# 3. Растровые изображения и работа с файлами изображений

## 3.1. Пример слайд-шоу.

В прошлой лекции мы начали работу с графическими файлами и разработали пример слайд-шоу.

//test47\_slides

using MathPanel;

using System;

using System.Collections.Generic;

///сборки

///[DLL]System.dll,System.Xaml.dll,WindowsBase.dll,PresentationFramework.dll,PresentationCore.dll,System.Drawing.dll,System.Net.dll,System.Net.Http.dll,System.Core.dll[/DLL]

///

namespace DynamoCode

{

public class Script

{

public void Execute()

{

Dynamo.Console("test47\_slides");

//путь к папке

string sDir = @"C:\c\_devel\images\";

//массив файлов-изображений

string[] fnames = { "pat1\_rot.png", "pat2\_rot.png", "pat3\_rot.png", "pat4\_rot.png", "pat5\_rot.png",

"pat6\_rot.png", "pat7\_rot.png", "pat8\_rot.png", "pat9\_rot.png", "pat10\_rot.png",

"pat11\_rot.png", "pat12\_rot.png"

};

//по всем файлам

for (int i = 0; i < fnames.Length; i++)

{

var fn = fnames[i];

//загрузить файл в компонент Image

Dynamo.SetBitmapImage(sDir + fn);

//заснуть на 500 мсек

System.Threading.Thread.Sleep(500);

}

}

}

}

В этой главе мы продолжим развивать новую функциональность для обработки изображений.

## 3.2. Класс BitmapSimple – массив значений ARGB

В проекте добавим папку Geom. В ней создадим новый класс BitmapSimple. Уберем из namespace .Geom. И сделаем класс public. Сохранили, запустили, работает, пишем код дальше.

Это класс для 2-х мерного изображения, ARGB в карте слева направо и сверху вниз (рис.3.1).

Добавляем в код пространства имен

using System.Drawing;

using System.Drawing.Imaging;

using System.Runtime.InteropServices;

using System.IO;

и первый конструктор

namespace MathPanel

{

/// <summary>

/// 2-х мерное изображение, ARGB в карте слева направо и сверху вниз

/// </summary>

public class BitmapSimple

{

public int width, height;//ширина, высота изображения

public int[] map;//массив значений точек изображения

/// <summary>

/// 2-х мерное изображение

/// </summary>

/// <param name="w">ширина</param>

/// <param name="h">высота</param>

/// <param name="colors">цвета повторяющиеся</param>

public BitmapSimple(int w, int h, System.Drawing.Color[] colors)

{

width = w;

height = h;

map = new int[width \* height];

for (int i = 0; i < width \* height; i++)

{

map[i] = colors[i % colors.Length].ToArgb();

}

}

Добавляем также ссылку на сборку System.Drawing. Сохранили, запустили, работает.

# Color - структура представляет цвета в терминах каналов альфа, красного, зеленого и синего (ARGB). Цвет каждого пикселя представлен в виде 32-разрядного числа: по 8 бит для альфа-канала, красного, зеленого и синего (ARGB). Каждый из четырех компонентов — это число от 0 до 255, где 0 означает отсутствие интенсивности и 255, представляющие полную интенсивность. Альфа-компонент задает прозрачность цвета: 0 является полностью прозрачным, а 255 — полностью непрозрачным. Чтобы определить альфа-, красный, зеленый или синий компонент цвета, используйте A, R свойство, G или B соответственно. Пользовательский цвет можно создать с помощью одного из FromArgb методов.

# Метод ToArgb() возвращает 32-разрядное значение ARGB этой структуры Color.

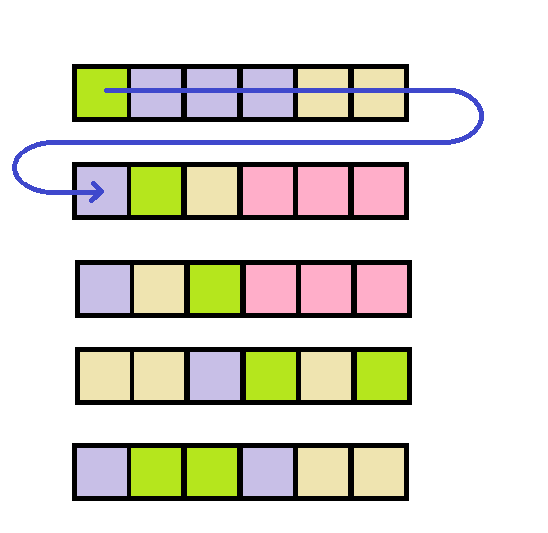


Рис.3.1. Представление данных в BitmapSimple

## 3.3. Сохранение растрового изображения в файл.

# Добавим метод для сохранения нашего изображения в файл.

/// <summary>

/// сохранить 2-х мерное изображение в файл

/// </summary>

/// <param name="fname">файл</param>

public void Save(string fname)

{

/\*

Создаем объект типа Bitmap, который используется для работы с изображениями, определяемыми данными пикселей.

Инкапсулирует точечный рисунок GDI+, состоящий из данных пикселей графического изображения и атрибутов рисунка.

Создаем объект Graphics , который инкапсулирует поверхность рисования GDI+.

Интерфейс GDI+ - это модель рисования общего назначения для приложений .NET.

В среде .NET интерфейс GDI+ используется в нескольких местах, в том числе при отправке документов на принтер,

отображения графики в Windows-приложениях и визуализации графических элементов на веб-странице.

\*/

Bitmap image = new Bitmap(width, height);

Graphics g = Graphics.FromImage(image);

//image.LockBits блокирует объект Bitmap в системной памяти, чтобы пиксели можно было изменять.

BitmapData bmData = image.LockBits(new Rectangle(0, 0, image.Width, image.Height), ImageLockMode.ReadWrite, PixelFormat.Format32bppArgb);

int offset = 0;

int stride = bmData.Stride;

//Получаем указатель на выделенный блок памяти

IntPtr Scan0 = bmData.Scan0;

int nOffset = stride - image.Width \* 4;

int k = 0;

//Далее в двойном цикле двигаемся по нашему массиву map, извлекаем из каждого элемента структуру Color

for (int y = 0; y < image.Height; y++)

{

for (int x = 0; x < image.Width; x++)

{

int argb = map[k];

var cc = System.Drawing.Color.FromArgb(argb);

//Пишем цвета в память, увеличиваем смещение

Marshal.WriteByte(Scan0, offset, (byte)cc.B);

Marshal.WriteByte(Scan0, offset + 1, (byte)cc.G);

Marshal.WriteByte(Scan0, offset + 2, (byte)cc.R);

Marshal.WriteByte(Scan0, offset + 3, (byte)cc.A);

k++;

offset += 4;

}

offset += nOffset;

}

//затем разблокируем память, сохраняем в файл в фрпмате PNG или JPEG, освобождаем ресурсы.

image.UnlockBits(bmData);

g.Flush();

if (File.Exists(fname)) File.Delete(fname);

//сохраняем как PNG или JPEG

image.Save(fname, fname.IndexOf(".png") > 0 ? System.Drawing.Imaging.ImageFormat.Png :

System.Drawing.Imaging.ImageFormat.Jpeg);

image.Dispose();

}

Читаем комментарии в коде. Теперь напишем простой скрипт.

//test37\_bitmap1

using MathPanel;

using System.Net.Sockets;

using System;

///сборки

///[DLL]System.dll,System.Xaml.dll,WindowsBase.dll,PresentationFramework.dll,PresentationCore.dll,System.Drawing.dll,System.Net.dll,System.Net.Http.dll,System.Core.dll[/DLL]

///

namespace DynamoCode

{

public class Script

{

public void Execute()

{

Dynamo.Console("test37\_bitmap1");

//путь к папке

string sDir = @"C:\c\_devel\images\";

//массив цветов

System.Drawing.Color[] colors = {

System.Drawing.Color.Red,

System.Drawing.Color.Orange,

System.Drawing.Color.Yellow,

System.Drawing.Color.Green,

System.Drawing.Color.Blue,

System.Drawing.Color.Magenta,

System.Drawing.Color.Cyan,

System.Drawing.Color.White,

System.Drawing.Color.Green,

};

//создать объект BitmapSimple

var bm = new BitmapSimple(40, 40, colors);

var fn = sDir +"test37\_bitmap1.png";

//сохранить в файл

bm.Save(fn);

//загрузить файл в компонент Image

Dynamo.SetBitmapImage(fn);

}

}

}

Запускаем программу, выбираем для выполнения наш скрипт, выполняем, переключаемся на «Image» и видим такое вот «чудо» (рис.3.2).



Рис.3.2. Генерация изображения из повторяющихся цветов.

## 3.4. Загрузка растрового изображения из файла.

Теперь добавим еще 2 метода. Конструктор с параметром в виде графического файла

/// <summary>

/// 2-х мерное изображение из файла

/// </summary>

/// <param name="fname">файл изображения</param>

public BitmapSimple(string fname)

{

FromFile(fname);

}

и помощник для него.

/// <summary>

/// 2-х мерное изображение из файла

/// </summary>

/// <param name="fname">файл изображения</param>

public void FromFile(string fname)

{

/\*

Создаем объект imageOrig типа Image (абстрактный класс, из которого выводятся Bitmap и Metafile) прямо

из имени файла (есть нужный метод в классе).

Из этого объекта создаем объект image типа Bitmap и освобождаем ресурс.

\*/

Image imageOrig = new Bitmap(fname);

Bitmap image = new Bitmap(imageOrig);

imageOrig.Dispose();

//Находим размеры изображения, выделяем требуемую память

width = image.Width;

height = image.Height;

map = new int[width \* height];

//GDI+ не достоверен - формат BGR, а НЕ RGB.

BitmapData bmData = image.LockBits(new Rectangle(0, 0, image.Width, image.Height), ImageLockMode.ReadWrite, PixelFormat.Format32bppArgb);

int offset = 0;

int stride = bmData.Stride;

//image.LockBits блокирует объект Bitmap в системной памяти, чтобы пиксели можно было читать/изменять.

//Получаем указатель на выделенный блок памяти

IntPtr Scan0 = bmData.Scan0;

int bBl, bGr, bRd, bAl;

int nOffset = stride - image.Width \* 4;

int k = 0;

//Затем в двойном цикле читаем данные пиксилей из изображения и копируем в наш массив map

for (int y = 0; y < image.Height; y++)

{

for (int x = 0; x < image.Width; x++)

{

bBl = Marshal.ReadByte(Scan0, offset);

bGr = Marshal.ReadByte(Scan0, offset + 1);

bRd = Marshal.ReadByte(Scan0, offset + 2);

bAl = Marshal.ReadByte(Scan0, offset + 3);

map[k++] = System.Drawing.Color.FromArgb(bAl, bRd, bGr, bBl).ToArgb();

offset += 4;

}

offset += nOffset;

}

//В финале разблокируем блок памяти для освобождения ресурсов

image.UnlockBits(bmData);

}

Наш объект получает информацию из графического файла. Читаем комментарии в коде.

Добавим еще один простой метод

/// <summary>

/// изменить пиксель

/// </summary>

public void Pixel(int x0, int y0, int alpha, int red, int green, int blue, int size\_x = 1, int size\_y = 1)

{

/\*

Изменяем пиксель в позиции x0, y0. Для этого из параметров alpha, red, green, blue формируем цвет cc.

Находим элемент массива map и присваиваем его. Подвергнуться изменению может не только один пиксель,

но и его соседи в прямоугольнике с размером size\_x, size\_y (по умолчанию они равны 1).

\*/

var cc = System.Drawing.Color.FromArgb(alpha, red, green, blue).ToArgb();

int k;

for (int y = y0; y < height && y < y0 + size\_y; y++)

{

for (int x = x0; x < width && x < x0 + size\_x; x++)

{

k = y \* width + x;

map[k] = cc;

}

}

}

Пишем скрипт с новыми методами

//test37\_bitmap2

using MathPanel;

using MathPanelExt;

using System.Net.Sockets;

using System;

///сборки

///[DLL]System.dll,System.Xaml.dll,WindowsBase.dll,PresentationFramework.dll,PresentationCore.dll,System.Drawing.dll,System.Net.dll,System.Net.Http.dll,System.Core.dll[/DLL]

///

namespace DynamoCode

{

public class Script

{

public void Execute()

{

Dynamo.Console("test37\_bitmap2");

//путь к папке

string sDir = @"C:\c\_devel\images\";

var fn = sDir + "test37\_bitmap1.png";

//создать объект BitmapSimple из файла

var bm = new BitmapSimple(fn);

//нанести черный прямоугольник на него

bm.Pixel(15, 15, 255, 0, 0, 0, 10, 10);

var fn\_2 = sDir + "test37\_bitmap2.png";

//сохранить

bm.Save(fn\_2);

Dynamo.SetBitmapImage(fn\_2);

}

}

}

Запускаем, выполняем, получаем небольшой черный квадрат в центре исходного изображения (рис.3.3).

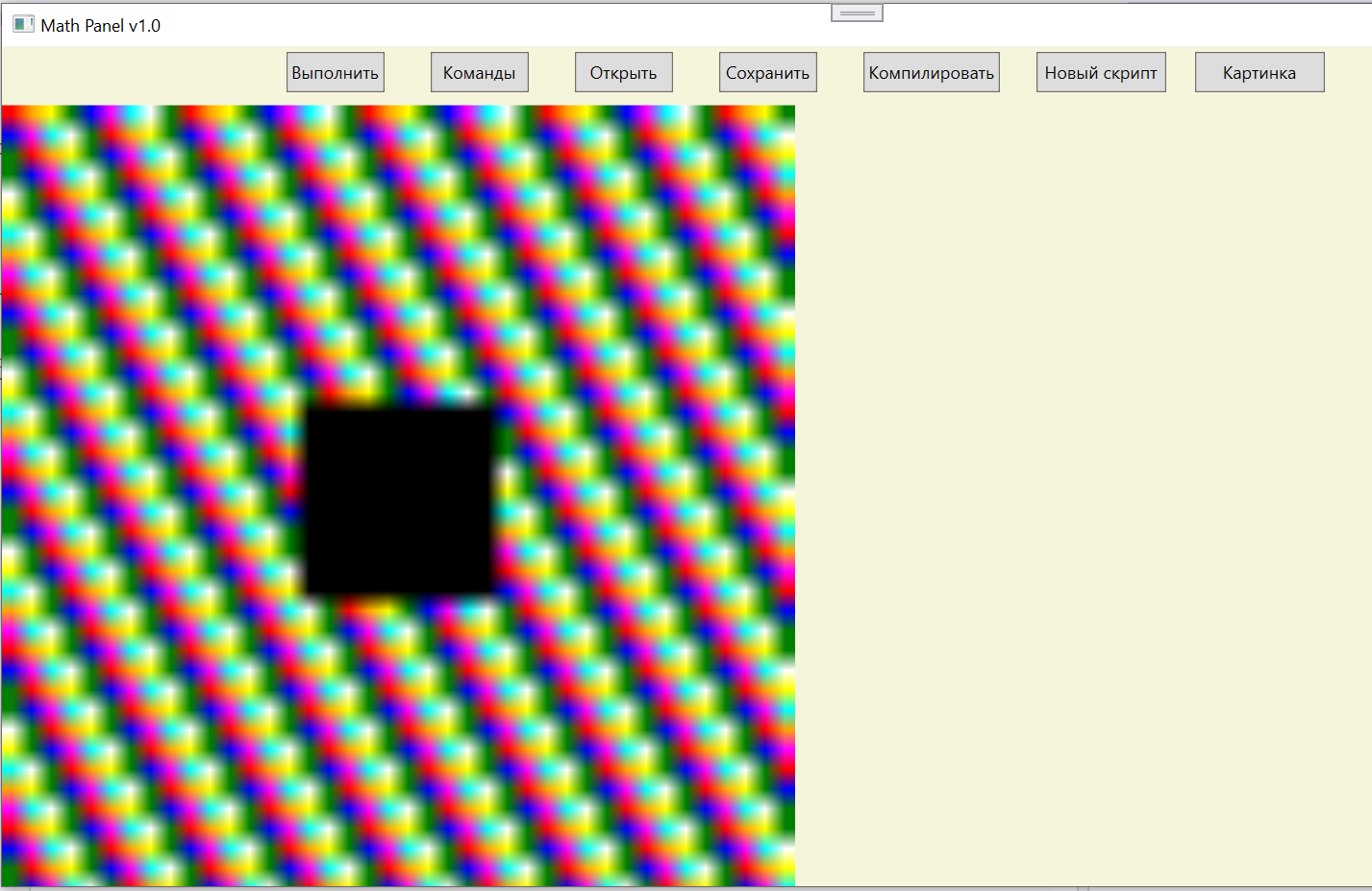


Рис.3.3. Черный квадрат в центре исходного изображения

## 3.5. Растровое изображение с градиентом.

Теперь создадим еще один конструктор для создания изображения с градиентом цвета.

/// <summary>

/// 2-х мерное изображение, с градиентом

/// </summary>

/// <param name="w">ширина</param>

/// <param name="h">высота</param>

/// <param name="col1">цвет сверху (слева)</param>

/// <param name="col2">цвет снизу (справа)</param>

/// <param name="bVert">true сверху (col1) вниз (col2), else слева направо</param>

public BitmapSimple(int w, int h, System.Drawing.Color col1, System.Drawing.Color col2, bool bVert = true)

{

width = w;

height = h;

map = new int[width \* height];

int k = 0, alpha, red, green, blue;

byte a1 = col1.A;

byte r1 = col1.R;

byte g1 = col1.G;

byte b1 = col1.B;

byte a2 = col2.A;

byte r2 = col2.R;

byte g2 = col2.G;

byte b2 = col2.B;

if (bVert)

{

//Если градиент вертикальный, то все просто.

//Для каждой линии изображения вычисляем значения ARGB пикселя как линейную зависимость

for (int j = 0; j < height; j++)

{

alpha = a1 + ((a2 - a1) \* j) / height;

red = r1 + ((r2 - r1) \* j) / height;

green = g1 + ((g2 - g1) \* j) / height;

blue = b1 + ((b2 - b1) \* j) / height;

int argb = System.Drawing.Color.FromArgb(alpha, red, green, blue).ToArgb();

for (int i = 0; i < width; i++)

{

//записать цвет в массив

map[k++] = argb;

}

}

}

else

{

//Если градиент горизонтальный, то чуть сложнее.

//Для каждого столбца изображения вычисляем значения ARGB пикселя.

//Образуем цикл по элементам столбца, вычисляем индекс элемента i \* width + j

//и присваиваем ему значения ARGB

for (int j = 0; j < width; j++)

{

alpha = a1 + ((a2 - a1) \* j) / width;

red = r1 + ((r2 - r1) \* j) / width;

green = g1 + ((g2 - g1) \* j) / width;

blue = b1 + ((b2 - b1) \* j) / width;

int argb = System.Drawing.Color.FromArgb(alpha, red, green, blue).ToArgb();

//Forming a loop over the column elements, calculating the index of the element

for (int i = 0; i < height; i++)

{

map[i \* width + j] = argb;

}

}

}

}

Если градиент вертикальный, то все просто. Для каждой линии изображения вычисляем значения ARGB пикселя как линейную зависимость.

Если градиент горизонтальный, то чуть сложнее. Для каждого столбца изображения вычисляем значения ARGB пикселя. Образуем цикл по элементам столбца, вычисляем индекс элемента i \* width + j и присваиваем ему значения ARGB.

Пишем скрипт

//test37\_bitmap3

using MathPanel;

using System.Net.Sockets;

using System;

///сборки

///[DLL]System.dll,System.Xaml.dll,WindowsBase.dll,PresentationFramework.dll,PresentationCore.dll,System.Drawing.dll,System.Net.dll,System.Net.Http.dll,System.Core.dll[/DLL]

///

namespace DynamoCode

{

public class Script

{

public void Execute()

{

Dynamo.Console("test37\_bitmap3");

//путь к папке

string sDir = @"C:\c\_devel\images\";

//Создаем 4 битмапа с градиентом от белого к голубому и по очереди загружаем в компонент Image

string[] fns = { "test37\_bitmap3\_a.png", "test37\_bitmap3\_b.png", "test37\_bitmap3\_c.png", "test37\_bitmap3\_d.png" };

var bm = new BitmapSimple(200, 200, System.Drawing.Color.White, System.Drawing.Color.Blue, false);

bm.Save(sDir + fns[0]);

var bm2 = new BitmapSimple(200, 200, System.Drawing.Color.White, System.Drawing.Color.Blue, true);

bm2.Save(sDir + fns[1]);

var bm3 = new BitmapSimple(200, 200, System.Drawing.Color.Blue, System.Drawing.Color.White, false);

bm3.Save(sDir + fns[2]);

var bm4 = new BitmapSimple(200, 200, System.Drawing.Color.Blue, System.Drawing.Color.White, true);

bm4.Save(sDir + fns[3]);

for ( int i = 0; i < 100; i++ )

{

Dynamo.SetBitmapImage(sDir + fns[i % 4]);

System.Threading.Thread.Sleep(1000);

}

}

}

}

Создаем 4 битмапа с градиентом от белого к голубому и по очереди загружаем в компонент Image.

Запускаем, выполняем, переключаемся на «Картинка» и смотрим анимацию (рис.3.4).

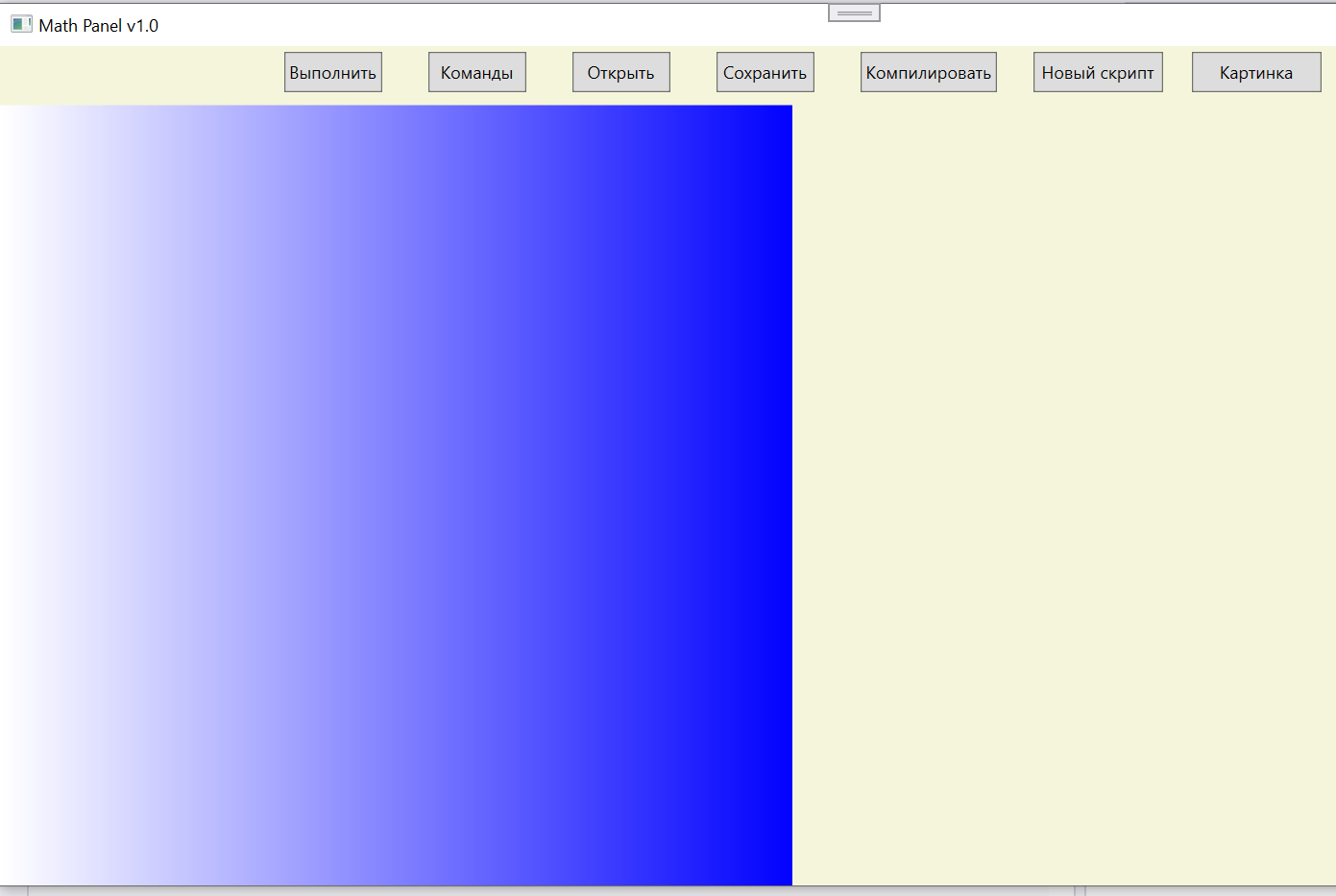


Рис.3.4. Элемент анимации из градиентов

## 3.6. Добавление шума

Создадим новый метод, который имитирует каплю краски.

/// <summary>

/// капля

/// </summary>

/// <param name="clr">цвет капли</param>

/// <param name="x\_0">позиция по горизонтали</param>

/// <param name="y\_0">позиция по вертикали</param>

/// <param name="wi">ширина капли</param>

/// <param name="he">высота капли</param>

/// <param name="dInitForce">начальная сила капли</param>

public void Drop(System.Drawing.Color clr, int x\_0, int y\_0, int wi, int he, double dInitForce, bool bLinear = true)

{

//имитирует каплю краски

if (wi <= 0 || he <= 0 || x\_0 < 0 || x\_0 >= width || y\_0 < 0 || y\_0 >= height) return;

if (dInitForce <= 0 || dInitForce > 1) dInitForce = 1;

int k, i, j, x, y;

//вытянутость по ширине

double dEllipse = (wi \* 1.0) / he;

//максимальный радиус капли

double dRadiusMax = Math.Sqrt(wi \* wi + he \* he \* dEllipse \* dEllipse) / 2;

for (i = 0; i < wi; i++)

{ //слева направо

for (j = 0; j < he; j++)

{ //сверху вниз

x = x\_0 - wi / 2 + i;

y = y\_0 - he / 2 + j;

if (x < 0 || x >= width || y < 0 || y >= height) { }

else

{

double dRadius = Math.Sqrt((x - x\_0) \* (x - x\_0) + (y - y\_0) \* (y - y\_0) \* dEllipse \* dEllipse);

//не превышать максимальный радиус

if (dRadiusMax <= dRadius) continue;

k = y \* width + x;

int argb = map[k];

var clr\_orig = System.Drawing.Color.FromArgb(argb);

//сила капли постепенно убывает

int argb\_new = Merge(clr\_orig, clr, dInitForce \* Math.Pow((dRadiusMax - dRadius) / dRadiusMax, bLinear ? 1 : 2));

map[k] = argb\_new;

//string s = string.Format("x={0}, y={1}, k={2}, i={3}, j={4}, argb={5}, new={6}", x, y, k, i, j, argb, argb\_new);

//Dynamo.Log(s);

}

}

}

}

И 2 вспомогательных метода

/// <summary>

/// устанавливает значение в диапазоне 0-255

/// </summary>

/// <param name="cc">цвет</param>

public static int SafeColor(int cc)

{

//не давать выходить за 0-255

if (cc < 0) return 0;

if (cc > 255) return 255;

return cc;

}

/// <summary>

/// слить 2 цвета

/// </summary>

/// <param name="clr\_orig">цвет основной</param>

/// <param name="clr">цвет капли</param>

/// <param name="dInitForce">начальная сила капли</param>

public static int Merge(System.Drawing.Color clr\_orig, System.Drawing.Color clr, double dInitForce)

{

//Суммируем 2 значения – новый цвет с весом dInitForce и исходный цвет

//с весом (1 - dInitForce). И так по всем каналам

int alpha, red, green, blue;

alpha = clr\_orig.A;

red = SafeColor((int)(dInitForce \* clr.R + clr\_orig.R \* (1 - dInitForce)));

green = SafeColor((int)(dInitForce \* clr.G + clr\_orig.G \* (1 - dInitForce)));

blue = SafeColor((int)(dInitForce \* clr.B + clr\_orig.B \* (1 - dInitForce)));

return System.Drawing.Color.FromArgb(alpha, red, green, blue).ToArgb();

}

Создадим скрипт, рисующий 2 варианта неба с 3-мя облаками.

//test37\_bitmap4

using MathPanel;

//using MathPanelExt;

using System.Net.Sockets;

using System;

///сборки

///[DLL]System.dll,System.Xaml.dll,WindowsBase.dll,PresentationFramework.dll,PresentationCore.dll,System.Drawing.dll,System.Net.dll,System.Net.Http.dll,System.Core.dll[/DLL]

///

namespace DynamoCode

{

public class Script

{

public void Execute()

{

Dynamo.Console("test37\_bitmap4");

//путь к папке

string sDir = @"C:\c\_devel\images\";

string [] fns = { "test37\_bitmap4\_a.png", "test37\_bitmap4\_b.jpg" };

//сине-красное небо с 3-мя светло-серыми облаками

var bm = new BitmapSimple(800, 600, System.Drawing.Color.Blue, System.Drawing.Color.Red, true);

var cc = System.Drawing.Color.LightGray;

bm.Drop(cc, 100, 100, 80, 80, 0.9, false);//резкое затухание капли

bm.Drop(cc, 300, 100, 100, 50, 0.9, true);//линейное затухание капли

bm.Drop(cc, 500, 150, 50, 100, 0.9, false);

bm.Save(sDir + fns[0]);

//темносине-красное небо с 3-мя серыми облаками

var bm2 = new BitmapSimple(800, 600, System.Drawing.Color.DarkBlue, System.Drawing.Color.Red, true);

cc = System.Drawing.Color.Gray;

bm2.Drop(cc, 100, 100, 80, 80, 0.9, false);//резкое затухание капли

bm2.Drop(cc, 300, 100, 100, 50, 0.9, true);//линейное затухание капли

bm2.Drop(cc, 500, 150, 50, 100, 0.9, false);

bm2.Save(sDir + fns[1]);

for ( int i = 0; i < 100; i++ )

{

Dynamo.SetBitmapImage(sDir + fns[i % 2]);

System.Threading.Thread.Sleep(1000);

}

}

}

}

Первый вариант неба сохраняем в формате png, второй в формате jpeg. Видим, что 2-й файл получился в 3 раза меньше (jpeg использует сжатие с небольшой потерей качества, на глаз незаметно). Результат выполнения на рис.3.5.

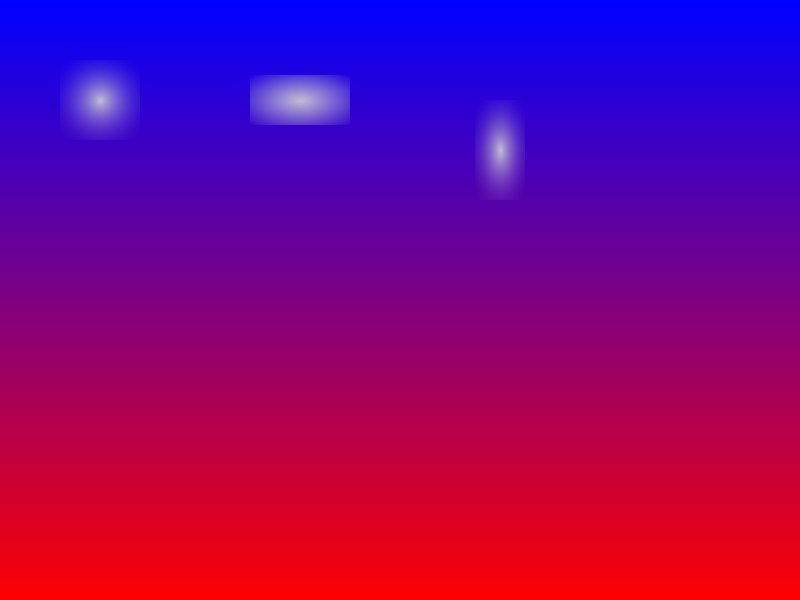


Рис. 3.5. Сине-красное небо с 3-мя светло-серыми облаками

Сделаем еще метод, который добавляет немного шума в изображение.

/// <summary>

/// добавить случайности

/// </summary>

/// <param name="nNoise">число отклонений</param>

/// <param name="iNoiceStrenth">сила отклонения</param>

public void Randomize(int nNoise, int iNoiceStrenth = 2)

{

if (nNoise <= 0 || iNoiceStrenth < 1) return;

int k, alpha, red, green, blue, i;

//создать объект типа "генератор случайных чисел"

Random rnd = new Random();

for (i = 0; i < nNoise; i++)

{ //в цикле по числу отклонений

//выбрать случайный элемент в массиве пикселей

k = rnd.Next(0, width \* height);

int argb = map[k];

var clr = System.Drawing.Color.FromArgb(argb);

//слегка изменить значения в каналах

alpha = clr.A;

red = SafeColor(clr.R + rnd.Next(0, iNoiceStrenth) - iNoiceStrenth / 2);

green = SafeColor(clr.G + rnd.Next(0, iNoiceStrenth) - iNoiceStrenth / 2);

blue = SafeColor(clr.B + rnd.Next(0, iNoiceStrenth) - iNoiceStrenth / 2);

//сохранить пиксель

map[k] = System.Drawing.Color.FromArgb(alpha, red, green, blue).ToArgb();

}

}

В цикле по числу вариаций находим случайный элемент в массиве, затем немного меняем значения в каналах и сохраняем пиксель. Напишем небольшой скрипт. Результат его выполнения на рис.3.6.

//test37\_bitmap5

using MathPanel;

using System.Net.Sockets;

using System;

///сборки

///[DLL]System.dll,System.Xaml.dll,WindowsBase.dll,PresentationFramework.dll,PresentationCore.dll,System.Drawing.dll,System.Net.dll,System.Net.Http.dll,System.Core.dll[/DLL]

///

namespace DynamoCode

{

public class Script

{

public void Execute()

{

Dynamo.Console("test37\_bitmap5");

//путь к папке

string sDir = @"C:\c\_devel\images\";

string [] fns = { "test37\_bitmap5\_a.png", "test37\_bitmap5\_b.png", "test37\_bitmap5\_с.png", "test37\_bitmap5\_d.png", "test37\_bitmap5\_e.png" };

int [] nNoise = { 100000, 100000, 100000, 100000, 100000 };

int [] iNoiceStrenth = { 20, 40, 60, 80, 100 };

//сгенерировать 5 иображений с увеличивающимся уровнем шума

for (int i = 0; i < fns.Length; i++)

{

var bm = new BitmapSimple(800, 600, System.Drawing.Color.Blue, System.Drawing.Color.Red, true);

bm.Randomize(nNoise[i], iNoiceStrenth[i]);

bm.Save(sDir + fns[i]);

}

//слайд-шоу

for ( int i = 0; i < 100; i++ )

{

Dynamo.SetBitmapImage(sDir + fns[i % fns.Length]);

System.Threading.Thread.Sleep(1000);

}

}

}

}

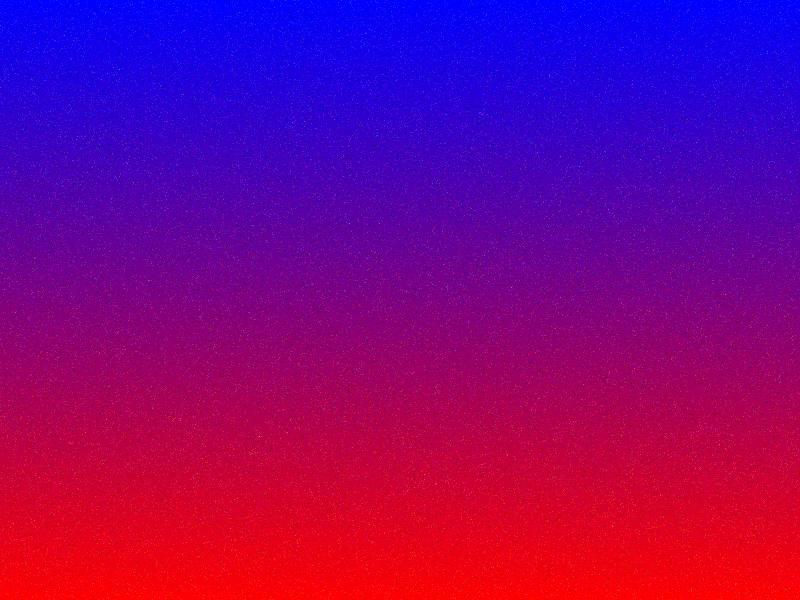


Рис. 3.6. Зашумленный градиент

## 3.7. Оттенки серого и сглаживание

Переходим к обработке изображений. Сделаем 3 метода

//трансформации

/// <summary>

/// из цветного в серый

/// </summary>

/// <param name="bAver">true - брать среднее, иначе максимальное</param>

public void Gray(bool bAver = true)

{ //2021-07-13

int alpha, red, green, blue;

int k = 0;

for (int y = 0; y < height; y++)

{

for (int x = 0; x < width; x++)

{

int argb = map[k];

var cc = System.Drawing.Color.FromArgb(argb);

alpha = cc.A;

//Преобразуем изображение в оттенки серого. Находим среднее значение по каналам

if ( bAver )red = (cc.R + cc.G + cc.B) / 3;

else

{ //находим максимальный уровень

red = cc.R;

if (red < cc.G) red = cc.G;

if (red < cc.B) red = cc.B;

}

green = red;

blue = red;

map[k++] = System.Drawing.Color.FromArgb(alpha, red, green, blue).ToArgb();

}

}

}

Преобразуем изображение в оттенки серого. Находим среднее или максимальное значение по каналам.

/// <summary>

/// в черно-белый

/// </summary>

/// <param name="threshold">порог</param>

public void BlackWhite(int threshold = 127)

{

int alpha, red, green, blue;

int k = 0;

for (int y = 0; y < height; y++)

{

for (int x = 0; x < width; x++)

{

int argb = map[k];

var cc = System.Drawing.Color.FromArgb(argb);

alpha = cc.A;

//Преобразуем изображение в черно-белое. Находим среднее значение по каналам и сравниваем с порогом.

red = (cc.R + cc.G + cc.B) / 3;

red = red > threshold ? 255 : 0;

green = red;

blue = red;

map[k++] = System.Drawing.Color.FromArgb(alpha, red, green, blue).ToArgb();

}

}

}

Преобразуем изображение в черно-белое. Находим среднее значение по каналам и сравниваем с порогом.

/// <summary>

/// сделать расплавчатым

/// </summary>

/// <param name="num">число итераций</param>

/// <param name="step">область соседей +-</param>

public void Smooth(int num = 1, int step = 1)

{

//Уменьшаем контрастность изображения. Выделяем память под временный массив

int[] map\_2 = new int[width \* height];

int alpha, red, green, blue;

int k, m, n, x\_2, y\_2;

for (int l = 0; l < num; l++)

{

for (int y = 0; y < height; y++)

{

for (int x = 0; x < width; x++)

{

k = y \* width + x;

int argb = map[k];

map\_2[k] = argb;

alpha = 0;

red = 0;

green = 0;

blue = 0;

n = 0;

//Находим соседей текущей точки , накапливаем значения по каналам

for (int i = -step; i <= step; i++)

{

y\_2 = y + i;

if (y\_2 < 0 || y\_2 >= height) continue;

for (int j = -step; j <= step; j++)

{

x\_2 = x + j;

if (x\_2 < 0 || x\_2 >= width) continue;

n++;

m = y\_2 \* width + x\_2;

argb = map[m];

var cc = System.Drawing.Color.FromArgb(argb);

alpha += cc.A;

red += cc.R;

green += cc.G;

blue += cc.B;

}

}

//n – число найденных соседей

if (n > 0)

{

map\_2[k] = System.Drawing.Color.FromArgb(alpha / n, red / n, green / n, blue / n).ToArgb();

}

}

}

//В заключении копируем из временного массива в основной

for (int y = 0; y < height \* width; y++) map[y] = map\_2[y];

}

}

Уменьшаем контрастность изображения. Выделяем память под временный массив.

Находим соседей текущей точки, накапливаем значения по каналам. В заключении копируем из временного массива в основной.

Напишем скрипт.

//test37\_bitmap6

using MathPanel;

using System.Net.Sockets;

using System;

///сборки

///[DLL]System.dll,System.Xaml.dll,WindowsBase.dll,PresentationFramework.dll,PresentationCore.dll,System.Drawing.dll,System.Net.dll,System.Net.Http.dll,System.Core.dll[/DLL]

///

namespace DynamoCode

{

public class Script

{

public void Execute()

{

Dynamo.Console("test37\_bitmap6");

//путь к папке

string sDir = @"C:\c\_devel\images\";

string [] fns = {"world1960.jpg", "test37\_bitmap6\_a.png", "test37\_bitmap6\_b.png", "test37\_bitmap6\_с.png" };

//Здесь берем исходное изображение и с помощью новых методов создаем 3 модификации.

for (int i = 1; i < fns.Length; i++)

{

var bm = new BitmapSimple(sDir + fns[0]);

if (i == 1)

bm.Gray();

else if (i == 2)

bm.BlackWhite();

else if (i == 3)

bm.Smooth(1, 10);

bm.Save(sDir + fns[i]);

}

//слайд-шоу

for ( int i = 0; i < 100; i++ )

{

Dynamo.SetBitmapImage(sDir + fns[i % fns.Length]);

System.Threading.Thread.Sleep(1000);

}

}

}

}

Здесь берем исходное изображение и с помощью новых методов создаем 3 модификации. Результат на рис.3.7.

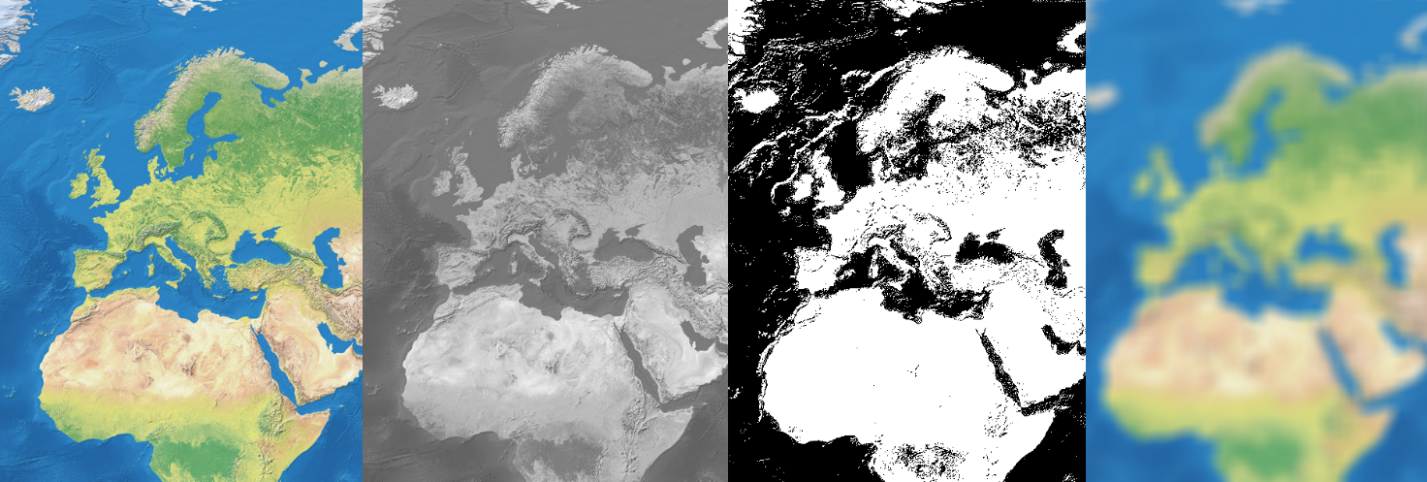


Рис.3.7. Результат наложения исходного и трех полученных изображений.

## 3.8. Фильтрация и Sobel-фильтр

Реализуем метод фильтрации с помощью правил. Правило определяет, как найти соседа текущей точки (смещения x, y) и вес (value) этой точки в вычисления.

/// <summary>

/// фильтровать с помощью правил

/// </summary>

/// <param name="fil">список правил вида (x, y, value)</param>

/// <param name="num">число итераций</param>

public void Filter(List<Tuple<int, int, double>> fil, int num = 1)

{

//Выделяем память под временный массив

int[] map\_2 = new int[width \* height];

int alpha, red, green, blue;

int k, m, n, x\_2, y\_2;

for (int l = 0; l < num; l++)

{

for (int y = 0; y < height; y++)

{

for (int x = 0; x < width; x++)

{

k = y \* width + x;

int argb = map[k];

map\_2[k] = argb;

alpha = 255;

red = 0;

green = 0;

blue = 0;

n = 0;

foreach (var tup in fil)

{ //для каждого правила находим соседей текущей точки

y\_2 = y + tup.Item2;

if (y\_2 < 0 || y\_2 >= height) continue;

x\_2 = x + tup.Item1;

if (x\_2 < 0 || x\_2 >= width) continue;

n++;

m = y\_2 \* width + x\_2;

argb = map[m];

var cc = System.Drawing.Color.FromArgb(argb);

//alpha += (int)(cc.A \* tup.Item3);

red += (int)(cc.R \* tup.Item3);

green += (int)(cc.G \* tup.Item3);

blue += (int)(cc.B \* tup.Item3);

}

//n – число найденных соседей

if (n > 0)

{

map\_2[k] = System.Drawing.Color.FromArgb(SafeColor(alpha),

SafeColor(red), SafeColor(green), SafeColor(blue)).ToArgb();

}

}

}

//В заключении копируем из временного массива в основной

for (int y = 0; y < height \* width; y++) map[y] = map\_2[y];

}

}

Реализуем новый фильтр Собеля**,** который используется в области обработки изображений. Часто его применяют в алгоритмах выделения границ. Ниже даны маски оператора Собеля для вертикального и горизонтального направления.

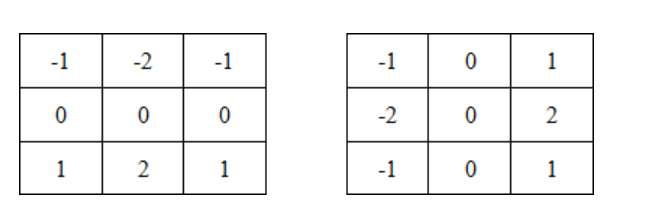


Рис.3.8. Маски оператора Собеля

В соответствии с этими масками получаем код

/// <summary>

/// Sobel фильтр

/// если фон белый, то раннее определение границ, если фон черный, то позднее.

/// </summary>

public void Sobel(bool bHoriz = true, int num = 1)

{

List<Tuple<int, int, double>> fil = new List<Tuple<int, int, double>>();

Tuple<int, int, double> m00, m01, m02, m10, m11, m12, m20, m21, m22;

if (bHoriz)

{

m00 = new Tuple<int, int, double>(-1, -1, 1);

m01 = new Tuple<int, int, double>(-1, 0, 2);

m02 = new Tuple<int, int, double>(-1, 1, 1);

m10 = new Tuple<int, int, double>(0, -1, 0);

m11 = new Tuple<int, int, double>(0, 0, 0);

m12 = new Tuple<int, int, double>(0, 1, 0);

m20 = new Tuple<int, int, double>(1, -1, -1);

m21 = new Tuple<int, int, double>(1, 0, -2);

m22 = new Tuple<int, int, double>(1, 1, -1);

}

else

{

m00 = new Tuple<int, int, double>(-1, -1, 1);

m01 = new Tuple<int, int, double>(-1, 0, 0);

m02 = new Tuple<int, int, double>(-1, 1, -1);

m10 = new Tuple<int, int, double>(0, -1, 2);

m11 = new Tuple<int, int, double>(0, 0, 0);

m12 = new Tuple<int, int, double>(0, 1, -2);

m20 = new Tuple<int, int, double>(1, -1, 1);

m21 = new Tuple<int, int, double>(1, 0, 0);

m22 = new Tuple<int, int, double>(1, 1, -1);

}

fil.Add(m00);

fil.Add(m01);

fil.Add(m02);

fil.Add(m10);

fil.Add(m11);

fil.Add(m12);

fil.Add(m20);

fil.Add(m21);

fil.Add(m22);

Filter(fil, num);

}

Пишем скрипт, который применяет фильтр к 4-м картинкам.

//test42\_filter\_sobel

using MathPanel;

using System.Net.Sockets;

using System;

using System.Collections.Generic;

///сборки

///[DLL]System.dll,System.Xaml.dll,WindowsBase.dll,PresentationFramework.dll,PresentationCore.dll,System.Drawing.dll,System.Net.dll,System.Net.Http.dll,System.Core.dll[/DLL]

///

namespace DynamoCode

{

public class Script

{

public void Execute()

{

Dynamo.Console("test42\_filter\_sobel");

//путь к папке

string sDir = @"C:\c\_devel\images\";

//имена файлов

string[] fnames = { "black\_rects", "white\_rects", "cubes", "white\_lines" };

//применяет фильтр к 4-м картинкам

for (int i = 0; i < fnames.Length; i++)

{

var fn = fnames[i];

var bm = new BitmapSimple(sDir + fn + ".png");

DateTime dt1 = DateTime.Now;

bm.Sobel(true);// false, 1);

var fn\_2 = fn + "\_horz.png";// +"\_vert.png";

bm.Save(sDir + fn\_2);

}

//слайд-шоу

for (int i = 0; i < 100; i++)

{

Dynamo.SetBitmapImage(sDir + fnames[i % fnames.Length] + "\_horz.png");

System.Threading.Thread.Sleep(1000);

}

}

}

}

Видно, что фильтр хорошо выделяет границы объектов в изображениях.

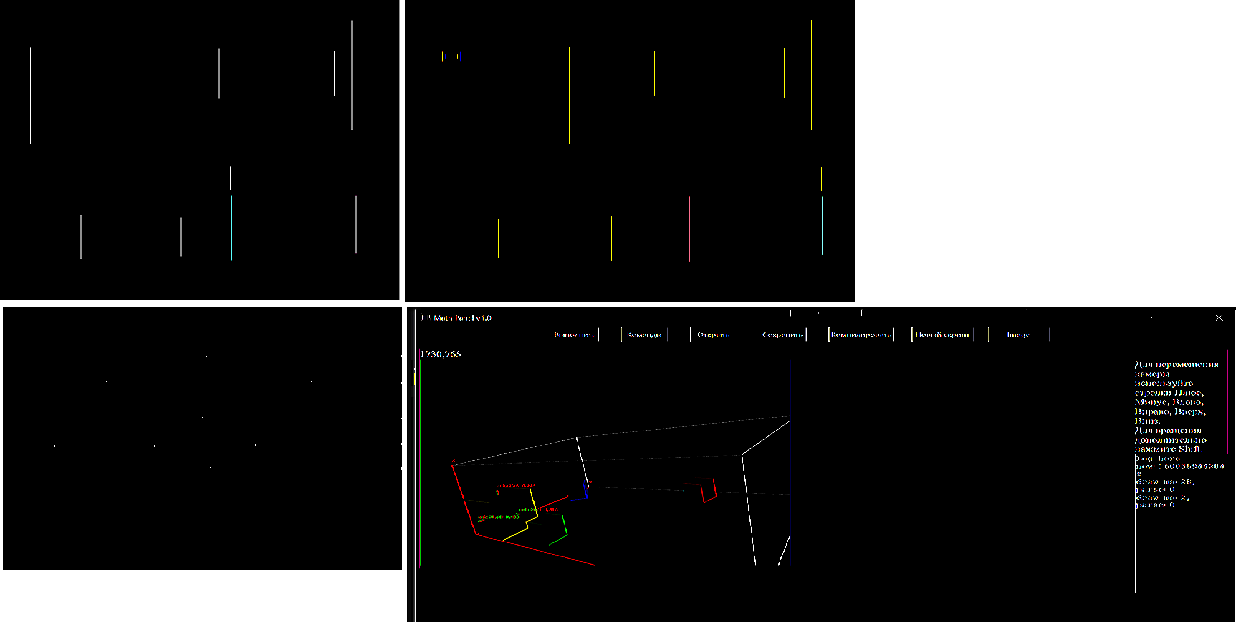


Рис.3.9. Результат совмещения отфильтрованных изображений

## 3.9. Работа с альфа-каналом

Реализуем еще 2 метода для работы с прозрачностью изображения.

/// <summary>

/// изменить альфу

/// </summary>

/// <param name="x0">горизонтальная позиция</param>

/// <param name="y0">вертикальная позиция</param>

/// <param name="alpha">новое значение alpha</param>

/// <param name="size\_x">горизонтальный размер</param>

/// <param name="size\_y">вертикальный размер</param>

public void Alpha(int x0, int y0, int alpha, int size\_x = 1, int size\_y = 1)

{

int k, red, green, blue;

for (int y = y0; y < height && y < y0 + size\_y; y++)

{

for (int x = x0; x < width && x < x0 + size\_x; x++)

{

k = y \* width + x;

var cc = System.Drawing.Color.FromArgb(map[k]);

red = cc.R;

green = cc.G;

blue = cc.B;

var cc2 = System.Drawing.Color.FromArgb(alpha, red, green, blue).ToArgb();

map[k] = cc2;

}

}

}

/// <summary>

/// наложить битмап с учетом альфы

/// </summary>

/// <param name="bm">накладываемый BitmapSimple</param>

public void Put(BitmapSimple bm)

{

if (width != bm.width || height != bm.height) return;

int alpha, red, green, blue, k;

for (int y = 0; y < height; y++)

{

for (int x = 0; x < width; x++)

{

k = y \* width + x;

var clr = System.Drawing.Color.FromArgb(bm.map[k]);

if (clr.A == 255) map[k] = bm.map[k]; //замещение

else

{ //сложение

var clr\_orig = System.Drawing.Color.FromArgb(map[k]);

alpha = clr\_orig.A;

double dInitForce = (1.0 \* clr.A) / 255;

red = SafeColor((int)(dInitForce \* clr.R + clr\_orig.R \* (1 - dInitForce)));

green = SafeColor((int)(dInitForce \* clr.G + clr\_orig.G \* (1 - dInitForce)));

blue = SafeColor((int)(dInitForce \* clr.B + clr\_orig.B \* (1 - dInitForce)));

var cc2 = System.Drawing.Color.FromArgb(alpha, red, green, blue).ToArgb();

map[k] = cc2;

}

}

}

}

Напишем простой скрипт

//test43\_pixels

using MathPanel;

using System.Net.Sockets;

using System;

using System.Collections.Generic;

///сборки

///[DLL]System.dll,System.Xaml.dll,WindowsBase.dll,PresentationFramework.dll,PresentationCore.dll,System.Drawing.dll,System.Net.dll,System.Net.Http.dll,System.Core.dll[/DLL]

///

namespace DynamoCode

{

public class Script

{

public void Execute()

{

Dynamo.Console("test43\_pixels");

//путь к папке

string sDir = @"C:\c\_devel\images\";

string[] fnames = { "red" };

System.Drawing.Color[] col\_green = { System.Drawing.Color.Green };

System.Drawing.Color[] col\_black = { System.Drawing.Color.Black };

for (int i = 0; i < fnames.Length; i++)

{

var fn = fnames[i];

//Создаем битмап зеленого цвета

var bm = new BitmapSimple(800, 600, col\_green);

//рисуем на ней красный квадрат

DateTime dt1 = DateTime.Now;

bm.Pixel(300, 200, 255, 255, 0, 0, 200, 200);

var fn\_2 = sDir + fn + "\_pix.png";

//сохраняем

bm.Save(fn\_2);

Dynamo.SetBitmapImage(fn\_2);

DateTime dt2 = DateTime.Now;

TimeSpan diff = dt2 - dt1;

int ms = (int)diff.TotalMilliseconds;

//display time span

Dynamo.Console("ms=" + ms);

//Создаем битмап черного цвета

var bm2 = new BitmapSimple(800, 600, col\_black);

//в первой половине устанавливаем alpha=200 (почти непрозрачно),

bm2.Alpha(0, 0, 200, 400, 600);

//во второй половине – 100(полупрозрачно)

bm2.Alpha(400, 0, 100, 400, 600);

//Накладываем 2-ую битмап на первую

bm.Put(bm2);

//сохраняем

fn\_2 = sDir + fn + "\_pix\_alfa.png";

bm.Save(fn\_2);

System.Threading.Thread.Sleep(2000);

Dynamo.SetBitmapImage(fn\_2);

}

}

}

}

Создаем битмап зеленого цвета, рисуем на ней красный квадрат, сохраняем. Создаем битмап черного цвета, в первой половине устанавливаем alpha=200 (почти непрозрачно),

во второй половине – 100 (полупрозрачно). Накладываем 2-ую битмап на первую, сохраняем. Результат на рис.3.10.

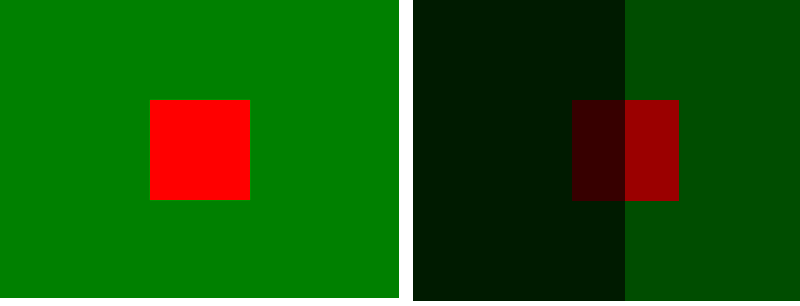


Рис.3.10. Наложение битмапов с учетом альфа-канала.