Computational hydrodynamics in air flows modeling

Using the Unreal Engine based on the numerical solution of the Navier-Stokes equations

Pavel V. Yudin
Innovative projects promotion Department
Vladivostok State University of Economics and Service
Vladivostok, Russia
YudinPV@mail.ru, https://orcid.org/0000-0002-3405-9743

Margarita I. Fedina
Institute of Information Technology
Vladivostok State University of Economics and Service
Vladivostok, Russia
Chyerra@yandex.ru

Evgeniy V. Strizh
Institute of Information Technology
Vladivostok State University of Economics and Service
Vladivostok, Russia
Strizh.evgeniy@gmail.com

Olga V. Yudina
Research and Development Department
Planparaliya Ltd.
Vladivostok, Russia
Orvik80@mail.ru

Viktoria Khomotiuk
Technical Assistant
Sacramento, USA
Viktoriakhomotiuk@gmail.com

***Abstract*— Computational fluid dynamics (CFD) is a class of methods of mathematical and computer modeling that is of great scientific and practical importance for solving problems of modeling the interaction of airflows. Airflows can be different in temperature, density, and other physical parameters. CFD is important in the design of climate control systems in industrial, public buildings and premises, the design of life support systems and firefighting.**

Of particular importance is the problem of modeling the distribution of air masses in premises with obstacles in the form of equipment located in them, as well as in rooms of complex configuration. The article proposes a combined modeling method based on the numerical solution of the Navier-Stokes system of equations. This algorithm is encapsulated in an application created on the Unreal Engine platform - an environment for computer simulation of 3D games. The novelty of the proposed method is the use of the game "engine" to solve important industrial problems in poultry, livestock, design of climate systems. The article shows the possibilities for visualization of the modeling process in the created software product.

Keywords—computational hydrodynamics, modeling of airflows, Unreal Engine, Navier-Stoke equations, airflow simulation, computational fluid dynamics, CFD

I. INTRODUCTION

The ability to maintain the temperature within certain limits is an actual problem in many industries and spheres of human activity. In production facilities where working conditions are regulated or there is a threat of occupational disease, the microclimate is especially important.

It is necessary to strictly regulate the microclimate in enterprises where there are clear temperature requirements for working conditions, non-observance of which will lead to disruption of the production process, and subsequently, to losses. The control of the indoor microclimate is especially significant when performing works related to high or low temperatures [27] - for example, in metalworking, agriculture, food production, the chemical industry, aerospace industry, firefighting, sports facilities, public buildings (offices, shopping centers) and many others.

Improper production conditions can lead to serious problems associated with the health of employees, deterioration of equipment and facilities, disruption of the production process [30]. The climate control of premises for keeping farm animals, such as poultry and pigs, plays a significant role since their productivity and the economic efficiency of the production process rely on the well-being of the animals' living environment.

Modeling the distribution of airflows that facilitate proper gas and heat transfer in the house or barn, solves the problems of equipment configuration in the design of agricultural production facilities. [34, 35].

To manage the microclimate, enterprises spend a lot of money on excessively high-quality ventilation systems. The costs of their decisions can raise doubts since there is a good chance that it would be possible to manage with lower capital costs or by the proper distribution between climate control systems.

To solve the problem associated with the right choice of ventilation methods, one can resort to the use of software tools for simulating the behavior of the air in the premises. Even though the above method can reduce production costs, it is used infrequently due to ignorance, unwillingness or lack of suitable software. As part of this work, we will work on the last aspect of this problem.

II. Actuality

There are enough programs on the international market to simulate airflow. All of them have pros and cons, but for the most part, they are mathematical, not allowing the user to control the interaction processes with different physical characteristics of airflows: denser cold and more discharged and saturated with humidity warm streams.

We focus on such a problem as the lack of suitable software. This work is directed precisely at the solution. The novelty of the work lies in combining software methods and mechanisms. A subprogram that solves the system of Navier-Stokes equations for modeling the characteristics of temperature, density and airflow velocity by numerical methods is encapsulated in a platform for creating 3D gaming applications - the Unreal Engine program. Unreal Engine allows simulating the key relationships within the flows and between adjacent air mass flows, taking into account the configuration of the 3D model of the room and airflow sources (supply and exhaust ventilation), distributed by the user arbitrarily inside the model. The resulting software product will help to determine the choice of a ventilation system for the premises. It will help to make the right choices and avoid problems and losses associated with the wrong indoor climate, which is caused by an improperly selected ventilation system.

III. Formulation of a problem

The main objective of the project is to create a software product that is an application software for simulating the behavior of air flows in confined spaces, with the help of Unreal Engine platform. The result of the work is a fully functioning program that has all the necessary functionality, for example, tracking air characteristics in the visual interface.

To achieve the result we need:

- to create a base of theoretical materials for the development of the program;
- to conduct a search and analysis of analog programs;
- in the process of creation of the program, to study and use a set of methods of computational fluid dynamics;
- to study the basic differential Navier-Stokes equations for modeling viscous Newtonian fluid (air mass flows) and to study methods for their solution;
- to find a suitable method for modeling the Navier-Stokes equation;
- to implement the algorithm for the programmatic numerical solution of the Navier-Stokes equation to further include this subprogram into the architecture of the main program;
- to do modeling and visualization of airflow in enclosed spaces in the Unreal Engine environment.

The functioning software obtained as a result of the development will simulate the air flows in the premises, based on the numerical solution of the original system of Navier-Stokes differential three-dimensional equations. It will help to design the ventilation system under the given parameters and the necessary air distribution system.

IV. Theoretical base

Computational fluid dynamics (CFD) is a combination of many different theoretical, experimental and numerical methods that are designed to simulate the flow of liquids and gases, their heat transfer and mass exchange, reactive flows and other elements associated with them [3]. It also includes physical and mathematical methodology. The basis of any study in the field of computational fluid dynamics is the formulation of basic equations of fluid dynamics or gas dynamics [20, 21]. In this case, four main equations can be distinguished, namely:

- the continuity equation;
- the equation of conservation of momentum;
- energy conservation equation;
- equation of state (for gases).

The above equations describe a model of the flow of a medium. Depending on the features of the problem being solved, the model can be supplemented with equations to take into account turbulence, transport of substances, chemical reactions, multiphase nature, electromagnetic interactions, and other points as necessary. The system of nonlinear differential equations of the second-order is composed of all the above equations [7]. There is a wide list of tasks that can be solved with the help of computational fluid dynamics methods by using commercial programs:

Automotive industry:

- determination of the resistance level of the main body of the vehicle to the moving airflow;
- interior ventilation;
- ventilation of the engine compartment;
- fuel combustion design.

Aerospace industry:

- detailed modeling of airflow around missiles and aircraft;
- fire safety of the cabin of flying vehicles;
- modeling of physical and chemical phenomena in turbojet engines;
- modeling and design of turbines.

Technological processes for the manufacture of materials:

- design of plastic and metal molding;
- modeling of biological and chemical processes in reactors.

Production:

- designing the work of freezers;
- designing the operation of ventilation systems.

Construction:

- accurate calculation of wind loads on structures;
- design of ventilation and air vents in the buildings;
- determination of possible resistances of air ducts and water distribution devices.

Energetics:

- accounting of burners for burning fuel in boilers;
- calculations of nitrogen oxide emissions by boilers.

Climate control in premises:

- modeling the behavior of airflows in the room and calculating temperatures and humidity levels;
- calculation for the selection of ventilation equipment in industrial premises.

Agriculture:

- designing the operation of ventilation systems in buildings for growing and keeping farm animals and poultry;
- calculations for the selection of ventilation equipment in meat processing industrial premises.

Computational fluid dynamics methods are used not only in commercial projects and industry but also in science and other industries [16]. For example, making weather simulations or using simulations in environmental engineering.

In general, computational hydrodynamics carries a significant role in the development of the modern technical world. Also, the application of this direction in physics is extremely important in emergencies, since through its methods it is possible to simulate the directions of the spread of pollution in the water and air and prevent the spread of fires in forests and cities [6, 7, 8]. One of the most important equations in hydrodynamics are the Navier - Stokes equations, which are a system of partial differential equations. It is with their help that one can describe the motion of the viscous Newtonian fluid, as well as non-discharged gases [24].

A Newtonian fluid is a viscous fluid that obeys Newton's viscous friction law, that is, its tangential stress and velocity gradient are linearly dependent. The proportionality coefficient between these values is known as viscosity. The viscosity of such a fluid depends only on pressure and temperature. The viscosity of a non-Newtonian fluid is velocity-dependent [25].

The Navier-Stokes equations are used in mathematical modeling of many natural phenomena and technical problems. The system consists of two equations: equation of motion and equation of continuity. In hydrodynamics, the only one vector equation of motion is usually called the Navier – Stokes equation [28].

In the vector form for a liquid, it is written as follows:

$$\frac{\partial \vec{V}}{\partial t} = -\left(\vec{V}\nabla\right)\vec{V} + V\Delta\vec{V} - \frac{1}{\rho}\nabla P + \vec{f} \qquad (1)$$

Where ∇ - the Hamilton operator;

Δ - Laplace operator,

$\vec{V}$ - velocity vector,

t – time,

V - kinematic viscosity coefficient,

ρ – density,

p – pressure,

$\vec{f}$ - vector of density of mass forces.

Sometimes, in the case of an incompressible fluid, the continuity equation is called the incompressibility equation, and in the case of a compressible fluid, continuity equation [10]. The solution to the Navier-Stokes equation is the velocity at each point in the space. Based on the obtained velocity field, pressure or temperature can be calculated. Depending on the tasks, other equations are added to the equation, for example, the heat equation, or vice versa, some variables are not taken into account, and various assumptions are made. For example, the introduction of the Lawrence force and the Maxwell equation will allow the Navier-Stokes equation to describe the phenomenon of electrohydrodynamics and magnetohydrodynamics, as well as quite successfully simulate the behavior of plasma and interstellar gas. The problem of solving the Navier-Stokes equation is the lack of analytical solutions. The proof or refutation of the existence of an exact solution to the Cauchy problem for the three-dimensional Navier-Stokes equation is one of the "millennium problems" [22, 23].

Programs that simulate airflows and use computational fluid dynamics have been existing for many years. To create our unique and competitive product, we conducted a search and analysis of analog programs [2, 4, 9]. In a general analysis of a group of programs with computational fluid dynamics, five characteristics can be distinguished related to the qualities and functionality of the software [6, 17]. These are the presence or absence of open source code, a shell, a specialized focus, the possibility of integration with an automatic design system (with a system of automatic projecting), and a combination of all the above characteristics in one comprehensive package. Programs such as AltairAcuSolve, ADINA, SolidWorks Flow Simulation, Autodesk CFD, OpenFOAM were investigated for capabilities. As a result, the main functions of analog programs for inclusion in our software product were highlighted.

When designing software, the following questions arose:

- What programming language to use for writing a program?

- What software environment to use for visualization?

To accomplish our tasks, we need an environment with good visualization, with the ability to create a convenient user interface for viewing parameters, and the ability to easily create a virtual world for modeling. As systems that are completely suitable for us, we chose game programming systems, since they were developed to create virtual worlds, as well as physical interaction and object behavior. The system must support three-dimensional graphics. Among the programs that suit us according to the above criteria, our project team identified Unreal Engine, Unity 3D and CryEngine [11, 15, 31, 32]. Their pros and cons are presented in table 1.

TABLE I. GAME DEVELOPMENT ENVIRONMENT

System name	Programming language	Pros	Cons
Unreal Engine	C++	High-quality editor with great functionality. Visual Programming System Blueprints. Full access to the source code. Better visualization compared to competitors	Scanty documentation Slow compilation speed. The system is difficult to learn.
Unity 3D	C#	High compilation speed. Good documentation. The system is easy to learn.	Do not have access to the source code.
CryEngine	C++	Good visualization. Full access to source code.	Many errors in the engine code. Poor quality documentation.

As a result, we chose the Unreal Engine system as the most suitable for our tasks. Unreal Engine is a game software environment developed and supported by Epic Games [32].

As soon as we chose a development system, the question of choosing a programming language for writing code disappeared, since the Unreal Engine supports only C ++ code as an embedded code. This language has a high speed of calculating mathematical actions, which is very important for us.

In our program, to determine the behavior of particles, we use the Navier-Stokes equation already mentioned. In the numerical solution of the differential equation, the Euler method is used [1, 14]. In this method, to find the speed at the current time, the speed at the previous time is used. Space is divided into cells, each cell contains the speed value, the calculation for each cell is carried out separately.

To determine the particle motion, we use the algorithm that we developed for solving the Navier-Stokes equation [19]. Based on this algorithm, we have developed a library that performs basic mathematical operations for calculating vectors [18]. For visualization, we made a program that in a simplified form shows the distribution of particles in space. The speed spreads around the surrounding cells as follows: each cell exchanges with neighboring cells with a certain amount of speed, which depends on the value of viscosity or diffusion. The velocity transfer in the vertical direction can be represented as the displacement of the velocity value along the velocity vectors.

As a result of these calculations, the continuity equation will be violated [5]. To save it, you must use the property of the vector field. Every such field can be represented as the sum of the gradient and solenoidal fields [12]. As a result, the potential field needs to be subtracted from the current field to get an incompressible field.

The algorithm was written in C ++, and visualization in the program was done with such a tool as an open library.

We decided to make a subprogram that calculates the behavior of airflow, and built-in this subprogram into Unreal. The program was integrated into the development environment.

We had a choice of objects built into the game development system by engineers, and we tried to decide which of them to use for creation the objects simulating an air environment. Table 2 presents the approaches for creating objects:

TABLE II. PROS AND CONS OF TYPES FOR CREATING PARTICLES

Approach name	Pros	Cons
Particle System	Better visualization of air flows, as the system is a collection of particles	Huge resource cost. Inability to control each particle and set its characteristics.
Pawn class object	The ability to easily change the external part, by changing the material A wide range of built-in functions and systems The ability to create properties of particles and change them	Not the best visualization since it is necessary to combine the volume of air in the large particles and manage them

Basing on the comparison table, we decided to create particles as objects of the Pawn class with the control class Controller. After creating the particles and the logic of their behavior, it was necessary to create objects that generate and distribute airflows indoors, in our case, this is a ventilation system. We also created a 3D model of the room for testing the program.

V. PRACTICAL SIGNIFICANCE, PROPOSALS AND RESULTS OF EXPERIMENTAL STUDIES

The software implementation began in parallel with the research. At the same time, a subroutine was created using the Navier-Stokes system of equations in the Code:: Blocks environment, and a framework for the main program was created in the Unreal Engine environment.

In our program, to determine the behavior of particles, we use the Navier-Stokes equation that already mentioned. In the numerical solution of the differential equation, the Euler method is used, in which to find the speed at the current time, the speed at the previous time is used [13, 14, 24, 28]. This method was chosen due to the high speed of its calculation.

For writing the program, the C ++ programming language was used. The program can be divided into two modules. The first module performs calculations to find the speed values. The second one uses the results for visualization, and also performs user interaction. To create a window application and visualization, free libraries, Freeglut and OpenGL, were used.

As shown in Fig.1, a space of size N ^ 3 is divided into cells, each cell is described using density and velocity in three dimensions: u, v, w.

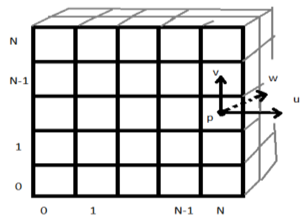


Fig. 1. Representation of space

Each cell exchanges a certain amount of speed with its neighbors, so the speed spreads in the system.

The input data for the program are the resolution of the space grid, time step, diffusion, viscosity, the value of the velocity added by the user, and the amount of substance. Arrays are also created to store density and velocity values in three directions for two points in time.

Since the three-dimensional arrays of elements have been initialized as one-dimensional, for easy access to the elements, macro was used (Listing 1).

LISTING 1. ONE-DIMENSIONAL ARRAY INITIALIZATION

```
#define IX3(i,j,k) ((i)+(N+2)*(j)+(N+2)*(N+2)*(k))
```

Also, a macro was used to change variables in places (Listing 2).

LISTING 2. CHANGE VARIABLES PROCEDURE.

```
#define SWAP(x0,x) {float * tmp=x0;x0=x;x=tmp;}
```

Fig. 2 shows the flow rate calculation scheme.

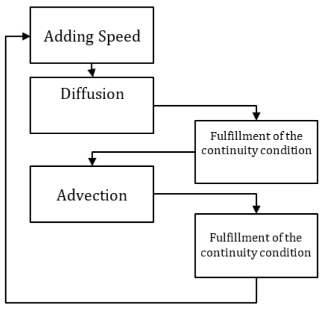

Adding Speed
Diffusion
Fulfillment of the continuity condition
Advection
Fulfillment of the continuity condition

Fig. 2. Calculation of the flow rate

The velocity field is calculated using the vel_step function (Listing 3).

LISTING 3. VELOCITY FIELD CALCULATION

```
void vel_step ( int N, float * u, float * v, float * w,
float * u0, float * v0,  float * w0, float visc, float dt )
{
    add_source ( N, u, u0, dt );
    add_source ( N, v, v0, dt );
    add_source ( N, w, w0, dt );
    SWAP ( u0, u ); diffuse3d ( N, 1, u, u0, visc, dt );
    SWAP ( v0, v ); diffuse3d ( N, 2, v, v0, visc, dt );
    SWAP ( w0, w ); diffuse3d ( N, 3, w, w0, visc, dt );
    project3d ( N, u, v, w, u0, v0 );
    SWAP ( u0, u );
    SWAP ( v0, v ); SWAP ( w0, w );
    advect3d ( N, 1, u, u0, u0, v0, w0, dt );
    advect3d ( N, 2, v, v0, u0, v0, w0,dt );
    advect3d ( N, 3, w, w0, u0, v0, w0,dt );
    project3d ( N, u, v, w, u0, v0 );
}
```

The density is calculated using the dens_step function (Listing 4).

LISTING 4. DENSITY CALCULATION

```
void dens_step ( int N, float * x, float * x0, float * u,
float * v, float * w, float diff, float dt )
{
    add_source ( N, x, x0, dt );
    SWAP ( x0, x ); diffuse3d ( N, 0, x, x0, diff, dt );
    SWAP ( x0, x ); advect3d ( N, 0, x, x0, u, v, w, dt );
}
```

The add_source function, using the interface, adds the values added by the user to the current density and speed indicators (Listing 5).

LISTING 5. DENSITY AND SPEED CHANGING

```
void add_source ( int N, float * x, float * s, float dt )
{
    int i, size=(N+2)*(N+2)*(N+2);
    for ( i=0 ; i<size ; i++ ) x[i] += dt*s[i];
}
```

Diffusion can be represented as the exchange of each cell with an adjacent amount of velocity or density, depending on the value of viscosity or diffusion (2).

$$u_i^{n+1} = u_i^n + \nu(u_{i+1}^n + u_{i-1}^n - k u_i^n), \qquad (2)$$

Where u is the value of speed or density,

n – time step,

i – cell number,

ν – diffusion coefficient,

k – number of neighboring cells.

Fig.3 shows such an exchange for one direction:

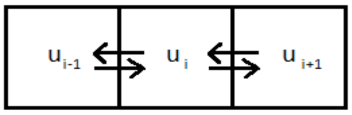


Fig. 3. The scheme of exchange between cells

However, such a formula (2) is unstable [26]. Therefore, the inverse Euler method is used to find the desired value (3), (4).

$$u_i^n = u_i^{n+1} - \nu(u_{i+1}^{n+1} + u_{i-1}^{n+1} - k u_i^{n+1}) \qquad (3)$$

$$u_i^{n+1} = (u_i^n + \nu(u_{i+1}^{n+1} + u_{i-1}^{n+1}))/(1 + k\nu), \qquad (4)$$

Where u – speed or density value,

n – time step,

i – cell number,

v – diffusion coefficient,

k – number of neighboring cells.

The distribution of matter and velocity through the surrounding cells occurs in the diffuse3d function (Listing 6):

LISTING 6. THE DISTRIBUTION OF MATTER AND VELOCITY

```
void diffuse3d ( int N, int b, float * x, float * x0, float
diff, float dt )
{
    float a=dt*diff*N*N*N;
    lin_solve3d ( N, b, x, x0, a, 1+6*a );
}
void lin_solve3d ( int N, int b, float * x, float * x0,
float a, float c )
{
    int i, j, k,l;
    for ( l=1 ; l<=4 ; l++ ) {
    for ( i=1 ; i<=N ; i++ ) {
for ( j=1 ; j<=N ; j++ ) {
for ( k=N ; k>1 ; k-- ) {
x[IX3(i,j,k)] = ( x0[IX3(i,j,k)] + a*( x[IX3(i-1,j,k)] +
x[IX3(i+1,j,k)] + x[IX3(i,j-1,k)] +
x[IX3(i,j+1,k)]+x[IX3(i,j,k-1)]+x[IX3(i,j,k+1)]) )/c;
    }}}
}
    set_bnd ( N, b, x );
}
```

Advection can be represented as a displacement of a substantial value along velocity vectors. Euler schemes for calculating the advection equation are unstable; therefore, the semi-Lagrangian approach was used. In contrast to the Lagrangian approach, where the finite trajectories of particles are distributed irregularly, in the semi-Lagrangian approach, the final trajectories will lie at the grid nodes [29]. Using formulas (5), (6), (7), the value of *d* is calculated, which moves along the velocity field *u* (Listing 7).

$$x_{\text{н}} = x_i - \Delta t u_i, \tag{5}$$

$$\alpha = x_{\text{н}} - x_j, \tag{6}$$

$$d_i^{n+1} = (1-\alpha)d_j^n + \alpha d_{j+1}^n, \tag{7}$$

Where $x_{\text{н}}$ – coordinate of the starting point of the trajectory on the interval [j, j + 1]

x_i – coordinate of the desired cell,

i, j – cell numbers,

Δt – time interval,

u_i – speed in the desired cell,

d – density or speed value,

n – time step.

The location of these values is shown in Fig.4.

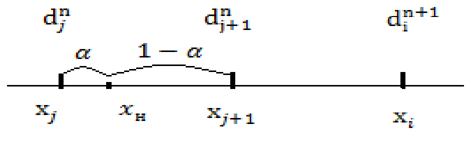


Fig. 4. Location of quantities

LISTING 7. THE CALCULATION OF THE ADVECTION EQUATION

```
void advect3d ( int N, int b, float * d, float * d0, float
* u, float * v, float *w, float dt )
{
    int i, j,k, i0, j0, i1, j1,k0,k1;
    float x, y,z, s0, t0, s1, t1,p0,p1, dt0;
    dt0 = dt*N;
    for ( i=1 ; i<=N ; i++ ) {
            for ( j=1 ; j<=N ; j++ ) {
                for ( k=1 ; k<=N ; k++ ) {
        x = i-dt0*u[IX3(i,j,k)]; y = j-dt0*v[IX3(i,j,k)];z
= k-dt0*w[IX3(i,j,k)];
        if (x<0.5f) x=0.5f; if (x>N+0.5f) x=N+0.5f;
i0=(int)x; i1=i0+1;
        if (y<0.5f) y=0.5f; if (y>N+0.5f) y=N+0.5f;
j0=(int)y; j1=j0+1;
        if (z<0.5f) z=0.5f; if (z>N+0.5f) z=N+0.5f;
k0=(int)z; k1=k0+1;
        s1 = x-i0; s0 = 1-s1;
        t1 = y-j0; t0 = 1-t1;
        p1 = z-k0; p0 = 1-p1;
        d[IX3(i,j,k)] =
p0*(s0*(t0*d0[IX3(i0,j0,k0)]+t1*d0[IX3(i0,j1,k0)])+
+s1*(t0*d0[IX3(i1,j0,k0)]+t1*d0[IX3(i1,j1,k0)]))+
+p1*(s0*(t0*d0[IX3(i0,j0,k1)]+t1*d0[IX3(i0,j1,k1)])+
+s1*(t0*d0[IX3(i1,j0,k1)]+t1*d0[IX3(i1,j1,k1)]));
}}}
    set_bnd ( N, b, d );
}
```

As a result of all these calculations, the continuity equation will be violated for the velocity field. To maintain the continuity condition, the vector field characteristic is used. Any vector field can be represented as the sum of the potential (gradient) and solenoidal (vortex) fields [33].

The divergence of the solenoidal field is zero, that means that this field corresponds to the continuity condition.

Thus, to obtain a vortex field, we need to find the field gradient, and then subtract it from the current field (Listing 8).

LISTING 8. THE FIELD GRADIENT FINDING

```
void project3d ( int N, float * u, float * v,float* w,
float * p, float * div )
{
    int i, j,k;
    for ( i=1 ; i<=N ; i++ ) {
for ( j=1 ; j<=N ; j++ ) {
for ( k=N ; k>1 ; k-- ) {
        for ( k=1 ; k<=N ; k++ ) {
        div[IX3(i,j,k)] = ( u[IX3(i+1,j,k)] - u[IX3(i-
1,j,k)] + v[IX3(i,j+1,k)] - v[IX3(i,j-1,k)] +
w[IX3(i,j,k+1)] - w[IX3(i,j,k-1)]) / (-3.f) /N;
        p[IX3(i,j,k)] = 0;
        }
    }
}
    set_bnd ( N, 0, div ); set_bnd ( N, 0, p );
    lin_solve3d ( N, 0, p, div, 1, 6 );
    for ( i=1 ; i<=N ; i++ ) {
for ( j=1 ; j<=N ; j++ ) {
    for ( k=1 ; k<=N ; k++ ) {
        u[IX3(i,j,k)] -= N*(p[IX3(i+1,j,k)]-p[IX3(i-
1,j,k)])/2.f;
        v[IX3(i,j,k)] -= N*(p[IX3(i,j+1,k)]-p[IX3(i,j-
1,k)])/2.f;
        w[IX3(i,j,k)] -= N*(p[IX3(i,j,k+1)]-p[IX3(i,j,k-
1)])/2.f;
    }
}
}
    set_bnd ( N, 1, u ); set_bnd ( N, 2, v );set_bnd ( N, 3,
w );
}
```

Next, we determine the interaction of particles with the boundaries of space. The behavior of the velocity in the boundary conditions is shown in Fig. 5 and Listing 9.

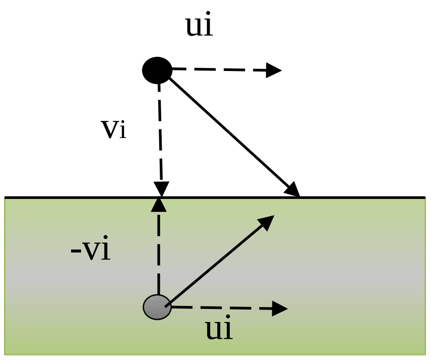


Fig. 5. Speed in boundary conditions

LISTING 9. THE BEHAVIOR OF THE VELOCITY IN THE BOUNDARY CONDITIONS

```
void set_bnd ( int N, int b, float * x )
{
    int i,j;
    for ( i=1 ; i<=N ; i++ ) {
           for ( j=1 ; j<=N ; j++ ) {
    x[IX3(0  ,i,j)] = b==1 ? -x[IX3(1,i,j)] : x[IX3(1,i,j)];
    x[IX3(N+1,i,j)] = b==1 ? -x[IX3(N,i,j)] : x[IX3(N,i,j)];
    x[IX3(i,0 ,j )] = b==2 ? -x[IX3(i,1,j)] : x[IX3(i,1,j)];
    x[IX3(i,N+1,j)] = b==2 ? -x[IX3(i,N,j)] : x[IX3(i,N,j)];
    x[IX3(i,j ,0 )] = b==3 ? -x[IX3(i,j,1)] : x[IX3(i,j,1)];
    x[IX3(i,j,N+1)] = b==3 ? -x[IX3(i,j,N)] : x[IX3(i,j,N)];
    }
x[IX3(i ,0,0)] = (x[IX3(i,1,0)]+x[IX3(i,0,1)])/2;
x[IX3(i ,N+1,0)] = (x[IX3(i,N,0)]+x[IX3(i,0,1)])/2;
x[IX3(i ,0,N+1)] = (x[IX3(i,1,N+1)]+x[IX3(i,0,N)])/2;
x[IX3(i ,N+1,N+1)] = (x[IX3(i,N,N+1)]+x[IX3(i,N+1,N)])/2;
x[IX3(0 ,i,0)] = (x[IX3(1,i,0)]+x[IX3(0,i,1)])/2;
x[IX3(N+1,i,0)] = (x[IX3(N,i,0)]+x[IX3(N+1,i,1)])/2;
x[IX3(0 ,i,N+1)] = (x[IX3(1,i,N)]+x[IX3(0,i,N)])/2;
x[IX3(N+1,i,N+1)] = (x[IX3(N,i,N+1)]+x[IX3(N+1,i,N)])/2;
x[IX3(0 ,0,i)] = (x[IX3(1,0,i)]+x[IX3(0,1,i)])/2;
x[IX3(0 ,N+1,i)] = (x[IX3(1,N+1,i)]+x[IX3(0,N,i)])/2;
x[IX3(N+1,0,i)] = (x[IX3(N,0,i)]+x[IX3(N+1,1,i)])/2;
x[IX3(N+1, N+1,i)] = (x[IX3(N,N+1,i)]+x[IX3(N+1,N,i)])/2;
    }
x[IX3(0  ,0  ,0  )] = (x[IX3(1 ,0  ,0 )]+x[IX3(0  ,1,0
)]+x[IX3(0  ,0 ,1 )])/3;
x[IX3(0  ,0  ,N+1)] = (x[IX3(1 ,0  ,N+1 )]+x[IX3(0
,1,N+1)]+x[IX3(0  ,0,N)])/3;
x[IX3(0  ,N+1,0  )] = (x[IX3(1 ,N+1,0 )]+x[IX3(0  ,N,0
)]+x[IX3(0  ,N+1,1 )])/3;
x[IX3(0  ,N+1,N+1)] = (x[IX3(1,N+1,N+1)]+x[IX3(0
,N,N+1)]+x[IX3(0  ,N+1,N )])/3;
x[IX3(N+1,0  ,0  )] = (x[IX3(N ,0 ,0  )]+x[IX3(N+1,1,0
)]+x[IX3(N+1,0 ,1 )])/3;
x[IX3(N+1,0  ,N+1)] = (x[IX3(N ,0
,N+1)]+x[IX3(N+1,1,N+1)]+x[IX3(N+1,0 ,N )])/3;
x[IX3( N+1, N+1, 0  )] = ( x[IX3( N , N+1, 0 )] + x[IX3(N+1
,N, 0  )] + x[IX3( N+1, N+1, 0 )]) /3;
x[IX3( N+1,N+1,N+1 )] = ( x[IX3( N, N+1, N+1 ) ] + x[IX3(
N+1,N , N+1 )] + x[IX3(N+1, N+1, N )] )/3;
}
```

After completing the work on software aimed on solving the Navier-Stokes system of equations by numerical methods, it was implemented in the main program.

Unreal Engine is a software environment in which the ability to inherit classes is implemented. The Actor object acts as the parent class (base). This object can be of any object (located in the scene) with which other classes are combined, which, in turn, act as derived classes, that is, they inherit various methods from the base. They also include the Pawn class. Most of all in this class we are interested in is its functionality, namely, the ability to control it through another Controller class. This feature allows the Controller class to pass information to the Pawn class, thereby controlling its behavior.

Particles of the Pawn class were created in our program. Then these particles were endowed with properties created based on the basic characteristics of the air environment, namely temperature, density, and humidity. This data are presented as floating-point numbers. A programming panel with these variables is shown in Fig. 6.

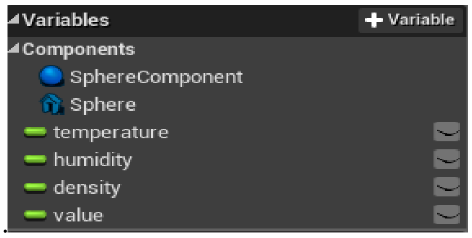
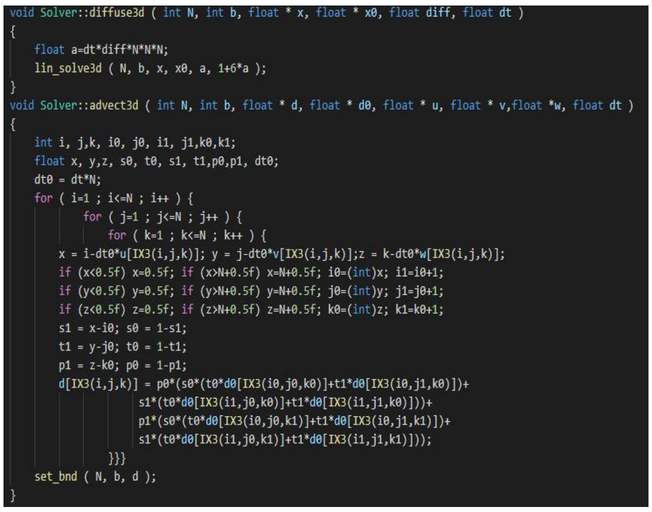


Fig. 6. Panel with variables in Unreal Engine

After that, a program was integrated into the software created in the Unreal Engine environment, inside of which a calculation algorithm was programmed. This algorithm is aimed at finding numerical solutions of the Navier-Stokes system of equations. The code section from the subroutine responsible for diffusion and advection and calculating the equations of motion for three-dimensional space is shown in Fig.7.



```
void Solver::diffuse3d ( int N, int b, float * x, float * x0, float diff, float dt )
{
    float a=dt*diff*N*N*N;
    lin_solve3d ( N, b, x, x0, a, 1+6*a );
}
void Solver::advect3d ( int N, int b, float * d, float * d0, float * u, float * v,float *w, float dt )
{
    int i, j,k, i0, j0, i1, j1,k0,k1;
    float x, y,z, s0, t0, s1, t1,p0,p1, dt0;
    dt0 = dt*N;
    for ( i=1 ; i<=N ; i++ ) {
            for ( j=1 ; j<=N ; j++ ) {
                for ( k=1 ; k<=N ; k++ ) {
        x = i-dt0*u[IX3(i,j,k)]; y = j-dt0*v[IX3(i,j,k)];z = k-dt0*w[IX3(i,j,k)];
        if (x<0.5f) x=0.5f; if (x>N+0.5f) x=N+0.5f; i0=(int)x; i1=i0+1;
        if (y<0.5f) y=0.5f; if (y>N+0.5f) y=N+0.5f; j0=(int)y; j1=j0+1;
        if (z<0.5f) z=0.5f; if (z>N+0.5f) z=N+0.5f; k0=(int)z; k1=k0+1;
        s1 = x-i0; s0 = 1-s1;
        t1 = y-j0; t0 = 1-t1;
        p1 = z-k0; p0 = 1-p1;
        d[IX3(i,j,k)] = p0*(s0*(t0*d0[IX3(i0,j0,k0)]+t1*d0[IX3(i0,j1,k0)])+
                    s1*(t0*d0[IX3(i1,j0,k0)]+t1*d0[IX3(i1,j1,k0)]))+
                    p1*(s0*(t0*d0[IX3(i0,j0,k1)]+t1*d0[IX3(i0,j1,k1)])+
                    s1*(t0*d0[IX3(i1,j0,k1)]+t1*d0[IX3(i1,j1,k1)]));
                }}}
    set_bnd ( N, b, d );
}
```

Fig. 7. Part of the code responsible for diffusion and advection

The following algorithm is executed in the subroutine:

- a three-dimensional array is created;
- an air particle is placed in each cell of the array;
- the velocity vector is calculated for each particle based on its current characteristics. In this case, the particles surrounding it are also taken into account.

The subroutine receives input data arrays with current values of the characteristics of the particles.

The output data of the subroutine are data arrays with finite vectors, that were obtained as a result of the above algorithm. After that, the data obtained are assigned to the particles in the form of a pulse.

The originally created subroutine performed calculations in 2D form. In the same format, it also performed visualization, that, in turn, could work in two modes: density display mode and velocity field display mode. After painstaking work with the code, the system was able to be converted into 3D format. But the subroutine worked with some inaccuracies. The program code was gradually refined, and the accuracy of calculations and visualization improved. Fig.8 shows the visualization of speed in the form of a vector field.

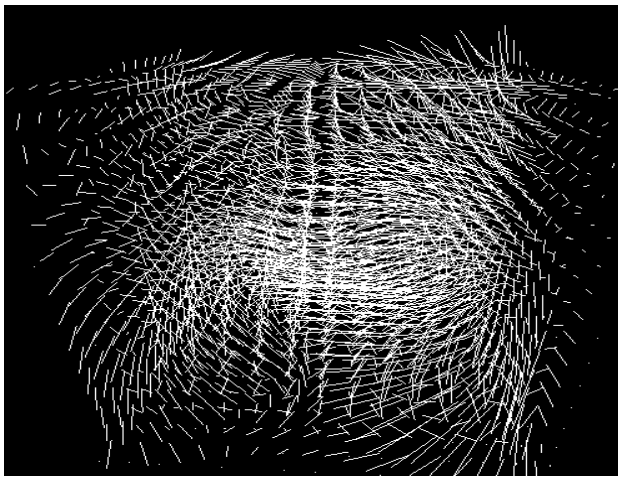


Fig. 8. The final version of the visualization in 3D format (speed field display mode)

Code for drawing vectors:

LISTING 10. VECTORS DRAWING

```
glVertex3f (x, y ,-z);
glVertex3f (x+u[IX3(i,j,k)], y+v[IX3(i,j,k)],-
(z+w[IX3(i,j,k)]));
```

The density display mode of the final version of the program is shown in Fig.9.

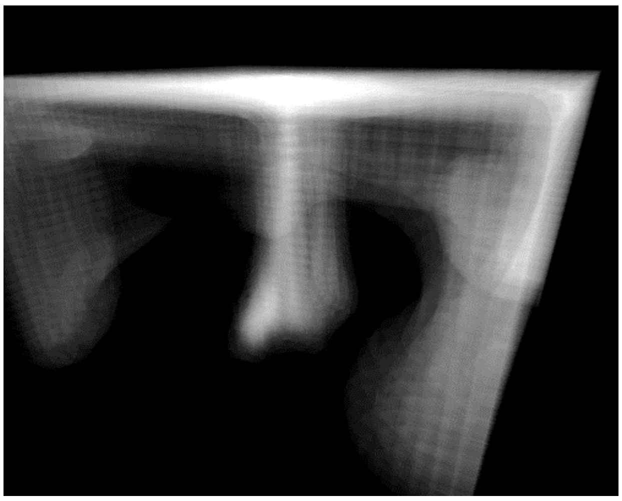


Fig. 9. The final version of visualization in 3D format (density display mode)

Density is drawn in the form of white planes that forms multiple cubes. The transparency at the nodes of the cube depends on the values of the calculated density. To test the program, a 3D model of the room was created. For realism, scaling was performed during its construction. The model was created directly in the Unreal Engine. Fig.10 shows the created model of the room for poultry-breeding production [35].

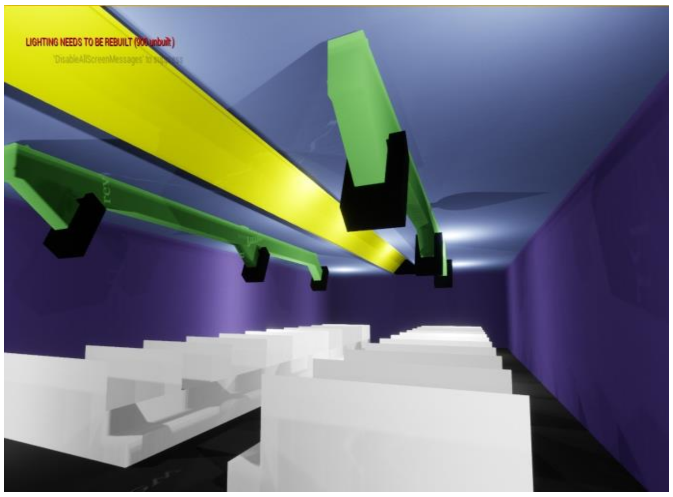

LIGHTING NEEDS TO BE REBUILT (900 unbuilt)
'DisableAllScreenMessages' to suppress

Fig. 10. 3D model of the room

The user’s interaction with the software, at the moment, is done through a text block, where the user can enter a number that is responsible for the temperature of the air entering the room via the ventilation system. Fig. 11 shows the mapping of airflow particles in an Unreal Engine environment.

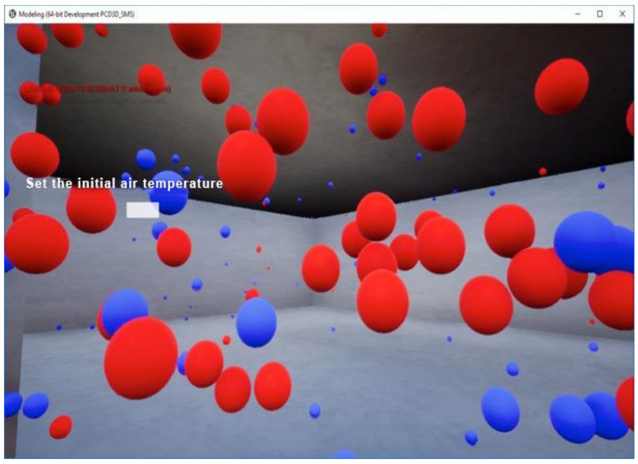

Modeling (64-bit Development PCD3D_SM5)
Set the initial air temperature

Fig. 11. Display of airflow particles in the Unreal Engine environment

The program user can set the initial air temperature. If it was not done, the temperature is set by default to 16 degrees Celsius. Also, there were implemented the visualization for the exchange of airflows particles by physical characteristics and the visualization for the airflows around objects inside a room model. Interface management is carried out with the use of keyboard and mouse. Using the right mouse button, density is added; and using the left mouse button, a speed pulse is created. The cube can also be rotated using the keys. The program can switch between two display modes: density and velocity field.

VI CONCLUSION

Modeling allows predicting the behavior of objects in the system, obtaining new data on the properties of the object, and simplifies the systematization of the data known. At present, the creation of a proper indoor microclimate is important in various spheres of human life and industries. One of the ways to control the microclimate in enclosed space is to regulate the temperature and speed of the airflow in supply, exhaust ventilation or control airflow from air conditioning systems. This work aims to create an application program for modeling the behavior of airflows in industrial premises. To complete the work and achieve our goal, we:

- studied the software that available on the market and appropriate for simulating the behavior of airflows;
- studied the physical and mathematical formulas and laws associated with the behavior of air, for their implementation in software;
- implemented software for the numerical solution of the Navier-Stokes system of equations;
- studied and mastered the Unreal Engine application software;
- modeled and combined software aimed at solving the Navier-Stokes system of equations with a program implemented in Unreal Engine;
- established the interaction of objects with the outside world and among themselves based on the Unreal Engine environment.

In the result of our work, we created application software for simulation of airflows in enclosed spaces that can be effectively used in various fields of the economy.

We plan to continue the development of the program. At the next stage, we will implement more accurate relationships between airflow particles and the environment, as well as introduce functions related to pressure and humidity into the program, and develop more convenient user interface.

ACKNOWLEDGMENT

P.V.Y. thanks to Sergei V. Menschikov and Svetlana V. Parshkova for the cooperation in research and for assistance in providing technical documentation.

REFERENCES

[1] “Adams method”, https://www.cfd-online.com/Wiki/Adams_methods.

[2] “ADINA”, http://www.adina.com/

[3] “ANSYS CFD – Computational fluid dynamics”, CADFEM, https://www.cadfem-cis.ru/products/ansys/fluids/

[4] “Catalogue. SolidWorks Flow Simulation”, https://www.syssoft.ru/SolidWorks/SolidWorks-Flow-Simulation/

[5] “Cauchy momentum equation”, https://en.wikipedia.org/wiki/Cauchy_momentum_equation.

[6] “Comparing CFD Software. RESOLVED analytics”, https://www.resolvedanalytics.com/theflux/comparing-cfd-software.

[7] “Computational fluid dynamics”, https://en.wikipedia.org/wiki/Computational_fluid_dynamics.

[8] “Computational fluid dynamics. Handbook24”, https://spravochnick.ru/fizika/mehanika_sploshnyh_sred/vychislitelnaya_gidrodinamika/

[9] “Computational Fluid Dynamics Solver – Altair AcuSolve”, https://altairhyperworks.com/product/acusolve.

[10] “Continuity equation”, https://en.wikipedia.org/wiki/Continuity_equation

[11] “CryEngine”, https://ru.wikipedia.org/wiki/ CryEngine.

[12] “Elements of field theory. Mathematical analysis”, http://www.314159265.ru/images/MA_conspects/Field_theory.pdf.

[13] “Equation of motion of a continuous medium”, https://en.wikipedia.org/wiki/Cauchy_momentum_equation.

[14] “Euler method”, https://en.wikipedia.org/wiki/Euler_method

[15] “FAQ: Licensing and activation. Unity”. https://unity3d.com/ru/unity/faq/2491.

[16] “Features of the numerical modeling of air flows in concert and theatre halls”, http://mm-technologies.ru/wp-content/uploads/2016/12/osobennosti-chislennogo-modelirovaniya-povedeniya-vozdushnyx-potokov-v-obemax-koncertnyx-i-teatralnyx-zalov.pdf.

[17] “Fluid simulation. Computer science”, https://www.cs.ubc.ca/~rbridson/fluidsimulation/fluids_notes.pdf.

[18] “Freeglut”, https://ru.wikipedia.org/wiki/Freeglut.

[19] S.N. Kharlamov, “Algorythms in modelling the hydrodynamics processes”, Tomsk, 2008, p.80.

[20] G.Lamb, “Hydrodynamics”, Russia, Moscow, 1947, http://log-in.ru/books/lamb-g-gidrodinamika-lamb-g-nauka-i-obrazovanie/

[21] L.D.Landau, E.M.Lifshitz, “Hydrodynamics”, Moscow, 1986, https://www.twirpx.com/file/891058/

[22] “Lecture 13. Numerical solution of differential equations’, https://toehelp.ru/theory/informat/lecture13.html.

[23] “Millennium Prize Problems—Navier–Stokes Equation”, Clay Mathematics Institute, http://www.claymath.org/millennium-problems/navier–stokes-equation.

[24] “Navier-Stokes equations”, https://en.wikipedia.org/wiki/Navier–Stokes_equations.

[25] “Newtonian fluid”, https://en.wikipedia.org/wiki/Newtonian_fluid.

[26] “Real-Time Fluid Dynamics for Games. Dynamic Graphics Project”, http://www.dgp.toronto.edu/people/stam/reality/Research/pdf/GDC03.pdf.

[27] Sanitary Regulations and Norms of Russia. Hygienic requirements for microclimate of industrial premises, http://docs.cntd.ru/document/901704046

[28] L.I.Sedov, “Continuum mechanics”, Moscow, Science, 1970, https://www.twirpx.com/file/67973/

[29] M.A.Tolstykh, “Semi-Lagrangian method”, Siberian center for Environmental Research and Training, http://www.scert.ru/conferences/cites/2007/presentation/Presentation/School/tolstykh.pdf.

[30] O.G. Turovets, “Organization of production and management. Manual”, Moscow, 2007, p. 544.

[31] «Unity3D. Start of work, practical advices. Review”, https://habr.com/ru/post/161463/

[32] «Unreal Engine», https://ru.wikipedia.org/wiki/Unreal_Engine

[33] Y.M.Volchenko, “Potentiality and solenoidality”, Maths for technical universities, https://math.volchenko.com/Lectures/PotField.pdf.

[34] P.V.Yudin, “Environmental resource saving biogas production for the poultry breeding farm on the example of “Mikhailovsky broiler”, DOI: 10.1109/EastConf.2019.8725381.

[35] P.V.Yudin, “The math and economy methods improvement in calendar planning for the poultry production”, research dissertation, Vladivostok, Russia, 2004.