p}\,\delta_{ij} - \mu\left[\overline{\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} - \frac{1}{3} e_{KK}\delta_{ij}}\right]\right\} =$$

$$\overline{g_i} - \overline{f_i} - \frac{\partial}{\partial t}\overline{\rho' u_i'} - \frac{\partial}{\partial x_j}\overline{(\rho u_j)' u_i'} \qquad \text{Eq.3.11}$$

Usually $\frac{\partial}{\partial t}(\overline{\rho' u_i'})$ is neglected on the grounds that fluctuations in ρ' and U_i' are uncorrelated. The quantities $\overline{(\rho u_j)' u_i'}$ are known as the Reynolds stresses.

Similarly, the energy equation may be transformed into the following time-averaged form of the

equation:-

$$\frac{\partial}{\partial t}(\overline{\rho}\,\overline{T}) + \frac{\partial}{\partial x_j}\left\{\rho\,\overline{u_j}\,\overline{T} - \alpha\left[\frac{\partial T}{\partial x_j}\right]\right\} = S_T - \frac{\partial}{\partial x_j}\overline{(\rho\,u_j)'\,T'} \qquad \text{Eq.3.12}$$

Where $\frac{\partial}{\partial x_j}\overline{(\rho\,u_j)'\,T'}$ is the turbulent heat flux.

For general flow situations, the Boussinesq eddy viscosity concept may be expressed as:-

$$-\overline{(\rho\,u_j)'u_i'} = \mu_t\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) - \frac{2}{3}k\,\delta_{ij} \qquad \text{Eq.3.13}$$

The second term in Eq.3.13 is necessary to make the expression applicable also to normal stresses (when i = j), i.e. both sides will be equal to twice the kinetic energy of the fluctuating motion when a summation of normal stress is taken, see Rodi [19]. Similarly an eddy thermal diffusivity may be defined:-

$$-\overline{(\rho\,u_j)'T'} = \Gamma_t\left(\frac{\partial T}{\partial x_j}\right) \qquad \text{Eq.3.14}$$

Where Γ_t is the turbulent diffusivity. Like the eddy viscosity, Γ_t is not a fluid property but depends on the state of turbulence. In fact, the Reynolds analogy between thermal transport and momentum transport suggests that Γ_t is closely related to μ_t, ($\Gamma_t = \mu_t / \sigma_t$).

Adopting the same assumptions as in Section 3.2.1, applying the eddy viscosity concept, and omitting the overscore convention for time-averaging, the time averaged momentum equation becomes:-

$$\frac{\partial}{\partial t}(\rho\,u_i) + \frac{\partial}{\partial x_j}(\rho\,u_i u_j) = -\frac{\partial \Pi}{\partial x_j} + \frac{\partial}{\partial x_j}\left\{(\mu + \mu_t)\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right)\right\} - g_i \qquad \text{Eq.3.15}$$

Where Π = total pressure = $p + \frac{2}{3}k$

and the time averaged energy equation takes the form:-

$$\frac{\partial}{\partial t}(\rho T) + \frac{\partial}{\partial x_j}(\rho\,u_j\,T) = \frac{\partial}{\partial x_j}\left\{\left(\alpha + \frac{\mu_t}{\sigma_t}\right)\frac{\partial T}{\partial x_j}\right\} + S_T \qquad \text{Eq.3.16}$$

The equation for the transport of turbulence energy k can be derived in exact form from the Navier-Stokes equation, and is given as follows:-

$$\frac{\partial k}{\partial t} + u_j \frac{\partial k}{\partial x_j} = - \frac{\partial}{\partial x_j}\left[\overline{u_j'\left(\frac{u_i' u_i'}{2} + \frac{p'}{\rho}\right)} + \nu \frac{\partial k}{\partial x_j}\right] - \overline{u_i' u_j'} \frac{\partial u_i}{\partial x_j} - \beta g_i \overline{u_i' \theta'} -$$

I--------II----------III----------IV-------V----------VI----------

$$\nu \overline{\frac{\partial u_i'}{\partial x_j} \frac{\partial u_i'}{\partial x_j}} \qquad \text{Eq.3.17}$$

VII--------

in which the various terms represent the following:-

I - Rate of change

II - Convection

III - Turbulent diffusion

IV - Molecular diffusion

V - Production by shear (G_K)

VI - Buoyant production/destruction (G_B)

VII - Viscous dissipation (ε)

Certain assumptions concerning unknown correlations must be made before the transport equation can practically form part of the turbulence model. The turbulent diffusion flux of k is normally assumed to be proportional to the gradient of k:-

$$- \overline{u_i'\left(\frac{u_j' u_i'}{2} + \frac{P}{\rho}\right)} = \frac{\nu_t}{\sigma_k} \frac{\partial k}{\partial x_j} \qquad \text{Eq.3.18}$$

where σ_t is an empirical diffusion constant. The production by shear is approximated as follows:-

$$G_K = - \overline{u_i' u_j'} \frac{\partial u_i}{\partial x_j} \cong \nu_t \left(\frac{\partial u_j}{\partial x_i} + \frac{\partial u_i}{\partial x_j}\right) \frac{\partial u_i}{\partial x_j} \qquad \text{Eq.3.19}$$

The buoyant production/destruction term is modelled using the gradient approximation:-

$$G_B = - \beta g_i \overline{u_i' \theta'} \cong \beta g_i \frac{\nu_t}{\sigma_t} \frac{\partial \theta}{\partial x_i} \qquad \text{Eq.3.20}$$

where σ_t is the turbulent Prandtl No.

The transport equation for k then becomes:-

$$\frac{\partial k}{\partial t} + u_j \frac{\partial k}{\partial x_j} = \frac{\partial}{\partial x_j}\left[\left(\frac{\nu_t}{\sigma_k} + \nu\right)\frac{\partial k}{\partial x_j}\right] + \nu_t\left(\frac{\partial u_j}{\partial x_i} + \frac{\partial u_i}{\partial x_j}\right)\frac{\partial u_i}{\partial x_j} + \beta g_i \frac{\nu_t}{\sigma_t}\frac{\partial \theta}{\partial x_i} - \varepsilon \qquad \text{Eq.3.21}$$

An exact form of the equation for the rate of dissipation of turbulence energy may also be derived from the Navier-Stokes equation and is given as follows:-

$$\frac{\partial \varepsilon}{\partial t} + u_j \frac{\partial \varepsilon}{\partial x_j} = -\nu \frac{\partial}{\partial x_k} u_k' \overline{\frac{\partial u_i'}{\partial x_k}\frac{\partial u_i'}{\partial x_k}} - \frac{\nu}{\rho}\frac{\partial}{\partial x_i}\overline{\frac{\partial p'}{\partial x_l}\frac{\partial u_i'}{\partial x_l}} + \frac{\partial^2 \varepsilon}{\partial x_k{}^2} -$$

I---------II----------III----------

$$2\nu \frac{\partial u_i}{\partial x_k}\left(\overline{\frac{\partial u_i'}{\partial x_l}\frac{\partial u_k'}{\partial x_l} + \frac{\partial u_l'}{\partial x_i}\frac{\partial u_l'}{\partial x_k}}\right) - 2\nu \frac{\partial^2 u_i'}{\partial x_k \partial x_l}\overline{u_k' \frac{\partial u_i'}{\partial x_l}} -$$

IV----------

$$2\nu \overline{\frac{\partial u_i'}{\partial x_k}\frac{\partial u_i'}{\partial x_l}\frac{\partial u_k'}{\partial x_l}} - 2\left(\nu \overline{\frac{\partial^2 u_i'}{\partial x_k \partial x_l}}\right)^2 - \beta g_i \overline{u_i' \theta'}$$

V----------VI----------VII-------- Eq.3.22

in which the various terms represent the following:-

I - Rate of change

II - Convection

III - Diffusion

IV - Generation by the mean motion

V - Generation due to vortex stretching

VI - Destruction due to viscous action

VII - Buoyancy production/destruction

The production terms due to mean motion are omitted from the model form of the equation because they tend to be less important when the Reynolds No. is large and local turbulence structure becomes nearly isotropic, see Tennekes and Lumley [17].

The diffusion terms are modelled as in the k-equation, adopting the gradient diffusion assumption:-

$$- \nu \frac{\partial}{\partial x_k} \overline{u_k' \frac{\partial u_i'}{\partial x_k} \frac{\partial u_i'}{\partial x_k}} - \frac{\nu}{\rho} \frac{\partial}{\partial x_i} \overline{\frac{\partial p'}{\partial x_l} \frac{\partial u_i'}{\partial x_l}} + \frac{\partial^2 \varepsilon}{\partial x_k^2} \cong \frac{\partial}{\partial x_i} \left\{ \left(\frac{\nu_t}{\sigma_\varepsilon} + \nu \right) \frac{\partial \varepsilon}{\partial x_i} \right\} \qquad \text{Eq.3.23}$$

where σ_ε is an empirical diffusion coefficient.

The terms describing generation due to vortex stretching and dissipation due to viscous action must be modelled as a difference rather than the terms themselves, see Rodi [55]. The most widely used approximation for these terms is given as follows:-

$$- 2\nu \overline{\frac{\partial u_i'}{\partial x_k} \frac{\partial u_i'}{\partial x_l} \frac{\partial u_k'}{\partial x_l}} - 2 \left(\nu \overline{\frac{\partial^2 u_i'}{\partial x_k \partial x_l}} \right)^2 \cong \frac{\varepsilon}{k} (C_1 P - C_2 \varepsilon) \qquad \text{Eq.3.24}$$

where C_1 and C_2 are empirical constants and P is production by shear.

Using a similar approach to that adopted for the k-equation, the buoyancy term could be modelled simply by replacing P with $G_K + G_B$, however several numerical experiments to date have led to some controversy concerning the definition of a generally applicable coefficient for G_B, Rodi [19] has suggested that subsequent to replacing P with $G_K + G_B$, the C_1 coefficient should be supplemented with another coefficient being a function of the flux Richardson number ($R_f = - G_B / G_K$):-

$$\frac{\varepsilon}{k} (C_1 P - C_2 \varepsilon) = \frac{\varepsilon}{k} \{ [C_1 (G_K + G_B) (1 + C_3 R_f)] - C_2 \varepsilon \} \qquad \text{Eq.3.25}$$

where

$$G_B = - \beta g_i \overline{u_i' \theta'} \cong \beta g_i \frac{\nu_t}{\sigma_t} \frac{\partial \theta}{\partial x_i} \qquad \text{Eq.3.26}$$

However no universally valid correction has been identified and Rodi [56] suggests that a reasonable first approximation is to set $R_f = 0$, which is valid for vertical buoyant shear layers anyway. A common practice is to include a C_3 coefficient with the buoyancy term, and to adjust the value of the coefficient depending on the particular problem, i.e.:-

$$\frac{\varepsilon}{k} (C_1 P - C_2 \varepsilon) = \frac{\varepsilon}{k} (C_1 G_K - C_2 \varepsilon + C_3 G_B) \qquad \text{Eq.3.27}$$

Thus, the resulting model form of the ε-equation takes the following form:-

$$\frac{\partial \varepsilon}{\partial t} + u_i \frac{\partial \varepsilon}{\partial x_i} = \frac{\partial}{\partial x_i}\left\{\left(\frac{\nu_t}{\sigma_\varepsilon} + \nu\right)\frac{\partial \varepsilon}{\partial x_i}\right\} +$$

$$\frac{\varepsilon}{k}\left\{C_1 \nu \left(\frac{\partial u_j}{\partial x_i} + \frac{\partial u_i}{\partial x_j}\right)\frac{\partial u_i}{\partial x_j} - C_2\varepsilon + C_3 \beta g_i \frac{\nu_t}{\sigma_t}\frac{\partial \theta}{\partial x_i}\right\} \qquad \text{Eq.3.28}$$

Using the eddy viscosity concept i.e. $\nu_t \alpha \hat{V}L$, the velocity scale $\hat{V}$ is defined as $\sqrt{k}$ and the length scale L is defined as $C_\mu k^{3/2}/\varepsilon$ where C_μ is an empirical constant, thus the eddy viscosity is derived from the following relationship:-

$$\mu_t = C_\mu \rho \frac{k^2}{\varepsilon} \qquad \text{Eq.3.29}$$

A closed set of equations is thus provided for the turbulence model.

3.1.3 The Final Model Equation Set

The complete k-ε turbulence model together with the recommended empirical constants are presented as follows:-

3.1.3.1 Conservation Equations for the Time-Averaged Dependent Variables

$$\frac{\partial}{\partial t}(\rho u_i) + \frac{\partial}{\partial x_j}(\rho u_j u_i) = -\frac{\partial \Pi}{\partial x_i} + \frac{\partial}{\partial x_j}\left\{(\mu + \mu_t)\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right)\right\} - g_i \qquad \text{Eq.3.30}$$

where $\Pi = p + \frac{2}{3}k$

$$\frac{\partial}{\partial t}(\rho T) + \frac{\partial}{\partial x_j}(\rho u_j T) = \frac{\partial}{\partial x_j}\left\{\left(\alpha + \frac{\mu_t}{\sigma_t}\right)\frac{\partial T}{\partial x_j}\right\} + S_T \qquad \text{Eq.3.31}$$

3.1.3.2 Eddy Viscosity

$$\mu_t = C_\mu \rho \frac{k^2}{\varepsilon} \qquad \text{Eq.3.32}$$

3.1.3.3 Conservation Equations for Turbulence Kinetic Energy and its Dissipation Rate

$$\frac{\partial k}{\partial t} + u_i \frac{\partial k}{\partial x_i} = \frac{\partial}{\partial x_i}\left\{\left(\frac{\nu_t}{\sigma_k} + \nu\right)\frac{\partial k}{\partial x_i}\right\} + \nu_t\left(\frac{\partial u_j}{\partial x_i} + \frac{\partial u_i}{\partial x_j}\right)\frac{\partial u_i}{\partial x_j} + \beta g_i \frac{\nu_t}{\sigma_t}\frac{\partial \theta}{\partial x_i} - \varepsilon \qquad \text{Eq.3.33}$$

$$\frac{\partial \varepsilon}{\partial t} + u_i \frac{\partial \varepsilon}{\partial x_i} = \frac{\partial}{\partial x_i}\left\{\left(\frac{\nu_t}{\sigma_\varepsilon} + \nu\right)\frac{\partial \varepsilon}{\partial x_i}\right\} +$$

$$\frac{\varepsilon}{k}\left\{C_1 \nu\left(\frac{\partial u_j}{\partial x_i} + \frac{\partial u_i}{\partial x_j}\right)\frac{\partial u_i}{\partial x_j} - C_2\varepsilon + C_3\beta g_i \frac{\nu_t}{\sigma_t}\frac{\partial \theta}{\partial x_i}\right\} \qquad \text{Eq.3.34}$$

3.1.3.4 Empirical Constants

Values for the empirical constants are those recommended by Launder et al. and presented by Launder and Spalding [28]:-

C_μ	C_1	C_2	σ_k	σ_ε	C_3
0.09	1.44	1.92	1.0	1.3	1.0

3.2 The Numerical Procedure

General procedures for the numerical solution of the defining equation set are presented by Patankar [15] and Spalding [16] and two of these procedures are now summarised.

3.2.1 The General Differential Equation

Spalding [16] observes that the conservation equations for the time-averaged dependent variables and the turbulence kinetic energy and dissipation rate may be expressed in the form of a general differential equation:-

$$\frac{\partial}{\partial t}(\rho\phi) + \frac{\partial}{\partial x_j}(\rho u_j \phi) = \frac{\partial}{\partial x_j}\left(\Gamma_\phi \frac{\partial \phi}{\partial x_j}\right) + S_\phi \qquad \text{Eq.3.35}$$

where ϕ may be any of the dependent variables (velocity components, temperature) or one of the turbulence quantities (turbulence kinetic energy k, or dissipation rate ε), Γ is the diffusion coefficient for the prominent diffusion term and S_ϕ is the source term which accounts for all other remaining terms in the dependent variable and turbulence equations.

The basis of the numerical method is to integrate the general differential equation over a series of discrete volumes into which the flow domain is divided, the source terms of the resulting set of discretised equations are then linearised in order to derive a set of linear algebraic equations that may be solved using a variety of numerical solution methods.

3.2.2 The Finite Difference Form of the General Differential Equation

The flow domain is first sub-divided into finite volumes, grid points being located at the geometric centres of these volumes, see Figure 2.

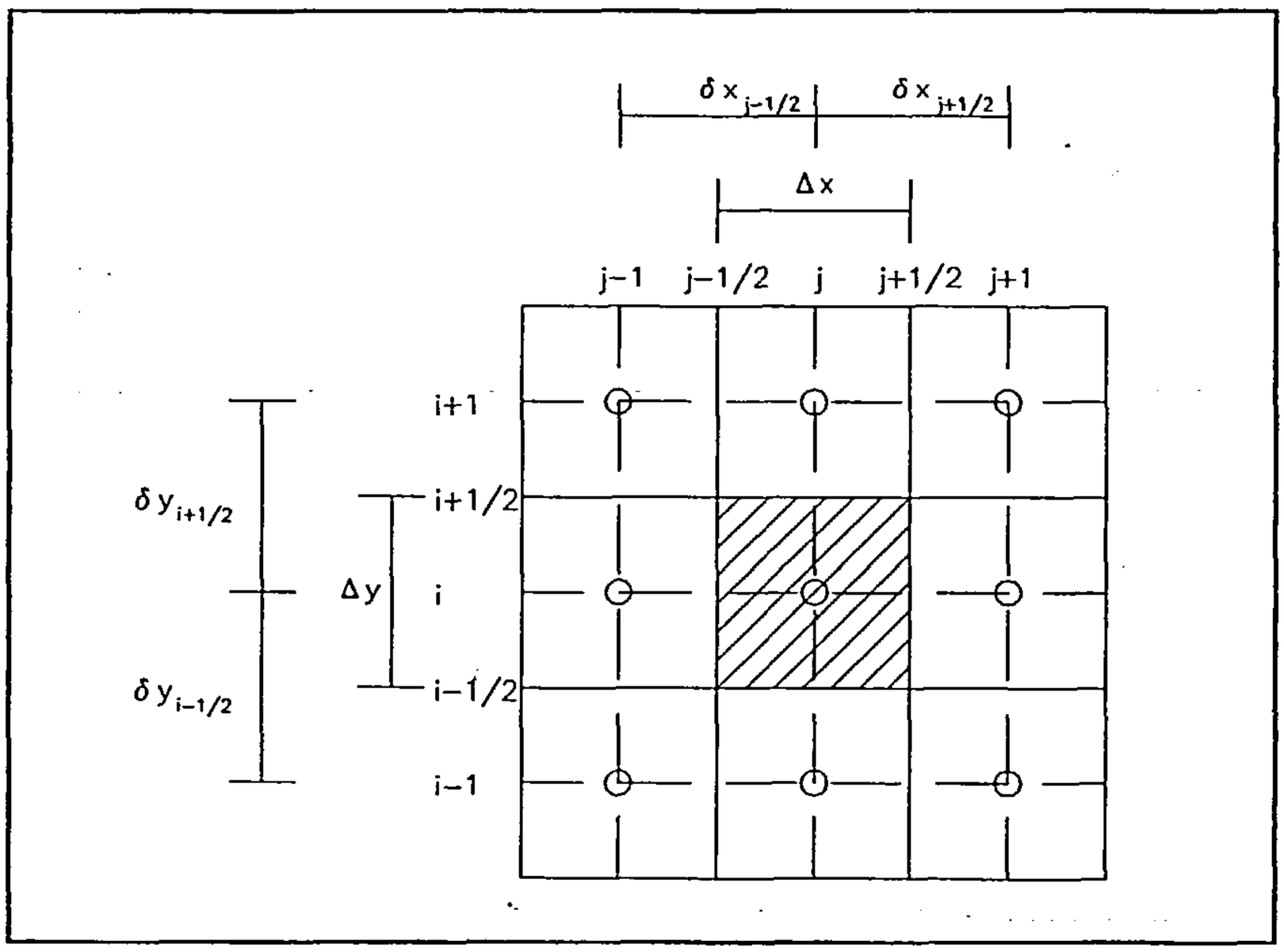


Figure 2 - X-Y section of the finite volume grid for dependent variables

For the purpose of clarity, the development of the numerical method is initially described using the two-dimensional, steady-state form of the equation set:-

$$\frac{d}{\partial x}(\rho u \phi) + \frac{d}{\partial y}(\rho v \phi) = \frac{\partial}{\partial x}\left(\Gamma_{\phi}\frac{\partial \phi}{\partial x}\right) + \frac{\partial}{\partial y}\left(\Gamma_{\phi}\frac{\partial \phi}{\partial y}\right) + S_{\phi} \qquad \text{Eq.3.36}$$

Additionally, the finite volume grid is assumed to comprise equal sized volumes throughout the flow domain, although the procedure actually adopted will allow non-uniform grids and this will be discussed later.

The general differential equation may be integrated over the control volume shown in Figure 2, i.e. $\Delta X \times \Delta Y \times 1$:-

$$\int_{i-1/2}^{i+1/2}\int_{j-1/2}^{j+1/2} \frac{d}{\partial x}(\rho u \phi) + \frac{d}{\partial y}(\rho v \phi) - \frac{\partial}{\partial x}\left(\Gamma_\phi \frac{\partial \phi}{\partial x}\right) - \frac{\partial}{\partial y}\left(\Gamma_\phi \frac{\partial \phi}{\partial y}\right) - S_\phi \, dx \, dy =$$

$$\Gamma_{i\,j+1/2}\left(\frac{\partial \phi}{\partial x}\right)_{j+1/2} \Delta y - \Gamma_{i\,j-1/2}\left(\frac{\partial \phi}{\partial x}\right)_{j-1/2} \Delta y + \Gamma_{i+1/2\,j}\left(\frac{\partial \phi}{\partial x}\right)_{i+1/2} \Delta x - \Gamma_{i-1/2\,j}\left(\frac{\partial \phi}{\partial x}\right)_{i-1/2} \Delta x -$$

$$(\rho u \phi)_{i\,j+1/2} \Delta y + (\rho u \phi)_{i\,j-1/2} \Delta y - (\rho v \phi)_{i+1/2\,j} \Delta x + (\rho v \phi)_{i-1/2\,j} \Delta x +$$

$$\int_{i-1/2}^{i+1/2}\int_{j-1/2}^{j+1/2} S_\phi \, dx \, dy = 0 \qquad \text{Eq.3.37}$$

If a piecewise linear profile is adopted for the spatial derivatives:-

$$\frac{\Gamma_{i\,j+1/2}(\phi_{j+1} - \phi_j)\,\Delta y}{\delta x_{j+1/2}} - \frac{\Gamma_{i\,j-1/2}(\phi_j - \phi_{j-1})\,\Delta y}{\delta x_{j-1/2}} + \frac{\Gamma_{i+1/2\,j}(\phi_{i+1} - \phi_i)\,\Delta x}{\delta y_{i+1/2}} -$$

$$\frac{\Gamma_{i-1/2\,j}(\phi_i - \phi_{i-1})\,\Delta x}{\delta y_{i-1/2}} - (\rho u \phi)_{i\,j+1/2} \Delta y + (\rho u \phi)_{i\,j-1/2} \Delta y - (\rho v \phi)_{i+1/2\,j} \Delta x +$$

$$(\rho v \phi)_{i-1/2\,j} \Delta x + \int_{i-1/2}^{i+1/2}\int_{j-1/2}^{j+1/2} S_\phi \, dx \, dy = 0 \qquad \text{Eq.3.38}$$

The integral $\int_{i-1/2}^{i+1/2}\int_{j-1/2}^{j+1/2} S_\phi \, dx \, dy = 0$ is evaluated to $\overline{S}\ \Delta x\, \Delta y$ where $\overline{S}$ is the average value of S over the control volume. In order to account for the fact that S may be a function of ϕ, the $\overline{S}$ term may be expressed as:-

$$\overline{S} = S_c + S_p \phi_p \qquad \text{Eq.3.39}$$

Equation 3.38 may be simplified through the introduction of two variables, the convection coefficient C and the diffusion coefficient D. The convection and diffusion coefficients for the i j+1/2 interface in Figure 2 would be:-

$$D_{i\,j+1/2} = \frac{\Gamma_{i\,j+1/2}}{\delta x_{j+1/2}} \Delta y \qquad \text{Eq.3.40}$$

$$C_{i\,j+1/2} = (\rho u)_{i\,j+1/2} \Delta y \qquad \text{Eq.3.41}$$

Incorporating these coefficients in the finite difference equation results in the following:-

$$D_{i\,j+1/2}\,(\phi_{i\,j+1} - \phi_{i\,j}) - D_{i\,j+1/2}\,(\phi_{i\,j+1} - \phi_{i\,j}) + D_{i+1/2\,j}\,(\phi_{i+1\,j} - \phi_{i\,j}) - D_{i-1/2\,j}\,(\phi_{i\,j+1} - \phi_{i-1\,j})$$
$$- C_{i\,j+1/2}\;\phi_{i\,j+1/2} + C_{i\,j-1/2}\;\phi_{i\,j-1/2} - C_{i+1/2\,j}\,\phi_{i+1/2\,j} + C_{i-1/2\,j}\;\phi_{i-1/2\,j} + \overline{S}\;\Delta x\,\Delta y = 0$$

Eq.3.42

3.2.3 Treatment of the Combined Convection-Diffusion Coefficients

If a strict central difference scheme was adopted, the control volume interface values of ϕ would be approximated by a linear interpolation, e.g.:-

$$\phi_{j+1/2} \sim (\phi_j + \phi_{j+1})/2 \qquad \text{Eq.3.43}$$

assuming that the interface lies midway between the grid points. The final discretised equation would then take the following form:-

$$a_{i\,j}\,\phi_{i\,j} = a_{i\,j+1/2}\,\phi_{i\,j+1} + a_{i\,j-1/2}\,\phi_{i\,j-1} + a_{i+1/2\,j}\,\phi_{i+1\,j} + a_{i-1/2\,j}\,\phi_{i-1\,j} + \overline{S}\;\Delta x\,\Delta y \qquad \text{Eq.3.44}$$

where,

$$a_{i\,j+1/2} = D_{i\,j+1/2} - C_{i\,j+1/2}/2$$
$$a_{i\,j-1/2} = D_{i\,j-1/2} + C_{i\,j-1/2}/2$$
$$a_{i+1/2\,j} = D_{i+1/2\,j} - C_{i+1/2\,j}/2$$
$$a_{i-1/2\,j} = D_{i-1/2\,j} + C_{i-1/2\,j}/2$$
$$a_{i\,j} = a_{i\,j+1/2} + a_{i\,j-1/2} + a_{i+1/2\,j} + a_{i-1/2\,j} + (C_{i\,j+1/2} - C_{i\,j-1/2} - C_{i+1/2\,j} - C_{i-1/2\,j})$$

It is noted at this point that in order to provide a physically realistic scheme, in the absence of a source term, the grid point coefficient must be equal to the sum of the neighbour coefficients, i.e.:-

$$a_{i\,j} = \Sigma\, a_{n\,b} \qquad \text{Eq.3.45}$$

This may be appreciated by considering a situation in the absence of convection when the discretised equation would appear thus:-

$$a_{i\,j}\,\phi_{i\,j} = D_{i\,j+1/2}\,\phi_{i\,j+1} + D_{i\,j-1/2}\,\phi_{i\,j-1} + D_{i+1/2\,j}\,\phi_{i+1\,j} + D_{i-1/2\,j}\,\phi_{i-1\,j} \qquad \text{Eq.3.46}$$

In this case, if the neighbour values of ϕ were to be increased by an arbitrary constant K, the grid point value must also increase by K and the only way of ensuring this is to make the grid point coefficient equal to the sum of the neighbour coefficients.

Similarly, in order to ensure that the grid point values increase or decrease as the neighbour grid point values, the coefficients must be of the same sign and thus a convention is adopted to ensure that the coefficients are always positive.

For this reason, at high absolute values of Peclet No., when C exceeds 2D, there is a possibility that the grid point neighbour coefficients could become negative whilst the grid point coefficient is positive resulting in a physically unrealistic scheme, and indeed a numerically unstable one.

A solution to this problem, originally proposed by Courant Isaacson and Rees [10], is to adopt the upwind scheme, whereby the interface value of ϕ takes on the value at the upwind grid point, i.e.:-

$$\left.\begin{aligned} \phi_{i\,j+1/2} &= \phi_{i\,j} \qquad C_{i\,j+1/2} > 0; \\ \phi_{i\,j+1/2} &= \phi_{i\,j+1}; \qquad C_{i\,j+1/2} < 0; \end{aligned}\right\}$$

This scheme may be represented in the following 'pseudo-code' form:-

$$C_{i\,j+1/2}\,\phi_{i\,j+1/2} = \max(C_{i\,j+1/2}, 0)\,\phi_{i\,j} - \max(-C_{i\,j+1/2}, 0)\,\phi_{i\,j+1} \qquad \text{Eq.3.47}$$

Equation 3.44 may then be modified to include this new formula for convective flux:-

$$a_{i\,j}\,\phi_{i\,j} = a_{i\,j+1/2}\,\phi_{i\,j+1} + a_{i\,j-1/2}\,\phi_{i\,j-1} + a_{i+1/2\,j}\,\phi_{i+1\,j} + a_{i-1/2\,j}\,\phi_{i-1\,j} + \overline{S}\,\Delta x\,\Delta y \qquad \text{Eq.3.48}$$

where,

$$\begin{aligned} a_{i\,j+1/2} &= D_{i\,j+1/2} + \max(-C_{i\,j+1/2}, 0) \\ a_{i\,j-1/2} &= D_{i\,j-1/2} + \max(C_{i\,j-1/2}, 0) \\ a_{i+1/2\,j} &= D_{i+1/2\,j} + \max(-C_{i+1/2\,j}, 0) \\ a_{i-1/2\,j} &= D_{i-1/2\,j} + \max(C_{i-1/2\,j}, 0) \\ a_{i\,j} &= a_{i\,j+1/2} + a_{i\,j-1/2} + a_{i+1/2\,j} + a_{i-1/2\,j} + (C_{i\,j+1/2} - C_{i\,j-1/2} - C_{i+1/2\,j} - C_{i-1/2\,j}) \end{aligned}$$

It will be noticed that the grid point coefficient will only be equal to the sum of the neighbour

coefficients when the continuity equation is satisfied, this may be ensured by combining the discretised dependent variable equation with the discretised form of the continuity equation.

The two-dimensional continuity equation takes the following form:-

$$\frac{\partial(\rho u)}{\partial x} + \frac{\partial(\rho v)}{\partial y} = 0 \qquad \text{Eq.3.49}$$

This may be integrated over the control volume shown in Figure 2:-

$$C_{i\,j+1/2} - C_{i\,j-1/2} - C_{i+1/2\,j} - C_{i-1/2\,j} = 0 \qquad \text{Eq.3.50}$$

where C is as defined in Section 3.3.2.

Eq.3.50 is now multiplied through by ϕ and then subtracted from the discretised form of the general differential equation (Eq.3.48):-

$$a_{i\,j}\phi_{i\,j} = a_{i\,j+1/2}\phi_{i\,j+1} + a_{i\,j-1/2}\phi_{i\,j-1} + a_{i+1/2\,j}\phi_{i+1\,j} + a_{i-1/2\,j}\phi_{i-1\,j} + b \qquad \text{Eq.3.51}$$

where,

$$a_{i\,j+1/2} = D_{i\,j+1/2} + \max(-C_{i\,j+1/2}, 0)$$
$$a_{i\,j-1/2} = D_{i\,j-1/2} + \max(C_{i\,j-1/2}, 0)$$
$$a_{i+1/2\,j} = D_{i+1/2\,j} + \max(-C_{i+1/2\,j}, 0)$$
$$a_{i-1/2\,j} = D_{i-1/2\,j} + \max(C_{i-1/2\,j}, 0)$$
$$a_{i\,j} = a_{i\,j+1/2} + a_{i\,j-1/2} + a_{i+1/2\,j} + a_{i-1/2\,j} - S_p\,\Delta x\,\Delta y$$
$$b = S_c\,\Delta x\,\Delta y$$

3.2.4 Alternative Combined Convection-Diffusion Coefficient Schemes

Considering the one-dimensional form of the general differential equation (Eq.3.48), without sources:-

$$\frac{\partial}{\partial x}(\rho u \phi) = \frac{\partial}{\partial x}\left(\Gamma_\phi \frac{\partial \phi}{\partial x}\right) \qquad \text{Eq.3.52}$$

Assuming Γ_ϕ to be constant and adopting the following boundary conditions for the domain

$0 < x < L$:-

$$\left.\begin{aligned} \phi &= 0; \quad x = 0 \\ \phi &= \phi_L; \quad x = L \end{aligned}\right\}$$

the solution to Eq.3.52 is shown by Patankar [15] to be:-

$$\frac{\phi - \phi_0}{\phi_L - \phi_0} = \frac{\exp(Px/L)-1}{\exp(P)-1} \qquad (P = \text{Peclet No.}) \qquad \text{Eq.3.53}$$

This solution is presented graphically in Figure 3. It will be noticed that the upwind scheme is only correct for large values of $|P|$.

In order to provide a differencing scheme which is equally applicable to low P values as well as high P values, various schemes have been proposed, including the exponential and hybrid schemes after Spalding [16], and the power-law scheme after Patankar [15].

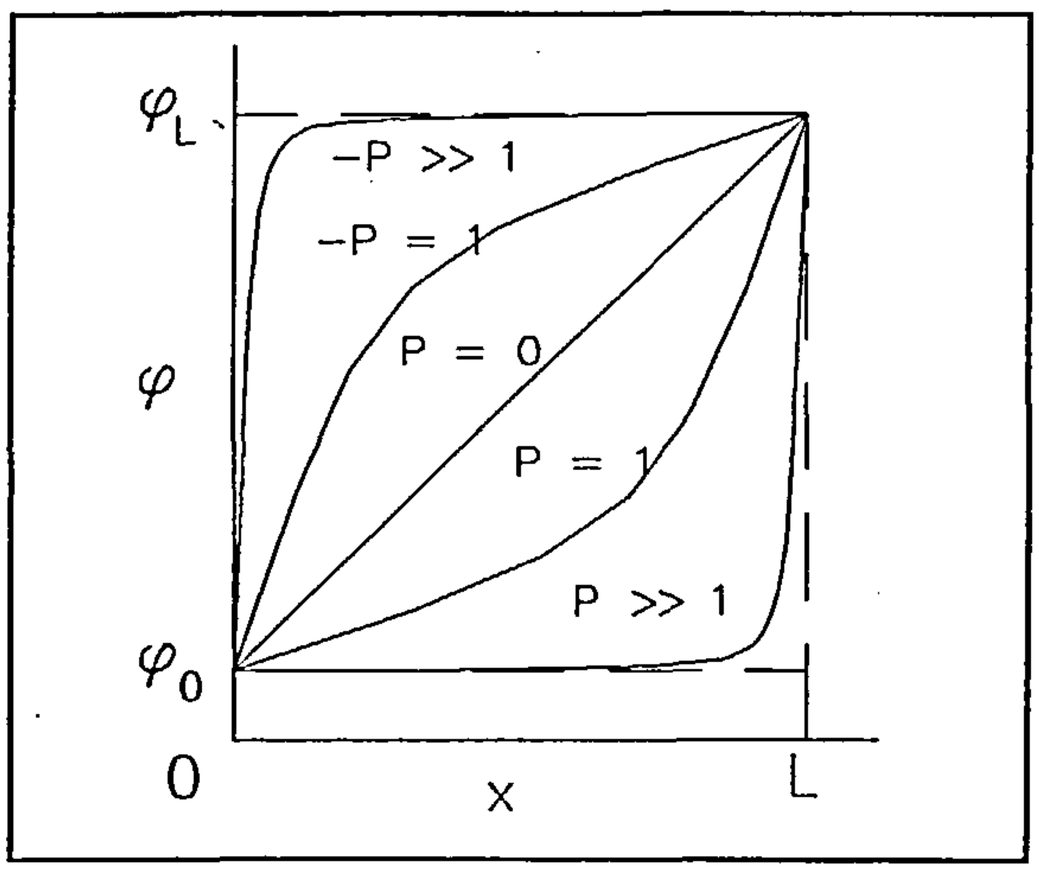


Figure 3 - exact solution to one-dimensional convection-diffusion problem

Patankar [15] proposes a method whereby the various schemes may be expressed as functions of Peclet number in the formulae for the discretised equation coefficients, e.g.:-

$$a_{i\,j+1/2} = D_{i\,j+1/2}\, f(P) + \max(-C_{i\,j+1/2}, 0) \qquad \text{Eq.3.54}$$

where f(P) is the function of Peclet number which represents the particular combined convection-diffusion scheme:-

Central difference	$1 - 0.5\|P\|$
Upwind	1
Hybrid	$\max(0, 1 - 0.5\|P\|)$
Power Law	$\max(0, (1 - 0.1\|P\|)^5)$
Exponential	$\|P\| / [\exp(\|P\|) - 1]$

3.2.5 Inclusion of the Unsteady Term

Returning to the unsteady form of the general differential equation:-

$$\frac{\partial}{\partial t}(\rho\phi) + \frac{\partial}{\partial x_j}(\rho\, u_j\, \phi) = \frac{\partial}{\partial x_j}\left(\Gamma_\phi \frac{\partial \phi}{\partial x_j}\right) + S\phi \qquad \text{Eq.3.55}$$

and assuming that the grid point value of ϕ is constant throughout the control volume indicated in Figure 2, the temporal derivative may be integrated over the control volume as follows:-

$$\int_{i-1/2}^{i+1/2}\int_{j-1/2}^{j+1/2}\int_{t}^{t+\Delta t} \frac{\partial \phi}{\partial t}\, dt\, dx\, dy = \rho\, \Delta x\, \Delta y\, (\phi_p{}^1 - \phi_p{}^0) \qquad \text{Eq.3.56}$$

where $\phi_p{}^1$ indicates the future time row value of ϕ and $\phi_p{}^0$ indicates the current time row value of ϕ.

If it is assumed that the grid point values of ϕ are a weighted average of the current and future time row values, subsequent to rearrangement of the terms in accordance with the procedure outlined in Sections 3.2.1 - 3.2.3, we arrive at the following:-

$$\begin{aligned} a_{i\,j}\phi_{i\,j} = {} & a_{i\,j+1/2}\,[f\phi^1{}_{i\,j+1} + (1-f)\,\phi^0{}_{i\,j+1}] + a_{i\,j-1/2}\,[f\phi^1{}_{i\,j-1} + (1-f)\,\phi^0{}_{i\,j-1}] + \\ & a_{i+1/2\,j}\,[f\phi^1{}_{i+1\,j} + (1-f)\,\phi^0{}_{i+1\,j}] + a_{i-1/2\,j}\,[f\phi^1{}_{i-1\,j} + (1-f)\,\phi^0{}_{i-1\,j}] + \\ & [a^0{}_{i\,j} - (1-f)\,a_{i\,j+1/2} - (1-f)\,a_{i\,j-1/2} - (1-f)\,a_{i+1/2\,j} - (1-f)\,a_{i-1/2\,j}]\,\phi^0{}_{i\,j} \end{aligned} \qquad \text{Eq.3.57}$$

where f is the weighting factor.

As noted by Patankar [15], unless the weighting factor f is taken to be 1.0 (i.e. a fully implicit scheme where the grid point values take on the future time row values), the coefficient of $\phi^0{}_{i\,j}$ may at times become negative thus leading to a physically unrealistic scheme (see Section 3.2.3). Thus the fully implicit scheme becomes the preferred scheme and the final unsteady three-dimensional

discretised equation set may be presented as follows:-

$$a_{i\,j\,k}\phi_{i\,j\,k} = a_{i\,j+1/2\,k}\phi_{i\,j+1\,k} + a_{i\,j-1/2\,k}\phi_{i\,j-1\,k} + a_{i+1/2\,j\,k}\phi_{i+1\,j\,k} + a_{i-1/2\,j\,k}\phi_{i-1\,j\,k} + a_{i\,j\,k+1/2}\phi_{i\,j\,k+1} + a_{i\,j\,k-1/2}\phi_{i\,j\,k-1} + b \qquad \text{Eq.3.58}$$

where,

$$a_{i\,j+1/2\,k} = D_{i\,j+1/2\,k} f(P_{i\,j+1/2\,k}) + \max(-C_{i\,j+1/2\,k}, 0)$$

$$a_{i\,j-1/2\,k} = D_{i\,j-1/2\,k}\ f(P_{i\,j-1/2\,k}) + \ \max(C_{i\,j-1/2\,k}, 0)$$

$$a_{i+1/2\,j\,k} = D_{i+1/2\,j\,k}\, f(P_{i+1/2\,j\,k}) + \ \max(-C_{i+1/2\,j\,k}, 0)$$

$$a_{i-1/2\,j\,k} = D_{i-1/2\,j\,k}\, f(P_{i-1/2\,j\,k}) + \ \max(C_{i-1/2\,j\,k}, 0)$$

$$a_{i\,j\,k+1/2} = D_{i\,j\,k+1/2}\, f(P_{i\,j\,k+1/2}) + \ \max(-C_{i\,j\,k+1/2}, 0)$$

$$a_{i\,j\,k-1/2} = D_{i\,j\,k-1/2}\, f(P_{i\,j\,k-1/2}) + \ \max(C_{i\,j\,k-1/2}, 0)$$

$$a^0_{i\,j\,k} = \rho\, \Delta x\ \Delta y\, \Delta z\, /\, \Delta t$$

$$a_{i\,j\,k} = a_{i\,j+1/2\,k} + a_{i\,j-1/2\,k} + a_{i+1/2\,j\,k} + a_{i-1/2\,j\,k} + a_{i\,j\,k+1/2} + a_{i\,j\,k-1/2} + a^0_{i\,j\,k} - S_p\, \Delta x\ \Delta y\, \Delta z$$

$$b = S_c\, \Delta x\ \Delta y\, \Delta z + a^0_{i\,j\,k}\, \phi^0_{i\,j\,k}$$

3.2.6 Representation of the Momentum Equation Pressure Terms - the Staggered Grid

The pressure term $-\partial\Pi/\partial x$ (Π = total pressure) appears as one of the momentum equation source terms (ref. Section 3.2). When the pressure term is integrated over the control volume shown in Figure 2, the resulting contribution to, say, the U-momentum equation is $\Pi_{i\,j-1/2} - \Pi_{i\,j+1/2}$. If a linear interpolation practice is employed, this term becomes:-

$$(\Pi_{i\,j-1} + \Pi_{i\,j})/2 - (\Pi_{i\,j} + \Pi_{i\,j+1})/2 = (\Pi_{i\,j-1} - \Pi_{i\,j+1})/2 \qquad \text{Eq.3.59}$$

thus, the pressure differentials are established from alternate grid point pressures rather than adjacent ones, this practice can lead to a serious numerical problem whereby a 'zig-zag' pressure field would behave as a uniform field, as illustrated in Figure 4.

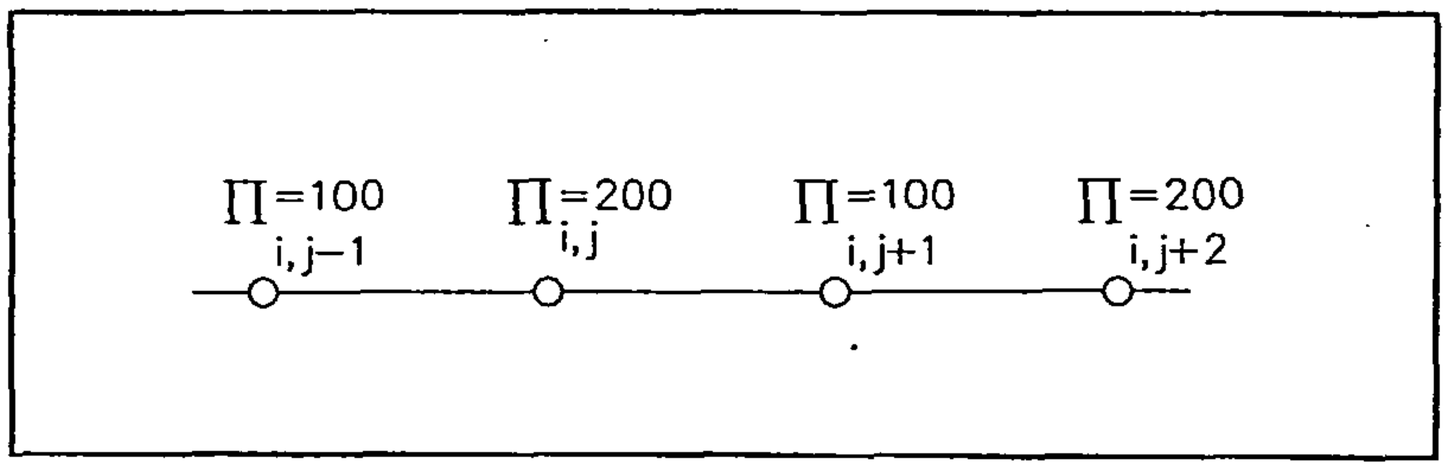


Figure 4 - 'zig-zag' pressure field

A similar problem arises in the construction of the discretised continuity equation. Harlow and Welch [57] proposed a remedy for this problem which is to adopt staggered grids for the velocity components as illustrated in Figure 5.

Using this procedure, pressure differentials for the momentum equations are extracted from adjacent grid points and the discretised continuity equation similarly may be constructed from adjacent grid point velocity components.

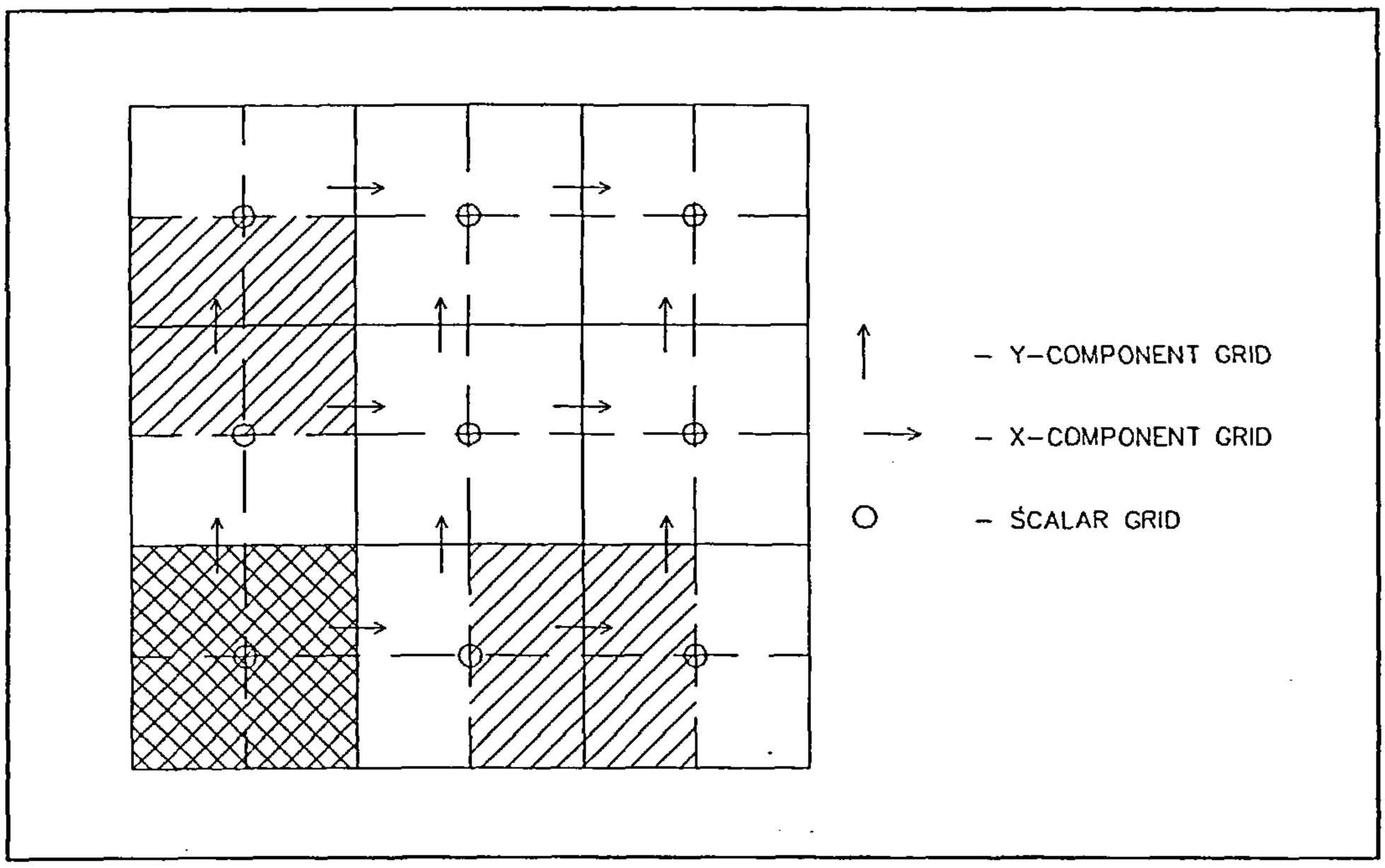


Figure 5 - staggered grid for velocity components

Note - velocities for velocity component control volume interface flow rates are derived from interpolations as indeed are diffusion coefficients for scalar grid control volume interfaces and certain velocity component control volume interfaces, see Section 3.3.9.

3.2.7 Derivation of the Unknown Pressure Field - the SIMPLE Algorithm

As noted in Section 3.3.5, the pressure term forms one of the momentum equation source terms, however there is no equation as such from which the pressure field may be derived. In order to develop a method of deriving this unknown pressure field, the pressure term is extracted from the source term and integrated separately. The basis of the SIMPLE algorithm is to represent the pressures and velocities in terms of 'guessed' values and corrections and to cast the discretised continuity equation into the form of a pressure correction equation, i.e.:-

$$\Pi = \Pi^* + \Pi' \qquad \text{Eq.3.60}$$

$$u = u^* + u' \qquad \text{Eq.3.61}$$

the discretised U-momentum equation derived generally in Section 3.2 may be represented as:-

$$a_{i\,j+1/2}\, u_{i\,j+1/2} = \Sigma\, a_{nb} u_{nb} + b + A(\Pi_{i\,j} - \Pi_{i\,j+1}) \qquad \text{Eq.3.62}$$

where

$$a_{nb}u_{nb} = a_{i\,j+1/2}\, u_{i\,j+1} + a_{i\,j-1/2}\, u_{i\,j-1} + a_{i+1/2\,j}\, u_{i+1\,j} + a_{i-1/2\,j}\, u_{i-1\,j}$$

and

$$A = \Delta y \times 1$$

and in terms of guessed velocities and pressures:-

$$a_{i\,j+1/2}\, u^*_{i\,j+1/2} = \Sigma\, a_{nb} u^*_{nb} + b + A(\Pi^*_{i\,j} - \Pi^*_{i\,j+1}) \qquad \text{Eq.3.63}$$

If Eq 3.63 is subtracted from Eq 3.62 the velocity correction equation results:-

$$a_{i\,j+1/2}\, u'_{i\,j+1/2} = \Sigma\, a_{nb} u'_{nb} + A(\Pi'_{i\,j} - \Pi'_{i\,j+1}) \qquad \text{Eq.3.64}$$

At this point, the SIMPLE procedure dispenses with the $\Sigma\, a_{nb}u'_{nb}$ term, in order to procure a manageable pressure correction equation of a similar form to the discretised general differential equation and thus open to the same solution techniques. Detailed discussion of the motivation behind this approximation is given by Patankar [15], however in summary, the pressure correction equation may be perceived to be an intermediate algorithm that has no effect on the final solution assuming that convergence is achieved, this is due to the fact that the pressure correction equation is not actually required during the final iteration. It is due to the omission of these neighbouring velocity corrections that the algorithm is called a 'Semi-Implicit' method.

The velocity correction equations now become:-

$$u'_{i\,j+1/2} = d_{i\,j+1/2}(\Pi'_{i\,j} - \Pi'_{i\,j+1}) \qquad \text{Eq.3.65}$$

or

$$u_{i\,j+1/2} = u^{*}_{i\,j+1/2} + d_{i\,j+1/2}(\Pi'_{i\,j} - \Pi'_{i\,j+1}) \qquad \text{Eq.3.66}$$

where

$$d_{i\,j+1/2} = A/a_{i\,j+1/2}$$

The two-dimensional continuity equation takes the form:-

$$\frac{\partial(\rho\, u)}{\partial x} + \frac{\partial(\rho\, v)}{\partial y} = 0 \qquad \text{Eq.3.67}$$

Integrating the continuity equation over the control volume shown in Figure 2 results in the following discretised form of the equation:-

$$[(\rho u)_{i\,j+1/2} - (\rho u)_{i\,j-1/2}]\,\Delta y + [(\rho v)_{i+1/2\,j} - (\rho v)_{i-1/2\,j}]\,\Delta x = 0 \qquad \text{Eq.3.68}$$

The velocity components in the discretised continuity equation are now replaced with velocity correction equations, which results in the following pressure correction equation:-

$$a_{i\,j}\,\Pi'_{i\,j} = a_{i\,j+1/2}\,\Pi'_{i\,j+1} + a_{i\,j-1/2}\,\Pi'_{i\,j-1} + a_{i+1/2\,j}\,\Pi'_{i+1\,j} + a_{i-1/2\,j}\,\Pi'_{i-1\,j} + b \qquad \text{Eq.3.69}$$

where

$$a_{i\,j+1/2} = \rho_{i\,j+1/2}\, d_{i\,j+1/2}\,\Delta y$$
$$a_{i\,j-1/2} = \rho_{i\,j-1/2}\, d_{i\,j-1/2}\,\Delta y$$
$$a_{i+1/2\,j} = \rho_{i+1/2\,j}\, d_{i+1/2\,j}\,\Delta x$$
$$a_{i\,-1/2\,j} = \rho_{i-1/2\,j}\, d_{i-1/2\,j}\,\Delta x$$
$$b = [(\rho u^{*})_{i\,j-1/2} - (\rho u^{*})_{i\,j+1/2}]\,\Delta y + [(\rho v^{*})_{i-1/2\,j} - (\rho v^{*})_{i+1/2\,j}]\,\Delta x$$

The pressure correction equation completes the equation set required to calculate the flow field variables.

3.2.8 The SIMPLER Algorithm

The SIMPLE algorithm has been modified and provided with a less arbitrary method of establishing the pressure field, the revised algorithm is called SIMPLE(R)evised, after Patankar [15]. The pressure field in SIMPLER is derived from a very similar equation to the pressure correction equation employed by SIMPLE, with the significant exception that no approximations are introduced in the derivation of the equation, which may be described as follows:-

The momentum equation is first arranged in the following form:-

$$u_{i\,j+1/2} = \frac{\Sigma\, a_{nb} u_{nb} + b}{a_{i\,j+1/2}} + d_{i\,j+1/2}(\Pi^*_{i\,j} - \Pi^*_{i\,j+1}) \qquad \text{Eq.3.70}$$

a 'pseudo-velocity' may then be defined:-

$$\hat{u}_{i\,j+1/2} = \frac{\Sigma\, a_{nb} u_{nb} + b}{a_{i\,j+1/2}} \qquad \text{Eq.3.71}$$

Thus the momentum equation becomes:-

$$u_{i\,j+1/2} = \hat{u}_{i\,j+1/2} + d_{i\,j+1/2}(\Pi^*_{i\,j} - \Pi^*_{i\,j+1}) \qquad \text{Eq.3.72}$$

This equation has a very similar form to the velocity correction equation (Eq.3.66). Thus, in a similar fashion to that employed in the derivation of the pressure correction equation, the pseudo-velocity equations are substituted into the discretised continuity equation, resulting in the following pressure equation:-

$$a_{i\,j}\,\Pi^*_{i\,j} = a_{i\,j+1/2}\,\Pi^*_{i\,j+1} + a_{i\,j-1/2}\,\Pi^*_{i\,j-1} + a_{i+1/2\,j}\,\Pi^*_{i+1\,j} + a_{i-1/2\,j}\,\Pi^*_{i-1\,j} \qquad \text{Eq.3.73}$$

where

$a_{i\,j+1/2} = \rho_{i\,j+1/2}\, d_{i\,j+1/2}\, \Delta y$

$a_{i\,j-1/2} = \rho_{i\,j-1/2}\, d_{i\,j-1/2}\, \Delta y$

$a_{i+1/2\,j} = \rho_{i+1/2\,j}\, d_{i+1/2\,j}\, \Delta x$

$a_{i\,-1/2\,j} = \rho_{i-1/2\,j}\, d_{i-1/2\,j}\, \Delta x$

$b = [(\rho\,\hat{u})_{i\,j-1/2} - (\rho\,\hat{u})_{i\,j+1/2}]\,\Delta y + [(\rho\,\hat{v})_{i-1/2\,j} - (\rho\,\hat{v})_{i+1/2\,j}]\,\Delta x$

The SIMPLER algorithm may then be represented in the form of the flow chart presented in Figure 6.

3.2.9 Solution of the Finite Volume Discretised Equation Set

The equation set is solved using a line-by-line method incorporating the Tri-Diagonal-Matrix-Algorithm.

3.2.9.1 The Tri-Diagonal-Matrix-Algorithm (TDMA) and the Line-By-Line Method

This method is perhaps best described by initial consideration of a one-dimensional problem, in which case the discretised equation set has the following form:-

$$a_j \phi_j = a_{j+1/2} \phi_{j+1} + a_{j-1/2} \phi_{j-1} + b \qquad \text{Eq.3.77}$$

and in matrix form:-

$$\begin{bmatrix} a_1 & a_2 & & & \\ a_1 & a_2 & a_3 & & \\ & a_2 & a_3 & a_4 & \\ & & a_3 & a_4 & a_5 \\ & & & a_4 & a_5 \end{bmatrix} \begin{bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \phi_4 \\ \phi_5 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \end{bmatrix}$$

It is this characteristic form of the three matrix diagonals that gives the TDMA its name.

For the purpose of describing the algorithm, the equations are represented using the following convention:-

$$a_i \phi_i = b_i \phi_{i+1} + c_i \phi_{i-1} + d_i \qquad \text{Eq.3.78}$$

The boundary equations have a special form whereby either the boundary value of ϕ is known, or is not required (e.g. specification of a flux), thus $a_1 = 0$ and a_2 is therefore known in terms of a_3. Because a_2 may be expressed in terms of a_3, a_3 may correspondingly be expressed in terms of a_4 and ultimately a_n may be expressed in terms of a_{n+1}.

This may be put into the form of the following algorithm:-

$$\phi_i = A_i \phi_{i+1} + B_i \qquad \text{Eq.3.79}$$

and thus

$$\phi_{i-1} = A_{i-1}\,\phi_i + B_{i-1} \qquad \text{Eq.3.80}$$

Substituting Eq.3.80 in Eq.3.78:

$$a_i\,\phi_i = b_i\,\phi_{i+1} + c_i(A_{i-1}\,\phi_i + B_{i-1}) + d_i \qquad \text{Eq.3.81}$$

Referring to Eq.3.79, the coefficients A_i and B_i may be expressed in terms of A_{i-1} and B_{i-1}:

$$A_i = \frac{b_i}{a_i - c_i\,A_{i-1}} \qquad \text{Eq.3.82}$$

$$B_i = \frac{d_i + c_i\,B_{i-1}}{a_i - c_i\,A_{i-1}} \qquad \text{Eq.3.83}$$

At the starting boundary, $c_i = 0$ (Eq.3.78), thus:

$$A_1 = \frac{b_1}{a_1} \qquad \text{Eq.3.84}$$

$$B_1 = \frac{d_1}{a_1} \qquad \text{Eq.3.85}$$

Equations 3.82 and 3.83 may then be employed until node n-1 is encountered, at that point, $b_n = 0$ and thus $A_n = 0$, and from Eq.3.79 $\phi_n = B$. From this point, back substitution of A_i and B_i in Eq.3.79 may be conducted, thus allowing ϕ_i to be derived.

The TDMA may be applied to a two-dimensional situation as follows:-

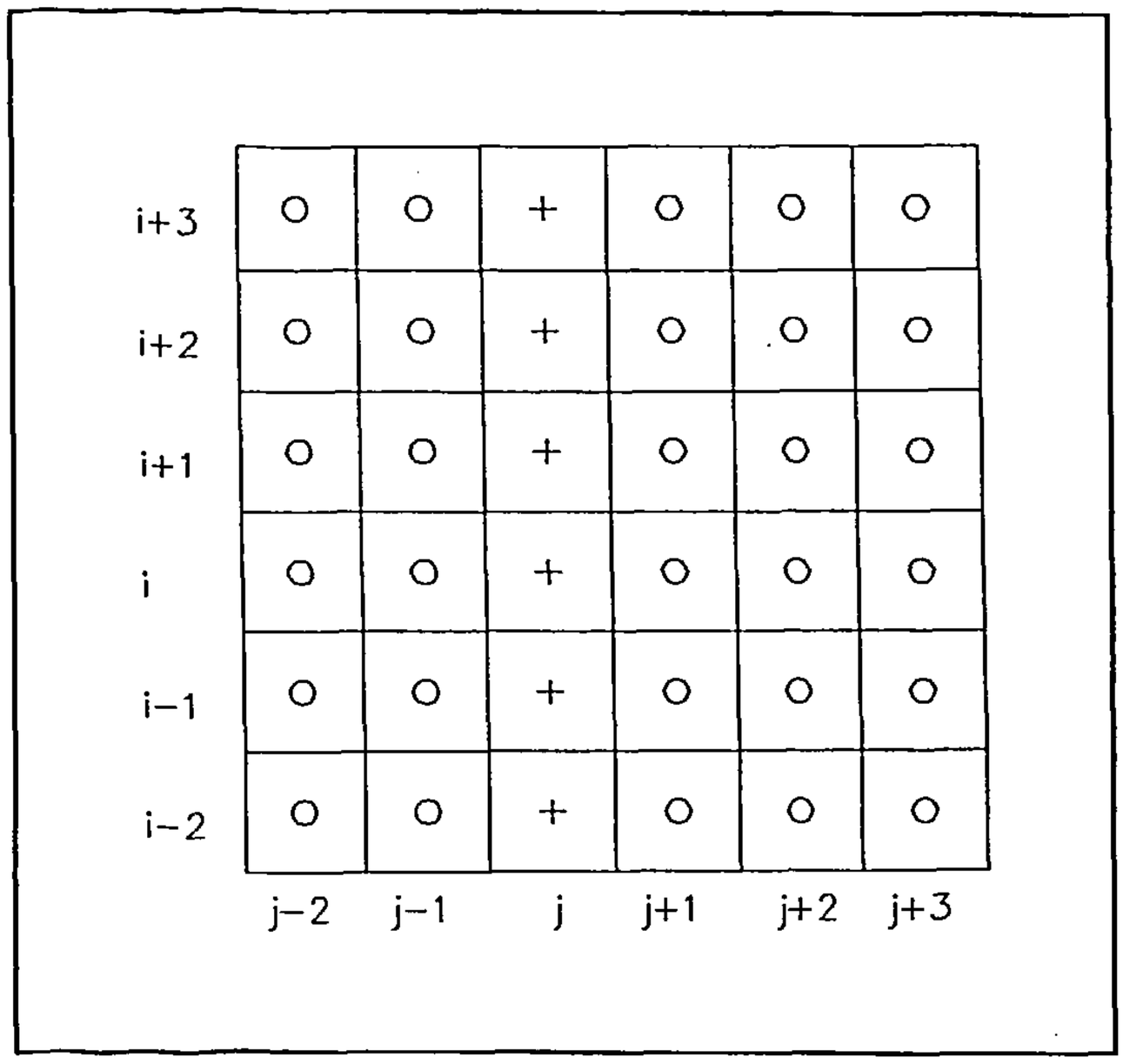


Figure 7 - 'line-by-line' method

Referring to the crossed grid points in Figure 7 , an iterative technique is employed whereby the grid point values of the lines adjacent to the line being solved are taken to be the previous iteration values. In this way, the TDMA is swept across successive lines throughout the calculation domain until convergence is achieved for all grid point values.

In the three-dimensional situation, successive planes are solved throughout one of the dimensions, e.g. the X-Y plane may be solved along the Z-dimension in successive sweeps. In a similar fashion to the two-dimensional problem, grid point values for planes adjacent to the plane being solved are taken to be the previous iteration values.

3.2.10 The Non-Uniform Cartesian Grid

Although the numerical procedure described in this section is applicable to general orthogonal curvilinear coordinate systems, the additional calculations required for various distances, areas and volumes require a substantial effort. Thus, despite the economies offered in terms of computer storage requirements, a regular Cartesian system has been adopted due to ease of problem definition coupled with computational economy, Figure 8 indicates the convention adopted for the principal axes of the coordinate system in relation to the origin of the calculation domain.

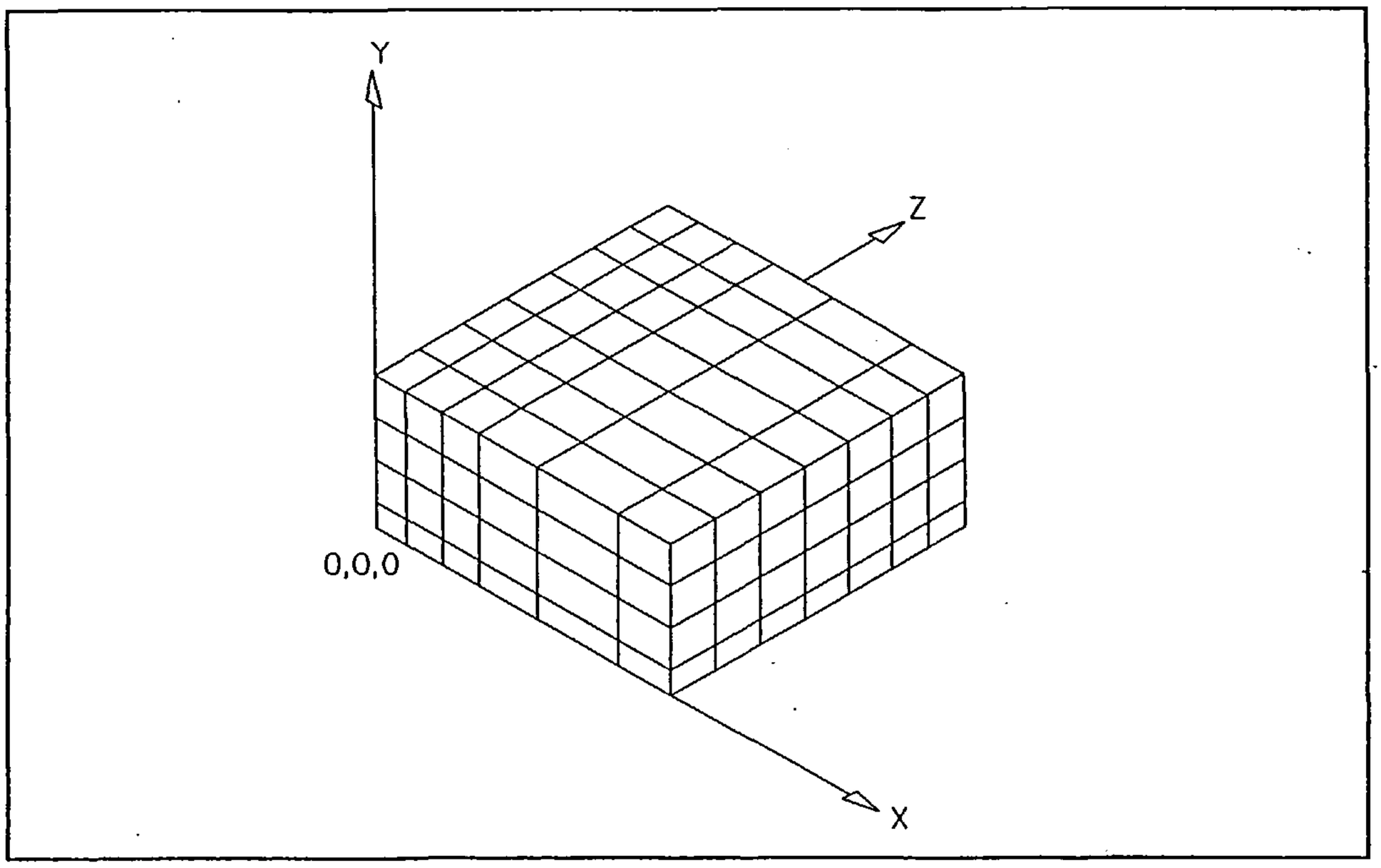


Figure 8 - origin of finite volume domain in relation to principal axes of coordinate system

In order to cater for geometrically irregular calculation domains, control volumes may be made equivalent to solid volumes using the procedure detailed in section 3.4. Using this method, quite complex geometries may be represented, although obviously curved and angled surfaces must be represented by a series of rectilinear bodies and correspondingly the finer the mesh, the more accurate this representation becomes.

Again, for reasons of computational economy, it is desirable to use a fairly coarse grid where the ϕ - x variation is gradual and to use a fine grid where there are steep ϕ - x gradients or in order to represent the geometric features of air terminal devices, flow path obstructions, sloping surfaces, etc. This is achieved through the employment of a non-uniform grid.

In order to derive control volume interface flow rates and conductances, an appropriate interpolation scheme must be adopted:-

Referring to Figure 9:-

$$\text{Interpolation factor, } f = \frac{1/2\ \Delta x_{j+1}}{\delta x_{j+1/2}} \qquad \text{Eq.3.86}$$

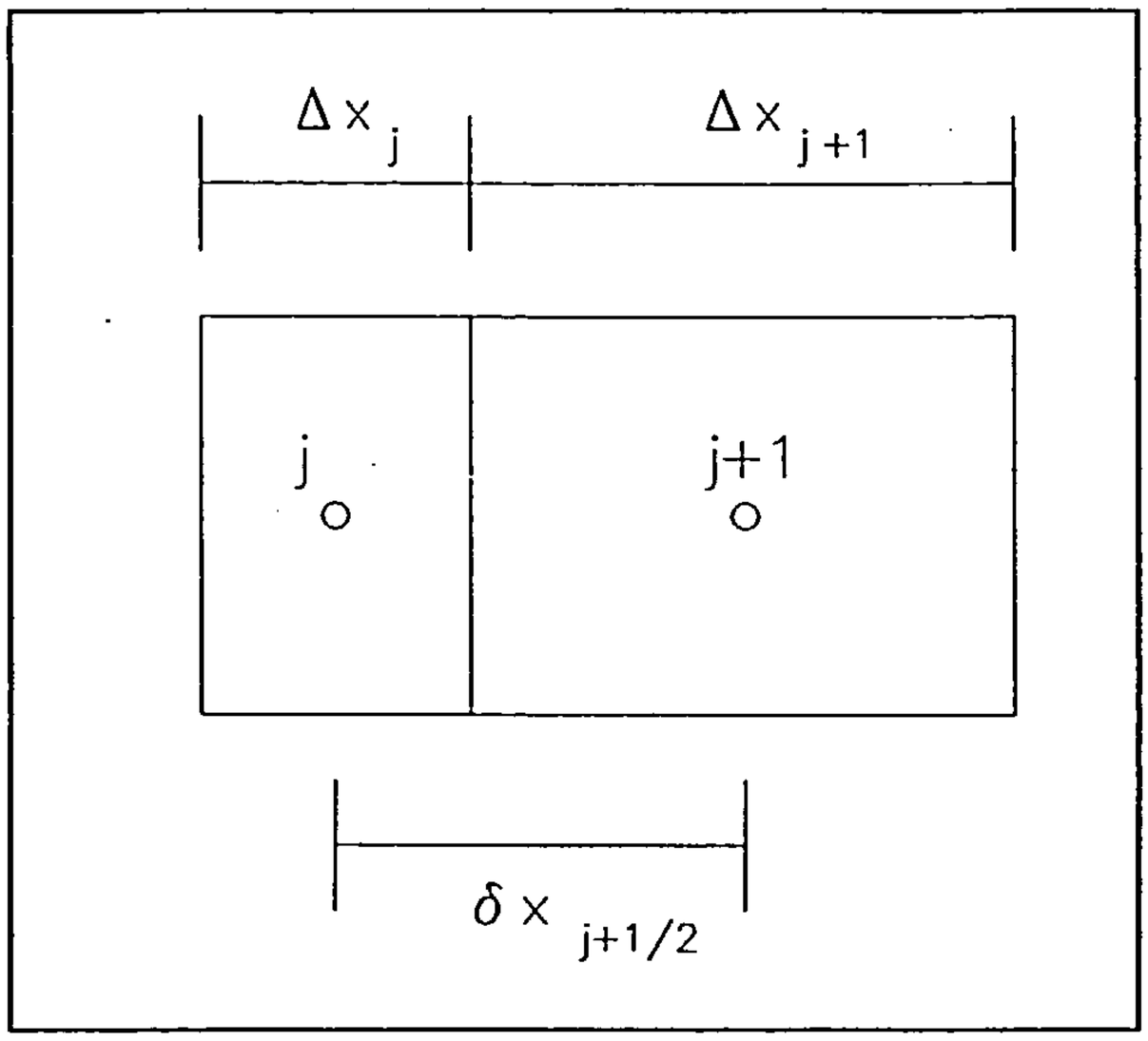


Figure 9 - interpolation scheme

3.2.10.1 Interface Velocities - Linear Interpolation

The interpolation method employed is a conventional linear interpolation, i.e.:-

$$\phi_{j+1/2} = f\,\phi_j + (1\text{-}f)\,\phi_{j+1} \qquad \text{Eq.3.87}$$

3.2.10.2 Interface Diffusion Coefficient (Γ) - Harmonic Interpolation

The interpolation method adopted for interface diffusion coefficients is that described by Patankar [15] whereby an appropriate interpolation may be derived by considering the interface flux which is to be represented:-

$$J_{j+1/2} = \frac{\Gamma_{j+1/2}\,(\phi_j - \phi_{j+1})}{\delta x_{j+1/2}} \qquad \text{Eq.3.88}$$

The appropriate expression for $\Gamma_{j+1/2}$ is the one that reproduces the required $J_{j+1/2}$, i.e.:-

$$J_{j+1/2} = \frac{\Gamma_{j+1/2}\,(\phi_j - \phi_{j+1})}{(1/2\Delta x_j)/\Gamma_j + (1/2\Delta x_{j+1})/\Gamma_{j+1}} \qquad \text{Eq.3.89}$$

This equation is now combined with the interpolation factor:-

$$\Gamma_{j+1/2} = \left(\frac{1\text{-}f}{\Gamma_{j+1}} + \frac{f}{\Gamma_j}\right)^{-1} \qquad \text{Eq.3.90}$$

This practice is known as harmonic interpolation.

3.2.11 Convergence and the Iterative Procedure

Non-linearities and equation interlinkages are dealt with through the iterative nature of the scheme, however the existence of severe non-linearities and strong interlinkages can lead to instabilities in the solution scheme. In order to alleviate such problems, the method of underrelaxation is adopted whereby the dependent variables at each iteration take on a certain percentage of the previous iteration values.

If the current iteration grid point value of the dependent variable is calculated to be ϕ_P' and the previous iteration value is ϕ_P^*, then the current iteration value is derived from:-

$$\phi_P = \phi_P^* + \alpha (\phi_P' - \phi_P^*) \qquad [\alpha < 1.0] \qquad \text{Eq.3.91}$$

where α is the underrelaxation factor. By using very small values of α (0.1), only very slow changes in ϕ are allowed.

Where significant buoyancy effects occur, considerable underrelaxation may be required due to the conventional de-coupling of the energy/momentum equations. In order to improve the convergence of strongly buoyant flows, more appropriate methods of energy/momentum equation coupling have been sought. One method proposed by Galpin and Raithby [58] involves a Newton-Raphson linearisation of the discretised equation coefficients, however the improvement in convergence characteristics appears offset by the additional computational requirements and the method has therefore not been implemented in this procedure.

During each iteration, the iterative solution of each of the discretised equation sets is only taken to a partial state of convergence by defining the number of passes made by the solver. At the end of the solver routine, residuals are calculated for each of the grid point variables. Convergence is deemed to have been achieved when the absolute values of all of the dependent variable residuals are less than a specified tolerance. The residuals are determined as follows:-

$$R = \Sigma a_{nb}\, \phi_{nb} + b - a_P\, \phi_P \qquad \text{Eq.3.92}$$

3.3 Boundary Conditions and Sources

3.3.1 Shear Stress at the Wall Boundary

As noted in Section 2.2.4, in the near-wall region, the turbulence Reynolds number, Re_T decreases to a point where molecular viscosity influences the production, destruction and diffusion of turbulence energy and thus the transport equations for k and ε become inapplicable and an alternative method for modelling shear stress must be applied. For the purpose of computational efficiency, the procedure being described employs the wall function method whereby the thin fluid layer adjacent to the wall is assumed to be in a state of local equilibrium in which the shear stress is uniform and the log law of the wall may be applied, resulting in a velocity profile of the following form:-

$$U^* = \frac{1}{\kappa} \ln(E\, y^*) \qquad \text{Eq.3.93}$$

where

$$U^* = U/U_\tau$$

$$U_\tau = \tau_w/(\rho C_\mu^{1/4} k^{1/2})$$

$$y^* = \rho C_\mu^{1/4} k^{1/2} y/\mu_{LAM}$$

the log-law constants κ and E are empirically derived, e.g. the values employed by Ideriah [33], based on the work of Patel, $\kappa = 0.4187$ and $E = 9.793$.

An alternative method which is favoured by several researchers [48][42][43] is the 'power-law' method whereby the velocity profile adjacent to a wall is approximated to a power-law profile:

$$u_t = \left(\frac{y}{\delta}\right)^n$$

Both the above methods have been employed in this study.

3.3.2 Heat Flux at the Wall Boundary

Although specification of heat flux at the wall boundary presents no problem, a specified wall boundary temperature must be transformed to a wall heat flux in a similar fashion to the prescription of wall shear stress. The conventional approach is to apply a one-dimensional Couette flow analysis to the equilibrium near-wall layer, which results in a similar relationship to the velocity profile relationship:-

$$T^* = \sigma_t \, (U^* + P\,[\sigma / \sigma_t]) \qquad \text{Eq.3.94}$$

where T is a dimensionless quantity that represents the resistance of the wall layer to heat transfer. The most widely adopted P function is that proposed by Jayatillaka [36]:-

$$P\,[\sigma / \sigma_t] = 9.24\,\{\,(\sigma / \sigma_t)^{3/4} - 1\}\,\{1 + 0.28 \exp(-0.007\,\sigma / \sigma_t)\} \qquad \text{Eq.3.95}$$

However, this method has been shown to underestimate wall heat fluxes [43][50] and therefore an alternative ad hoc treatment using empirical relationships has been adopted for the purpose of this study.

Empirical relationships for the specification of surface heat transfer coefficients are based on the so-called 'dimensionless' numbers, the relationships taking the following form:-

$$Nu = C\,Re^m\,Pr^n \qquad \text{Eq.3.96}$$

where C, m and n are constants determined from experimental data.

The relationship adopted for the purpose of this study is that resulting from the work of McAdams [59]. The relationship was derived for parallel flow over copper plates at a reference temperature of 21.1 degrees Centigrade:-

$$h_c = 5.678\left[a + b\left(\frac{V}{0.3048}\right)^n\right] \qquad \text{Eq.3.97}$$

where

h_c = surface convection coefficient (W m^{-2} K^{-1})

V = parallel flow velocity (m s^{-1})

a, b and n are constants determined from experimental data. Values of these constants for rough walls are as follows:-

$$\left.\begin{aligned} a &= 1.09 \\ b &= 0.23 \\ n &= 1.00 \end{aligned}\right\} \quad v < 4.88 \text{ m s}^{-1} \qquad \text{Eq.3.98}$$

$$\left.\begin{aligned} a &= 0.00 \\ b &= 0.53 \\ n &= 0.71 \end{aligned}\right\} \quad 4.88 \text{ m s}^{-1} < v \leq 30.48 \text{ m s}^{-1} \qquad \text{Eq.3.99}$$

The velocity term may be adjusted for temperatures other than 21.1 degrees Centigrade using the following relationship:-

$$V_0 = \frac{284.26\ V}{(273.16 + \theta_m)} \qquad \text{Eq.3.100}$$

3.3.3 Turbulence Kinetic Energy k and the Rate of Dissipation of Turbulence Kinetic Energy ε in the Near-Wall Region

In the absence of better information, diffusion of turbulence kinetic energy to the wall is set to zero, see Launder and Spalding [28]:-

$$\frac{\partial k}{\partial x_i} = 0 \qquad \text{Eq.3.101}$$

The length scale in the near-wall region is presumed proportional to the distance from the wall , see Launder and Spalding [28]:-

$$\varepsilon_w = C_\mu^{3/4} k_P / y\kappa \qquad \text{Eq.3.102}$$

Additionally, the source term G_k, employed in the equations for both turbulence kinetic energy and the dissipation rate of turbulence kinetic energy, is modified in order that the velocity gradients parallel to the wall may be extracted from the wall shear stress.

3.3.4 Inflow Conditions

Velocities: specified

Temperatures: specified

Turbulence kinetic energy, k: specified

Dissipation of turbulence kinetic energy, ε: specified

Values for k and ε are taken from Kurabuchi, Fang, and Grot [50]:-

For a straight duct end or a nozzle type jet:

$$k = 0.01 - 0.03\ U_0^2 \qquad l = 0.05 - 0.25\ L_0$$

For a radial diffuser:

$$k = 0.05 - 0.15\ U_0^2 \qquad l = 0.01 - 0.02\ L_0$$

where

U_0 = Area averaged inflow velocity

L_0 = Representative length of intake

$$\varepsilon = C_\mu k^{3/2}/l$$

3.3.5 Outflow Conditions

Velocities: specified

Temperatures: $\dfrac{\partial T}{\partial x_i} = 0$

Turbulence kinetic energy: $\dfrac{\partial k}{\partial x_i} = 0$

Dissipation of turbulence kinetic energy: $\dfrac{\partial \varepsilon}{\partial x_i} = 0$

3.3.6 Air Supply/Extract and Infiltration Processes - the Pressure and Pressure Correction Equations

In the case of building air flow problems, the boundary pressures are usually unknown but the

normal velocities at the boundary are known (normally zero) or may be specified. Thus, the pressure and pressure correction equations may be modified for the near-wall situation, whereby the normal boundary velocities are incorporated directly into the equations rather than using the velocity correction or pseudo-velocity formulae, consequently the boundary pressures are not required.

For problems where all pressure/pressure correction equation boundary conditions are defined in terms of normal velocities, the calculated values of pressure will be relative to each other because there is no reference pressure. Where absolute values of pressure are required or pressure itself is to be employed as the boundary condition, the appropriate grid point pressures are set equal to the required values. The pressure correction at those grid points is then set to zero and the pressure/pressure correction equations are solved for the *remaining* cells.

Where infiltration processes are considered critical, a two-tier process may be adopted in order to determine the correct boundary conditions for the pressure/pressure correction equations. Air flow around the building may be calculated first by defining the building as an enclosure type polyhedron (see section 4.1.1) and employing the appropriate wind direction and mean wind velocity as the boundary conditions. From this initial calculation, surface pressures may be derived which are used as the pressure boundary conditions for a more refined building model.

3.3.7 Casual Convective Heat Gains from Occupants, Lighting and Equipment

A significant amount of heat is generated in buildings by artificial lighting, miscellaneous equipment (computers, typewriters, etc.) and the occupants themselves. Casual heat gains may be introduced to the energy equation directly, by means of the energy equation volumetric source term:-

$$\frac{\partial}{\partial t}(\rho T) + \frac{\partial}{\partial x_j}(\rho u_j T) = \frac{\partial}{\partial x_j}\left\{\left(\alpha + \frac{\mu_t}{\sigma_t}\right)\frac{\partial T}{\partial x_j}\right\} + S_T \qquad \text{Eq.3.103}$$

where S_T = volumetric rate of heat generation

With a knowledge of the control cells volumes, the appropriate volumetric rate of heat generation may be specified for any given location, thus enabling the correct net heat gain to be introduced to the energy equations for the relevant cells.

3.3.8 Shortwave Radiation - Solar Gain

Shortwave radiation may be introduced to the energy equation in a similar manner to casual

convective heat gain. Having established the instantaneous net surface flux arising from solar heat gain, this may be included in the volumetric source term of the appropriate cells by first converting it into a volumetric flux. Refer to Chapter 4 for a more detailed description.

3.3.9 Incorporation of Longwave Radiative Heat Exchange Within the Energy Equation Source Term

Longwave radiative flux may be incorporated within the finite volume cell energy equation source terms in a similar fashion to casual heat gains. However, the dependence of the source term on the cell temperature T may be exploited in terms of convergence characteristics (i.e. improved convergence) through a suitable linearisation. As previously noted in Section 3.2.2, the source term S may be expressed as $S_C + S_P\phi_P$ in order to account for the fact that S may be a function of ϕ. Patankar [15] suggests that the optimum source term linearisation is the one that makes the straight line $S = S_C + S_P\phi_P$ a tangent to the S-ϕ curve. If a smaller magnitude of S_P is employed, this results in a low estimate of the decrease in S with an increase in ϕ, whereas too large a magnitude results in prolonged convergence due to excessive underrelaxation.

Thus, a source term having the general form $S = A - B\phi^n$ may be linearised as follows:-

$$S = S^* + \left(\frac{\partial S}{\partial \phi}\right)^* (\phi - \phi^*) \qquad \text{Eq.3.104}$$

where S^* is the previous iteration value of S.

Therefore,

$$S = A - B\phi^{*n} - nB\phi^{*n-1} (\phi - \phi^*)$$

Extracting the variable dependent component of this expression:-

$$S_P = -nB\phi^{*n-1} \qquad \text{Eq.3.105}$$

and

$$S_C = A + B\phi^{*n} (n - 1) \qquad \text{Eq.3.106}$$

Thus, expressions for radiative flux which contain fourth powers may be linearised using this procedure prior to inclusion within the energy equation source terms.

3.4 Heat Conduction Through Solid Regions

An important component of heat transfer processes in buildings is transient heat conduction into and out of external walls and internal partitions. Under transient conditions, the rate of heat transfer will depend on incident solar radiation, incident longwave radiation and time variant adjacent air temperatures.

Transient three-dimensional heat conduction through solids may be described by the Fourier equation:-

$$\frac{1}{\alpha}\frac{\partial T}{\partial t} = \frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} + \frac{\partial^2 T}{\partial z^2} \qquad \text{Eq.3.107}$$

or in Cartesian-Tensor notation:-

$$\frac{\partial T}{\partial t} = \frac{\partial}{\partial x_j}\left(\alpha \frac{\partial T}{\partial x_j}\right) \qquad \text{Eq.3.108}$$

It will be noticed that Eq.3.108 is a contraction of the time-averaged energy equation under zero velocity conditions:-

$$\frac{\partial}{\partial t}(\rho T) + \frac{\partial}{\partial x_j}(\rho u_j T) = \frac{\partial}{\partial x_j}\left\{\left(\alpha + \frac{\mu_t}{\sigma_t}\right)\frac{\partial T}{\partial x_j}\right\} \qquad \text{Eq.3.109}$$

Thus, the calculation domain may be extended over both fluid and solid regions comprising the internal air spaces, internal partitions and external walls. When solving the momentum equations for control volumes in the solid regions, the grid point coefficients may be set equal to a very large number which will ensure that the velocities become of negligible value. Subsequently, when solving the energy equation, the diffusion coefficients for the fluid domain may be derived from the laminar/turbulent viscosities (using the Reynolds analogy), whilst the diffusion coefficients for the solid domain may be derived from the thermophysical properties of the appropriate material. The problem may then be solved as a convection-conduction problem throughout the calculation domain, however velocities and therefore Peclet numbers are set to zero in the solid regions and the time averaged energy equation becomes equivalent to the Fourier equation. Heat flux at the fluid/solid interface is derived from a wall function (see Section 3.3.2).

3.5 Conclusions

a) The partial differential equation set describing advection, convection and conduction processes, incorporating the k-ε turbulence model transport equations has been presented and the development of these equations into the final mathematical model has been described. Details of the numerical technique SIMPLER used to solve the defining equation set have been provided.

b) Methods for prescribing the thermal, momentum and turbulence boundary conditions have been described. A general method for incorporating radiation heat exchange within the same solution scheme as developed for advection, convection and conduction processes has also been presented.

c) A method for handling transient three-dimensional heat conduction in solid regions, using the same numerical procedure as for fluid regions has been discussed and the convenience of this scheme demonstrated.

d) A general method for incorporating radiation heat exchange within the same solution scheme as developed for advection, convection and conduction processes has also been presented.

4.0 SHORTWAVE AND LONGWAVE RADIATION PROCESSES

The preceding chapter has demonstrated a numerical method for the calculation of transient turbulent advection, convection and conduction heat transfer in buildings, through the discretisation of the three-dimensional forms of the mass, momentum and energy equations. It has also been demonstrated that radiative heat flux may be incorporated within the energy equation source terms, in the form of a volumetric flux. This chapter considers a detailed treatment of shortwave and longwave radiation heat transfer in buildings, the derivation of net surface radiative flux and the transformation of this surface flux into a volumetric flux for incorporation within the finite volume solution scheme.

In the context of this study, longwave radiation refers to the thermal electromagnetic radiation emitted by a surface as a result of its temperature. Shortwave radiation is also a form of thermal radiation although having a particular wavelength distribution (between approximately 0.2μm and 2.2μm).

This chapter should be read in conjunction with Appendix I which provides details of the various geometric procedures referred to throughout.

4.1 Geometrical Considerations

Radiation heat transfer processes in buildings are generally planar surface phenomena. Thus, a method must be adopted that will enable planar polygonal surfaces to be defined and manipulated within the same Cartesian coordinate system as the finite volume scheme described in Chapter 3.

The method adopted for this study is commonly used in the field of computer graphics and has been described by Mortenson [61]. The method involves the definition of a planar polyhedron in terms of the coordinates of the vertices bounding each polygonal surface. Thus, a list of vertex coordinates may be established for each polyhedron and subsequently a list of vertex sequences defining closed loops of edges bounding the faces may be generated. Figure 10 illustrates the description of a polyhedron using this method. Note that a last-to-first edge is implied in the surface vertex sequences.

Having established an ordered vertex sequence for the surfaces comprising a polyhedron, various surface quantities may subsequently be derived. Considering any of the constituent polygons illustrated in Figure 10, the polygon area may be derived from an algebraic trapezoidal summation

(see Foley and Van Dam [62]):-

$$AREA = (x_p^2 + y_p^2 + z_p^2)^{1/2} \qquad \text{Eq.4.1}$$

where

$$\left.\begin{aligned} x_p &= \sum_{i=1}^{n} (z_i y_j - y_i z_j) \\ y_p &= \sum_{i=1}^{n} (x_i z_j - z_i x_j) \\ z_p &= \sum_{i=1}^{n} (y_i x_j - x_i y_j) \end{aligned}\right\} \quad \begin{aligned} &j = i + 1; && i < n \\ &j = 1; && i = n \\ &n = \text{number of vertices} \end{aligned}$$

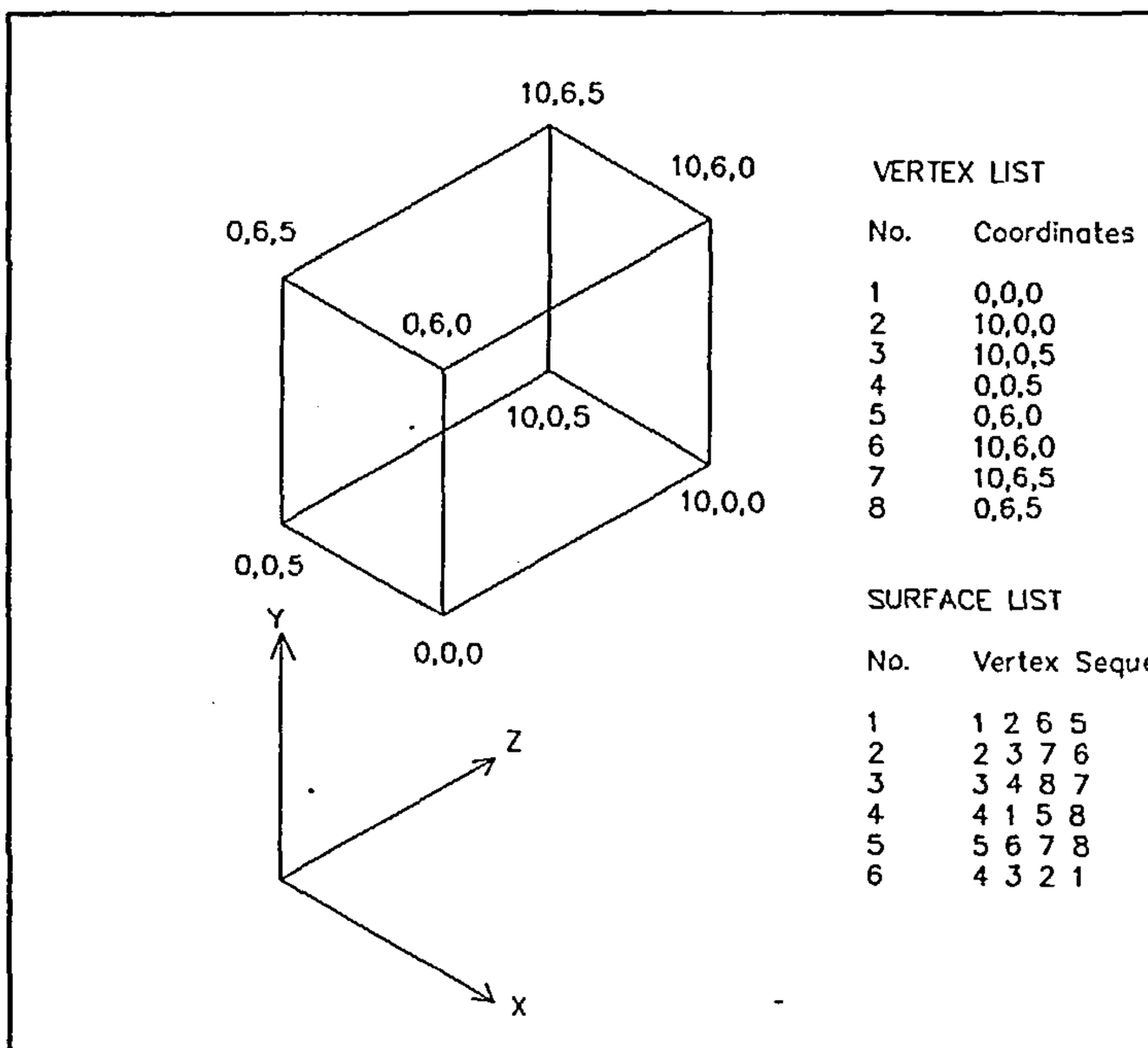


Figure 10 - ordered vertex description of polyhedra

For angular transformations that are required for various geometrical surface operations, the orientation of each polygonal surface is required. Surface orientation is defined using the normal convention of azimuth (α) and elevation (β) angles. Here the azimuth angle is defined as the clockwise angle between a projection of an outward facing surface normal onto the X-Z plane and the principal Z-axis of the finite volume Cartesian coordinate system (see Figure 9). Similarly, the elevation angle is defined as the angle between an outward facing normal to the surface and the X-Z

plane. Azimuth and elevation angles, as defined (for anti-clockwise surface vertex sequences) are illustrated in Figure 11 and may be derived from the following expressions,:-

$$\begin{cases} \alpha_p = \tan^{-1}(x_p / z_p); & z_p \neq 0 \\ \alpha_p = -90; & z_p = 0 \text{ and } x_p < 0 \\ \alpha_p = 0; & z_p = 0 \text{ and } x_p = 0 \\ \alpha_p = 90; & z_p = 0 \text{ and } x_p > 0 \end{cases}$$

$$\begin{cases} \beta_p = \tan^{-1}[y_p / (x_p^2 + z_p^2)^{1/2}]; & x_p^2 + z_p^2 \neq 0 \\ \beta_p = -90; & x_p^2 + z_p^2 = 0 \text{ and } y_p < 0 \\ \beta_p = 0; & x_p^2 + z_p^2 = 0 \text{ and } y_p = 0 \\ \beta_p = 90; & x_p^2 + z_p^2 = 0 \text{ and } y_p > 0 \end{cases}$$

where x_p, y_p and z_p are as defined above

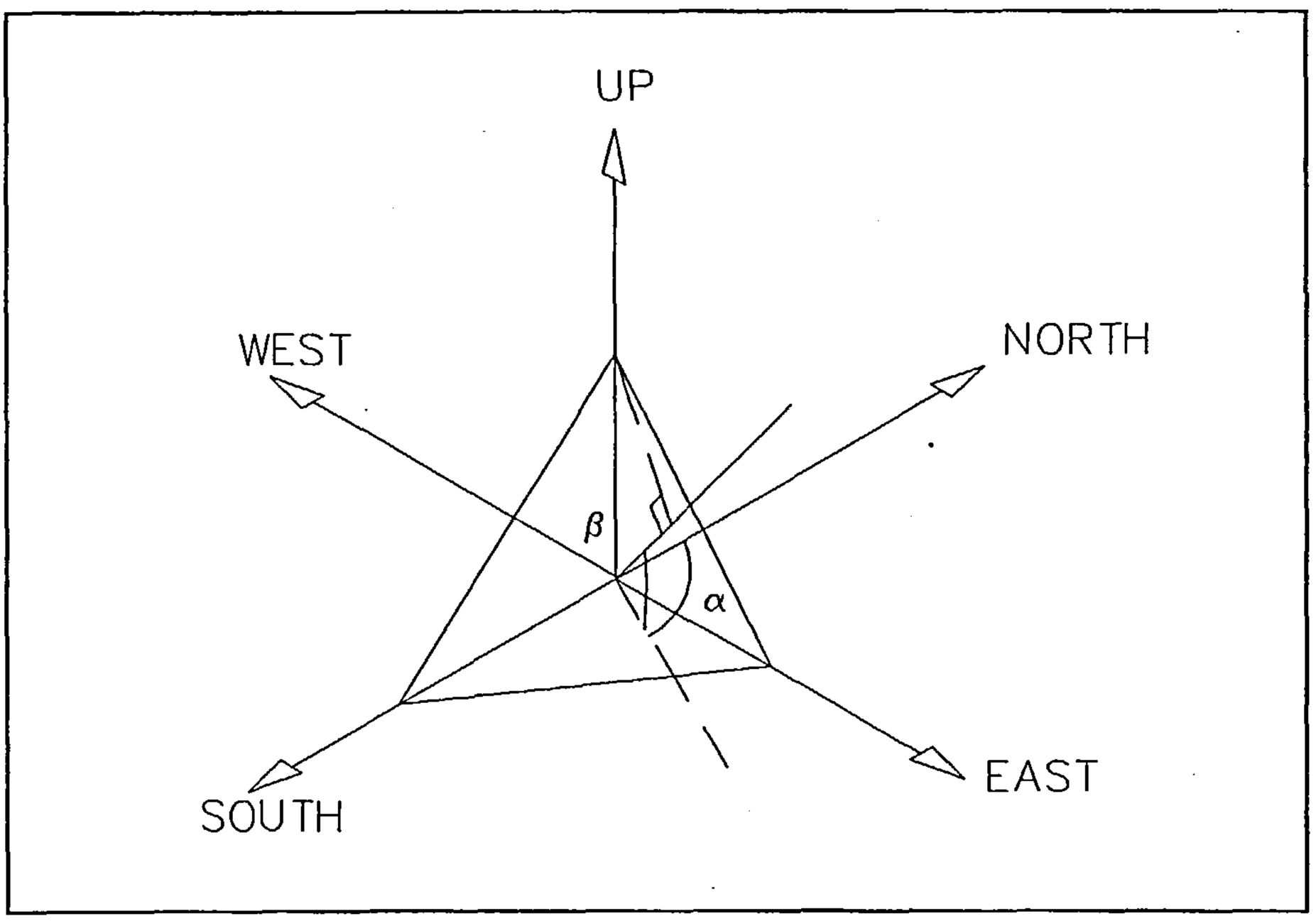


Figure 11 - azimuth and elevation angles

4.1.1 Surface Polyhedron Types

As already noted, the outward facing normal to a surface may be made to point in the opposite direction by reversing the surface vertex sequence. Thus, external surface polyhedra and internal

surface polyhedra may be defined simply by adopting a bounding vertex sequence convention; anti-clockwise for surfaces pointing outwards from the polyhedron and clockwise for inward facing surfaces. Using this convention, various polyhedron types may be established in order to enable accurate calculation of radiative heat exchange in addition to facilitating the definition of the finite volume domain boundaries and associated thermophysical properties. The various polyhedron types are illustrated in Figure 12 and described as follows:-

4.1.1.1 External polyhedra

Used to define external building surfaces in order to account for external insolation and heat transfer with the external environment.

4.1.1.2 Internal polyhedra

Used to define internal building surfaces in order to account for internal radiative exchanges and to define internal zonal boundaries.

4.1.1.3 Interface polyhedra

Polyhedra that are nested between external and internal polyhedra in order to define multi-layered constructions and therefore have no exposed surfaces. Interface polyhedra have the same vertex ordering as external polyhedra by convention, in order to ensure the correct surface matching for the identification of connections (see Section 4.1.3).

4.1.1.4 Enclosure polyhedra

Enclosure polyhedra are similar in function to internal surface polyhedra except that the surfaces face outwards from the interior of the polyhedron. This type of polyhedron may be used to define building spaces that are enclosed by other building spaces.

4.1.1.5 Obstacle polyhedra

Obstacle polyhedra are used to simulate the effect of structural columns, furniture, etc. which may be located in the flow domain. Obstacle polyhedra are similar to enclosures except that all cells enclosed by the polyhedra are static.

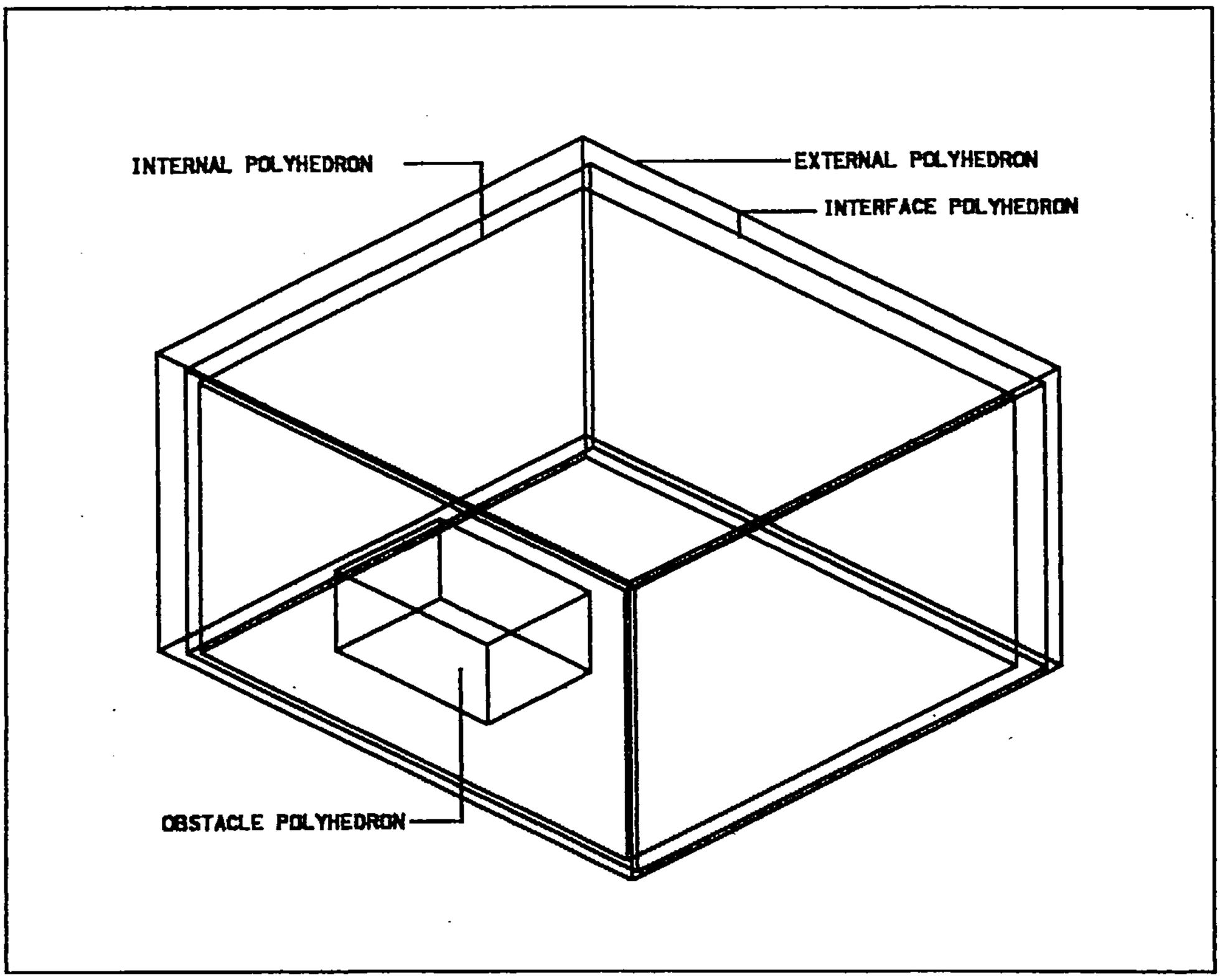


Figure 12 - surface polyhedron types

4.1.2 Polygonal Surface Definition and the Finite Volume Grid

Most building geometries exhibit a high degree of rectilinearity and this may be exploited in terms of the efficiency with which the finite volume grid is generated by defining the building geometry within the finite volume grid using the most appropriate orientation. The actual geographical building orientation may then be arrived at through the definition of an azimuth angle between the geographical North-south axis and the finite volume coordinate system X-Z principal axis. This system may be perceived as the use of a 'local' coordinate system used for the definition of the finite volume grid and associated surface descriptions. This local coordinate system then being orientated with respect to a 'real-world' coordinate system by means of rotation through the site azimuth angle. Both systems share a common arbitrary origin which is located geographically through the site latitude and longitude angles. These coordinate systems are illustrated in Figure 13.

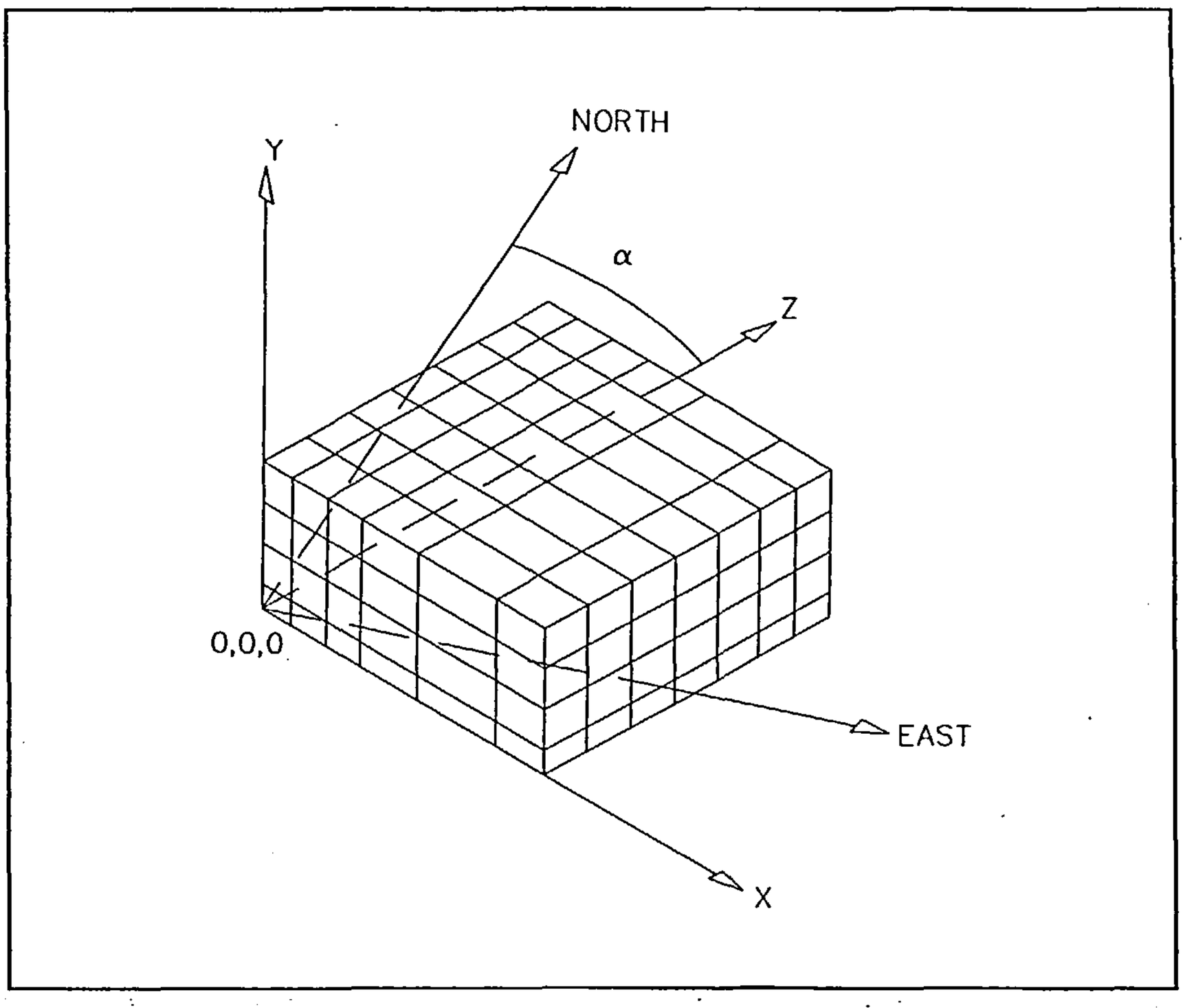


Figure 13 - local and real-world coordinate systems

4.1.3 Identification of Constituent Surface Finite Volume Cells and the Definition of Thermophysical Properties

Volumetric surface radiation flux must be introduced to the energy equations for the correct finite volume cells and similarly, mean surface radiant temperatures must be correctly obtained from the constituent cell temperatures. A method must therefore be sought in order to associate the relevant finite volume cells with each defined building surface.

Considering Figure 14, the surface illustrated may be rotated sequentially about the Y and X axes, using the surface azimuth and altitude angles, in conjunction with the rotation matrices detailed in Appendix I. Thus, the three-dimensional surface is transformed into a two-dimensional surface, parallel with the X-Y plane. Maximum and minimum x-y coordinates may then be derived from the transformed surface vertices and these maxima and minima used to generate a two-dimensional grid extending across the entire surface. Each grid point, subsequent to a test for surface containment, is first translated by a very small distance along the Z-axis and then transformed to the local finite volume coordinate system (by rotation). The transformed grid point will then be located within the correct finite volume cell, adjacent to the surface of interest. The X, Y and Z planes associated with the finite volume cell may then be determined by simple grid containment tests in the X, Y and Z

dimensions.

Having identified all finite volume cells associated with each surface, the total volume occupied by the cells comprising a surface may be calculated. The surface volume is later used to transform surface radiant flux densities into volumetric fluxes for inclusion within the energy equations for the appropriate cells.

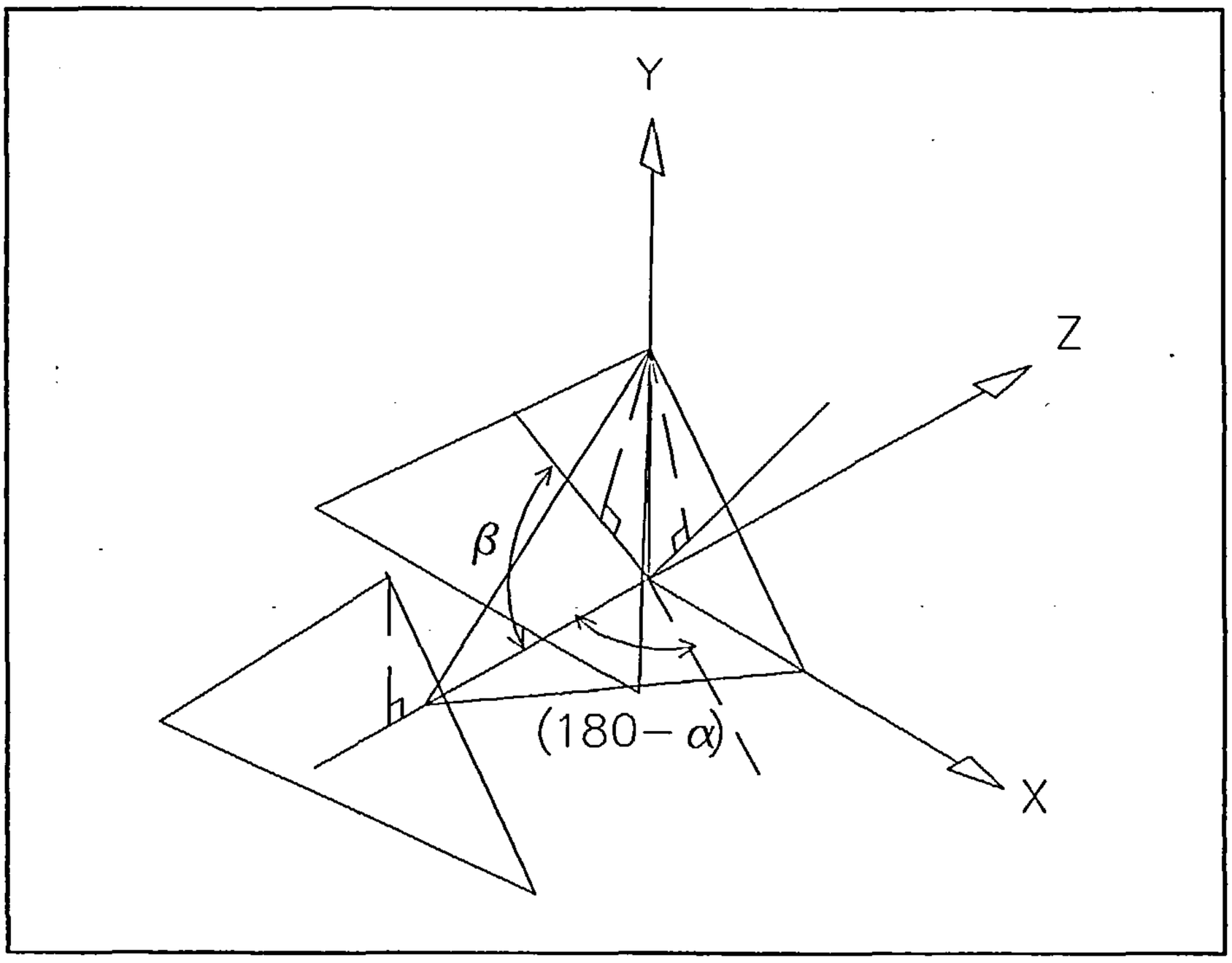


Figure 14 - transformation of 3D surface to 2D surface

A similar procedure is adopted in order to determine which finite volume cells lie in solid regions and to subsequently assign the correct thermophysical properties to each cell. Within the problem data set, a set of surface-to-surface connecting material types are established, each material type being referenced by an integer which points to a record in the materials database file. Thus, using the same procedure as for surface identification, the connecting surfaces may be transformed into two-dimensional planes and a grid generated across the surfaces, however in this case, the grid is 'marched' from one surface to the other along the *Z*-axis in order to identify all finite volume cells between the surfaces. Multi-layered constructions may be defined using interface polyhedra. In the case of multi-layered constructions, where nested interface polyhedra are employed, a convention is employed whereby the connections between the interface polyhedra are defined from the outside moving inwards. This convention ensures that the inner connecting polyhedron will lie behind the outer connecting polyhedron with respect to the principal X-Y plane, subsequent to rotations.

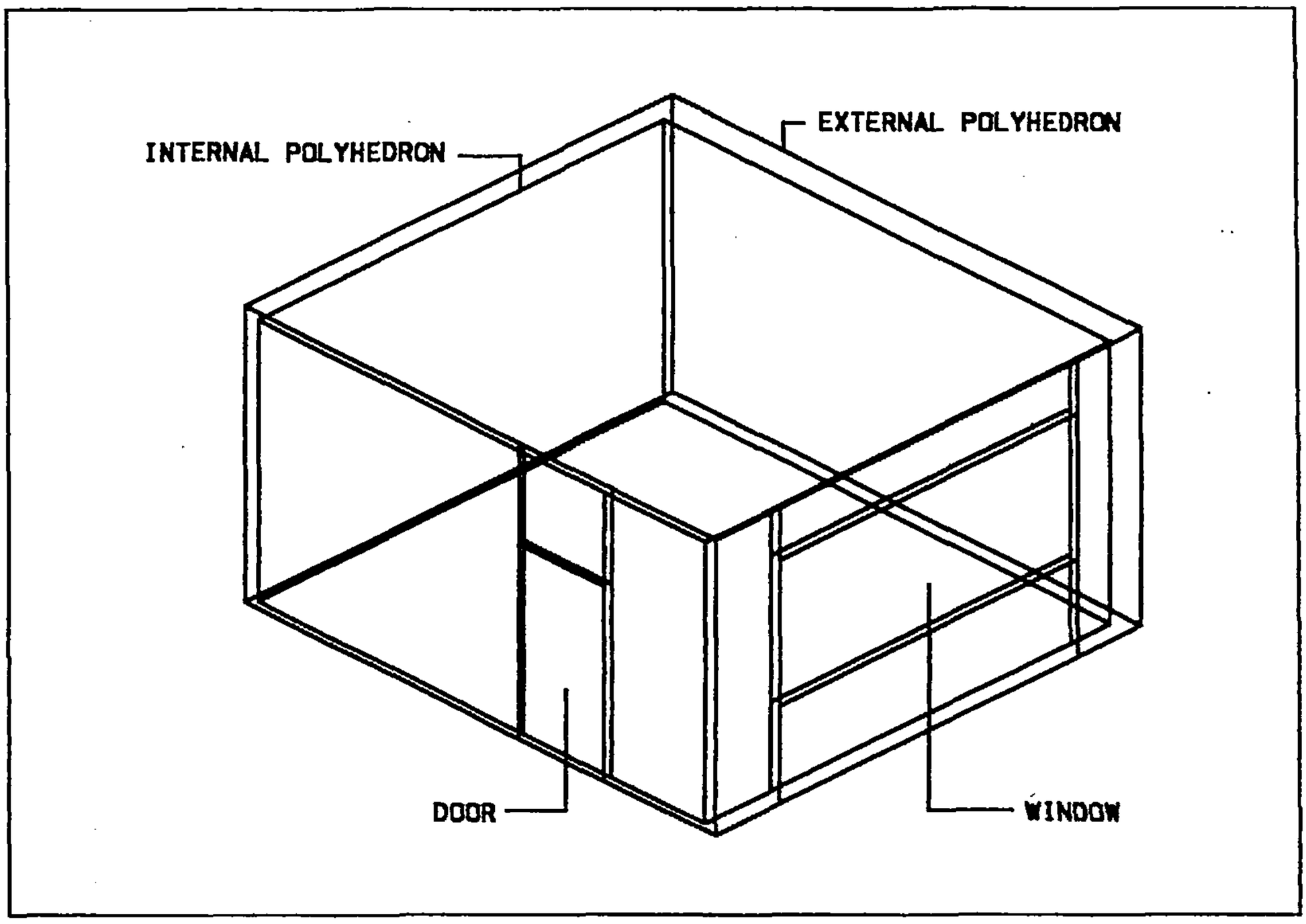


Figure 15 - definition of surface doors and windows

Windows and doors are defined by identifying the surfaces between which the door or window lies. Thus, building surfaces must be subdivided in order to enable the correct geometry of doors and windows to be established, however this will normally be required in order to achieve a reasonably accurate representation of local surface-to-surface radiative exchange (see Section 4.3). Surface subdivision for the purpose of window definition is illustrated in Figure 15. Finite volume cells lying within doors and windows are identified using the same 'marching' grid procedure as described above for the identification of surface-to-surface connections.

4.2 Shortwave Radiation Processes

Direct and diffuse solar radiation incident on an opaque external building facade will be partially reflected and partially absorbed. The absorbed component will tend to elevate the surface temperature, causing an increase in convective heat transfer to the outside and also an increase in conductive heat transfer towards the internal surface of the building.

If the external facade of the building incorporates a transparent aperture such as a window, a proportion of the incident direct and diffuse shortwave radiation will pass through the aperture. The direct component will take the form of the window area projected onto an internal surface, thus forming an irradiated 'patch', the location of this patch being time-dependent and determined by

solar position. In the case of an opaque internal surface, this irradiated patch will be partially reflected and partially absorbed, whilst in the case of a transparent internal surface, the patch will also be partially transmitted. Internal irradiated patches may result in significant increases in local surface temperature. This rise in local surface temperature will result in an increase in conductive heat transfer towards adjacent surfaces in addition to an increase in convective heat transfer to the contained air volume thus giving rise to differential buoyancy forces. In this study, the internally reflected component of direct solar radiation is assumed to be diffusely reflected. The procedures for calculating direct and diffuse solar radiation incident on building surfaces and determining the geometrical location of time-dependent directly irradiated internal solar 'patches' are described in detail in this section.

4.2.1 Solar Angles

In order to calculate the intensity of direct and diffuse solar radiation, incident on sloping surfaces at any particular time, it is necessary to determine the position of the sun. Solar position is normally defined in terms of two angles, the solar azimuth (a) and the solar altitude (z), both of which may be derived from the so-called 'basic earth angles'. Solar azimuth, altitude and the basic earth angles are illustrated in Figure 16. Mathematical derivations of the solar angles may be found in several texts (e.g. Duffie and Beckman [63]) and are consequently merely stated as follows:-

$$a = \sin^{-1} [\cos (L) \cos (d) \cos (h) + \sin (L) \sin (d)] \qquad \text{Eq.4.2}$$

$$z = \tan^{-1} \{\sin (h) / [\sin (L) \cos (h) - \cos (L) \tan (d)]\} \qquad \text{Eq.4.3}$$

where

a = solar altitude

z = solar azimuth

L = site latitude (north positive, south negative)

h = solar hour angle = $15 \left| 12 - t_s \right|$, t_s = solar time

d = solar declination = 23.45 sin(280.1 + 0.9863 Y), Y = year day number (1 - 365)

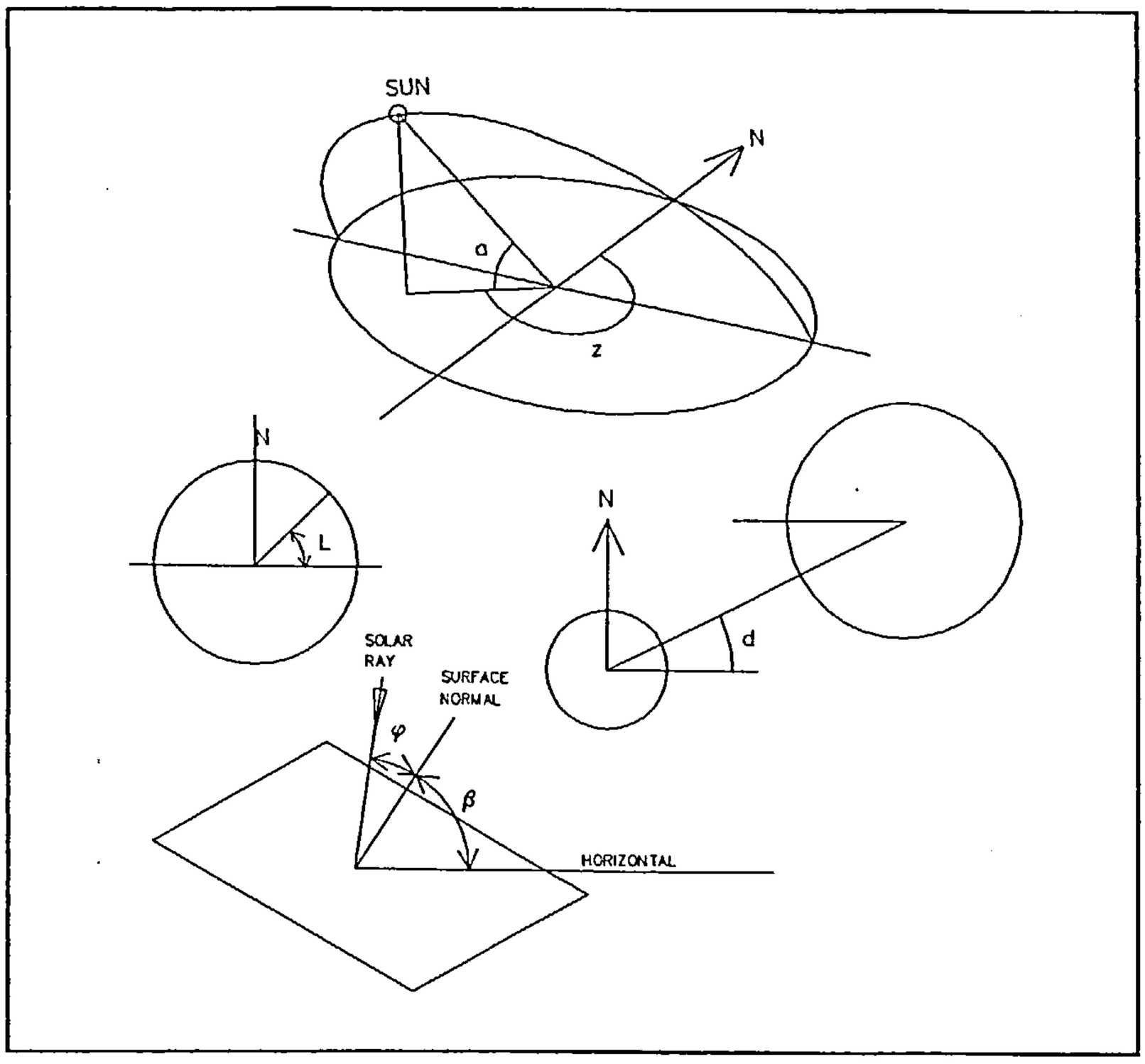


Figure 16 - solar angles

In order to determine the intensity of direct solar radiation incident on a sloping surface, it is necessary to know the angle between an incident solar ray and a normal to the surface, this angle is known as the solar angle of incidence and is given as follows:-

$$\phi = \cos^{-1} [\sin (a) \cos (1 - \beta) + \cos(a) \cos (\omega) \sin (1 - \beta)] \qquad \text{Eq.4.4}$$

where

ω = surface - solar azimuth angle = $|a - \alpha|$, α = surface azimuth angle

β = surface angle of elevation

4.2.2 Intensity of Direct and Diffuse Radiation on Inclined Surfaces

Solar radiation incident on external building surfaces comprises three components; direct, sky diffuse and ground reflected. The direct component is simply derived from the known intensity of the direct component on a horizontal surface, combined with the calculated solar angle of incidence and solar altitude:-

$$I_d = I_{dh} \cos \phi / \sin a \qquad \text{Eq.4.5}$$

The ground reflected component is assumed to be an isotropic diffusely reflected average of direct and diffuse solar radiation incident on the horizontal:-

$$I_{gr} = 0.5 (I_{dh} + I_{diffh}) r \qquad \text{Eq.4.6}$$

where

I_{gr} = diffuse ground reflected radiation (W m^{-2})
r = ground reflectivity

The direct, sky diffuse and ground reflected intensities may then be combined with each external surface area together with the volume occupied by the finite volume cells comprising the surface, in order to derive volumetric fluxes for inclusion within the appropriate finite volume cell energy equations:-

$$q_f = (I A)/V \qquad \text{Eq.4.7}$$

where

I = Intensity (W m^{-2})
A = surface area (m^2)
V = volume occupied by finite volume cells comprising the surface (m^3)

4.2.3 The Projection of Direct and Diffuse Solar Radiation Onto Internal Surfaces Through Windows

Internal air flow and heat transfer is significantly influenced by the size, location and construction of external surface windows, due to the resulting insolation of internal surfaces. It is therefore necessary

to determine the location and magnitude of the proportion of solar radiation incident on an external building surface, that projects through a window in the surface and onto internal building surfaces, as a function of time. The magnitude of the incident direct and diffuse radiation on external surfaces has been dealt with previously.

4.2.3.1 Direct Component

In order to determine the magnitude of the direct component of solar radiation projecting through a window, the incident direct flux is merely modified by the overall transmissivity of the window and combined with the area of the transparent proportion of the window:-

$$Q_d = [I_{dh}\, \tau\, (1 - P)\, A \cos \phi] / \sin a \qquad \text{Eq.4.8}$$

where

P = shading factor

A = area of window

τ = window transmissivity

In order to determine the location of an internal irradiated area arising from projection of the direct solar component through an external window, the following algorithm has been adopted:-

a) Using the same method as described in Section 4.1.3, the external window surface is rotated in order that it is parallel with the X-Y plane and then overlaid with a two-dimensional grid.

b) Subsequent to point containment tests (for non-rectilinear windows), the geometric centre of each grid cell P (x y z), is rotated back to the 'real-world' coordinate system. A unit vector S', is then constructed to lie parallel with the incident solar rays, pointing inwards, towards the interior of the building. The unit solar vector is constructed using the solar azimuth and altitude angles:-

$$[x_{S'}\, y_{S'}\, z_{S'}] = [x_P\, y_P\, z_P] - [\cos(a)\sin(z) \quad \sin(a) \quad \cos(a)\cos(z)] \qquad \text{Eq.4.9}$$

c) The unit solar vector is then scaled up to extend well beyond all building surfaces, in the direction of a solar ray, to result in the solar vector S:-

$$[x_S\, y_S\, z_S] = K\, [x_{S'}\, y_{S'}\, z_{S'}] \qquad \text{Eq.4.10}$$

where K is a very large number

d) The solar vector S is then tested for intersection with each internal building surface plane (including enclosures and obstacles), subsequent to checking for surface visibility. If an intersection occurs, the point of intersection on the surface plane is then tested for containment within the surface boundaries. If the point is contained, the distance between the window grid point and the point of intersection is calculated and stored (i.e. the magnitude of the window-surface vector).

e) For each sequential surface intersection with the solar vector, starting from the window grid point and finishing with the intersection farthest from the window, the associated finite volume cells are identified by simple grid containment tests in the X, Y and Z dimensions. For each intermediate, transparent surface encountered, the incident flux is divided into absorbed, transmitted and reflected components:-

$$Q_a = Q_d P a \qquad \text{Eq.4.11}$$

$$Q_\tau = Q_d P \tau \qquad \text{Eq.4.12}$$

$$Q_r = Q_d P r \qquad \text{Eq.4.13}$$

where P is the area of the grid cell expressed as a proportion of the window area .

The reflected component is added to the accumulated reflected flux associated with the relevant surface, for further processing. Reflected flux is assumed to be diffusely reflected and is apportioned on the basis of enclosure view factors (see Section 4.3.1).

The absorbed flux is incorporated within the associated finite volume cell energy equation source term, in the form of a volumetric flux:-

$$Q_v = Q_a / \Delta x \, \Delta y \, \Delta z \qquad \text{Eq.4.14}$$

where Δx, Δy and Δz are the finite volume cell dimensions in the X, Y and Z dimensions, respectively.

The transmitted flux then becomes the incident flux for the next surface in sequence and this process

continues until the final, opaque surface is encountered.

f) Stages b) - e) are the repeated for all window grid cells.

4.2.3.2 Diffuse component

The transmission of the diffuse component through a window may be calculated on the basis of the following relationship:-

$$Q_{diff} = (I_{diff\beta} + I_{gr\beta})\, A\, \tau \qquad \text{Eq.4.15}$$

The transmitted diffuse component of solar radiation and the direct component subsequent to internal reflection are both distributed over internal building surfaces on the basis of radiation view factor relationships.

4.3 Longwave Radiation Processes

In the context of this study, longwave radiation refers to the thermal electromagnetic radiation emitted by a surface as a result of its temperature. The total energy emitted by a fully absorbing or 'blackbody' surface is proportional to the absolute temperature of the surface raised to the fourth power:-

$$E_b = \sigma T^4 \qquad \text{Eq.4.16}$$

where

E_b = blackbody emissive power (W m^{-2})

T = absolute temperature (K)

σ = Stefan-Boltzman constant = 5.669 x 10^{-8} W m^{-2} K^{-4}

For reflecting or 'nonblackbody' surfaces, an absorptivity may be defined:-

$$\alpha = \frac{E}{E_b} \qquad \text{Eq.4.17}$$

where

E = emissive power of body (W m^{-2})

E_b = blackbody emissive power (W m^{-2})

α = absorptivity

In this study, all building surfaces are assumed to obey the Kirchoff identity, which states that the emissivity of a body is equal to the absorptivity of that body, i.e.:-

$$\varepsilon = \alpha \qquad \text{Eq.4.18}$$

Thus, the emissive power of a nonblackbody becomes:-

$$E = \varepsilon \sigma T^4 \qquad \text{Eq.4.19}$$

4.3.1 Radiation View Factor

Considering two arbitrarily orientated radiating black surfaces as illustrated in Figure 17, the proportion of radiant energy leaving surface 1 and arriving at surface 2 is known as the surface radiation view factor for surfaces 1->2. The source and target surfaces to which the view factor relates are indicated through the use of subscripts whereby the first subscript refers to the source surface and the second subscript refers to the target surface. Thus, the energy leaving surface 1 and arriving at surface 2 is:-

$$q_{12} = E_{b1} A_1 F_{12} \qquad \text{Eq.4.20}$$

and the energy leaving surface 2 and arriving at surface 1 is similarly:-

$$q_{21} = E_{b2} A_2 F_{21} \qquad \text{Eq.4.21}$$

If the surfaces are at the same temperature, the net heat exchange must be zero:-

$$E_{b1} A_1 F_{12} - E_{b2} A_2 F_{21} = 0 \qquad \text{Eq.4.22}$$

and also:-

$$E_{b1} = E_{b2} \qquad \text{Eq.4.23}$$

thus:-

$$A_1 F_{12} = A_2 F_{21} \qquad \text{Eq.4.24}$$

This relationship is known as one of reciprocity and holds for nonblackbodies if the angular distribution of emitted radiation is diffuse.

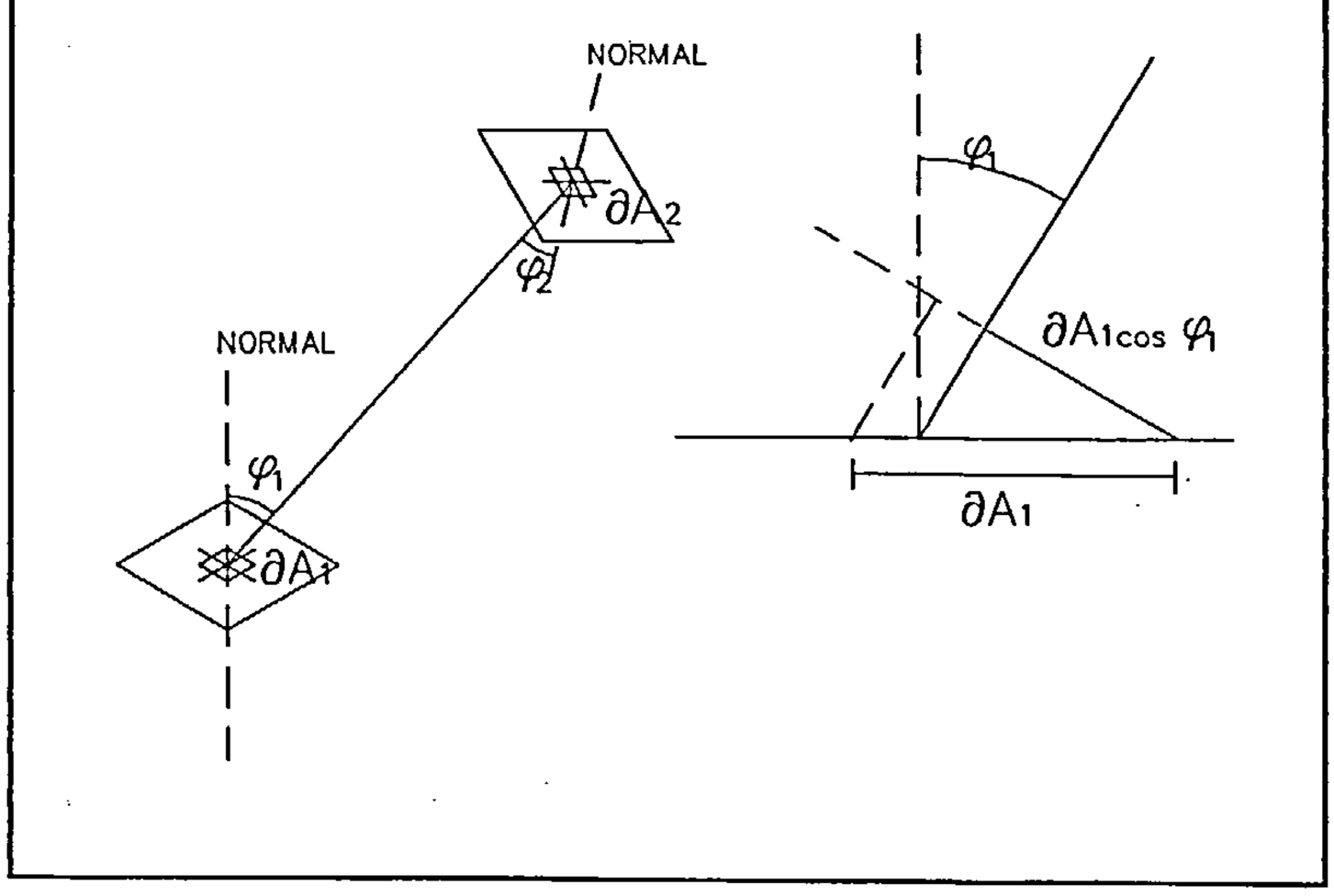


Figure 17 - radiation view factors

Considering Figure 22, the energy leaving dA_1 travelling towards dA_2 at an angle of ϕ_1 to the normal to dA_1 is:-

$$q_{dA1->dA2} = I_b \, dA_1 \cos \phi_1 \qquad \text{Eq.4.25}$$

where I_b is the blackbody intensity, that is the blackbody radiation emitted per unit area and per unit solid angle in a specified direction. Thus, the radiation arriving at an elemental area dA_n at a distance r from A_1 is:-

$$q_{dA1->dAn} = I_b \, dA_1 \cos \phi_1 \frac{dA_n}{r^2} \qquad \text{Eq.4.26}$$

where

$\frac{dA_n}{r^2}$ = solid angle subtended by the area dA_n

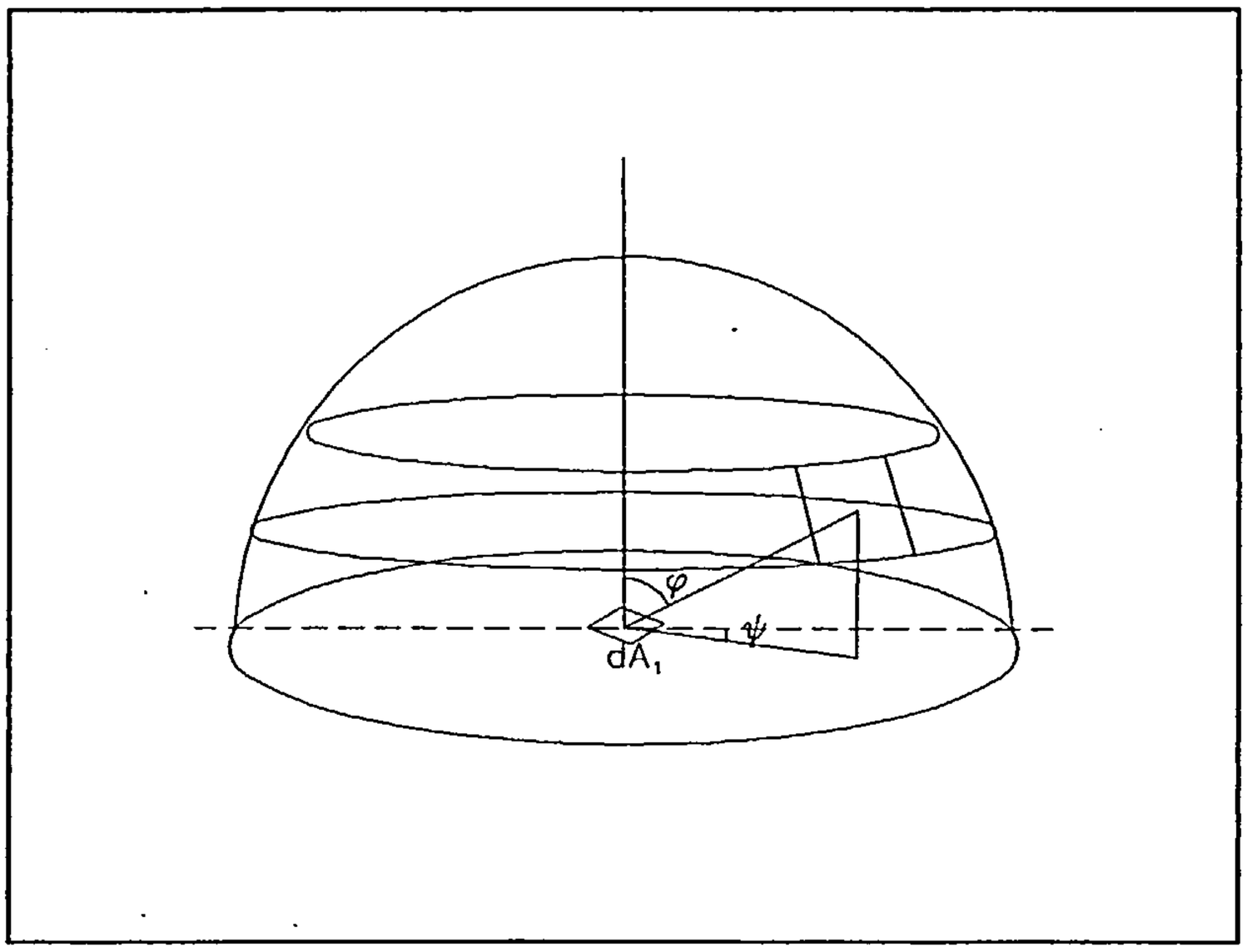


Figure 18 - unit hemisphere

The elemental area dA_n may be expressed in terms of the spherical coordinates illustrated in Figure 18:-

$$dA_n = r \sin\phi \; d\psi \; r \, d\phi \qquad \text{Eq.4.27}$$

Equations 4.26 and 4.27 may be combined and integrated over a hemisphere centred at dA_1, as illustrated in Figure 18, in order to derive the emissive power from dA_1.

Thus,

$$E_b \, dA_1 = I_b \, dA_1 \int_0^{2\pi} \int_0^{2\pi} \sin\phi \cos\phi \, d\phi d\psi = \pi \, I_b dA_1 \qquad \text{Eq.4.28}$$

and therefore:-

$$E_b = \pi \, I_b \qquad \text{Eq.4.29}$$

For the situation illustrated in Figure 17, the elemental area dA_n is given by:-

$$dA_n = \cos\phi_2 \, dA_2 \qquad \text{Eq.4.30}$$

and the energy arriving at dA_2 which is emitted from dA_1 is:-

$$dq_{1->2} = E_{b1} \cos \phi_1 \cos \phi_2 \frac{dA_1 dA_2}{\pi r^2} \qquad \text{Eq.4.31}$$

and similarly, the energy arriving at ∂A_1, which is emitted from ∂A_2 is:-

$$dq_{2->1} = E_{b2} \cos \phi_2 \cos \phi_1 \frac{dA_2 dA_1}{\pi r^2} \qquad \text{Eq.4.32}$$

and therefore the net energy exchange is:-

$$q_{1->2} = (E_{b1} - E_{b2}) \int_{A_2} \int_{A_1} \cos \phi_1 \cos \phi_2 \frac{dA_1 dA_2}{\pi r^2} \qquad \text{Eq.4.33}$$

where the integral is equivalent to $A_1 F_{12}$.

Exact analytical integrations are only practical for simple geometric relationships between participating surfaces and to further complicate the problem, internal building surfaces quite often involve the problem of partial obstruction, whereby surfaces are not wholly visible to other enclosure surfaces. Thus, in order to evaluate the integral for general polygonal surfaces with arbitrary orientation to each other and to allow for the possibility of partial obstruction requires an algorithmic, numerical approach.

One numerical algorithm has been proposed by Moore and Numan [64]. This approach involves the subdivision of a surface into a number of elemental areas, dA_i. A unit hemisphere constructed around dA_i, is then divided into a number of equal solid angles through 'strip and patch' subdivision as illustrated in Figure 23. An elemental view factor may now be defined as the ratio of the radiative flux leaving dA_i that arrives at dA_j to the total radiative flux leaving dA_i:-

$$df_{\partial Ai->\partial Aj} = \frac{dq_i}{q} \qquad \text{Eq.4.34}$$

and from Equations 4.26 and 4.30:-

$$\frac{dq_i}{q} = \frac{\cos \phi_i \, dA_j \cos \phi_j}{\pi r^2} \qquad \text{Eq.4.35}$$

Similarly, the point configuration factor may be defined as the fraction of the radiative energy leaving dA_i which is received by A_j:-

$$c_{ij} = \int_{A_j} \frac{\cos \phi_i \cos \phi_j}{\pi r^2} dA_j \qquad \text{Eq.4.36}$$

Thus, from Equation 4.33:-

$$A_i F_{ij} = \int_{A_j} c_{ij} \, dA_i \qquad \text{Eq.4.37}$$

Returning to the unit hemisphere illustrated in Figure 18, each solid angle may be associated with a viewed polygon, a polygon being defined as 'viewed' if the axis of the solid angle is not intersected by any other polygon. The point configuration factor is then found by summing the contributions of each viewed polygon and the view factor subsequently derived from Equation 4.37.

The fact that this algorithm guarantees that all radiation is theoretically accounted for (i.e. $\sum_j F_{i->j} = 1.0$ may be proved by considering Equation 4.36 which may be rewritten as follows:-

$$c_{ij} = \int_{\Omega_j} \frac{\cos \phi_i}{\pi} d\Omega \qquad \text{Eq.4.38}$$

noting that the area dA_j viewed from dA_i is $\cos \phi_j dA_j$.

Now for $df_{dAi->Aj} = 1.0$, the integral must be:-

$$\int_{\Omega_j} \cos \phi_i \, d\Omega = \pi \qquad \text{Eq.4.39}$$

If all $d\Omega_n$ are the same, the integral may be approximated by the following summation:-

$$\sum_n \cos \phi_n \, d\Omega_n = d\Omega \sum_n \cos \phi_n \qquad \text{Eq.4.40}$$

By definition, for $d\Omega_n$ all to be equal, the areas subtended at the surface of the unit hemisphere must

be equal. The area of a horizontal strip of the sphere is given by:-

$$S = 2 \pi R h \qquad \text{Eq.4.41}$$

and therefore to divide the hemisphere into equal strips requires h to be equal. The subtended areas may now be made equal simply through equal subdivision of the strips.

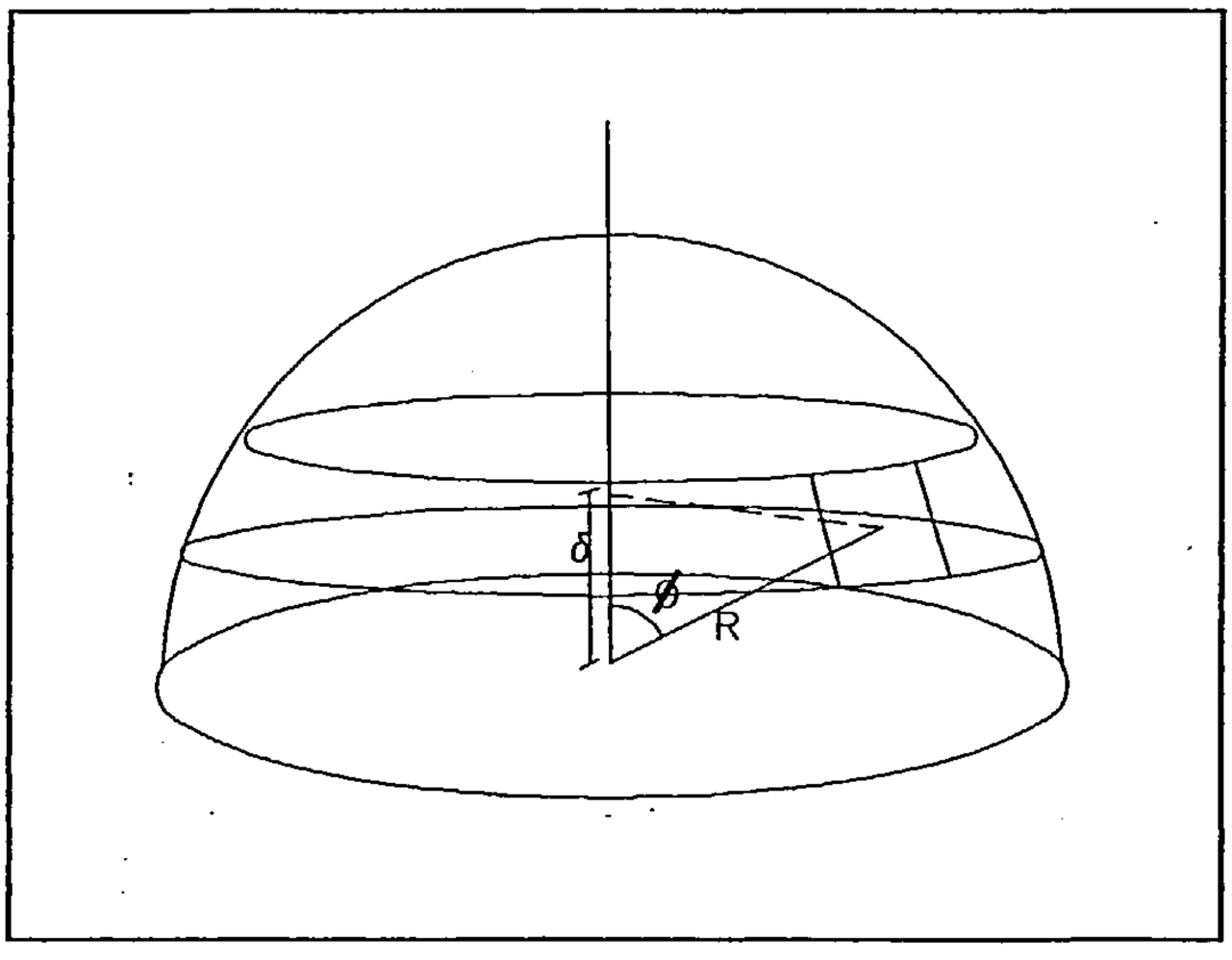


Figure 19 - unit hemisphere

For the unit hemisphere illustrated in Figure 19:-

$$\cos \phi = \delta \qquad \text{Eq.4.42}$$

where δ is the projection of the radius onto the normal. If the hemisphere is divided into n strips, the cosines will be:-

$$\frac{1}{2n}, \frac{1}{2n}+\frac{1}{n}, \frac{1}{2n}+\frac{2}{n}, \ldots\ldots, \frac{1}{2n}+\frac{(n-1)}{n}$$

Thus the summation using the formula for arithmetic series will be:-

$$S_n = (n/2)\,[2a + (n-1)\,b] \qquad \text{Eq4.43}$$

where

$$a = \frac{1}{2n}$$

$$b = \frac{1}{n}$$

Thus,

$$S_n = n/2 \qquad \text{Eq.4.44}$$

If each strip is then subdivided into m equal patches, then the cosine summation will be mn/2. Since each patch subtends a solid angle of $2\pi/mn$ steradians, Equation 4.67 becomes:-

$$(2\pi/mn)\,(mn/2) = \pi \qquad \text{Eq.4.45}$$

and therefore $\sum_{j}^{j} f_{i->j} = 1.0$.

The Moore and Numan method is implemented in the present study using the following algorithm:-

a) The surface, for which view factors are being calculated, is rotated until it is parallel with the principal X-Y plane (see Section 4.1.3) and subsequently overlaid with a two-dimensional grid.

b) The centre of each grid cell is used as the centre of a unit hemisphere and a patch is then traversed around a series of strips into which the hemisphere is divided. The width of the patch and the height of each strip is determined by the division azimuth (α) and altitude(β) angles, as illustrated in Figure 20. Thus the vertex coordinates of each patch may be derived through similar rotations as those described in Section 4.1.2.2:-

$$x_1 = x + \sin\alpha' \cos\beta' \qquad \text{Eq.4.46}$$

$$x_2 = x + \sin\alpha \cos\beta' \qquad \text{Eq.4.47}$$

$$x_3 = x + \sin\alpha \cos\beta \qquad \text{Eq.4.48}$$

$$x_4 = x + \sin\alpha' \cos\beta \qquad \text{Eq.4.49}$$

$$y_1 = y + \cos\alpha' \cos\beta' \qquad \text{Eq.4.50}$$

$$y_2 = y + \cos\alpha \cos\beta' \qquad \text{Eq.4.51}$$

$$y_3 = y + \cos\alpha \cos\beta \qquad \text{Eq.4.52}$$

$$y_4 = y + \cos\alpha' \cos\beta \qquad \text{Eq.4.53}$$

$$z_1 = z_2 = z + \sin\beta' \qquad \text{Eq.4.54}$$

$$z_3 = z_4 = z + \sin\beta \qquad \text{Eq.4.55}$$

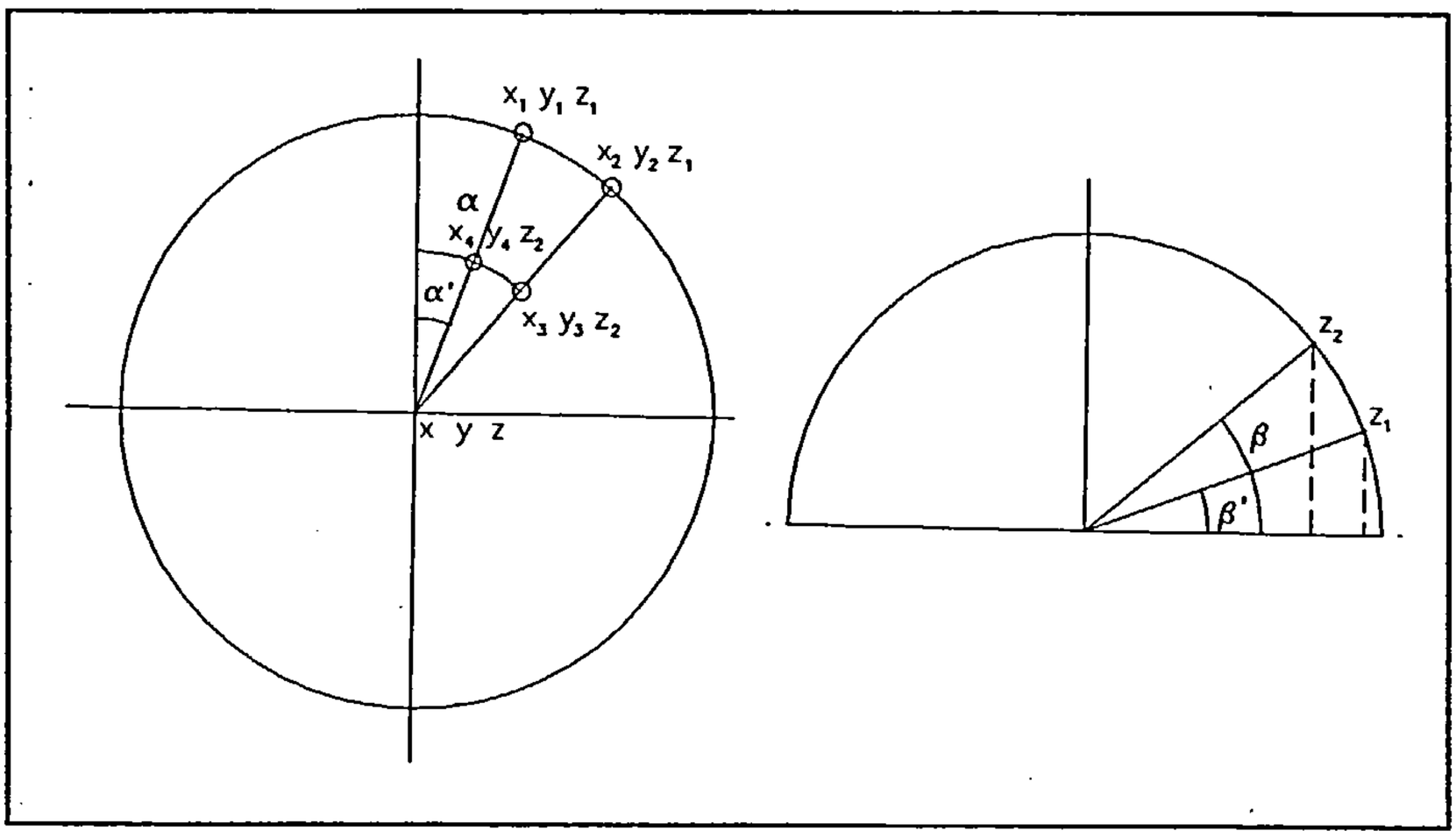


Figure 20 - azimuth and altitude division angles

c) Vectors are then constructed between the patch vertices and the hemisphere centre. The vectors thus derived are then scaled up to extend well beyond all building surfaces:-

$$[a' b' c'] = K\,[a\ b\ c] \qquad \text{Eq.4.56}$$

where K is a very large number.

d) Each scaled vector is then tested for intersection with the plane of all other participating surfaces. If an intersection occurs, the coordinates of the point of intersection are tested for containment within the surface and if found to be contained, the length of the vector between the hemisphere centre and the intersection point is recorded.

e) Process d) is repeated for all surfaces in order to determine the nearest intersection to the hemisphere centre.

f) The cosines $\cos\phi_1$ and $\cos\phi_2$ are calculated for the smallest vector identified in e), using the visibility algorithm detailed in Section 4.1.2.4 and the point configuration factor (Equation 4.36) subsequently calculated and added to the cumulative configuration factor for the particular surface pairing.

g) The processes b) - f) are repeated for all grid cells identified in a) and finally the view factors (Equation 4.37) are calculated.

4.3.2 Radiative Heat Exchange Between Nonblackbody Building Surfaces

Having established a method for determining the geometric view factor for arbitrarily orientated and partially obscured surfaces, the calculation of radiative heat transfer between black surfaces becomes a relatively trivial task since all of the radiant energy incident upon the surfaces is absorbed. The net radiant exchange for the nonblackbody surfaces illustrated in Figure 17 would therefore simply be:-

$$q_{1->2} = F_{12} A_1 (E_{b1} - E_{b2}) \qquad \text{Eq.4.57}$$

However, in the case of building surfaces which are normally nonblackbodies, the situation becomes more complex. Radiant energy received by nonblackbody surfaces will not be entirely absorbed and a proportion of it will be reflected towards other participating surfaces and similarly a proportion of this reflected energy will be reflected by other surfaces and so on. Thus, the solution method must account for these multiple reflections if it is to be deemed accurate.

The conventional approach to the numerical solution of radiation problems of diffuse reflections between nonblackbody surfaces is based on a formulation involving the terms irradiation and radiosity:-

$$J = \varepsilon E_b + \rho G \qquad \text{Eq.4.58}$$

where

G = irradiation = total radiation incident upon a surface per unit time and per unit area

J = radiosity = total radiation which leaves a surface per unit time and per unit area

However, despite the rigour and relative simplicity of the radiosity matrix method, a primary objective of this study is to develop a single solution scheme for conjugate convection, conduction and radiation heat transfer processes. Therefore, a more approximate analytical method, which expresses radiative heat transfer in terms of future time row values of temperature rather than radiosities, is preferred.

As a result of numerical experiments, Clarke [5] has concluded that for a typical building enclosure, approximately 99% of all radiant heat transferred between the surfaces of the enclosure is absorbed after the third reflection. Considering the two arbitrarily orientated nonblackbody surfaces illustrated

in Figure 26, the radiative flux emitted by surface 1 is $\varepsilon_1 A_1 \sigma T_1^4$ and the proportion of this flux received by surface 2 will be $\varepsilon_1 A_1 F_{12} \sigma T_1^4$. Subsequent to the radiative flux arriving at surface 2 from surface 1, a proportion of the incident flux will be reflected back towards surface 1, $(1 - \varepsilon_2) \varepsilon_1 A_1 F_{12} \sigma T_1^4$. Similarly, a proportion of this flux is then reflected back from surface 1, $(1 - \varepsilon_1)(1 - \varepsilon_2) \varepsilon_1 A_1 F_{12}^2 F_{21} \sigma T_1^4$ and this process continues to infinity.

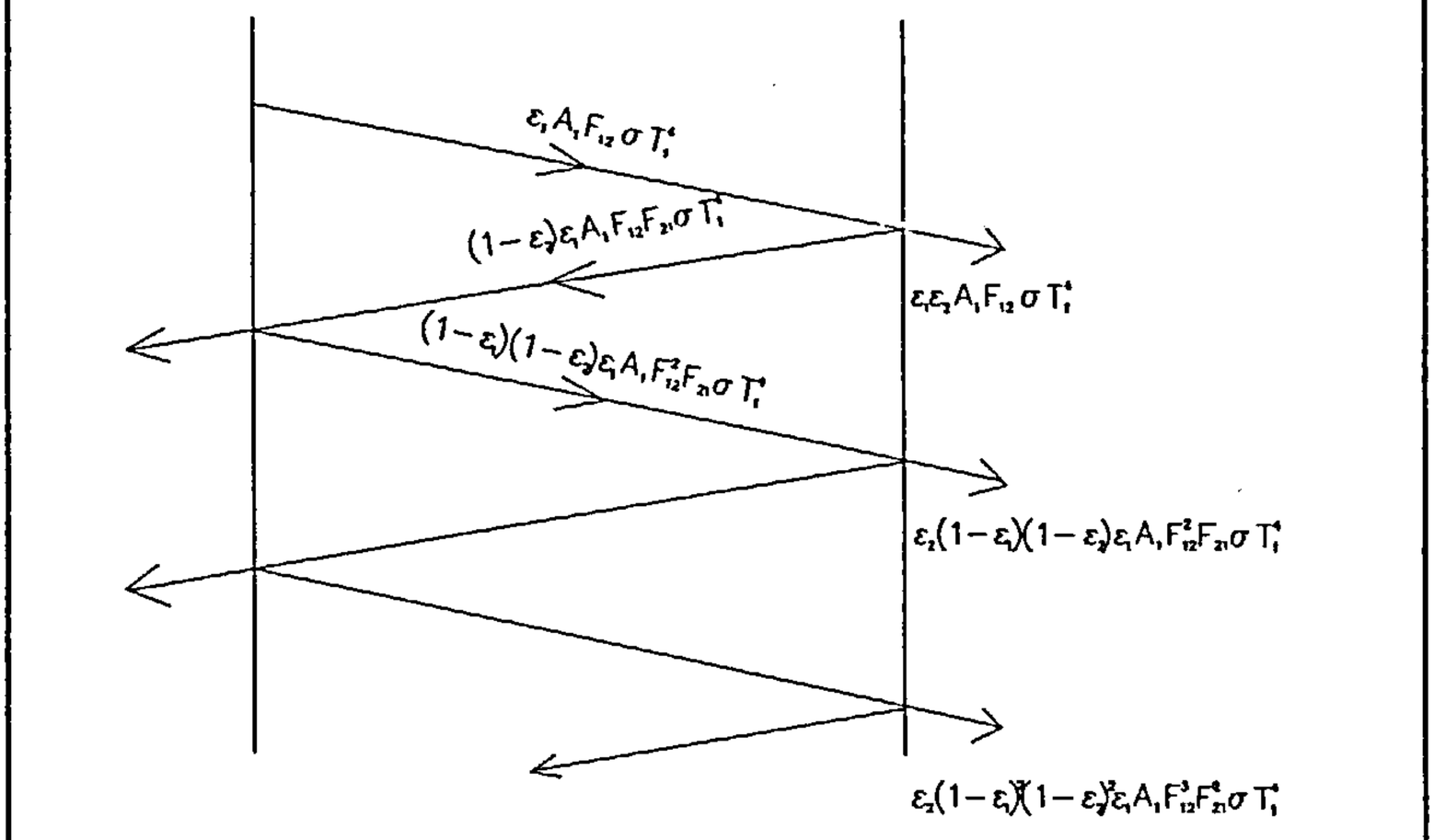


Figure 21 - radiant exchange between two surfaces

Thus, the radiative flux received by surface 2 from surface 1 through infinite reflection is:-

$$q_{(2)1->2} = \varepsilon_1 \varepsilon_2 A_1 F_{12} \sigma T_1^4 + (1 - \varepsilon_1) \varepsilon_2 (1 - \varepsilon_2) \varepsilon_1 A_1 F_{12}^2 F_{21} \sigma T_1^4 + (1 - \varepsilon_1)^2 \varepsilon_2 (1 - \varepsilon_2)^2 \varepsilon_1 A_1 F_{12}^3 F_{21}^2 \sigma T_1^4 + \ldots . \quad \text{Eq.4.59}$$

This expression is a geometric progression having the form:-

$$a + ar + ar^2 + \ldots . + ar^{n-1} \quad \text{Eq.4.60}$$

Now, in general, the sum to infinity of a geometric progression is:-

$$S_n = a\left(\frac{1 - r^n}{1 - r}\right) \quad \text{Eq.4.61}$$

where r, the common ratio, is $(1 - \varepsilon_1)(1 - \varepsilon_2) F_{12} F_{21}$ and the first term a, is $\varepsilon_1 \varepsilon_2 A_1 F_{12} \sigma T_1^4$. If r lies between -1 and +1, assuming that r^n approaches zero as n increases, the sum to infinity

becomes:-

$$S_n = \frac{a}{1 - r} \qquad \text{Eq.4.62}$$

Thus, Equation 4.59 becomes:-

$$q_{(2)1->2} = \frac{\varepsilon_1 \varepsilon_2 A_1 F_{12} \sigma T_1^4}{1 - [(1 - \varepsilon_1)(1 - \varepsilon_2) F_{12} F_{21}]} \qquad \text{Eq.4.63}$$

Similarly, considering the three interacting surfaces illustrated in Figure 22, the radiative flux received by surface 2 from surface 1 through infinite reflection is:-

$$\begin{aligned} q_{(3)1->2} = {} & \varepsilon_1 \varepsilon_2 (1 - \varepsilon_n) A_1 F_{1n} F_{n2} \sigma T_1^4 + \varepsilon_2 (1 - \varepsilon_1)(1 - \varepsilon_2)(1 - \varepsilon_n)^2 \\ & \varepsilon_1 A_1 F_{1n}^2 F_{n2}^2 F_{21} \sigma T_1^4 + \varepsilon_2 (1 - \varepsilon_1)^2 (1 - \varepsilon_2)^2 (1 - \varepsilon_n)^3 \\ & \varepsilon_1 A_1 F_{1n}^3 F_{n2}^3 F_{21}^2 \sigma T_1^4 + \ldots . \end{aligned} \qquad \text{Eq.4.64}$$

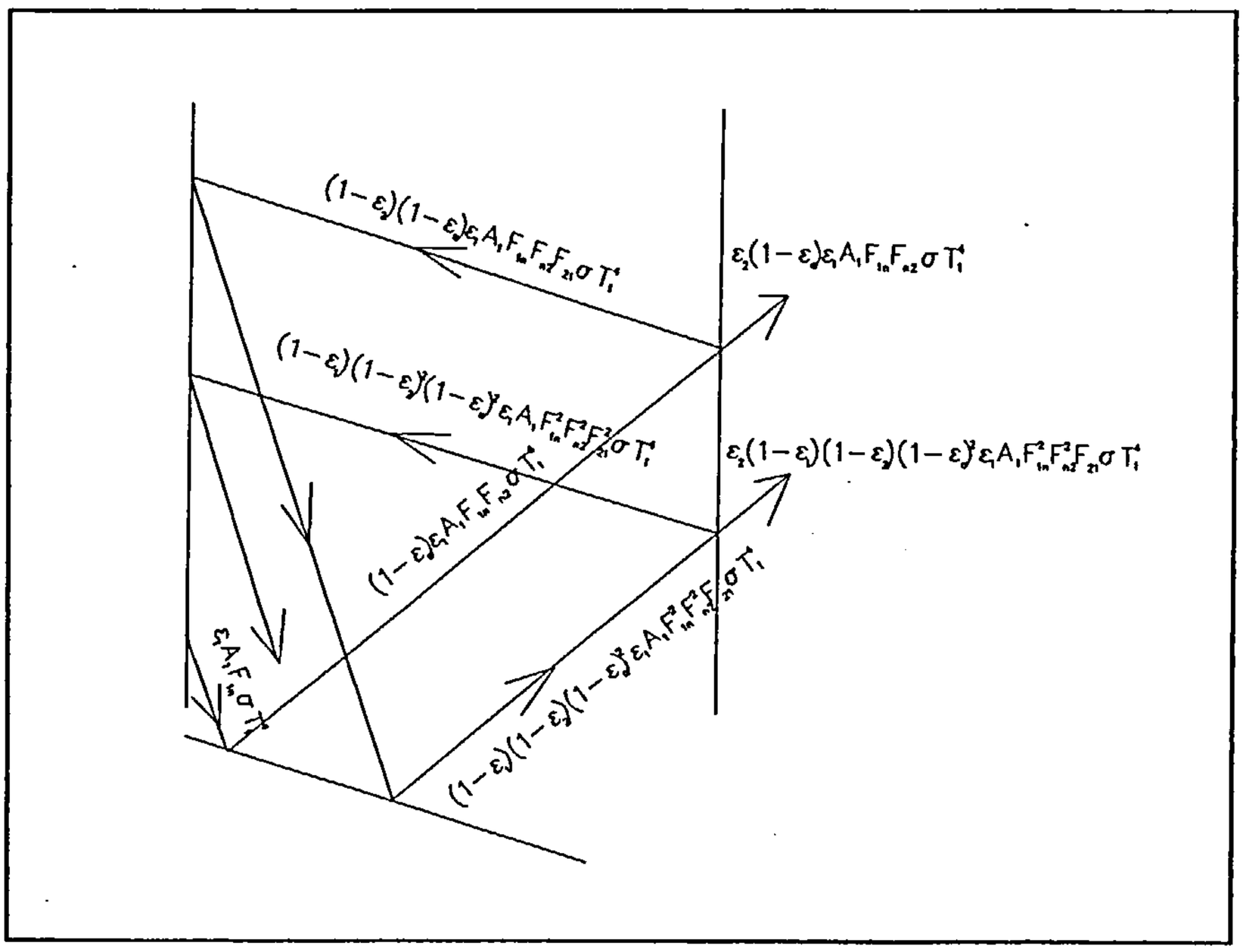


Figure 22 - radiant exchange between three surfaces

Identifying a similar geometric progression as for the two surface case, Equation 4.64 becomes:-

$$q_{(3)1->2} = \frac{\varepsilon_1 \varepsilon_2 (1 - \varepsilon_n) A_1 F_{1n} F_{n2} \sigma T_1^4}{1 - (1 - \varepsilon_1)(1 - \varepsilon_2)(1 - \varepsilon_n) F_{1n} F_{n2} F_{21}} \qquad \text{Eq.4.65}$$

and therefore the flux arriving at surface 2 from surface 1 via all other participating enclosure surfaces, accounting for three surface reflections only is:-

$$q_{(3)1->2} = \sum_{i=1}^{n} \frac{\varepsilon_1 \varepsilon_2 (1 - \varepsilon_i) A_1 F_{1i} F_{i2} \sigma T_1^4}{1 - [(1 - \varepsilon_1)(1 - \varepsilon_2)(1 - \varepsilon_i) F_{1i} F_{i2} F_{21}]} \qquad \text{Eq.4.66}$$

In the same fashion, four surface reflections may be accounted for and so on. The approximate total flux arriving at surface 2 from surface 1 due to infinite two and three surface reflections may be derived by adding equations 4.63 and 4.66:-

$$q_{1->2} = \frac{\varepsilon_1 \varepsilon_2 A_1 F_{12} \sigma T_1^4}{1 - [(1 - \varepsilon_1)(1 - \varepsilon_2) F_{12} F_{21}]} + \sum_{i=1}^{n} \frac{\varepsilon_1 (1 - \varepsilon_i) \varepsilon_2 A_1 F_{1i} F_{i2} \sigma T_1^4}{1 - [(1 - \varepsilon_1)(1 - \varepsilon_2)(1 - \varepsilon_i) F_{1i} F_{i2} F_{21}]} \qquad \text{Eq.4.67}$$

where $i \neq 1, 2$

and similarly the flux arriving at surface 1 from 2 is:-

$$q_{2->1} = \frac{\varepsilon_1 \varepsilon_2 A_2 F_{21} \sigma T_2^4}{1 - [(1 - \varepsilon_2)(1 - \varepsilon_1) F_{21} F_{12}]} + \sum_{i=1}^{n} \frac{\varepsilon_1 (1 - \varepsilon_i) \varepsilon_2 A_2 F_{2i} F_{i1} \sigma T_2^4}{1 - [(1 - \varepsilon_2)(1 - \varepsilon_1)(1 - \varepsilon_i) F_{2i} F_{i1} F_{12}]} \qquad \text{Eq.4.68}$$

where $i \neq 1, 2$

and thus the net exchange of radiative flux between surfaces 1 and 2 is found by subtracting Equation 4.67 from Equation 4.68:-

$$q_{2,1} = \frac{\varepsilon_1 \varepsilon_2 A_1 F_{12} \sigma (T_2^4 - T_1^4)}{1 - [(1 - \varepsilon_1)(1 - \varepsilon_2) F_{12} F_{21}]} + \sum_{i=1}^{n} \frac{\varepsilon_1 (1 - \varepsilon_i) \varepsilon_2 - A_2 F_{2i} F_{i1} \sigma T_2^4}{1 - [(1 - \varepsilon_2)(1 - \varepsilon_1)(1 - \varepsilon_i) F_{2i} F_{i1} F_{12}]}$$

$$-\sum_{i=1}^{n} \frac{\varepsilon_1 (1-\varepsilon_i)\, \varepsilon_2 A_1 F_{1i} F_{i2} \sigma T_1^4}{1-[(1-\varepsilon_1)(1-\varepsilon_2)(1-\varepsilon_i) F_{1i} F_{i2} F_{21}]} \qquad \text{Eq.4.69}$$

The net radiative flux exchange between surfaces must now be expressed as a volumetric flux to be included within the finite volume cell energy equations as described in Section 3.4.10.

Considering Figure 23, the view factor for radiation from surface 1 to surface 2 may be expressed as:-

$$F_{1->2} = F_{1,\,1->2} + F_{1,2->2} + F_{1,3,2->2} + F_{1,4,2->2}$$

Thus, the surface radiation view factors derived using the method detailed in section 4.3.1 may be incorporated within the finite volume cell expressions by simply dividing each participating surface factor by the number of cells that lie in the surface plane.

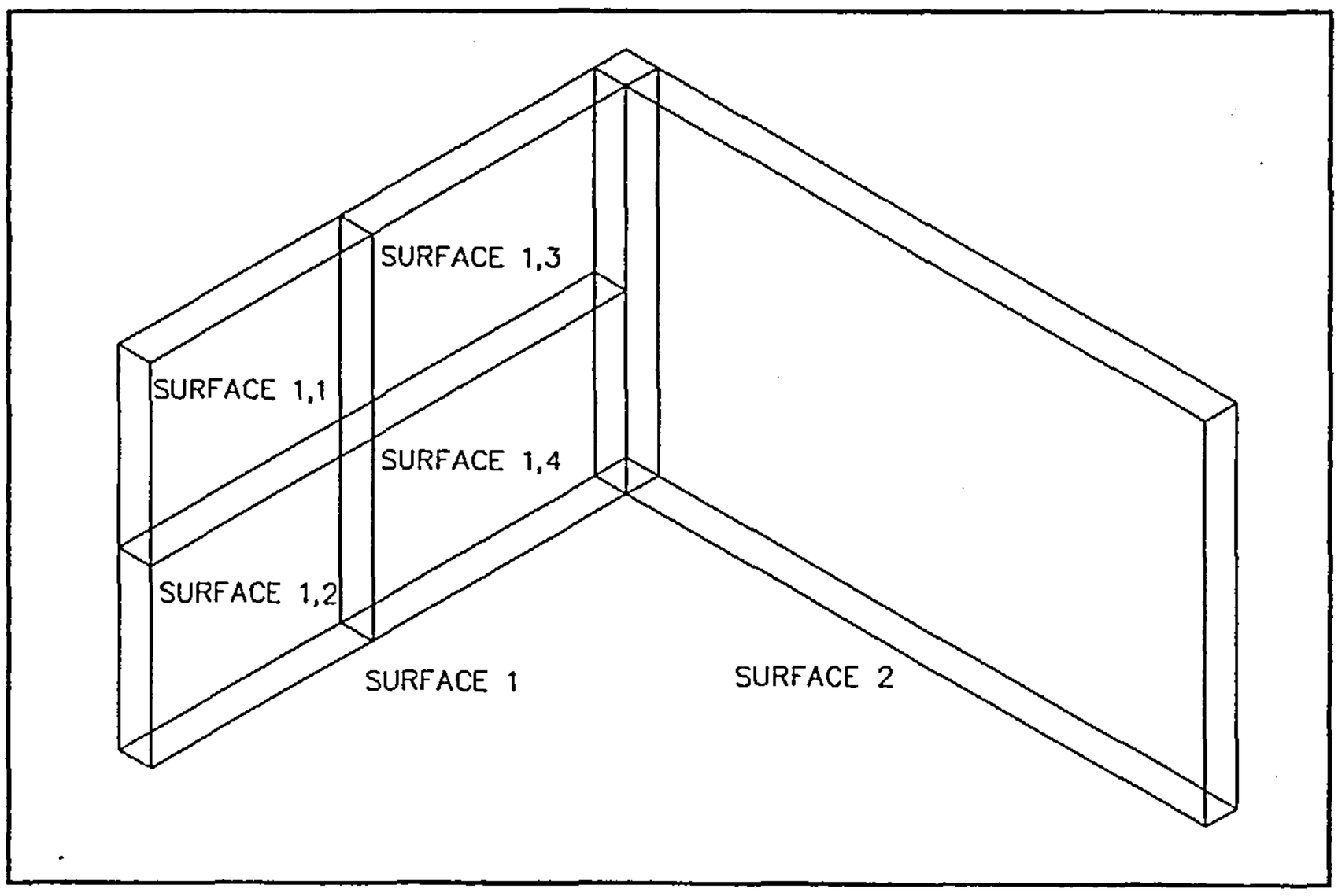


Figure 23 - radiating surface composed of finite volume cell surfaces

In order to derive the most appropriate expression for radiant flux to be incorporated within the energy equation source term, of the form indicated in Section 3.3.9, Equation 4.69 may be expressed in terms of T_1:-

$$S = S_C + S_P \phi_P = A - BT_1^4 \qquad \text{Eq.4.70}$$

Thus from Equations 3.105 and 3.106,

$$S_C = A + 3\,B\,T_1^{*4} \qquad \text{Eq.4.71}$$

and

$$S_P = -4\,B\,T_1^{3} \qquad \text{Eq.4.72}$$

where

$$A = \frac{\varepsilon_1 \varepsilon_2 A_2 F_{21} \sigma T_2^4}{1 - [(1-\varepsilon_2)(1-\varepsilon_1) F_{21} F_{12}]} + \sum_{i=1}^{n} \frac{\varepsilon_1 (1-\varepsilon_i) \varepsilon_2 A_2 F_{2i} F_{i1} \sigma T_2^4}{1 - (1-\varepsilon_2)(1-\varepsilon_1)(1-\varepsilon_i) F_{2i} F_{i1} F_{12}}$$

$i \neq 1, 2$

$$B = \frac{\varepsilon_1 \varepsilon_2 A_1 F_{12} \sigma}{1 - [(1-\varepsilon_1)(1-\varepsilon_2) F_{12} F_{21}]} + \sum_{i=1}^{n} \frac{\varepsilon_1 (1-\varepsilon_i) \varepsilon_2 A_1 F_{1i} F_{i2} \sigma}{1 - (1-\varepsilon_1)(1-\varepsilon_2)(1-\varepsilon_i) F_{1i} F_{i2} F_{21}}$$

$i \neq 1, 2$

The S_c term requires modification due to the fact that T is expressed in absolute temperature:-

$$S_c = A + 3\,B\,T_1^{*4} - 1092\,B\,T_1^{*3} \qquad \text{Eq.4.73}$$

4.4 Application of the Coupled Convection-Conduction-Radiation Model

The coupled model has been applied to the Annex 20 test room (see Chapter 5) under typical winter heating conditions with and without the radiation model to demonstrate the influence of radiation on the air flow and temperature distribution.

The Annex 20 test room geometry has been modified to incorporate a simple side-wall diffuser located at 0.3 m below the ceiling as shown in Figure 24. The supply volume flow rate is set at 0.0525 m^3s^{-1}, which is equivalent to an air change rate of 5 a.c hr^{-1} and results in a discharge velocity of 3.5 ms^{-1}. The window surface temperature is set at 5 °C and the supply temperature 25 °C.

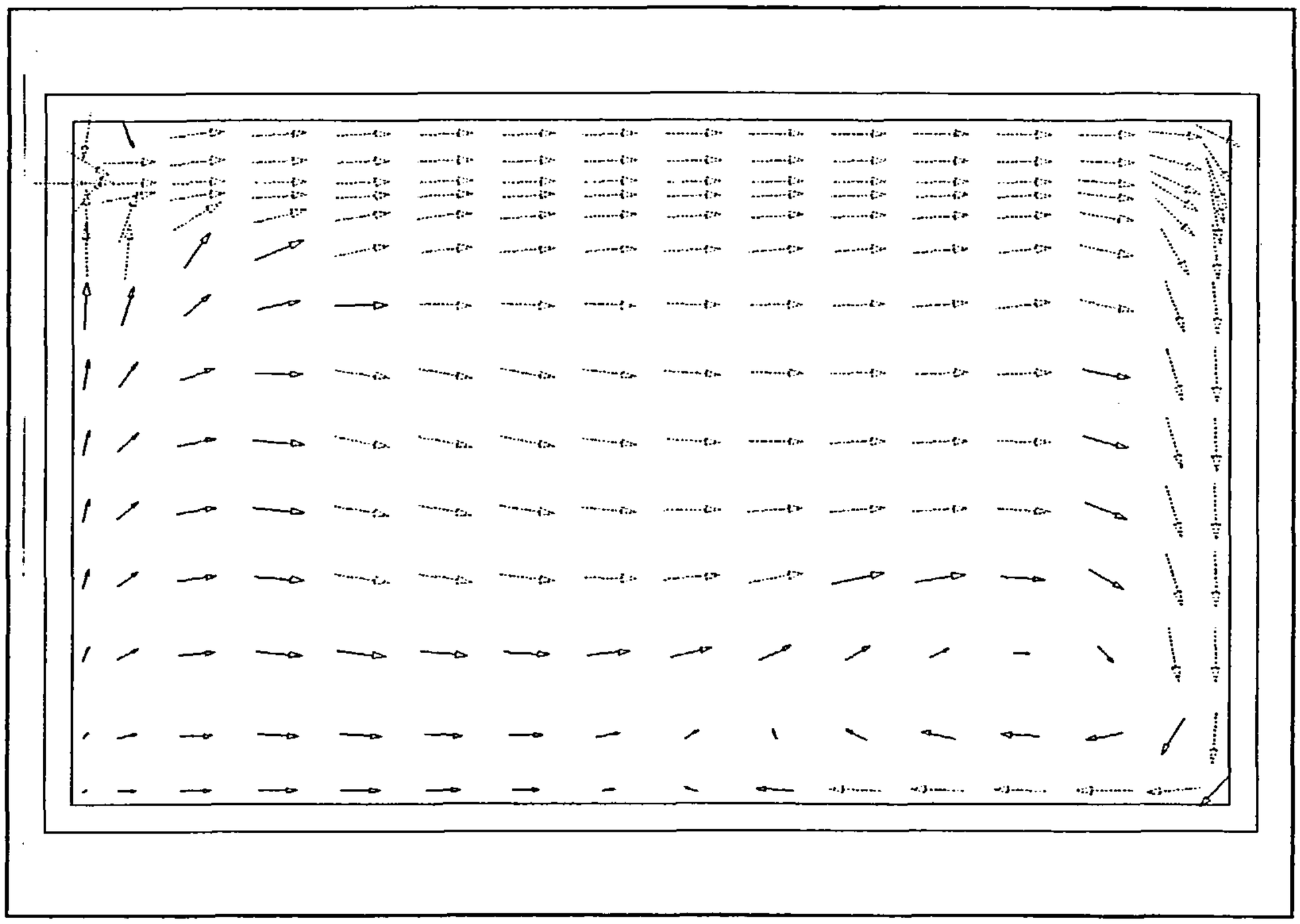


Figure 25 - simulated velocity vectors on centre-line of Z-dimension in X-Y plane (without radiation model)

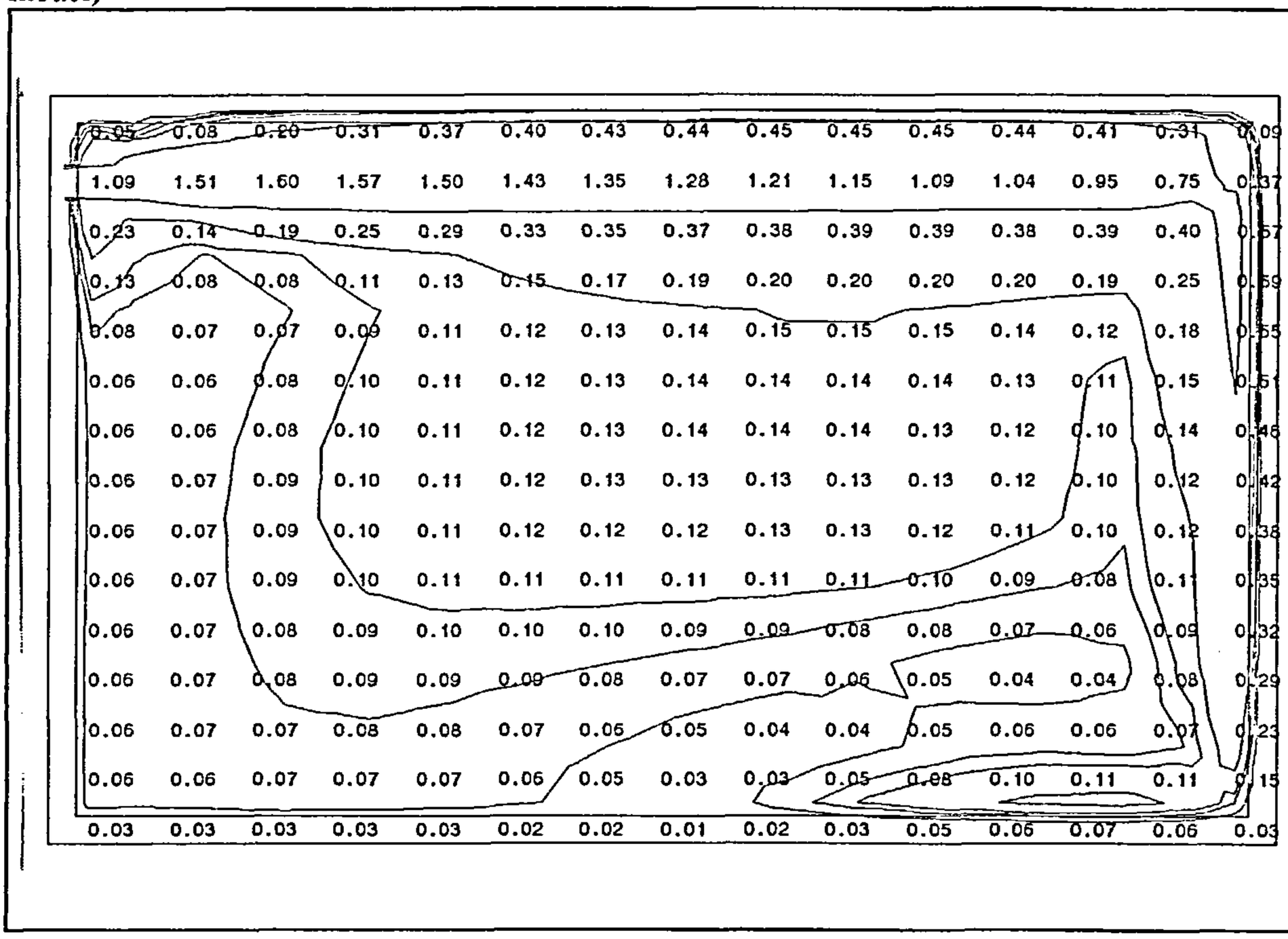


Figure 26 - simulated velocity contours on centre-line of Z-dimension in X-Y plane (without radiation model)

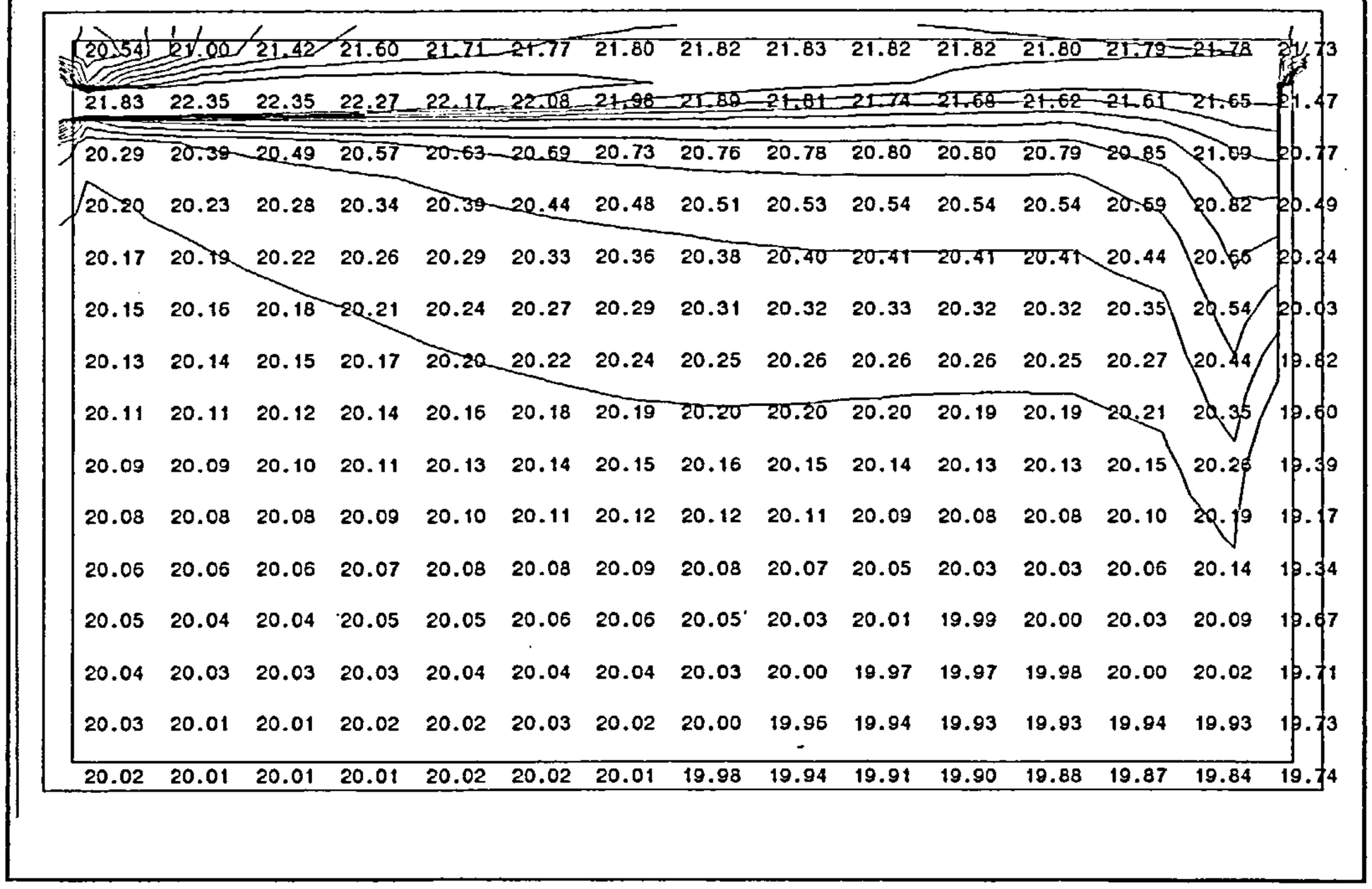


Figure 27 - simulated temperature contours on centre-line of Z-dimension in X-Y plane (without radiation model)

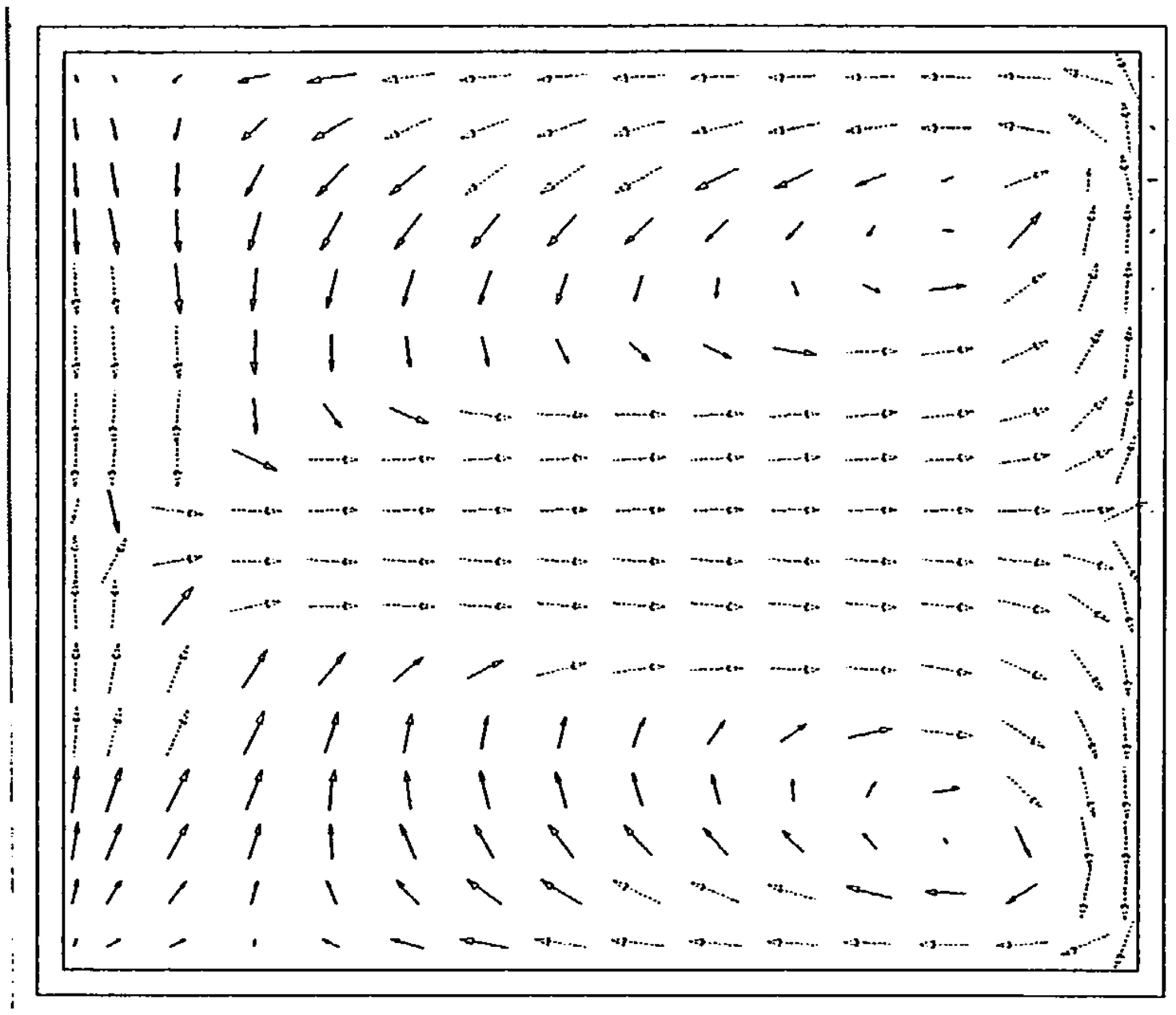


Figure 28 - simulated velocity vectors at 0.05 m below ceiling in X-Z plane (without radiation model)

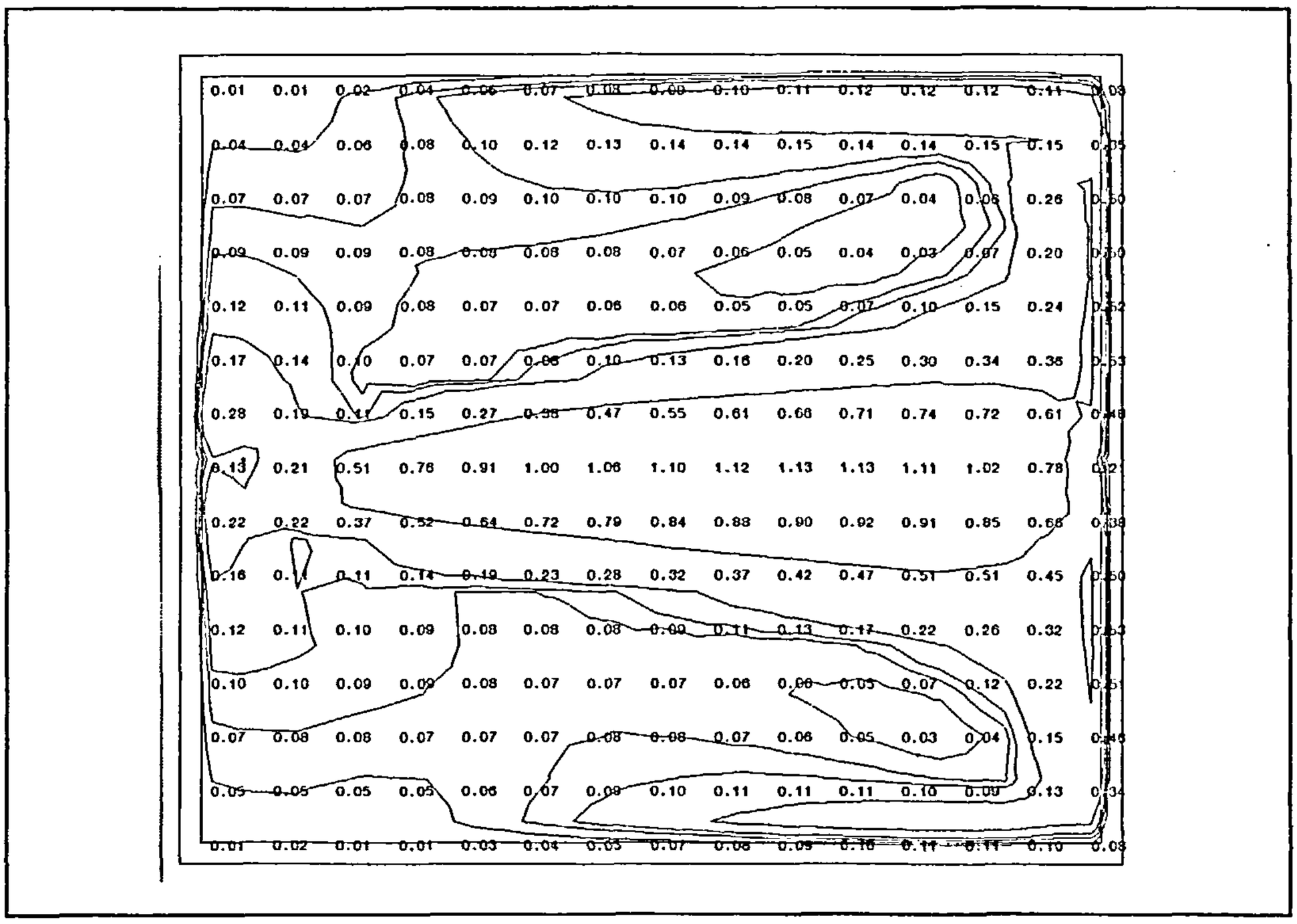


Figure 29 - simulated velocity contours at 0.05 m below ceiling in X-Z plane (without radiation model)

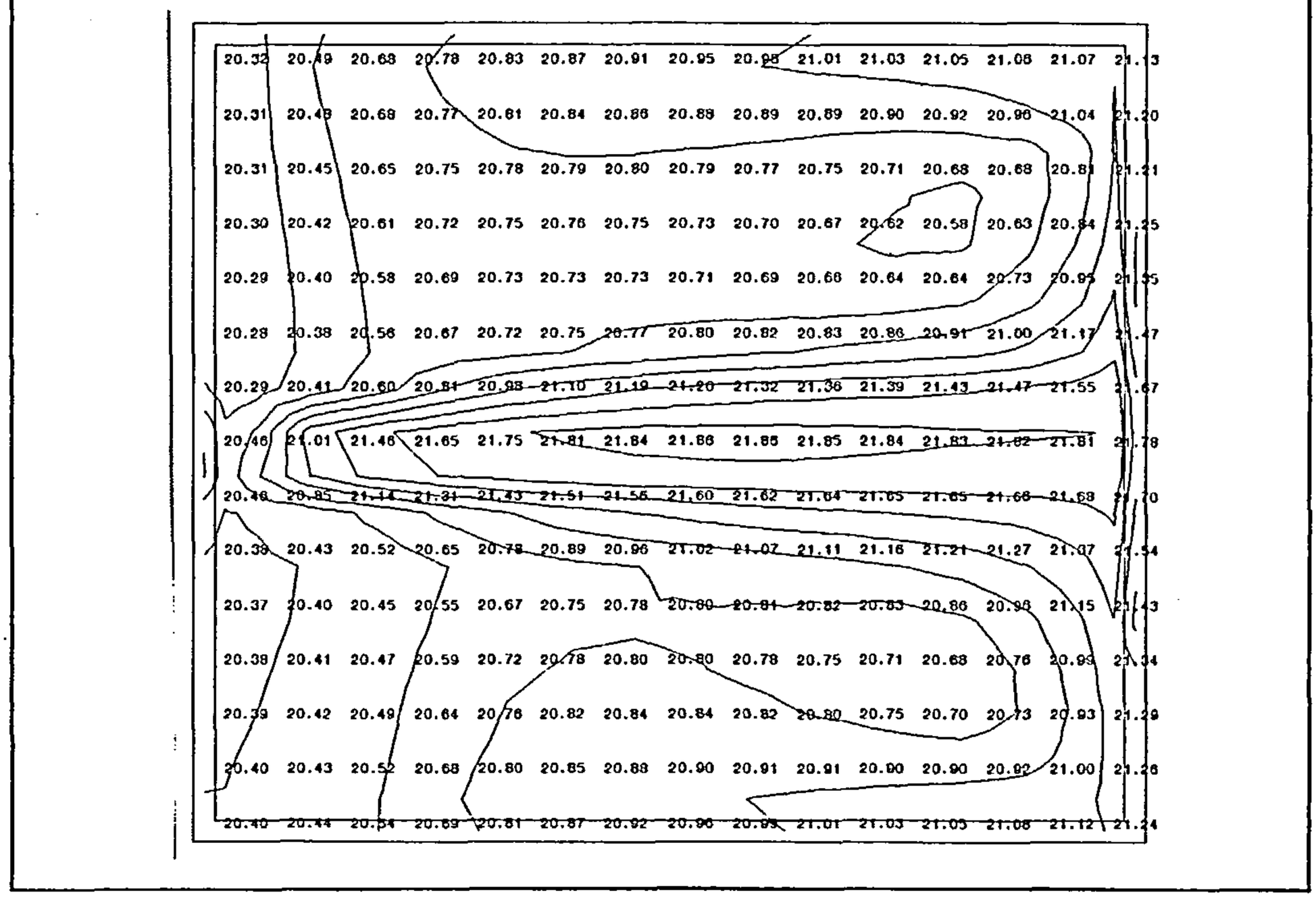


Figure 30 - simulated temperature vectors at 0.05 m below ceiling in X-Z plane (without radiation model)

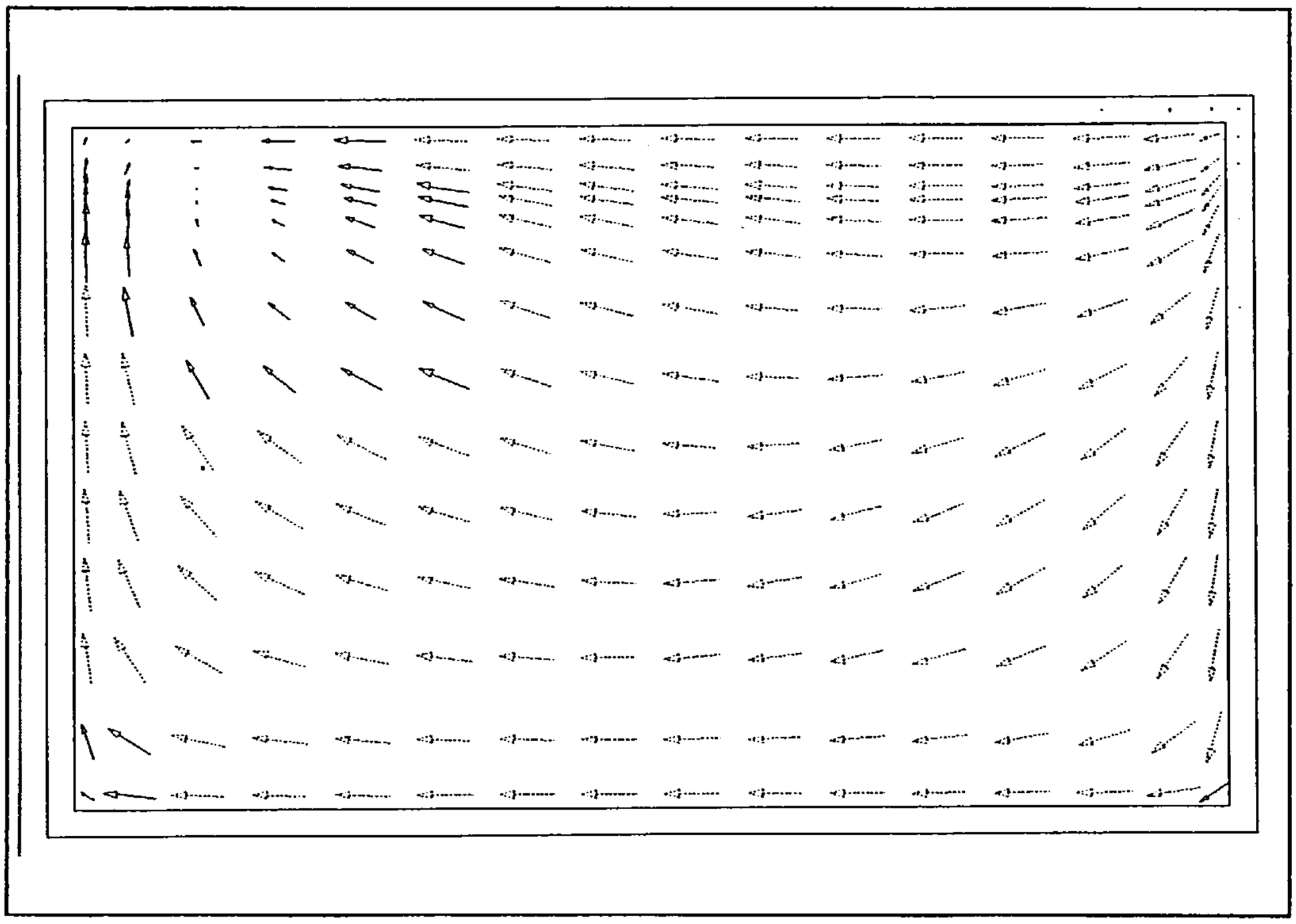


Figure 31 - simulated velocity vectors at 3.6 m along Z-axis in X-Y plane (without radiation model)

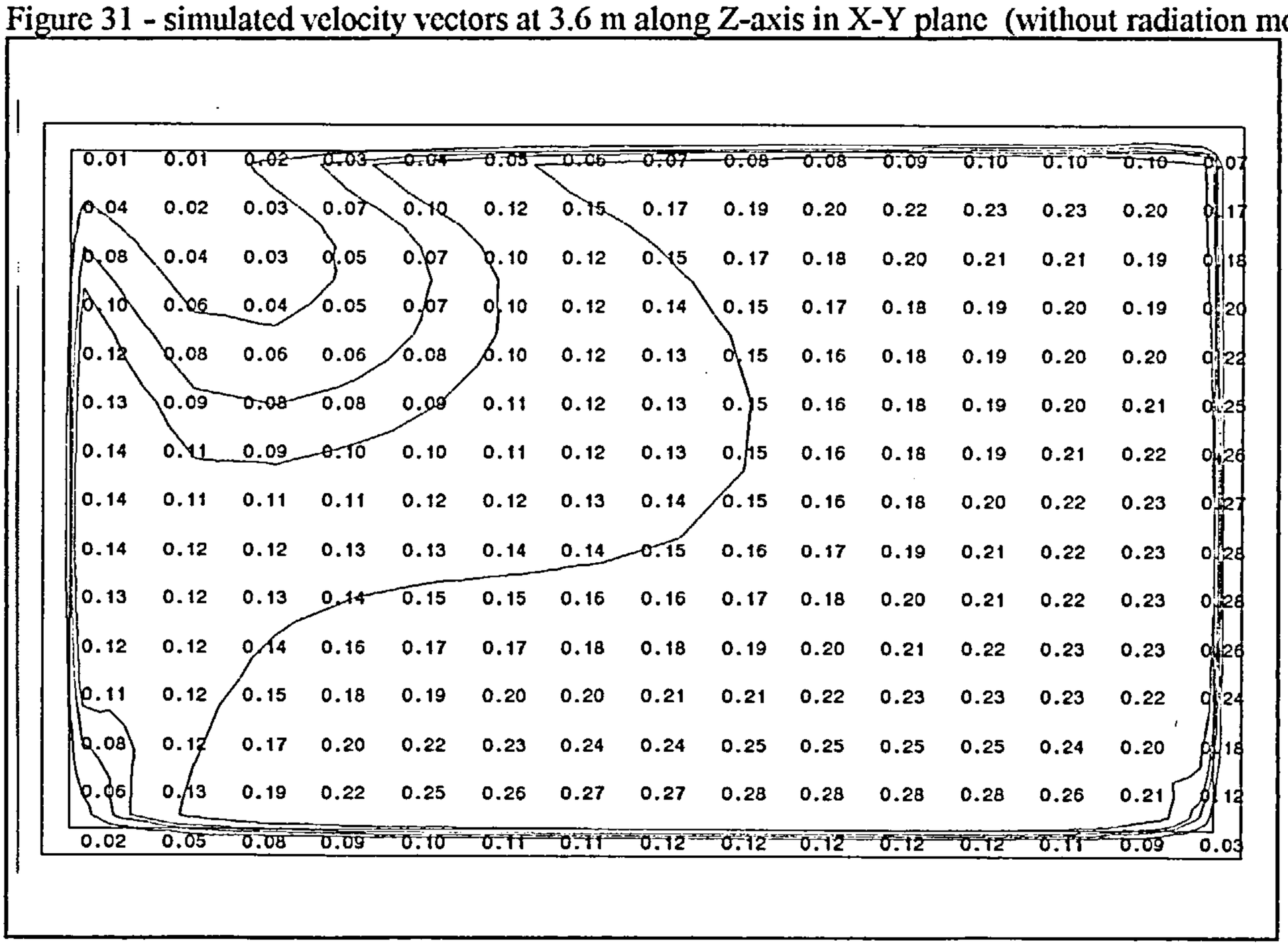


Figure 32 - simulated velocity contours at 3.6 m along Z-axis in X-Y plane (without radiation model)

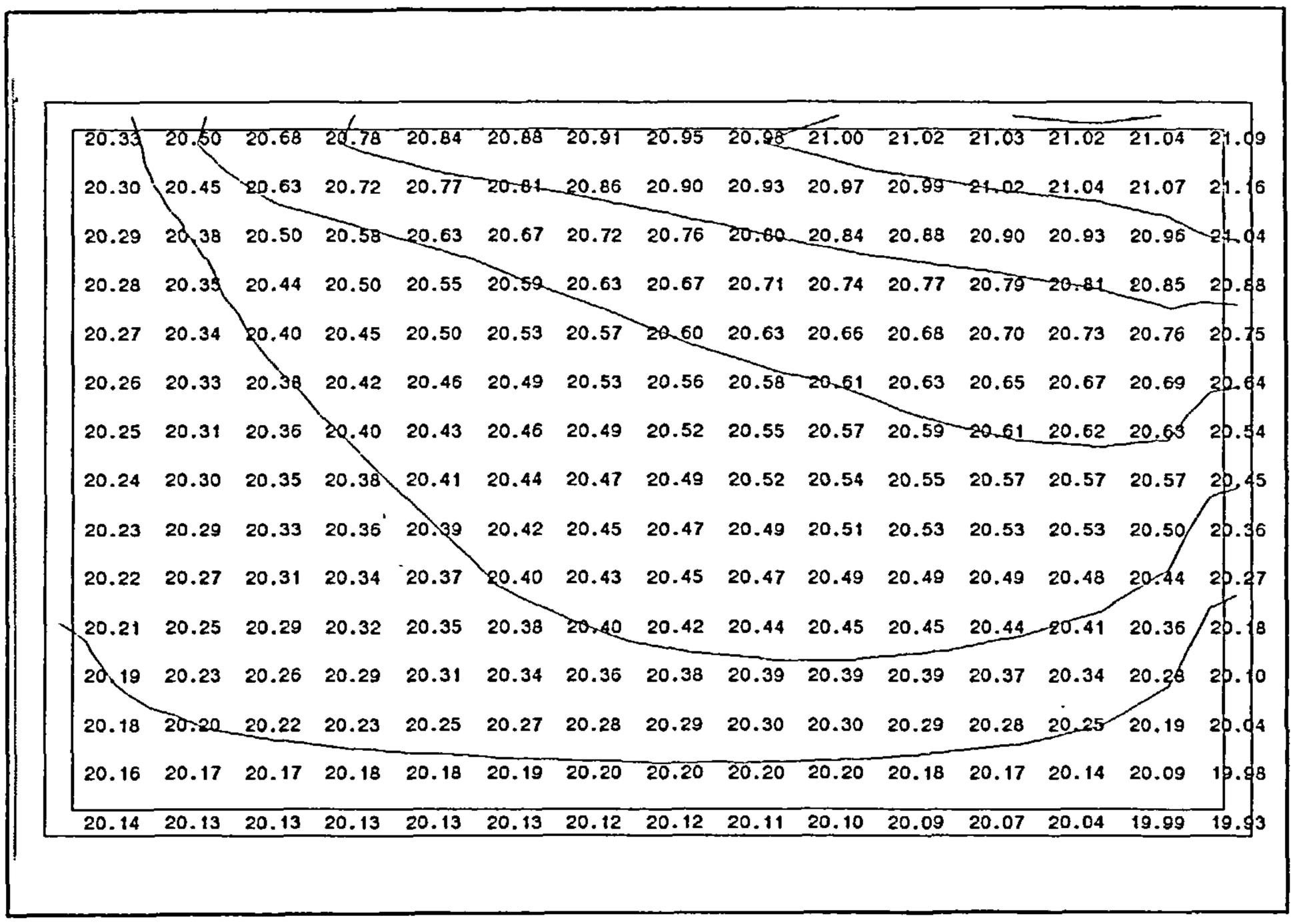


Figure 33 - simulated temperature contours at 3.6 m along Z-axis in X-Y plane (without radiation model)

A further simulation has been conducted using the coupled radiation model to demonstrate the influence of shortwave solar radiation on air flow and temperature distribution. The simulation has been conducted for a latitude of 50 °N at a time of 10.00 hrs on 21st January assuming that the window faces due east and the window glass has a transmissivity of 0.9.

Figures 34-42 show the results of simulation incorporating the radiation model. The air flow distribution may now be seen to be almost dominated by the buoyant flow up the surface of the north wall driven by the incident solar flux. Air temperatures throughout the occupied volume of the office are elevated significantly by the penetrating solar radiation, varying between 29.8 °C towards the floor and 31.0 °C towards the ceiling. Mean air speeds througout the occupied volume vary between 0.08 m s^{-1} - 0.7 m s^{-1}.

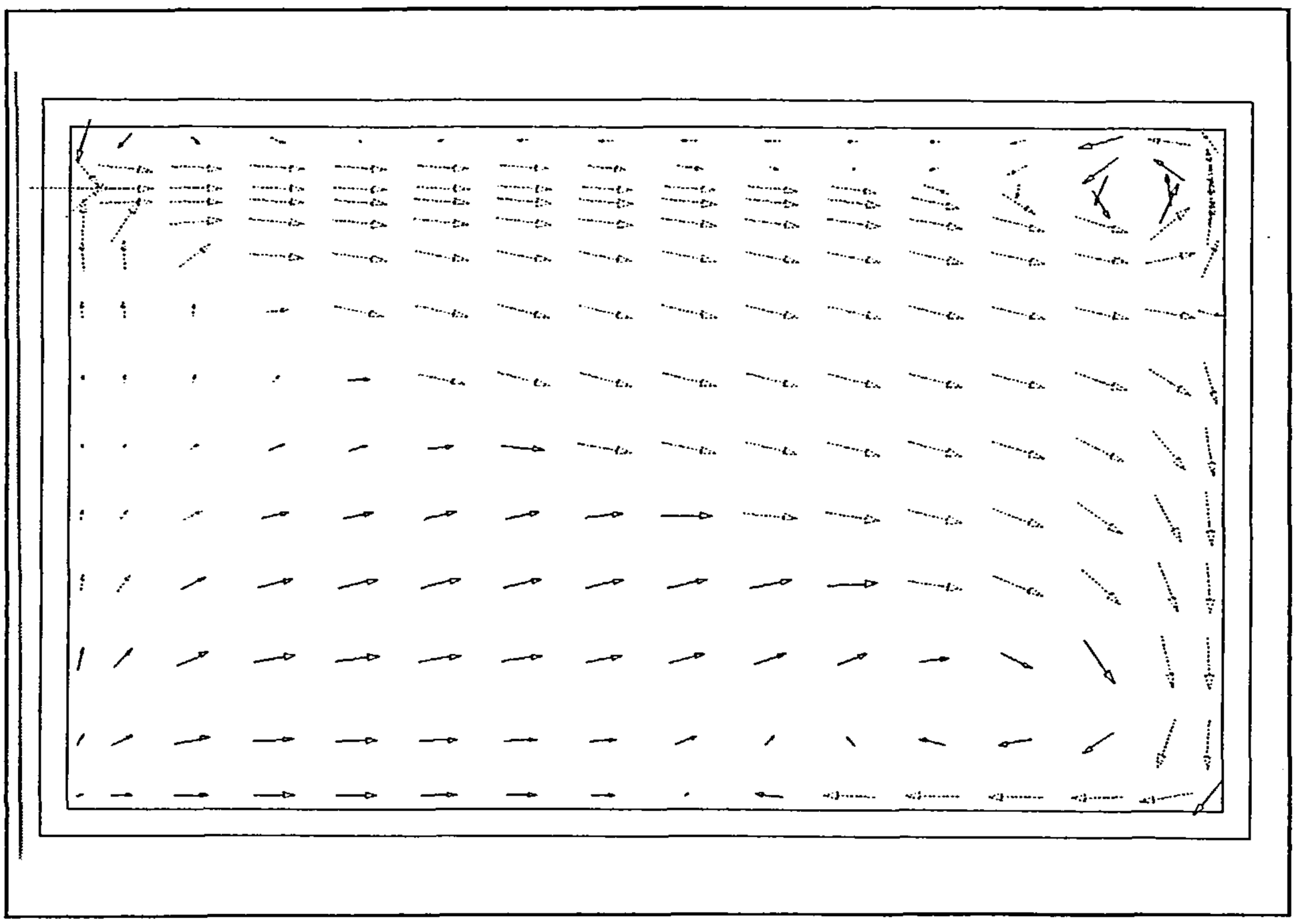


Figure 34 - simulated velocity vectors on centre-line of Z-dimension in X-Y plane (with radiation model)

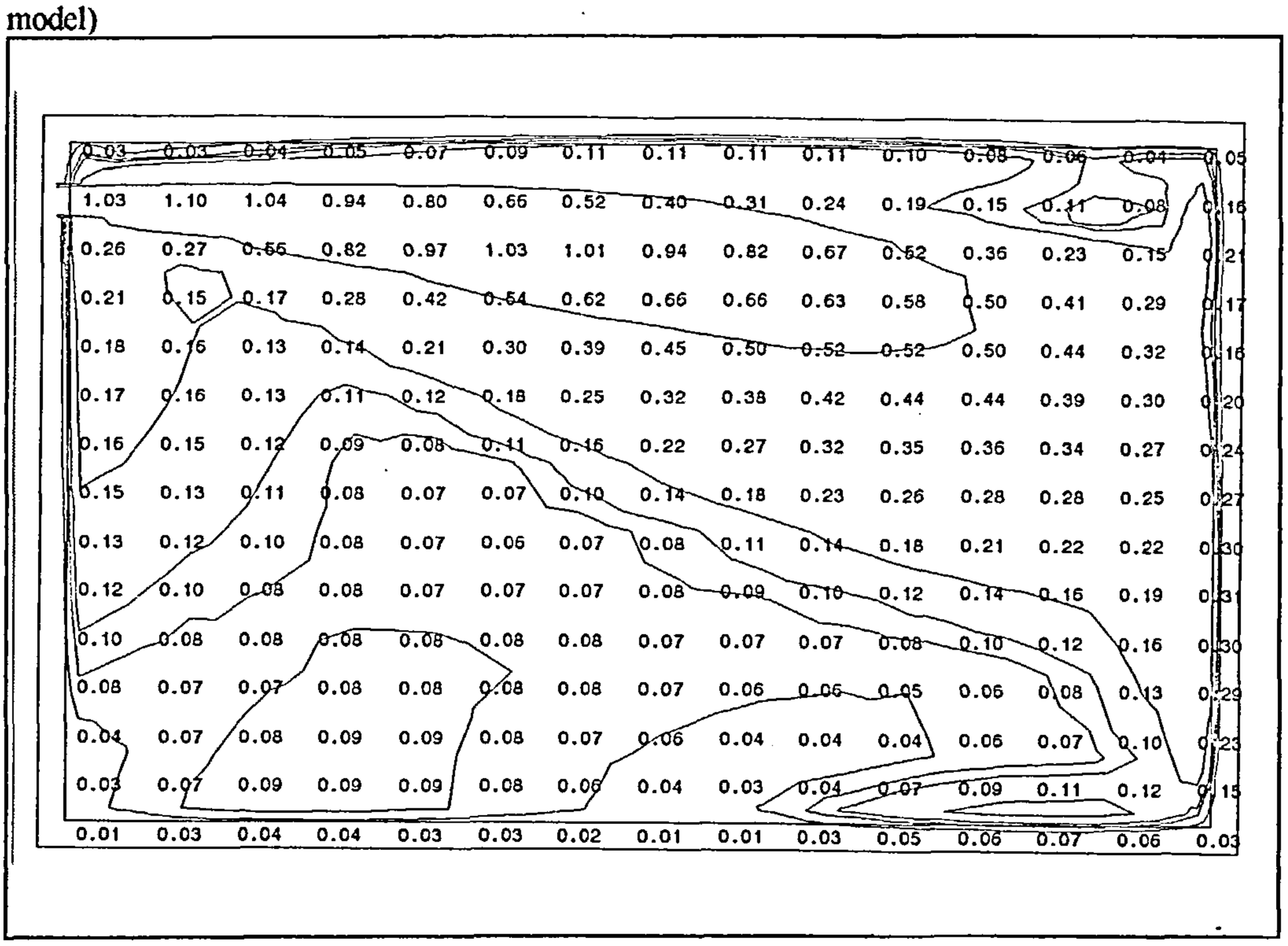


Figure 35 - simulated velocity contours on centre-line of Z-dimension in X-Y plane (with radiation model)

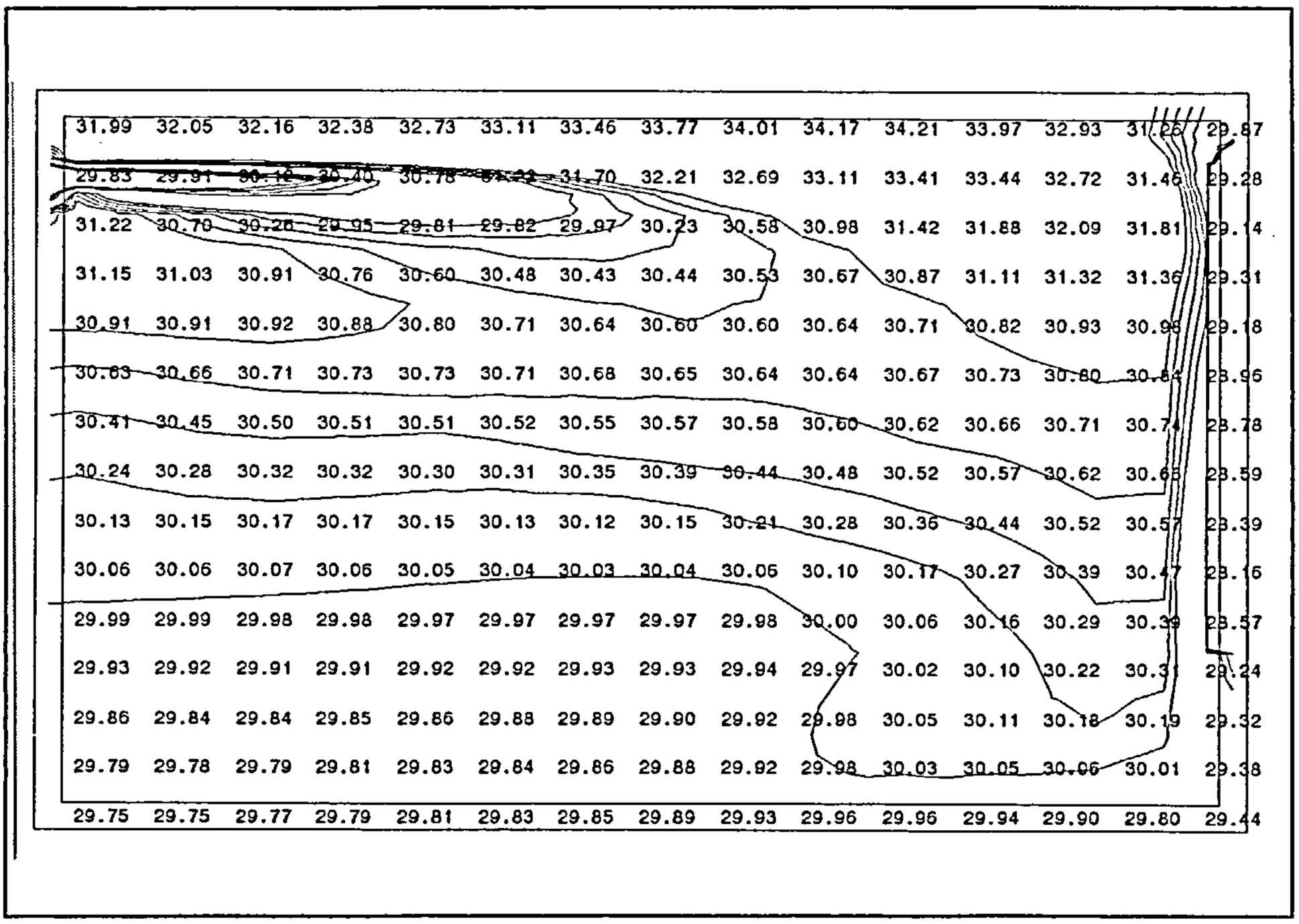


Figure 36 - simulated temperature contours on centre-line of Z-dimension in X-Y plane (with radiation model)

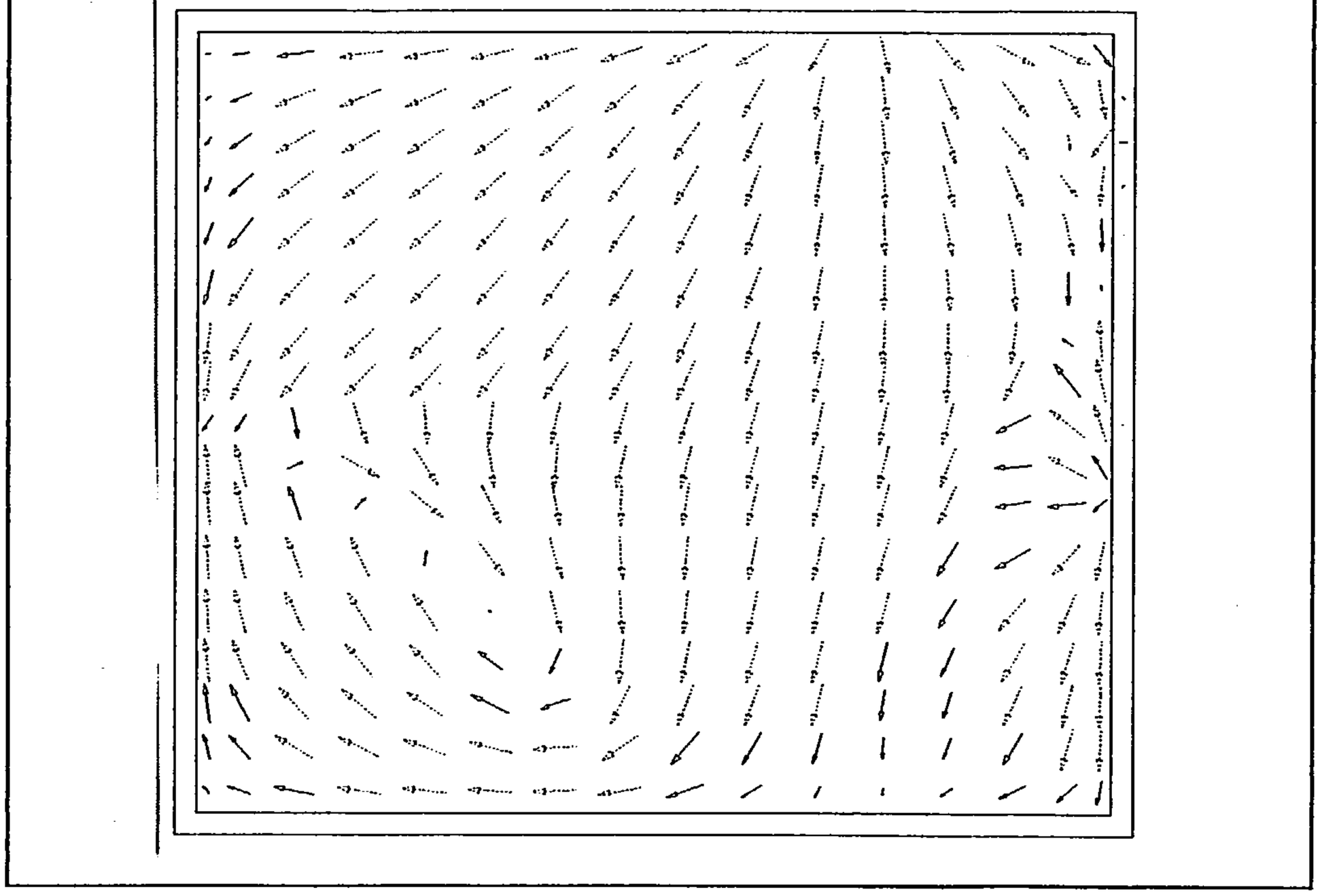


Figure 37 - simulated velocity vectors at 0.05 m below ceiling in X-Z plane (with radiation model)

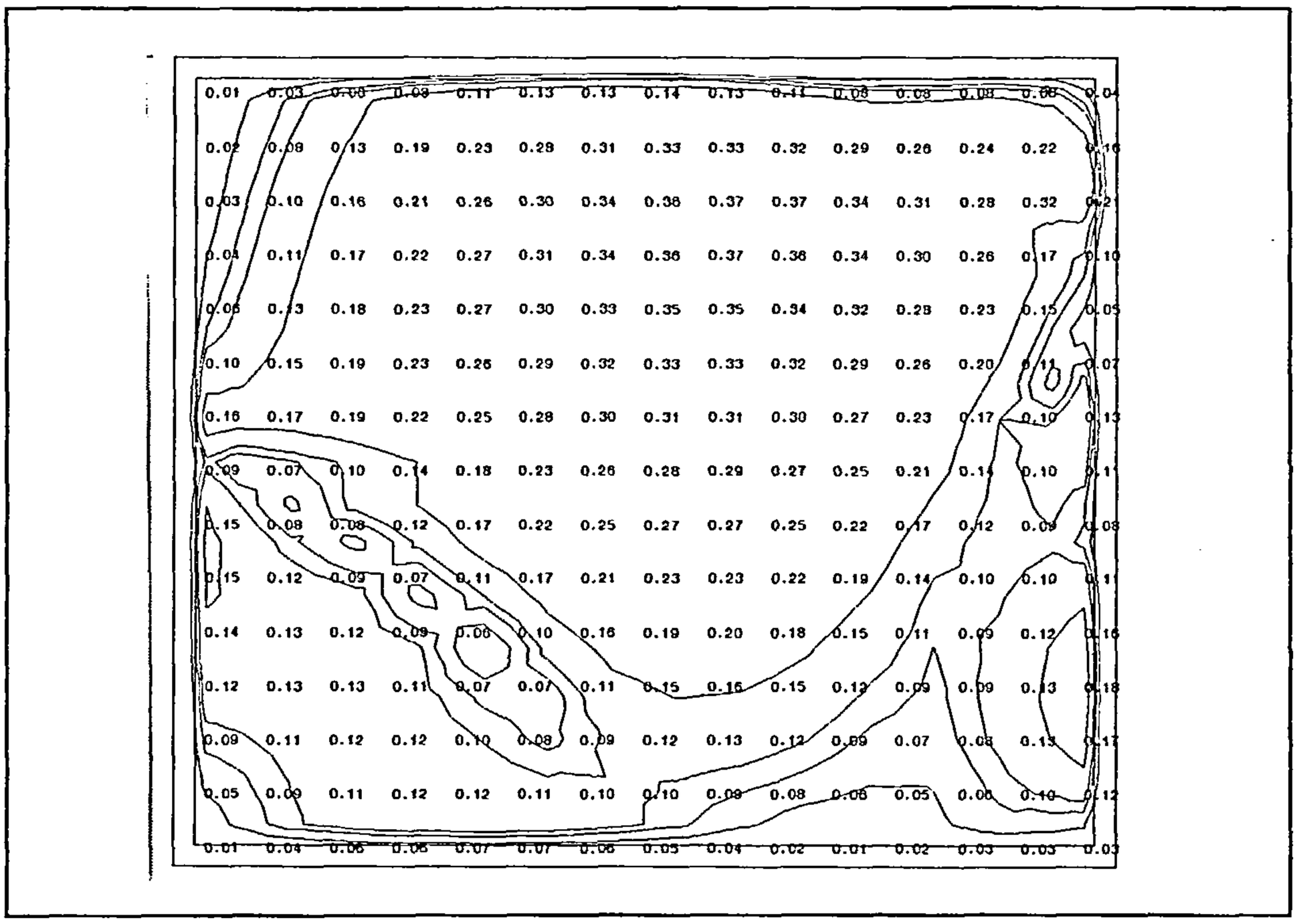


Figure 38 - simulated velocity contours at 0.05 m below ceiling in X-Z plane (with radiation model)

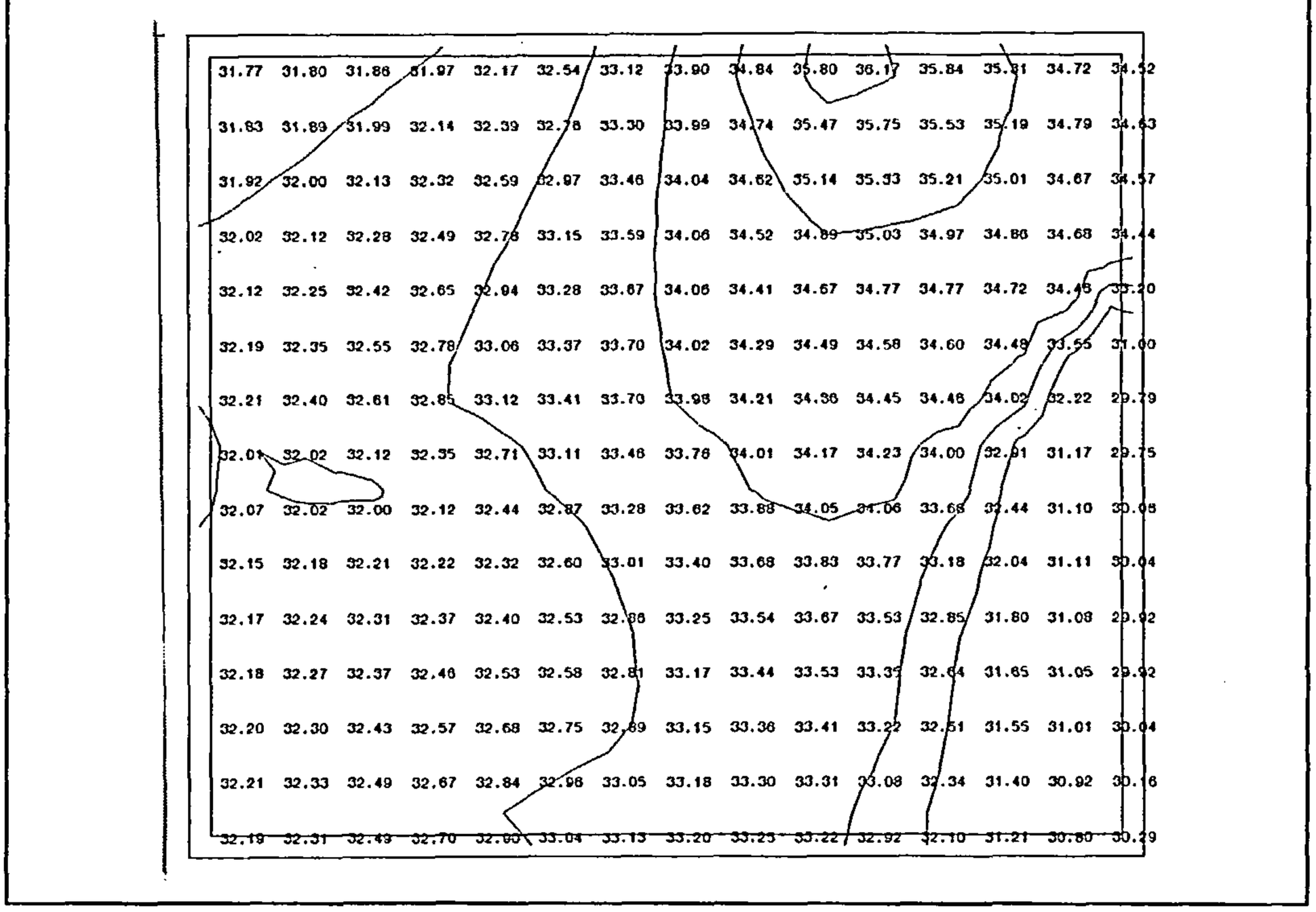


Figure 39 - simulated temperature vectors at 0.05 m below ceiling in X-Z plane (with radiation model)

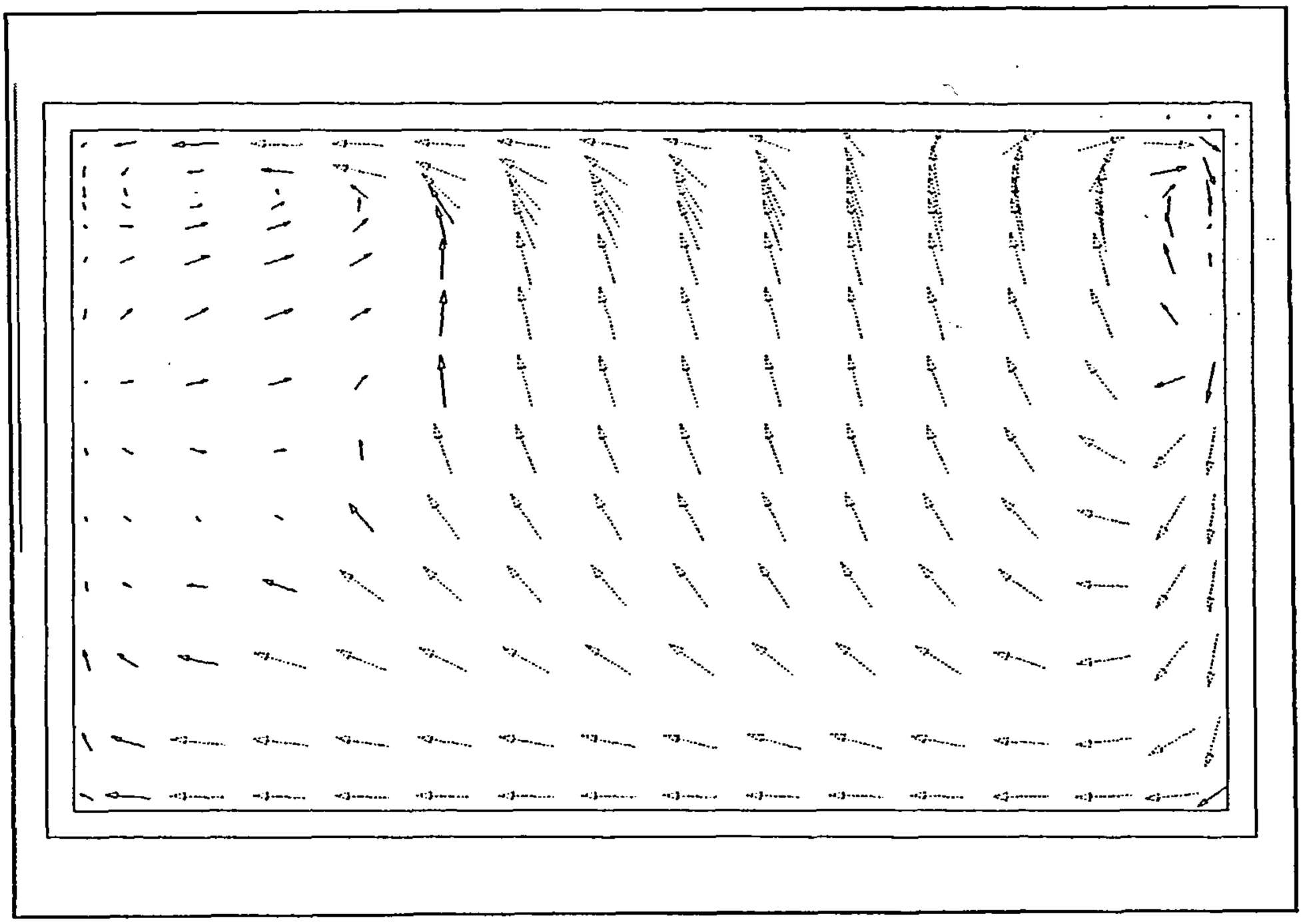


Figure 40 - simulated velocity vectors at 3.6 m along Z-axis in X-Y plane (with radiation model)

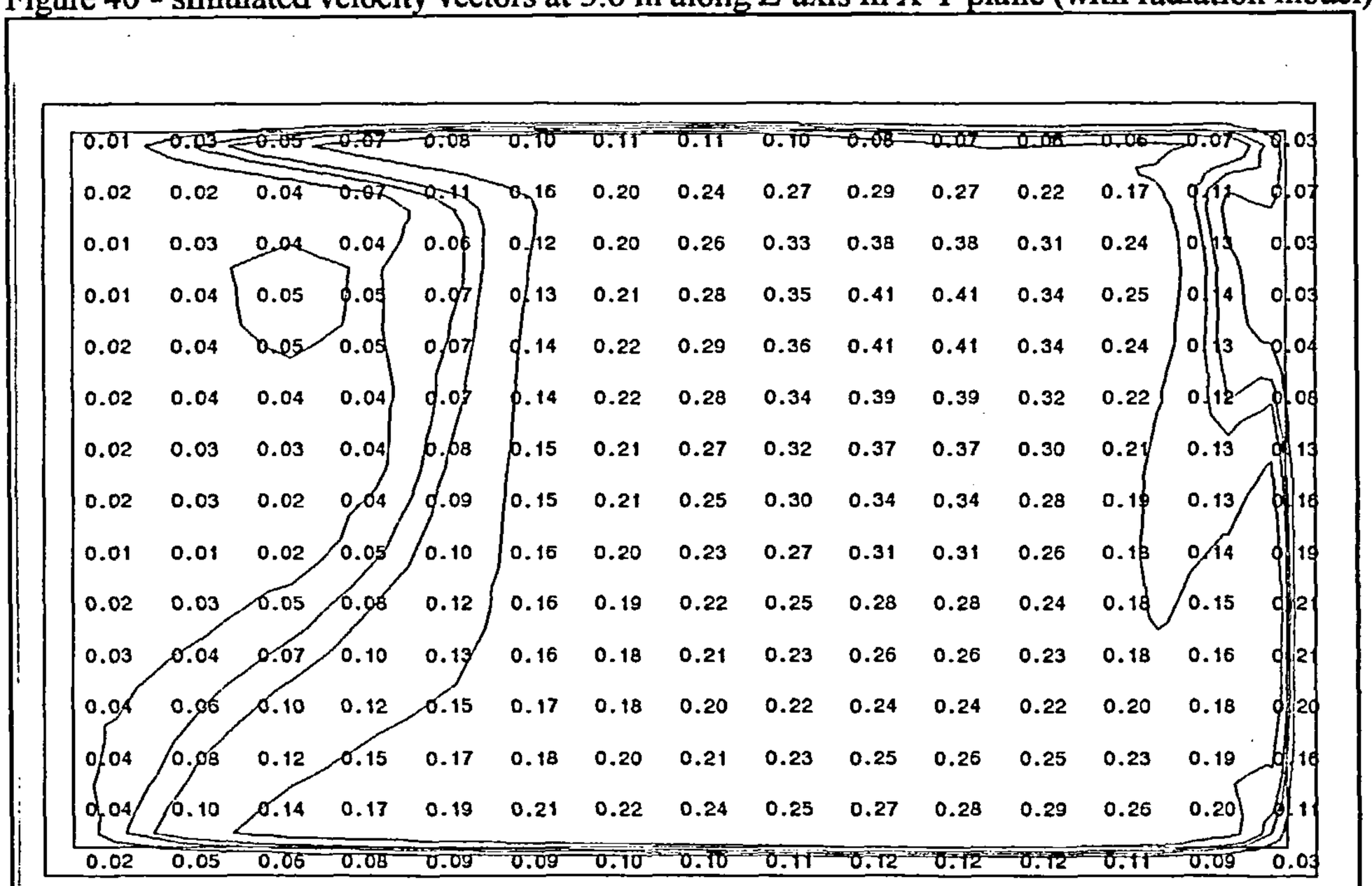


Figure 41 - simulated velocity contours at 3.6 m along Z-axis in X-Y plane (with radiation model)

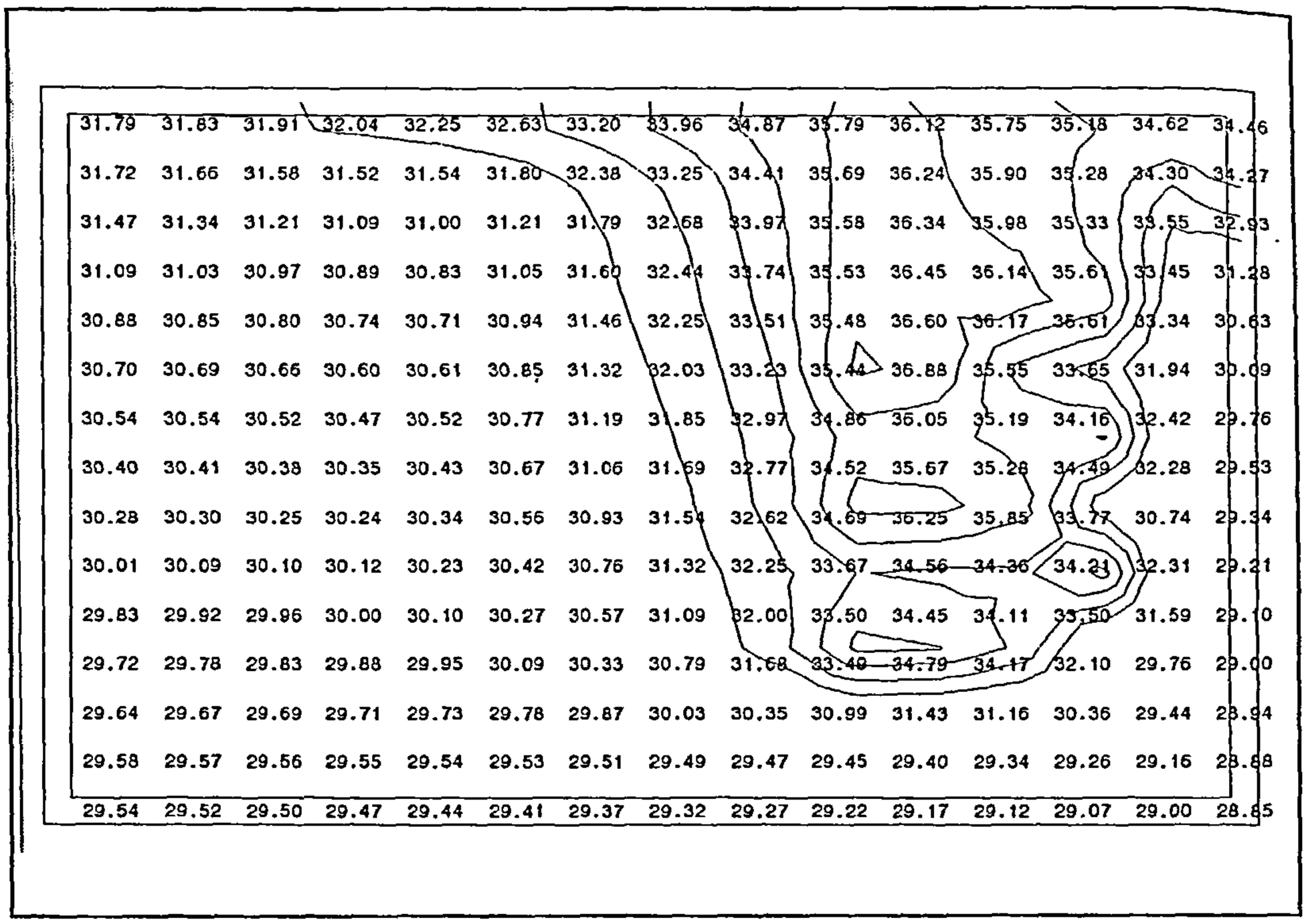


Figure 42 - simulated temperature contours at 3.6 m along Z-axis in X-Y plane (with radiation model)

4.4 Conclusions

a) A general method for the geometrical manipulation of polyhedral bodies has been presented, using methods developed in the field of computer graphics. It has been demonstrated that these methods may be used to calculate radiation view factors and internal solar tracking.

b) A review of calculation methods for solar angles and solar radiation incident on sloping surfaces has been presented together with the development of an algorithm for internal solar tracking.

c) The concept of the radiation view factor has been reviewed and the details of a numerical method for the calculation of view factors for arbitrarily orientated surfaces has been presented.

d) A general method for the calculation of radiative heat transfer between nonblackbody surfaces based on the radiosity concept has been discussed. Subsequently a more approximate method, involving the surface temperatures directly, has been presented. This latter method enables the solution for inter-surface longwave radiative heat exchange to be incorporated within the same solution scheme as that described in Chapter 3 for convection-conduction processes.

e) The influence of shortwave radiation on air flow and temperature distrbution and the potential of the radiation model has been demonstrated by example. The coupled radiation model could be applied to any situation where radiation may have a significant effect on air flow and temperature distribution, such as passive solar design.

5.0 COMPARISON OF MODEL PREDICTIONS WITH MEASURED RESULTS

In order to assess the predictive capabilities of the model, the results of calculations have been compared with measured results provided by Annex 20 of the International Energy Agency (IEA).

Annex 20 was established by the IEA in order to investigate the physical processes associated with airflows in buildings, within the framework of the International Energy Program.

As part of one of the projects undertaken by Annex 20, concerned with airflows in single zone spaces, known as Subtask 1, test rooms of a similar geometry have been established at a number of sites throughout Europe and North America. Arising from this work, sets of measured data have been made available for various categories of flow regime generated within the test rooms. The data supplied by Annex 20 for this study were measured by Jorma Heikkinen at the Technical University of Helsinki.

Three test cases representing different general flow categories have been selected by Annex 20 for investigation:

a) Test case B

Forced convection (isothermal) with supply air diffuser and exhaust opening. Three different supply air flow rates are considered. The specification of test case B is described by Heikkinen[65].

Case B1: air flow rate = 0.0158 m^3 (1.5 ac hr^{-1}), discharge velocity = 1.84 ms^{-1}
Case B2: air flow rate = 0.0315 m^3 (3.0 ac hr^{-1}), discharge velocity = 3.68 ms^{-1}
Case B3: air flow rate = 0.0630 m^3 (6.0 ac hr^{-1}), discharge velocity = 7.26 ms^{-1}

b) Test case D

Free convection under winter heating conditions with a radiator located beneath a cold window. Three combinations of radiator and window surface temperature are considered. The specification of test case D is described by Lemaire [66].

Case D1: radiator surface temperature = 46 °C
window surface temperature = 10 °C

Case D2: radiator surface temperature = 55 °C

window surface temperature = 5 °C

Case D3: radiator surface temperature = 65 °C

window surface temperature = 0 °C

c) Test case E

Mixed convection under summer cooling conditions with a supply diffuser, exhaust opening and warm window. Three combinations of supply flow rate, supply air temperature and window surface temperature are considered. The specification of test case E is described by Heikkinen [67].

Case E1: air flow rate = 0.0158 m^3 (1.5 ac hr^{-1}), discharge velocity = 1.84 ms^{-1}

supply air temperature = 10 °C

window surface temperature = 30 °C

Case E2: air flow rate = 0.0315 m^3 (3.0 ac hr^{-1}), discharge velocity = 3.68 ms^{-1}

supply air temperature = 15 °C

window surface temperature = 30 °C

Case E3: air flow rate = 0.0630 m^3 (6.0 ac hr^{-1}), discharge velocity = 7.26 ms^{-1}

supply air temperature = 15 °C

window surface temperature = 35 °C

The measured data includes sets of measured values of air speed and temperature at pre-defined grid points throughout the test room. The data is presented as records comprising the three-dimensional Cartesian co-ordinates of the measured point, measured air speed and measured temperature.

5.1 The Test Room

The dimensions, construction and configuration of the test room used by Heikkinen conform to the specification described by Lemaire[68].

The room dimensions are 4.2 m x 3.6 m x 2.5 m high. The walls of the room comprise 0.1 m insulation and a wooden sheet on the inner surface.

The test room dimensions are illustrated in Figure 43.

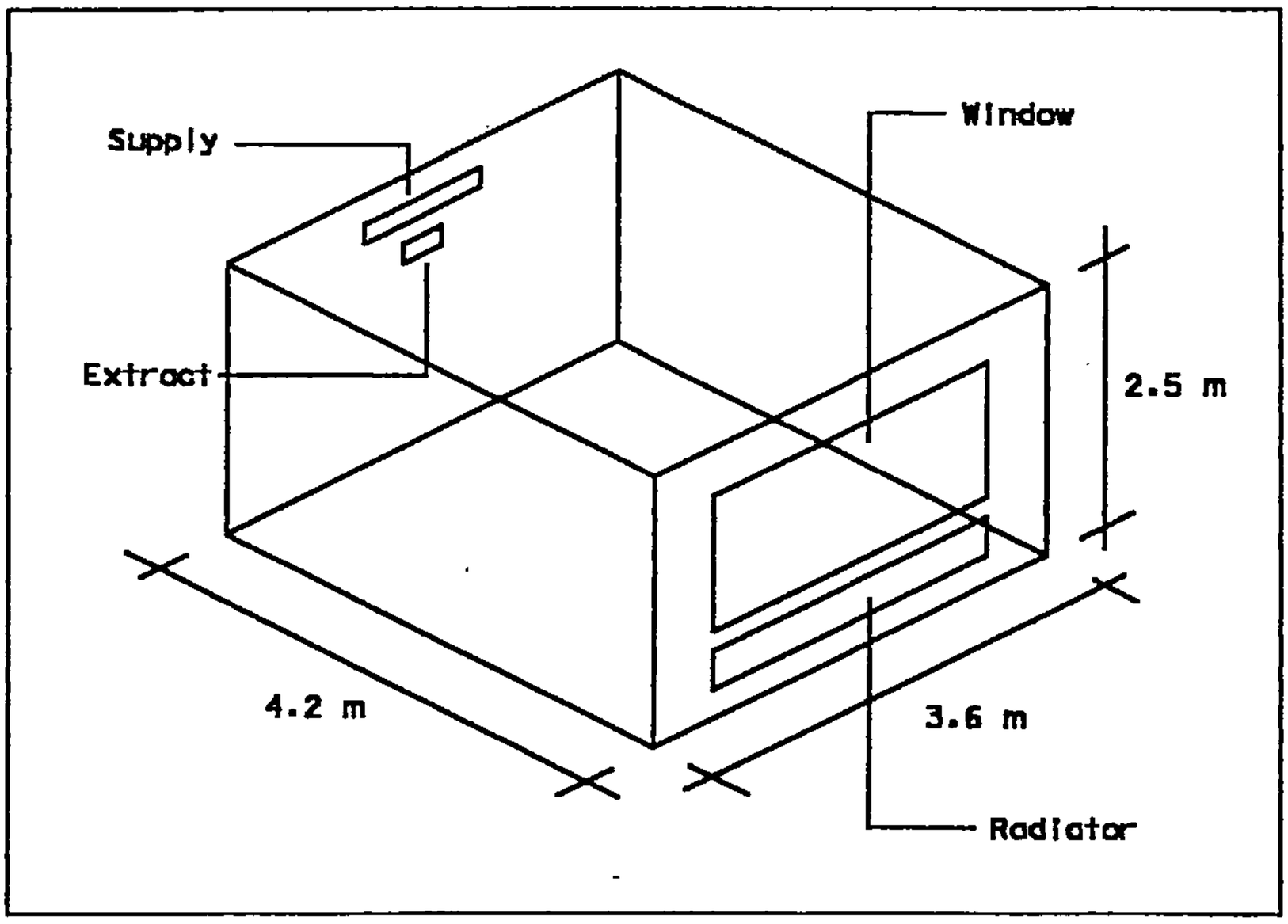


Figure 43 - test room

5.1.1 The Test Room Components

A number of components are available in order to provide the required boundary conditions for various flow regimes:

a) A supply air diffuser located in the rear wall.

b) An exhaust opening located below the supply diffuser in the rear wall.

c) A window located in the front wall.

d) A radiator located beneath the window in the front wall.

The configuration of room components is dependent upon the studied test case (refer to description of test cases B, D and E).

All room components are located so as to provide symmetry about the centre plane in the z-dimension. The location of room components are illustrated in Figure 24.

5.1.1.1 The Dimensions, Location and Construction of the Window

The window is constructed of a single sheet of 6 mm glass, the dimensions of which are 1.585 m x 1.985 m. The glass is mounted flush to the inner wall surface into an aperture of dimensions 1.6 m x 2.0 m. The top of the window is located 0.2 m from the ceiling in the front wall of the room.

5.1.1.2 The Dimensions and Location of the Supply and Exhaust Openings

The supply diffuser selected by Neilsen [69] for Subtask 1 is of the HESCO type. The diffuser comprises 84 nozzles in a 4 row x 21 configuration. The direction of each nozzle is individually adjustable.

The diffuser is mounted on the rear wall with a distance of 0.2 m between the ceiling and the inner upper surface of the diffuser. The dimensions of the opening required for the diffuser are 0.17 m x 0.71 m.

The exhaust opening is located in the rear wall, 0.2 below the inner lower surface of the supply diffuser. The dimensions of the exhaust opening are 0.2 m x 0.3 m

5.1.1.3 The Dimensions and Location of the Radiator

The radiator is a Stelrad single panel type P1, the dimensions of which are 0.3 m x 2.0 m. The radiator is mounted on wooden blocks, 0.1 m from the floor and 0.05 m from the front wall.

5.2 Test Case B

Test case B2 has been selected for the purpose of comparison. The measured results are presented in the form of graphical velocity contours and mean velocity plots in Figures 44 and 45.

Geometric polyhedra (see Section 4.1) are defined to represent the external and internal dimensions of the test room. The ASCII geometry file for test case B2 (*ann20b2.001*) in included in Appendix III.

A significant problem encountered in the specification of the model geometry for test case B is the description of the air supply diffuser (see Section 5.1.1.2). In order to model the diffuser explicitly,

the required grid resolution would be computationally impractical. As a first approximation, the 'basic model' approach outlined by Neilsen [70] was adopted. The 'basic model' comprises an opening of 0.18 m x 0.062 m equivalent to the effective flow area of the diffuser nozzles and retaining the diffuser aspect ratio. A velocity of 3.68 m s^{-1} at an upward angle of 40° to the horizontal is prescribed for the diffuser outlet. Outflow turbulence kinetic energy and dissipation rate are derived using the expressions detailed in Section 3.4.4. The ASCII boundary conditions file for test case B2 *(ann20b2.b01)* is included in Appendix III.

The finite volume grid comprises 18 x 14 x 16 cells in the X, Y and Z dimensions respectively, an irregular grid mesh being required to fit the supply and extract dimensions and to provide finer spacing in the vicinity of walls.

Convergence was achieved after 300 iterations, taking approximately 5 minutes on a 350 MHz Pentium II type PC. Convergence was deemed to have been achieved when all dependent variable residuals were less than 10^{-6} in magnitude.

The results of calculation for the 'basic model' are presented in the form of graphical mean air speed contours, velocity vectors and variational profiles in Figures 46-47 and 50-51.

In an attempt to provide a more realistic representation of the HESCO diffuser, a 'multiple jet model' was developed. The outlet area of the diffuser was divided into five equal squares having a dimension of 0.0498 m, spread across the actual diffuser width, in order to stimulate the spread of the diffuser. The dimensions of each square was obtained from the required volume flow rate and outlet velocity. The outlet velocity, direction, turbulence kinetic energy and dissipation rate for each nozzle were each prescribed in the same fashion as for the 'basic model'.

The finite volume grid was increased in resolution to 25 x 16 x 21 cells in order to cater for the multiple jet configuration.

Convergence was again achieved after 300 iterations, taking approximately 5 minutes on a 350 MHz Pentium II type PC. Convergence was deemed to have been achieved when all dependent variable residuals were less than 10^{-6} in magnitude.

The results of calculation for the 'multiple jet model' are presented in the form of graphical mean air speed contours, velocity vectors and variational profiles in Figures 48-49 and 52-53.

Comparing the measured mean air speed contours in Figures 44 and 45 with the `basic model' and `multiple jet model' both in terms of flow patterns and mean air speeds. Mean air speeds predicted using the `basic model' are significantly higher than measured results throughout the flow domain whereas the `multiple jet model' results compare very favourably. The predicted air speed in the near floor region using the `multiple jet model' ranges between 0.02 - 0.14 ms^{-1}.

Referring to Figures 45, 51 and 53 which illustrate the mean air speed contours in the X-Y plane at 0.05 mg below the ceiling, very good agreement can be observed between measured results and the `multiple jet model' in terms of the overall shape and projection of the jet along the ceiling, although the measured results would suggest slightly greater penetration than the predicted results. The `basic model' results in too wide a spread of the jet.

Referring to Figure 54 which shows the variation of mean air speed with relative distance along the Y-axis, it can be seen that the predicted mean air speeds compare well with the measured air speeds for both the `basic model' and the `multiple jet model'. For both model types, the mean air speed can be seen to exceed the measured air speed at the height of the diffuser by 0.1 - 0.15 ms^{-1}, although there is better agreement with the `multiple jet model'.

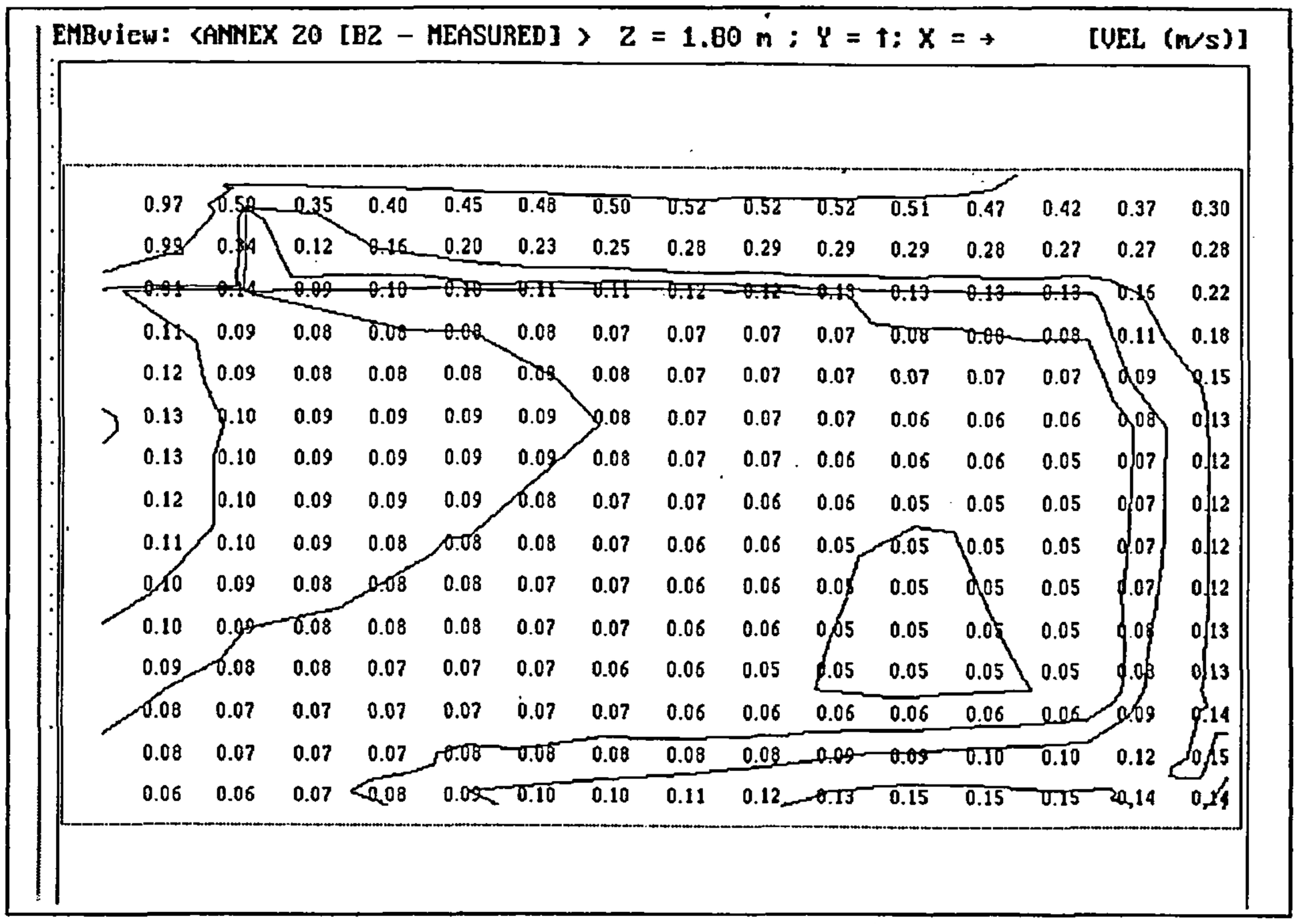


Figure 44 - test case B2: measured velocity contours on centre-line of Z-dimension in X-Y plane

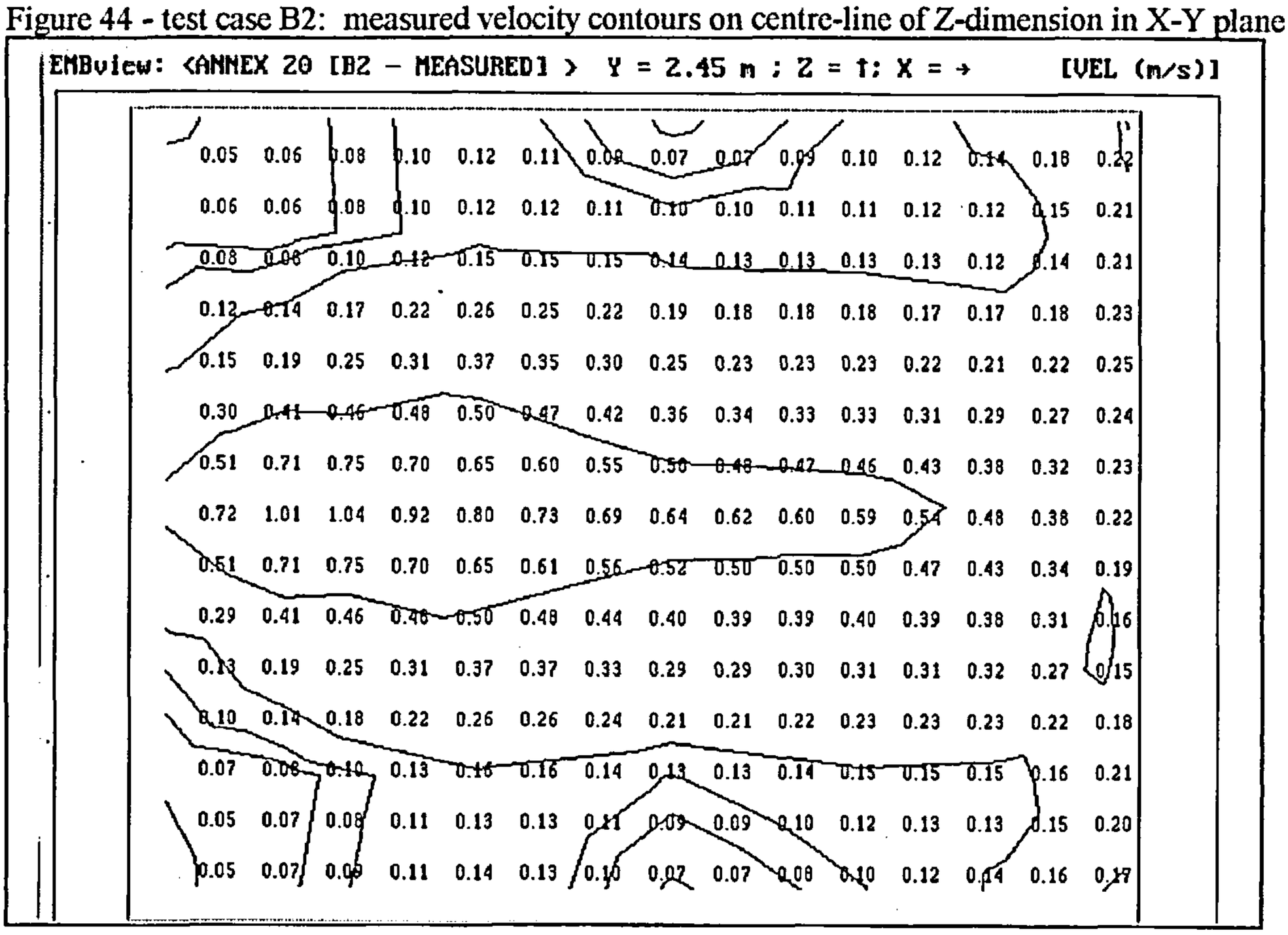


Figure 45 - test case B2: measured velocity contours at 0.05 m below ceiling in X-Z plane

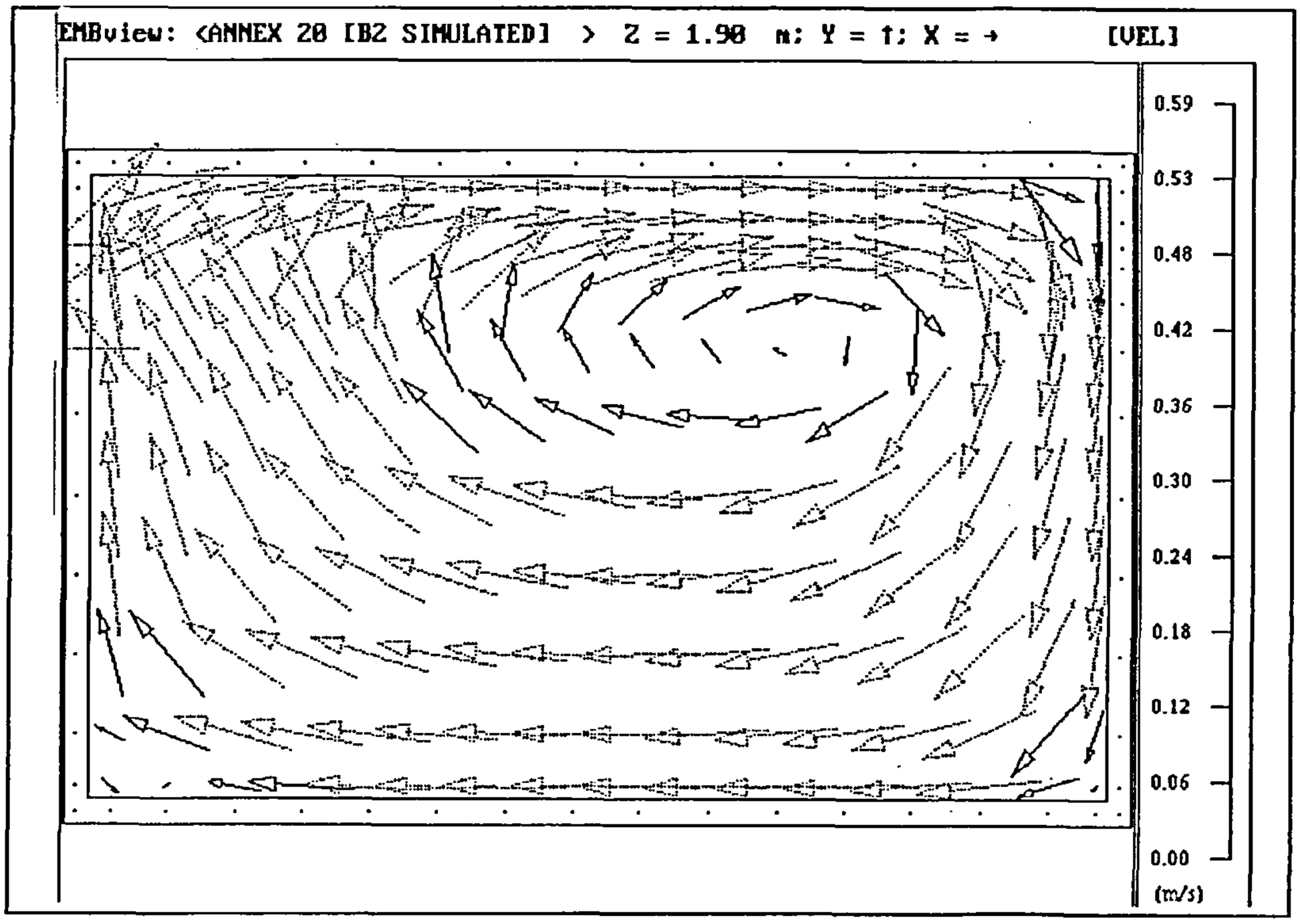


Figure 46 - test case B2: simulated velocity vectors on centre-line of Z-dimension in X-Y plane (basic model)

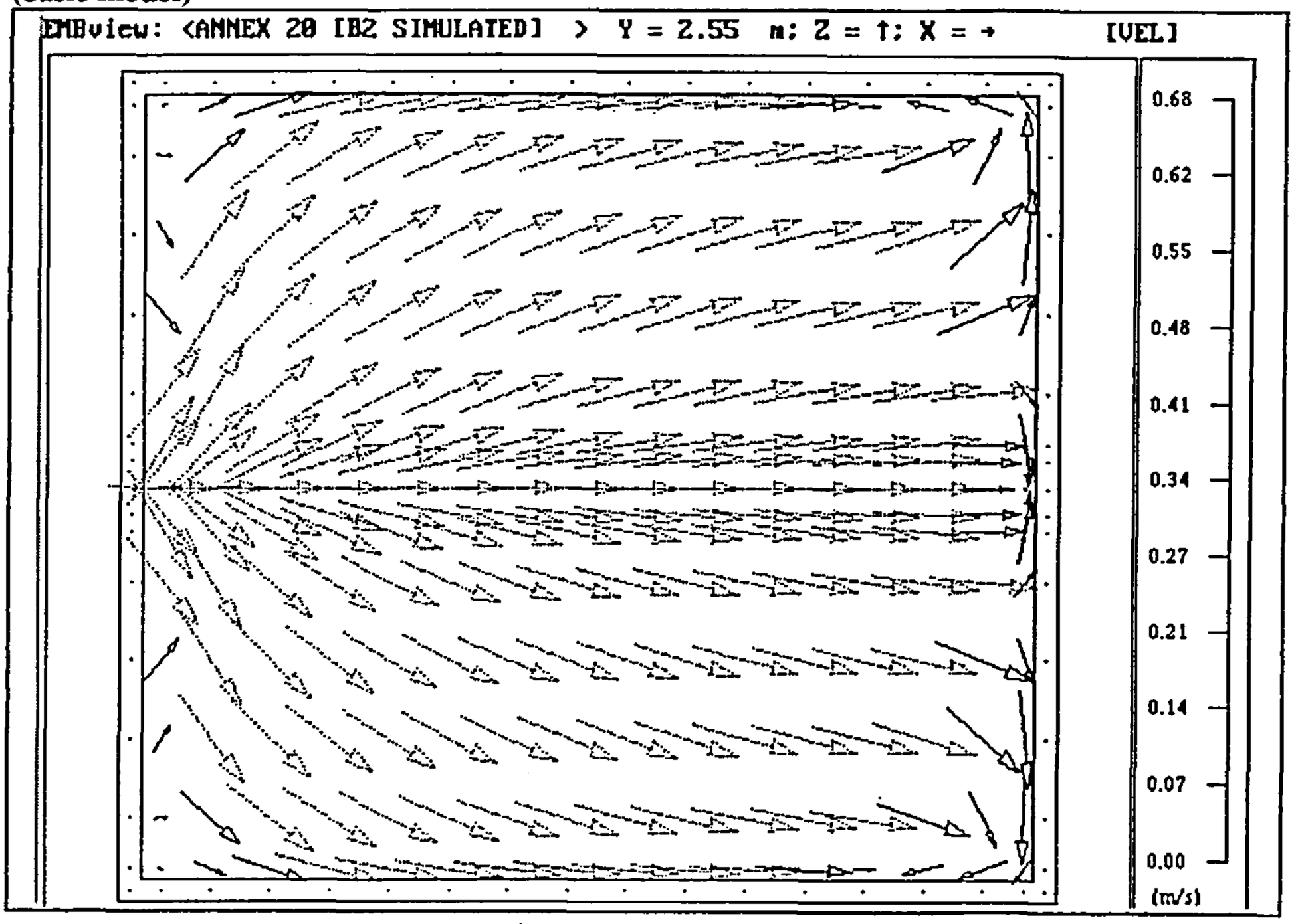


Figure 47 - test case B2: simulated velocity vectors at 0.05 m below ceiling in X-Z plane (basic model)

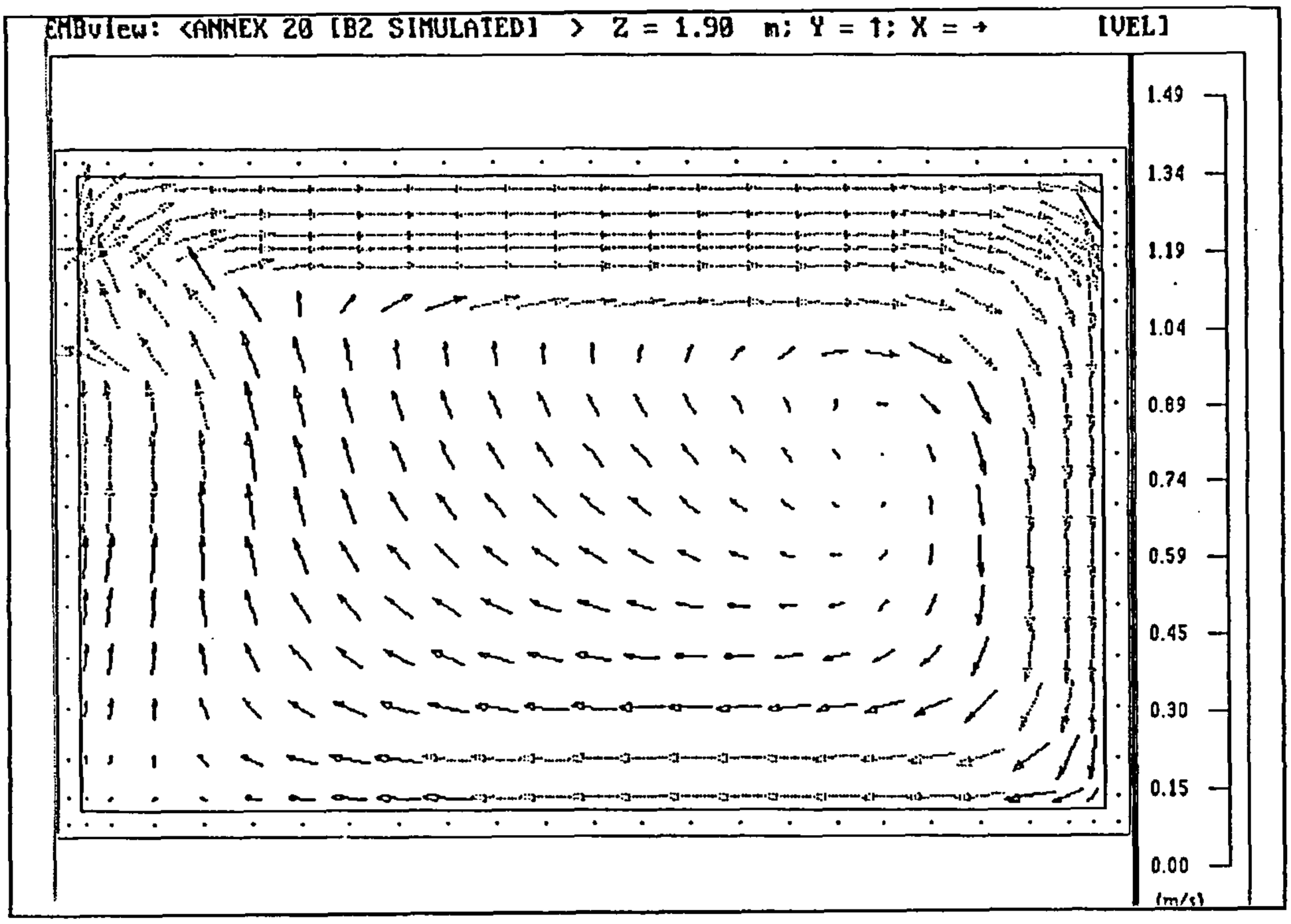


Figure 48 - test case B2: simulated velocity vectors on centre-line of Z-dimension in X-Y plane (multiple jet model)

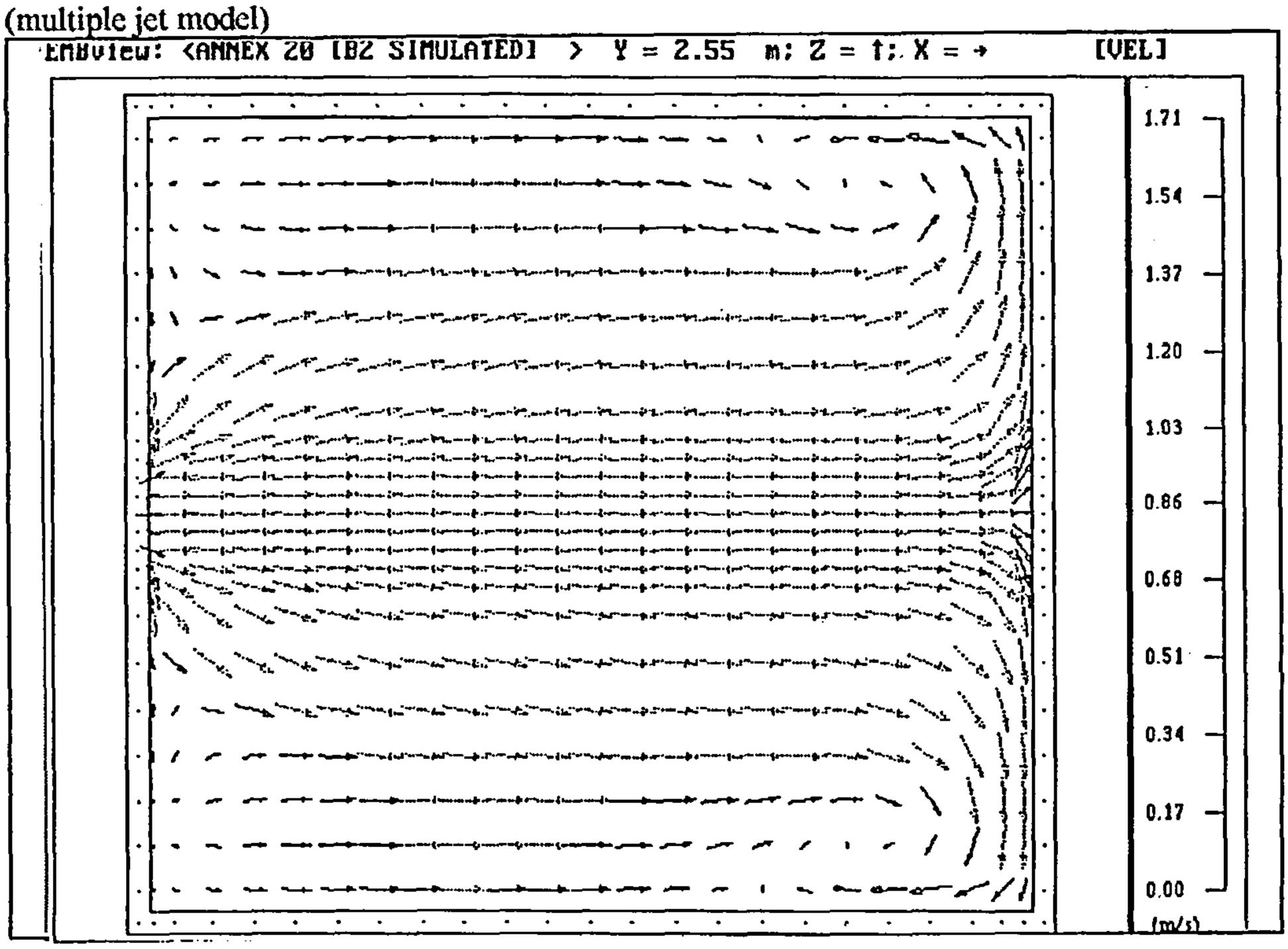


Figure 49 - test case B2: simulated velocity vectors at 0.05 m below ceiling in X-Z plane (multiple jet model)

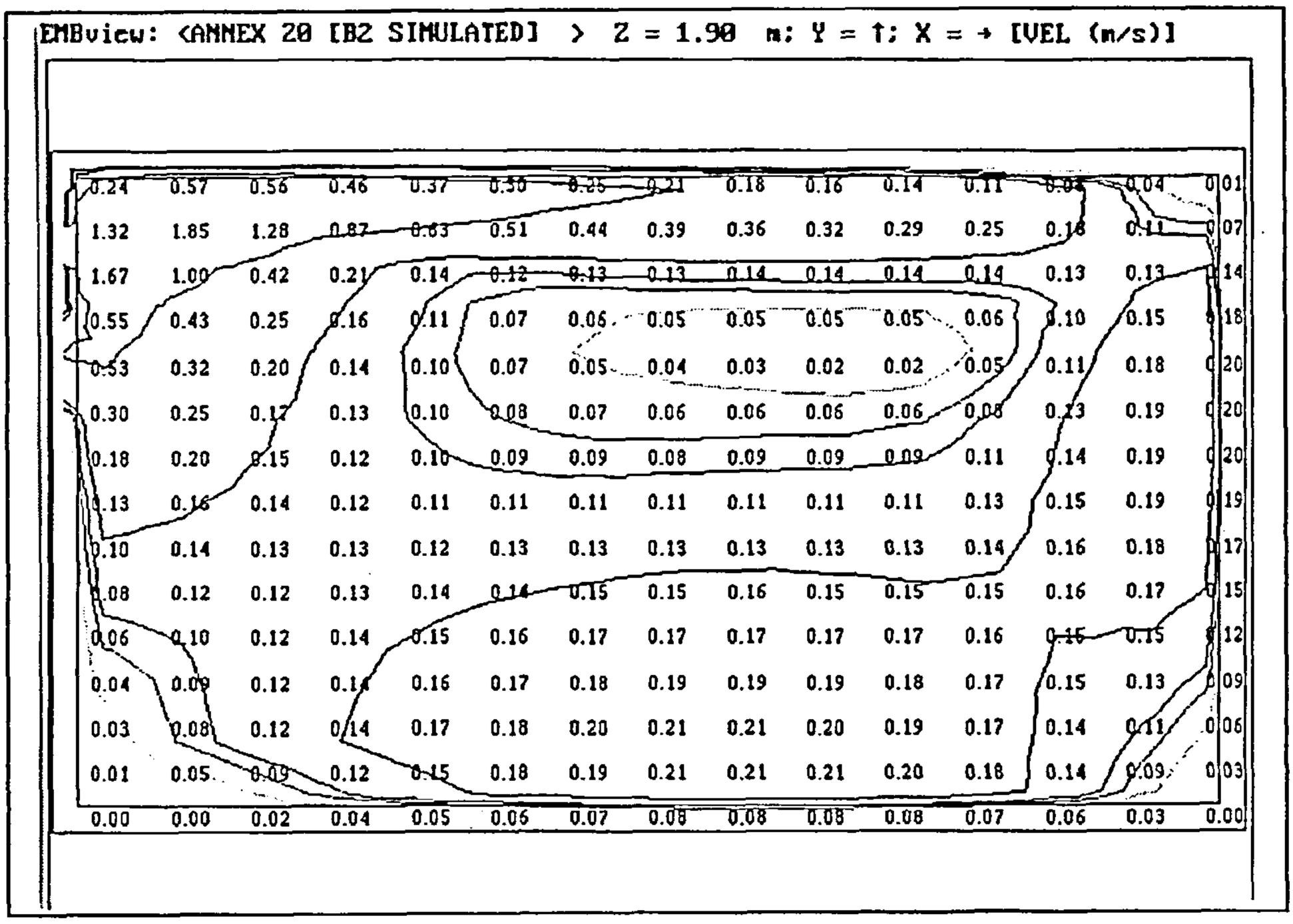


Figure 50 - test case B2: simulated velocity contours on centre-line of Z-dimension in X-Y plane (basic model)

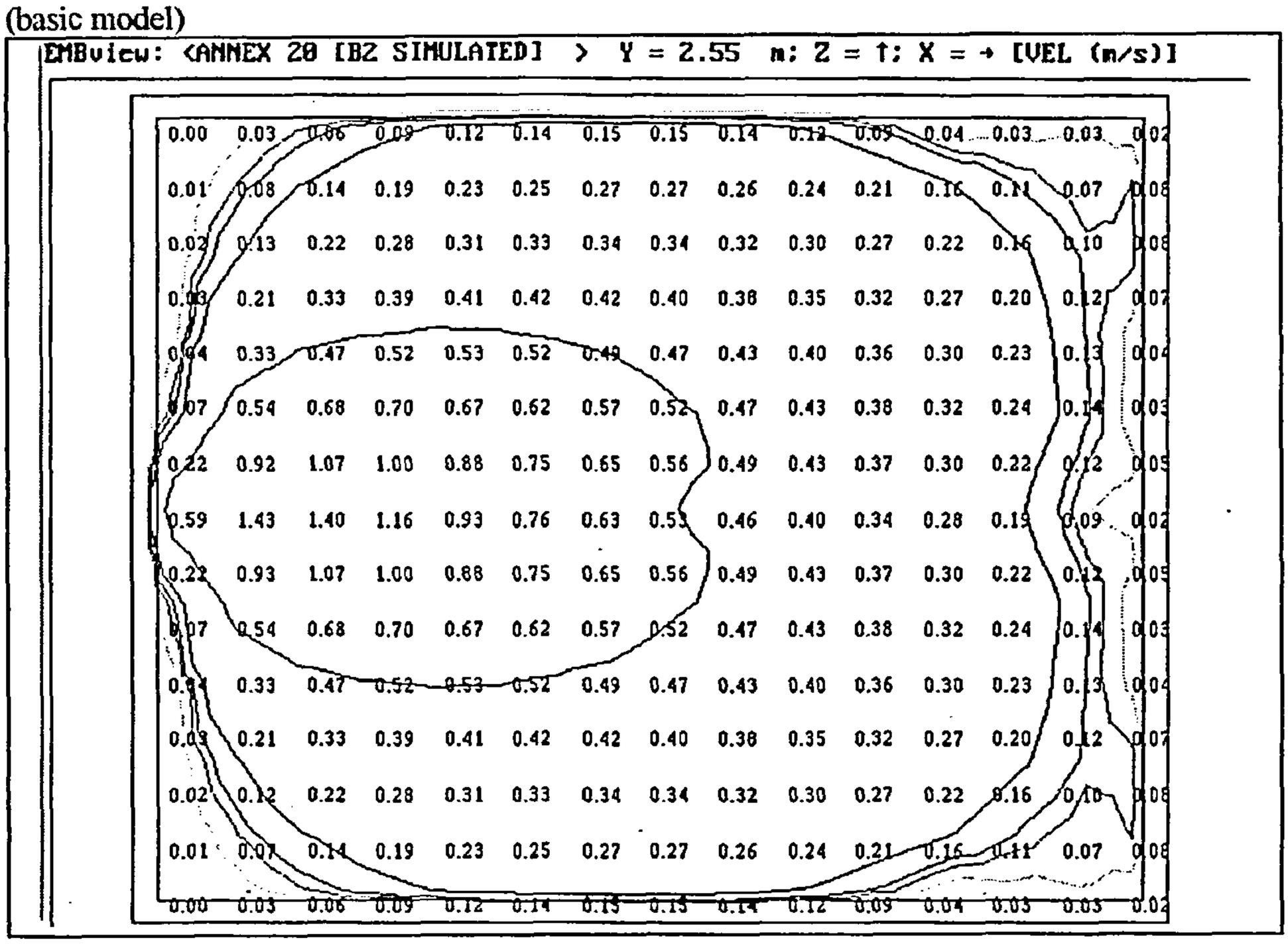


Figure 51 - test case B2: simulated velocity contours at 0.05 m below ceiling in X-Z plane (basic model)

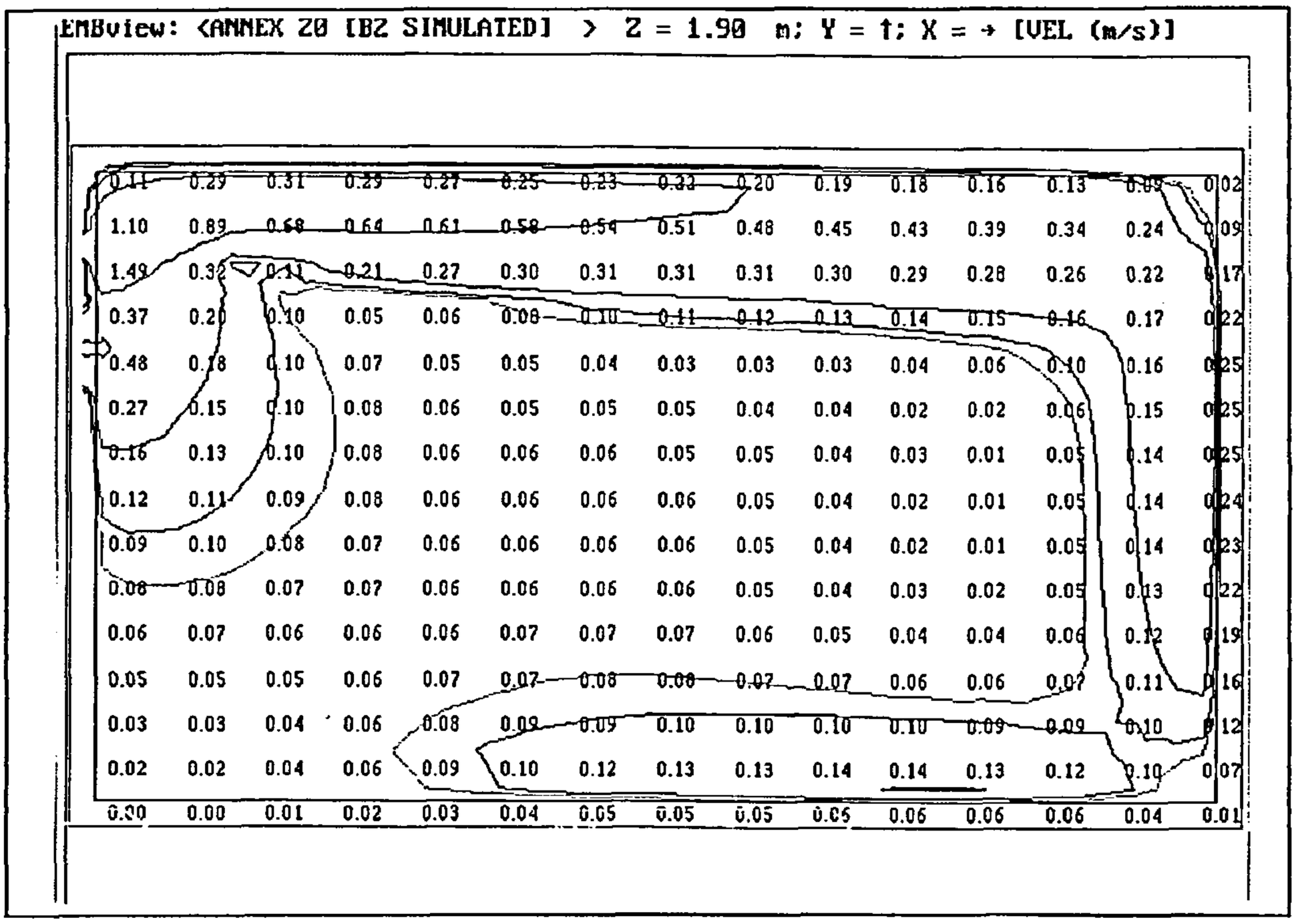


Figure 52 - test case B2: simulated velocity contours on centre-line of Z-dimension in X-Y plane (multiple jet model)

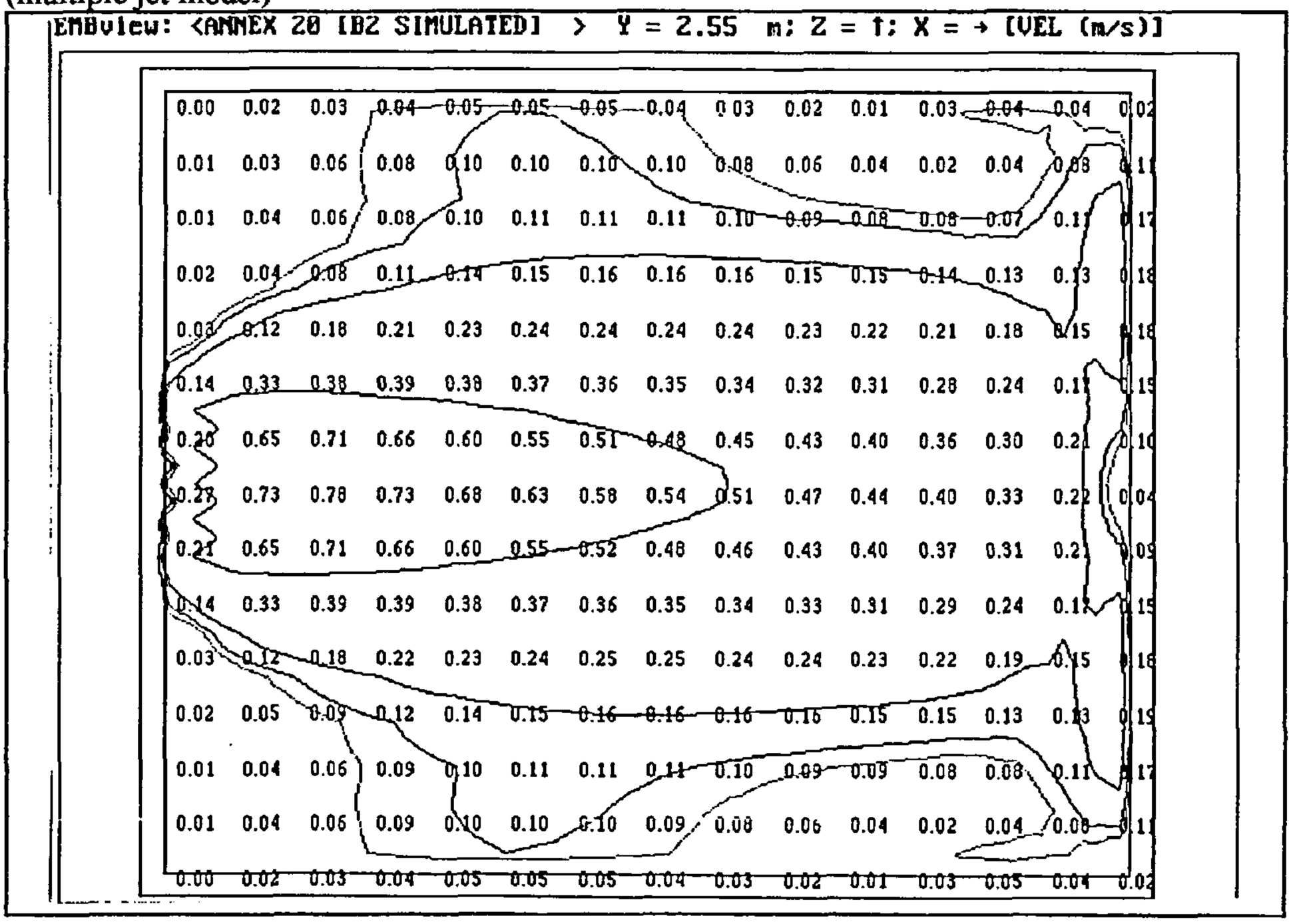


Figure 53 - test case B2: simulated velocity contours at 0.05 m below ceiling in X-Z plane (multiple jet model)

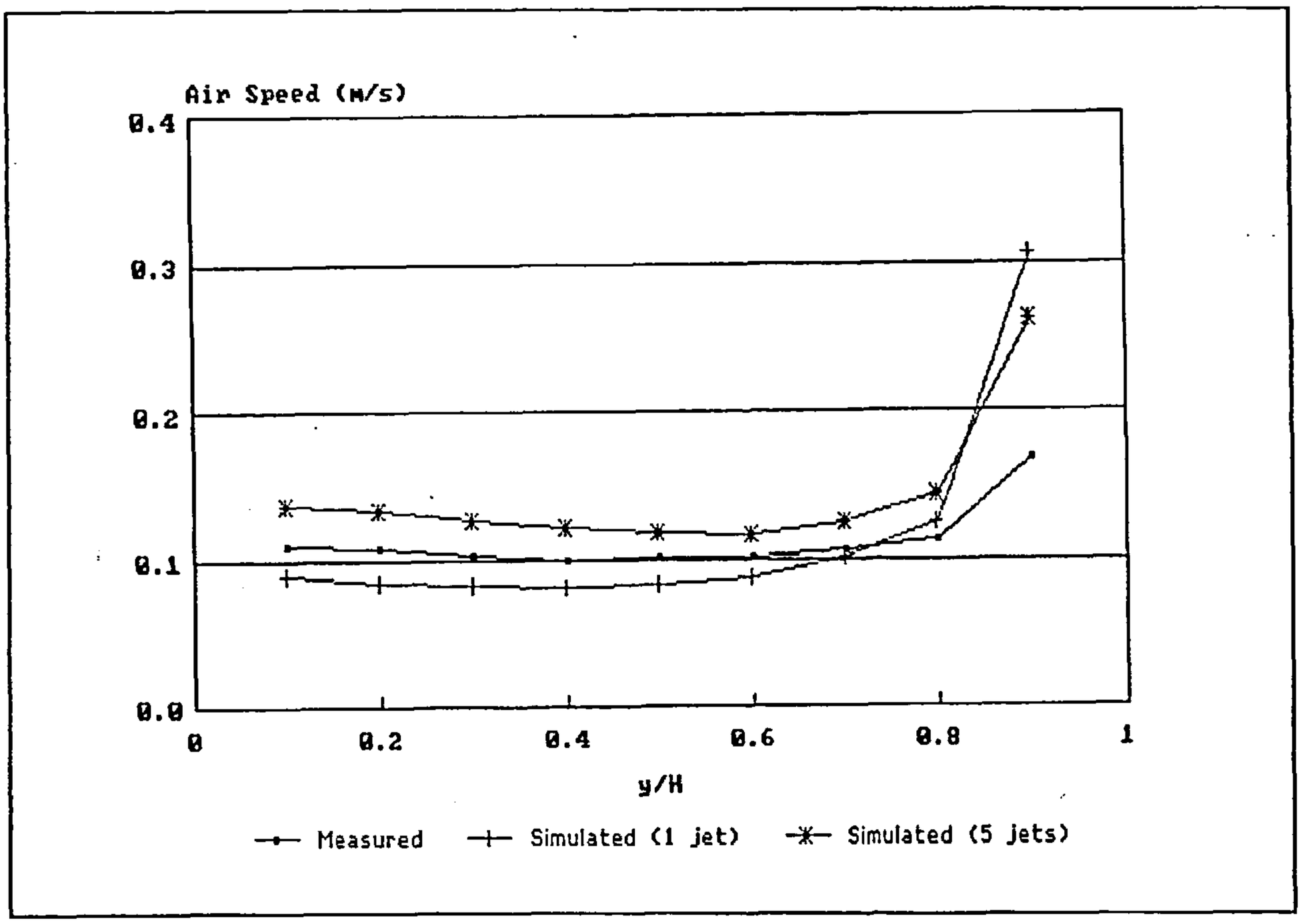


Figure 54 - test case B2: variation of mean air speed with relative distance on the Y-axis

5.3 Test Case E

Test case E2 has been selected for the purpose of comparison.

On the basis of the test case B results, the `multiple jet model' was selected as the preferred diffuser model for test case E. Two sets of calculation have been conducted for test case E2 for the purpose of comparison, firstly excluding a radiation model and secondly including the radiation model described in Section 4.3. Boundary conditions for velocities, turbulence kinetic energy and dissipation rate were prescribed in the same fashion as for test case B2. Surface heat transfer is calculated using the method detailed in Section 4.4.2. Heat transfer from the walls to the outside is assumed to be zero (i.e. adiabatic boundary conditions).

A finite volume grid of 21 x 16 x 19 cells was employed for the simulation, an irregular grid being defined to fit the supply jets, extract aperture and window. The geometric polyhedron representation of the test room (see Section 4.1), developed for test case B2 was modified to include additional surfaces in order to represent the window in the radiation model.

Convergence was achieved after approximately 300 iterations for both calculations including and excluding the radiation model. Calculations took approximately 6 minutes on a 350 MHz Pentium II

II type PC. Convergence was deemed to have been achieved when all dependent variable residuals were less than 10-6 in magnitude.

Initial results indicated severe asymmetry about the test room z-axis centre-line which were first considered to imply a numerical problem. Various grid resolutions were applied to investigate the dependence of the asymmetry on grid dimensions, however the same asymmetry was manifest in all cases. In order to minimise the possibility of a coding error, the same model and boundary conditions were defined using a commercially available code (FLOVENT) where the same asymmetry was again seen to occur. The underlying reason for the asymmetry was deemed to be beyond the scope of this study.

The measured results are presented in the form of graphical velocity and temperature contours and mean velocity plots in Figures 55-58. The calculated results are presented in the form of graphical velocity vectors, temperature and velocity contours and mean velocity plots in Figures 59-70. Velocity and temperature profiles are presented in Figures 71-74.

Due to the asymmetry, the predicted results must be viewed with some caution, although there is seen to be quite good agreement between the mean air speed contours in both the X-Y and X-Z planes, particularly in the case where the radiation model has been excluded, the temperatures are depressed by 0.5-1K below the radiation model results throughout the flow domain. Referring to the speed and temperature profiles in Figures 51-66 for the measured results and predicted results using the radiation model, it can be seen that while the mean air speeds compare favourably, the predicted temperatures are about 1K below the measured temperatures, suggesting that the surface heat exchange method might benefit from some refinement.

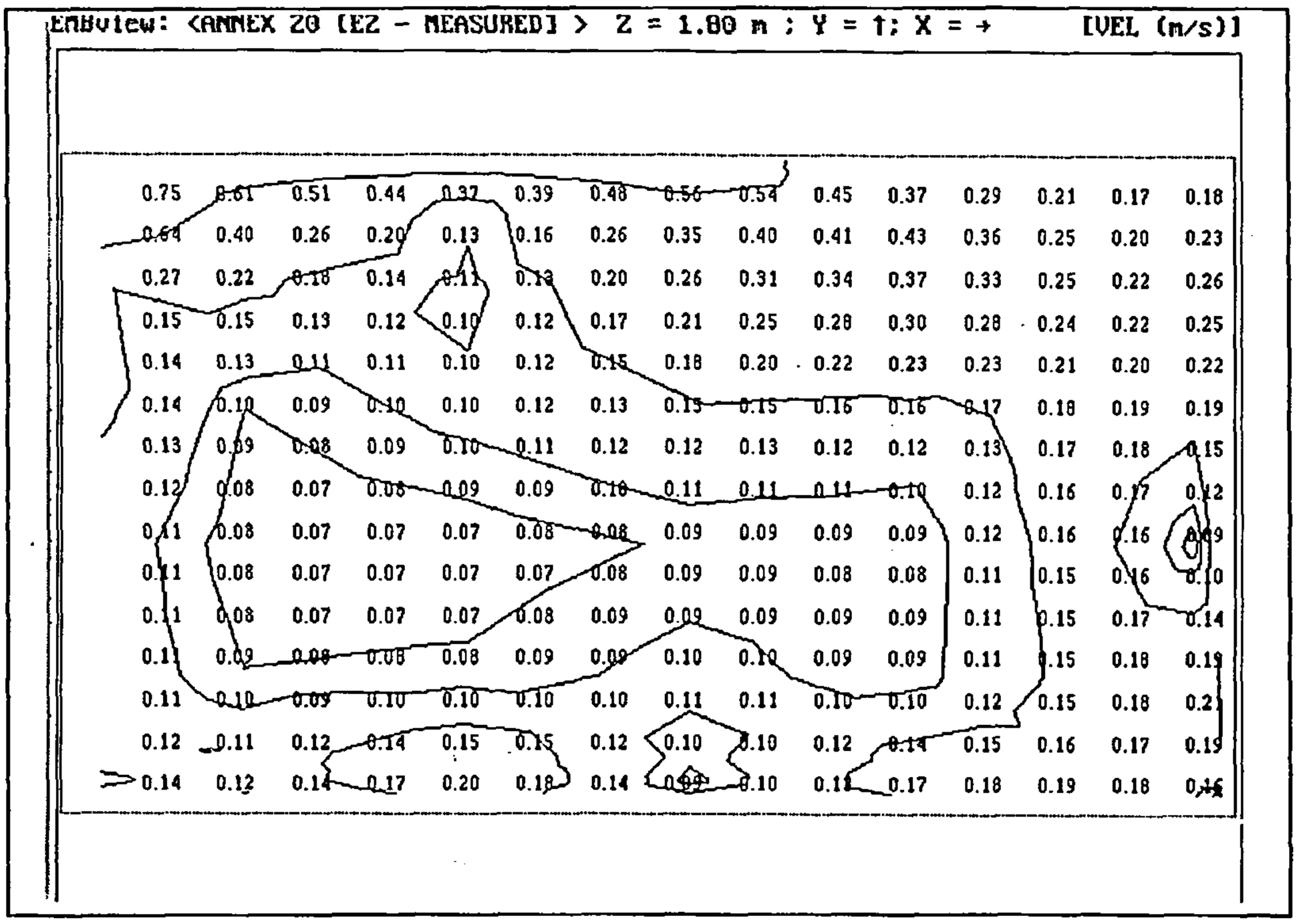


Figure 55 -test case E2: measured velocity contours on centre-line of z-dimension in x-y plane

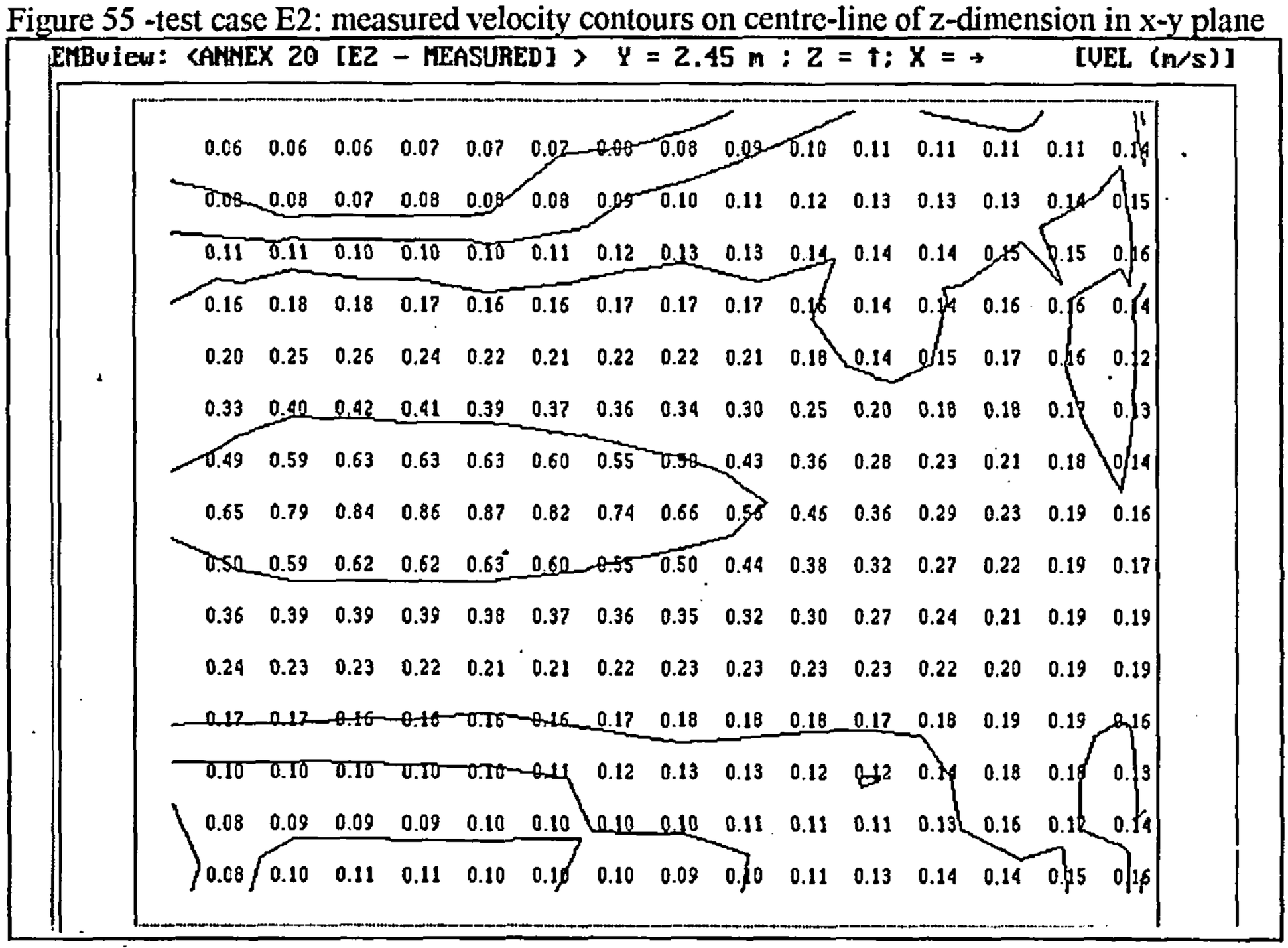


Figure 56 -test case E2: measured velocity contours at 0.05 m below ceiling in x-z plane

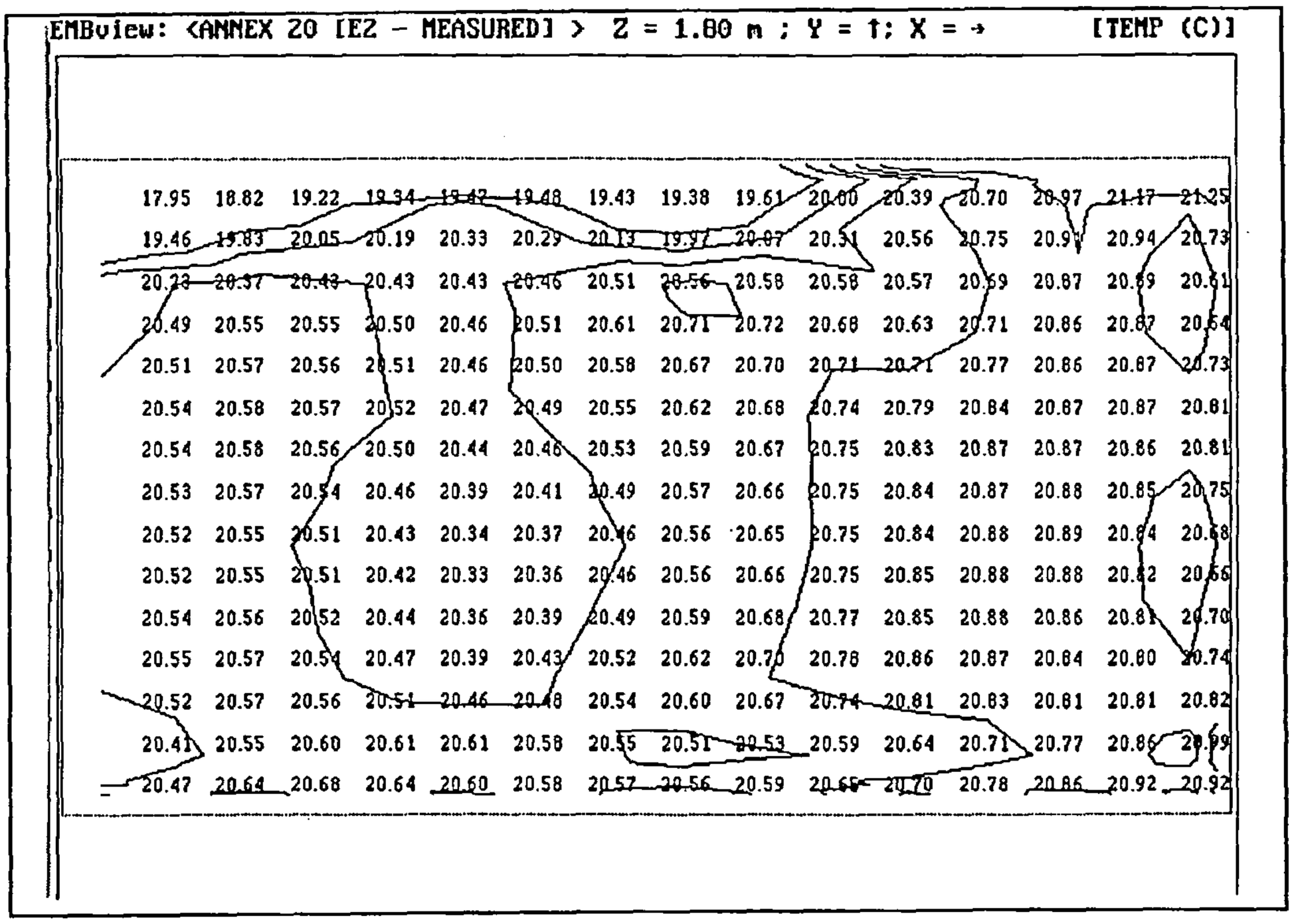


Figure 57 -test case E2: measured temperature contours on centre-line of z-dimension in x-y plane

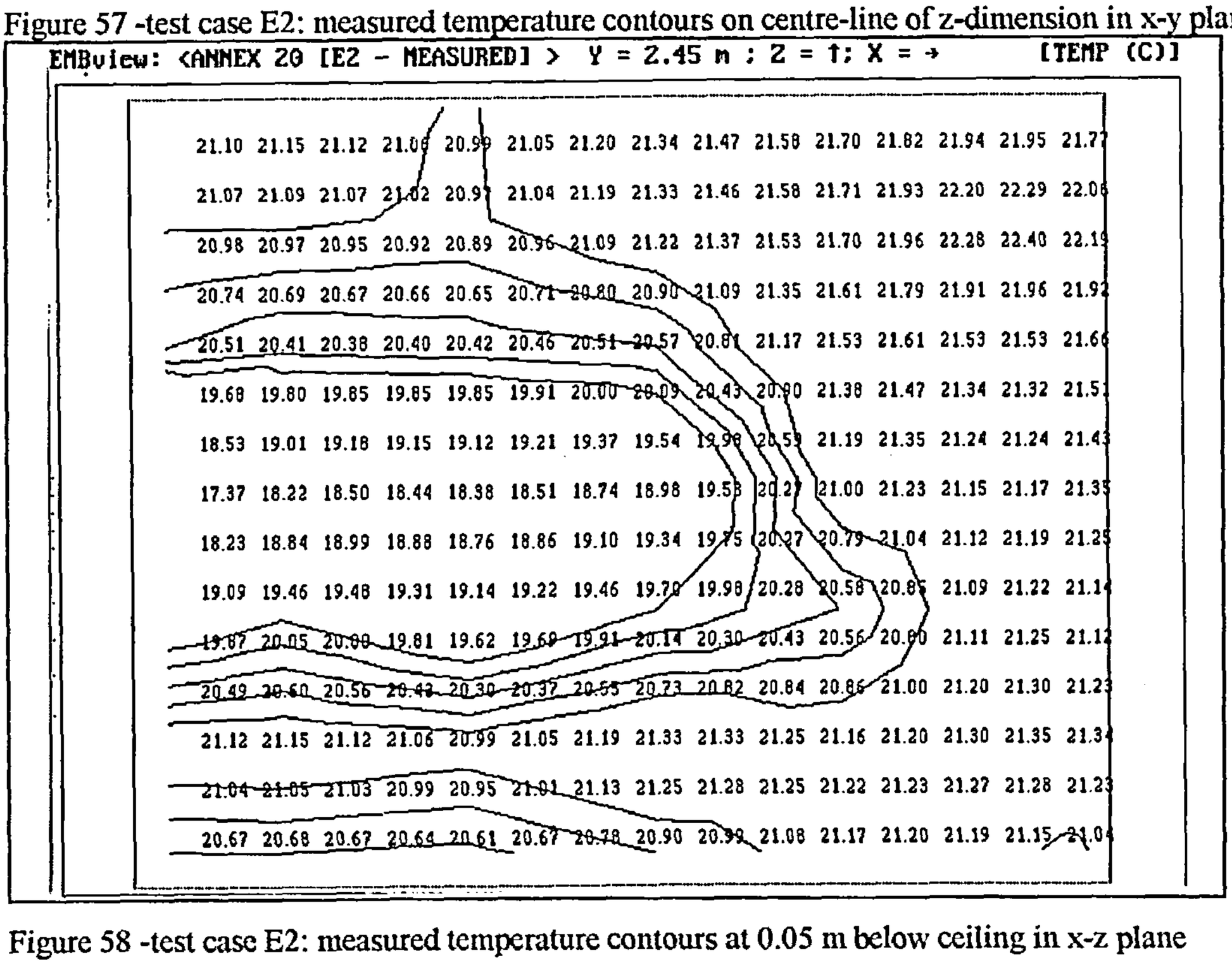


Figure 58 -test case E2: measured temperature contours at 0.05 m below ceiling in x-z plane

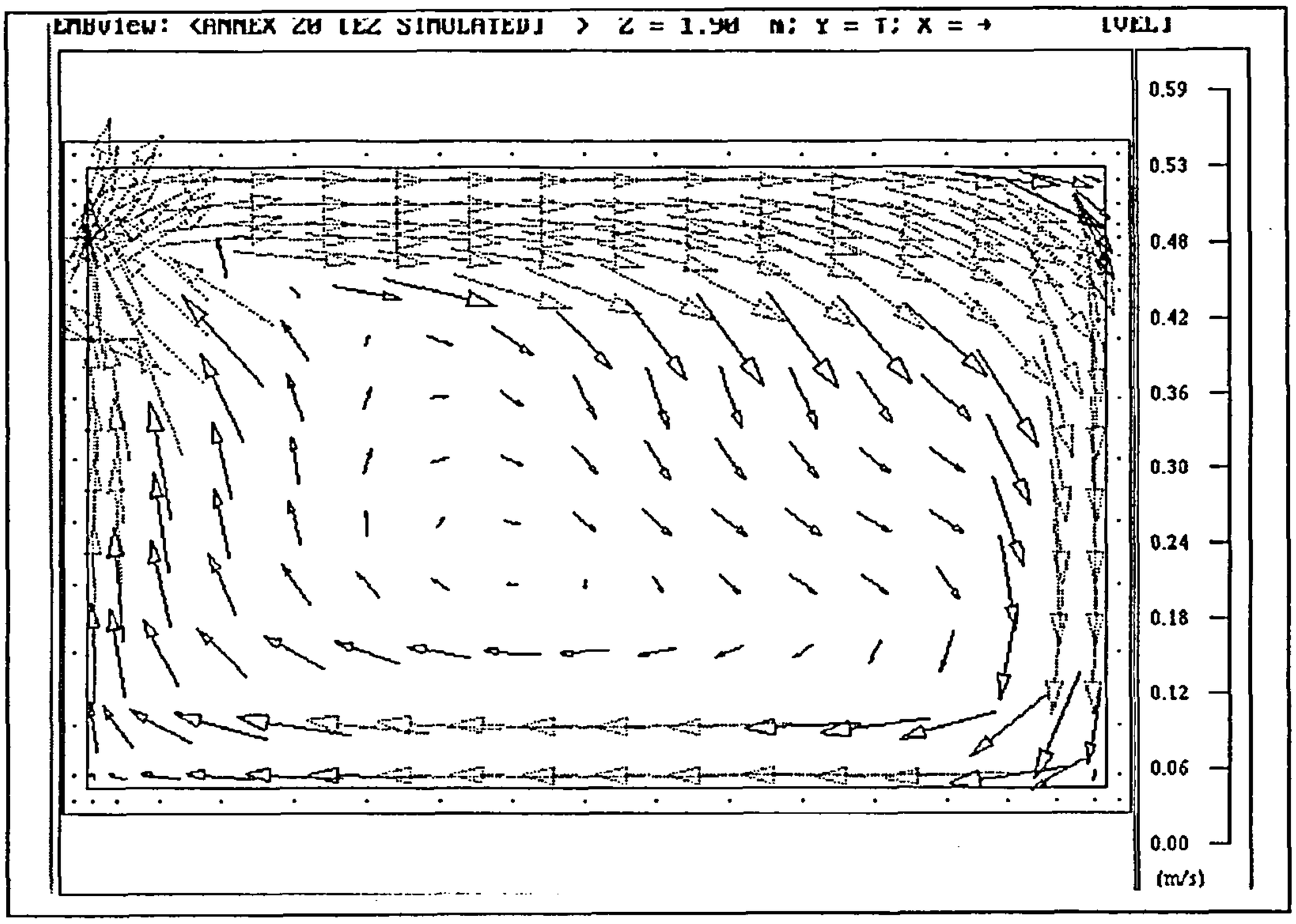


Figure 59 -test case E2: simulated velocity vectors on centre-line of z-dimension in x-y plane (excluding radiation model)

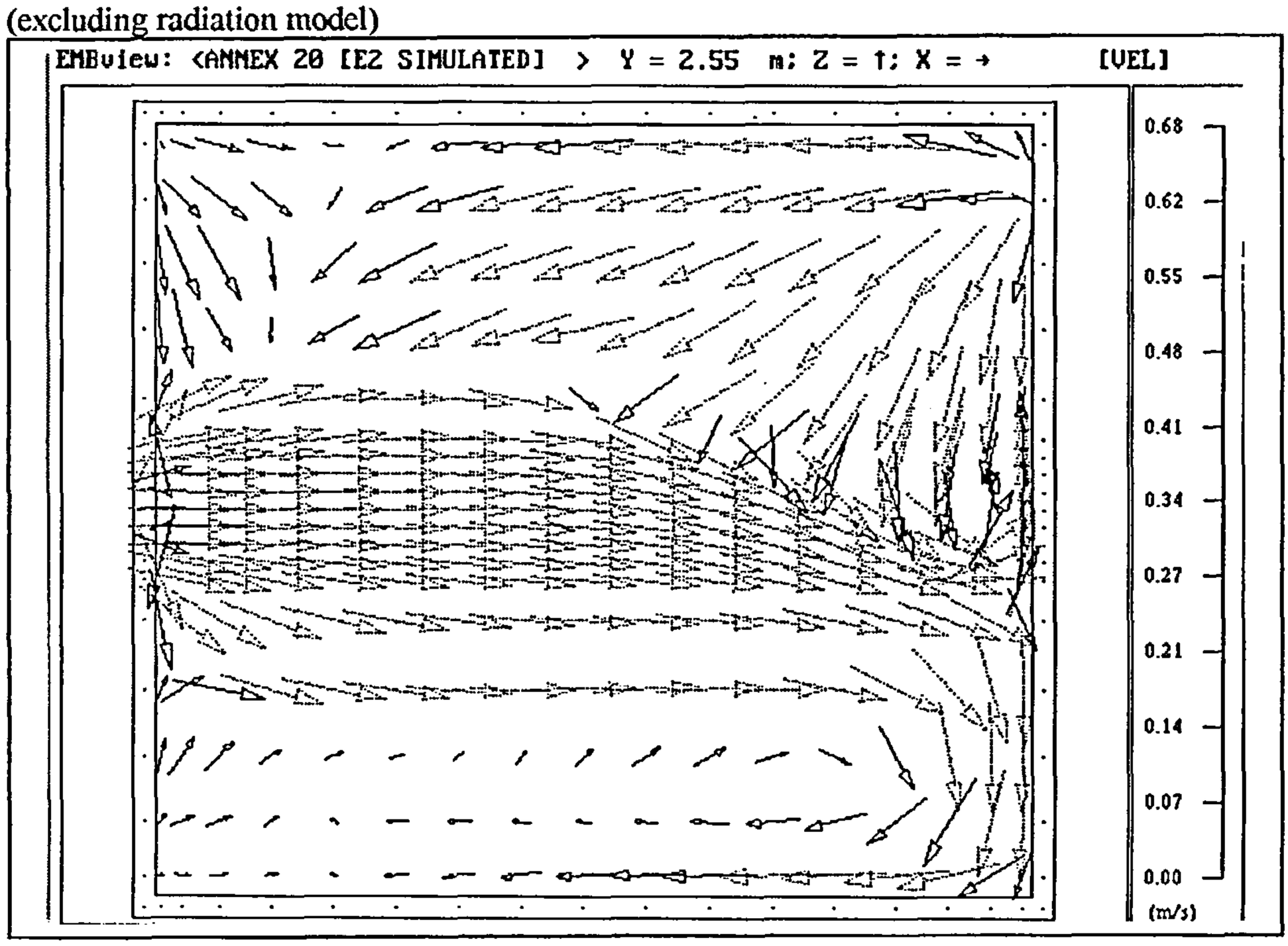


Figure 60 -test case E2: simulated velocity vectors at 0.05 m below ceiling in x-z plane (excluding radiation model)

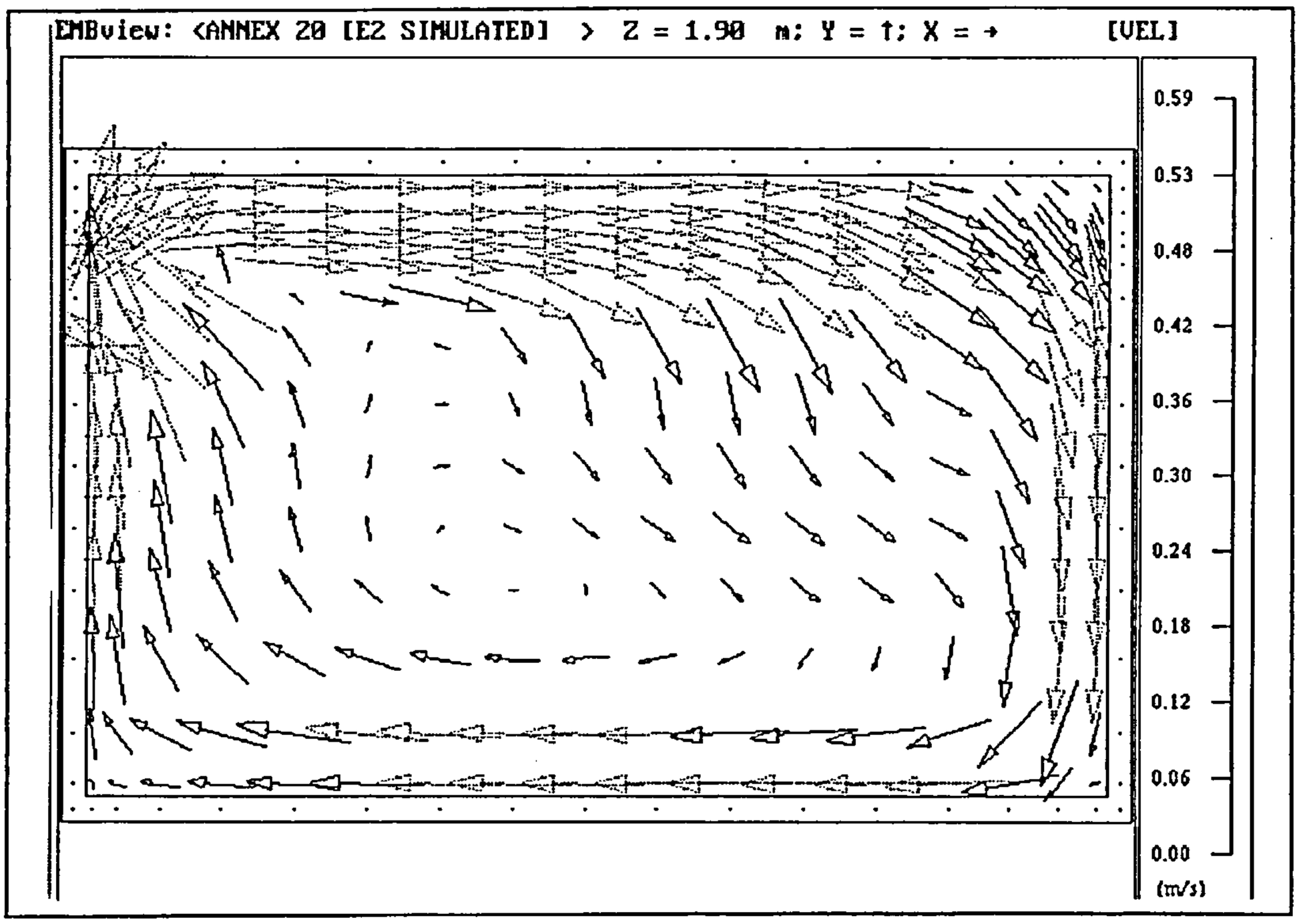


Figure 61 -test case E2: simulated velocity vectors on centre-line of z-dimension in x-y plane (including radiation model)

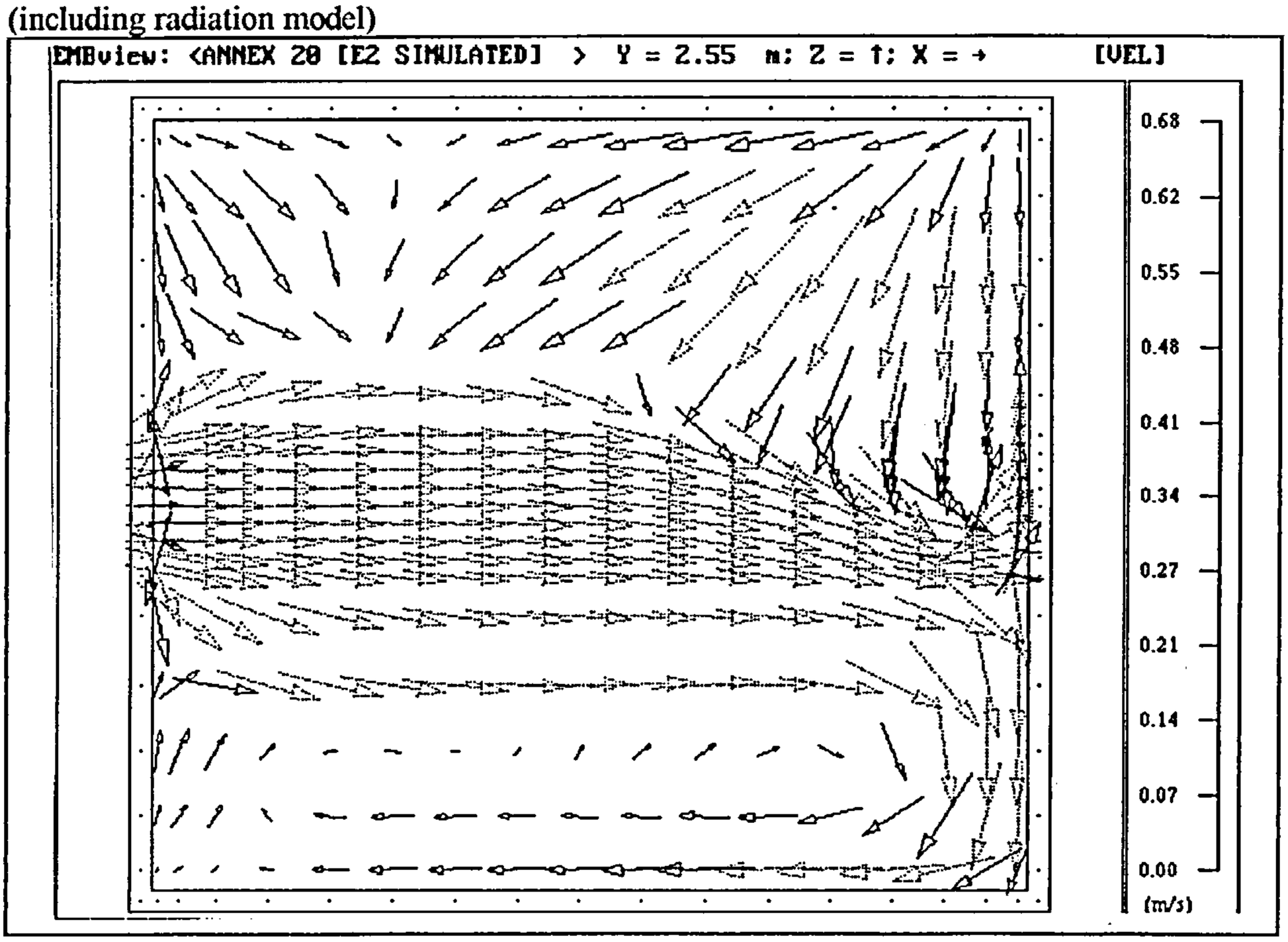


Figure 62 -test case E2: simulated velocity vectors at 0.05 m below ceiling in x-z plane (including radiation model)

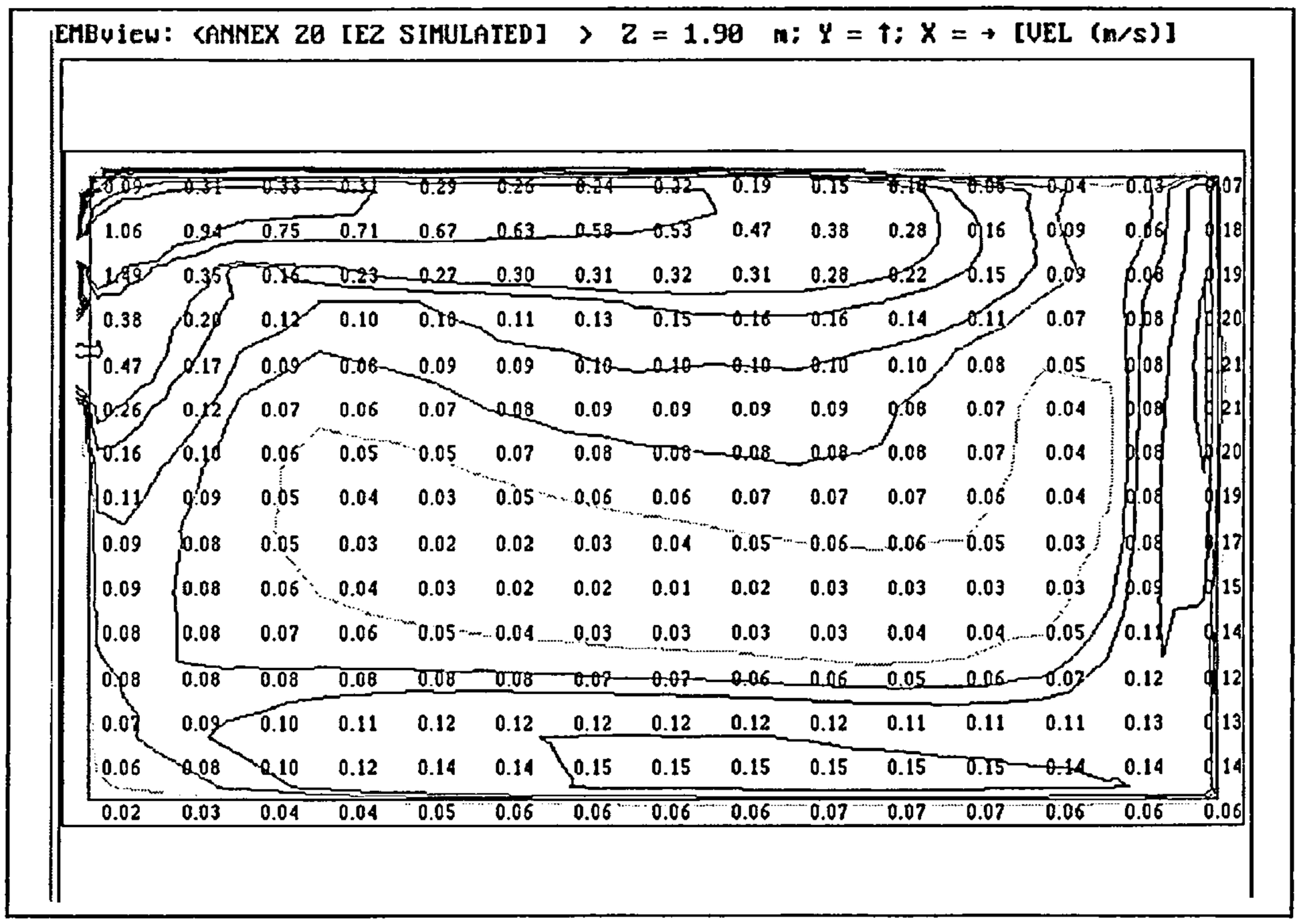


Figure 63 -test case E2: simulated velocity contours on centre-line of z-dimension in x-y plane (excluding radiation model)

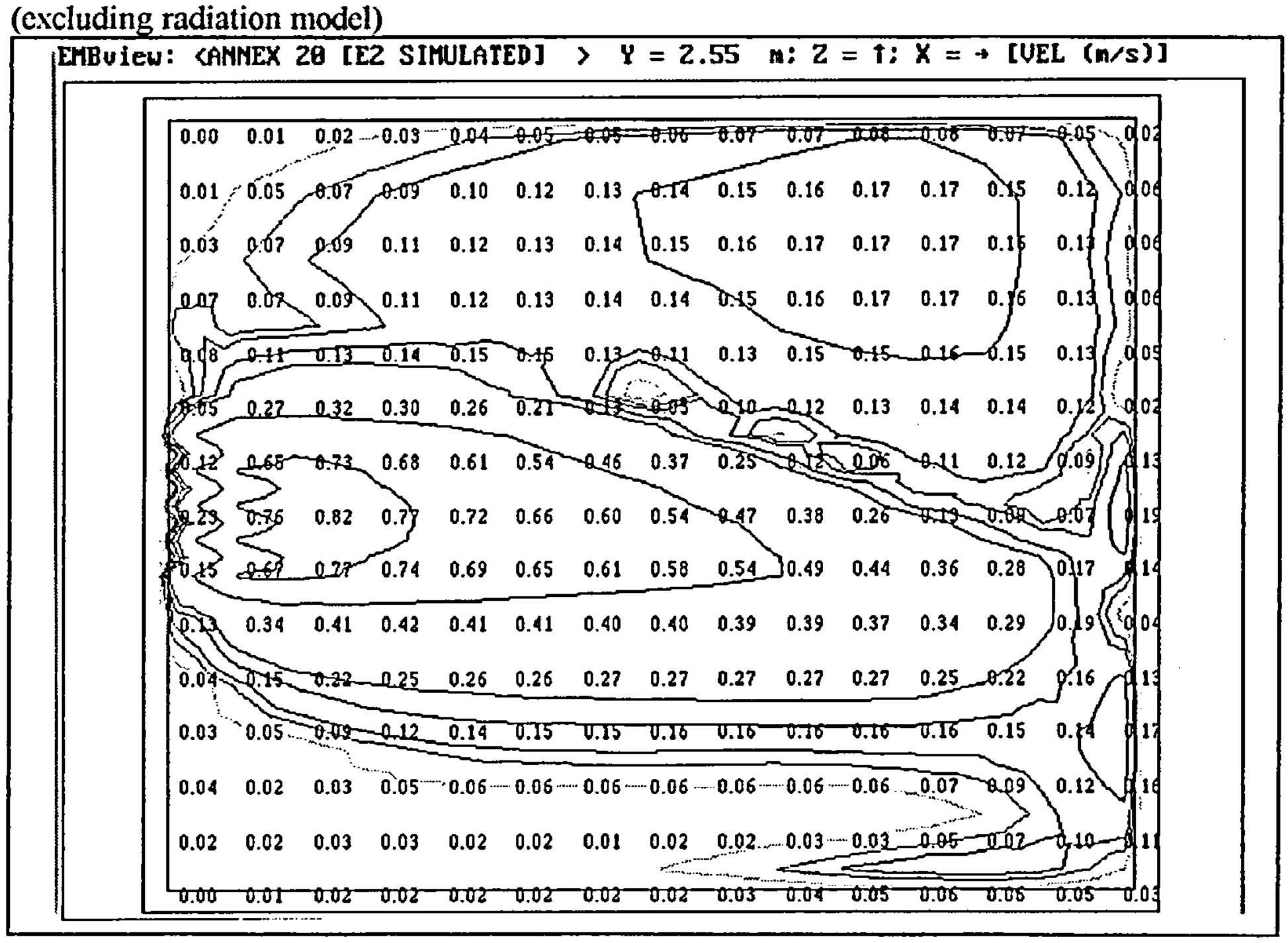


Figure 64 -test case E2: simulated velocity contours at 0.05 m below ceiling in x-z plane (excluding radiation model)

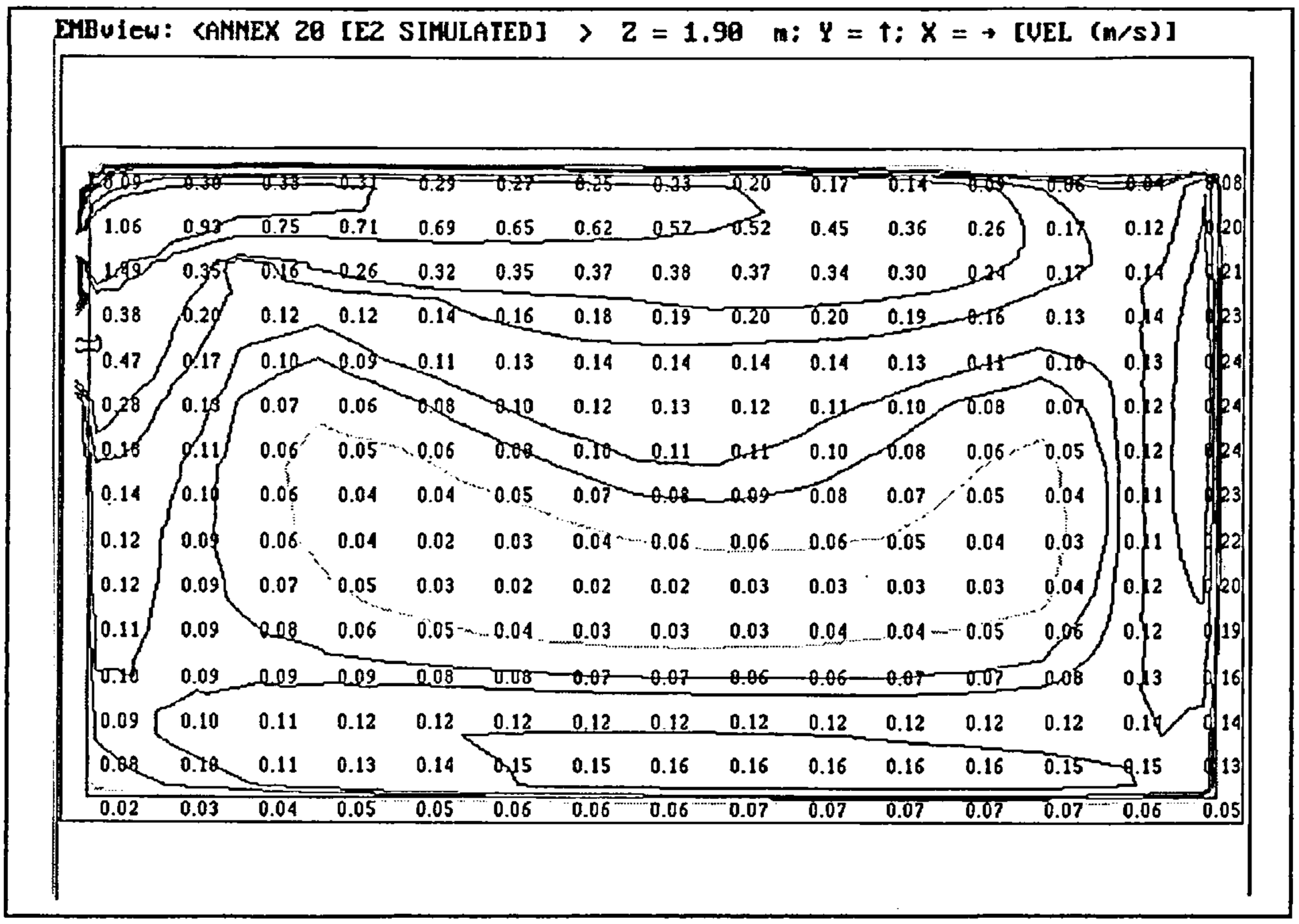


Figure 65 -test case E2: simulated velocity contours on centre-line of z-dimension in x-y plane (including radiation model)

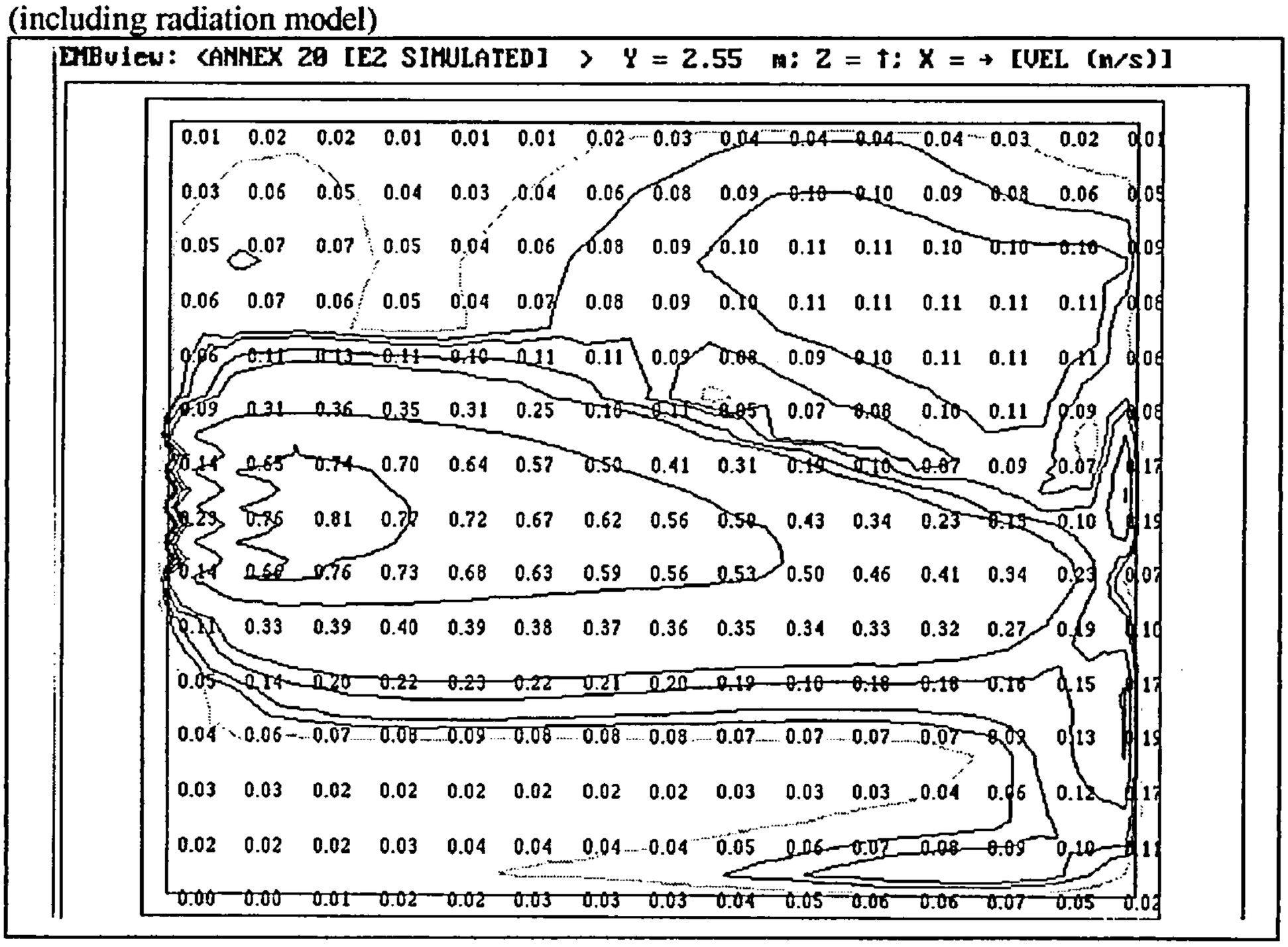


Figure 66 -test case E2: simulated velocity contours at 0.05 m below ceiling in x-z plane (including radiation model)

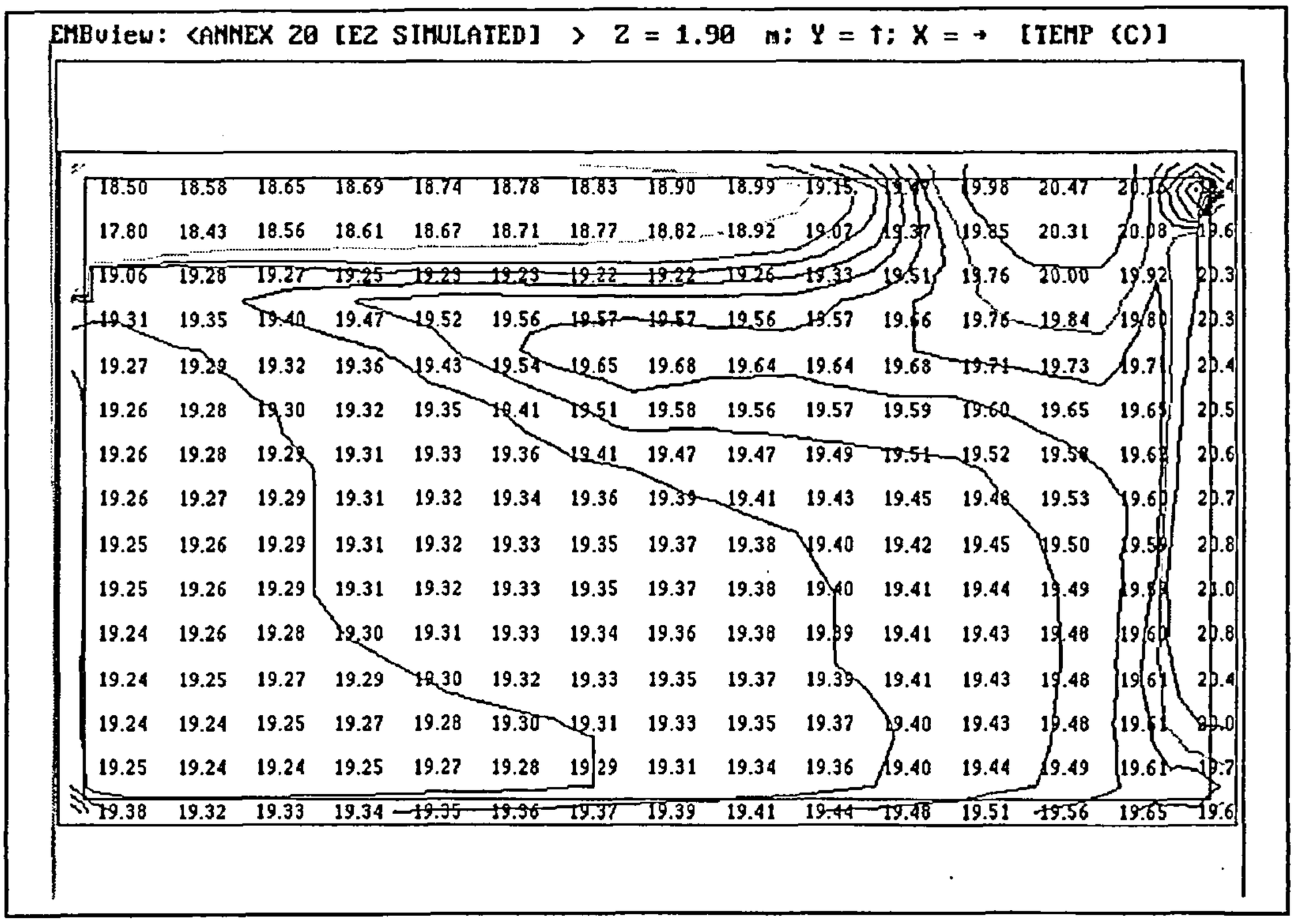


Figure 67 -test case E2: simulated temperature contours on centre-line of z-dimension in x-y plane (excluding radiation model)

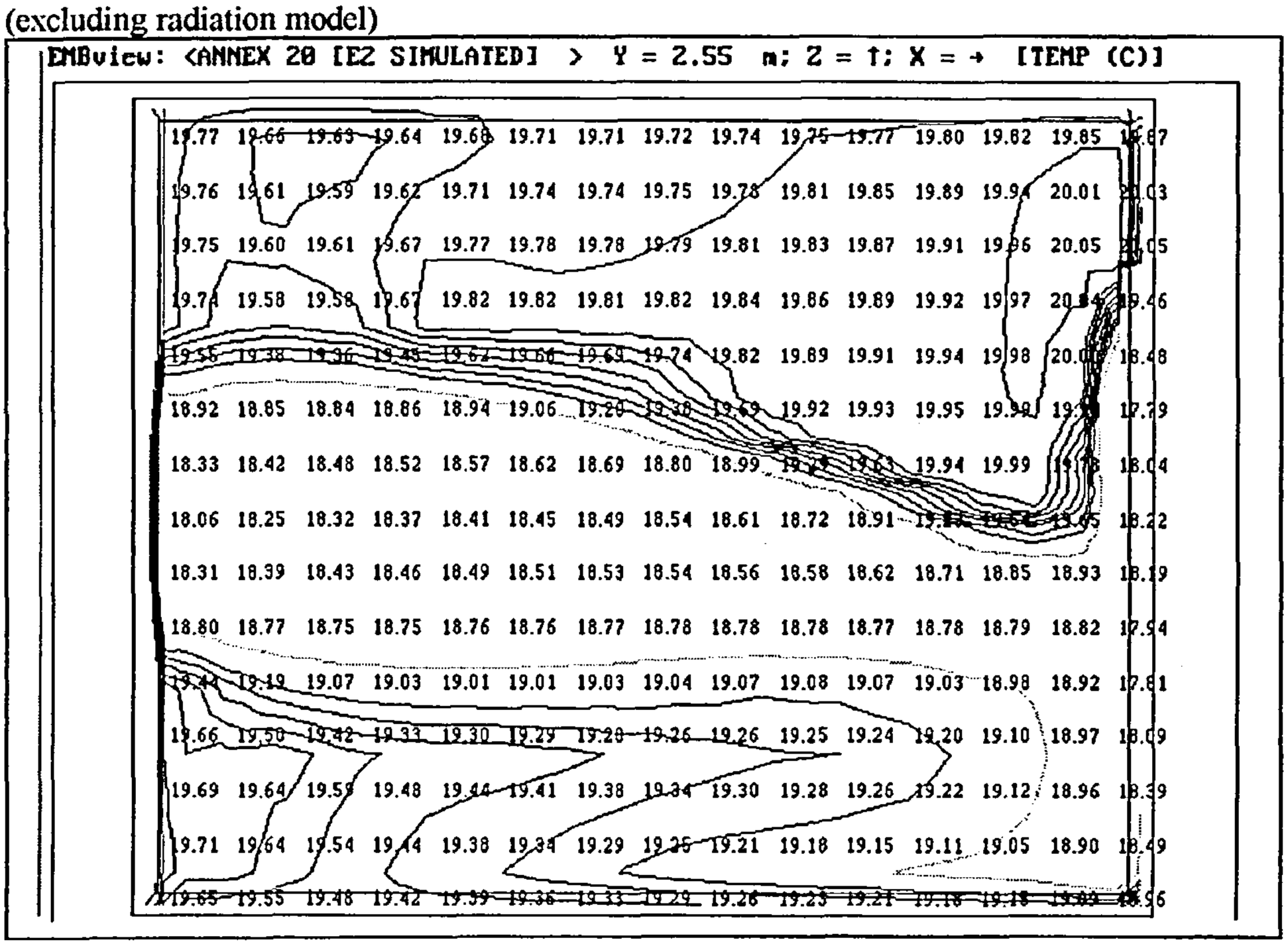


Figure 68 -test case E2: simulated temperature contours at 0.05 m below ceiling in x-z plane (excluding radiation model)

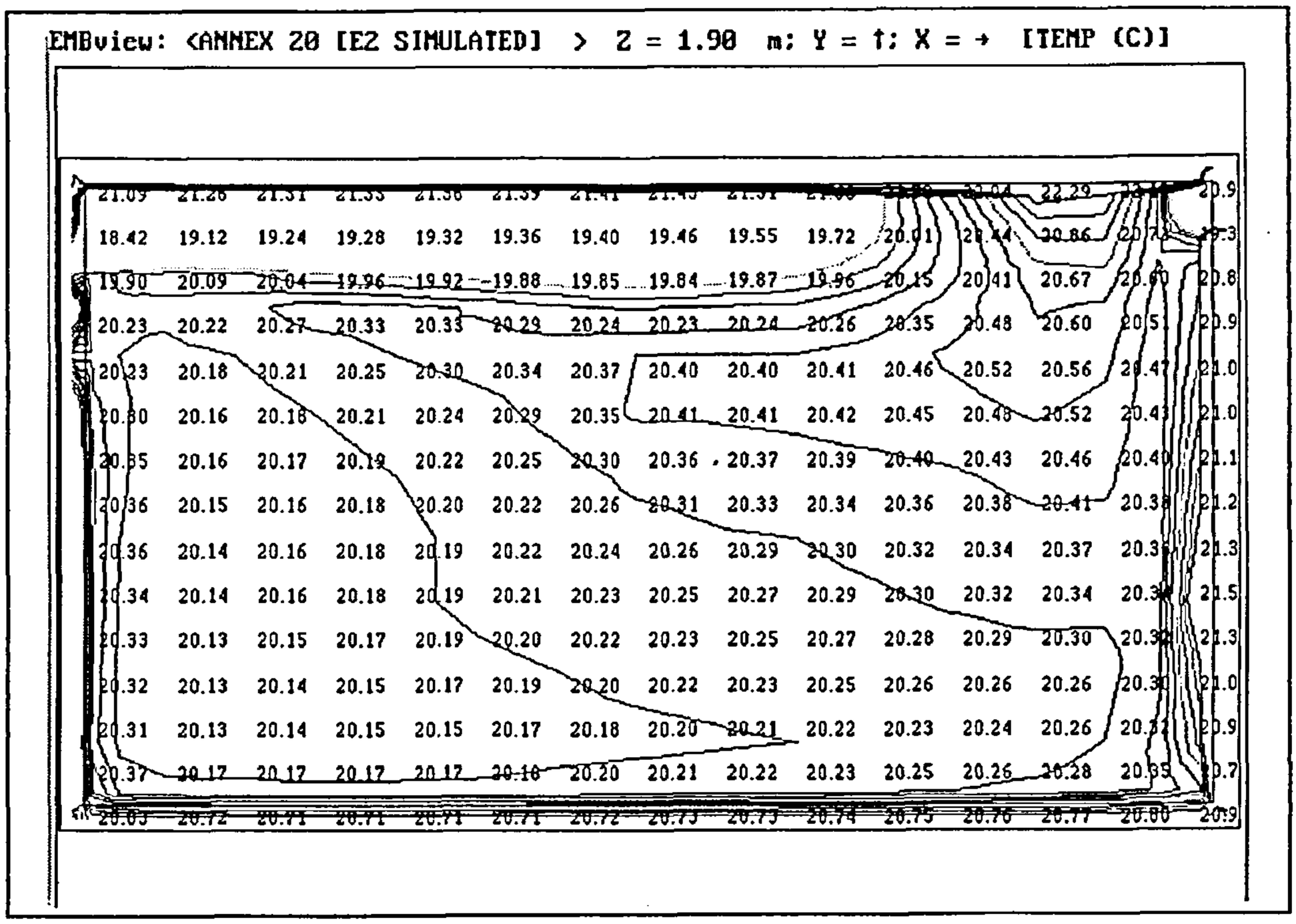


Figure 69 -test case E2: simulated temperature contours on centre-line of z-dimension in x-y plane (including radiation model)

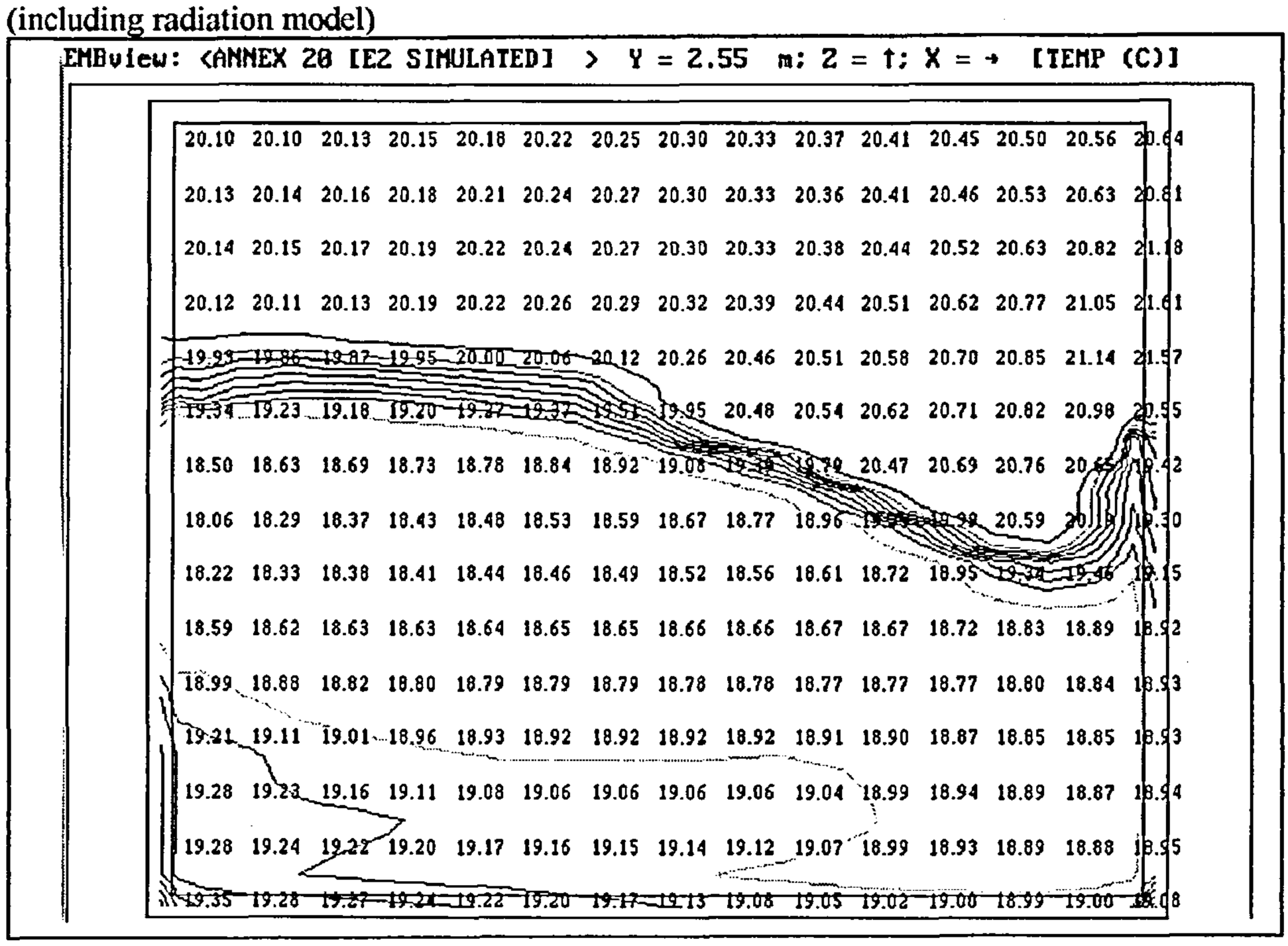


Figure 70 -test case E2: simulated temperature contours at 0.05 m below ceiling in x-z plane (including radiation model)

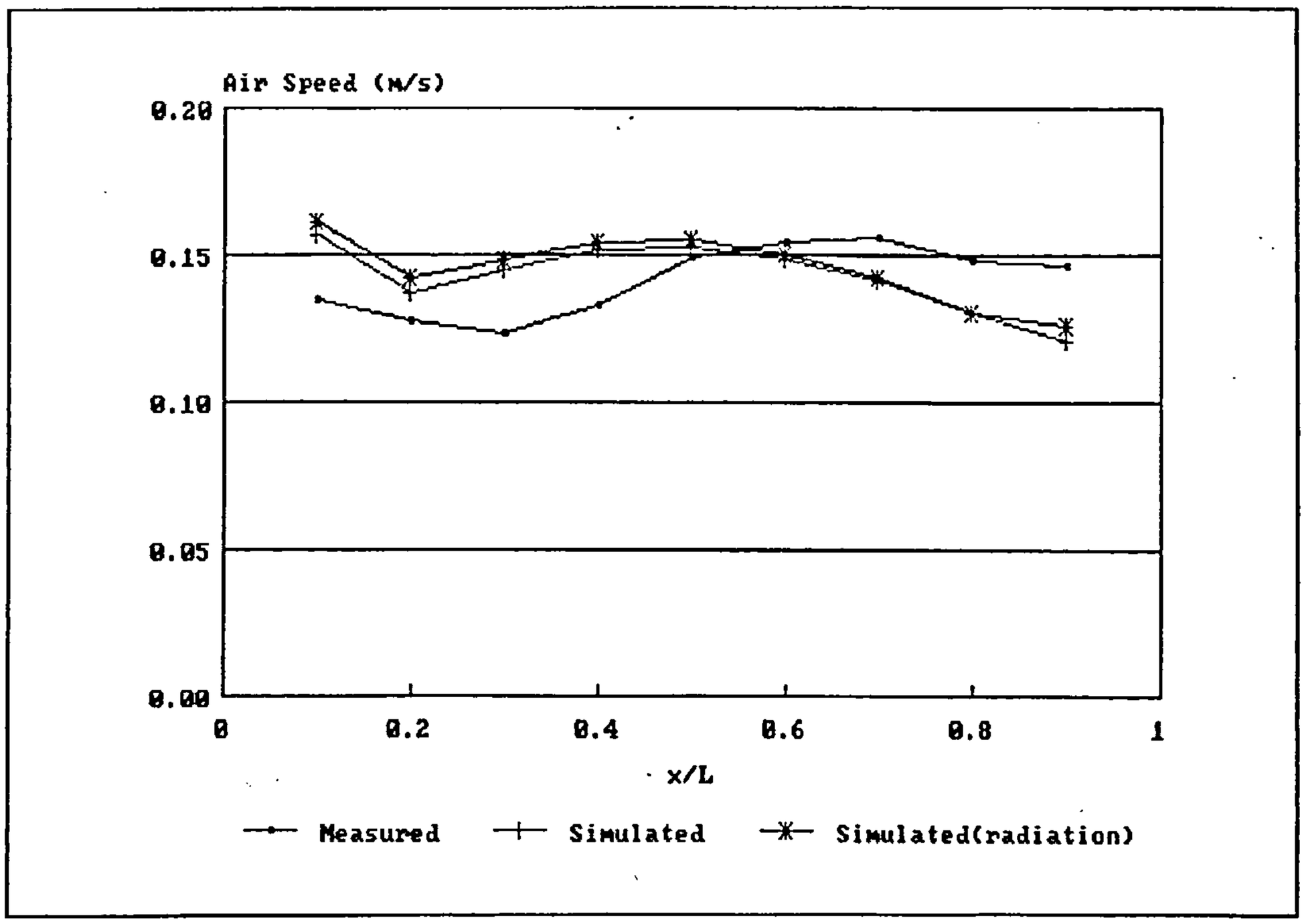


Figure 71 -test case E2: variation of mean air speed with relative distance on the x-axis

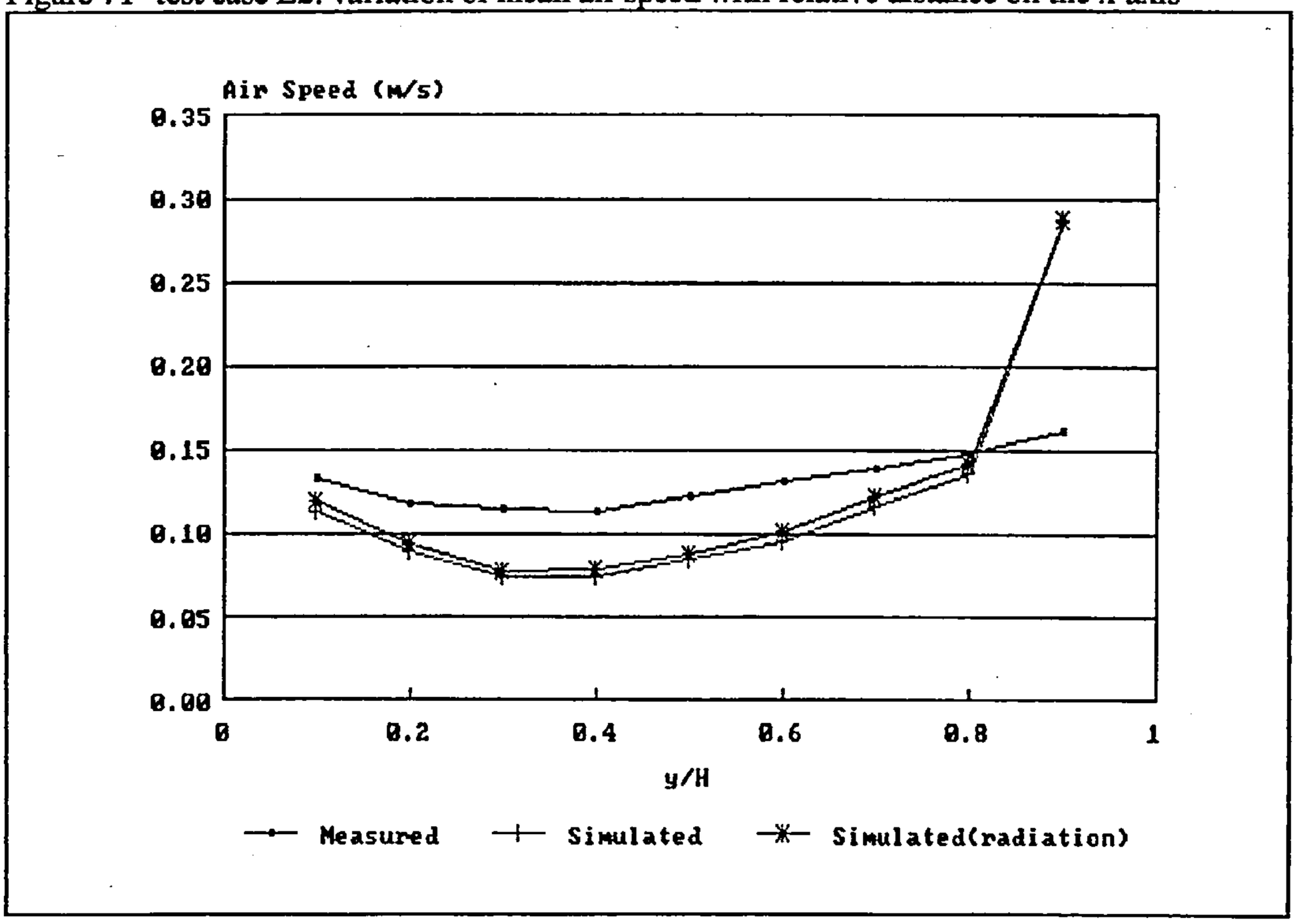


Figure 72 -test case E2: variation of mean air speed with relative distance on the y-axis

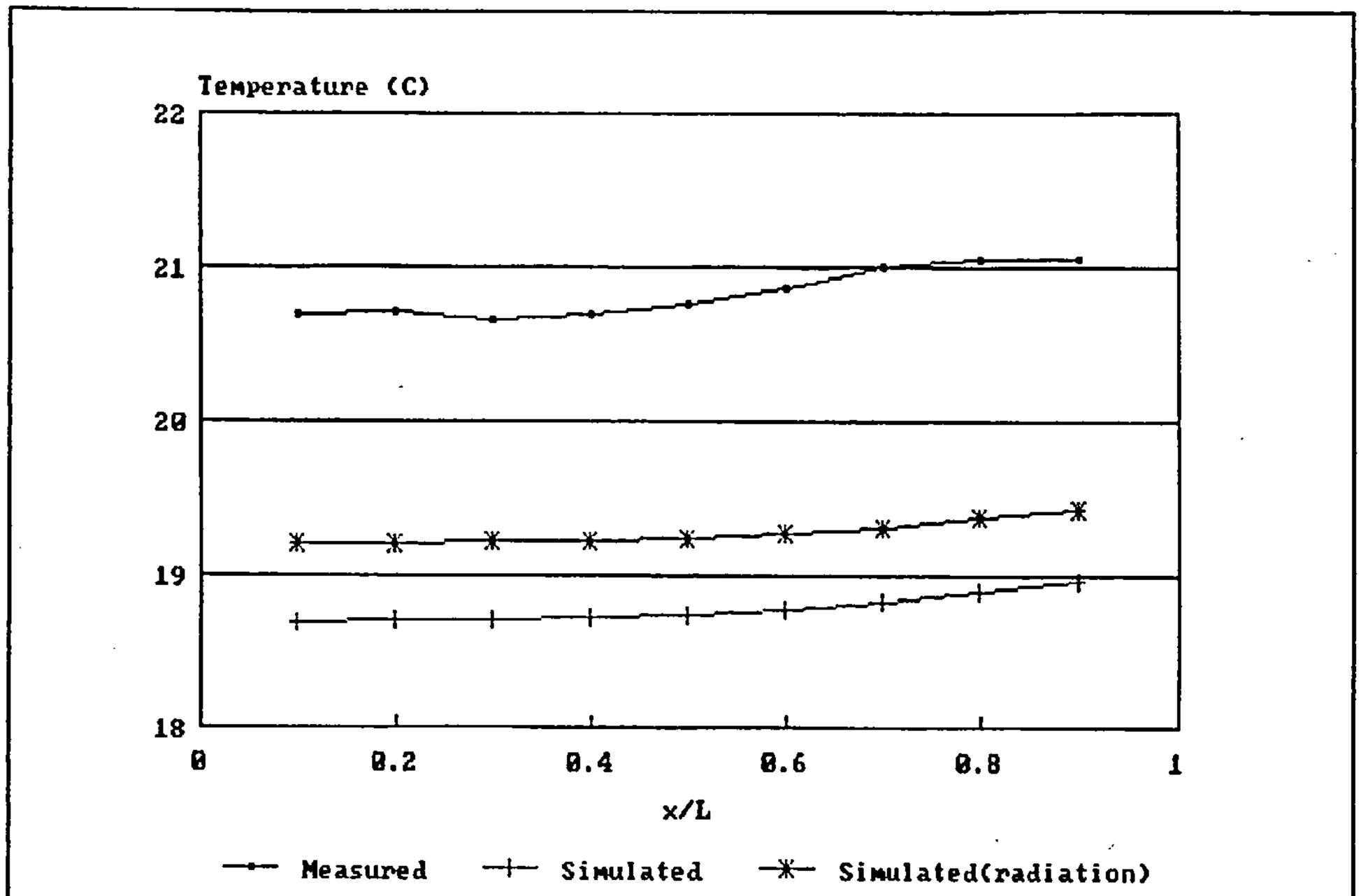


Figure 73 -test case E2: variation of temperature with relative distance on the x-axis

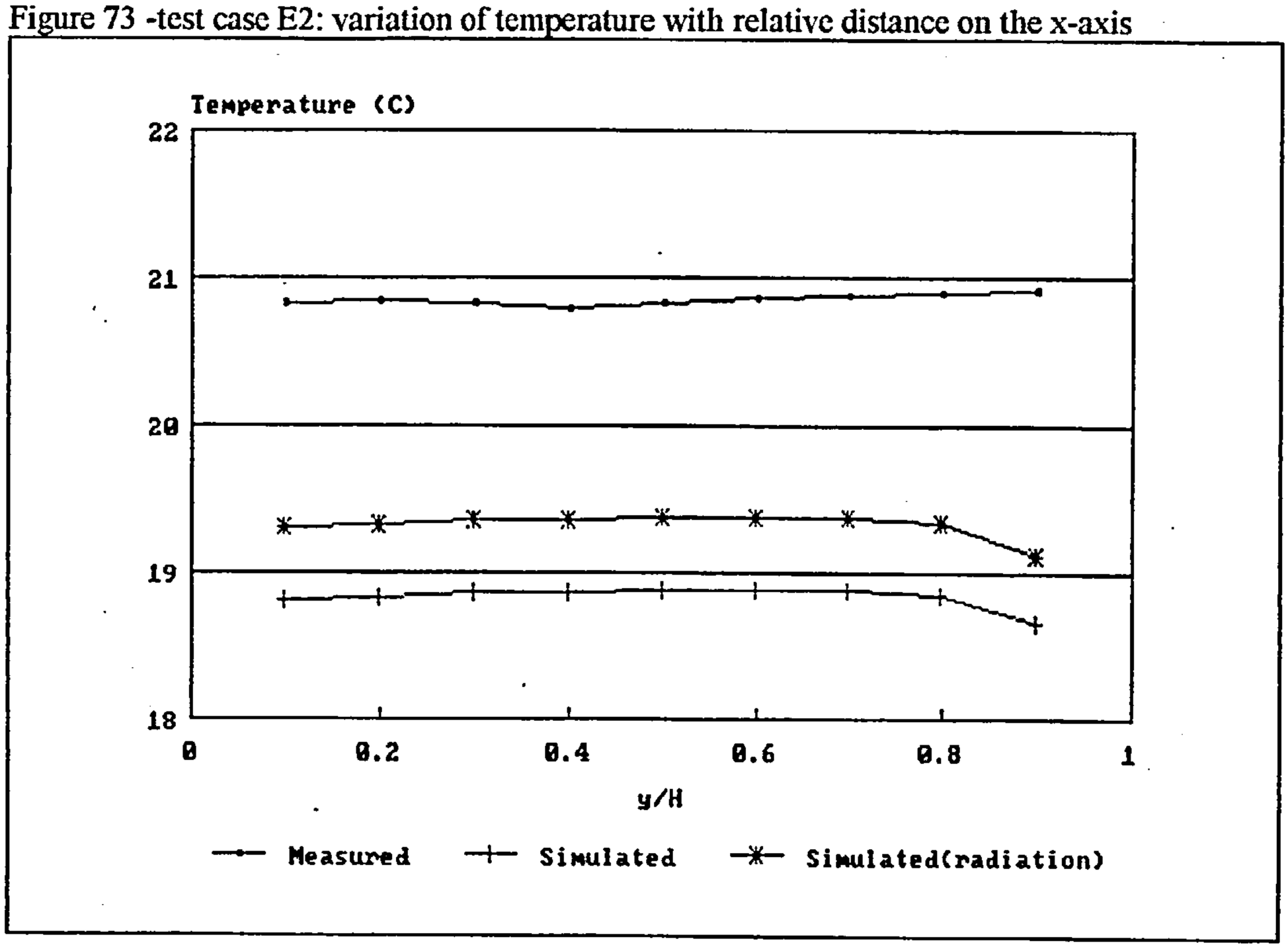


Figure 74 -test case E2: variation of temperature with relative distance on the y-axis

5.4 Test Case D

Test case D2 has been selected for the purpose of comparison.

Unfortunately only a limited set of measured results were made available for test case D. the results comprise measurements for only half of the test room. Accordingly measured results are only presented graphically for the z-axis centre-line. Graphical velocity contours, temperature contours and mean velocity plots are presented in Figures 75 and 76. Velocity and temperature profiles are presented in Figures 89-92.

Two sets of calculations have been conducted for test case D2 for the purpose of comparison, firstly excluding a radiation model and secondly including the radiation model described in Section 4.3

Surface heat transfer is calculated using the method detailed in Section 4.4.2. Heat transfer from the walls to the outside is assumed to be zero (i.e. adiabatic boundary conditions).

A finite volume grid of 19 x 12 x 15 cells was employed for the simulation, an irregular grid being defined to fit the radiator and the window. The geometric polyhedron representation of the test room (see Section 4.1), developed for test case B2 was modified to include additional surfaces in order to represent the window and radiator in the radiation model.

Convergence was achieved after approximately 300 iterations for both calculation including and excluding the radiation model. Calculations took approximately 6 minutes on a 350 MHz Pentium II type PC. Convergence was deemed to have been achieved when all dependent variable residual were less than 10^{-6} in magnitude.

The calculated results are presented in the form of graphical velocity vectors, temperatures, velocity contours and mean velocity plots in Figures 58-69. Velocity and temperature profiles are presented in Figures 89-92.

There is seen to be reasonable agreement between the mean speed contours in both the X-Y and X-Z planes in terms of the general flow pattern, although the magnitude of the predicted air speed is somewhat lower than the measured air speeds throughout the flow domain for both sets of calculations. The discrepancy in air speed magnitudes may be due in part to measurement inaccuracies at these very low air speeds (< 0.1 ms^{-1}).

Referring to the temperature contours in Figures 75-76, 83-86 and the temperature profiles in

Figures 89-90, reasonable agreement can be seen to exist between the predicted results using the radiation model and the measured results. As expected, in the absence of a radiation model, the predicted results over-estimate the temperatures throughout the flow domain. the predicted results using the radiation model slightly under-estimate the flow temperatures, again indicating the possible need for refinement of the method for calculating surface heat transfer.

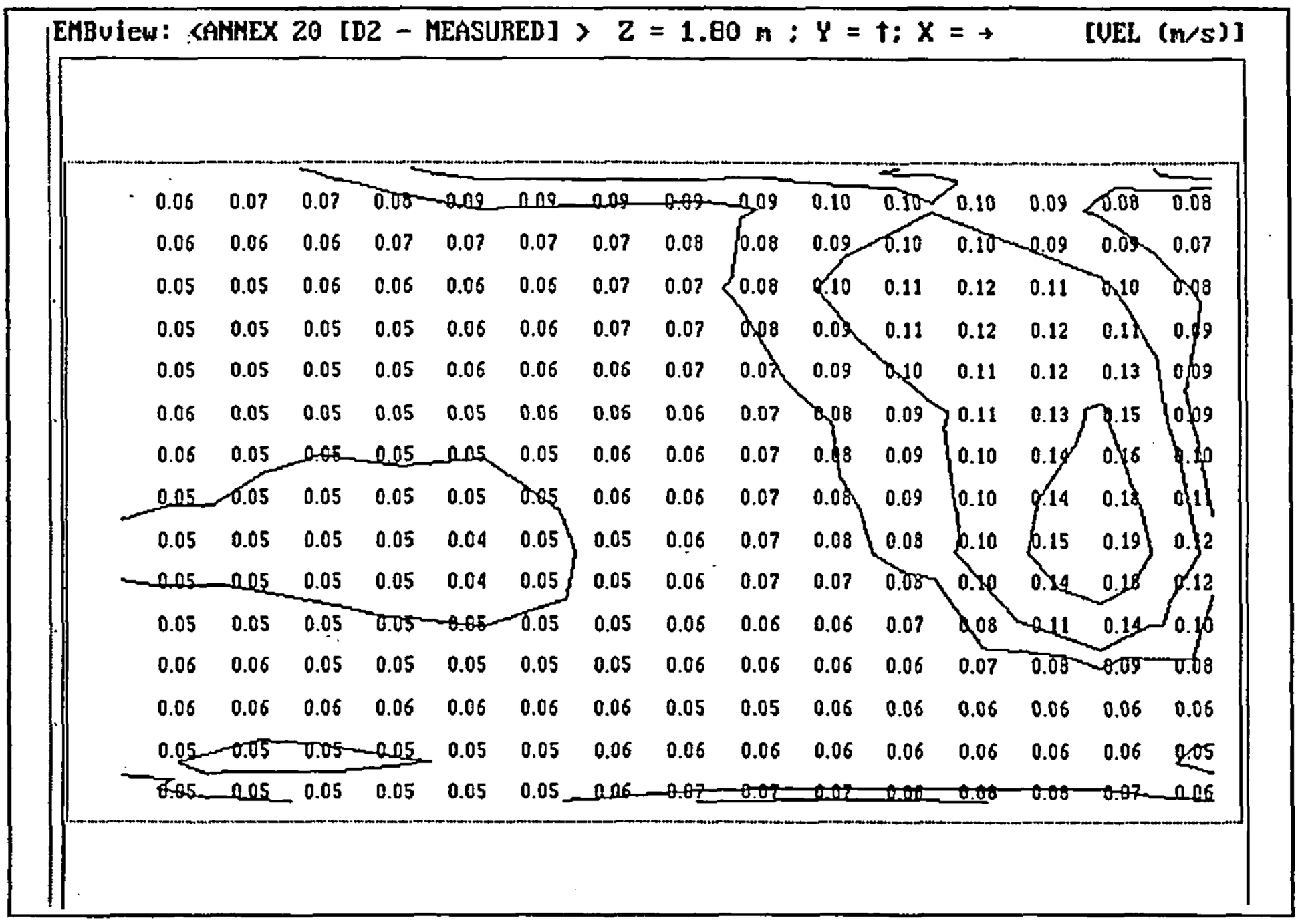


Figure 75 -test case D2: measured velocity contours on centre-line of z-dimension in x-y plane

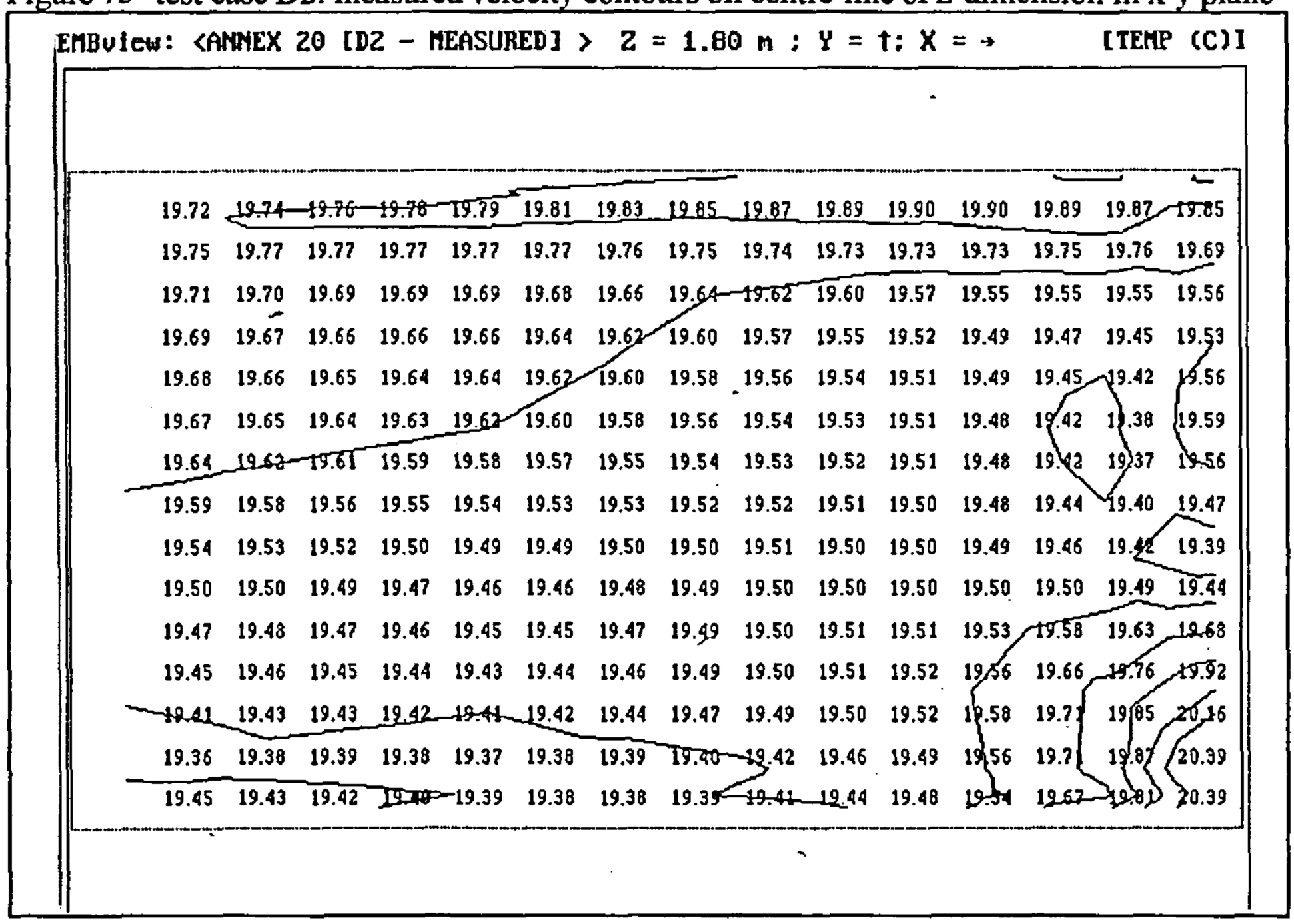


Figure 76-test case D2: measured temperature contours on centre-line of z-dimension in x-y plane

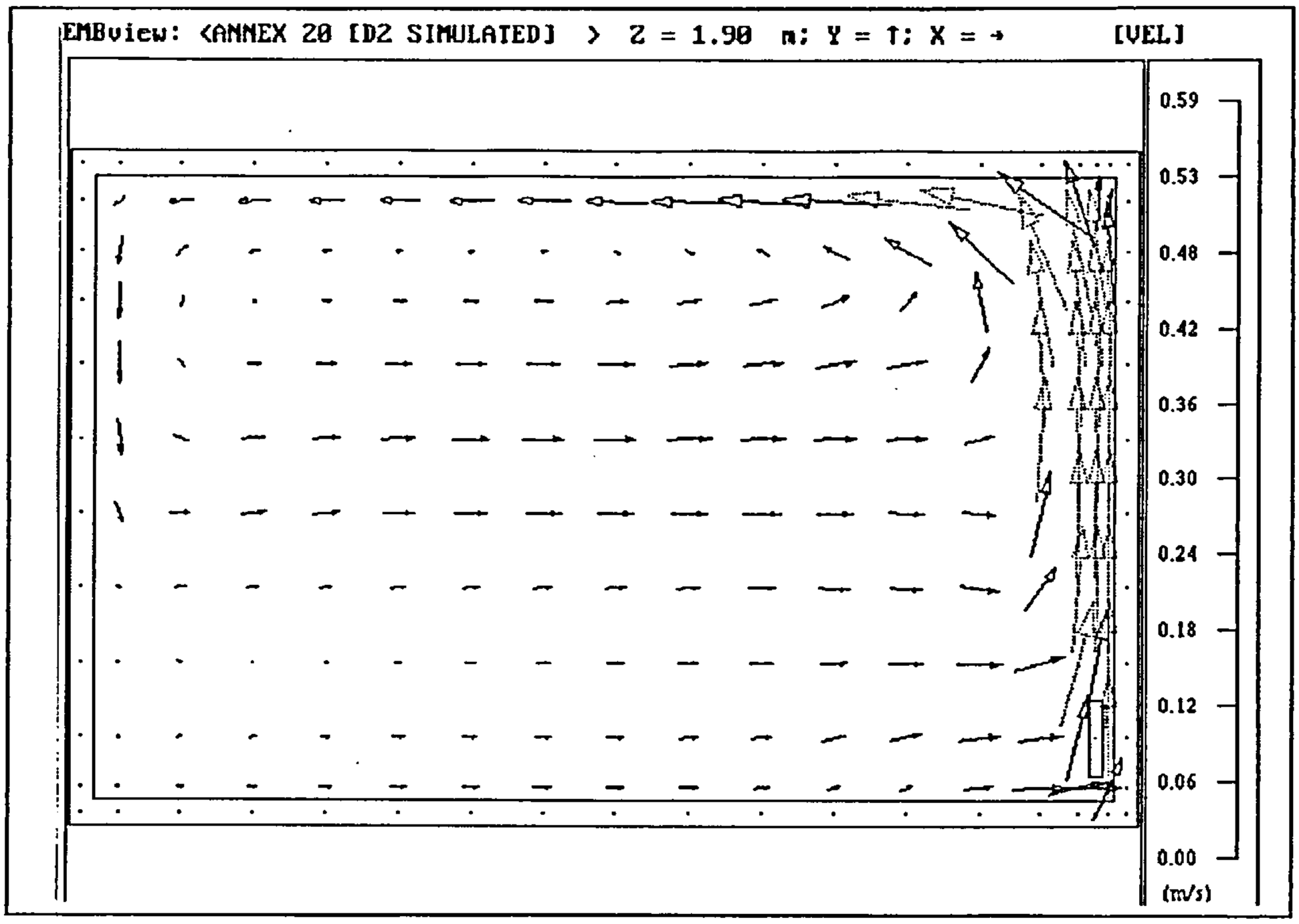


Figure 77 -test case D2: simulated velocity vectors on centre-line of z-dimension in x-y plane (excluding radiation model)

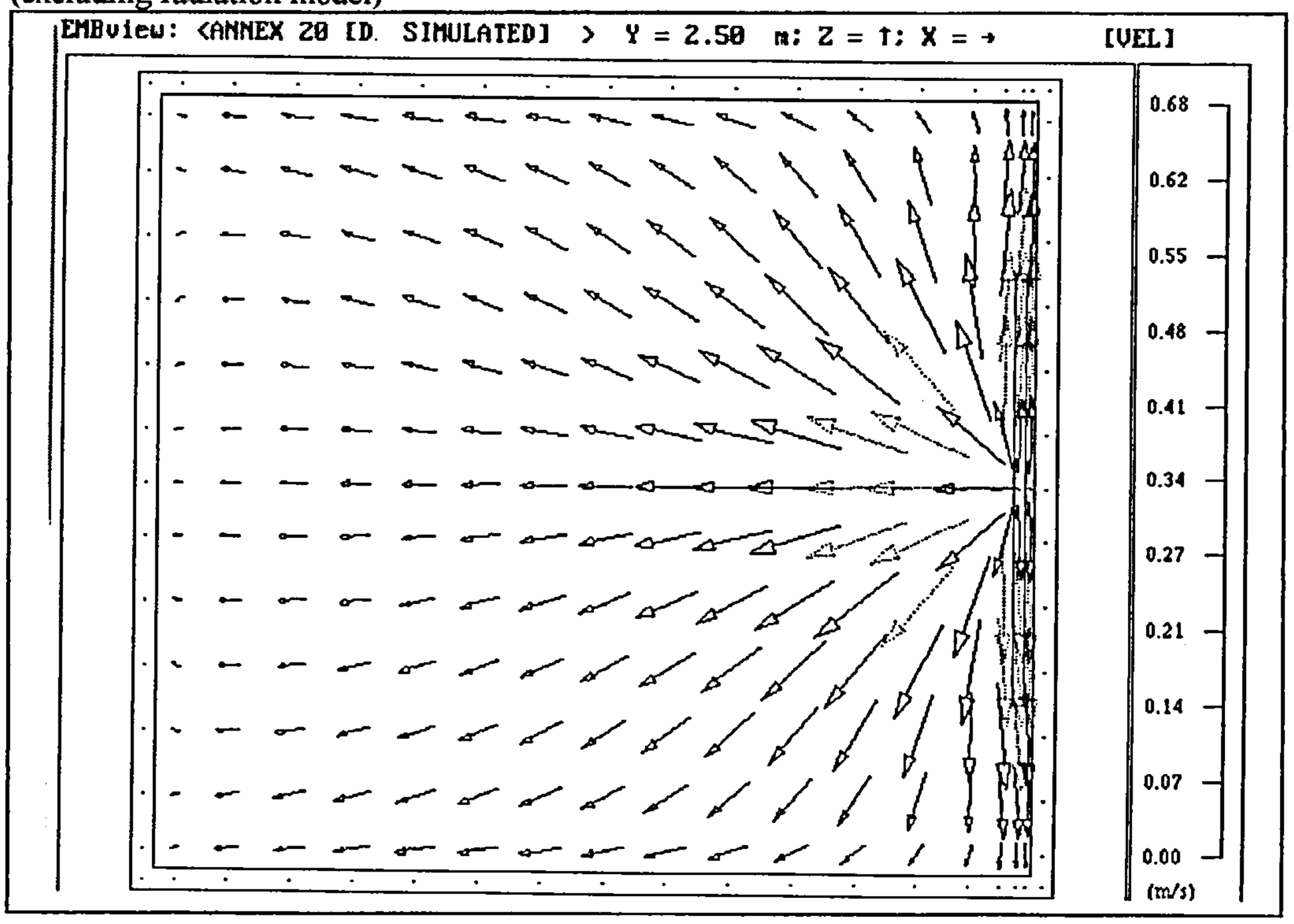


Figure 78 -test case D2: simulated velocity vectors at 0.05 m below ceiling in x-z plane (excluding radiation model)

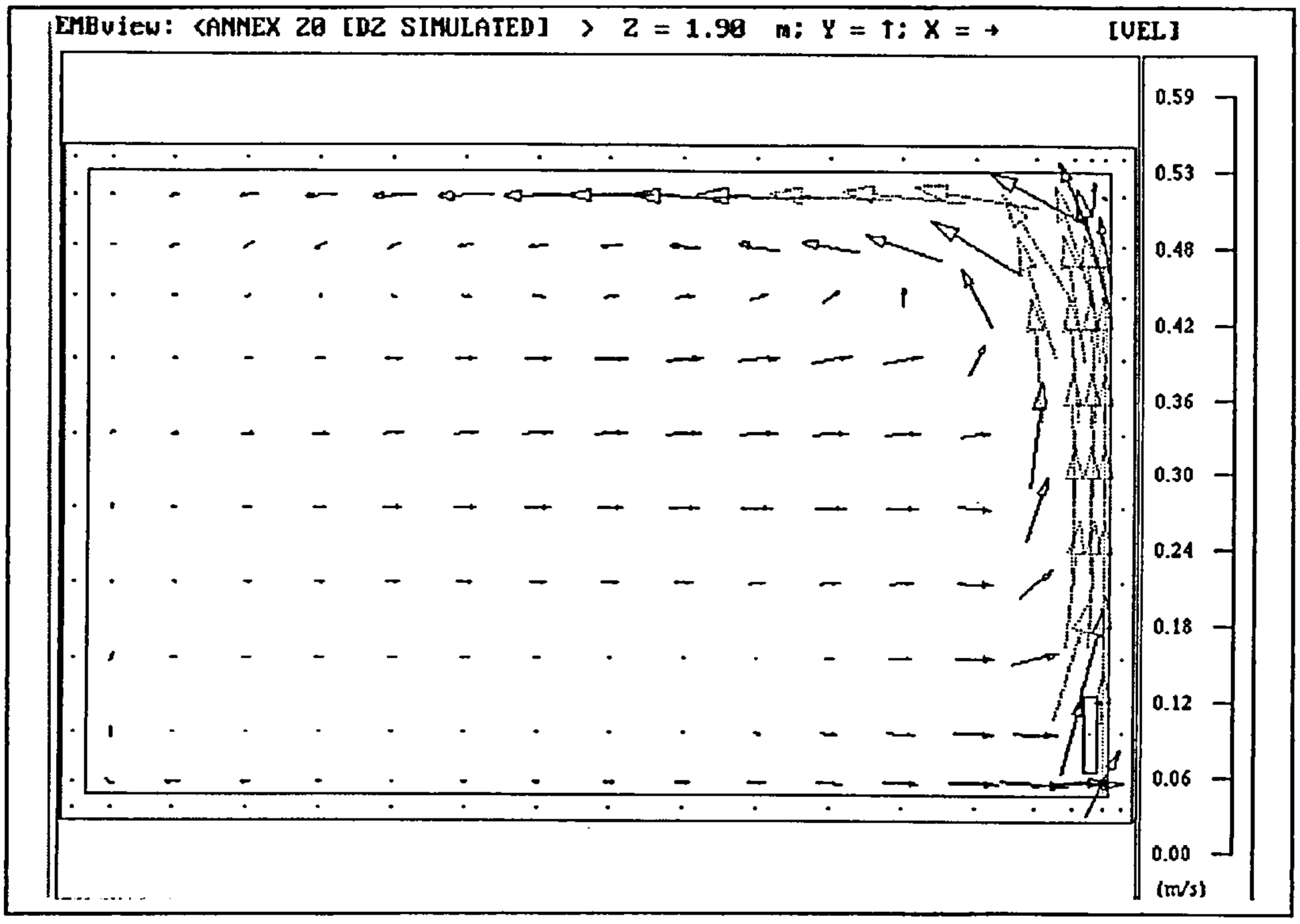


Figure 79 -test case D2: simulated velocity vectors on centre-line of z-dimension in x-y plane (including radiation model)

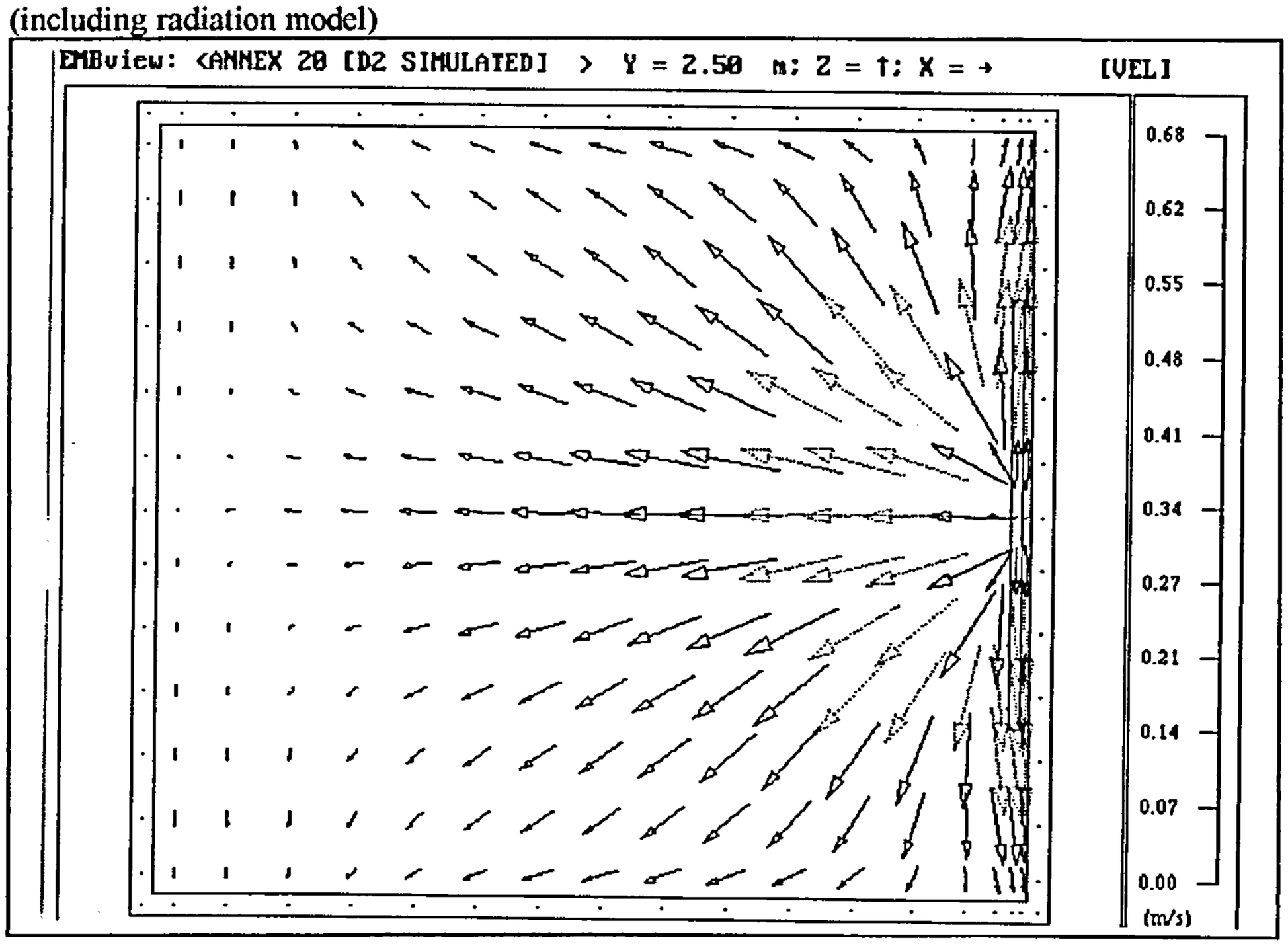


Figure 80 -test case D2: simulated velocity vectors at 0.05 m below ceiling in x-z plane (including radiation model)

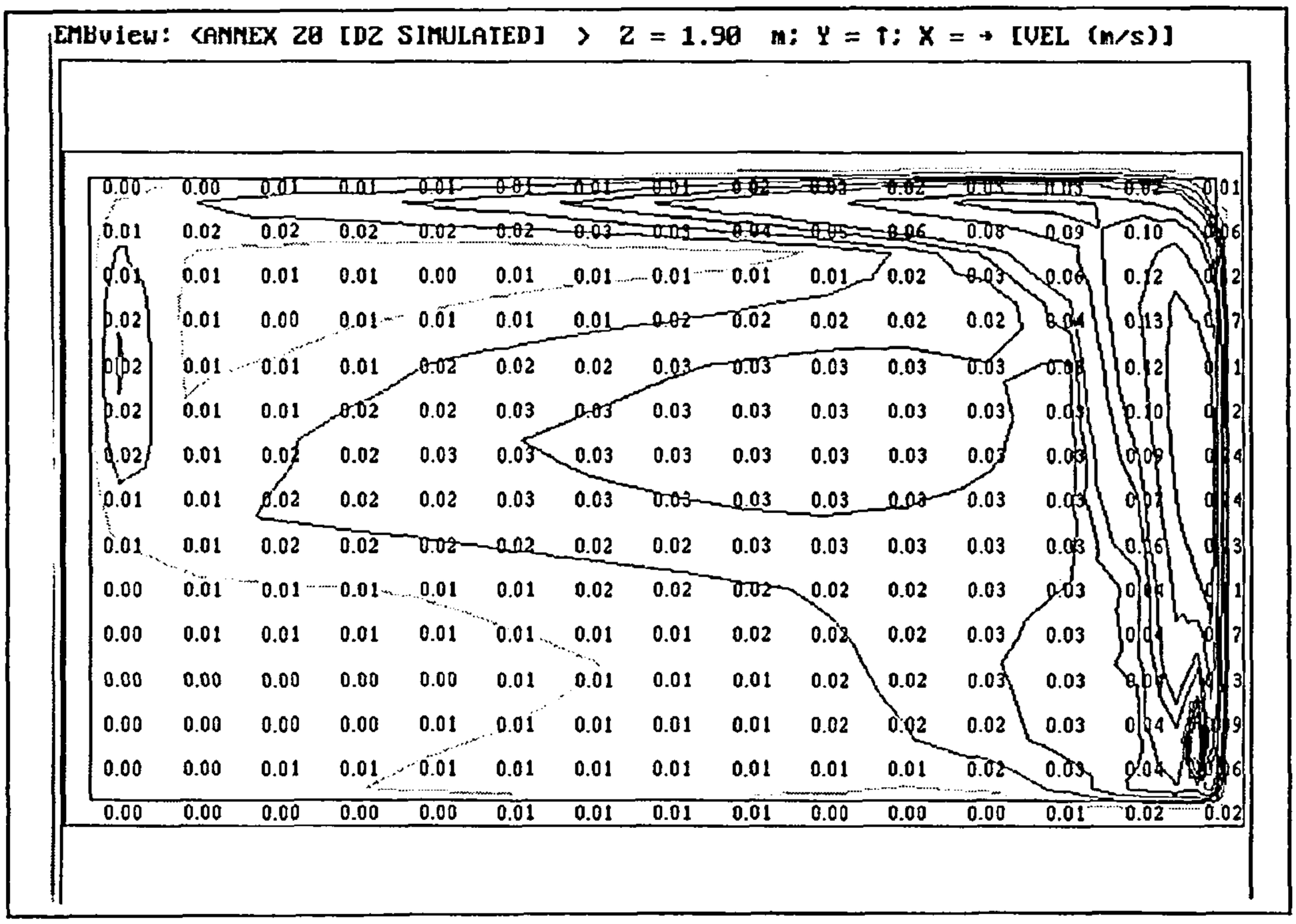


Figure 81 -test case D2: simulated velocity contours on centre-line of z-dimension in x-y plane (excluding radiation model)

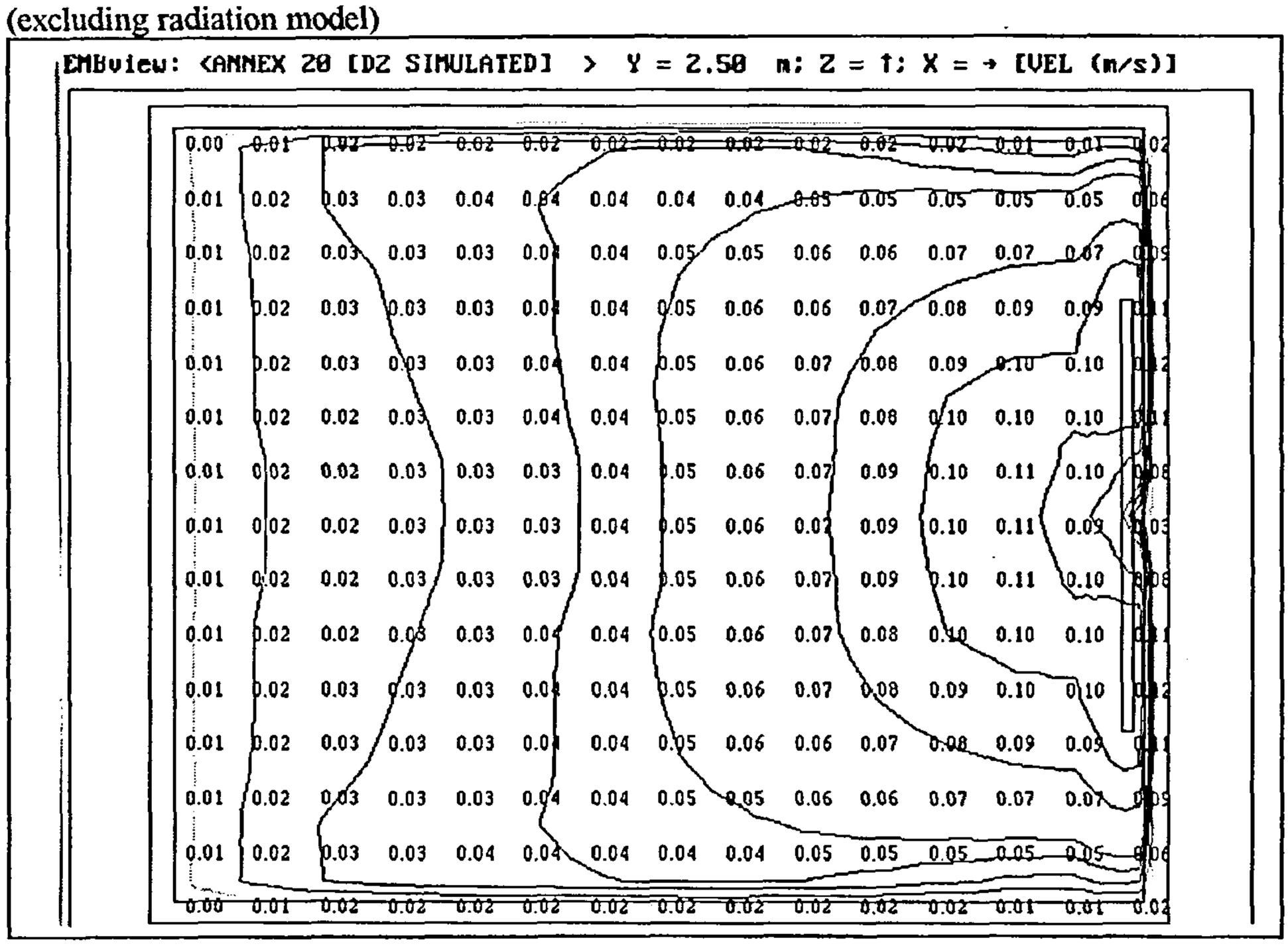


Figure 82 -test case D2: simulated velocity contours at 0.05 m below ceiling in x-z plane (excluding radiation model)

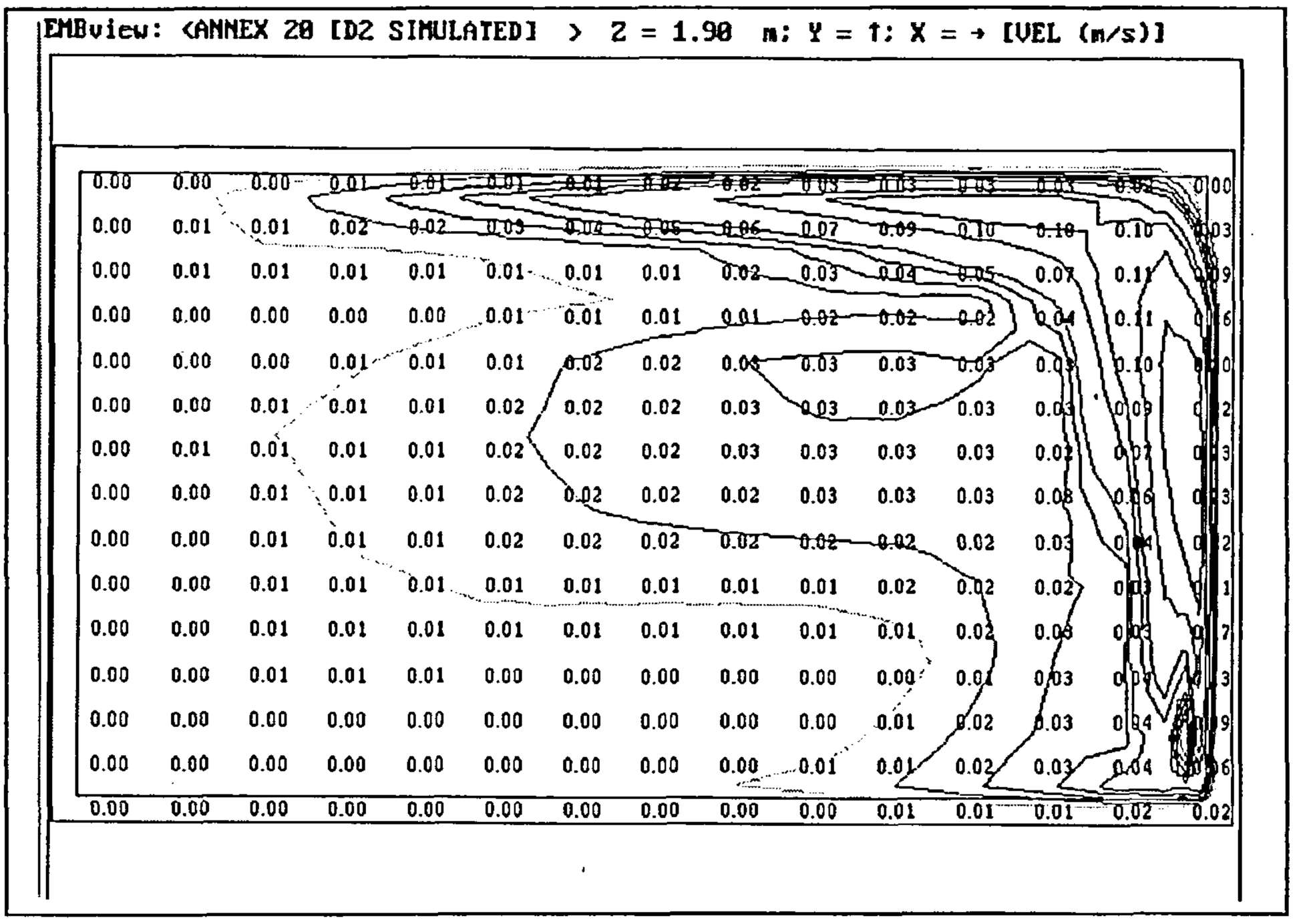


Figure 83 -test case D2: simulated velocity contours on centre-line of z-dimension in x-y plane (including radiation model)

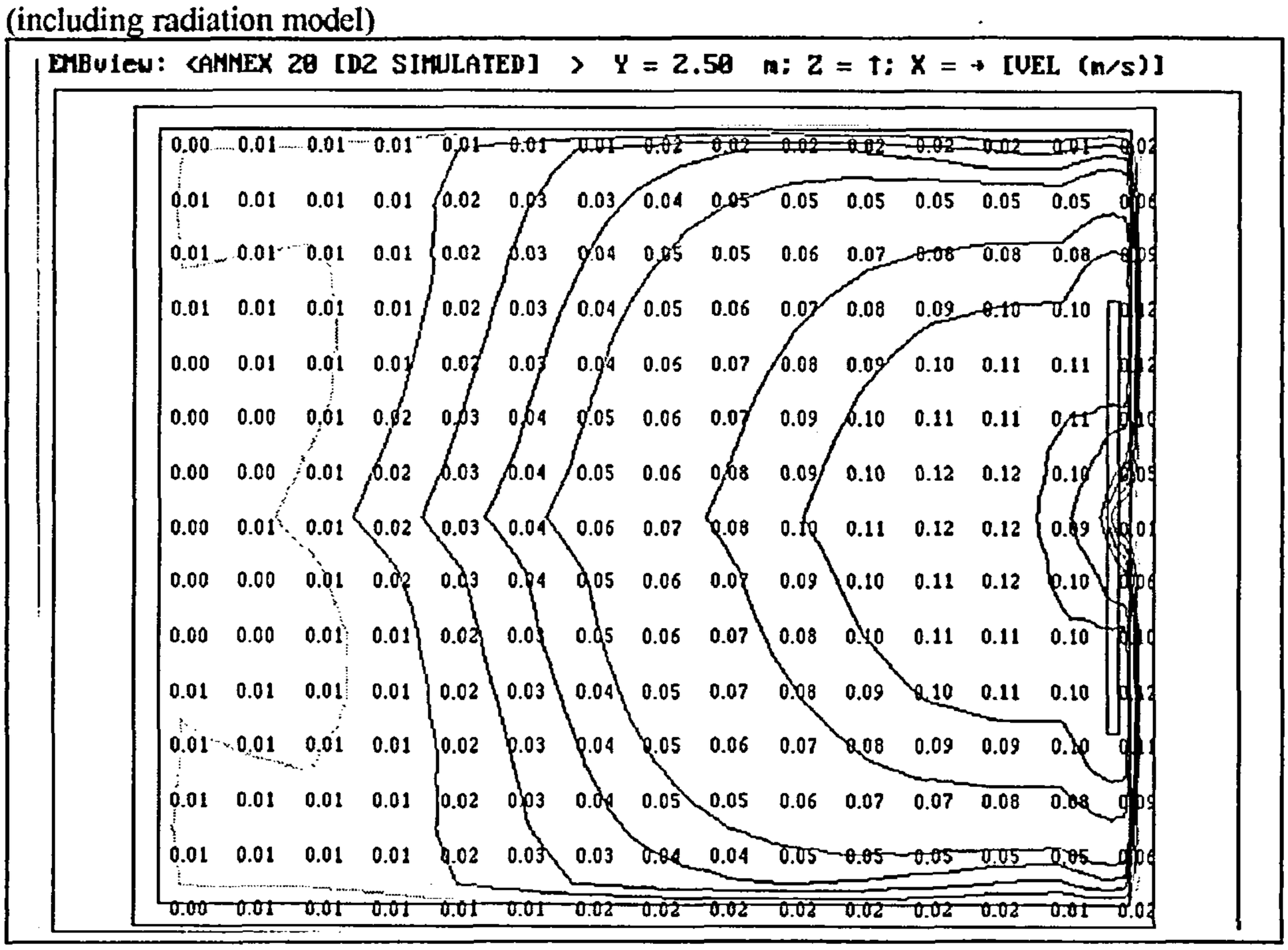


Figure 84 -test case D2: simulated velocity contours at 0.05 m below ceiling in x-z plane (including radiation model)

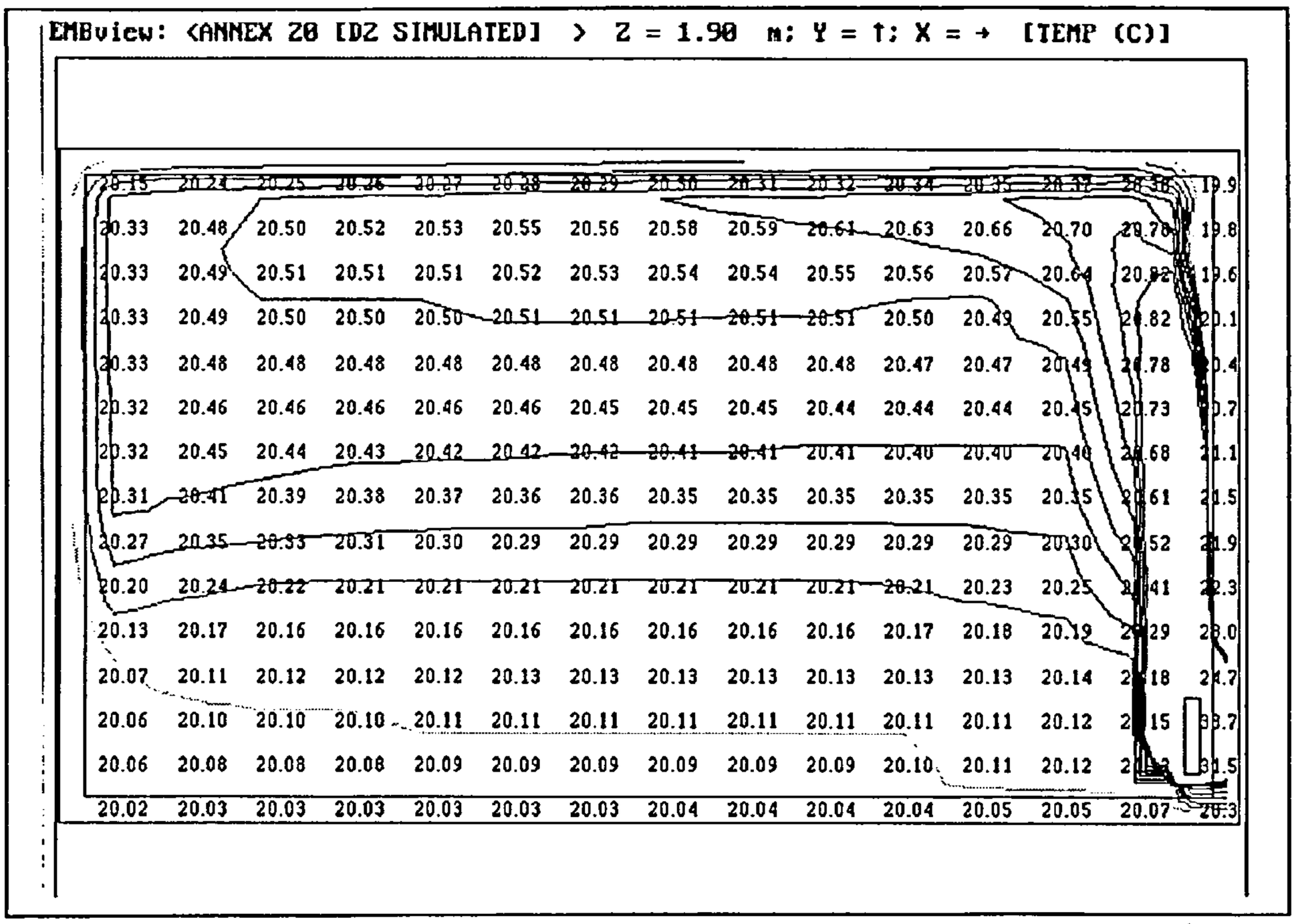


Figure 85 -test case D2: simulated temperature contours on centre-line of z-dimension in x-y plane (excluding radiation model)

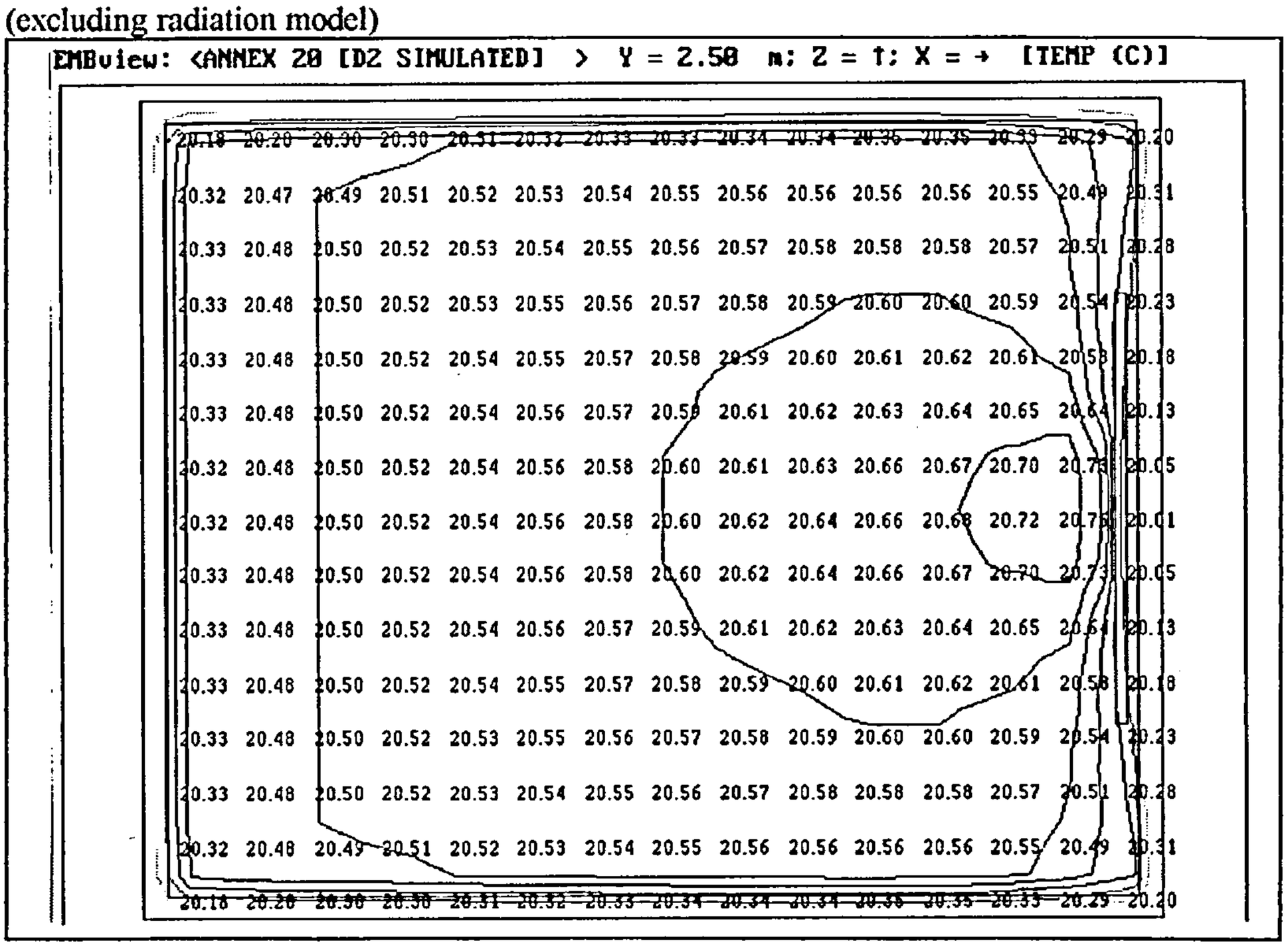


Figure 86 -test case D2: simulated temperature contours at 0.05 m below ceiling in x-z plane (excluding radiation model)

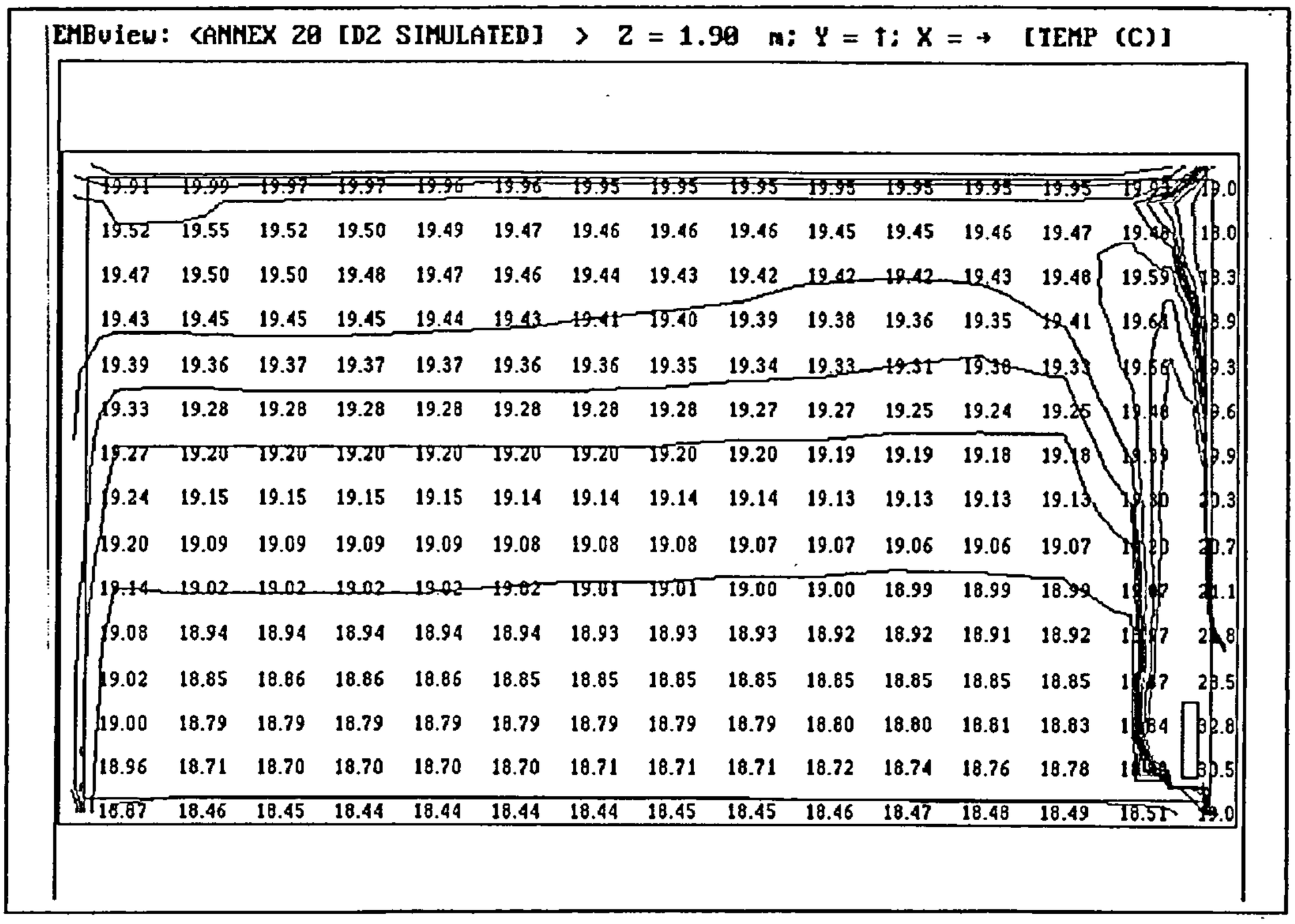


Figure 87 -test case D2: simulated temperature contours on centre-line of z-dimension in x-y plane (including radiation model)

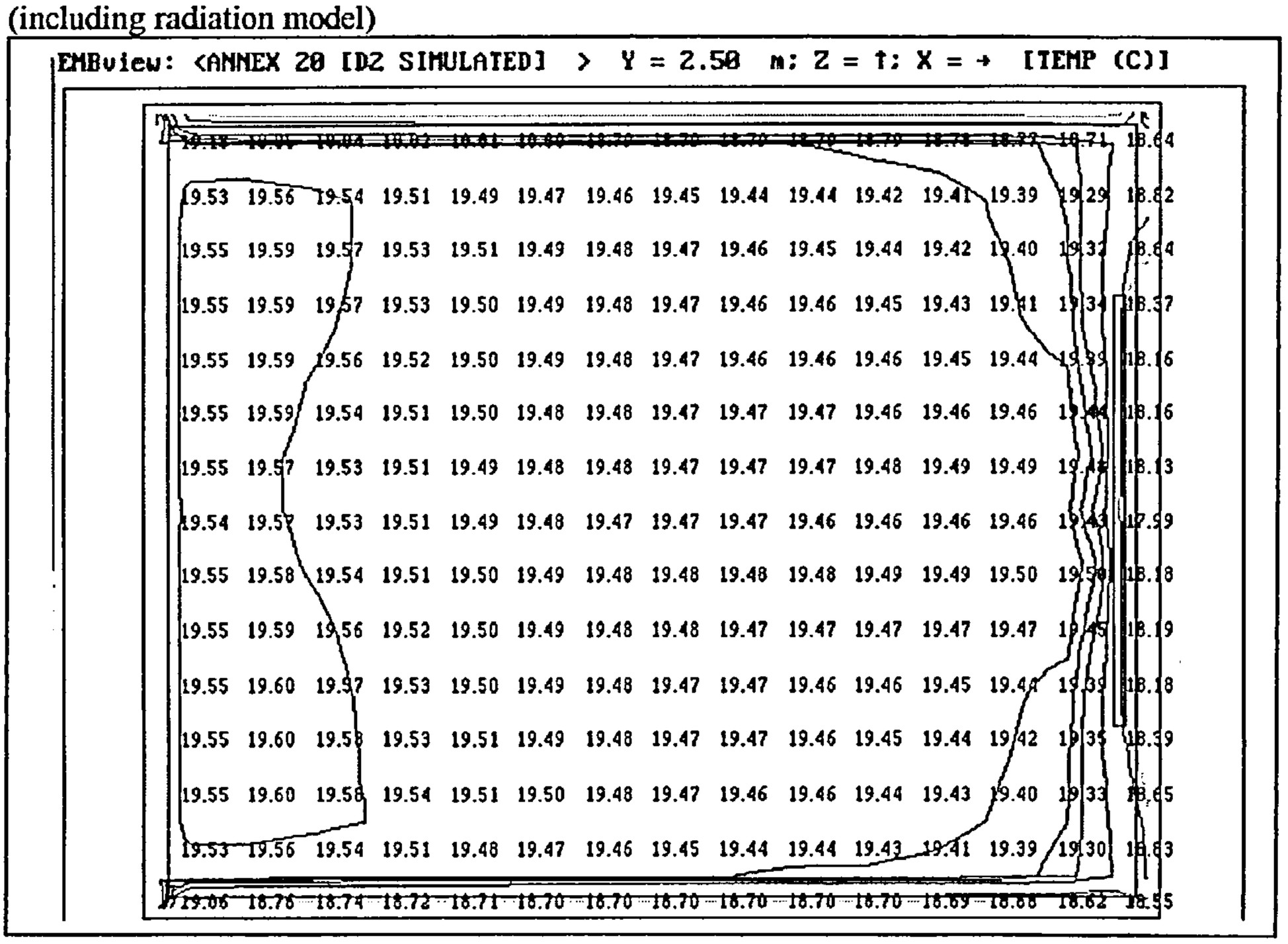


Figure 88 -test case D2: simulated temperature contours at 0.05 m below ceiling in x-z plane (including radiation model)

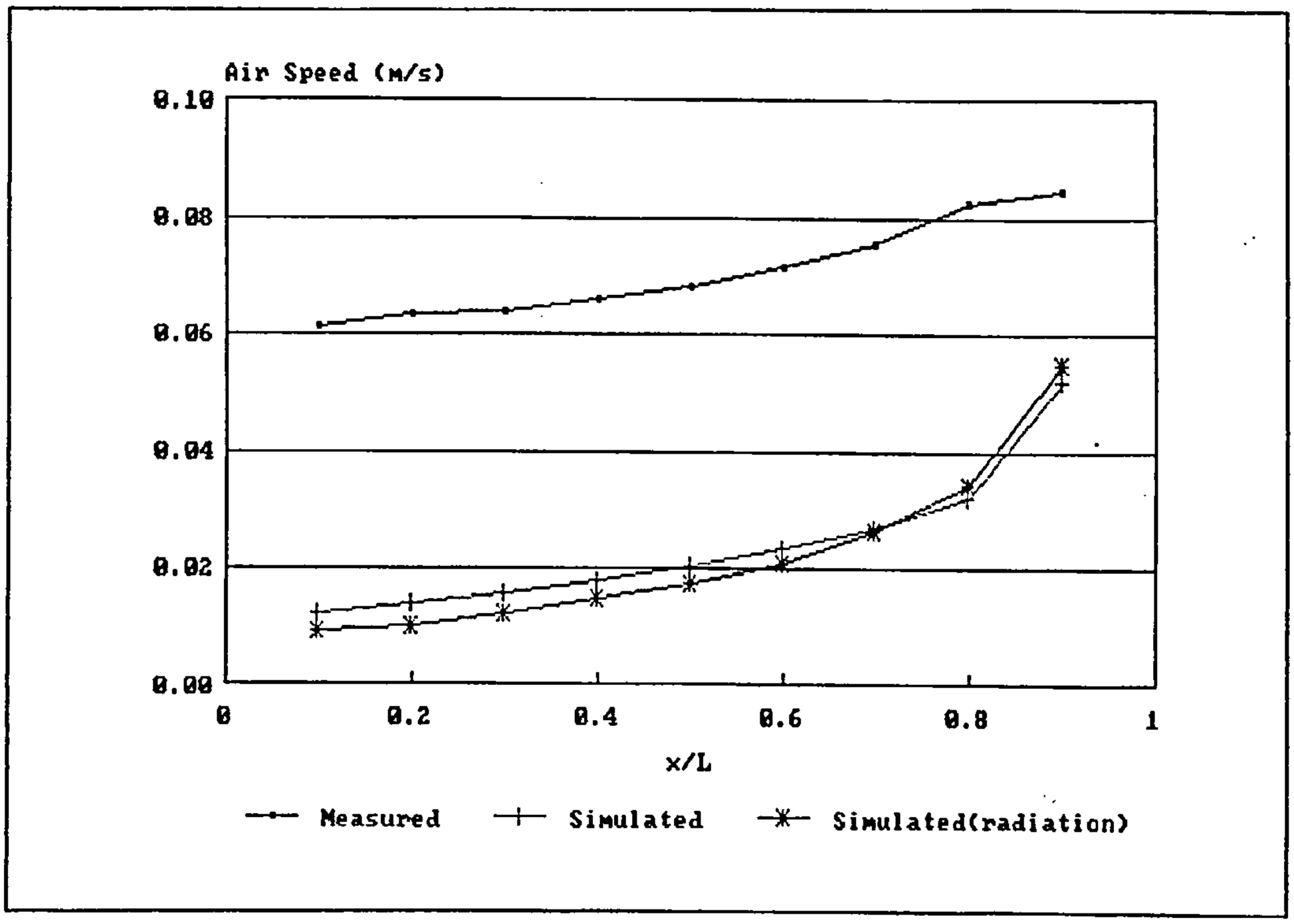


Figure 89 -test case D2: variation of mean air speed with relative distance on the x-axis

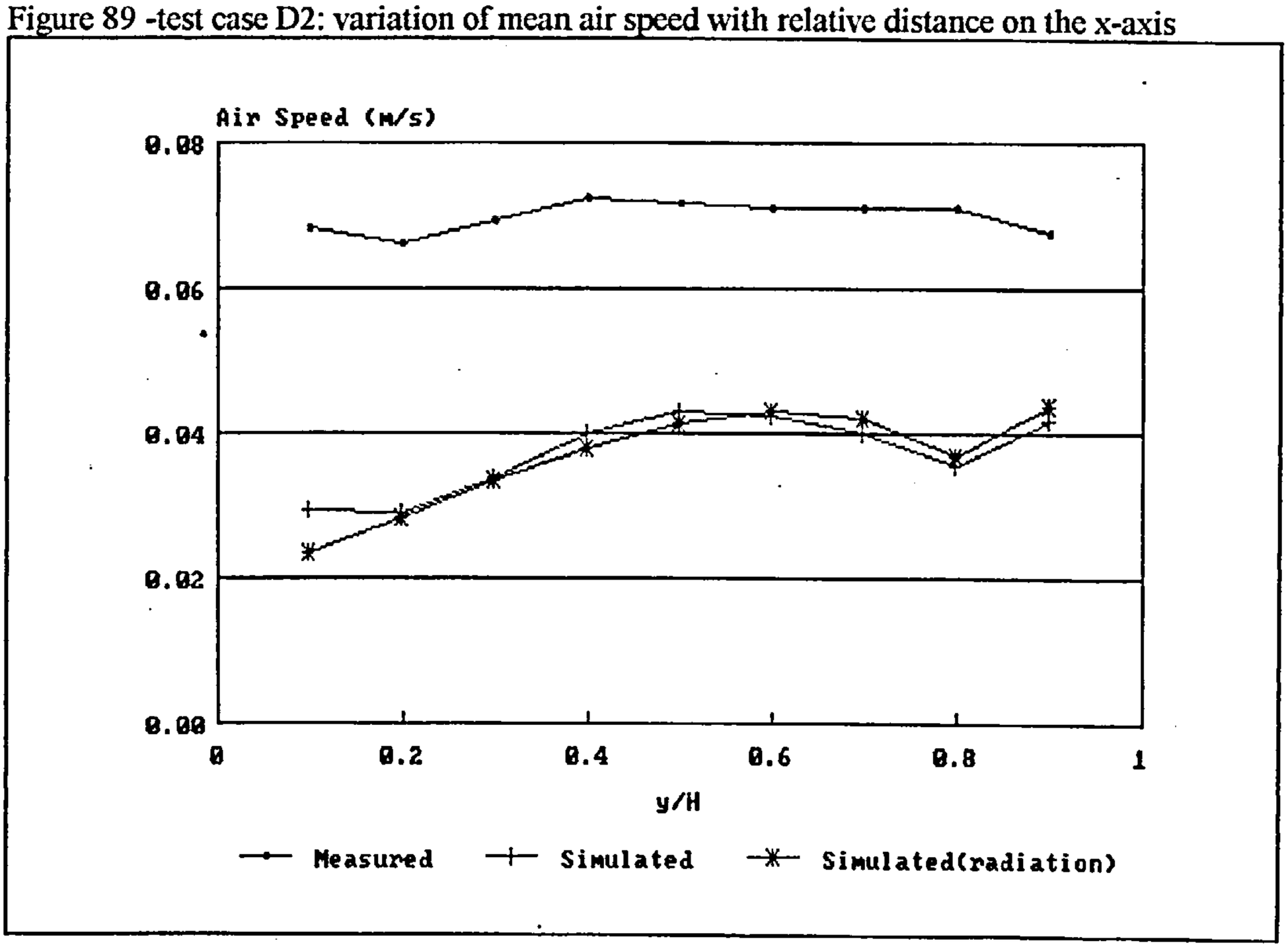


Figure 90 -test case D2: variation of mean air speed with relative distance on the y-axis

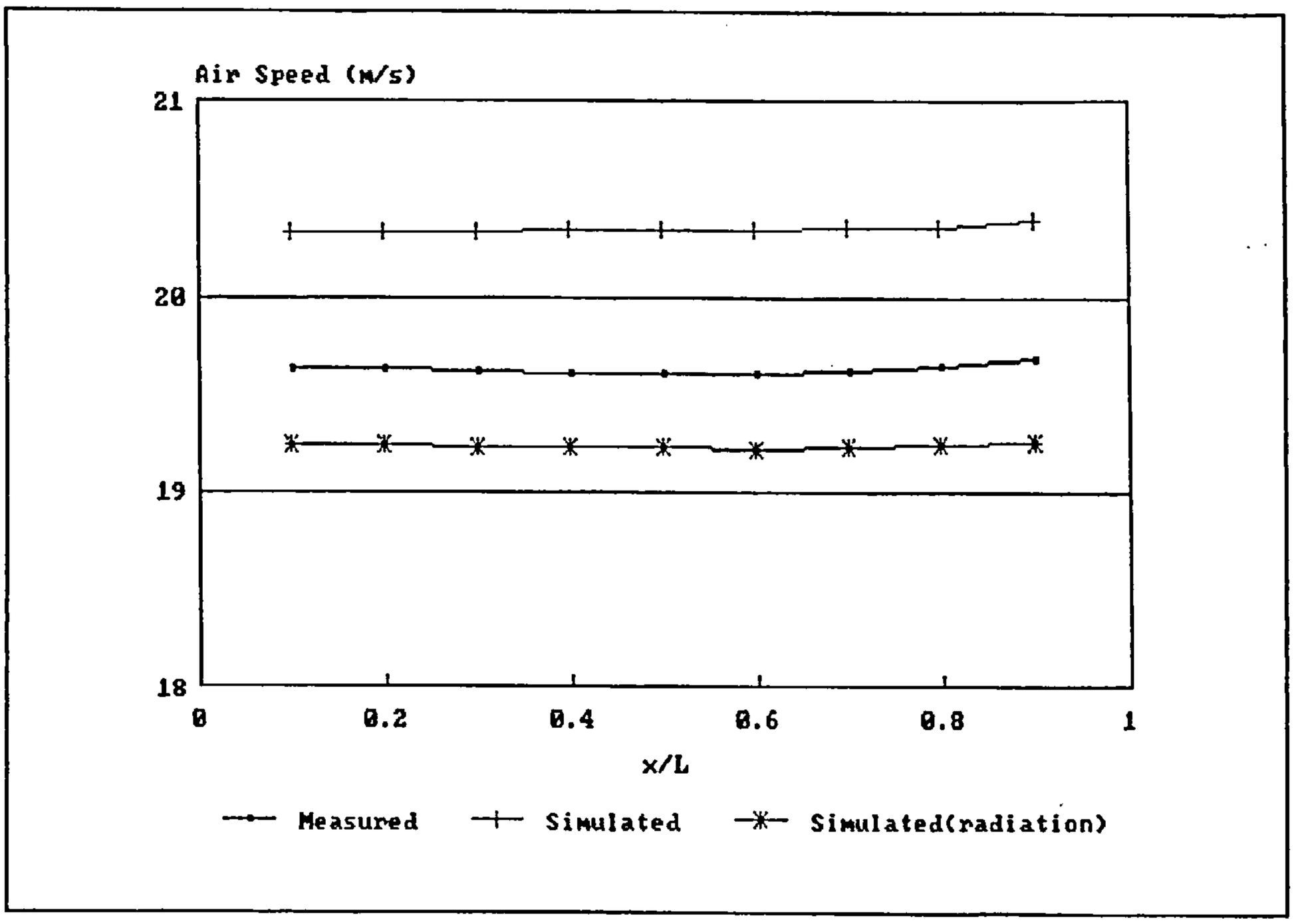


Figure 91 -test case D2: variation of temperature with relative distance on the x-axis

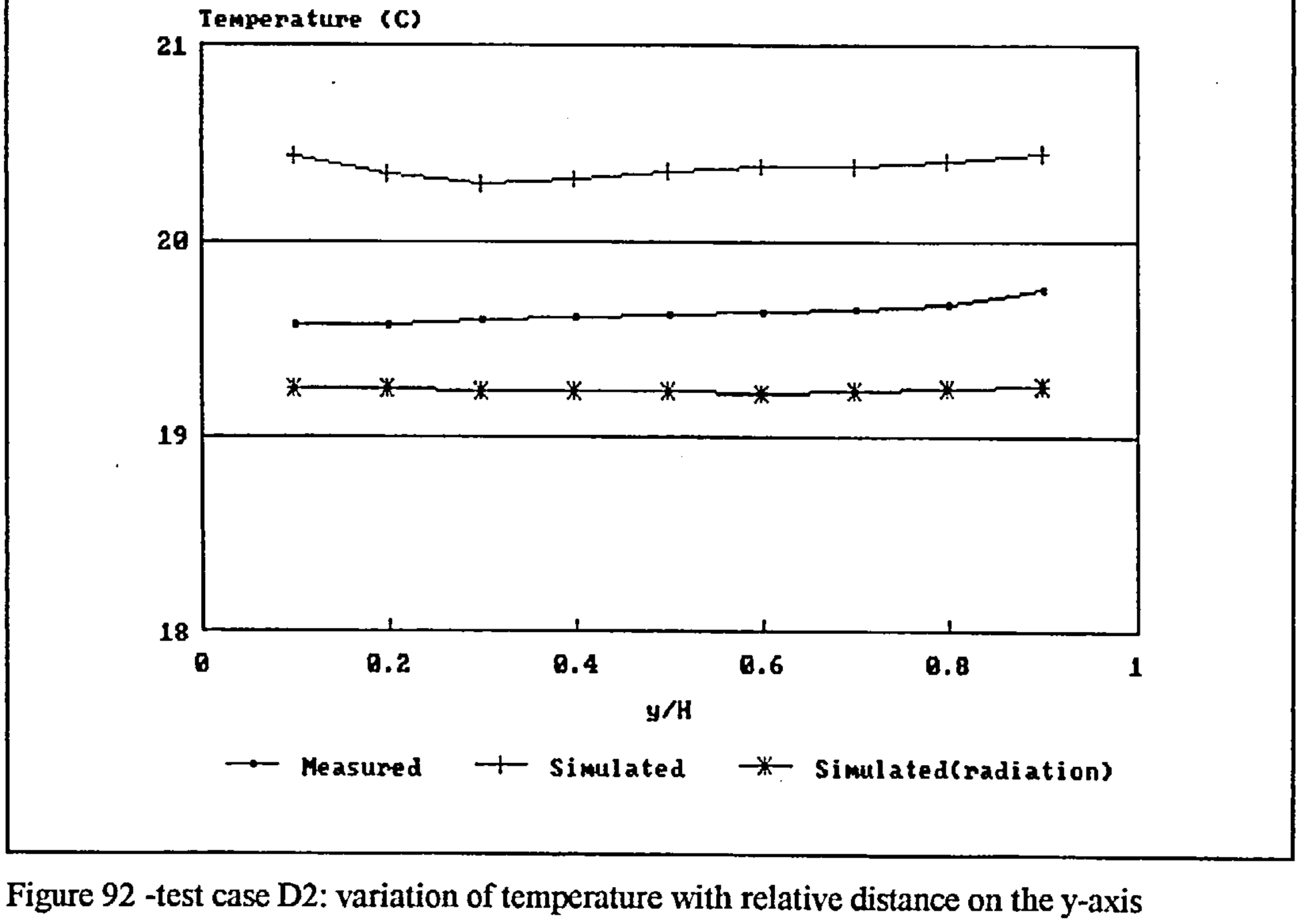


Figure 92 -test case D2: variation of temperature with relative distance on the y-axis

5.5 Results Obtained by IEA Annex 20 Participants

5.5.1 Test Case B2

A number of researchers; Heikkinen and Piira [71], Fontaine [72], Furst [73], Chen [74], Vogl and Renz [75], Said [76], Lemaire and Elkhuizen [77], Tjelflaat [78], Johansson et al [79] have simulated test case B2 using commercial CFD codes with a number of different air inlet models. A uniform air flow distribution is evident from the results and the distribution compares well with measured results.

Contours of air speeds resulting from simulation have been provided by various researchers; Heikkinen and Piira [71], Lemaire and Elkhuizen [77], Skovgaard and Neilsen [80] and Vogl and Renz [75]. Predicted air speeds at near floor level are in the range 0.1 to 0.2 m s^{-1} compared to measured speeds of 0.1 m s^{-1} and slightly higher.

5.5.2 Test Case E2

Lemaire [81] has applied the WISH3D commercial code to test case E2 and demonstrated that flow reversal across the window could occur depending on the supply inlet model employed. Simulations were found to slightly over-estimate the jet penetration. Simulations were conducted for half of the test room , assuming a plane of symmetry, however some indication that an asymmetric flow pattern may result when a coarse grid is extended across the whole room was provided.

Heikkinen [82] employed the FLUENT commercial code and obtained a reasonable representation of the flow pattern, mean velocities, jet penetration length and temperatures with measured results.

Vogl and Renz [83] similarly employed the FLUENT code and again achieved reasonable comparison with measured data.

5.5.3 Test Case D2

Lemaire [84] applied the WISH3D code to test case D2 and observed that the flow pattern was driven by buoyant flow from the radiator upwards across the cold window surface. Prescribed heat fluxes were employed for the radiator and window, conventional wall functions having been found to significantly under-predict surface convection coefficients.

Vogl and Renz [85] also applied the FLUENT code to test case D2 and achieved flow patterns and velocity distributions that were broadly consistent with measured results.

Furst [86] demonstrated that a reverse flow (downward) could be achieved if the heat transfer from the radiator was under-estimated by the adopted wall function.

5.6 Conclusions

a) Measured results for three building test cases have been provided by Annex 20 of the International Energy Agency. The first test case (B2) represents a forced convection isothermal flow, the second (E2) mixed convection under summer cooling conditions and the third (D2) free convection under winter heating conditions.

b) A HESCO multiple nozzle type diffuser was employed for test cases B2 and E2. The model has been used to predict velocities within the test room for test case B2, using two diffuser representations; a simple single jet approximation and a more complex five jet approximation. While the former diffuser type led to an over-estimation of mean air speed throughout the flow domain, the five jet representation resulted in very good agreement with measured results.

c) Although predicted results for the mixed convection test case E2 were found to contain a significant degree of asymmetry about the centre-line of the z-axis, good agreement was achieved between predicted and measured results. Results were obtained including and excluding the radiation model developed in Section 4.3. Better agreement was achieved with the radiation model, the model under-estimating air temperatures in the absence of the radiation model.

d) Good agreement was achieved between predicted and measured results for test case D2. However, predicted mean air speeds were found to be somewhat lower than measured air speeds, although this discrepancy could be partly due to the inaccuracy of air speed measurement at very low velocities. In the absence of the radiation model, predicted air temperatures were found to be over-estimated. Air temperatures predicted with the coupled radiation model were slightly under-estimated indicating the possible need for some refinement in the method for estimating surface heat transfer.

6.0 CONCLUSION

A summary of the developments in the field of Computational Fluid Dynamics, together with a review of turbulence models and associated solution methods has been presented in Chapter 1. It has been concluded from existing research that the Launder and Spalding k-ε turbulence model is capable of providing good correlations between measured and predicted results for a wide range of building air flows although research into the validation of the model for purely buoyancy driven flows is limited. A review of the research into the application of CFD to building air flows has demonstrated the need for coupled convection-conduction-radiation models and highlighted the limitations of existing approaches.

Details of the underlying mathematical model of advection, convection and conduction processes incorporating the k-ε turbulence model is presented in Chapter 3, together with details of the SIMPLER numerical procedure adopted for the solution of the equation set. General methods for handling boundary conditions and conduction in solid regions are also discussed. An approach for incorporating longwave radiation heat exchange within the same numerical solution scheme adopted for convection and conduction processes is also outlined in Chapter 3 and it is this approach that forms the basis of the coupled longwave radiation model.

Considerations for modelling shortwave and longwave radiation heat transfer processes in building spaces are discussed in Chapter 4 together with a general vector geometry method for calculating radiation view factors and internal solar tracking. An approximate method for modelling longwave radiation with a potentially high degree of accuracy is described and a procedure for incorporating this method directly within the CFD numerical solution scheme is detailed. The potential for application has been demonstrated with particular regard to air flows that may be significantly influenced by shortwave radiation and consequently the coupled model would be particularly useful in the area of passive solar design.

Finally, predictions derived from the resulting fully coupled convection-conduction-radiation model are compared with experimental measured data for three test cases, provided by Annex 20 of the International Energy Agency. It is demonstrated that the coupled model is capable of producing more realistic correlations then the convection-conduction model alone.

The method of modelling surface heat flux using wall functions requires some further research. An adhoc treatment for surface convection coefficients using an empirical approach has been adopted for this study due to concern raised by various researchers as to the accuracy of conventional energy equation wall functions.

Simulation conducted for the IEA Annex 20 test case E2 resulted in a severely asymmetric flow pattern. While it was decided to limit the investigation into this phenomenon it is an aspect of the work that requires further investigation.

Although it has been demonstrated that the k-ε turbulence model is capable of producing good correlations between measured and simulated results, it adds a considerable computational load and can introduce instabilities into the solution scheme. The development of an accurate but more computationally efficient method of modelling turbulence would greatly improve the potential of CFD as a practical design tool.

Where the air flow pattern within a building space is driven by bouyancy, considerable underrelaxation may be required in order to procure a solution due to de-coupling of the energy and momementum equations. Further research is required to determine methods for coupling the energy and momentum equations in order to improve the convergence characteristics.

APPENDIX 1 - GEOMETRIC PROCEDURES

This appendix details the various point and surface geometric operations required for building surface and construction definition, internal solar tracking and radiation view factor derivation.

AI.1 Surface Plane Equation

For the determination of points of intersection between lines and surfaces, required for internal solar tracking and the derivation of radiation view factors, the equation of the plane in which the polygon surface lies is required, i.e.:-

$$Ax + By + Cz + D = 0 \qquad \text{Eq.AI.1}$$

Considering the points (x, y, z), (x_1, y_1, z_1), (x_2, y_2, z_2) and (x_3, y_3, z_3) lying in the same plane, then:-

$$Ax + By + Cz + D = 0 \qquad \text{Eq.AI.2}$$

$$Ax_1 + By_1 + Cz_1 + D = 0 \qquad \text{Eq.AI.3}$$

$$Ax_2 + By_2 + Cz_2 + D = 0 \qquad \text{Eq.AI.4}$$

$$Ax_3 + By_3 + Cz_3 + D = 0 \qquad \text{Eq.AI.5}$$

and if there is a solution to this set of homogeneous equations, the determinant of its coefficients must be zero:-

$$\begin{vmatrix} x & y & z & 1 \\ x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ x_3 & y_3 & z_3 & 1 \end{vmatrix} = 0 \qquad \text{Eq.AI.6}$$

Expanding by cofactors about the first row:-

$$x\begin{vmatrix} y_1 & z_1 & 1 \\ y_2 & z_2 & 1 \\ y_3 & z_3 & 1 \end{vmatrix} - y\begin{vmatrix} x_1 & z_1 & 1 \\ x_2 & z_2 & 1 \\ x_3 & z_3 & 1 \end{vmatrix} + z\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} - \begin{vmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{vmatrix} = 0 \qquad \text{Eq.AI.7}$$

The determinants, including their signs are the coefficients A, B, C and D. The determinants may be expanded in order to arrive at expressions for the coefficients:-

$$A = y_1 (z_2 - z_3) + y_2 (z_3 - z_1) + y_3 (z_1 - z_2) \qquad \text{Eq.AI.8}$$

$$B = -x_1 (z_2 - z_3) - x_2 (z_3 - z_1) - x_3 (z_1 - z_2) \qquad \text{Eq.AI.9}$$

$$C = x_1 (y_2 - y_3) + x_2 (y_3 - y_1) + x_3 (y_1 - y_2) \qquad \text{Eq.AI.10}$$

$$D = -x_1 (y_2 z_3 - y_3 z_2) - x_2 (y_1 z_3 - y_3 z_1) - x_3 (y_1 z_2 - y_2 z_1) \qquad \text{Eq.AI.11}$$

The plane equation coefficients may therefore be derived with a knowledge of three non-collinear points lying on the plane.

AI.2 Rotation

With reference to Figure 93, the point P may be rotated about the Z-axis to point P' using simple trigonometric relationships:-

$$x = r \cos \phi \qquad \text{Eq.AI.12}$$

$$y = r \sin \phi \qquad \text{Eq.AI.13}$$

and

$$x' = r \cos (\theta + \phi) = r \cos \phi \cos \theta - r \sin \phi \sin \theta \qquad \text{Eq.AI.14}$$

$$y' = r \sin (\theta + \phi) = r \cos \phi \sin \theta - r \sin \phi \cos \theta \qquad \text{Eq.AI.15}$$

$$z' = z \qquad \text{Eq.AI.16}$$

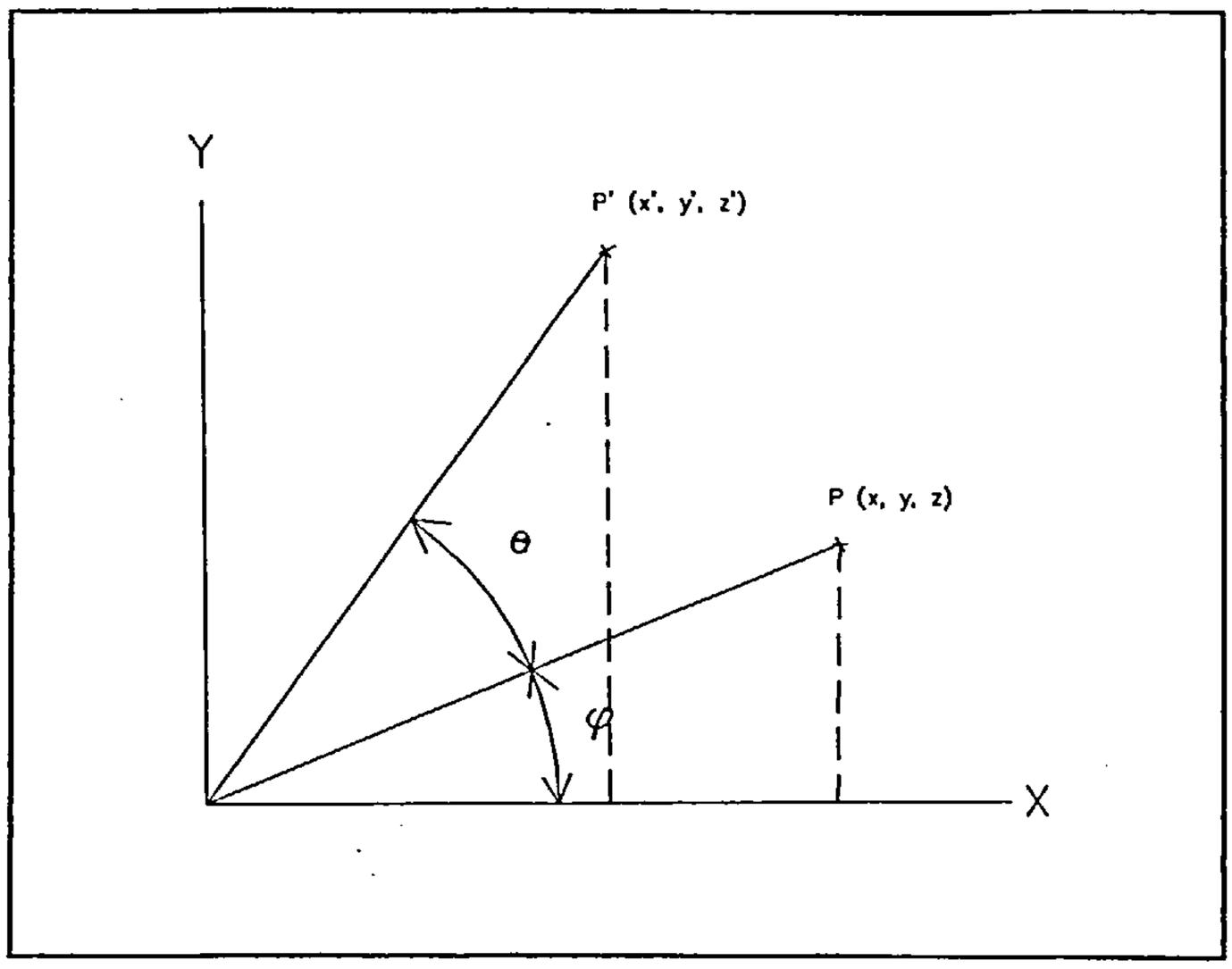


Figure 93 - rotation of a point about the Z-axis

Thus,

$$x' = x \cos\theta - y \sin\theta \qquad \text{Eq.AI.17}$$

and

$$y' = x \sin\theta + y \cos\theta \qquad \text{Eq.AI.18}$$

or in matrix form:-

$$[x'\ y'\ z'] = [x\ y\ z]\begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \text{Eq.AI.19}$$

A similar treatment results in matrices for rotations about the X and Y axes. Thus, the three rotation matrices are given as follows:-

$$Rx(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix} \qquad \text{Eq.AI.20}$$

$$Ry(\theta) = \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} \qquad \text{Eq.AI.21}$$

$$Rz(\theta) = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \text{Eq.AI.22}$$

AI.3 Translation and the Use of Homogeneous Coordinates

Points may be translated simply by adding the appropriate translation distances to the coordinates of the point. Thus, the translation of the point P(x, y, z) to a point P'(x' y' z'), a distance δx units parallel to the X-axis, δy units parallel to the Y-axis and δz units parallel to the Z-axis, would involve the following additions:-

$$[x'\ y'\ z'] = [x\ y\ z] + [\delta x\ \delta y\ \delta z] \qquad \text{Eq.AI.23}$$

Because rotation involves a multiplication, and translation an addition, these transformations are not easily combined. A common method to enable concatenation of these transformations involves the use of homogeneous coordinates. A three-dimensional point P(x, y, z) in homogeneous coordinates would be expressed as P(x, y, z, W), where W is a non-zero scale factor. In this manner, the actual Cartesian coordinates may be found through division by the scale factor, i.e. x/W, y/W and z/W and indeed this division may be obviated through the definition of a unity scale factor. Having now introduced a fourth element to the coordinate vector, translation may be conducted through matrix multiplication. Thus, the above translation becomes:-

$$[x'\ y'\ z'\ 1] = [x\ y\ z\ 1]\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \delta x & \delta y & \delta z & 1 \end{bmatrix} \qquad \text{Eq.AI.24}$$

and similarly, adopting homogeneous coordinates, the rotation matrices become:-

$$Rx(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \text{Eq.AI.25}$$

$$Ry(\theta) = \begin{bmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \text{Eq.AI.26}$$

$$Rz(\theta) = \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \text{Eq.AI.27}$$

Thus, an arbitrary number of translations and rotations may be performed on a point by multiplying various rotation and translation matrices together and then multiplying the point row vector (expressed in homogeneous coordinates) by the resulting transformation matrix. The concatenated transformation matrix is of the following form:-

$$T = \begin{bmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ t_x & t_y & t_z & 1 \end{bmatrix} \qquad \text{Eq.AI.28}$$

Note that a surface may be transformed simply by transforming each of the bounding vertices.

AI.4 Surface Visibility

A surface may be defined as being visible from a particular view point if the angle between an outward facing normal to the surface and a line drawn between the point of intersection of the normal with the surface and the view point lies in the range 0 - 90 degrees.

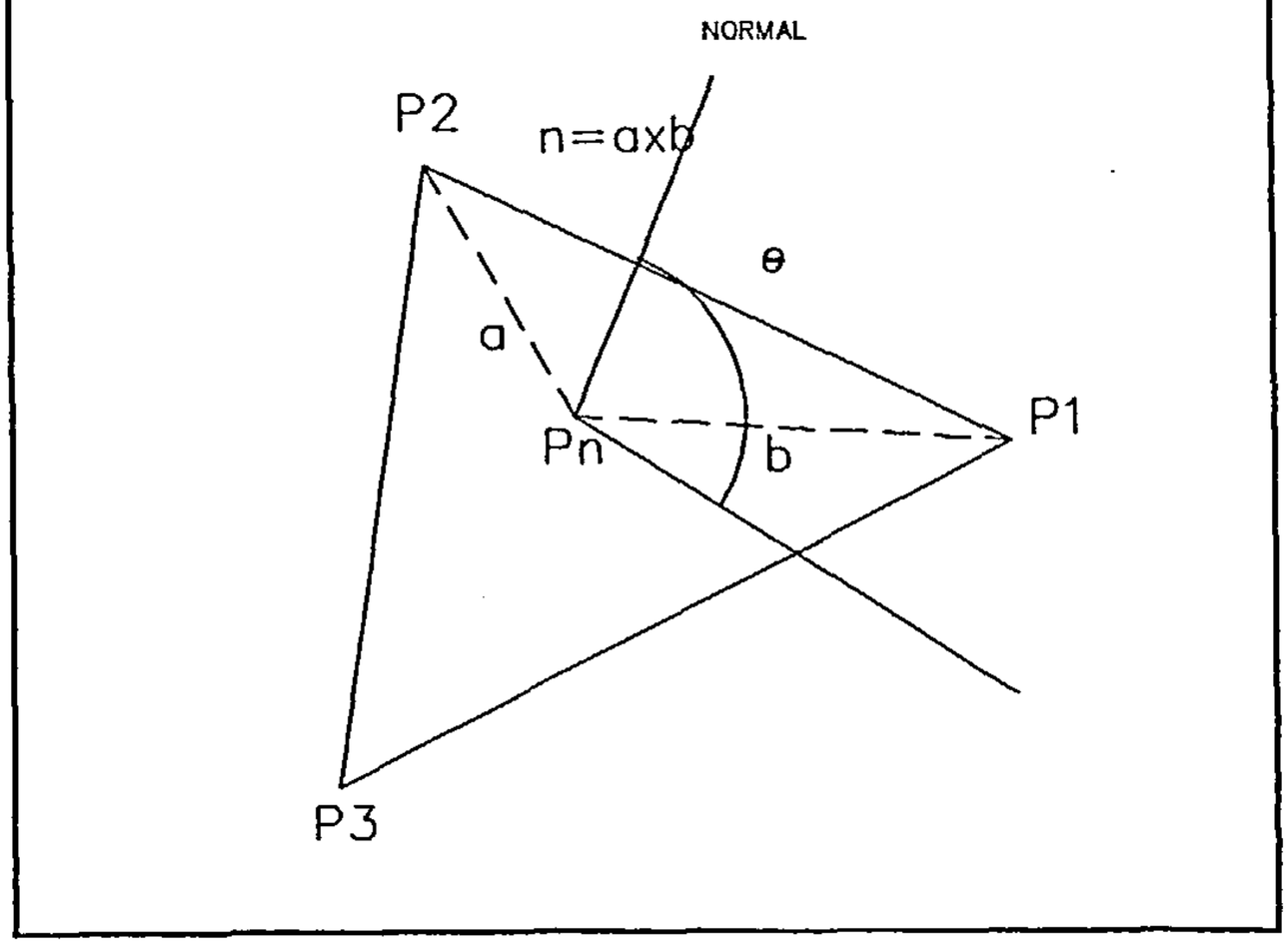


Figure 94 - surface visibility

With reference to Figure 94, a normal to the surface defined by points P_1, P_2 and P_3 may be constructed by taking advantage of the properties of the vector cross product. The cross product of the vectors **a** and **b** (defined by the points $P_n P_1$ and $P_n P_2$ respectively) is:-

$$\mathbf{a} \times \mathbf{b} = \begin{bmatrix} i & a_x & b_x \\ j & a_y & b_y \\ k & a_z & b_z \end{bmatrix} \qquad \text{Eq.AI.29}$$

Deriving the value of the determinant for Eq.AI.29:-

$$\mathbf{a} \times \mathbf{b} = (a_y b_z - a_x b_y)i - (a_x b_z - a_x b_x)j - (a_x b_y - a_y b_x)k \qquad \text{Eq.AI.30}$$

and in component form:-

$$\mathbf{a} \times \mathbf{b} = [(a_y b_z - a_x b_y) \quad -(a_x b_z - a_x b_x) \quad (a_x b_y - a_y b_x)] \qquad \text{Eq.AI.31}$$

If $\mathbf{n} = \mathbf{a} \times \mathbf{b}$, then **n** is always perpendicular to **a** and **b**.

Note that for internal building surfaces (i.e. clockwise vertex ordering), the normal will point in the opposite direction.

Having established a normal to the surface at the point P, a vector may be constructed in the direction of the view point VP, **v**. The angle θ between the vectors **n** and **v** may then be derived from

the vector dot product:-

$$\theta = \cos^{-1} \frac{\mathbf{n}.\mathbf{v}}{|\mathbf{n}||\mathbf{v}|} \qquad \text{Eq.AI.32}$$

where $|\mathbf{n}|$ and $|\mathbf{v}|$ represent the magnitudes of vectors **n** and **v**:-

$$|\mathbf{n}| = \sqrt{n_x^2 + n_y^2 + n_z^2} \qquad \text{Eq.AI.33}$$

Note that the magnitude of θ is used in the derivation of radiation view factors, detailed in Section 4.3.1.

AI.5 Line/Surface Intersection

If a line and surface intersect, then they will have a common point P_i as illustrated in Figure 95. In order to determine this point, the equation of the plane in which the surface lies must be solved in addition to the equations for the line:-

$$A\,x_i + B\,x_i + C\,z_i + D = 0 \qquad \text{Eq.AI.34}$$

$$x_i = (x_1 - x_0)\,u_i + x_0 \qquad \text{Eq.AI.35}$$

$$y_i = (y_1 - y_0)\,u_i + y_0 \qquad \text{Eq.AI.36}$$

$$z_i = (z_1 - z_0)\,u_i + z_0 \qquad \text{Eq.AI.37}$$

u_i may be derived by substitution of Equations AI.35 - AI.37 into Equation AI.34:-

$$u_i = \frac{A\,x_i + B\,x_i + C\,z_i + D}{A\,(x_1 - x_0) + B\,(y_1 - y_0) + C\,(z_1 - z_0)} \qquad \text{Eq.AI.38}$$

If $0 < u_i < 1$, then the intersection occurs and Equations AI.35 - AI.37 may be solved for x_i, y_i and z_i.

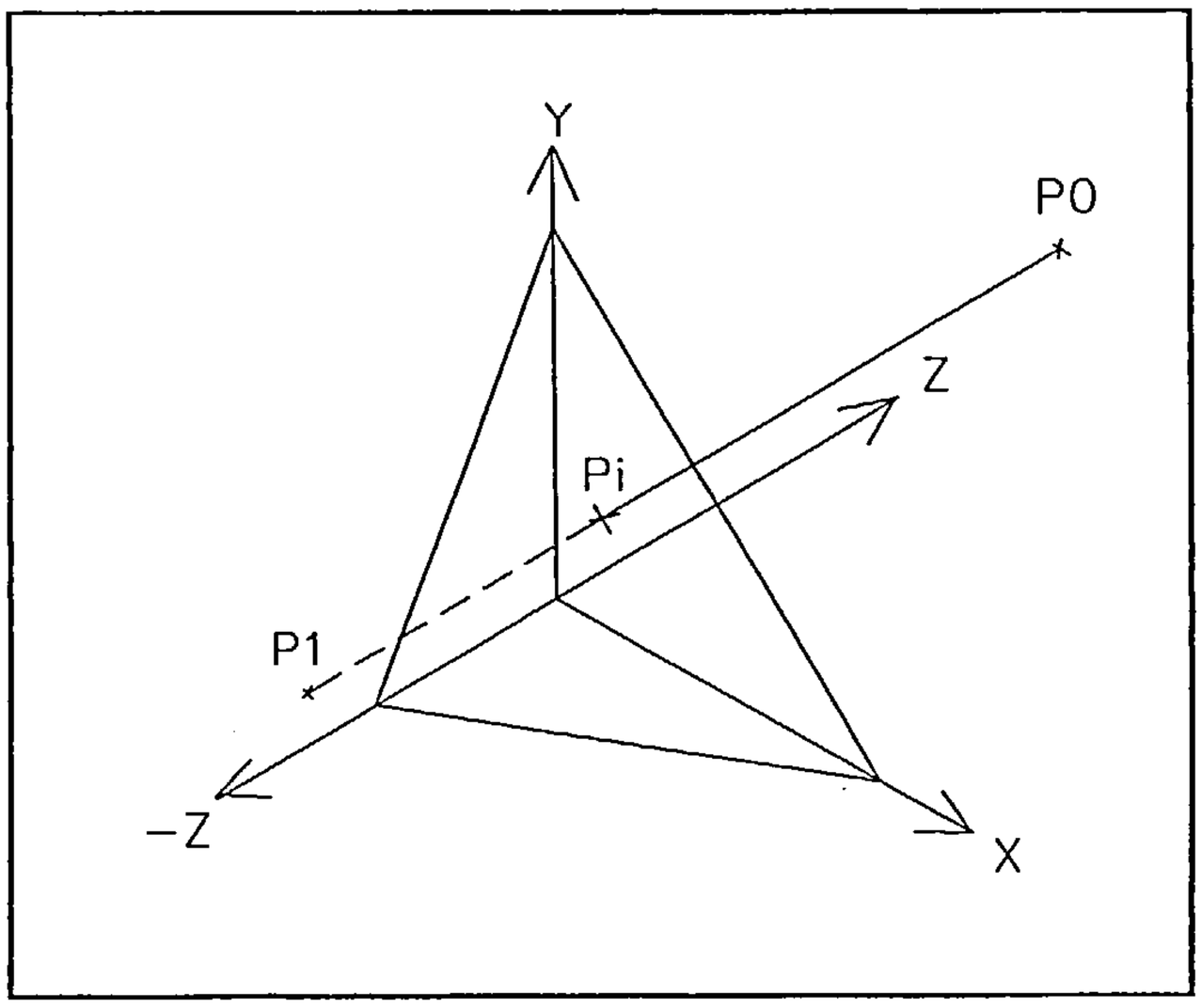


Figure 95 - line/surface intersection

AI.6 Polygon Point Containment Tests

In order to determine whether or not the point P_T, lies within the polygon illustrated in Figure 96, the maximum and minimum x-y coordinates of the polygon are first determined. The x-y coordinates of P_T are then tested to determine whether or not they lie in the intervals x_{max} - x_{min} and y_{max} - y_{min}. If P_T is found to lie within the maximum and minimum x-y intervals, the intersections of $y = y_T$ are calculated for each edge of the polygon. The x-coordinates of the intersections are then paired in ascending order (there always being an even number of intersections) and the x-coordinate of P_T is then tested to see if it lies in the paired x-coordinate intervals, i.e. x_1 - x_2, x_3 - x_4, etc.

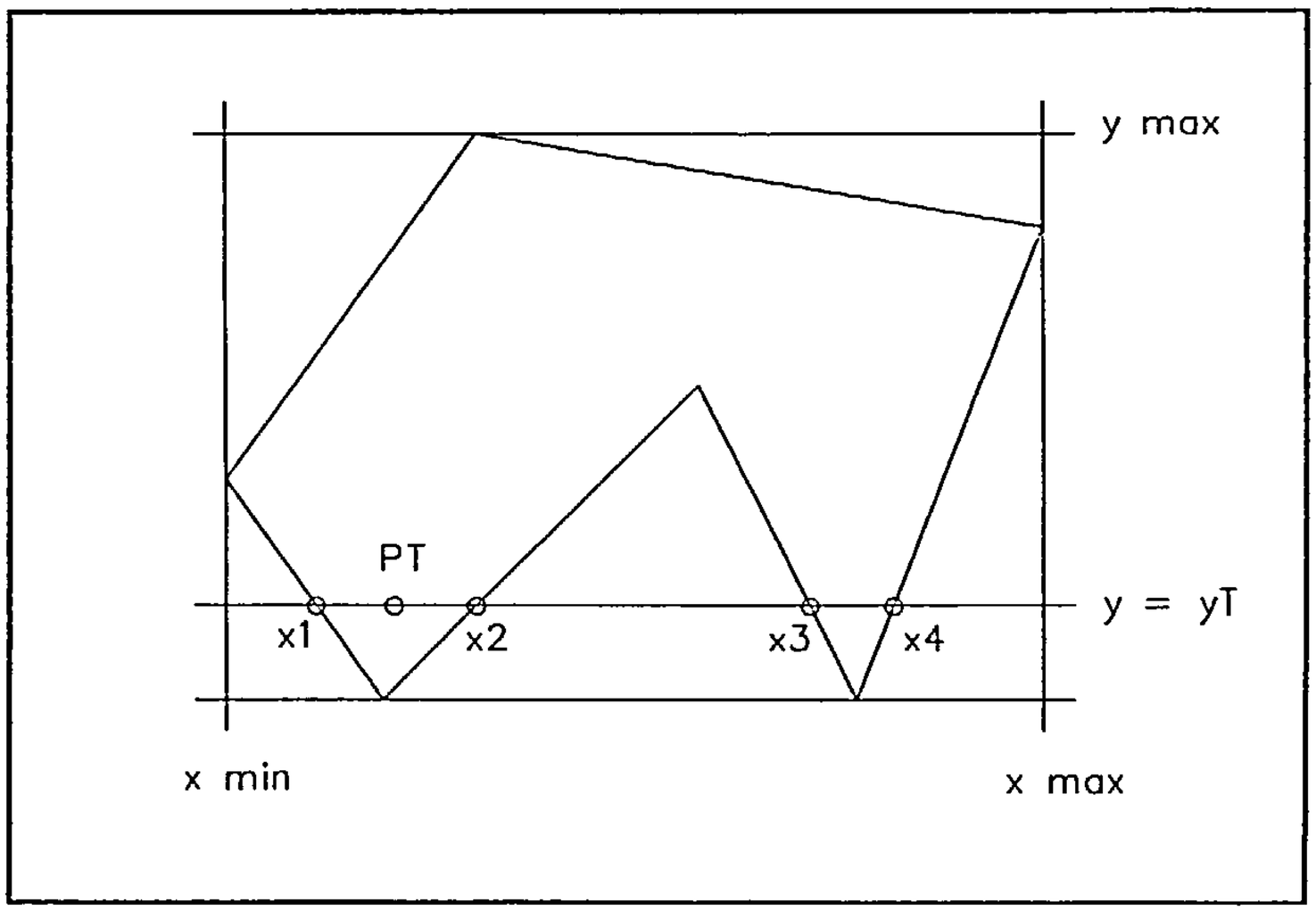


Figure 96 - point containment test

APPENDIX II - THE COMPUTER CODE

This appendix details the structure of the group of computer programs that have been developed from the numerical procedures described in Chapters 3 and 4. The program code has been developed using the 'C' programming language. The resulting group of program modules have been collectively named EMB, standing for Energy and Mass transfer in Buildings.

The code was orginally compiled and linked using Version 6.0 of the Microsoft C compiler. All source code complies with the ANSI standards with the exception of graphics functions employed by the EMBview module which are detailed in Section AII.3.4.5. All EMB program modules may therefore be compiled and linked to execute on various hardware platforms with the exception of EMBview which has been developed to run specifically on a PC.

AII.1 The Finite Volume Grid, Data Structures and Memory Allocation

The numerical solution scheme requires the calculation domain to be sub-divided into a series of parallel planes in the X, Y and Z dimensions, resulting in a three-dimensional rectilinear finite volume grid. The practice adopted in this study is to locate the grid points for the dependent scalar quantities at the geometric centres of the finite volumes thus defined. A convention is adopted whereby the planar sub-divisions in the X, Y and Z directions are referred to as columns, rows and divisions, respectively.

The finite volume cell dimensions for scalar quantities are stored in a two-dimensional array, *gd[3][MAXDIM]*, where the first index is referenced using the defined constants *COL, DIV* and *ROW*, and *MAXDIM* is a constant defining the maximum number of cells allowed in any direction. The number of columns, rows and divisions defined for a specific problem are stored in the variables *ncols*, *nrows* and *ndivs*. The U, V and W velocity component grids are staggered with respect to the scalar quantity grid by locating the U-component grid points midway between the scalar grid points in the X-direction and similarly the V and W components in the Y and Z directions respectively. Due to the employment of staggered grids for the velocity components, the number of velocity component cells in the direction of the component are always one less than the number of scalar cells in that direction, i.e. *ncols-1* for the U-component, *nrows-1* for the V-component and *ndivs-1* for the W-component. A complete grid is illustrated in Figures 97 and 98 and all finite volume grid dimensions for both the scalar grid and staggered velocity component grids, together with the variable names as they appear in the code are presented in Figure 99.

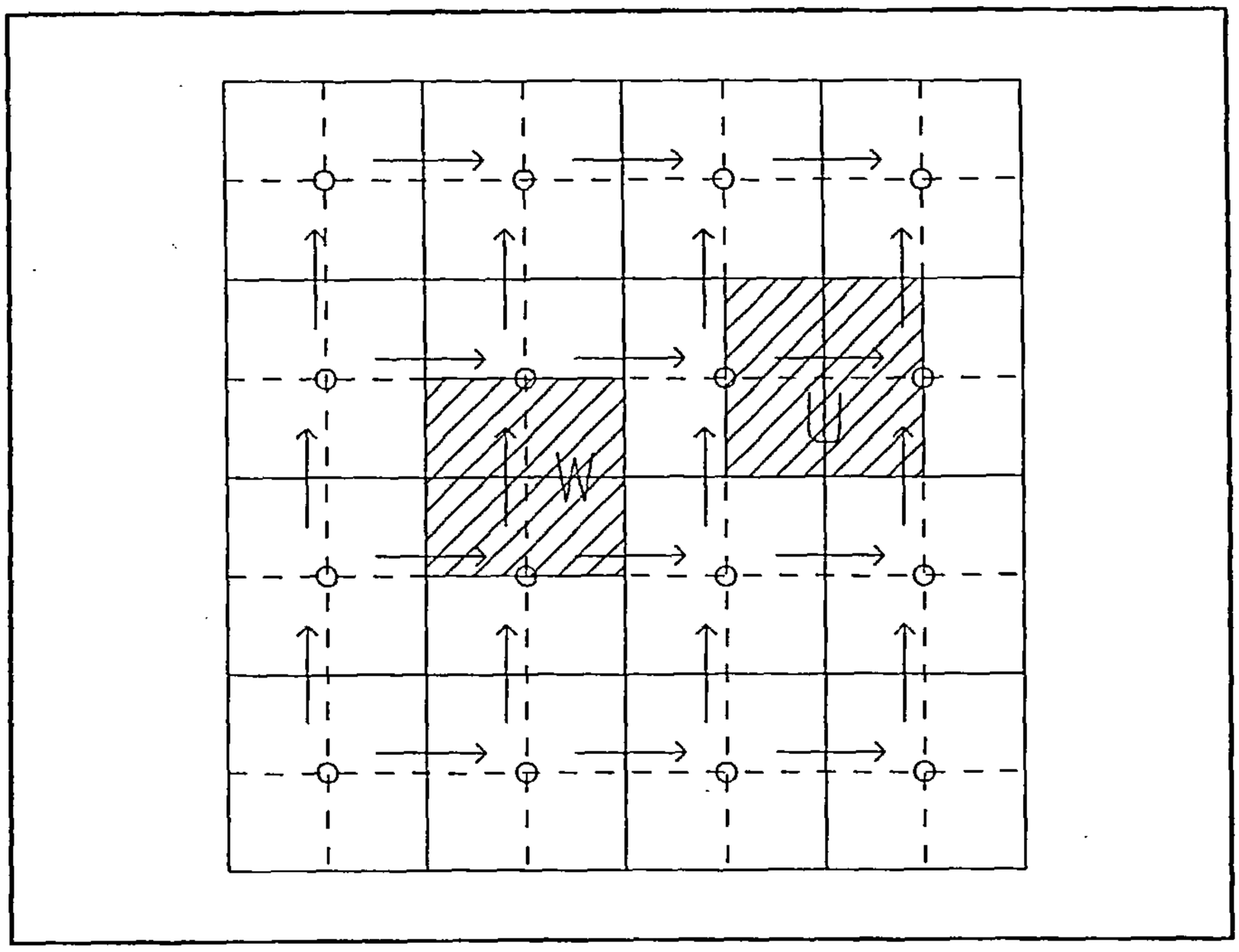


Figure 97 - X-Z plane of finite volume grid

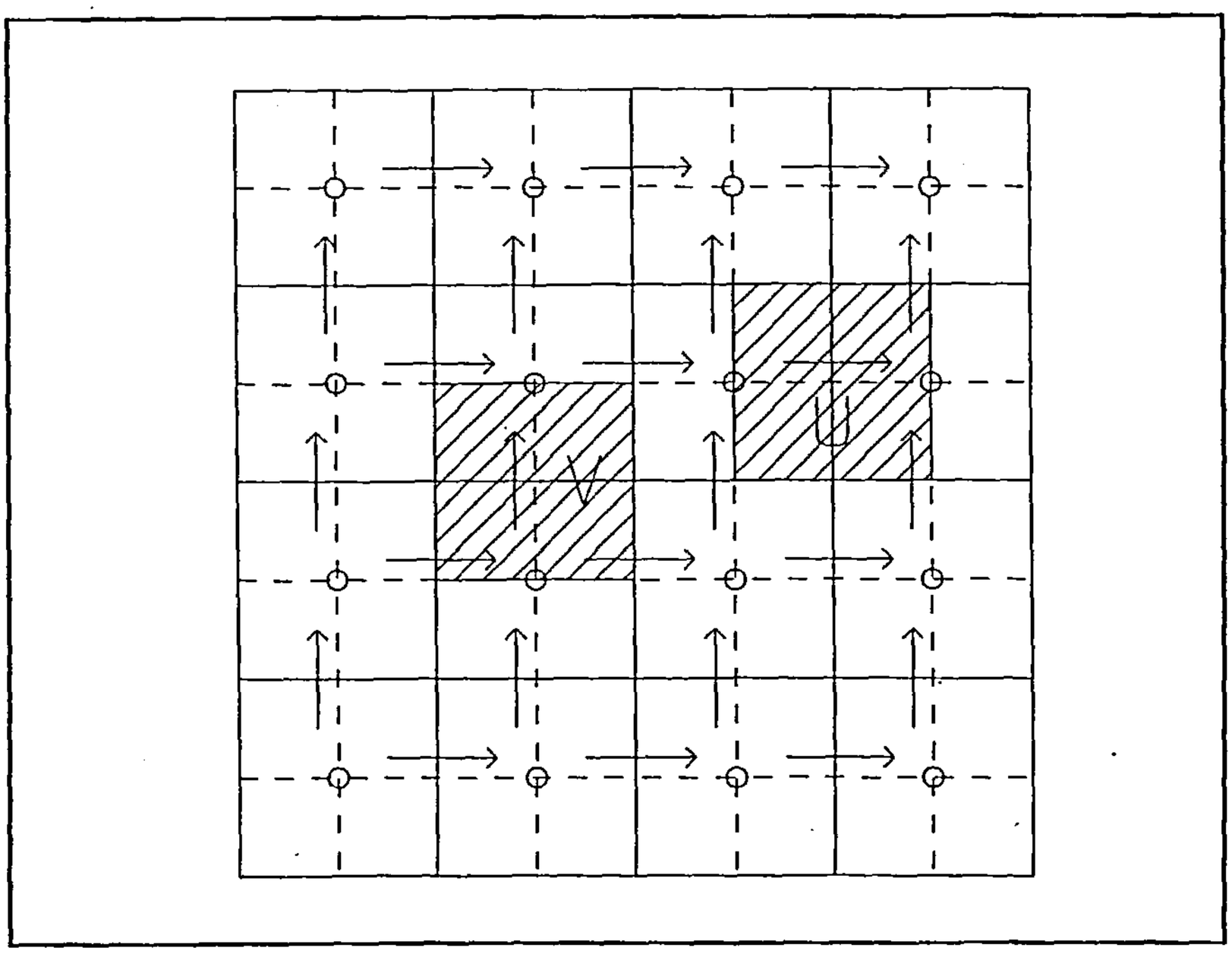


Figure 98 - X-Y plane of finite volume grid

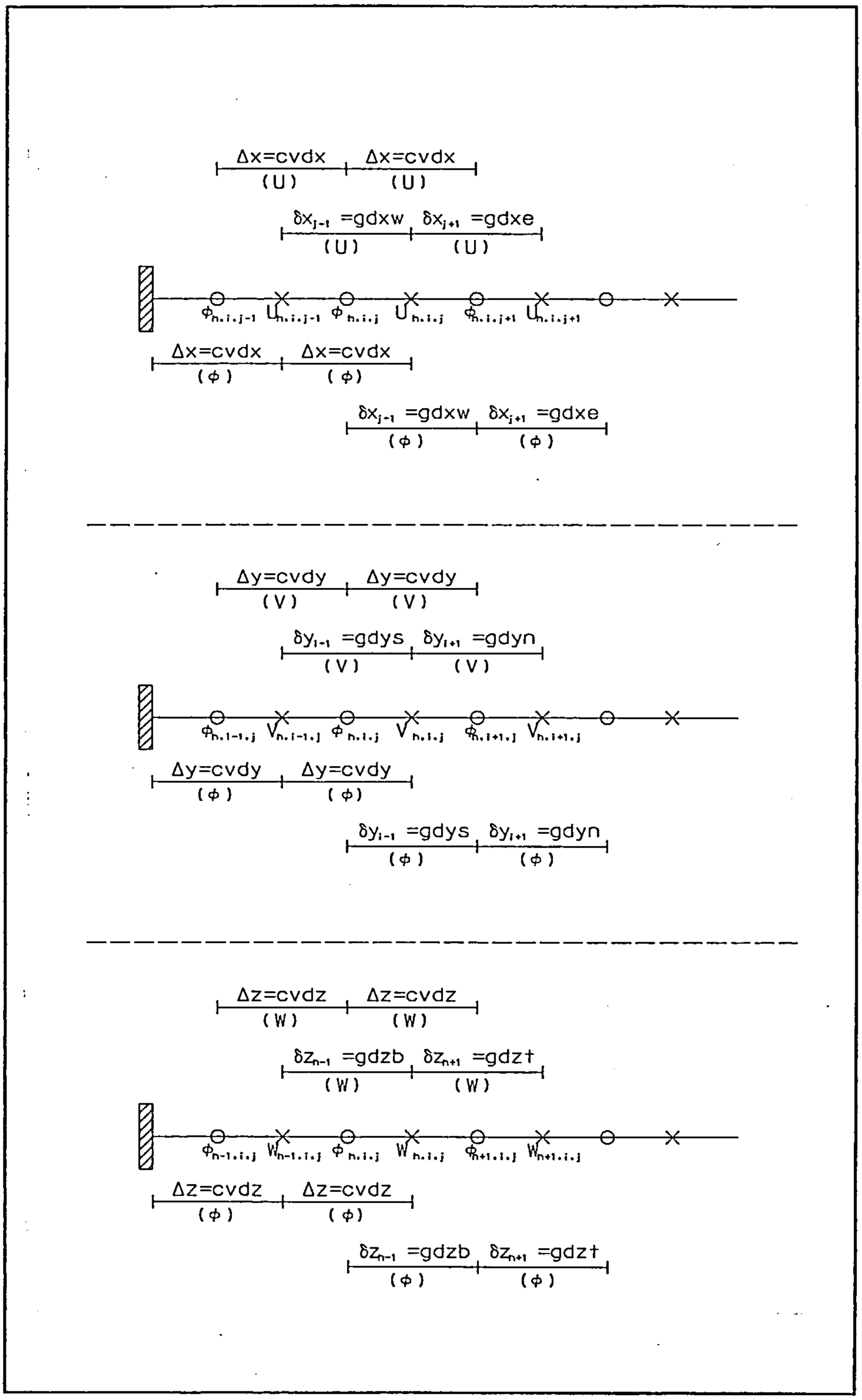


Figure 99 - Finite Volume Grid Dimensions

All grid point variables are stored in a four-dimensional array of single precision floating point numbers, the base address of which is assigned to the pointer ****grid*. The first allocated dimension of the *grid* array is the variable index (the size of which is defined using the constant *MAXVAR*) and the remaining three are array element location indices, the sizes of which are problem specific and are stored in the variables *ndivs*, *nrows* and *ncols*. The array is dynamically allocated, using the C contiguous memory allocation function *calloc()* in the following code:-

```
if((grid=(float ****)calloc(MAXVAR, sizeof(float ***))) == NULL)
    leave("Insufficient memory for grid");

for(g=0; (g<=MAXVAR-1); g++)
  if((grid[g]=(float ***)calloc(ndivs+2, sizeof(float **))) == NULL)
    leave("Insufficient memory for grid");

for(g=0; (g<=MAXVAR-1); g++)
  for(h=0; (h<=ndivs+1); h++)
    if((grid[g][h]=(float **)calloc(nrows+2, sizeof(float *))) == NULL)
      leave("Insufficient memory for grid");

for(g=0; (g<=MAXVAR-1); g++)
  for(h=0; (h<=ndivs+1); h++)
    for(i=0; (i<=nrows+1); i++)
      if((grid[g][h][i]=(float *)calloc(ncols+2, sizeof(float))) == NULL)
        leave("Insufficient memory for grid");
```

The *grid[][][][]* variable index is referenced through a series of constants which are defined in the *defs.h* include file as follows:-

U_PSVEL:	U-component pseudo-velocity ($\hat{U}$)
V_PSVEL:	V-component pseudo-velocity ($\hat{V}$)
W_PSVEL:	W-component pseudo-velocity ($\hat{W}$)
U_STAR:	U-component guessed velocity (U^*)
V_STAR:	V-component guessed velocity (V^*)
W_STAR:	W-component guessed velocity (W^*)
U_VEL:	U-component velocity (U)
V_VEL:	V-component velocity (V)

W_VEL:	W-component velocity (W)
TEMP:	Temperature (T)
TURB_KE:	Turbulence kinetic energy (k)
TURB_DISS:	Dissipation rate of turbulence kinetic energy (ε)
PRESSURE:	Pressures (P)
PRESSURE_C:	Pressure corrections (P')
U_COEFF:	Velocity U-component grid point coefficient
V_COEFF:	Velocity V-component grid point coefficient
W_COEFF:	Velocity W-component grid point coefficient
GDE_N:	General differential equation coefficient - NORTH (h, i+1, j)
GDE_S:	General differential equation coefficient - SOUTH (h, i-1, j)
GDE_E:	General differential equation coefficient - EAST (h, i, j+1)
GDE_W:	General differential equation coefficient - WEST (h, i, j-1)
GDE_T:	General differential equation coefficient - TOP (h+1, i ,j)
GDE_B:	General differential equation coefficient - BOTTOM (h-1, i, j)
GDE_BP:	General differential equation 'b' term
GDE_AP:	General differential equation grid point coefficient (h, i, j)
VISCOSITY:	Laminar or turbulent viscosity
FLUX:	Total thermal flux to cell
CASUAL:	Casual heat gains to cell
U_VEL0:	U-component velocity (U) at previous time increment
V_VEL0:	V-component velocity (V) at previous time increment
W_VEL0:	W-component velocity (W) at previous time increment
TEMP0:	Temperature (T) at previous time increment
TURB_KE0:	Turbulence kinetic energy (k) at previous time increment
TURB_DISS0:	Dissipation rate of turbulence kinetic energy (ε) at previous time increment

A series of finite volume cell 'switches' and the polyhedron-surface index numbers to which the cell belongs are stored in grid parameter structures:-

```
struct gridpar_st {

    struct gridparbf_st gridpar_bf;
    unsigned char mat, nb, bd[6], sf[6];
```

```
};
```

where *mat* is the materials database index number (see Section 4.1.3), *nb* is the number of polyhedra to which the cell belongs and *bd[6]* and *sf[6]* are arrays of polyhedra and surface pairings. The gridparbf_st structure is a bit field structure which is used for the storage of two-state variables:-

```
struct gridparbf_st {
    unsigned bfstatic:1;            /* Static cell flag */
    unsigned bfisothermal:1;        /* Isothermal cell flag */
    unsigned bfisouvel:1;           /* Constant U component cell flag */
    unsigned bfisovvel:1;           /* Constant V component cell flag */
    unsigned bfisowvel:1;           /* Constant W component cell flag */
    unsigned bfexternal:1;          /* External air cell flag */
    unsigned bfexsurface:1;         /* External surface cell flag */
    unsigned bfintsurface:1;        /* Internal surface cell flag */
    unsigned bfobsurface:1;         /* Obstacle surface cell flag */
    unsigned bfencsurface:1;        /* Enclosure surface cell flag */
    unsigned bfirradiated:1;        /* Irradiated cell flag */
    unsigned bfobstacle:1;          /* Obstacle cell flag */

};
```

The *gridpar_st* structures are stored in the form of a three-dimensional array which is dynamically allocated using a similar algorithm as that used for the allocation of the *grid* array.

The data required for building space polyhedra as defined in Section 4.1, are stored in geometric body structures:-

```
struct body_st {
    unsbyte nv, ns, type;
    double vt[MAXV+1][4];
    struct surface *sf[MAXS+1];

};
```

where the various fields are defined as follows:-

nv:	Number of vertices
ns:	Number of surfaces
type:	Polyhedron type (internal, external, enclosure, etc.)
vt[MAXVERT+1][4]:	Cartesian coordinates of each polyhedron vertex, using the homogeneous coordinate system discussed in Appendix I.
*struct surface *sf[MAXS+1]:*	Array of pointers to surface structures.

MAXVERT and *MAXS* are constants used to define the maximum number of vertices and surfaces that may be associated with a polyhedron.

The **sf[]* array is an array of pointers to the following polyhedron surface structures:-

```
struct surface {

  float vf[MAXBODY+1][MAXS+1];
  unsbyte sfv[MAXSV+1],  type;
  float A, B, C, D,  flux, absorp, trans;
  float alpha, beta, area, volume, hc, mrt;

};
```

where the various fields are defined as follows:-

vf[MAXBODY+1][MAXS+1]:	Radiation view factors. MAXBODY is constant defining the maximum number of geometric bodies allowed. MAXS is as defined earlier.
sfv[MAXSV+1]:	Bounding vertex sequence for surface. The vertex index numbers used in the sequence point to the vertex array element defined in the associated body structure.
type:	Surface type (Wall, window, door, isothermal surface).

A, B, C, D:	Plane equation coefficients for the surface.
flux:	Used in the calculation of incident solar radiation.
absorp, tran:	Absorptivity and transmissivity, used for the calculation of solar flux.
alpha, beta:	Surface azimuth and elevation angles.
area:	Surface area.
volume:	Total volume occupied by finite volume cells associated with the surface.
hc:	External surface heat transfer coefficient.
mrt:	Mean radiant temperature of surface.

The amount of data associated with each surface structure is extensive and thus to economise on system memory, the surface structures are dynamically allocated using the C *malloc()* function and each allocated structure assigned to an element of the surface structure pointer array associated with the appropriate body structure.

The ****grid* and ****gridpar* pointers are defined in *main.c* and declared as external variables to all functions by means of the include file *main.h*. An array of geometric body structures *bd[]* is similarly defined and declared.

Other variables which are required by several functions and are thus defined and declared as external are as follows:-

a) Finite volume dimension array: *gd[]* (storage class: double).

b) Number of divisions, rows and columns, as defined in the problem data file; *ndivs*, *nrows* and *ncols* (storage class: int).

c) Simulation data structure: *struct simdata_st simdata.*

The *simdata_st* is defined as follows:-

```
struct simdata_st {

    char project[100], date[20], infile[20];
    char bcfile[20], wfile[20], vfile[20];
    int maxicount, ssec, fsec, sday, fday, shour;
    int fhour, smonth, fmonth, tmodel, rmodel, imodel, dmodel, ni[MAXVAR];
    double dt,  igrxf[MAXVAR];
    double ogrxf[MAXVAR], rtemp;
    double dumpint, cevf, tol[MAXVAR];

};
```

where each field represents the following:-

project[100]:	project name
date[20]:	simulation date
infile[20]:	binary model file name
bcfile[20]:	boundary condition file name
wfile[20]:	weather file name
vfile[20]:	view factor file name
maxicount:	maximum number of outer loop iterations
ssec, shour,sday, smonth:	simulation start time
fsec, fhour, fday, fmonth:	simulation finish time
tmodel:	K-ε turbulence model (ON/OFF)
dmodel:	dynamic model (ON/OFF)
rmodel:	radiation model (ON/OFF)
imodel:	isothermal model (ON/OFF)
ni[MAXVAR]:	number of inner iterations for each dependent variable
igrxf[MAXVAR]:	inner loop relaxation factors (if SOR solver employed)
ogrxf[MAXVAR]:	outer loop relaxation factors
tol[MAXVAR]:	outer iteration termination tolerances for each dependent variable

cevf: viscosity multiplier for constant effective viscosity turbulence model

rtemp: calculation reference temperature

dt: time step for dynamic calculation

dumpint: calculation results output interval for dynamic calculation

d) Material properties structure array: *struct material_st mat[MAXMAT+1]*, where the material structures are defined as follows:-

```
struct material_st {

    double  rho, cp, k, absorp, trans;

};
```

where each field represents the following:-

rho: density

cp: specific heat capacity

k: thermal conductivity

absorp: absorptivity

trans: transmissivity

e) An array of weather data structures: *struct WeatherDataSt WeatherData[NMONTHS+1][NDAYS+1][NHOURS+1]*, where the weather data structures are defined as follows:-

```
struct WeatherDataStruct {

    int TotalHorizontalIrradiation, DiffuseHorizontalIrradiation;
    int DryBulbTemperature, WindSpeed, WindDirection;

};
```

f) A connection structure

```
struct connection_st {
        unsbyte bd1, sf1, bd2, sf2, mat, type;
};
```

where each field is defined as follows:-

bd1:	index number of first geometric body
sf1:	index number of first geometric body surface
bd2:	index number of second geometric body
sf2:	index number of second geometric body surface
mat:	index number of material record in materials database, *embsim.mat* (refer to d) above)
type:	connection type (wall, door, window, etc.)

AII.2 Overview of the EMB Program

The program comprises five executable files, two problem description files and four system data files. The relationship between the executable, problem description and system data files is illustrated graphically in Figure 100 and the function of each module is described in Section AII.2.1.

AII.2.1 Problem Definition and Data Entry - The EMBgen Program Module

The first stage in the process of problem definition is to establish the rectilinear perimeter of the geometric problem domain in terms of the extent of the domain along the principal X, Y and Z axes. The required grid resolution may then be determined for the various regions of the calculation domain by considering the domain geometry (location of internal and external surfaces, doors and windows), probable regions of steep dependent variable gradients, the location of flow obstacles and the geometry of air diffusers, grilles, heat sinks, heat sources, etc.

The calculation domain is then sub-divided into the appropriate finite volume cell dimensions along the X, Y and Z axes. Having established the extent and dimensions of the finite volume grid, a series of polyhedra are defined using the convention described in Section 4.1, in order to establish the geometry of internal and external surfaces, enclosures, interfaces and obstacles. A series of inter-surface connections may then be defined in order to establish surface types (walls, doors, windows and isothermal surfaces) and to assign the correct materials database index number to each finite

volume cell.

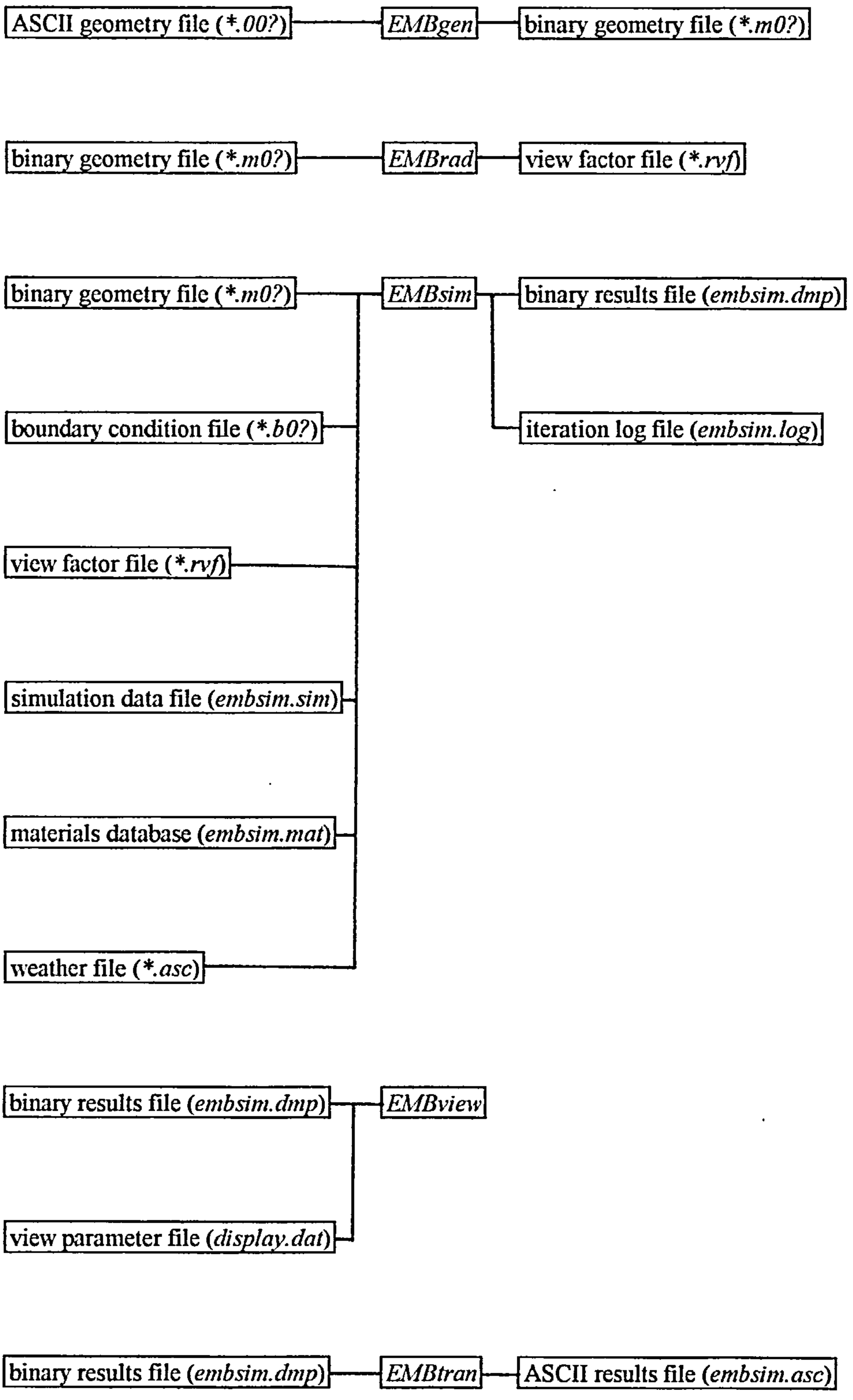


Figure 100 - Relationship between program files and executable modules

A data file is then prepared as an ASCII text file, using a text editor. An example of a geometry data file is included in Appendix III. The ASCII geometry data file is passed to the EMBgen executable module as the first of two arguments, the second being the name of the binary file to be generated, e.g:-

embgen <test.001> <test.m01>

The EMBgen module employs a series of functions that enable a set of parameters to be established for each finite volume cell, the parameters being stored in a *gridpar_st* structure as defined above. In addition to the finite volume cell parameters, the geometric body and surface structures are established for each polyhedron and the constituent surface azimuth, altitude angles and area are calculated. The function calls employed by EMBgen are illustrated in Figure 101 and each function together with arguments and return value is described in Section AII.3.2.

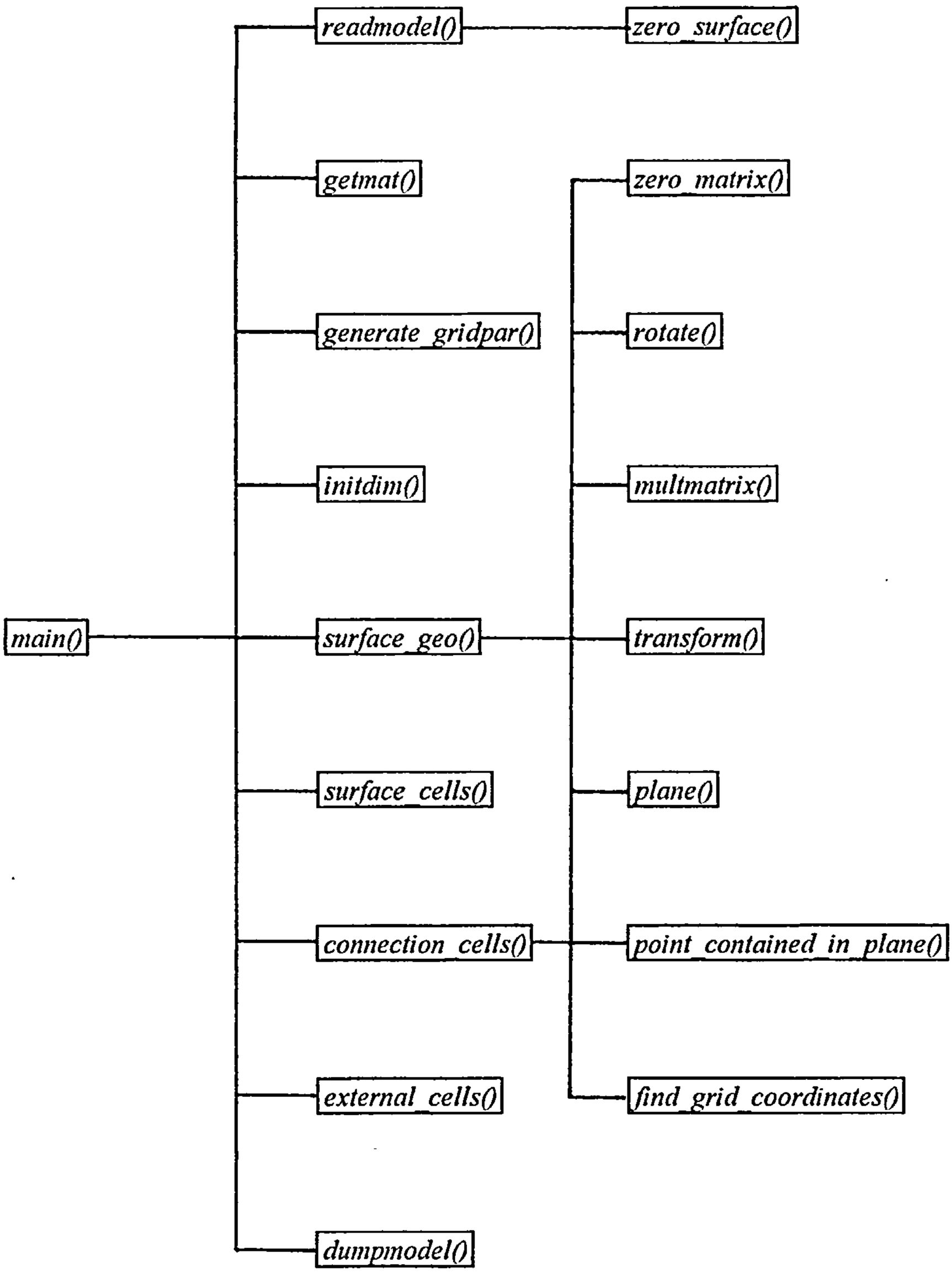


Figure 101 - Function calls made by the EMBgen *main()* function

AII.2.2 Calculation of Radiation View Factors - The EMBrad Program Module

The EMBrad module is used to calculate radiation view factors for the various polyhedron surface configurations defined in the ASCII model geometry file (refer to the EMBgen module under Section AII.2.1). The module requires two arguments, the first being the name of the binary model geometry file, the second being the name of the ASCII view factor file to be generated, eg:-

embrad <test.m01> <test.vwf>

The module employs a call to the function *view_factors()* which in turn makes a sequence of function calls which collectively represent the algorithm detailed in Section 4.3.1.

The function calls employed by EMBrad are illustrated in Figure 102 and each function together with arguments and return value is described in Section AII.3.2.

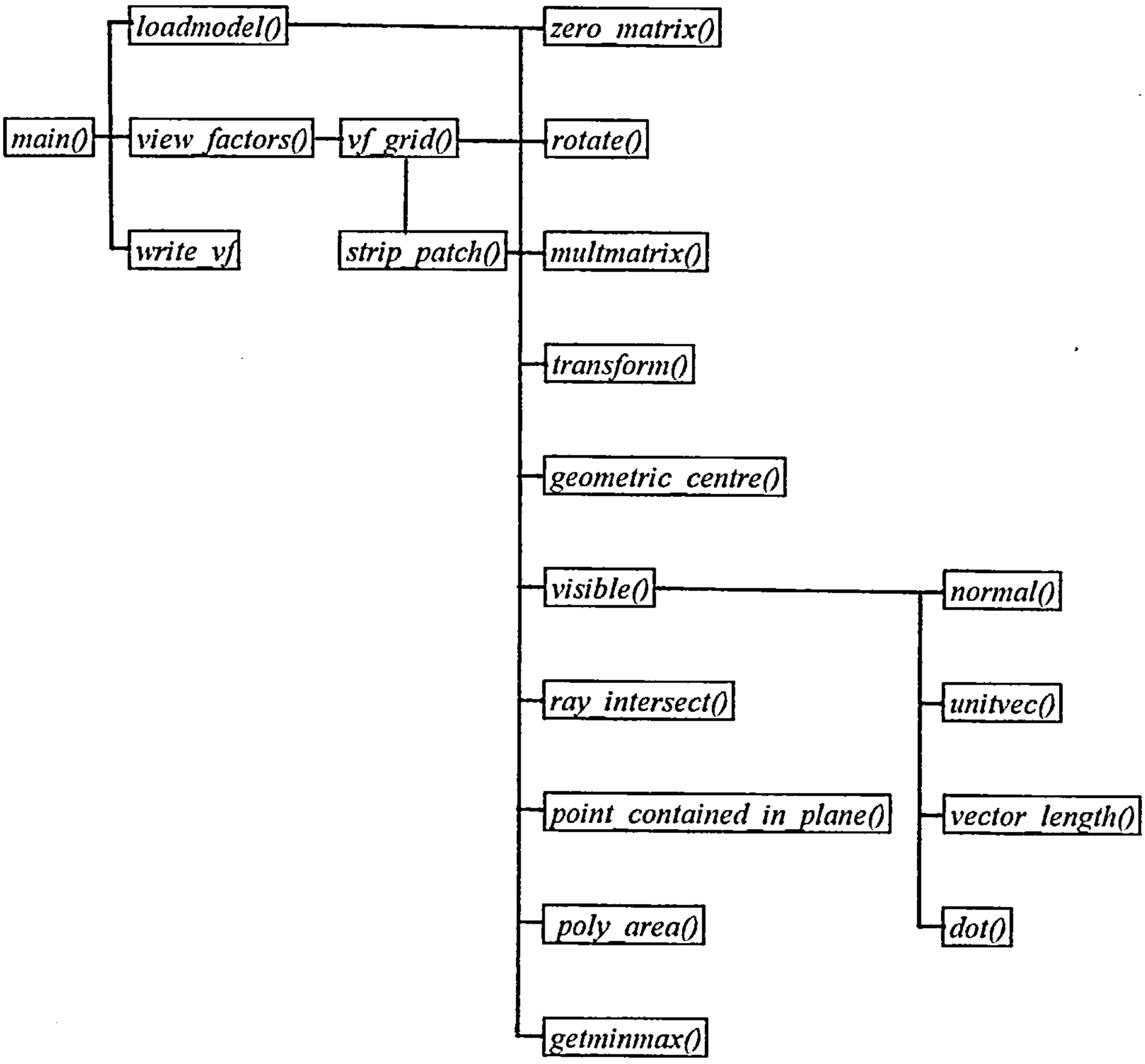


Figure 102 - Function calls made by the EMBrad *main()* function

AII.2.3 Problem Simulation - The EMBsim Program Module

The main calculation routines for problem solution are contained within the EMBsim module.

A number of data files must first be established prior to problem solution. The problem data files are illustrated in Figure 100 and summarised in Section AII.2.3.1 - AII.2.3.6.

The EMBsim module reads the external data files (problem definition and support files) and initialises the dependent variable residual log file.

The module may be run in dynamic or steady-state mode by setting the *smodel* switch in the *embsim.sim* simulation data file.

a) Dynamic Mode

The module establishes a dynamic loop, the limits of which are the defined start and stop times specified in the *embsim.sim* simulation data file. The loop is incremented by the time step *dt*, which is again specified in the *embsim.sim* data file. The *boundary()* and *simpler()* functions are then called for each time step from witin the dynamic loop to establish boundary conditions and solve the definig equation set, respectively. Calculation results are written to a series of *.dmp* files at each results time step specified in *embsim.sim*. Each new time step begins when convergence has been achieved in the the *simpler()* numerical solution scheme.

b) Steady-State Mode

The *boundary()* function is first called to establish surface boundary conditions and subsequently the *simpler()* function is called to solve the resulting problem equation set.

In both steady-state and dynamic modes, the dependent variable residuals are written to the residual log file *embsim.log*, as well as being displayed on the computer monitor, at the end of each of the solution scheme outer iterations.

The solution schem may be interrupted during the calculations using a CTRL-C interrupt, in which case the dependent variables are written to the results file *embsim.dmp* at the end of the current iteration and the EMBsim module execution is terminated.

The function calls employed by EMBsim are illustrated in Figures 103, 104 and 105. Each function together with arguments and return value is described in Sections AII.3.2. and AII.3.3.

AII.2.3.1 Binary Geometry File

The problem geometry is defined in an ASCII file and then transformed into a binary file using the EMBgen module (refer to Section AII.2.1).

AII.2.3.2 Boundary Condition File

Boundary conditions may be defined for the problem using an external weather data file and/or a boundary condition file. The boundary condition file is an ASCII file in which the value of dependent variables may be defined for ranges of grid points. The dependent variable values and grid parameter switches for the defined grid point range are read from the file using the *readboundary()* function. An example of a boundary condition file is included in Appendix III.

AII.2.3.3 Simulation Data File

The simulation data file contains the external data file names for the problem, convergence control criteria and various other items of problem specific data . The file is read into a simulation data structure. Refer to Section AII.1 for details of the simulation data structure.

AII.2.3.4 View Factor File

If radiation heat transfer is to be calculated for the defined problem, view factors must be specified for all participating building geometry polyhedron surface pairings. The view factors are specified in an ASCII text file which may be prepared using a text editor or generated automatically using the EMBrad module. View factors for each surface pairing are specified with the participating geometric body structure array indices and the contained surface structure array indices. An example of a view factor file is included in Appendix III.

AII.2.3.5 Materials Database

The materials database is an ASCII text file which contains a series of records of thermophysical properties for various materials. The file is read into a materials structure array using the *getmat()*

function. Refer to Section AII.1 for details of the materials structure. An example of a materials database is included in Appendix III.

AII.2.3.6 Weather File

When the EMBsim module is executed, the weather file specified in the embsim.sim simulation data file is read into a weather structure array (refer to Section AII.1). At each time step of the calculation, various items of weather data (external temperature, wind data and solar data) are obtained from the weather structure array using the current simulation month, day and hour as indices.

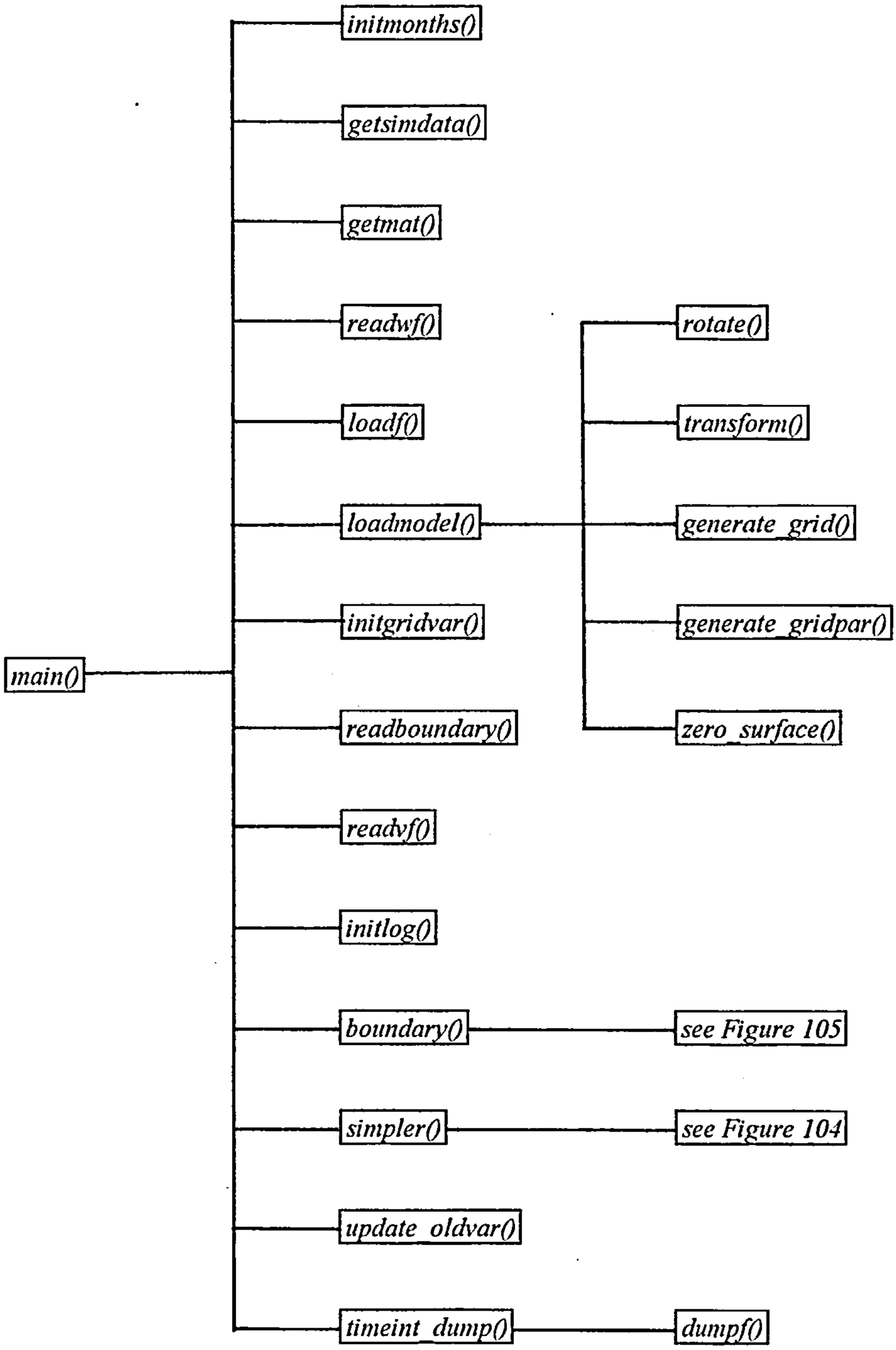


Figure 103 - Function calls made by the EMBsim *main()* function

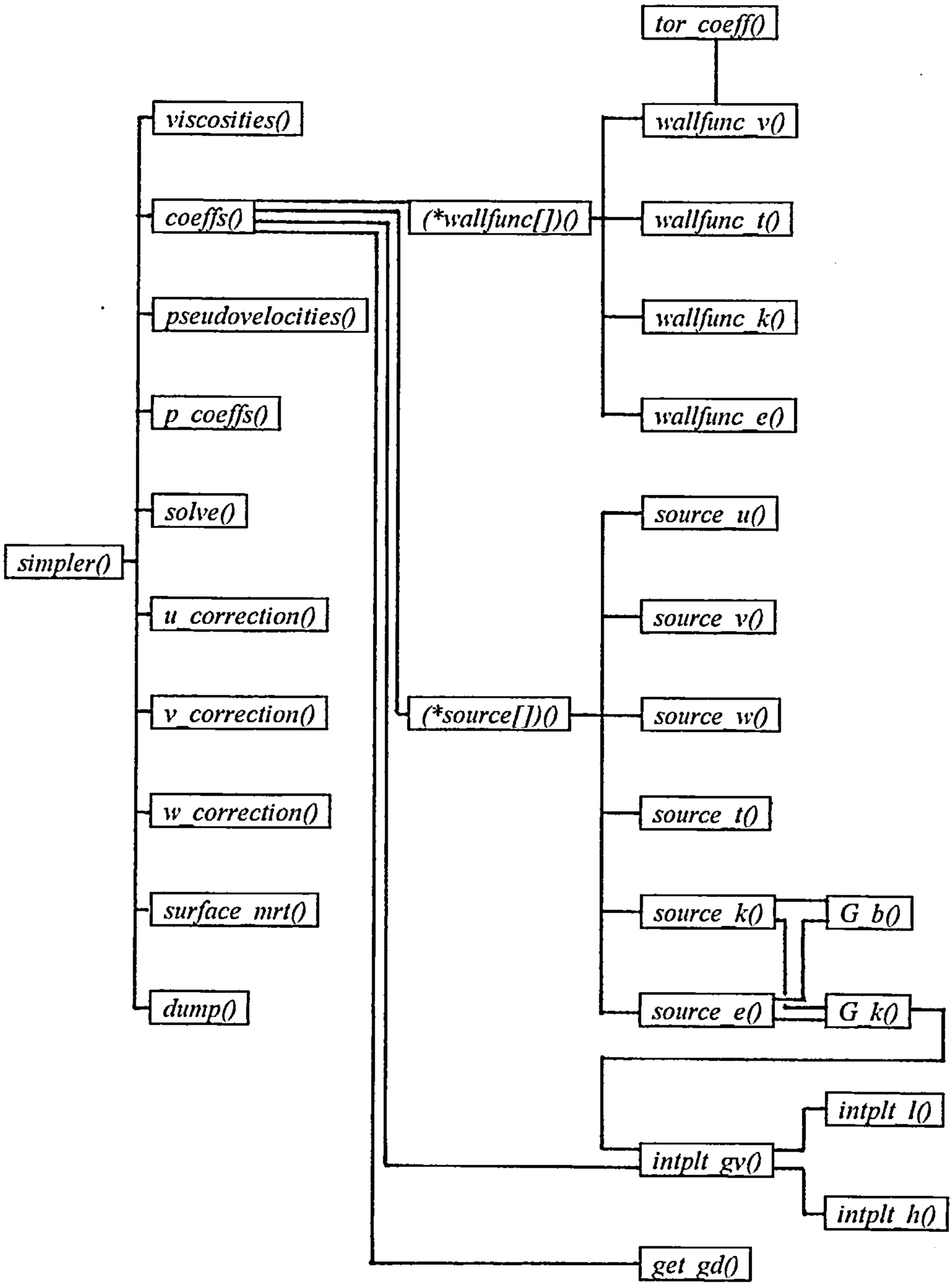


Figure 104 - Function calls made by *simpler()* function

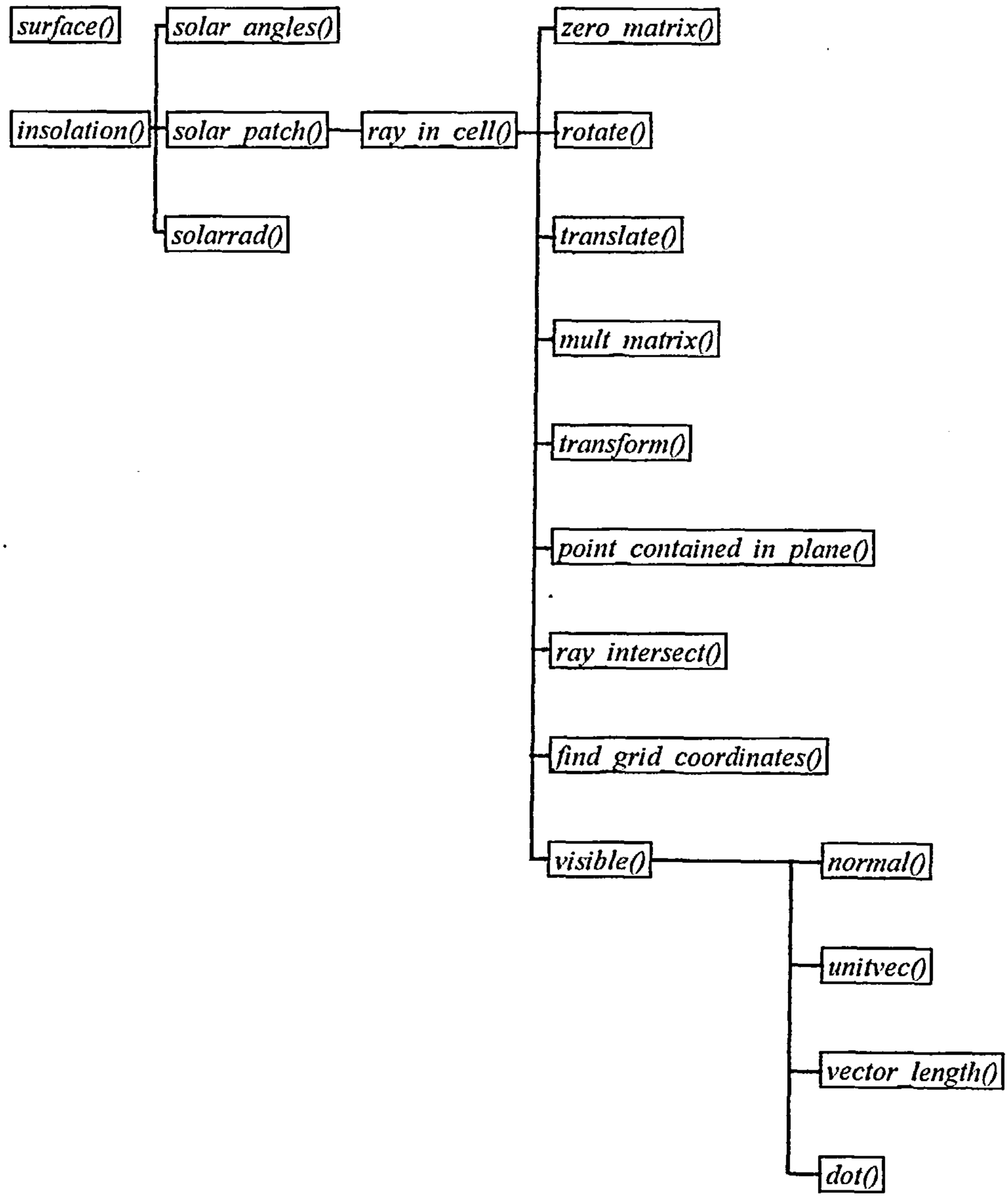


Figure 105 - Function calls made by the*boundary()* function

AII.2.4 Translation of Binary Results File to ASCII Format - The EMBtran Program Module

The EMBtran module is used to translate the binary results file embsim.dmp into an ASCII file embsim.asc. The dependent variables are written to the file as a series of x-y grid point tables for each grid division along the z-axis. A number of argument switches are available to enable the required variables to be extracted:-

/u	x-dimension velocities
/v	y-dimension velocities
/w	w-dimension velocities
/t	temperatures
/p	pressures
/k	turbulence kinetic energy
/e	dissipation rate of turbulence kinetic energy
/g	grid dimensions and polyhedra geometry

AII.2.5 Graphical Visualisation of Simulation Results - The EMBview Program Module

The EMBview module enables the three-dimensional building geometry, finite volume grid and the magnitudes of dependent variables to be viewed graphically on a PC hardware platform.

The module may also be used to inspect the three-dimensional geometry of the model and finite volume grid generated using EMBgen (see Section AII.2.1), prior to calculation.

Before executing the EMBview module, various display parameters may be defined using an external ASCII display data file which is read by the program into a display structure:

```
struct display_st {
char project[31], modelfile[15];
double dfactor, mintemp, maxtemp, maxvel;
double tz[MAXBAND+1], vz[MAXBAND+1];
int ntband, nvband, ftype;
```

};

An example of a display data file is included in Appendix III.

When the EMBview module is run, a three-dimensional image of the building geometry and defined contents (partitions, enclosures, obstacles, heat emitters, etc.) is displayed.

The building geometry is displayed with an axis frame which may be positioned to lie in the Y-Z, X-Z and X-Y planes by pressing ALT-X, ALT-Y or ALT-Z. The frame may then be moved along the selected axis to coincide with the center of a finite volume grid plane by pressing the UP or DOWN arrow keys. The entire displayed image (geometry and frame) may be rotated and scaled using the following keys:-

x - rotate clockwise around the X axis
X - rotate anti-clockwise around the X axis
y - rotate clockwise around the Y axis
Y - rotate anti-clockwise around the Y axis
z - rotate clockwise around the Z axis
Z - rotate anti-clockwise around the Z axis
i/I - zoom IN
o/O - zoom OUT

Having established the required orientation and scaling factor, dependent variable magnitudes may be displayed using the following keys:-

F1 - three-dimensional vector visualisation. This option may be used for flow visualisation purposes. A series of vectors are displayed indicating the direction of flow at the gid cell centre, the length of the vector being proportional to the magnitude.

F2 - two-dimensional solar irradiation display. This option enables solar irradiated cells to be highlighted within the selected plane.

F3 - two-dimensional vector display. This option is used to display velocity vectors across the plane selected using the axis frame. The length of each vector indicates the magnitudeof the velocity resolved in the selected plane. A velocity scale is displayed at the right hand side of the screen.

F4 - two-dimensional area fill temperature display. Each cell in the selected plane is shaded according to temperature. A temperature-shade key is displayed at the right hand side of the screen. The magnitude of the temperature increments are determined by the temperature range defined in the display.dat file, there being a maximum of ten increments.

F5 - two-dimensional finite grid display. This option is used to display the dimensions of the finite volume grid. A distance scale is displayed at the right hand side of the screen.

F6 - two-dimensional velocity contour display. The magnitude of each contour is defined in the display.dat file, each contour being drawn in a different colour. The display is overlaid with a magnitude grid to enable contours to be identified.

F7 - two-dimensional temperatue contour display. The magnitude of each contour is defined in the display.dat file, each contour being drawn in a different colour. The display is overlaid with a magnitude grid to enable contours to be identified.

The module may be exited by pressing the F10 key.

AII.2.5.1 Coordinate systems employed by Microsoft C graphics functions

A physical display monitor has a range of display points or pixels which depend upon the current video mode (display adaptor) e.g. 16 colour VGA on a PC has a resolution of 640 X 480 pixels. A particular area of the physical screen may be designated a viewport using the Microsoft *_setviewport* function and all subsequent graphics calls may then be directed to the resulting viewport.

A floating point coordinate system (i.e. based on physical units) may then be assigned to the current viewport using the Microsoft *_setwindow* function. This function enables a viewport to be scaled to the correct range for the required display objects. The displayed graphic object may the be manipulated using a system of units appropriate to the object.

EMBview employs an array of graphic window structures which incoroprate viewport coordinates and window coordinate ranges (i.e. ranges of physical units) for each display required e.g. three-dimensional geometry display, two-dimensional contour display, temperature scales, etc.

```
struct gwin_st {

    int vx1, vx2, vy1, vy2;
    double wx1, wx2, wy1, wy2;
    unsigned char wcoord;

};
```

A convention has been adopted whereby the origin of the window coordinate systems lie at the geometric centre of the viewports.

AII.2.5.2 Dependent variable display

With the exception of flow visualisation, the magnitudes of dependent variables are displayed in the form of vectors, contours or area fills in one of the three two-dimensional planes (Y-Z, X-Z and X-Y). For ease of manipulation each plane is transformed into a temporary display X-Y plane.

In order to map the three displayable planes (Y-Z, X-Z and X-Y) onto the temporary, X-Y plane, vector structures are defined for the three major axes:

```
struct vector_st {

    int dim, maxplane, xdim, ydim, maxxdiv, maxydiv, xcomp, ycomp;
    int zcomp;
    double lhd, xlhd, ylhd, minxdim, minydim;

};
```

where each field is defined as follows:

dim: defined axis constant for referencing grid dimensions COL, DIV or ROW from the *gd[3][MAXDIM]* array.

maxplane: number of divisions, rows or columns appropriate to the specific axis.

xdim:	defined axis constant to be mapped onto temporary display X axis.
ydim:	defined axis constant to be mapped onto temporary display Y axis.
maxxdiv :	number of divisions, rows or columns appropriate to the specific axis to be mapped onto the temporary number of X divisions.
maxydiv:	number of divisions, rows or columns appropriate to the specific axis to be mapped onto the temporary number of Y divisions.
xcomp:	axis to be mapped to temporary X axis for coordinate transformation.
ycomp:	axis to be mapped to temporary Y axis for coordinate transformation.
zcomp:	axis to be mapped to temporary Z axis for coordinate transformation.
lhd:	geometry dimension (length, height, depth) in current axis.
xlhd:	geometry dimension (length, height, depth) of axis to be mapped onto temporary X axis.
ylhd:	geometry dimension (length, height, depth) of axis to be mapped onto temporary Y axis.
minxdim:	minimum geometry vertex on axis to be mapped onto temporary X axis or coordinate transformations.
minydim:	minimum geometry vertex on axis to be mapped onto temporary Y axis or coordinate transformations.

The function calls employed by the EMBview module are illustrated in Figure 106 and each function together with arguments and return value is described in Section AII.3.4.5.

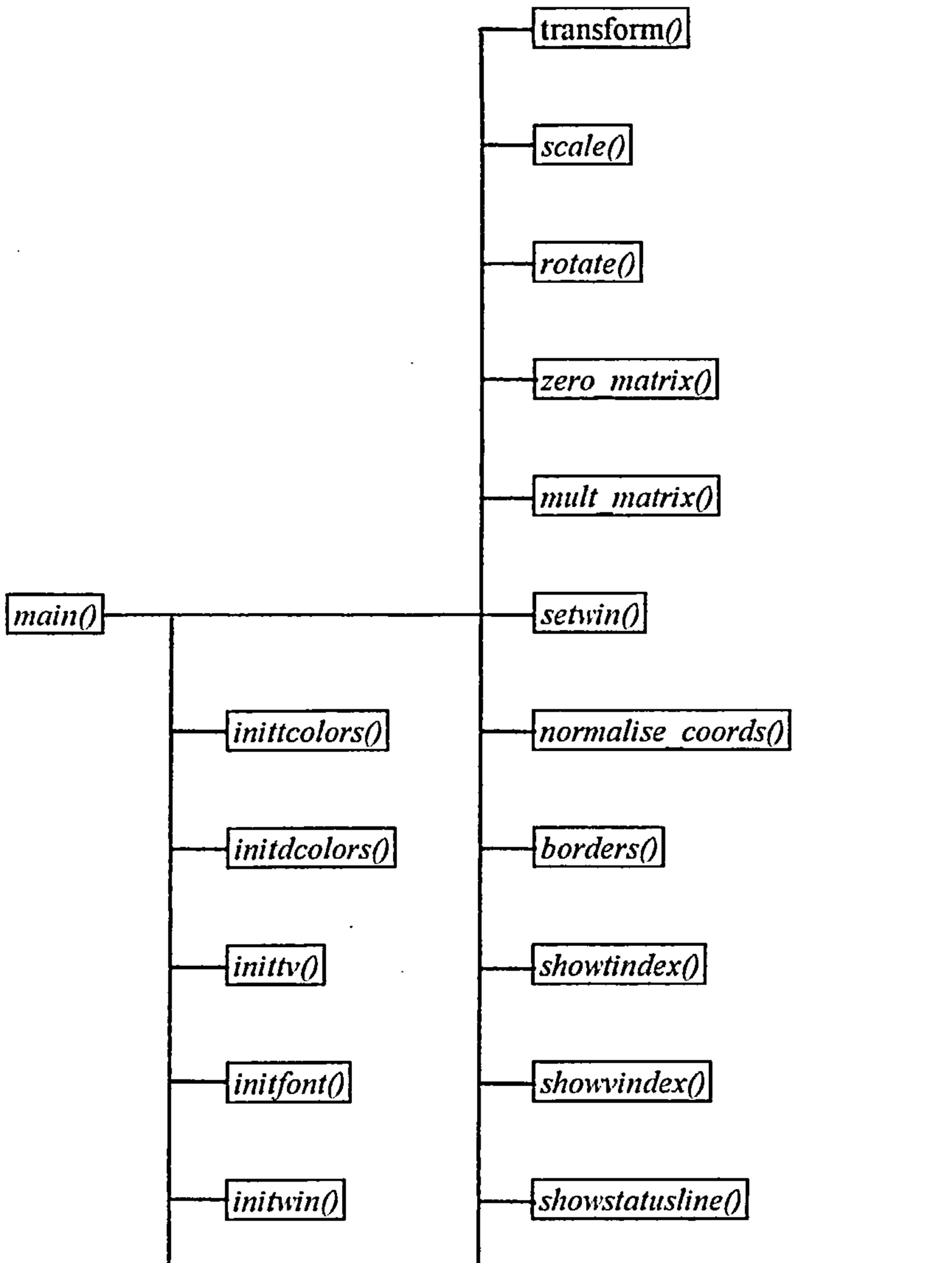
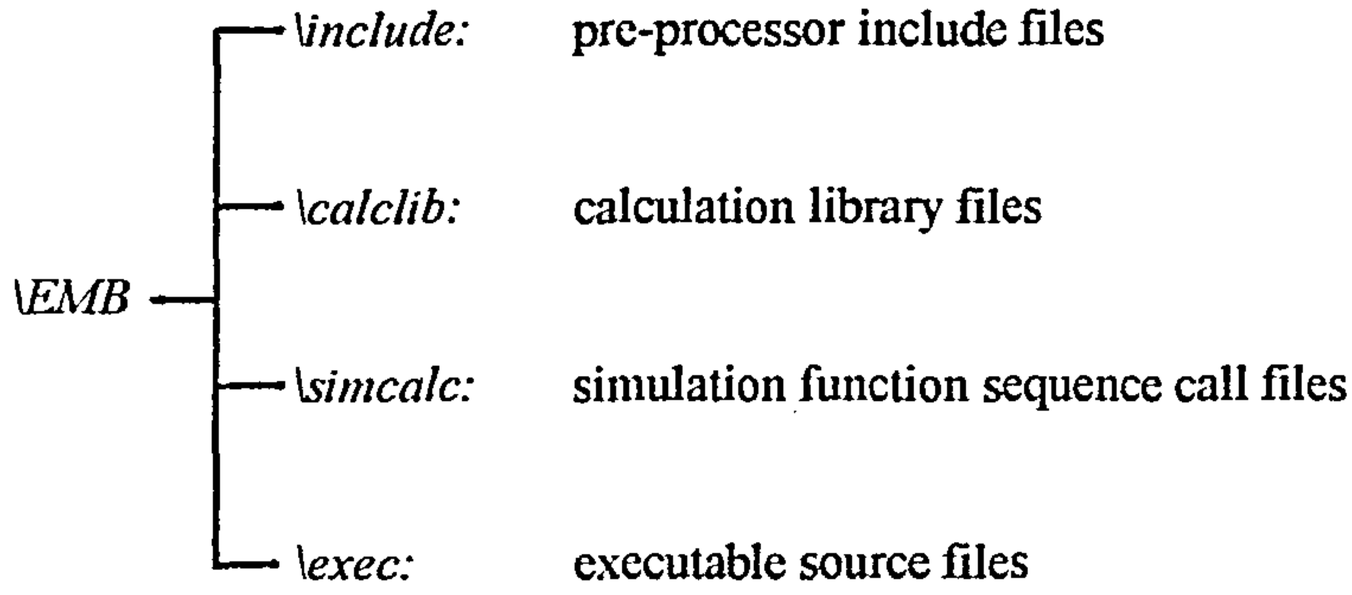


Figure 106 - Function calls made by the EMBview *main()* function

AII.3 Program Structure

The executable files are built from a group of calculation source files and data 'include' files. To facilitate program development, the source files are grouped into four sub-directories under the main EMB directory:



AII.3.1 Pre-processor include files; the *\include* sub-directory

Common data structures, constants and macros are defined in the include file *defs.h* which is made available to all source files using the C preprocessor *#include* directive. Similarly, external function declarations, constants and macros which are peculiar to specific function groups are defined in an include file having the same prefix file name as the function group source file, e.g. *geometry.h* is the include file for *geometry.c*. External data declarations (i.e. common data) are included in the *main.h* file.

AII.3.2 Calculation library files; the *\calclib* sub-directory

Calculation functions are grouped into a series of source files using logical associations, e.g. all interpolation functions are grouped into the *intplt.c* source file. The calculation source files are then compiled separately and archived into the form of an object library (using the *ar* utility under UNIX and the *lib* utility under MS-DOS Microsoft C). The calculation library may then be linked with each executable '*main*' object file, enabling common access to functions, e.g. the same geometrical transformation functions (rotations, etc.) are required for binary data file generation, radiation view factors and solar tracking and graphical results display. The *\calclib* source files and their associated functions, together with arguments, return values and function description are detailed in Sections AII.3.2.1 - AII.3.2.14.

AII.3.2.1 *GDE.C*

a) *void coeffs(var, var_type, divs, rows, cols)*

Arguments:

var (int):	Grid variable as defined in *DEFS.H*.
var_type (int):	Grid variable type as defined in *DEFS.H*. Type may be scalar (*SCALAR*), U-component of velocity (*U_COMPONENT*), V-component of velocity (*V_COMPONENT*) or W-component of velocity (*W_COMPONENT*).
divs (int):	Number of divisions in flow field.
rows (int):	Number of rows in flow field.
cols (int):	Number of columns in flow field.

Return value:

None.

Description:

coeffs() calculates the general differential equation grid point and finite volume interface coefficients in addition to the 'b' term (refer to Eq.3.58). Interpolations for interface velocities and diffusion coefficients are effected, where necessary, through calls to *intplt_gv()*. Source terms are obtained via calls to the appropriate source function, the correct function being selected by means of an array of pointers to source functions, the index of the array being the variable index *var*, which is passed into *coeffs()* from *simpler()*. The correct wall function is selected in a similar fashion, if the adjacent finite volume cells are static (i.e. if the *bfstatic* flag is set in the *gridpar_bf* bit field of the appropriate *gridpar* structure). In the case of the energy equation, the correct wall function is selected if either the current finite volume or an adjacent finite volume (but not both) is defined as *STATIC*, this relationship being established through application of an 'exclusive or' macro, which is defined in the*DEFS.H* file as *xor()*.

b) *double f(Pe)*

Arguments:

Pe (double): Peclet number.

Return Value:

Diffusion coefficient factor.

Description:

Used to derive the combined convection-diffusion coefficient for alternative discretisation schemes (see Section3.2.4).

AII.3.2.2 *PRESSURE.C*

a) *double p_coeffs(uvar, vvar, wvar)*

Arguments:

uvar (int): U-component variable, either pseudo-velocity (U) or guessed velocity (U), as defined in *DEFS.H* as *U_PSVEL* and *U_STAR*, respectively.

vvar (int): V-component variable, either pseudo-velocity (V) or guessed velocity (V), as defined in *DEFS.H* as *V_PSVEL* and *V_STAR*, respectively.

wvar (int): W-component variable, either pseudo-velocity (W) or guessed velocity (W), as defined in *DEFS.H* as *W_PSVEL* and *W_STAR*, respectively.

Return value:

The maximum absolute value of finite volume mass residual - the 'b' term of the pressure correction equation (see Eq.3.69).

Description:

p_coeffs() calculates the grid point and finite volume interface coefficients for the pressure and pressure correction equations (Equations 3.69 and 3.73 , respectively). *uvar*, *vvar* and *wvar* are passed to the function in order that the correct 'b' term may be calculated, i.e. *U_STAR*, *V_STAR* and *W_STAR* for the pressure correction equation, and *U_PSVEL*, *V_PSVEL* and *W_PSVEL* for the pressure equation.

AII.3.2.3 *FILE.C*

a) *int readmodel(fn, ncn, cn)*

Arguments:

fn (char *):	ASCII geometry model file name.
ncn (int *):	number of surface-surface connections.
cn (struct connection_st *):	array of connection structures.

Return value

None.

Description:

The *readmodel()* function is used to read an ASCII problem geometry data file from disk in a defined sequence and to assign the data values to the appropriate data structures. The data comprises finite volume grid dimensions, site latitude and orientation, building space polyhedra data sets and a list of surface-surface connection information, an example of an ASCII geometry file is included in Appendix III.

The *readmodel()* function calls the geometric *rotate()* function in order to generate a site orientation rotation matrix which is used in conjunction with the *transform()* function to convert the building space polyhedra vertex coordinates from a local coordinate system to a 'real-world' coordinate system (see Section 4.1.2).

Prior to reading each polyhedron surface data set, system memory is dynamically allocated for

a surface structure, using the *addsurface* macro which is defined in *DEFS.H*:-

*#define addsurface ((struct surface *)malloc(sizeof(struct surface)))*

The allocated surface structure is then assigned to the appropriate element of the surface structure pointer array, defined within the current geometric body structure array element, e.g.:-

if((bd[i].sf[j]=addsurface) == NULL) leave("Insufficient memory for surfaces");

Each set of surface-surface connection data is read into a *connection_st* structure, as defined in Section AII.1.

b) *int dumpmodel(fn)*

Arguments:

fn (char *): binary geometry model file name.

Return value

1 if the file is written successfully, otherwise 0.

Description:

dumpmodel() opens the disk file *fn*, for writing, the variables *ndivs, ncols, nrows, nbodies, orient* and *latitude* are then written to disk in the form of a header structure. The *gridpar[][][]* array is subsequently written to disk as a byte stream. Each element of the geometric body structure array *bd[]* is then written to disk, the dynamically allocated surface structures being written sequentially following the associated body structure.

The *dumpmodel()* function calls the geometric *rotate()* function in order to generate a site orientation rotation matrix which is used in conjunction with the *transform()* function to convert the building space polyhedra vertex coordinates from the 'real-world' coordinate system to a local coordinate system (see Section 4.1.2).

c) *int loadmodel(fn)*

Arguments:

fn (char *): binary geometry model file name.

Return value

1 if the file is read successfully, otherwise 0.

Description:

The file *fn* is opened for reading and the variables *ic, ndivs, nrows, ncols, nbodies, orient, latitude*are read via the file header structure, the finite volume grid parameter array *gridpar[][][][* is subsequently read in the form of a byte stream. Each element of the geometric body structure array *bd[]* is then read, the surface structures associated with each element are read subsequent to memory being dynamically allocated and assigned to the appropriate surface structure pointer array element.

The *loadmodel()* function calls the geometric *rotate()* function in order to generate a site orientation rotation matrix which is used in conjunction with the *transform()* function to convert the building space polyhedra vertex coordinates from a local coordinate system to a 'real-world' coordinate system (see Section 4.1.2).

d) *int dumpf(fn, status, month, day, hour, sec)*

Arguments:

fn (char *): dump file name.
status (int): flag to exit program or return to calling function.
month (int): simulation month.
day (int): simulation day.
hour (int): simulation hour.
sec (int): simulation second.

Return value:

1 if the file is written successfully, otherwise 0.

Description:

When the simulation is run in steady-state mode, *dump()* is called automatically on termination of the program (either through a CTRL-C interrupt or when the dependent variable residuals have diminished to the defined tolerances. The disk file *EMBSIM.DMP* is opened for writing and the variables *ic* (iteration count), *ndivs, nrows, ncols, nbodies, orient, latitude* and the simulation time (*cmonth, cday, chour* and *csec*) are written to the file in the form of a header structure, the main finite volume grid arrays *grid[][][][]* and *gridpar[][][][]* are subsequently written to the file in the form of byte streams. Each element of the geometric body structure array *bd[]* is then written to disk, the dynamically allocated surface structures being written sequentially following the associated body structure. The *EMBSIM.DMP* file is subsequently closed and the program exited.

The *dumpf()* function calls the geometric *rotate()* function in order to generate a site orientation rotation matrix which is used in conjunction with the *transform()* function to convert the building space polyhedra vertex coordinates from the 'real-world' coordinate system to a local coordinate system (see Section 4.1.2).

When run in dynamic mode, the *dump()* function is called by the EMBSIM module at the end of a results output interval, the duration of which is defined in the *EMBSIM.SIM* file. Each interval results file is prefixed with *EMB*, followed by the interval serial number (e.g. *EMB0001.DMP*, *EMB0002.DMP*, etc.).

e) *int loadf(fn)*

Arguments:

fn (char *): binary dump file name.

Return value:

1 if the file is read successfully, otherwise 0.

Description:

The file *fn* is opened for reading and the variables *ic*, *ndivs*, *nrows*, *ncols*, *nbodies*, *orient*, *latitude* and the simulation time (*cmonth*, *cday*, *chour* and *csec*) are read via the file header structure, the main finite volume grid arrays *grid[][][][][]* and *gridpar[][][][][* are subsequently read in the form of byte streams. Each element of the geometric body structure array *bd[]* is then read, the surface structures associated with each element are read subsequent to memory being dynamically allocated and assigned to the appropriate surface structure pointer array element.

The *loadf()* function calls the geometric *rotate()* function in order to generate a site orientation rotation matrix which is used in conjunction with the *transform()* function to convert the building space polyhedra vertex coordinates from a local coordinate system to a 'real-world' coordinate system (see Section 4.1.2).

f) *int readboundary(fn)*

Arguments:

fn (char *): boundary condition data file name.

Return value:

1 if the file is read successfully, otherwise 0.

Description:

The ASCII boundary condition data file *fn* (specified in *EMBSIM.SIM* and read into *simdata.bcfile*) is read from disk. Boundary conditions that may be specified comprise velocity components, temperatures, casual heat fluxes, static cells, turbulence kinetic energy and the dissipation rate of turbulence kinetic energy. The boundary condition information is entered in the data file in the form of rows of data specifying blocks of finite volume cells (in terms of starting and finishing X, Y and Z cell numbers), the appropriate variable index and the value of the variable to be specified. The format of the boundary condition data file is presented in Appendix III.

g) *int readwf(fn)*

Arguments:

fn (char *): ASCII weather data file name (**.asc*).

Return value:

1 if the file is read successfully, otherwise 0.

Description:

Reads each daily record of the *.asc* weather file into an array of *WeatherData_St* structures (see Section AII.1).

h) *int getmat(fn)*

Arguments:

fn (char *): ASCII materials database file name (*embsim.mat*).

Return value:

1 if the file is read successfully, otherwise 0.

Description:

Reads each record of the *embsim.mat* materials database file into an array of *material_st* structures (see Section AII.1).

i) *int getsimdata()*

Arguments:

None.

Return value:

1 if the file has been read successfully, otherwise 0.

Description:

getsimdata() reads various items of data required by the EMBSIM module into a *simdata_st* structure, as defined in Section AII.1.

5.3.2.4 *UTIL.C*

a) *void initgridvar()*

Arguments:

None.

Return value:

None.

Description:

Initialises grid point variables which are not set by the problem data file. Initial values are as follows:-

U-component of velocity:	0.1 ms^{-1}
V-component of velocity:	0.0 ms^{-1}
W-component of velocity:	0.0 ms^{-1}
Temperature:	20 C
Pressure:	0.0 Pa
Pressure correction:	0.0 Pa
Turbulence kinetic energy:	0.01
Dissipation of turbulence kinetic energy:	$C_\mu k^{1.5} / 0.05$

b) *void viscosities()*

Arguments:

None.

Return value:

None.

Description:

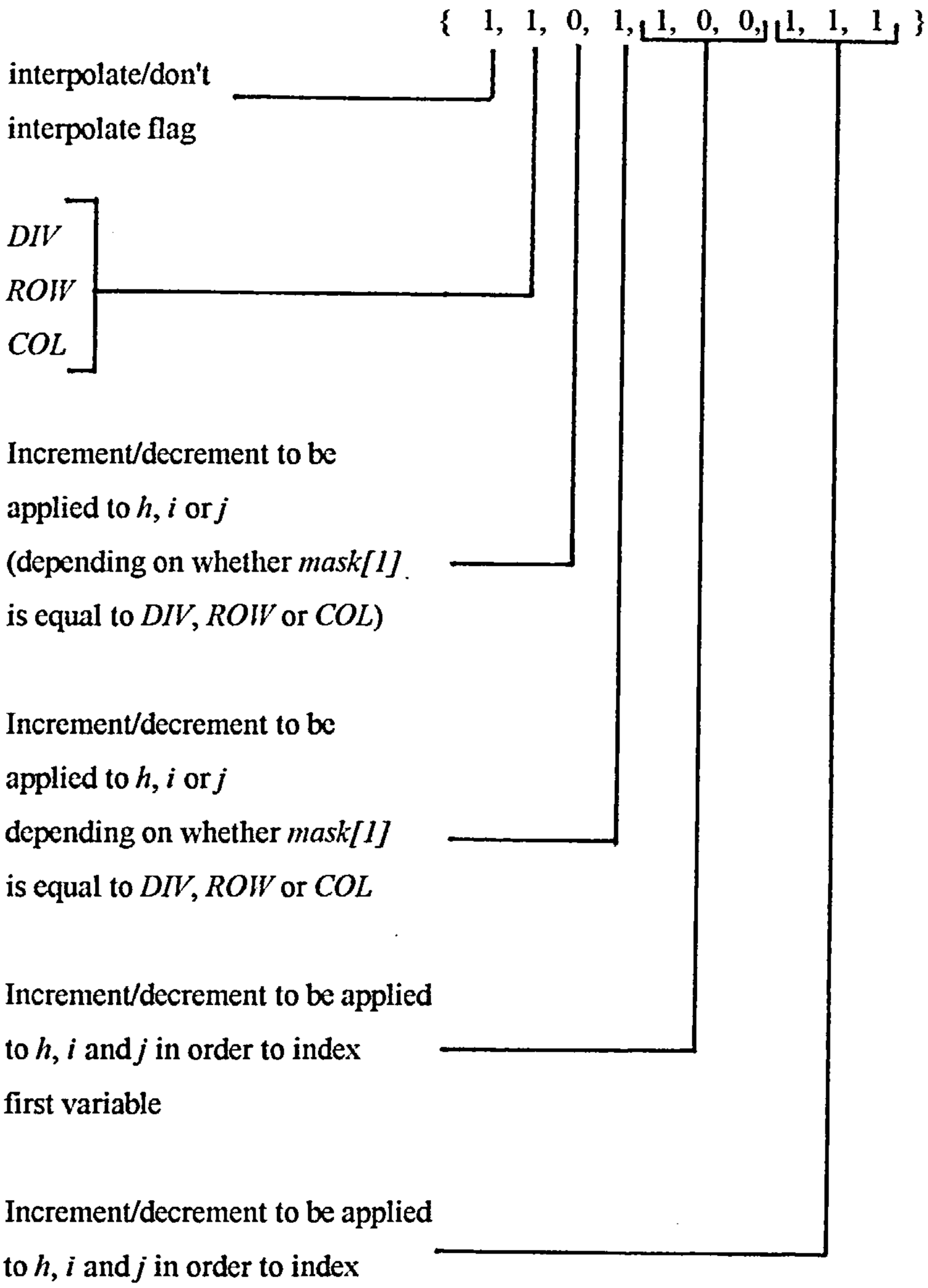
This function calculates the turbulent viscosities for each scalar grid point throughout the flow field, using the turbulence kinetic energy and dissipation of turbulence kinetic energy quantities (Equation 3.32).

5.3.2.5 *INTPLT.C*

a) *double intplt_gv(intplt,gv,h,i,j,v1,v2,mask)*

Arguments:

intplt (double (*)()):	pointer to appropriate interpolation function - linear or harmonic.
gv (int):	grid variable to be interpolated, as defined in *DEFS.H.*
h (int):	division.
i (int):	row.
j (int):	column.
v1 (double):	When an interpolation is required between two previously derived values, *v1* represents the first value.
v2 (double):	When an interpolation is required between two previously derived values, *v2* represents the second value.
mask (char *):	a ten element array of signed bytes, each byte being defined as follows:-



Return value:

Result of interpolation.

Description:

intplt_gv() is used by *coeffs()* and *G_k()* in order to derive finite volume interface velocities and diffusion coefficients through the application of arithmetic and harmonic interpolations, respectively (refer to Section 3.2.10). Arrays of interpolation masks are defined at the beginning of functions *coeffs()* and *G_k()*, the first element of the mask indicates whether or

not an interpolation is required for the current variable, the second element indicates the dimension across which the interpolation is required, the third element is the required increment or decrement to be applied to *h*, *i* or *j* depending on the interpolation dimension, the fourth element is applied in the same way as the third element, the third and fourth elements being used to calculate the interpolation factor f (see Section 3.2.10), the fifth, sixth and seventh elements are the required increments or decrements to be applied to *h*, *i* and *j* respectively, in order to derive the correct index for the first interpolation variable value, similarly the eighth, ninth and tenth elements are used in order to index the second variable value. At the beginning of the function *coeffs()*, these masks are set up in the form of 4x10 two-dimensional arrays for each variable to be interpolated, each array being a set of four ten-element masks for the variable types defined in *DEFS.H* (i.e. *U_COMPONENT*, V_*COMPONENT*, *W_COMPONENT* and *SCALAR*), the first dimension of the array being indexed via the defined constants for the variable types. If the first increment/decrement field for each variable is set as *PASSED* (defined in *DEFS.H*), *intplt_gv()* will conduct an interpolation between *v1* and *v2* rather than deriving the interpolation variables, thus enabling an interpolation between previous results of interpolations which is required by the function *G_k()* in the determination of velocity gradients.

b) *double intplt_l(f,v1,v2)*

Arguments:

f (double):	interpolation factor.
v1 (double):	first interpolation variable.
v2 (double):	second interpolation variable.

Return value:

Result of interpolation.

Description:

Conducts a linear interpolation between *v1* and *v2* via the interpolation factor *f* (see Section 3.2.10). *intplt_l()* is called by *intplt_gv()* for the determination of finite volume interface velocities.

c) *double intplt_h(f,v1,v2)*

Arguments:

f (double):	interpolation factor.
v1 (double):	first interpolation variable.
v2 (double):	second interpolation variable.

Return value:

Result of interpolation.

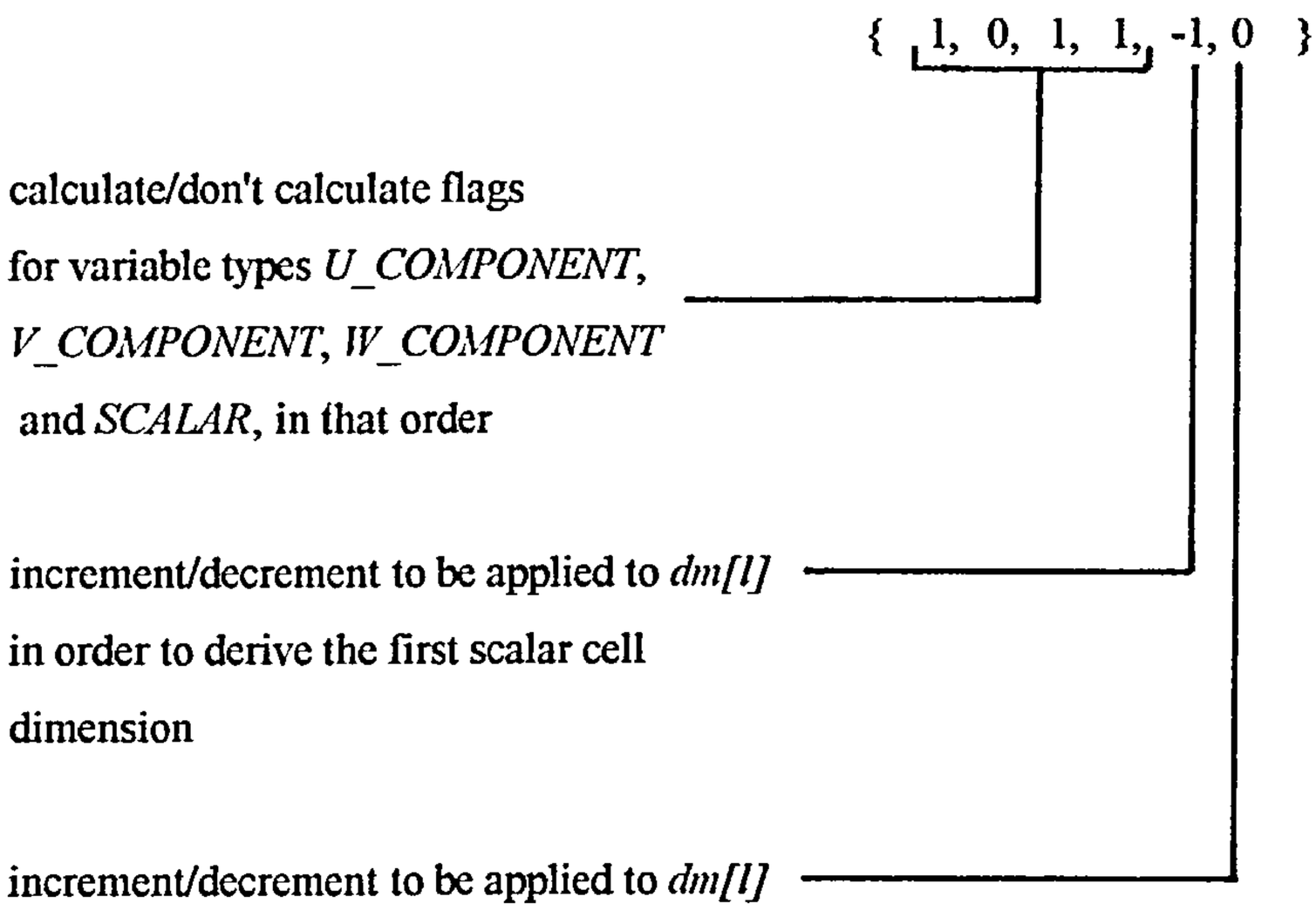
Description:

Conducts a harmonic interpolation between *v1* and *v2* via the interpolation factor *f* (see Section 3.2.10). *intplt_l()* is called by *intplt_gv()* for the determination of finite volume interface diffusion coefficients.

d) *double get_gd(dm,l,mask)*

Arguments:

dm (double *):	sub-array of the grid dimension array *gd[][]*, either *gd[DIV]*, *gd[ROW]* or *gd[COL]*.
l (double):	division, row or column.
mask (char *):	a six-element array of signed bytes where each byte is defined as follows:-



Return value:

Calculated grid dimension

Description:

This function calculates the grid dimensions for all variable types (velocity components and scalar variables), *gdyn*, *gdys*, *gdxe*, *gdxw*, *gdzt* and *gdzb*. The calculation is conducted simply by taking the arithmetic mean of the two scalar cell dimensions, due to the location of the finite volume grid points at the geometric centre of the finite volumes.

AII.3.2.6 *SOURCE.C*

a) *double G_k(h, i, j)*

Arguments:

h (int):	division.
i (int):	row.
j (int):	column.

Return value:

The G_k source element employed in the turbulence kinetic energy and dissipation of turbulence kinetic energy equations.

Description:

G_k0 determines all of the required finite volume interface velocity components in order to establish velocity gradients across the current cell, through calls to *intplt_gv()*. The G_k source term is then calculated from finite difference approximations for the velocity gradients, in conjunction with the appropriate turbulent viscosity.

b) *double G_b(h, i, j)*

Arguments:

- *h* (int): division.
  i (int): row.
  j (int): column.

Return value:

The G_b source element employed in the turbulence kinetic energy and dissipation of turbulence kinetic energy equations.

Description:

G_b() establishes the vertical temperature gradient across the cell and subsequently calculates the G_b term using the mean field thermal expansion coefficient *beta* (refer to Section 3.1.2).

c)-h) *void source_u(Sp, Sc, e, h, i, j)*

void source_v(Sp, Sc, e, h, i, j)

void source_w(Sp, Sc, e, h, i, j)

void source_t(Sp, Sc, e, h, i, j)

void source_k(Sp, Sc, e, h, i, j)

void source_e(Sp, Sc, e, h, i, j)

Arguments:

Sp (double *):	pointer to the grid point variable coefficient element of the source term.
Sc (double *):	pointer to the element of the source term excluded from the grid point variable coefficient.
e (double):	average dissipation of turbulence kinetic energy, required for the calculation of the turbulence kinetic energy source term when the current cell is adjacent to a static cell.
h (int):	division.
i (int):	row.
j (int):	column.

Return value:

None.

Description:

These functions calculate the source terms required by the appropriate differential equations. The source terms are returned to *coeffs()* via pointers to *Sp* and *Sc*.

AII.3.2.7 *PSVEL.C*

a) *void pseudovelocities(var, psvel, coeff, divs, rows, cols)*

Arguments:

var (int):	velocity component as defined in *DEFS.H* (*U_VEL*, *V_VEL* and *W_VEL*).
psvel (int):	'pseudo-velocity' component as defined in *DEFS.H* (*U_PSVEL*, *V_PSVEL* and *W_PSVEL*).
coeff (int):	momentum equation grid point velocity coefficient as defined in *DEFS.H* (*U_COEFF*, *V_COEFF* and *W_COEFF*).
divs (int):	number of divisions in flow field.
rows (int):	number of rows in flow field.
cols (int):	number of columns in flow field.

Return value:

None.

Description:

This function calculates 'pseudo-velocities' which are subsequently employed by the pressure equation (see Section 3.2.8).During calculation of the pseudo-velocities, the grid point coefficients are stored in the *grid[][][][]* sub-array determined by *coeff*, these coefficients are subsequently used in the calculation of pressure coefficients, pressure correction coefficients and velocity corrections.

AII.3.2.8 *VCORR.C*

a)-c) *void u_correction()*

void v_correction()

void w_correction()

Arguments:

None.

Return value:

None.

Description:

These functions calculate the U, V and W velocity components as defined in *DEFS.H* (*U_VEL*, *V_VEL* and *W_VEL*), using the velocity correction formulae detailed in Section 3.2.7.

AII.3.2.9 *SOLVE.C*

a) *double solve(var, divs, rows, cols)*

Arguments:

var (int):	variable equation to be solved.
divs (int):	number of divisions in flow field.
rows (int):	number of rows in flow field.
cols (int):	number of columns in flow field.

Return value:

Normalised maximum absolute residual (see Section 3.2.11).

Description:

This function solves the discretised equation set for the variable defined by *var*, using the coefficients and source terms previously calculated by *coeffs()* and *p_coeffs()*. The method employed may be either the SOR or the line-by-line methods, detailed in Section 3.2.9.

AII.3.2.10 *WALLFN.C*

a) *doubl tor_coeff(k, y)*

Arguments:

k (double):	turbulence kinetic energy.
y (double):	distance from wall or static cell.

Return value:

Velocity coefficient for wall shear stress.

Description:

tor_coeff() calculates the velocity coefficient for the wall shear stress, which is employed by the velocity component wall functions and the turbulence kinetic energy and dissipation of turbulence kinetic energy source term function *G_k()*.

b)-e) *double wallfunc_v(h, i, j, ha, ia, ja, wfe, Sp, Sc, wfk, yw, dx, face)*
double wallfunc_t(h, i, j, ha, ia, ja, wfe, Sp, Sc, wfk, yw, dx, face)
double wallfunc_k(h, i, j, ha, ia, ja, wfe, Sp, Sc, wfk, yw, dx, face)
double wallfunc_e(h, i, j, ha, ia, ja, wfe, Sp, Sc, wfk, yw, dx, face)

Arguments:

h (int):	division.
i (int):	row.
j (int):	column.
ha (int):	adjacent cell division.

ia (int): adjacent cell row.

ja (int): adjacent cell column.

wfe (double *): pointer to average dissipation of turbulence kinetic energy variable, this term is set by the *wallfunc_k()* function and subsequently passed into the *source_k()* function.

Sp (double *): pointer to grid point variable coefficient of source term.

Sc (double *): pointer to element of source term excluded from grid point variable coefficient.

wfk (double): turbulence kinetic energy.

yw (double): distance from wall.

dx (double): dimension of cell perpendicular to wall.

face (int): orientation of cell face (NORTH, SOUTH, EAST, WEST, TOP or BOTTOM).

Return value:

Differential equation finite volume wall interface coefficient.

Description:

These functions are used to calculate shear stresses, heat fluxes and turbulence quantities in near-wall regions (see Section 3.3).

AII.3.2.11 *GEOMETRY.C*

a) *int ray_intersect(sf, sc, fc, ic)*

Arguments

sf (struct surface *): pointer to structure associated with surface with which the intersection is to be calculated.

sc (double *): xyz coordinates of vector start point.

fc (double *): xyz coordinates of vector end point.

ic (double *): computed xyz coordinates of vector/surfcae intersection point.

Return Value

1 if an intersection occurs, otherwise 0.

Description

Determines whether or not a vector and plane intersect and if so calculates the xyz coordintes of the intersection point, using the algorithm detailed in Appendix I.

b) *void geometric_centre(nv, tvt, cg)*

Arguments

nv (int):	number of polgon vertices.
tvt (double (*)[4])	temporary array of xyz vertex coordinates.
cg (double *)	xyz coordinates of the computed geometric centre of the polygon.

Return Value

None.

Description

Calculates the geometric centre of a concave polygon, simply by taking the midpoint of the maximum and minimum coordinates of the X, Y and Z dimensions. The geometric centre coordinates are returned in the *cg* array.

c) *void zero_matrix(m)*

Arguments

m (double (*)[4]):	4x4 matrix to be zeroed.

Return Value

None.

Description

Assigns zero values to each element of a 4x4 matrix, prior to subsequent assignments.

d) *void translate(Xmv, Ymv, Zmv, translation_matrix)*

Arguments

Xmv (double): distance point is to be translated along the X axis.
Ymv (double): distance point is to be translated along the Y axis.
Zmv (double): distance point is to be translated along the Z axis.
translation_matrix (double (*)[4]): 4x4 point transformation matrix for translation.

Return Value

None.

Description

Constructs a translation matrix as described in Appendix I.

e) *void rotate(axis, angle, rotation_matrix)*

Arguments

axis (int): principal axis around which point is to be rotated.
angle (double): angle through which point is to be rotated.
rotation_matrix (double (*)[4]): 4x4 point transformation matrix for rotation.

Return Value

None.

Description

Constructs a rotation matrix as described in Appendix I.

f) *void scale(Xsc, Ysc, Zsc, scale_matrix)*

Arguments

Xsc (double):	ratio by which X coordinates are to be scaled.
Ysc (double):	ratio by which Y coordinates are to be scaled.
Zsc (double):	ratio by which Z coordinates are to be scaled.
scale_matrix (double (*)[4]):	4x4 point transformation matrix for scaling.

Return Value

None.

Description

Constructs a scaling matrix as described inAppendix I.

g) *void mult_matrix(m2, m1)*

Arguments

m2, m1 (double (*)[4]):	4x4 matrices to be multiplied together.

Return Value

None.

Description

Conducts a 4x4/4x4 matrix multplication and returns the result in the m2 matrix. The mult_matrix() function is used to construct a concatenated transformation matrix when

sequential rotation, translation or scaling operations are required.

h) *void transform(pt, mat, tpt)*

Arguments

pt (double *): xyz coordinates of point to be transformed.
mat (double (*)[4]): transformation matrix.
tpt (double *): xyz coordinates of transformed point.

Return Value

None.

Description

Transforms the three-dimensional coordinates of a point *pt[]*, using the transformation matrix *mat[]* by performing a 1x4/4x4 matrix multplication. The result of the multiplication is returned in *tpt[]*.

i) *void unitvec(cs, ce, cu)*

Arguments

cs (double *): xyz coordinates of vector start point.
ce (double *): xyz coordinates of vector end point.
cu (double *): xyz coordinates of unit vector end point.

Return Value

None.

Description

Calculates a unit vector with start point *cs[]*, in the direction *ce[]*. The coordinates of the end point of the unit vector are returned in *cu[]*.

i) *double dot(c1, c2, c3)*

Arguments

c1 (double *):	xyz coordinates of first vector start point.
c2 (double *):	xyz coordinates of common end point for both vectors.
c3 (double *):	xyz coordinates of second vector start point.

Return Value

The vector dot product.

Description

Calculates the vector dot product (scalar product) of two vectors sharing a common point:-

$$\mathbf{a}\cdot\mathbf{b} = [a_1 \;\; a_2 \; a_3]\begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

j) *void normal(c1, c2, c3, nc)*

Arguments

c1 (double *):	xyz coordinates of first vector start point.
c2 (double *):	xyz coordinates of common end point for both vectors.
c3 (double *):	xyz coordinates of second vector start point.
nc (double *):	xyz coordinates of normal vector end point.

Return Value

None.

Description

Calculates the end point coordinates of a unit normal, starting at a common point between

two vectors, by calculating the vector cross product between the two vectors, see Appendix I. The cross product is passed to the *unitvec()* function in order to derive the unit normal.

k) *double vector_length(c1, c2)*

Arguments

c1 (double *): xyz coordinates of vector start point.

c2 (double *): xyz coordinates of vector end point.

Return Value

Length of vector.

Description

Calculates the length of a vector, see Appendix I.

l) *int visible(sc, vp, csn)*

Arguments

sc (double *): xyz coordinates of point to be checked for visibility.

vp (double *): xyz coordinates of view point.

csn (double *): pointer to value of cosine of angle between the view point-viewed point vector and the outward normal vector to the viewed point.

Return Value

1 if point is visible, otherwise 0.

Description

Determines visibility of one point from another and if visibility is confirmed, computes the angle cosine between the viewed point-view point vector and an outward normal to the viewed

point, using the algorithm described in Appendix I.

m) *void plane(c1, c2, c3, A, B, C, D)*

Arguments

c1, c2, c3 (double *): xyz coordinates of first three non-collinear clockwise vertices of surface.

A, B, C, D (double *): pointers to values of coefficients of plane in which surface lies.

Return Value

None.

Description

Calculates coefficients of surface plane equation, using the algorithm detailed in Appendix I.

n) *double poly_area(nv, tvt)*

Arguments

nv (int): number of polygon vertices.

tvt (double (*)[4]): temporary array of xyz coordinates of polygon vertices.

Return Value

Area of polygon.

Description

Calculates the area of a concave or convex polygon, using the algebraic trapezoidal summation algorithm detailed in Appendix I.

o) *int point_contained_in_plane(nv, tvt, pt)*

Arguments

nv (int):	number of polygon vertices.
tvt (double (*)[4]):	temporary array of xyz coordinates of polygon vertices.
pt (double *):	xyz coordinates of point for which containment is to be tested.

Return Value

1 if point is contained, otherwise 0.

Description

Determines whether or not a point is contained within a concave or convex polygon, using the algorithm detailed in Appendix I.

5.3.2.12 *SOLAR.C*

a) *void solar_angles(lat, day, hour, ornt, phi, beta, gamma, theta)*

Arguments

lat (double):	latitude.
day (int):	day of year.
hour (double):	hour of day.
ornt (double):	surface orientation angle.
phi (double):	surface tilt angle.
beta (double *):	pointer to computed value of solar altitude.
gamma (double *):	pointer to computed value of solar azimuth.
theta (double *):	pointer to computed value of solar angle of incidence.

Return Value

None.

Description

Calculates solar angles using the equations detailed in Section 4.2.1.

b) *double solarrad(month day, hour, beta, theta, drct, diff)*

Arguments

month (int):	month of year.
day (int):	day of year.
hour (int):	hour of day.
beta (double):	solar altitude.
theta (double):	solar angle of incidence.
drct (double *):	pointer to derived value of direct component of solar flux.
diff (double *):	pointer to derived value of direct component of solar flux.

Return Value

Total derived incident solar flux (direct plus diffuse).

Description

Reads total and diffuse solar flux from the weather data array, *WeatherData* (see Section AII.1), the array is indexed using *month, day* and *hour*. The derived direct component (total-diffuse) is corrected for angle of incidence and solar altitude (see Section 4.2.1).

c) *void solar_vector(sc, wc, gamma,beta)*

Arguments

sc (double *):	computed xyz coordinates of solar vector end point.
wc (double *):	xyz coordinates of solar vector start point.
gamma (double):	solar azimuth.
beta (double):	solar altitude.

Return Value

None.

Description

Calculates solar vector described in Section 4.2.

d) *int ray_in_cell(wc, gamma, beta, gp, srad)*

Arguments

wc (double *):	xyz coordinates of window grid point.
gamma (double):	solar azimuth.
beta (double):	solar altitude.
gp (int):	computed division/row/column location of irradiated finite volume cell.
srad (double *):	pointer to computed value of solar flux incident on finite volume grid cell.

Return Value

1 if a finite volume grid cell is irradiated, otherwise 0.

Description

Determines whether or not a finite volume grid cell is irradiated by a solar ray projected from the window concerned, using the algorithm detailed in Section 4.2. If a cell is found to be irradiated, the division/row/column location of the cell is returned in *gp[]*. If transmitting/absorbing/reflecting surfaces are encountered in the path of the ray, the transmitted radiation (srad) is corrected by the transmissivity of each transmitting element, the final incident flux being returned to the variable address **srad*.

e) *void solar_patch(body, surface, irad, gamma,beta)*

Arguments

body (int):	index number of building polyhedron with which window surface is associated.
surface (int):	index number of building polyhedron window surface.
irad (double):	incident direct solar flux.
gamma (double):	solar azimuth.
beta (double):	solar altitude.

Return Value

None.

Description

Calculates the transmitted direct component of solar radiation and calculates the location of an internally irradiated solar patch arising from an external window surface, assigning the correct proportion of flux to the finite volume cells lying within the patch. The algorithm adopted by this function is described in Section 4.2.

f) *void insolation(lat, month, day, hour)*

Arguments

lat (double):	latitude.
month (int):	month of year.
day (int):	day of month.
hour (int):	hour of day.

Return Value

None.

Description

Computes solar angles, solar flux and conducts internal solar tracking through calls to *solar_angles()*, *solarrad()* and *solar_patch()*.

AII.3.2.13*MODEL.C*

a) *void find_grid_coordinates(pt, xyz)*

Arguments

pt (double *):	xyz coordinates of point.
xyz (int *):	computed division/row/column location array for finite volume grid cell.

Return Value

None.

Description

Uses simple grid containment tests in the X, Y and Z dimensions in order to identify the finite volume grid cell in which the point *pt[]* lies. The location of the cell (as division/row/column index) is returned in the *xyz[]* array.

b) *void surface_geo(body, surface)*

Arguments

body (int):	building polyhedron index number.
surface (int):	building polyhedron surface index number.

Return Value

None.

Description

Calculates the area and orientation angles of a building polyhedron surface, using the algorithms described in Section 4.1. The coefficients of the surface plane equation are also calculated using Equations AI.8-AI.11, the derivations of which are described in Appendix I.

c) *void connection_cells(cn)*

Arguments

cn (struct connection_st): a connection structure (see Section AII.1).

Return Value

None.

Description

The *connection_cells()* function uses the method described in Section 4.1 in order to assign the correct materials database index number to the appropriate finite volume cells. Additionally, the *gridpar[][][].gridparbf.static* switches are assigned the appropriate logical values (TRUE or FALSE). Surface absorptivities are also assigned by this function.

d) *void surface_cells(body, surface)*

Arguments

body (int): building polyhedron index number.
surface (int): building polyhedron surface index number.

Return Value

None.

Description

Determines which finite volume grid cells are associated with each building polyhedron surface, using the method described in Section 4.1. The *gridpar[][][].nb*, *gridpar[][][].bd[]* and *gridpar[][][].sf[]* variables are subsequently assigned the correct values. Additionally, the *surface_cells()* function assigns the corect values to the *gridpar[][][].gridpar_bf* surface type switches (see Section AII.1).

e) *void external_cells()*

Arguments

None.

Return Value

None.

Description

The *external_cells()* function identifies all finite volume grid cells external to all building polyhedra by conducting grid traverses. Initially the function sets the external condition switch (set in a *gridpar[][][].gridpar_bf* structure) to TRUE and then FALSE when an outward facing external surface is encountered and similarly back to TRUE again when an inward facing external face is met.

AII.3.2.14*RAD.C*

a) *void strip_patch(body, surface, pc1, pt1, tpt1, tm)*

Arguments

body (int):	building polyhedron index number.
surface (int):	building polyhedron surface index number.
pc1 (double (*)[4]):	xyz coordinates of source surface vertices.
pt1 (double *):	xyz coordinates of source surface grid point.

tpt1 (double *): xyz coordinates of source surface grid point, transformed to 2D plane.

tm (double (*)[4]): transformation matrix to transform 2D points projected from source surface grid point back to 'real-world' coordinates.

Return Value

None.

Description

Computes 'strip and patch' coordinates projected from source surface grid point and subsequently calculates elemental view factors, in accordance with the procedure described in Section 4.3.

b) *void vf_grid(body, surface)*

Arguments

body (int): building polyhedron index number.

surface (int): building polyhedron surface index number.

Return Value

None.

Description

Transforms 3D building surface into 2D plane and constructs a two-dimensional grid for use in view factor determination. The *vf_grid()* function subsequently calls *strip_patch()* in order to determine the elemental view factors for each grid cell. Subsequent to all grid cells being accounted for, the elemental view factors are combined in order to derive point configuration view factors which are similarly combined to form surface-surface view factors (refer to Section 4.3).

c) *void view_factors()*

Arguments

None.

Return Value

None.

Description

The *view_factors()* function calculates radiation view factors for all participating surface pairings throughout the calculation domain, using function calls to *vf_grid()*.

AII.3.3 Simulation function call sequence files; the *simcalc* sub-directory

The *simcalc* sub-directory contains two source files, *simpler.c* and *boundary.c*, which contain the function call sequences for the finite volume solution algorithm and radiation/climatic boundary conditions, required by the simulation module EMBsim.

AII.3.3.1 *SIMPLER.C*

a) *simpler()*

Arguments:

None.

Return Value:

None.

Description:

The iteration residual log file is first initialised through a call to *init_log()*and subsequently, the *simpler()* function initiates a conditional loop in which is embedded a sequence of

function calls which constitutes the coded form of the numerical solution algorithm presented in Figure 6. The conditional loop is terminated when either the dependent variable residuals are less than the tolerances defined in the *embsim.sim* file, or when a CTRL-C interrupt is initiated. On termination of the loop, the *dump()* function is automatically called. As each iteration or loop pass is processed, dependent variable residuals are displayed on the monitor in the form of graphs, allowing convergence progress to be monitored, residuals are additionally written to the run-time residual log file *EMBSIM.LOG* by means of calls to *write_log()* on completion of each iteration.

AII.3.3.2 *BOUNDARY.C*

a) *boundary()*

Arguments:

None.

Return Value:

None.

Description:

The *boundary()* function is used to update external surface temperatures, external surface heat transfer coefficients and calculate internal and external solar irradiation using the *surface()* and *insolation()* functions. Refer to Figure 105 for function calls made by *boundary()*.

AII.3.4 Executable module source files; the *\exec* sub-directory

The *\exec* sub-directory contains the executable module source files which all contain the module *main()* function in addition to various other module specific functions. The various executable module source files and associated functions are detailed in Sections AII.3.4.1 - AII.3.4.5.

AII.3.4.1 *EMBGEN.C*

a) *main()*

Arguments:

argc (int):	number of arguments.
argv (char **):	1st argument: ASCII building geometry file name.
	2nd argument: binary building geometry file name,.

Description:

Generates a binary geometry file from an ASCII geometry file using the function call sequence illustrated in Figure 101. Refer to Section AII.2.1 for operational details of the EMBgen module.

AII.3.4.2 *EMBRAD.C*

a) *main()*

Arguments:

argc (int):	number of arguments.
argv (char **):	1st argument: ASCII building geometry file name.
	2nd argument: binary building geometry file name,.

Description:

The *main()* function calls the loadmodel() funtion to read the binary geometry file, generates view factors for the geometric body surface pairings, using the *view_factors()* function and finally writes the surface pairing view factor data to an ASCII text file using the *write_vf()* function. Refer to Figure 102 and Section AII.2.2 for function calls made by the *view_factors()* function and operational details of the EMBrad module.

AII.3.4.3 *EMBSIM.C*

a) *main()*

Arguments:

argc (int):	not used.
argv (char **):	not used.

Description:

Reads external data files (problem definition and support files), initialises dependent variable residual log file and sets up solution scheme loop. Refer to Figures 103, 104 and 105 and Section AII.2.3 for function calls made by the EMBsim *main()* function and operational details of the EMBsim module.

AII.3.4.4 *EMBTRAN.C*

a) *main()*

Arguments:

argc (int):	number of arguments.
argv (char **):	switches to specify which variables are written to the text file (/u /v /w /t /p /k /e /g).

Description:

The *loadf()* function is called to read the binary results file *embsim.dmp* into the *gd[][]*, *grid[][][][]*, *gridpar[][][]* and *bd[]* arrays. The argument switches are read into a byte array which is passed to *adump()* in order that the required variables are processed. The *adump()* function is then called to write the dependent variables to the ASCII results text file *embsim.asc*.

b) *adump()*

Arguments:

fn (char *):	ASCII text file name.
vswitch (unsigned char *):	array of variable switches.

Description:

The adump() function writes dependent variables to an ASCII text file as a series of x-y grid point tables for each grid division along the z-axis. The grid dimensions and surface polyhedra geometry may also be written to the file. The variable switches determine which variables are processed.

AII.3.4.5 *EMBVIEW.C*

a) *main()*

Arguments:

argc (int):	not used.
argv (char **):	not used.

Description:

The *main()* function incorporates a loop which enables the model building geometry to be rotated and scaled graphically to obtain required views. Dependent variables may then be displayed as vectors or contours. The function calls employed by main() are illustrated in Figure 106. Some of the functions used by main() are general geometric functions which are required by several EMB modules and are consequently included in the geometry.c source file (see Section AII.3.2.11), the remaining functions are detailed below.

b) *void inittcolors()*

Arguments:

None

Return value:

None

Description:

Initialises fill colour shades for temperature distribution. Fill colours range between blue and red using varying blue/red densities.

The RGB macro is used to mix colour densities in order to assign the required shade to a long integer:

#define RGB(r, g, b) (0x3F3F3FL & ((long)(b) << 16 | (g) << 8 | (r)))

where *r*, *g* and *b* are the proportions of red, green and blue.

The resulting colour indices are assigned to each temperature range integer using the Microsoft *_remappalette* function.

c) *void initdcolors()*

Arguments:

None

Return value:

None

Description:

Initialises contour colours. A maximum of ten contours may be defined for each dependent variable, the magnitude of each variable contour being defined in the external display data file display.dat. A Microsoft C predefined colour index is assigned to each contour integer using the Microsoft *_remappalette* function.

d) *void setwin(v, w)*

Arguments:

v (int):	viewport index
w (int):	window index

Return value:

None

Description:

This function sets the current viewport and window coordinates to the values set in the graphic window structure array using the *v* and *w* indices (see Section AII.2.5.1).

e) *void normalise_coords()*

Arguments:

None

Return value:

None

Description:

This function transforms the coordinates of the building geometry structure vertices (see

Section AII.1) into the system adopted by EMBview (see Section AII.2.5.1).

f) *void inittv()*

Arguments:

None

Return value:

None

Description:

Initialises temperatures and velocities for contour and area fill display.

For each grid point in the finite volume grid, the three velocity components (U, V and W) are resolved into a single value of air speed for later contour display. Temperature ranges are also established for area fill display.

g) *void borders(display)*

Arguments:

display (int): display type index

Return value:

None

Description:

Draws borders for the various windows. The two display types are for three-dimensional visualisation purposes and for two-dimensional variable display.

h) *void initfont()*

Arguments:

None

Return value:

None

Description:

Reads in font information for text display, using the Microsoft *_registerfonts()* and *_setfont()* functions.

i) *void initvid()*

Arguments:

None

Return value:

None

Description:

Initialises the video mode using the Microsoft *_setvideomode()* function.

j) *void initwin()*

Arguments:

None

Return value:

None

Description:

This function initialises the graphics window structure array for the various viewport coordinates (e.g. temperature scale, status line, etc.) and establishes the correct window ranges for the current building geometry.

k) *void initvectinf()*

Arguments:

None

Return value:

None

Description:

The *initvectinf()* function initialises the vector structures required for mapping the three displayable planes (Y-Z, X-Z and X-Y) onto a temporary X-Y plane for ease of manipulation (see Section AII.2.5.2).

l) *void initarrowhead()*

Arguments:

None

Return value:

None

Description:

Initialises arrow dimensions using overall building geometry for scaling.

m) *void initframe()*

Arguments:

None

Return value:

None

Description:

The axis frames described in Section AII.2.5 use structures for display information:

```
struct axes_st {
        double cc[4];
        char txt[2];
};
```

cc[4]: untransformed coordinates of principal axis vertices.
txt[2]: axes labels (e.g. "X").

```
struct frame_st {

        double vertices[4][4];
        struct axes_st axes[2];

};
```

vertices[4][4]: untransformed coordinates of frame vertices.
axes[2]: axis structures for the two pricipal axes.

The *initframe()* function initialises the above frame structures.

n) *void showtindex()*

Arguments:

None

Return value:

None

Description:

The *showtindex()* function displays the temperature index in the viewport set up by a call to *setwin()*.

o) *void showvindex(fi, factor, txt)*

Arguments:

factor (double):	scaling factor
fi (int):	frame index
xt (*char):	index label

Return value:

None

Description:

The *showvindex()* function displays the velocity and distance index in the viewport set up by a call to *setwin()*. The factor and fi arguments are required due to the different screen area allocations required for grid and velocity display.

p) *void arrow(fixedhead, xtail, ytail, xtip, ytip)*

Arguments:

xtail (double):	x-coordinate of arrow tail
ytail (double):	y-coordinate of arrow tail
xtip (double):	x-coordinate of arrow tip
ytip (double):	y-coordinate of arrow tip
fixedhead (unsigned char)	arrow head size toggle

Return value:

None

Description:

Draws arrow between points *xtail-ytail* and *xtip-ytip*. The *fixedhead* toggle determines whether or not the arrow head is scaled in proportion to the arrow length.

q) *void showstatusline(var, xyz, dim)*

Arguments:

var (int):	display type index
xyz (int):	axiz
dim (double):	distance of axis frame from origin

Return value:

None

Description:

This function displays a status line at the top of the screen indicating the current project title, the distance that the axis frame is from the origin along the current principal axis, the

orientations of the other two principal axes and the type of display viewport (flow visualisation, velocity vectors, contours, etc.).

r) *void showframe(xyz, m, dim, zc)*

Arguments:

xyz (int):	frame axis index.
m[][4] (double):	transformation matrix.
dim (double):	distance of frame from origin.
zc (double):	z-coordinate view dimension used in transforming 3d coordinates to 2d display coordinates.

Return value:

None

Description:

The *showframe()* function transforms the vertices of the axis frame stored in the array of frame structures (see *initframe()*) using the transformation matrix *m[][4]*. The axis frame array is indexed using the *xyz* integer. Subsequent to tranformation the frame is displayed to the current viewport.

s) *void showgeometry(m, zc)*

Arguments:

m[][4] (double):	transformation matrix.
zc (double):	z-coordinate view dimension used in transforming 3d coordinates to 2d display coordinates.

Return value:

None

Description:

The building geometry vertices stored in the *bd[]* array are transformed to the current rotation angle and scaling factor using the transformation matrix *m[][4]*. Subsequent to tranformation the building geometries are displayed to the current viewport.

t) *void contour_grid(var, fi, plane, cd, m, zc)*

Arguments:

var (int):	index of dependent variable to be displayed.
fi (int):	frame index.
plane (int):	index of plane on current principal axis.
cd (double):	distance of frame from origin.
m[][4] (double):	transformation matrix.
zc (double):	z-coordinate view dimension used in transforming 3d coordinates to 2d display coordinates.

Return value:

None

Description:

This function is used to generate a grid of dependent variable magnitudes to be displayed in conjunction with the dependent variable contours. The dependent variable magnitudes are generated using interpolations over a uniform grid dimension. Final display coordintes are generated by transforming the interpolated finite volume grid points using the current transformation matrix.

u) *void vecout(xstart, ystart, xstop, ystop)*

Arguments:

xstart (double):	x-coordinate of start of contour segment.
ystart (double):	y-coordinate of start of contour segment.

xstop (double): x-coordinate of end of contour segment.

ystop (double): y-coordinate of end of contour segment.

Return value:

None

Description:

The vecout function is called by *conrec()* to generate a contour segment between the two-dimensional coordinates *xstart-ystart* and *xstop-ystop*.

v) *void conrec(vf, nband)*

Arguments:

vf (struct vector_st): a vector structure as detailed in Section AII.2.5.2.

nband (int): number of contour bands.

Return value:

None

Description:

The *conrec()* function uses a conventional contouring algorithm to generate dependent variable contours. The magnitudes for which contours are generated are defined in the display data file *display.dat*.

w) *void showvar(var, fi, cd, m, plane, zc)*

Arguments:

cd (double): distance of frame from origin.

m[][4] (double): transformation matrix.

zc (double): z-coordinate view dimension used in transforming 3d

coordinates to 2d display coordinates.

var (int): index of dependent variable to be displayed.

fi (int): frame index.

plane (int): index of plane on current principal axis.

Return value:

None

Description:

The *showvar()* function extracts dependent variable magnitudes from the finite volume dependent variable grid array *grid[][][][]* (see Section AII.1), for the selected division within the selected plane (Y-Z, X-Z or X-Y). The *showvar()* function then determines the coordinates of each of the appropriate finite volume grid centres, using the grid dimension array *gd[][MAXDIM]*, maps the coordinates to a temporary X-Y coordinate system and finally transforms the coordinates by the current rotation angle and scaling factor using factoring the transformation matrix *m[][4]*.

APPENDIX III - PROGRAM DATA FILES

AIII.1 Geometry File

```
# Data file for EMBGEN
# Blank lines or lines beginning with a '#' symbol will be ignored

#  No. X cells

19

#  X cell dimensions

#Ref.   Dimension

1       0.10000000
2       0.04980000
3       0.15020000
4       0.20000000
5       0.30000000
6       0.30000000
7       0.30000000
8       0.30000000
9       0.30000000
10      0.30000000
11      0.30000000
12      0.30000000
13      0.30000000
14      0.30000000
15      0.30000000
16      0.20000000
17      0.20000000
18      0.10000000
19      0.10000000

#  No. Y cells

16

#  Y cell dimensions

#Ref.   Dimension

1       0.10000000
2       0.10000000
3       0.30000000
4       0.30000000
5       0.25000000
6       0.25000000
7       0.25000000
8       0.25000000
9       0.21405637
```

10 0.17684363
11 0.10000000
12 0.04980000
13 0.06000000
14 0.10000000
15 0.10000000
16 0.10000000

No. Z cells

25

Z cell dimensions

#Ref. Dimension

1 0.10000000
2 0.20000000
3 0.20000000
4 0.20000000
5 0.20000000
6 0.20000000
7 0.20000000
8 0.24500000
9 0.04980000
10 0.11520000
11 0.04980000
12 0.11520000
13 0.04980000
14 0.11520000
15 0.04980000
16 0.11520000
17 0.04980000
18 0.24500000
19 0.20000000
20 0.20000000
21 0.20000000
22 0.20000000
23 0.20000000
24 0.20000000
25 0.10000000

Latitude (degrees)
55.0

Site orientation (degrees clockwise from North)
0.0

Data for surface bodies

No. of surface bodies
2

Data for surface body 1

Body type
[1: External; 2: Internal 3: Enclosure 4: Isothermal obstacle 5: Interface]
1

No. vertices
16

Cartesian coordinates of each vertex

#No	X	Y	Z
1	0.000	0.000	0.000
2	4.400	0.000	0.000
3	4.400	0.000	0.900
4	4.400	0.000	2.900
5	4.400	0.000	3.800
6	0.000	0.000	3.800
7	0.000	2.700	0.000
8	4.400	2.700	0.000
9	4.400	2.700	0.900
10	4.400	2.700	2.900
11	4.400	2.700	3.800
12	0.000	2.700	3.800
13	4.400	0.800	0.900
14	4.400	0.800	2.900
15	4.400	2.400	2.900
16	4.400	2.400	0.900

No. surfaces
10

Vertex ordering for each surface
NOTE - ANTICLOCKWISE order for external/obstacle/enclosure surface vertices,
CLOCKWISE order for internal surface vertices
FIRST THREE VERTICES MUST BE NON-COLLINEAR

#No	No.Vert	Vert
1	4	2 8 7 1
2	6	9 8 2 3 13 16
3	4	4 14 13 3
4	4	14 15 16 13
5	4	15 10 9 16
6	6	4 5 11 10 15 14
7	4	6 12 11 5
8	4	1 7 12 6
9	6	7 8 9 10 11 12
10	6	2 1 6 5 4 3

Data for surface body 2

Body type
[1: External; 2: Internal 3: Enclosure 4: Isothermal obstacle 5: Interface]
2

No. vertices
16

Cartesian coordinates of each vertex

#No	X	Y	Z
1	0.100	0.100	0.100
2	4.300	0.100	0.100
3	4.300	0.100	0.900
4	4.300	0.100	2.900
5	4.300	0.100	3.700
6	0.100	0.100	3.700
7	0.100	2.600	0.100
8	4.300	2.600	0.100
9	4.300	2.600	0.900
10	4.300	2.600	2.900
11	4.300	2.600	3.700
12	0.100	2.600	3.700
13	4.300	0.800	0.900
14	4.300	0.800	2.900
15	4.300	2.400	2.900
16	4.300	2.400	0.900

No. surfaces
10

Vertex ordering for each surface
NOTE - ANTICLOCKWISE order for external/obstacle/enclosure surface vertices,
CLOCKWISE order for internal surface vertices
FIRST THREE VERTICES MUST BE NON-COLLINEAR

#No	No.Vert	Vert
1	4	1 7 8 2
2	6	2 8 9 16 13 3
3	4	3 13 14 4
4	4	13 16 15 14
5	4	16 9 10 15
6	6	5 4 14 15 10 11
7	4	5 11 12 6
8	4	6 12 7 1
9	6	7 12 11 10 9 8
10	6	6 1 2 3 4 5

Connections
Define from outer polyhedron to inner polyhedron
No. of surface connections
10

#[0: Wall; 1: Window; 2: Door; 3: Isothermal external wall]

#No	Type	Bd1	Sf1	Bd2	Sf2	Mat
1	3	1	1	2	1	1
2	3	1	2	2	2	1
3	3	1	3	2	3	1
4	1	1	4	2	4	3
5	3	1	5	2	5	1

```
6   3    1   6   2   6    1
7   3    1   7   2   7    1
8   3    1   8   2   8    1
9   3    1   9   2   9    1
10  3    1   10   2  10    1
```

AIII.2 Boundary Condition File

```
#  Variable X1 X2 Y1 Y2 Z1 Z2 Value
#
#  Variable:  [K] - Boundary turbulence kinetic energy: (float)
#             [E] - Boundary dissipation of turbulence kinetic energy: (float)
#             [T] - Isothermal cell: Temperature (float)
#             [U] - Isovel cell: Velocity X-component (float)
#             [V] - Isovel cell: Velocity Y-component (float)
#             [W] - Isovel cell: Velocity Z-component (float)
#             [Q] - Energy equation source term: Flux (float)
#             [S] - Static cell: Const.(int)

#  TEST CASE: E2

F 1 1 12 12 9 9 0
F 1 1 12 12 11 11 0
F 1 1 12 12 13 13 0
F 1 1 12 12 15 15 0
F 1 1 12 12 17 17 0

U 0 0 12 12 9 9 2.5403
U 0 0 12 12 11 11 2.5403
U 0 0 12 12 13 13 2.5403
U 0 0 12 12 15 15 2.5403
U 0 0 12 12 17 17 2.5403

U 2 2 12 12 9 9 2.5403
U 2 2 12 12 11 11 2.5403
U 2 2 12 12 13 13 2.5403
U 2 2 12 12 15 15 2.5403
U 2 2 12 12 17 17 2.5403

# V 2 2 11 12 9 9 2.1315
# V 2 2 11 12 11 11 2.1315
# V 2 2 11 12 13 13 2.1315
# V 2 2 11 12 15 15 2.1315
# V 2 2 11 12 17 17 2.1315

T 0 0 12 12 9 9 30.0
T 0 0 12 12 11 11 30.0
T 0 0 12 12 13 13 30.0
T 0 0 12 12 15 15 30.0
T 0 0 12 12 17 17 30.0
```

```
K 0 0 12 12 9 9 0.22
K 0 0 12 12 11 11 0.22
K 0 0 12 12 13 13 0.22
K 0 0 12 12 15 15 0.22
K 0 0 12 12 17 17 0.22

E 0 0 12 12 9 9 0.0872
E 0 0 12 12 11 11 0.0872
E 0 0 12 12 13 13 0.0872
E 0 0 12 12 15 15 0.0872
E 0 0 12 12 17 17 0.0872

F 1 1 9 9 12 14 0

U 0 0 9 9 12 14 -0.525004

T 19 19 5 13 6 20 10.0
```

AIII.3 Materials Database File

```
# EMBSIM.MAT - Material thermophysical properties
# Density (kg/m3), Specific Heat Capacity (J/kgK), Thermal Conductivity (W/mK),
# Absorptivity Transmissivity

# 1: Common building brick
1600 840 0.04 0.90 0.00

# 2: Wood (Oak)
540 2400 0.15 0.90 0.00

# 3: Glass
2600 670 10.00 0.90 0.44

# 4: Insulation
50 670 0.02 0.90 0.00

# 5: Steel
7860 420 63 0.90 0.00
```

AIII.4 Display Data File

```
TEST [NO RADIATION]
0
2.0
20.0 22.0
0.1
0.05 0.08 0.1 0.15 0.5
```

20.2 20.4 20.6 20.8 21.0 21.2 21.4 21.6 21.8 22.0

REFERENCES

[1] CLARKE, J.A.
"Building Energy Simulation: The State-of-the-Art"
Internal Paper, Energy Simulation Research Unit, Dept. Architecture, University of Strathclyde

[2] MACKEY, C.O., WRIGHT, L.T.
"Periodic Heat Flow - Homogeneous Walls or Roofs"
ASHRAE Trans., vol.50 293-312, 1944

[3] STEPHENSON, D.G., MITALAS, G.P.
"Cooling Load Calculations by Thermal Response Factor Method"
ASHRAE Trans., vol.73.1.III 1-7, 1967

[4] MILBANK, N.O., HARRINGTON-LYNN, J.
"Thermal Response and the Admittance Procedure"
Building Research Station CP 16/74, 1974

[5] CLARKE, J.A.
"Energy Simulation in Building Design"
Adam Hilger Ltd., 1985

[6] HITTLE, D.C.
"The Building Loads Analysis System Thermodynamics (BLAST) Program, Version 2.0: Users Manual Volume I"
National Technical Information Service, 1979

[7] WALTON, N.W.
"Thermal Analysis Research Program Reference Manual"
U.S. Department of Commerce, National Bureau of Standards, 1983

[8] ROACHE, J.R.
"Computational Fluid Dynamics"
Hermosa Publishers,1972

[9] GOSMAN, A.D., PUN, W.M., RUNCHAL, A.K., SPALDING, D.B., WOLFSHTEIN, M.

"Heat and Mass Transfer in Recirculating Flows"
Academic Press,1969

[10] COURANT, R., ISAACSON, E., REES, M.
"On the Solution of Non-Linear Hyperbolic Differential Equations by Finite Differences"
Comm. Pure Appl Math., vol.5,p.243,1952

[11] SPALDING, D.B.
"A Novel Finite-Difference Formulation for Differential Expressions Involving Both First and Second Derivatives"
Int. J. Num. Methods Eng. vol 4, p 551, 1972

[12] PATANKAR, S.V.
"A Calculation Procedure for Two-Dimensional Elliptic Situations"
Num. Heat Transfer vol 2, 1979

[13] CARETTO, L.S., GOSMAN, A.D., PATANKAR, S.V., SPALDING, D.B.
"Two Calculation Procedures for Steady, Three-Dimensional Flows With Recirculation"
Proceedings of the Third International Conference on Numerical Methods in Fluid Mechanics, Vol.II, pp.60-68

[14] GOSMAN, S.V., IDERIAH, F.J.K.
"A General Computer Program for 2D Turbulent Recirculating Flows"
Imperial College London, 1976

[15] PATANKAR, S.V.
"Numerical Heat Transfer and Fluid Flow"
McGraw Hill, New York, 1980

[16] SPALDING, D.B.
"Mathematical Modelling of Fluid Mechanics, Heat Transfer and Chemical Reaction Processes: A Lecture Course"
Imperial College London, 1980

[17] TENNEKES, H., LUMLEY, J.L.
"A First Course in Turbulence"

MIT Press, 1972

[18] BOUSSINESQ, J.
"Theorie de l'Ecoulement Tourbillant"
Mem.Acad.Sci.,Vol.23,No.46

[19] RODI, W.
"Turbulence Models and their Application in Hydraulics - A State of the Art Review"
International Association for Hydraulic Research Publ., 1980

[20] WHITTLE, G. E.
"Numerical Air Flow Modelling"
BSRIA Technical Note TN 2/87 1987

[21] PRANDTL, L.
"Report on Investigation into Developed Turbulence"
Translated by D.B. Spalding from the German
Imperial College, Heat Transfer Section Report HTS/TN/80/1

[22] PRANDTL, L.
"Uber Ein Neues Formelsystem fur die Ausgebildete Turbulenz"
Nazhr Abad der Wissenschaft in Gottingen
Gottingen: Van den Loech und Ruprecht, pp.6-19

[23] SPALDING, D.B.
"Turbulence Models - A Lecture Course"
Imperial College London, Heat Transfer Section Report CFD/82/4

[24] KOLMOGOROV, A.N.
"Equations of Turbulent Motion of an Incompressible Fluid"
Izv. Akad. Nauk. SSR, Seria fizicheska Vi., No. 1-2, pp 56-58, 1942
Eng. Trans.: Imperial College, Mech. Eng. Dept. Rept. ON/6, 1968

[25] RODI, W., SPALDING, D.B.
"A Two Parameter Model of Turbulence and its' Application to Free Jets"
Warme und Staffubertragung 3 (1970) p.20

[26] SPALDING, D.B.
"The k-W Model of Turbulence"
Imperial College London, Mechanical Engineering Department Report TM/TN/A/16

[27] HARLOW, F.H., NAKAYAMA,P.
"Transport of Turbulence Energy Decay Rate"
Los Alamos Science Lab.,University of California Report LA-3854, 1968

[28] LAUNDER, B.E., SPALDING, D.B.
"The Numerical Computation of Turbulent Flows"
Computer Methods in Applied Mechanics and Engineering 3 (1973), pp.269 - 289

[29] CHOU, P.Y.
"On Velocity Correlations and the Solution of the Equations of Turbulent Fluctuation"
Quart. J. Appl. Math. 3, 1, pp. 38-54, 1945

[30] RODI, W.
"A New Algebraic Relation for Calculating the Reynolds Stresses"
ZAMM 56, T219-T221, 1976

[31] LAUNDER, B.E.
"Stress-Transport Closures: Into the Third Generation"
Proceedings of the First Symposium on Turbulent Shear Flows. Springer-Verlag, New York

[32] DEARDROFF, J.W.
"The Use of Subgrid Transport Equations in a Three-Dimensional Model of Atmospheric Turbulence"
Trans. ASME, J. Fluid. Eng., pp. 429-438, 1973

[33] IDERIAH, F.J.H.
"Prediction of Turbulent Cavity Flow Driven by Buoyancy and Shear"
Journal Mechanical Engineering Science, 22, No.6, 1980, pp.286-295

[34] SCHLICHTING, H.
"Boundary Layer Theory"

McGraw-Hill, 1979

[35] IDERIAH, F.J.K.
"Turbulent Natural and Forced Convection in Plumes and cavities"
PhD Thesis, University of London, 1977

[36] JAYATILLAKA, C.V.L.
"The Influence of Prandtl Number and Surface Roughness on the Resistance of the Laminar Sub-Layer to Momentum and Heat Transfer"
In 'Progress in Heat and Mass Transfer', Vol I, Pergamon Press, 1969, London

[37] NIELSEN, P.V.
"Prediction of Air Distribution in a Forced Ventilation Room"
Ingeniorens Ugeblad No.5, 1973

[38] ZOHRABIAN, A.S., MOKHTARZADEH-DEHGHAN, M.R., REYNOLDS, A.J.
"A Numerical Study of Buoyancy-Driven Flows of Mass and Energy in a Stairwell"
Proc. 9th AIVC Conference, Gent, Belgium, Vol.2

[39] JEDRZEWSKA-SCIBAT, T., LIPINSKI, D.M.
"Numerical Prediction of Air Distribution in Industrial Halls"
CLIMA 2000, Vol.4, Indoor Climate, VVS Kongres-VVS Messe, Copenhagen, 1985

[40] HJERTAGER, B.H., MAGNUSSEN, B.F.
"Numerical Prediction of Three-Dimensional Turbulent Buoyant Flow in a Ventilated Room"
Heat Transfer and Turbulent Buoyant Convection, Volume 2
Ed. Spalding, D.B., Afgan, N.
McGraw-Hill, 1976

[41] SAKAMOTO, Y., MATSUO, Y.
"Numerical Predictions of Three-Dimensional Flow in a Ventilated Room Using Turbulence Models"
Applied Mathematical Modelling, Volume 4, pp 67-72

[42] GOSMAN, A.D., NIELSEN, P.V., RESTIVO, A., WHITELAW, J.H.
"The Flow Properties of Rooms with Small Ventilation Openings"

Transactions of the ASME Journal of Fluids Engineering, Volume 102, pp 316-323

[43] CHEN QUINGYAN, G., VAN DER KOOI, J.
"Measurement and Computations on Air Movement and Temperate Distribution in a Climate Room"
XVII International Kongress fur Kaltetechnik, 1987

[44] SPALDING, D.B.
"Four Lectures on the PHOENICS Computer Code"
Imperial College London, Heat Transfer Section, Report No. CFD/82/5

[45] HAGHIGHAT, F., JIANG, Z., WANG, J.C.Y., ALLARD, F.
"Air Movement in Buildings Using Computational Fluid Dynamics
Transactions of the ASME Journal of Solar Engineering, Volume 114, pp 84-92, 1992

[46] OHIRA, N., OMORI, T.
"Ventilation Behaviour in a Void Space Furnished with Gas Water-Heaters"
Proceedings of 5th International Conference on Air Distribution in Rooms Roomvent '96 July 17-19, 1996

[47] HAVET, M., BLAY, D., VEYRAT, O.
"Analysis of Thermal Comfort in Large Enclosures Heated by Radiant Panels"
Proceedings of 4th International Conference on Air Distribution in Rooms Roomvent '94 June 15-17, 1994

[48] GAN, G.
"Evaluation of Room Air Distribution Systems Using Computational Fluid Dynamics"
Energy and Buildings Volume 23, pp 83-93, 1995

[49] MURAKAMI, S., KATO, S.
"Numerical and Experimental Study on Room Air Flow - 3-D Predictions using the k-ε Turbulence Models"
Buildings and Environment, Volume 24, 1, pp 85-97, 1989

[50] KURABUCHI, T., FANG, J.D., GROT, R.A.
"A Numerical Method for Calculating Indoor Air Flows Using a Turbulence Model"
National Institute of Standards and Technology, USA, 1989

[51] HAGHIGHAT, F., JIANG, Z., WANG, J.C.Y.
"Natural Convection and Air Flow Pattern in a Partitioned Room With Turbulent Flow"
USA, Preprint, ASHRAE Transactions, Vol 95, Part 2, 1989

[52] CHEN QUINGYAN.
"Indoor Airflow, Air Quality and Energy Consumption of Buildings"
PhD Thesis, Delft University, 1989

[53] HOLMES, M.J., LAM, J.K-W., RUDDICK, K.G., WHITTLE, G.E.
"Computation of Conduction, Convection and Radiation in the Perimeter Zone of an Office Space"
Proc. Int. Conf. 'Roomvent 90', Oslo, Norway, June 1990

[54] CHARTERED INSTITUTE OF BUILDING SERVICES ENGINEERS
"C3 Guide - Heat Transfer"
CIBSE, 1976

[55] RODI, W.
"On the Equation Governing the Rate of Turbulent Energy Dissipation"
Imperial College London, Mech. Eng. Dept. Report TM/TN/B/14, 1971

[56] RODI, W.
"Turbulent buoyant jets and plumes"
Edited by Rodi,W., Pergamon Press, 1982

[57] HARLOW, F.H., WELCH, J.E.
"Numerical Calculation of Time-Dependent Viscous Incompressible Flow of Fluid With Free Surface"
Phys. Fluids., Vol 8, p. 2182, 1965

[58] GALPIN, P.S., RAITHBY, G.D.
"Numerical Solution of Problems in Incompressible Fluid Flow: Treatment of the Temperature-Velocity Coupling"
Numerical Heat Transfer, Vol 10, pp. 105-129, 1986

[59] McADAMS, W.H.

"Heat Transmission"
New York: McGraw-Hill, 1954

[60] ITO, N., KIMURA, K., OKA, J.
"A Field Experiment Study on the Convection Heat Transfer Coefficient on the External Surface of a Building"
ASHRAE Trans. (2225) 184-91, 1972

[61] MORTENSON, E.
"Computer Graphics: An Introduction to the Mathematics and Geometry"
Heinmann, 1989

[62] FOLEY, D., VAN DAM, A.
"Fundamentals of Interactive Computer Graphics"
Addison Wesley, 1982

[63] DUFFIE, J.A., BECKMAN, W.A.
"Solar Engineering of Thermal Processes"
John Wiley and Sons, New York, 1980

[64] MOORE, G.R., NUMAN, M.Y.
"Form Factors: The Problem of Partial Obstruction"
Working Paper for the Martin Centre for Architectural and Urban Studies, November 1982

[65] HEIKKINEN, J.
"Specification of Test Case B (Forced Convection, Isothermal)"
IEA Annex 20 RID 1.13 April 1989

[66] LEMAIRE, A.D.
"Specification of Test Case D (Free Convection with Radiator)"
IEA Annex 20 RID 1.15 May 1989

[67] HEIKKINEN, J.
"Specification of Test Case E (Mixed Convection, Summer Cooling)"
IEA Annex 20 RID 1.14 April 1989

[68] LEMAIRE, A.D..
"Testrooms, Identical testrooms"
IEA Annex 20 RID 1.03 May 1989

[69] NEILSEN, P.V.
"Selection of air terminal device"
IEA Annex 20 RID 1.02 May 1988

[70] NEILSEN, P.V.
"Representation of boundary conditions at supply openings"
IEA Annex 20 RID 1.11 February 1989

[71] HEIKKINEN, J., PIIRA, K.
"Simulation of test case B2, isothermal forced convection"
IEA Annex 20, Internal Paper, December 1990

[72] FONTAINE, J.R.
"Simulation results, test case B2"
IEA Annex 20, Internal Paper, February 1991

[73] FURST, J.
"Simulation of test case B (forced convection isothermal)"
IEA Annex 20, Internal Paper, March 1990

[74] CHEN, Q.
"Simulation of test case B"
IEA Annex 20, Internal Paper, January 1990

[75] VOGL, N., RENZ, U.
"Simulation of simple test cases (case B)"
IEA Annex 20, Internal Paper, March 1991

[76] SAID, M.N.
"Simulation of test case B2, isothermal forced convection"
IEA Annex 20, Internal Paper, February 1991

[77] LEMAIRE, A.D., ELKHUIZEN, P.A.
"Simulation of test case B (forced convection, isothermal)"
IEA Annex 20, Internal Paper, October 1991

[78] TJELFLAAT, P.O..
"Simulation of test case B (forced convection, isothermal)"
IEA Annex 20, Internal Paper, June 1990

[79] JOHANSSON, S.H.
"Simulation of test case B1 and E1 in IEA program annex 20: air flow patterns within buildings"
IEA Annex 20, Internal Paper, October 1990

[80] SKOVGAARD, M.
"Simulation results (case B)"
IEA Annex 20, Internal Paper, June 1991

[81] LEMAIRE, A.D.
"Simulation of test case E, (mixed convection, summer cooling)"
IEA Annex 20, Internal Paper, March 1991

[82] HEIKKINEN, J
"Measurement and simulation results"
IEA Annex 20, Internal Paper, May 1991

[83] RENZ, U.
"Simulation results"
IEA Annex 20, Internal Paper, May 1991

[84] LEMAIRE, A.D.
"Simulation of test case D, (free convection with radiator)"
IEA Annex 20, Internal Paper, March 1991

[85] VOGL, N., RENZ, U.
"Simulation of simple test cases (case D)"
IEA Annex 20, Internal Paper, March 1991

[86] FURST, J.
"Simulation of test case D"
IEA Annex 20, Internal Paper, May 1990