ВИЗУАЛИЗАЦИЯ РЕЗУЛЬТАТОВ МОДЕЛИРОВАНИЯ ТЕЧЕНИЙ ГАЗОВ И ЖИДКОСТЕЙ. ЧАСТЬ 1. АНИМАЦИЯ ГРАФИКА ПОВЕРХНОСТИ.

Маркин Е. Е. Уральский государственный университет путей сообщения, Екатеринбург, Студент.
Скачков П. П. Уральский государственный университет путей сообщения, Екатеринбург, доцент кафедры естественно научных дисциплин.

УДК 519.6

Аннотация. Целью работы является разработка программы для анимации процессов в течениях газа. Решения полной системы уравнений Навье Стокса, описывающей двухмерные течения сжимаемого вязкого теплопроводного идеального газа, позволяют определить все термодинамические переменные таких течений. Для анализа результатов вычислений полезно рассмотрение визуализации поверхностей этих переменных в трехмерном пространстве и анимация этих поверхностей по времени протекания процесса.

Существенной особенностью программы является то, что она непосредственно взаимодействуя с массивом данных позволяет сдвигать, менять масштаб и поворачивать график в процессе анимации. Все эти действия можно с помощью манипулятора мышь, а также используя удобный интерфейс. График поверхности строиться в параллелепипеде с размерами (x1,x2), (y1,y2), (z1,z2), а повороты графика можно проводить на углы $\alpha = \angle(Oz, Oz')$ и $\beta = \angle(Ox, Ox')$.

Ключевые слова: система уравнений навье-стокса, график поверхности, преобразование системы координат, ASP.NET C#.

Abstract. The aim of the work is the development of a program for the animation of processes in gas flows. The solutions of the complete system of Navier-Stokes equations describing two-dimensional flows of a compressible viscous heat-conducting ideal gas make it possible to determine all the thermodynamic variables of such flows. To analyze the results of computations, it is useful to consider the visualization of the surfaces of these variables in three-dimensional space and the animation of these surfaces along the time of the process flow.

An essential feature of the program is that it directly interacts with the array of data allows you to shift, zoom and rotate the graph during the animation. All these actions can be done using the mouse, and using a user-friendly interface. The plot of the surface is constructed in a parallelepiped with dimensions (x1, x2), (y1, y2), (z1, z2), and rotations of the graph can be drawn at angles $\alpha = \angle(Oz, Oz')$ and $\beta = \angle(Ox, Ox')$.

Keywords: navier-stokes equations system, surface graph, coordinate system transformation, ASP.NET C#.

Введение

Рассматривается полная система уравнений Навье Стокса, решения которой описывают течения сжимаемого вязкого теплопроводного идеального газа [1]. Система дифференциальных уравнений для переменных и программа их вычисления содержатся в работе [4]. Результатом работы программы являются массивы, записываемые в файлы данных[2]. Эти массивы содержат значения давления, удельного объема и скоростей. Файлы открываются в ППП MathCAD (MatLab) и произвести анимацию процессов протекающих в течении газа[3].

Однако, использование программ MathCAD'а не позволяет преобразование системы координат в процессе анимации, что является существенным недостатком. Так, например, если необходимо повернуть график или изменить масштаб, то нужно заново войти в ППП, внести изменения в настройки и провести анимацию, сохраняя в виде видеоролика.

В дальнейшем планируется построить программу для прямой анимации графика векторного поля скоростей.

Постановка задачи

Разработать приложение, которое выводит на экран монитора график поверхности с возможностью поворачивать график, выводить оси координат OX, OY, OZ и легко перемещаться в разные моменты моделируемого времени

Математическая постановка задачи

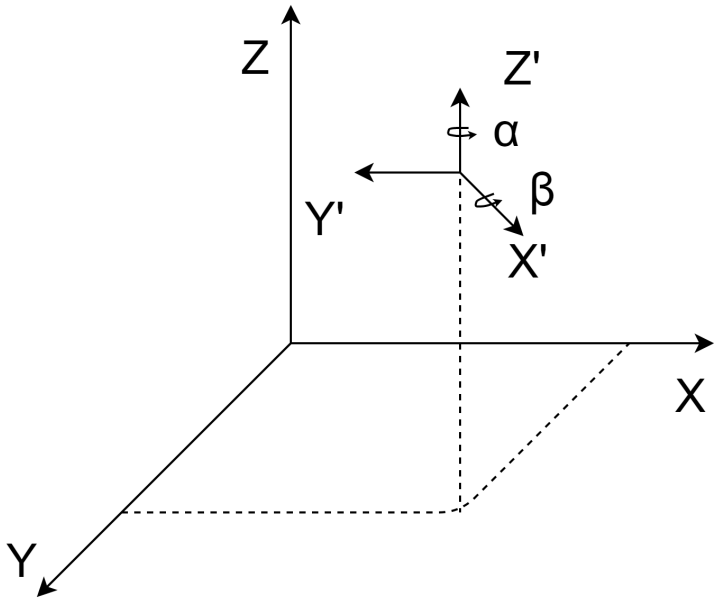

Рис. 1. Поворот и параллельный сдвиг системы координат.

График поверхности будет строиться в параллелепипеде с размерами (xx1,xx2), (yy1,yy2), (zz1,zz2). Для поворота системы координат введем понятие точки с координатами (x0,y0,z0), а также углов относительно которых будет осуществляться поворот всей системы, α – относительно оси OZ и β – относительно оси OX.

Для решения данной задачи воспользуемся формулами для вычисления (1). Сдвиг в точку с учетом углов α и β описывается матрицами преобразования координат

$$S' = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad S'' = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\beta & \sin\beta \\ 0 & -\sin\beta & \cos\beta \end{pmatrix}, \quad S''' = \begin{pmatrix} x - x_0 \\ y - y_0 \\ z - z_0 \end{pmatrix}. \qquad (1)$$

После перемножения матриц получим формулу для вычисления .

Обозначим, что в плоскости монитора лежат оси OX и OY, а ось OZ перпендикулярна экрану. Пусть переменные xx и yy обозначают точку на экране монитора, тогда (xx, yy) искомые точки графика будут найдены по формуле (2)

$$xx = \frac{x}{\frac{z}{A} + 1}, \quad yy = \frac{y}{\frac{z}{a} + 1}. \qquad (2)$$

A, a – коэффициенты перспективы, находятся экспериментально.

Написание программы

Создадим форму и расположим на ней элементы, показанные на рисунке 2. На форме главным образом располагаются поля для ввода размеров слоя (N, M), время моделирования

(TK), и частота кадров (FPS). После открытия файла данные поля отключаются, а вкладки активируются, на них можно менять область определения графика поверхности, регулировать масштабы осей и регулировать подписи[6].

Для работы с графиком поверхности объявим переменные, показанные на листинге 1. Переменные $X\_\min$, $X\_\max$, $Y\_\min$, $Y\_\max$, обозначают область определения графика поверхности, значения из массива равномерно заполняют поверхность с шагом h_n и h_m, где $h\_n = (X\_\max - X\_\min)/N$; $h\_m = (Y\_\max - Y\_\min)/M$. (4)

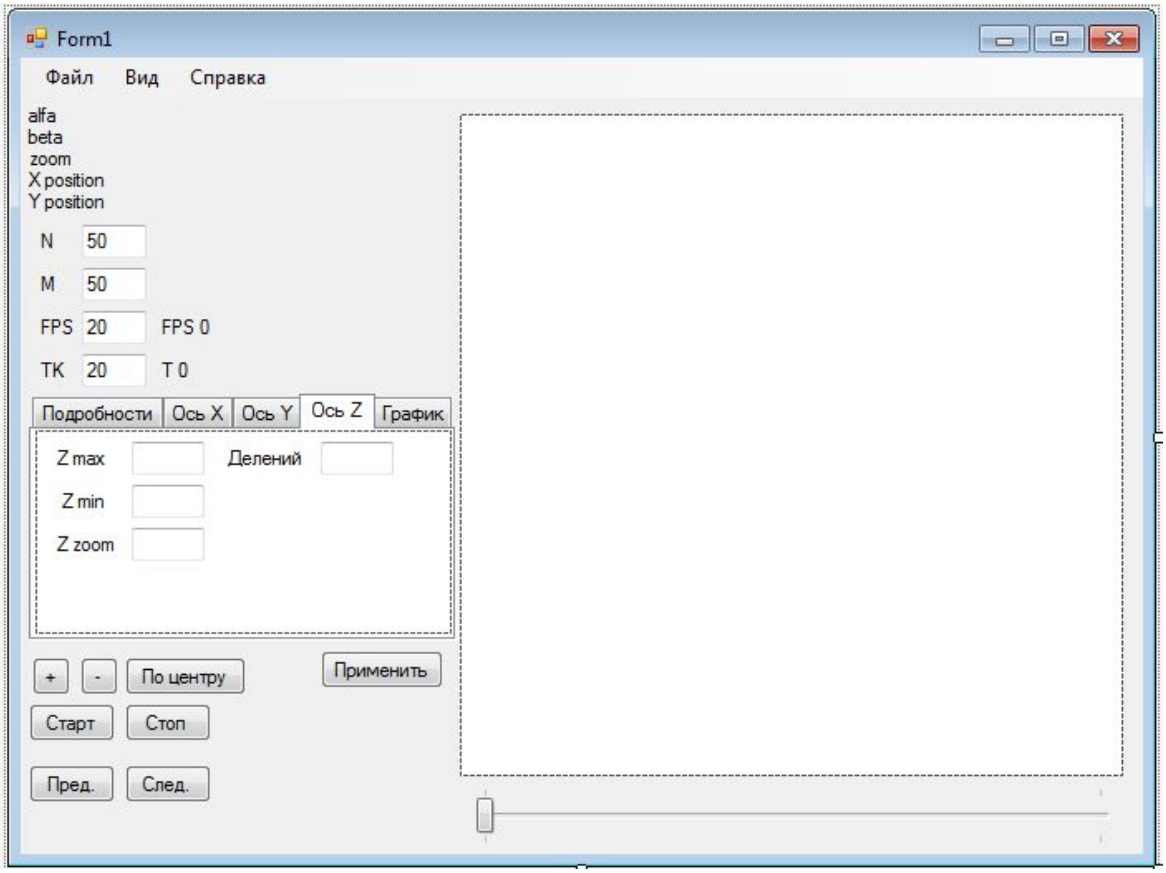

Рис. 2. Главное окно программы.

Пользователь должен вводить размеры моделируемой области (N,M), время моделирования (TK), область определения графика поверхности по осям OX, OY, OZ, и масштабирования графика в целом, так и по осям.

```
float X_min, Y_min, X_max, Y_max, Z_min, Z_max,
                    //Область определения функции
alfa, beta, // Углы обзора
ZOOM, Z_zoom,X_zoom,Y_zoom, //Масштаб общий, по осям OX,OY,OZ
x0, y0, z0, // Точка пересечения осей
XP, YP, // Смещение системы каординат при выводе на монитор
A, // коэфициент перспективы
mouse_speed;// скорость поворота систем координат
```

```
int yy, xx,//промежуточные значения
    xx1, xx2, yy1, yy2, //соответствуют координатам точек,
                        //которые отображаются на экране монитора
    C,FPS,//Количество кадров и частота кадров анимации
    x_del,y_del,z_del,// количество делений по осям
    N, M; // размеры выводимой поверхности
float h_n, h_m;//Шаг при построении поверхности

float[, ,] mas;// Массив элементов
```

Листинг 1. Объявление переменных

В программе обьявляются функции, перечислим основные из них:

- `void Zoom_XYZ(float x, float y, float z)`

  Функция для преобразования координат и масштабирования

- `void pictureBox_Paint(object sender, PaintEventArgs e)`

  Функция для рисования графика поверхности.

- `open_file(bool singl){`

  Функция чтения данных из файла

- `void timer1_Tick(object sender, EventArgs e)`

  Функция анимирования графика поверхности

Переменным N, M присваиваем значения с формы. Количество кадров (переменная C) определяется TK x FPS (время моделирования умножить на колличество кадров в секунду). Этих данных достаточно для загрузки массива в память.

На листинге 2 показан отрывок из функции заполнения массива данными из файла. Обратим внимание что изначально файл можно представить как одномерный массив, наша задача конвертировать его в трехмерный с индексами c,i,j, где индекс $c \in (0,C)$ номер кадра, индексы $i \in (0,N)$ и $j \in (0,M)$ соответствуют элементу слоя.

```
mas = new float[C, N, M];
FileStream fbr = new FileStream(file, FileMode.Open, FileAccess.Read);
BinaryReader bin = new BinaryReader(fbr);//открываем файл в потоке
int c = 0, i = 0, j = 0;
bool per = false;
while (true)
{
    try
    {
        mas[c, i, j] = bin.ReadSingle();
    }
    catch { per = true; }
```

```
    if ((M == j) && (N == i))
    {
        c++;
        i = 0; j = -1;
    }
    if (M == j) { i++; j = -1; }
    j++;
    if (per) break;
}
```

Листинг 2. Фрагмент функции импорта даных.

Теперь массив находится в памяти. Следующим шагом будет написание функции преобразования координат графика поверхности(x,y,z) в каординаты точки на мониторе(xx,yy). Функция приведена полностью на листинге 3. Переменные xx и yy являются глобальными, после вызова функции Zoom_xyz() считываем с них значение.

```
void Zoom_XYZ(float x, float y, float z)
{
    x *= X_zoom;
    y *= Y_zoom;
    z *= Z_zoom;

    float tx, ty, tz,
    xn, yn;

    tx = (x - x0) * Math.Cos(alfa) - (y - y0) * Math.Sin(alfa);
    ty = ((x - x0) * Math.Sin(alfa) + (y - y0) *
        Math.Cos(alfa)) * Math.Cos(beta) - (z - z0) * Math.Sin(beta);
    tz = ((x - x0) * Math.Sin(alfa) + (y - y0) *
        Math.Cos(alfa)) * Math.Sin(beta) + (z - z0) * Math.Cos(beta);
    xn=ZOOM * tx / (tz / A + 1) + XP;
    yn=ZOOM * ty / (tz / A + 1) + YP;

    xx=Convert.ToInt32(Math.Floor(pictureBox1.Width *
        (xn - X_min) / (X_max - X_min) + 0));
    yy=Convert.ToInt32(Math.Floor(pictureBox1.Height *
        (yn - Y_max) / (Y_min - Y_max) + 0));
}
```

Листинг 3. Преобразование координат.

И последняя основная функция приведена на листинге 4. В ней отрисовывается график поверхности. Весь слой делится на прямоугольники которые будут выводиться на экран по отдельности, ширина и длина прямоугольников определяется по формуле (4). Так же в этой функции отрисовываются оси, выводится надписи и тд..

```
private void pictureBox_Paint(object sender, PaintEventArgs e)
{
Graphics g=e.Graphics;
Point[] point=new Point[4];
g.Clear(Color.White);// очищение холста (канвы)
```

```
for (i=0; i < N-1; i++)//всего прямоугольников N*M
    for (j=0; j < M-1; j++)
    {
        Zoom_XYZ(h_n * i, h_m * j, mas[cadr, i, j]);
        point[0].X=xx;//вычисляем координаты четырех точек
        point[0].Y=yy;//которые определяют четырехугольник
        Zoom_XYZ(h_n * i, h_m + h_m * j, mas[cadr, i, j+1]);
        point[1].X=xx;
        point[1].Y=yy;
        Zoom_XYZ(h_n + h_n * i, h_m + h_m * j, mas[cadr, i+1, j+1]);
        point[2].X=xx;
        point[2].Y=yy;
        Zoom_XYZ(h_n + h_n * i, h_m * j, mas[cadr, i+1, j]);
        point[3].X=xx;
        point[3].Y=yy;
        g.DrawPolygon(Pens.Black, point);//выводим на экран
    }
```

Листинг 4. Отрывок функции вывода графика поверхности на экран

Функция pictureBox_Paint() является методом события Paint, обьекта PictureBox, расположенного на форме. Это нужно для того чтобы изображение не исчезало при перекрытии другми оконами или изменении размеров.

Теперь наш график ввыводится на экран однако нам нужно его поворачивать перемещать и масштабировать. Для этого настроим перемещение на правую кнопку мыши, поворот на левую, а масштабирование на колесико мыши. В событиях на обьекте PictureBox добавим метод MouseMove(перемещение мыши по обьекту). В нем определяем какая кнопка мыши нажата. На листинге 5 приведен отрывок кода для поворота системы координат, перемещение осуществляется подобным образом.

```
private void pictureBox_MouseMove(object sender, MouseEventArgs e){
    if (e.Button == MouseButtons.Left)
    {
        if (MousePosition.X - mouse_x < 0)
            alfa -= mouse_speed;//влево
        if (MousePosition.X - mouse_x > 0)
            alfa += mouse_speed;//вправо
        if (MousePosition.Y - mouse_y < 0)
            beta -= mouse_speed;//вверх
        if (MousePosition.Y - mouse_y > 0)
            beta += mouse_speed;//вниз
    }
    /* код программы
    mouse_x=MousePosition.X;
    mouse_y=MousePosition.Y;
    pictureBox1.Invalidate();
}
```

Листинг 5. Отрывок функции обработки поворота и перемещения графика.

Для того что бы настроить колесико мыши на масштабирование графика поверхности необходимо определить метод для события MouseWheel. Делается это в функции инициализации компонентов

```
    InitializeComponent();
    this.MouseWheel += new MouseEventHandler(this_MouseWheel);
}
void this_MouseWheel(object sender, MouseEventArgs e)
{
    if (e.Delta > 0)
    {
        ZOOM += 0.1F;//приближаем
    }
    else
    if (e.Delta < 0)
    {
        ZOOM -= 0.1F;//отдаляем
    }
    pictureBox1.invalidate();
}
```

Листинг 6. Добавление метода прокрутки колесика мыши.

И последнее что нужно сделать для анимации - добавить таймер, в нем изменять номер слоя и обновлять изображение (листинг 7).

```
private void timer_an_Tick(object sender, EventArgs e)
{
    if (cadr == C - 1) { timer1.Enabled=false; return; }
    cadr++;
    pictureBox1.Invalidate();
}
```

Листинг 7. Таймер анимации

Теперь можно запустить приложение на выполнение и протестовать его работу (рисунок 3).

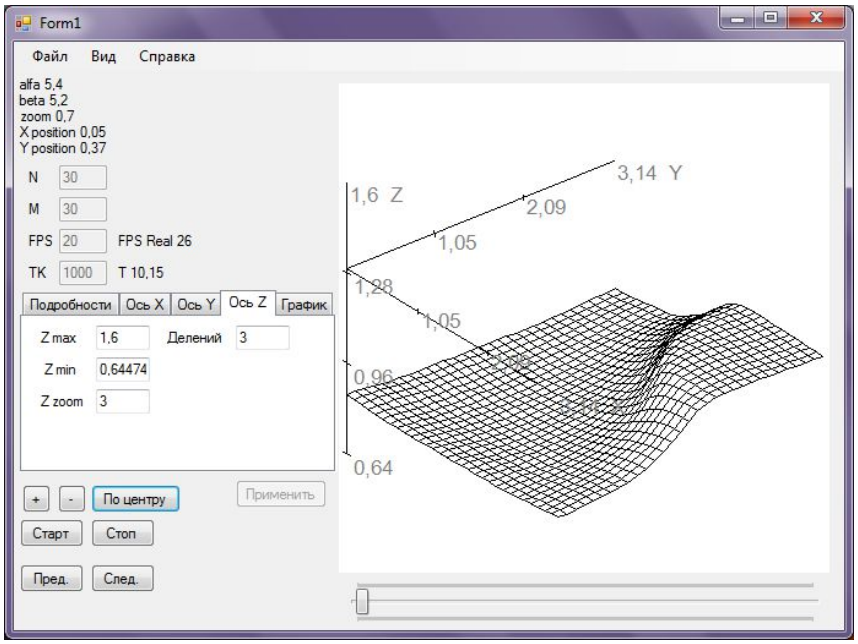

Рис. 3 Результата работы программы.

Список литературы

1. Баутин С. П.. Замыслов В. Е. Об одном представлении приближенных решений полной системы уравнений Навье-Стокса // Проблемы прикладной математики и механики. – Екатеринбург: УрГУПС.– 2011. Вып. 95(178)/6м.– С. 5 – 16.
2. Боресков А. В. Предисл.: Садовничий В. А. “Параллельные вычисления на GPU. Архитектура и программная модель CUDA”: Учебное пособие. Издательство Московского университета, 2012 Перепелт, 336 стр. ISBN: 978-5-211-06340-2
3. Кирьянов Д. В. “Mathcad 15 / Mathcad Prime 1.0” Год 2012/ Издательство БХВ-Петербург.
4. Маркин Е.Е., Скачков П.П. Гетерогенные параллельные вычисления на примере решения полной системы уравнений Навье-Стокса методом сеток. Международный студенческий научный вестник. Выпуск 5. 2017
5. 5. Эндрю Троелсен. “Pro C# 5.0 and the .NET 4.5 Framework” Год издания: 2012 ISBN: 978-1430242338 (en), 978-5-8459-1957-1 (ru).