Программа разделена на две части: на первом этапе выполняется настройка, на втором анализ потока и воспроизведение результата. Описательный алгоритм программы представлен на блок-схеме (приложение 1).

1. Класс Form1

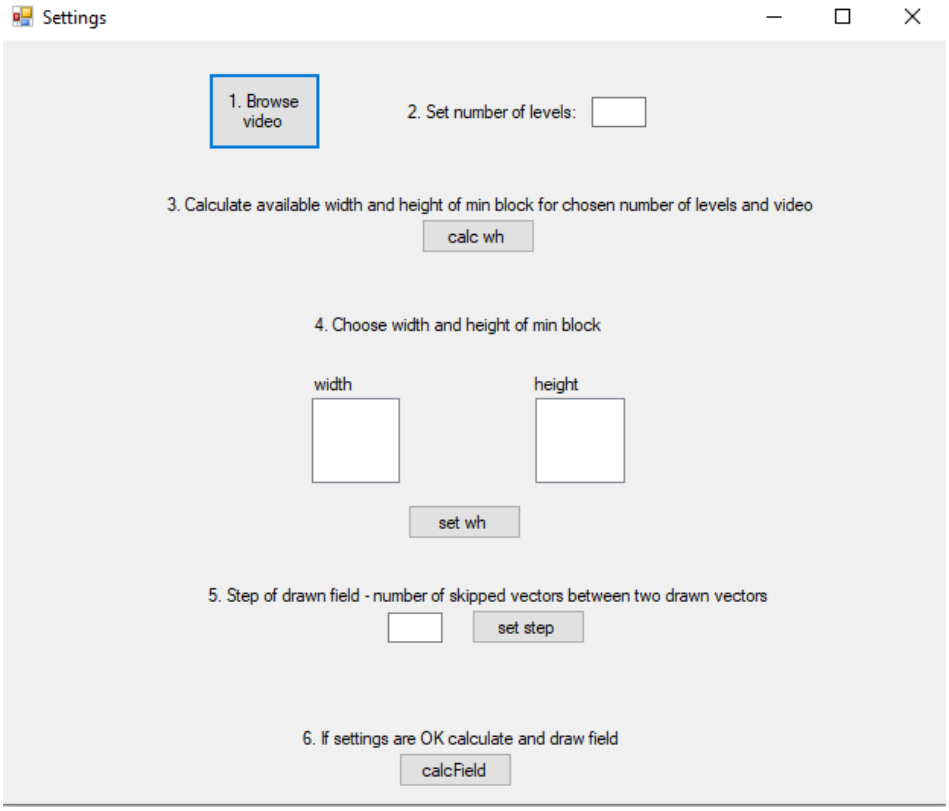


Рис. 1 – Класс Form1

Класс предназначен для установки настроек программы (рис. 1): выбор файла, количество слоев, высота и ширина минимального блока, шаг, по которому будет рисоваться векторное поле.

1. Метод button1_Click (Browse video)

```
private void button1_Click(object sender, EventArgs e)
{
    OpenFileDialog ofd = new OpenFileDialog();
    ofd.Filter = "Video Files (*.mp4, *.flv)| *.mp4;*.flv";
    if (ofd.ShowDialog() == DialogResult.OK)
    {
        capt = new VideoCapture(ofd.FileName);
    }

}
```

При нажатии на button1 открывается окно для выбора файла и создается объект capt этого файла.

2. Метод button3_Click (calc wh)

```
private void button3_Click(object sender, EventArgs e)
{
    int height, width, widthIntSmall, heightIntSmall;
    levels = int.Parse(textBox1.Text);
    Mat frame = new Mat();
```

```
    capt.SetCaptureProperty(Emgu.CV.CvEnum.CapProp.PosFrames, 0);
    capt.Read(frame);

    height = frame.Height;
    width = frame.Width;
    widthIntSmall = (int)Math.Ceiling(width / Math.Pow(2, levels - 1));
    heightIntSmall = (int)Math.Ceiling(height / Math.Pow(2, levels - 1));

    if (listBox1.Items.Count != 0)
    {
        listBox1.Items.Clear();
        listBox2.Items.Clear();
    }
    List<int> dividers = FindDiViders(widthIntSmall);
    for (int i = 0; i < dividers.Count; i++)
        listBox1.Items.Add(dividers[i]);
    dividers = FindDiViders(heightIntSmall);
    for (int i = 0; i < dividers.Count; i++)
        listBox2.Items.Add(dividers[i]);

}
```

При нажатии на button3 для выбранного значения levels рассчитывается ширина и высота изображения, уменьшенного в $2^{levels-1}$ раз, то есть изображения слоя L_m на основе чего составляется список допустимых широт и высот минимального блока.

3. Метод FindDividers

```
private List<int> FindDividers(int wh)
{
    var list = new List<int>();
    for(int i = wh; i > 1; i--)
    {
        if (wh % i == 0)
            list.Add(i);
    }
    return list;
}
```

Метод предназначен для нахождения делителей ширины или высоты изображения слоя L_m, то есть список list будет содержать допустимые значения широты и высоты минимального блока.

4. Метод button4_Click (set wh)

```
private void button4_Click(object sender, EventArgs e)
{
    xywh[0, 0] = 0;
    xywh[0, 1] = 0;
    xywh[1, 0] = Convert.ToInt32(listBox1.SelectedItem); // ширина мин блока
    xywh[1, 1] = Convert.ToInt32(listBox2.SelectedItem); // высота мин блока

}
```

button4 подтверждает выбранные значения ширины и высоты минимального блока.

5. Метод button5_Click (set step)

```
private void button5_Click(object sender, EventArgs e)
{
    drawVector = int.Parse(textBox2.Text);
}
```

button5 присваивает значение переменной drawVector, которое читает из textBox2.

6. Метод button2_Click (calcField)

```
private void button2_Click(object sender, EventArgs e)
{
    this.Hide();
    Form2 frm2 = new Form2();
    frm2.ShowDialog();
    this.Close();

}
```

Метод закрывает окно Settings и открывает окно Video.

2. Класс Form2

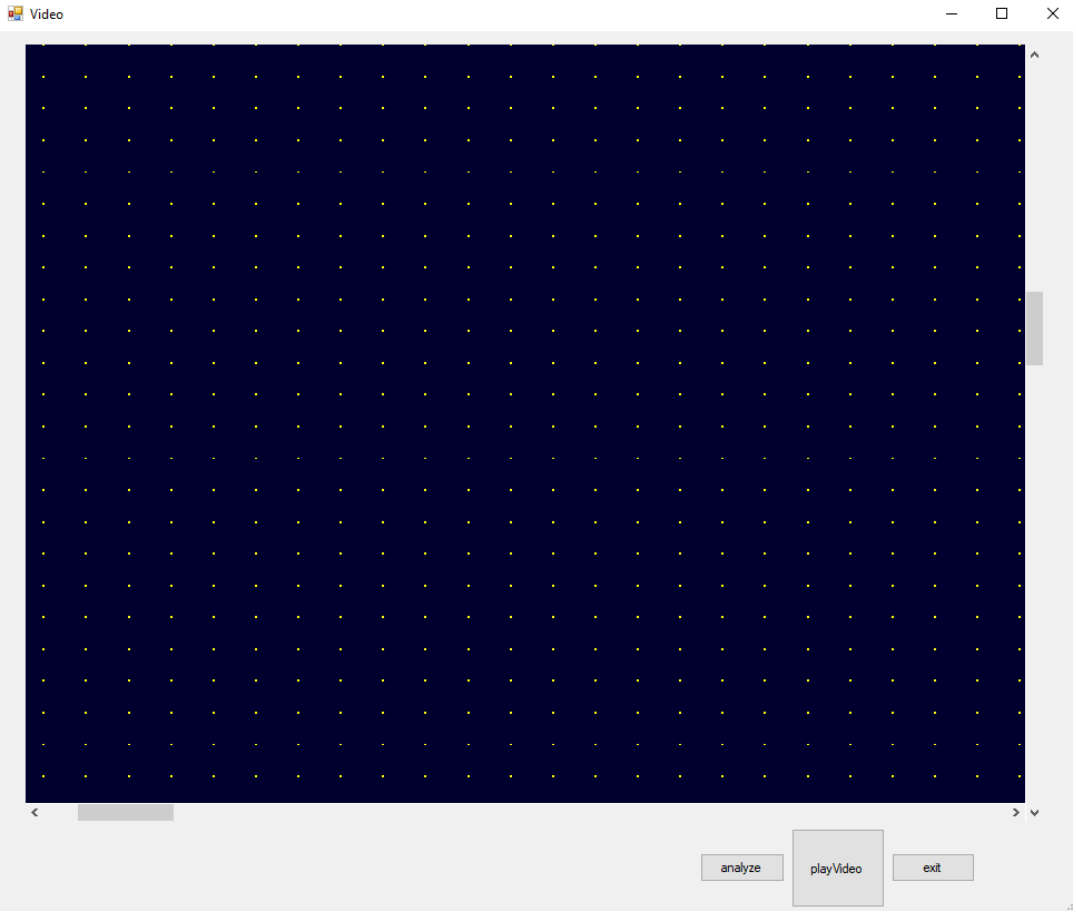


Рис. 2 – Класс Form2

Класс предназначен (рис. 2) для вычисления оптического потока и визуализации результата.

1. Метод button1_Click (analyze)

Метод анализирует видео, рисует векторное поле и складывает измененные кадры в список imagesList, рассмотрим его подробнее.

1) Чтение полей класса Form1: levels – количество используемых слоев, xywh содержит информацию о координатах левого верхнего угла блока, а также его ширине и высоте, drawVector представляет собой “шаг” векторного поля.

```
VideoCapture capt = Form1.capt;
int levels = Form1.levels;
var xywh = Form1.xywh;
int drawVector = Form1.drawVector;
```

2) Число $2^{levels-1}$ вынесено в отдельную переменную, так как будет часто использовано, переменная extra представляет количество дополнительных пустых пикселей, которые нужны для корректной аппроксимации производной интенсивности по x и y.

```
double delitel = Math.Pow(2, levels - 1);
int extra = 256;
```

3) Инициализация переменных: height, width – общей высоты и ширины изображения img3; totalFrames – количество кадров видео; переменная fps является глобальной, как и totalFrames получена с помощью метода GetCaptureProperty.

```
int height, width;
int totalFrames;
Mat frame = new Mat();
totalFrames =
      Convert.ToInt32(capt.GetCaptureProperty(Emgu.CV.CvEnum.CapProp.FrameCount));
fps = Convert.ToInt32(capt.GetCaptureProperty(Emgu.CV.CvEnum.CapProp.Fps));
```

4) Получение данных о первоначальной высоте и ширине кадра.

```
capt.SetCaptureProperty(Emgu.CV.CvEnum.CapProp.PosFrames, 0);
capt.Read(frame);
Image<Bgr, Byte> img3 = frame.ToImage<Bgr, Byte>();

height = img3.Height;
width = img3.Width;
```

5) Вычисление ширины и высоты изображения с учетом того, что при делении на $2^{levels-1}$ должно получиться целое число.

```
int widthInt = (int)Math.Ceiling(width / delitel) * (int)delitel;
int heightInt = (int)Math.Ceiling(height / delitel) * (int)delitel;
```

6) Инициализация изображений белого цвета в качестве пустых пикселей: imgAddW, imgAddH и imgAddS для случая если высота или ширина кадра не делятся нацело на $2^{levels-1}$, imgHor и imgVer присоединяются к изображению всегда в качестве дополнительных пикселей для последующего вычисления производной I_x, I_y.

```
Image<Bgr, Byte> imgAddW = new Image<Bgr, Byte>(widthInt - width, height,
                                                new Bgr(255, 255, 255));
Image<Bgr, Byte> imgAddH = new Image<Bgr, Byte>(width, heightInt - height,
                                                new Bgr(255, 255, 255));
Image<Bgr, Byte> imgAddS = new Image<Bgr, Byte>(widthInt - width, heightInt - height,
                                                new Bgr(255, 255, 255));

Image<Bgr, Byte> imgHor = new Image<Bgr, Byte>(extra, heightInt,
                                                new Bgr(255, 255, 255));
Image<Bgr, Byte> imgVer = new Image<Bgr, Byte>(widthInt + 2 * extra, extra,
                                                new Bgr(255, 255, 255));

Mat matAddSW = new Mat();
```

7) Начало цикла по k.

После установления указателя на кадр k происходит анализ, следует ли присоединять какие-либо из изображений imgAddW, imgAddH, imgAddS, затем присоединяются добавочные изображения imgHor и imgVer.

frame для удобства преобразуется в img1 типа Image<Gray, Byte>, производится медианная фильтрация img1 с апертурой в 13 пикселей; img3 необходимо для визуализации полученного векторного поля, в отличие от остальных изображений инициализируется в цветовой схеме Bgr.

```
for (int k = 0; k < totalFrames-1; k++)
{
     capt.SetCaptureProperty(Emgu.CV.CvEnum.CapProp.PosFrames, k);
     capt.Read(frame);
     if (widthInt - width != 0 && heightInt - height != 0)
     {
         CvInvoke.VConcat(frame, imgAddH, frame);
         CvInvoke.VConcat(imgAddW, imgAddS, matAddSW);
         CvInvoke.HConcat(frame, matAddSW, frame);
     }
     else if (widthInt - width != 0)
         CvInvoke.HConcat(frame, imgAddW, frame);
     else if (heightInt - height != 0)
         CvInvoke.VConcat(frame, imgAddH, frame);

     CvInvoke.HConcat(imgHor, frame, frame);
     CvInvoke.HConcat(frame, imgHor, frame);
     CvInvoke.VConcat(imgVer, frame, frame);
     CvInvoke.VConcat(frame, imgVer, frame);

     Image<Gray, Byte> img1 = frame.ToImage<Gray, Byte>();
     CvInvoke.MedianBlur(img1, img1, 13);
     img3 = frame.ToImage<Bgr, Byte>();
```

8) Аналогично шагу 7), но указатель на кадре k+1, и изображение типа Image<Gray, Byte>, соответствующее этому кадру это img2.

```
capt.SetCaptureProperty(Emgu.CV.CvEnum.CapProp.PosFrames, k+1);
capt.Read(frame);
if (widthInt - width != 0 && heightInt - height != 0)
{
     CvInvoke.VConcat(imgAddH, frame, frame);
     CvInvoke.VConcat(imgAddS, imgAddW, matAddSW);
     CvInvoke.HConcat(frame, matAddSW, frame);
}
else if (widthInt - width != 0)
     CvInvoke.HConcat(frame, imgAddW, frame);
else if (heightInt - height != 0)
CvInvoke.VConcat(imgAddH, frame, frame);
CvInvoke.HConcat(imgHor, frame, frame);
CvInvoke.HConcat(frame, imgHor, frame);
CvInvoke.VConcat(imgVer, frame, frame);
CvInvoke.VConcat(frame, imgVer, frame);

Image<Gray, Byte> img2 = frame.ToImage<Gray, Byte>();
CvInvoke.MedianBlur(img2, img2, 13);
```

9) Images – двумерный список изображений, где 0-й столбец отвечает k-му кадру, а 1-й k+1-му. Цикл по i создает пирамиду изображений с помощью линейной интерполяции

(коэффициент масштабирования = 2), причем на каждом из изображений используется медианная фильтрация.

```
Image<Gray, Byte>[,] images = new Image<Gray, Byte>[levels, 2];

int heightC, widthC;
heightC = img1.Height;
widthC = img1.Width;

images[0, 0] = img1;
images[0, 1] = img2;

for (int i = 1; i < levels; i++)
{
     widthC /= 2;
     heightC /= 2;
     images[i, 0] = img1.Resize(widthC, heightC, Inter.Linear);
     images[i, 1] = img2.Resize(widthC, heightC, Inter.Linear);
     CvInvoke.MedianBlur(images[i, 0], images[i, 0], 9);
     CvInvoke.MedianBlur(images[i, 1], images[i, 1], 9);
}
```

10) Позже найденное векторное поле будет преобразовано в двумерный вид, переменные totalBlocksRows и totalBlocksColumns содержат информацию о том, сколько в поле field будет векторов g по вертикали и горизонтали.

```
int totalBlocksRows, totalBlocksColumns;
widthC = widthInt / (int)delitel;  //ширина всех блоков на слое 2^(l-1)
heightC = heightInt / (int)delitel;
totalBlocksRows = (int)(heightC / (double)xywh[1, 1]) * (int)delitel;
totalBlocksColumns = (int)(widthC / (double)xywh[1, 0]) * (int)delitel;

var field = new int[totalBlocksRows, totalBlocksColumns][,];
```

11) Использование рекурсии может сильно замедлить работу программы, поэтому стоит применять распараллеливание. Одним из способов распараллеливания является цикл Parallel.ForEach.

Сначала формируется список входных данных inputList, элементы которого аналогичны переменной xywh, то есть содержат информацию о начале блоке, его длине и ширине.

Список составляется с учетом того, что каждое изображение имеет дополнительно присоединенные пиксели, вследствие чего самый первый блок имеет координаты (extra/delitel; extra/delitel).

Каждая итерация цикла Parallel.ForEach возвращает часть поля partField, начало которого задает делегат o. Найденные partField складываются в контейнер container. Для предотвращения “столкновения” данных и потери информации используется lock locker типа object.

```
List<int[,]> inputList = new List<int[,]>();

int extra2l = extra / (int)delitel;
int xIn = extra2l, yIn = extra2l;
while (yIn < heightC + extra2l)
```

```
    {
        xIn = extra2l;
        while (xIn < widthC + extra2l)
        {
            inputList.Add(new int[,] { { xIn, yIn }, { xywh[1, 0], xywh[1, 1] } });
            xIn += xywh[1, 0];
        }
        yIn += xywh[1, 1];
    }
    var container = new List<List<int[,]>>();
    Parallel.ForEach(inputList, (o) =>
    {
        var partField = new List<int[,]>();
        var g = new[] { 0.0, 0.0 };
        var g1 = new double[2];
        var xy = new[,] { { o[0, 0], o[0, 1] }, { o[1, 0], o[1, 1] } };

        g1 = CalcG(images, g, xy, levels - 1);
        Recurs(images, partField, Gx2(g1), XYx2(xy), levels - 2);

        lock (locker)
        {
           container.Add(partField);
        }
    });
```

12) Container содержит вектора g в неупорядоченном виде. Для восстановления порядка применяется метод FillField. Циклы по i, j пробегают все поле с шагом drawVector и рисуют вектора на img3, после чего на изображении выделяется область, которая не содержит дополнительных пикселей, то есть область оригинального размера кадра. Заметим, что если поле содержит бесконечный вектор, то он рисуется нулевым.

Завершающей операцией цикла по k (начало в шаге 7) является добавление полученного изображения img3 в список imageList.

```
    FillField(container, field, xywh, extra);

    for (int i = 0; i < field.GetLength(0); i += drawVector)
        for (int j = 0; j < field.GetLength(1); j += drawVector)
        {
            Point x = new Point(field[i, j][0, 0], field[i, j][0, 1]);
            Point y = new Point(field[i, j][1, 0], field[i, j][1, 1]);
            MCvScalar color = new MCvScalar(0, 255, 255);
            try
            {
                CvInvoke.ArrowedLine(img3, x, y, color, 1,
                                     LineType.EightConnected, 0, 0.3);
            }
            catch
            {
                CvInvoke.ArrowedLine(img3, x, x, color, 1,
                                     LineType.EightConnected, 0, 0.3);
            }

        }

   img3.ROI = new Rectangle(extra, extra, width, height);
   imagesList.Add(img3);
}
```

13) После того как все кадры в цикле по k (шаги 7-12) были обработаны кнопка playVideo становится видимой.

```
button3.Visible = true;
```

2. Метод Ix

```
private double Ix(Image<Gray, Byte>[,] images, int x, int y, int level)
{
    double I = (images[level, 0][y, x + 1].Intensity
              - images[level, 0][y, x - 1].Intensity) / 2.0;
    return I;
}
```

Метод возвращает значение производной интенсивности по x (формула (10)).

3. Метод Iy

```
private double Iy(Image<Gray, Byte>[,] images, int x, int y, int level)
{
    double I = (images[level, 0][y + 1, x].Intensity
              - images[level, 0][y - 1, x].Intensity) / 2.0;
    return I;
}
```

Метод возвращает значение производной интенсивности по y (формула (11)).

4. Метод dIk

```
private double dIk(Image<Gray, Byte>[,] images, double[] g,
                                double[] vk, int x, int y, int level)
{
    int degree = 1;
    int layer = level;
    double I, C;

    if (level == 0)
    {
        degree = 0; layer = 1;
    }
    C = Math.Pow(2, degree);

    int gxPLvx = (int)Math.Round(C * (g[0] + vk[0]));
    int gyPLvy = (int)Math.Round(C * (g[1] + vk[1]));

    x = (int)C * x; y = (int)C * y;

    int h = images[layer - 1, 0].Height, w = images[layer - 1, 0].Width;

    if(y + gyPLvy < 0 || y + gyPLvy >= h || x + gxPLvx < 0 || x + gxPLvx >= w)
        return 0;

    I = images[layer - 1, 0][y, x].Intensity
      - images[layer - 1, 1][y + gyPLvy, x + gxPLvx].Intensity;
    return I;
       }

  }
```

Метод рассчитывает производную интенсивности по времени (формула (15)). Если level – текущий слой, то производная считается на level-1 слое (кроме нулевого), то есть выполняется субпиксельное уточнение.

5. Метод CalcG

```
private double[] CalcG(Image<Gray, Byte>[,] images, double[] g, int[,] xywh, int level)
    {
        var vk = new double[2];
```

```
            var b = new double[2, 1];
            var W = w(images, xywh, level);
            int isInvertible = 1;
            var G = G_inv(images, xywh, level, W, ref isInvertible);

            if(isInvertible == 0)
            {
                vk[0] += g[0];
                vk[1] += g[1];
                return vk;
            }

            for (int k = 0; k < 5; k++)
            {
                b = bk(images, g, vk, xywh, level, W);
                vk[0] += G[0, 0] * b[0, 0] + G[0, 1] * b[1, 0];
                vk[1] += G[1, 0] * b[0, 0] + G[1, 1] * b[1, 0];
            }
            vk[0] += g[0];
            vk[1] += g[1];
            return vk;
        }
```

CalcG возвращает значение g для слоя level.

Сначала вычисляется матрица весов W и градиента G. Если матрица градиента вырожденная, то метод возвращает вектор g, который был на входе, если нет, то вычисляется вектор поправки vk, который затем складывается со входным g.

6. Метод G_inv

```
private double[,] G_inv(Image<Gray, Byte>[,] images, int[,] xywh, int level, double[,] W, ref
int isInvertible)
        {
            var G = new double[2, 2];
            double temp, det;
            double intx, inty;
            for (int j = xywh[0, 1]; j < xywh[1, 1] + xywh[0, 1]; j++)
                for (int i = xywh[0, 0]; i < xywh[1, 0] + xywh[0, 0]; i++)
                {
                    intx = Ix(images, i, j, level);
                    inty = Iy(images, i, j, level);
                    G[0, 0] += intx * intx * W[j - xywh[0, 1], i - xywh[0, 0]];
                    G[1, 1] += inty * inty * W[j - xywh[0, 1], i - xywh[0, 0]];
                    G[0, 1] += intx * inty * W[j - xywh[0, 1], i - xywh[0, 0]];
                    G[1, 0] = G[0, 1];
                }
            temp = G[0, 0];
            det = G[0, 0] * G[1, 1] - G[1, 0] * G[1, 0];
            if (det == 0)
            { det = 1;
                isInvertible = 0; }

            G[0, 0] = G[1, 1] / det;
            G[1, 1] = temp / det;
            G[0, 1] = G[0, 1] * (-1) / det;
            G[1, 0] = G[0, 1];
            return G;
        }
```

Метод производит вычисление обратной матрицы градиента (формула (14)). Где переменная isInvertible становится равной 0 в случае если определитель матрицы G равен нулю.

7. Метод w

```
private double[,] w(Image<Gray, Byte>[,] images, int[,] xywh, int level)
{
    var w = new double[xywh[1, 1], xywh[1, 0]];
    int x0 = xywh[0, 0] + xywh[1, 0] / 2;
    int y0 = xywh[0, 1] + xywh[1, 1] / 2;

    int sigD = 11, sigC = 15;
    int di, dj;
    double wd, wc;

    for (int j = xywh[0, 1]; j < xywh[1, 1] + xywh[0, 1]; j++)
        for (int i = xywh[0, 0]; i < xywh[1, 0] + xywh[0, 0]; i++)
        {
            di = i - x0;
            dj = j - y0;

            wd = Math.Exp(-(di * di + dj * dj) / (2.0 * sigD * sigD));
            wc = Math.Exp(-Math.Pow(images[level, 0][j, i].Intensity
             - images[level, 0][y0, x0].Intensity, 2) / (2.0 * sigC * sigC));

            w[j - xywh[0, 1], i - xywh[0, 0]] = wd*wc;
        }
    return w;
}
```

Метод реализует формулы (12)-(13) и вычисляет матрицу весов W.

8. Метод bk

```
private double[,] bk(Image<Gray, Byte>[,] images, double[] g, double[] vk,
                                          int[,] xywh, int level, double[,] W)
{
    var b = new double[2, 1];
    double Ik;
    for (int j = xywh[0, 1]; j < xywh[1, 1] + xywh[0, 1]; j++)
        for (int i = xywh[0, 0]; i < xywh[1, 0] + xywh[0, 0]; i++)
        {
            Ik = dIk(images, g, vk, i, j, level);
            b[0, 0] += Ik * Ix(images, i, j, level)
                          * W[j - xywh[0, 1], i - xywh[0, 0]];

            b[1, 0] += Ik * Iy(images, i, j, level)
                          * W[j - xywh[0, 1], i - xywh[0, 0]];
        }
    return b;
}
```

Вектор b вычисляется по формуле (16)

9. Метод Gx2

```
private double[] Gx2(double[] g)
{
    g[0] = 2 * g[0];
    g[1] = 2 * g[1];
    return g;
}
```

Возвращаемое методом значение g увеличивается в два раза.

10. Метод XYx2

```
private int[,] XYx2(int[,] xywh)
{
    xywh[0, 0] *= 2;
    xywh[0, 1] *= 2;
    return xywh;
}
```

XYx2 увеличивает в два раза координаты блока

11. Метод XYdivide2

```
private int[,] XYdivide2(int[,] xywh)
{
    xywh[0, 0] /= 2;
    xywh[0, 1] /= 2;
    return xywh;
 }
```

XYdivide2 уменьшает в два раза координаты блока

12. Метод Recurs

```
private void Recurs(Image<Gray, Byte>[,] images, List<int[,]> partFieldU, double[] g, int[,]
xywh, int level)
        {
            if (level == -1)
            {
                int x = (xywh[0, 0] + xywh[1, 0]) / 2;
                int y = (xywh[0, 1] + xywh[1, 1]) / 2;

                partFieldU.Add(new[,] { { x, y }, { x + (int)Math.Round(g[0]/2),
                                                    y + (int)Math.Round(g[1]/2) } });
                return;
            }
            var xywhRecurs = new int[,] { { xywh[0, 0], xywh[0, 1] },
                                          { xywh[1, 0], xywh[1, 1] } };
            var currentG = new double[2];

            currentG = CalcG(images, g, xywhRecurs, level);
            Recurs(images, partFieldU, Gx2(currentG), XYx2(xywhRecurs), level - 1);


            XYdivide2(xywhRecurs);
            xywhRecurs[0, 0] += xywhRecurs[1, 0];
            currentG = CalcG(images, g, xywhRecurs, level);
            Recurs(images, partFieldU, Gx2(currentG), XYx2(xywhRecurs), level - 1);


            XYdivide2(xywhRecurs);
            xywhRecurs[0, 1] += xywhRecurs[1, 1];
            currentG = CalcG(images, g, xywhRecurs, level);
            Recurs(images, partFieldU, Gx2(currentG), XYx2(xywhRecurs), level - 1);


            XYdivide2(xywhRecurs);
            xywhRecurs[0, 0] -= xywhRecurs[1, 0];
            currentG = CalcG(images, g, xywhRecurs, level);
            Recurs(images, partFieldU, Gx2(currentG), XYx2(xywhRecurs), level - 1);
         }
```

Метод, вызванный в основном коде, возвращает серию векторов для блока, координаты которого для самого глубокого слоя даны в переменной xywh. Recurs является рекурсивным методом и реализует идею деления блока на 4 части и последующего подсчета векторов g для каждого из новых блоков с учетом вектора для слоя выше.

Стоит отметить, что благодаря масштабированию, которое представлено методом XYx2, размеры нового блока совпадают с размерами предыдущего. Индексация уровней начинается с нуля, поэтому если level == -1, то это означает что g был посчитан для нулевого слоя, то есть, отрегулировав масштаб вектора и координаты его начала, можно добавить найденный вектор в список partFieldU.

13. Метод FillField

```
private void FillField(List<List<int[,]>> container, int[,][,] field, int[,] xywh, int extra)
        {
            int dw = xywh[1, 0];
            int dh = xywh[1, 1];
            int j, i;

            int[] g0 = { extra + dw/2, extra + dh/2 };

            foreach(var partField in container)
                foreach (var g in partField)
                {
                    j = (g[0, 0] - g0[0]) / dw;
                    i = (g[0, 1] - g0[1]) / dh;
                    field[i, j] = g;
                }
        }
```

Метод берет вектора из контейнера container и организует их в двумерное поле, где field имеет размерность количества векторов по вертикали на количества векторов по горизонтали и, таким образом, i, j – координаты самого вектора в поле field. Значения i, j вычисляются относительно вектора g0, dw и dh, так как начало любого вектора g можно разложить по базису dw, dh.

14. Метод button3_Click (PlayVideo)

```
private async void button3_Click(object sender, EventArgs e)
{
    for (int i = 0; i < imagesList.Count; i++)
    {
        imageBox1.Image = imagesList[i];
        await Task.Delay(1000 * 4 / fps);
    }
}
```

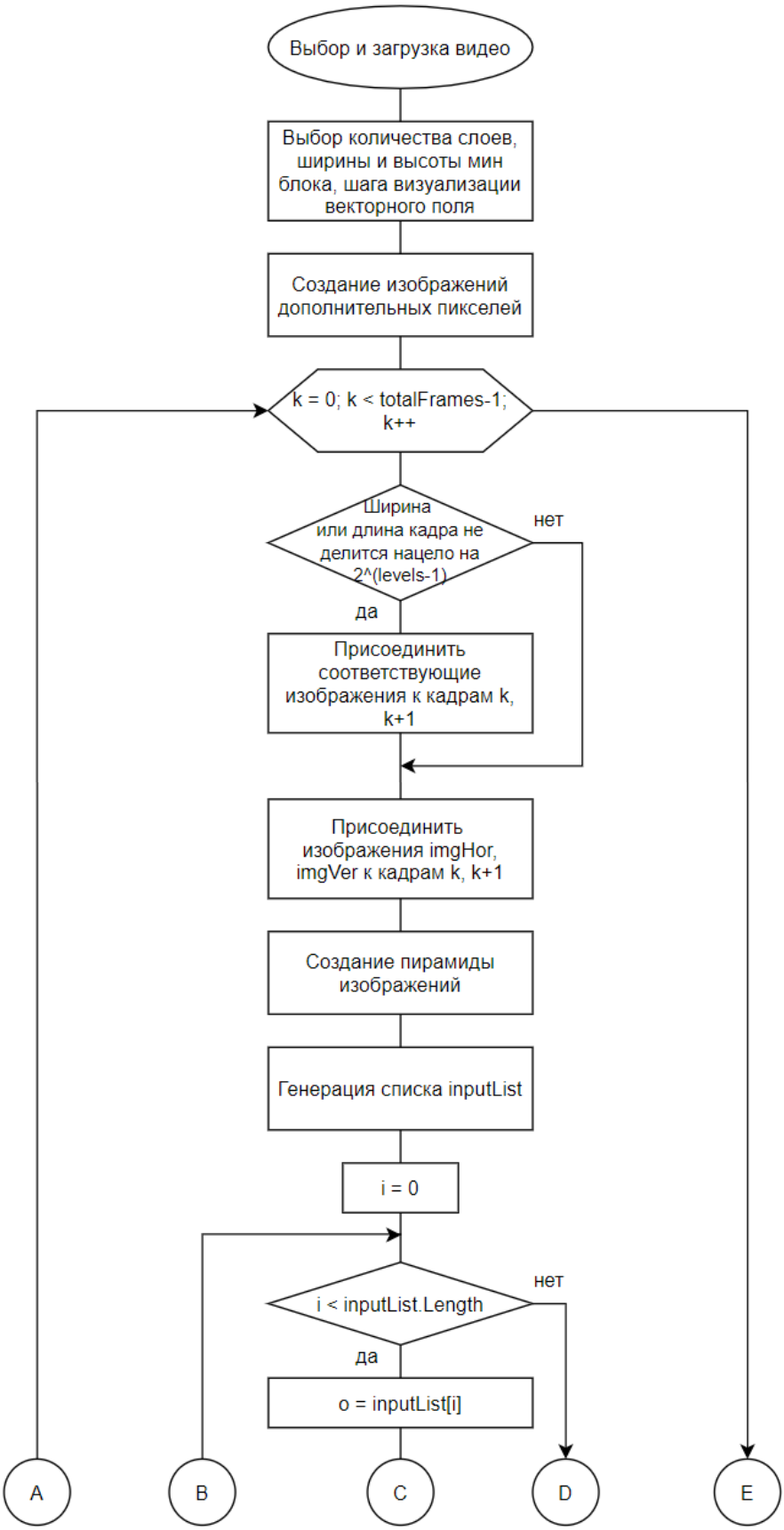
Метод проигрывает полученные кадры. Для создания пауз между кадрами используется Task.Delay, который задает скорость пролистывания изображений списка imagesList сравнимую со скоростью оригинального видео. Метод является асинхронным, так как используется класс Task и ключевое слово await.

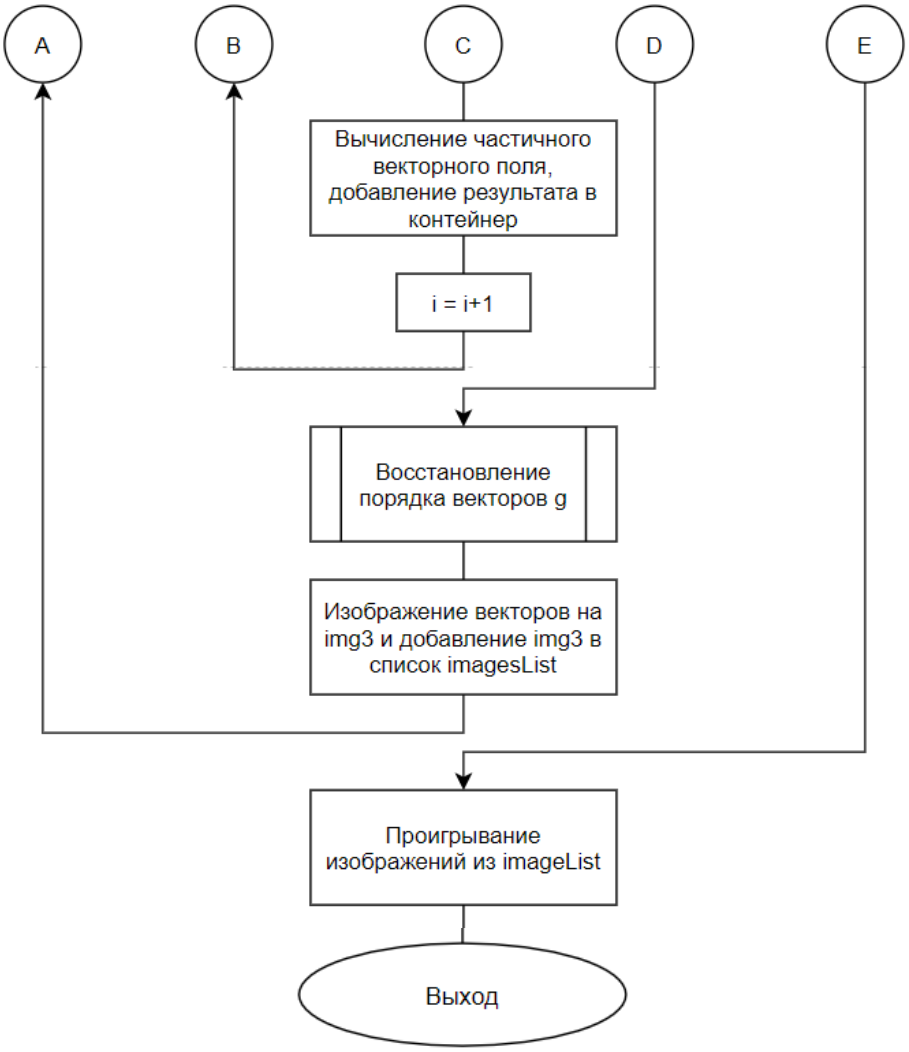
15. Метод button2_Click (exit)

```
private void button2_Click(object sender, EventArgs e)
{
    this.Close();
 }
```

Закрывает окно Video.

Приложение 1




A
B
C
D
E
Вычисление частичного векторного поля, добавление результата в контейнер
i = i+1
Восстановление порядка векторов g
Изображение векторов на img3 и добавление img3 в список imagesList
Проигрывание изображений из imageList
Выход