Universitatea
Transilvania
din Brașov
FACULTATEA DE MATEMATICĂ
ȘI INFORMATICĂ

Ghid de utilizare
Framework destinat cursului de procesare a imaginilor digitale

Autori: Fieraru Cristian-George
Lixandru Andreea-Bianca
Pepene Adina-Florentina

Braşov, 2023

Cuprins

Abstract **3**

1 Prezentarea framework-ului **4**
1.1 Cum descărcăm aplicaţia? 4
1.2 Arhitectura framework-ului 4
1.3 Design 5

2 Folosirea framework-ului **16**
2.1 Adăugarea unui algoritm 16
2.2 Adăugarea unei casete de dialog 17
2.3 Adăugarea unei bare de glisare 18
2.4 Adăugarea unei ferestre de plotare a unui vector 19
2.5 Folosirea clasei DrawingHelper 21

3 Ce se întâmplă dacă întâmpinăm probleme legate de framework? . . . **23**

ABSTRACT

Framework-ul a fost dezvoltat pentru a oferi studenţilor de la Facultatea de Matematică şi Informatică, din cadrul Universităţii Transilvania din Braşov, posibilitatea să adauge algoritmi specifici procesării imaginilor digitale într-un mod cât mai rapid şi mai elegant, dar şi să-i aplice asupra unor imagini pentru a putea vizualiza efectul acestora.

Acest framework oferă multiple posibilităţi de a vizualiza date despre imagini, precum: imaginea originală şi cea procesată, imaginea redimensionată, nivelul de culoare/gri din imagine. Mai mult de atât, aplicaţia conţine şi o fereastră de magnifier prin intermediul căreia pot fi observaţi pixelii dintr-o zonă selectată cu mouse-ul de către student.

Design-ul aplicaţiei a fost gândit cât mai simplu, astfel încât să le permită studenţilor să adauge uşor şi rapid butoanele în meniu, precum şi algoritmii implementaţi, să poată încărca simplu imaginile şi să le redimensioneze.

Capitolul 1

Prezentarea framework-ului

1.1 Cum descărcăm aplicaţia?

Framework-ul se află într-un repository public pe GitHub, ce poate fi accesat aici, de unde se poate clona sau se poate descărca un fişier .zip cu conţinutul framework-ului. Aplicaţia este cross-platform, fiind realizată în limbajul de programare C#, folosind WPF (Windows Presentation Foundation). Cerinţele minime necesare pentru a putea folosi aplicaţia sunt: utilizarea unui IDE ce permite folosirea limbajului de programare C#. Pentru acest framework nu există niciun fel de dependinţă.

1.2 Arhitectura framework-ului

Această aplicaţie cuprinde două proiecte: *Framework* şi *Algorithms*. Primul proiect conţine implementarea framework-ului, iar cel de-al doilea proiect este destinat utilizatorului pentru a putea crea algoritmii doriţi.

Pentru proiectul *Framework* a fost folosită o arhitectură de tipul MVVM (Model - View - ViewModel). Acest tip de arhitectură facilitează separarea dezvoltării interfeţelor grafice (View) de dezvoltarea logicii (Model), astfel încât vizualizarea să nu depindă de logică. ViewModel este responsabil pentru gestionarea interacţiunii utilizatorului şi a datelor în cadrul aplicaţiei, permiţând o separare clară între prezentarea informaţiilor şi logica aplicației.

În cadrul celui de-al doilea proiect, şi anume *Algorithms*, se află următoarele foldere:

1. Sections - conţine clasele specifice capitolelor ce vor fi discutate la cursul de Procesarea imaginilor digitale, al căror rol este de a permite adăugarea algoritmilor implementaţi de către utilizator sub forma unor metode statice;

2. Tools - conţine clasa Tools cu implementările aferente butoanelor ce vor fi discutate în secţiunea următoare;

3. Utilities - conţine clasa Utils ce permite utilizatorului inserarea unor metode ajutătoare, ce pot fi folosite de algoritmii care fac parte din secţiuni diferite.

   Un exemplu de metodă ce se găseşte în clasa Utils este *SetPixelColor*. Cu ajutorul acestei metode putem seta culoarea unui pixel aflat în imagine pe poziţia *(row,*

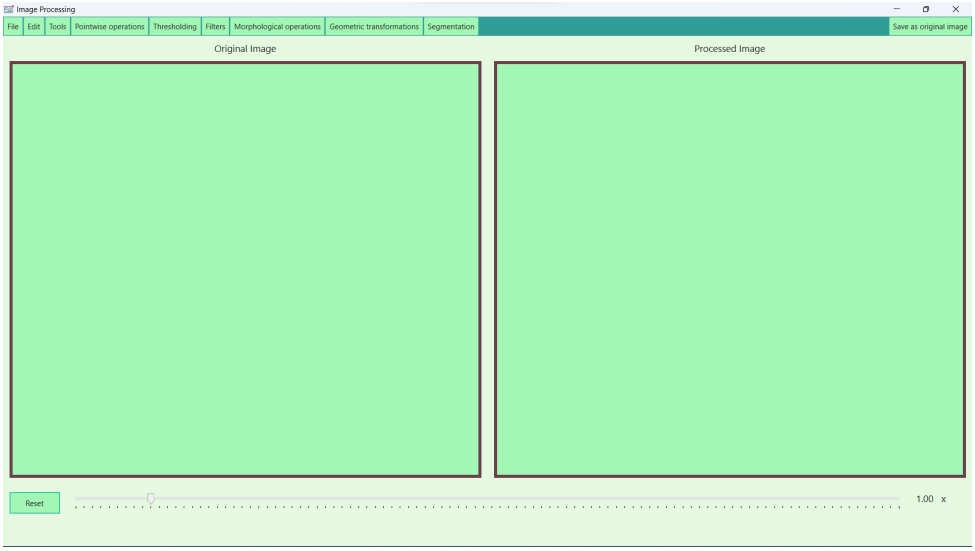
column). Dacă inputImage este o imagine color, TColor va fi Bgr. Altfel, TColor va fi Gray.

public static void SetPixelColor<TColor>(Image<TColor, byte> inputImage, int row, int column, TColor pixel) where TColor : struct, IColor

Pentru a putea folosi algoritmul şi pentru a vedea rezultatul algoritmului aplicat este nevoie să adăugăm un MenuItem în meniul aplicaţiei principale, care se află în proiectul *Framework*. Acest procedeu va fi detaliat în capitolul următor.

1.3 Design

În următoarea imagine este prezentată fereastra principală a framework-ului.



Se pot remarca cele două zone ale ferestrei în care vor apărea imaginea originală, respectiv cea procesată, în urma aplicării unui algoritm existent.

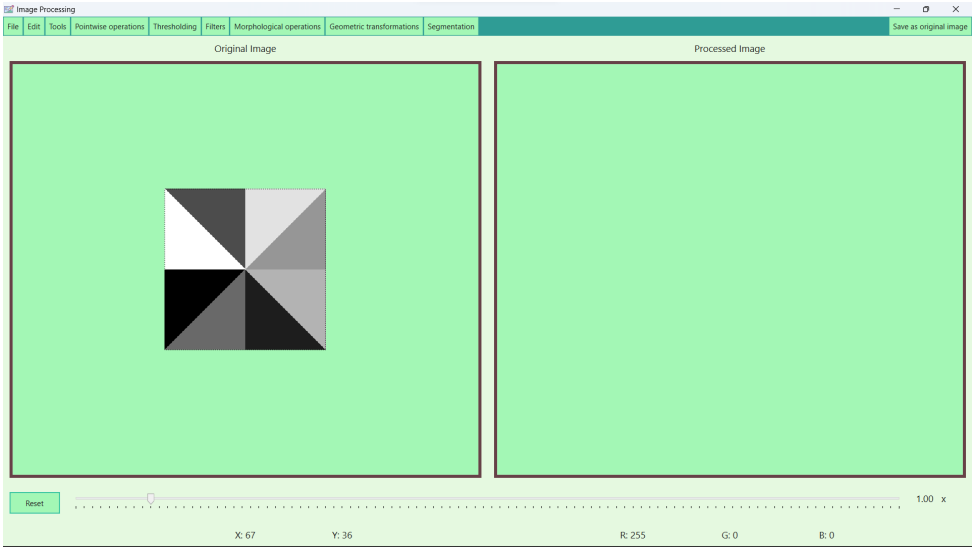
Bara de defilare existentă imediat sub cele două zone permite redimensionarea imaginilor simultan. Cele două panouri ce permit afişarea imaginilor conţin bare de defilare verticale şi orizontale, necesare atunci când imaginea depăşeşte dimensiunile ferestrei. Astfel, nu se va redimensiona automat imaginea pentru a avea loc în panou. Defilarea se face simultan în ambele imagini doar dacă există şi o imagine procesată.

Atunci când este încărcată o imagine în această fereastră, în partea de jos vor apărea valorile x şi y ale poziţiei mouse-ului pe imagine, împreună cu valoarea de gri pentru o imagine alb-negru, respectiv cea de RGB pentru o imagine color, corespunzătoare pixelului din imagine aflat la acea poziţie a mouse-ului.

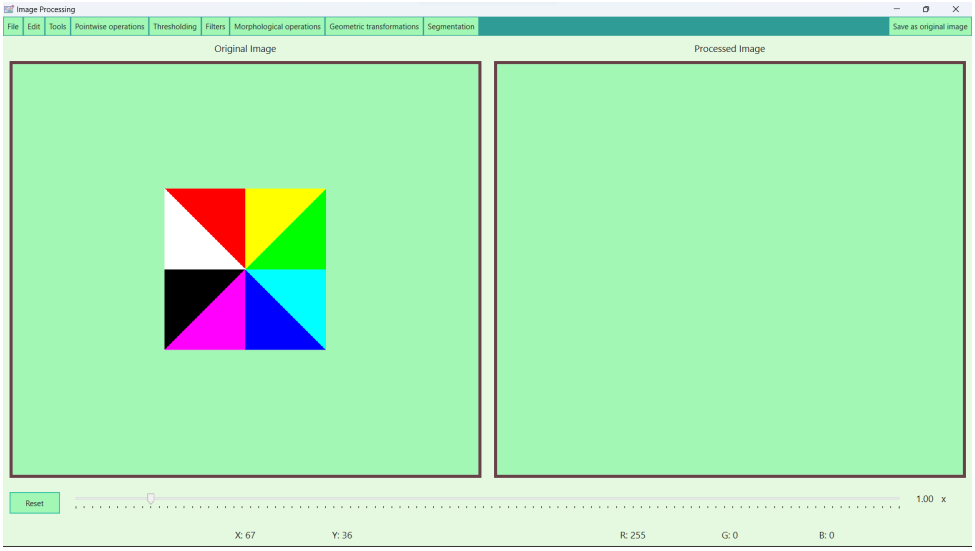
Se mai pot observa butoanele principale:

1. File

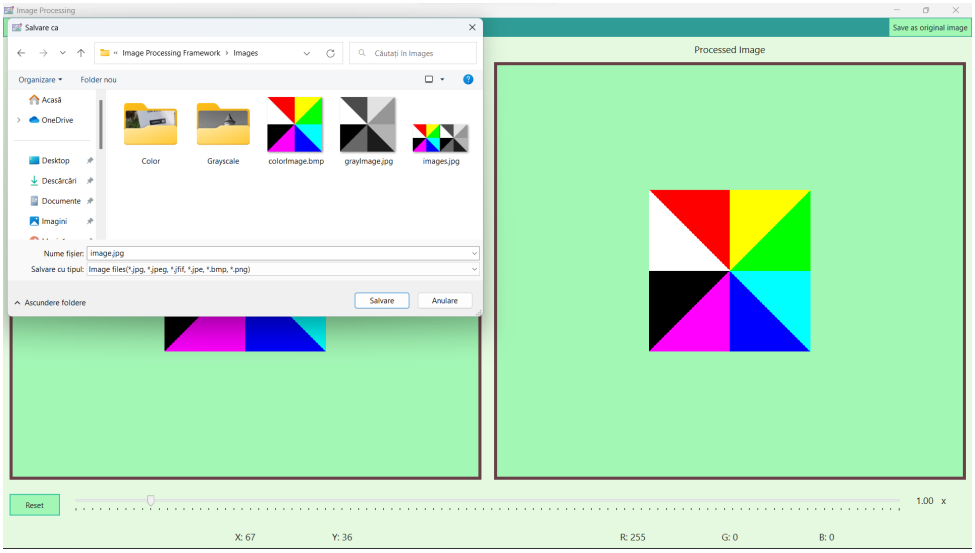
   - Load grayscale image - din fereastra Explorer se selectează imaginea ce se doreşte a fi încărcată în varianta alb-negru;



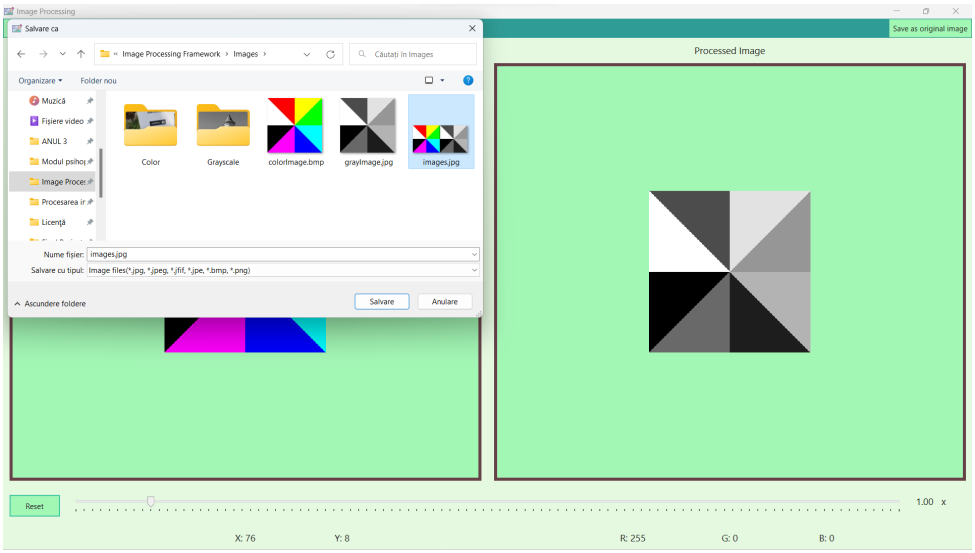
   - Load color image - din fereastra Explorer se poate încărca o imagine color spre a fi procesată;



- Save processed image - imaginea procesată cu ajutorul framework-ului poate fi salvată în calculatorul utilizatorului;



- Save both images - imaginea originală, cât şi cea procesată vor fi combinate într-o singură imagine ce poate fi salvată în calculatorul utilizatorului;



- Exit - va închide fereastra principală, precum şi celelalte ferestre deschise posibil anterior.

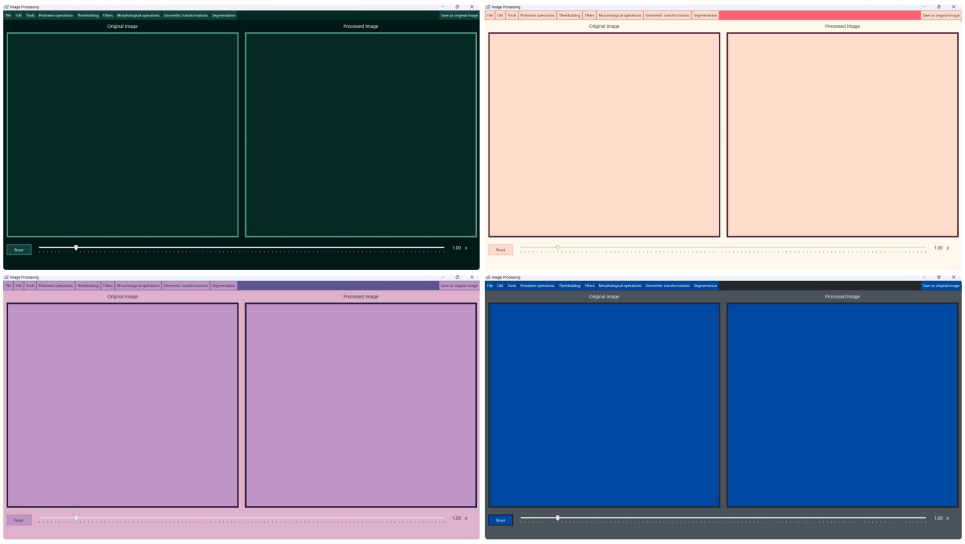
2. Edit

- Remove drawn shapes from initial canvas - şterge toate formele geometrice desenate pe canvasul unde este încărcată imaginea originală;
- Remove drawn shapes from processed canvas - şterge toate formele geometrice desenate pe canvasul unde este afişată imaginea procesată;
- Remove drawn shapes from both canvases - şterge toate formele geometrice desenate pe cele două canvasuri;
- Clear initial canvas - goleşte canvasul iniţial;

- Clear processed canvas - goleşte canvasul procesat;
- Clear both canvases - goleşte ambele canvasuri şi închide toate ferestrele deschise.

3. Tools

- Themes - permite utilizatorului să aleagă propriul stil de afişare al interfeţelor grafice disponibile;



Utilizatorul îşi poate crea propriile teme pe baza unor palete de culori. Pentru a efectua acest lucru, va trebui să urmăm aceşti paşi:

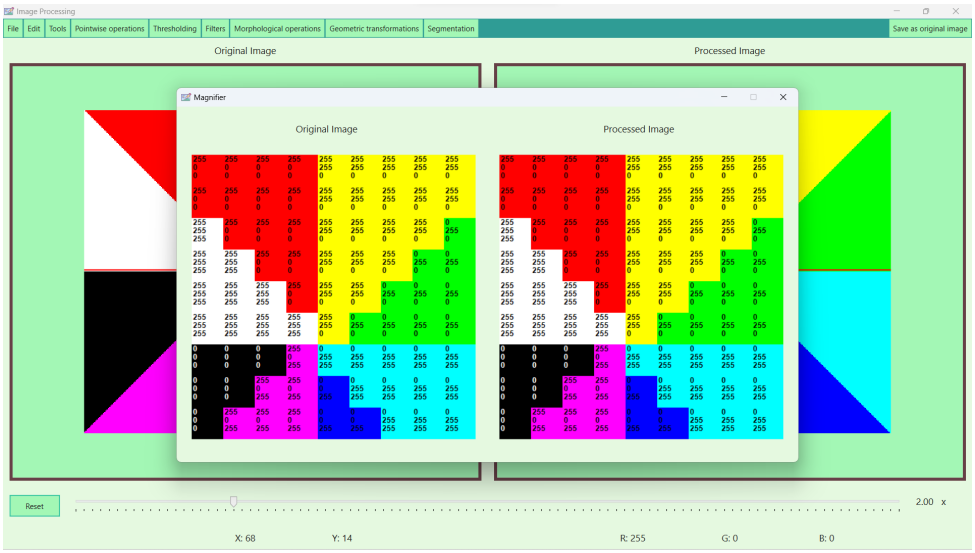
– Vom merge în folder-ul *Model*, subfolder-ul *ThemeModes* şi vom adăuga o clasă cu numele paletei, urmat de Theme pentru a respecta convenţia de denumire. Această clasă trebuie să implementeze interfaţa *IThemeMode*, apoi utilizatorul va iniţializa fiecare proprietate cu codul de culoare dorit. Trebuie creată o singură instanţă statică a acestei clase ce poate fi folosită oriunde în aplicaţie;

– Vom merge în folder-ul *ViewModel*, clasa *MainVM* şi vom adăuga un case nou în switch-ul metodei *SetThemeMode* pe care vom returna instanţa clasei menţionată anterior;

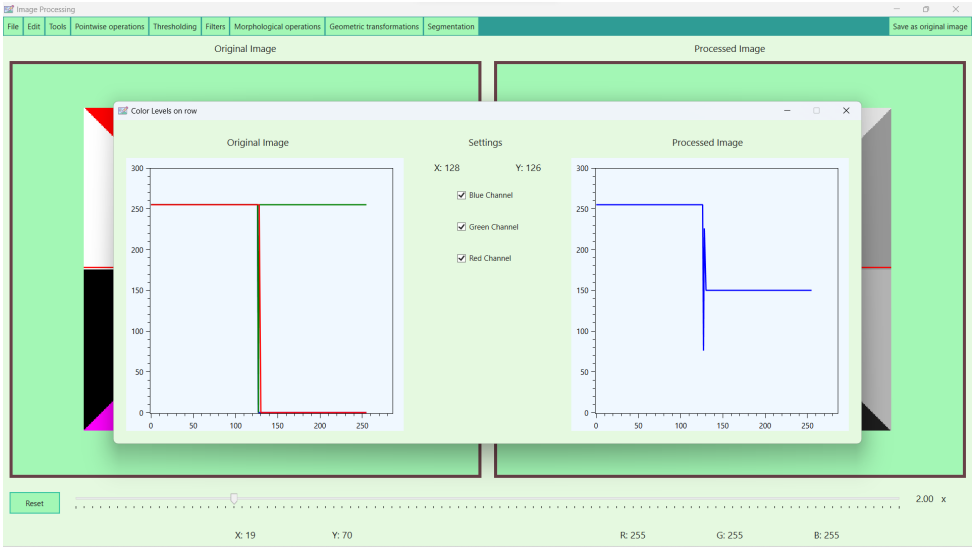
– Vom merge în folder-ul *View*, fişierul *MainWindow.xaml* şi vom adăuga un nou item în meniul Themes;

– Vom merge în folder-ul *View*, fişierul *MainWindow.xaml.cs* şi vom adăuga o nouă condiţie in metoda *InitializeThemeMode*, care să trateze cazul în care noul item adăugat în meniu este apăsat.

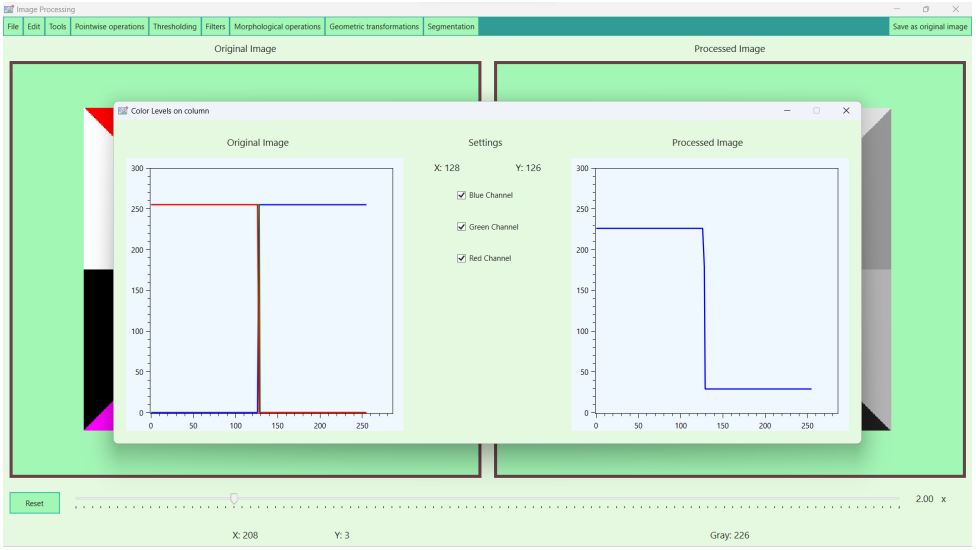
- Magnifier - va deschide o nouă fereastră care va oferi detalii despre imaginea color sau cea alb-negru, precum nuanţa culorii şi valoarea acesteia, pentru un dreptunghi de 9 x 9 creat în jurul punctului unde utilizatorul a dat click cu mouse-ul;



- Gray/Color levels on row - deschide o nouă fereastră în care se desenează un grafic cu nivelurile de gri, respectiv RGB, de pe linia orizontală creată acolo unde a fost dat click pe imagine;



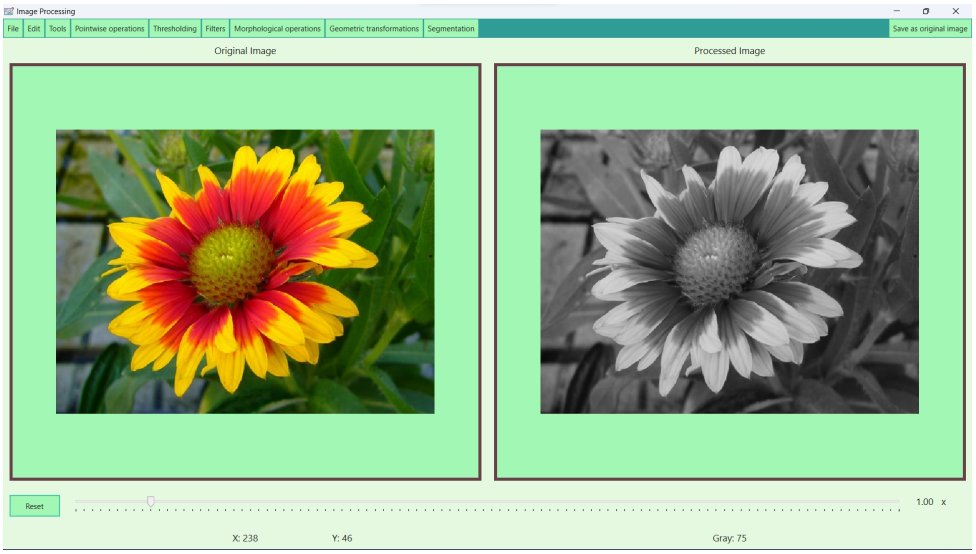
- Gray/Color levels on column - deschide o nouă fereastră în care se desenează un grafic cu nivelurile de gri, respectiv RGB, de pe linia verticală creată acolo unde a fost dat click pe imagine;



- Visualize image histogram

  Histograma unei imagini alb-negru redă distribuţia nivelurilor de gri din imagine. Este funcţia care asociază fiecărui nivel de gri frecvenţa de apariţie în imagine. Dacă se consideră o imagine iniţială cu o rezoluție de 8 biţi/pixel, pe axa abscisei graficului histogramei se regăsesc cele 256 niveluri de gri posibile, iar pe axa ordonatei sunt redate frecvenţele de apariţie ale fiecărui nivel de gri de pe axa abscisei.

  În cazul unei imagini color sunt redate distribuţiile nivelelor de culoare din imagine. Astfel, vom avea histograme pe fiecare canal de culoare în parte.

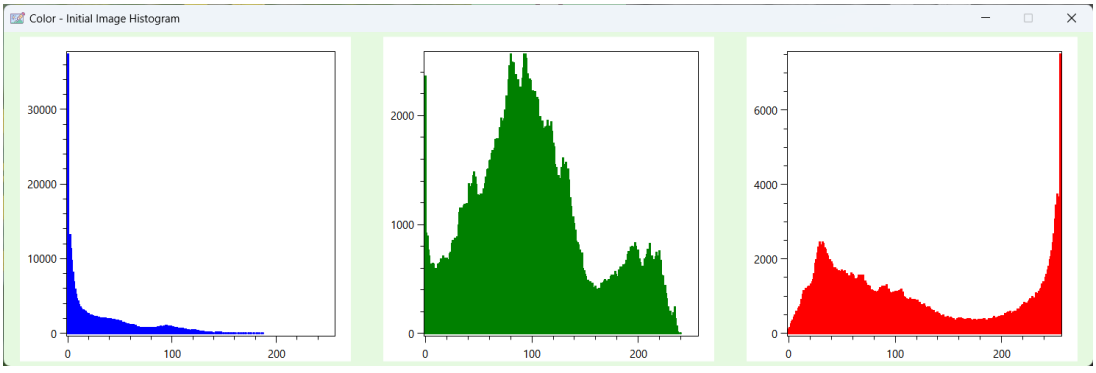


  Considerăm o imagine cu nivelurile de gri [0, 255] şi n numărul total de pixeli. Atunci histograma se defineşte ca fiind funcţia discretă

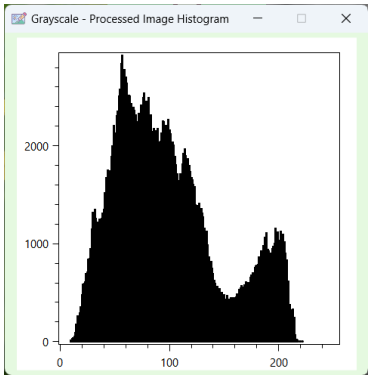
$$h : [0, 255] \to \mathbb{N}, h(k) = n_k,$$

  unde n_k = numărul de pixeli de culoare (nivel de gri) k din imagine.

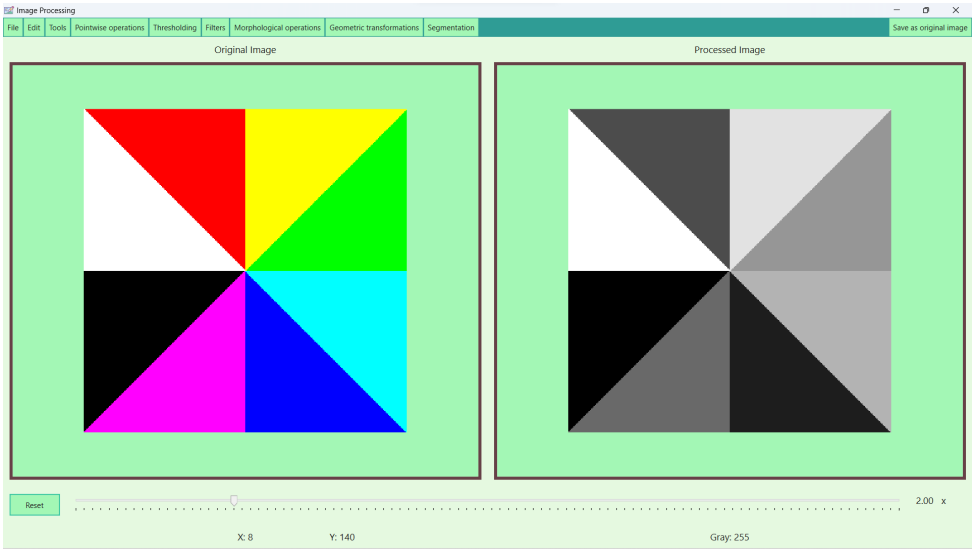
  - Initial image histogram - calculează histograma imaginii originale/iniţiale şi o reprezintă grafic;



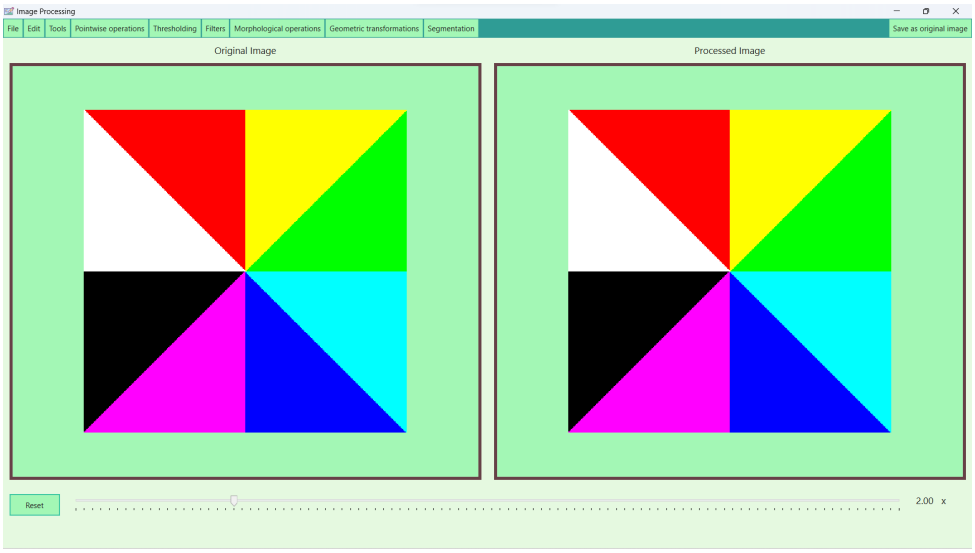
– Processed image histogram - calculează histograma imaginii procesate şi o reprezintă grafic.

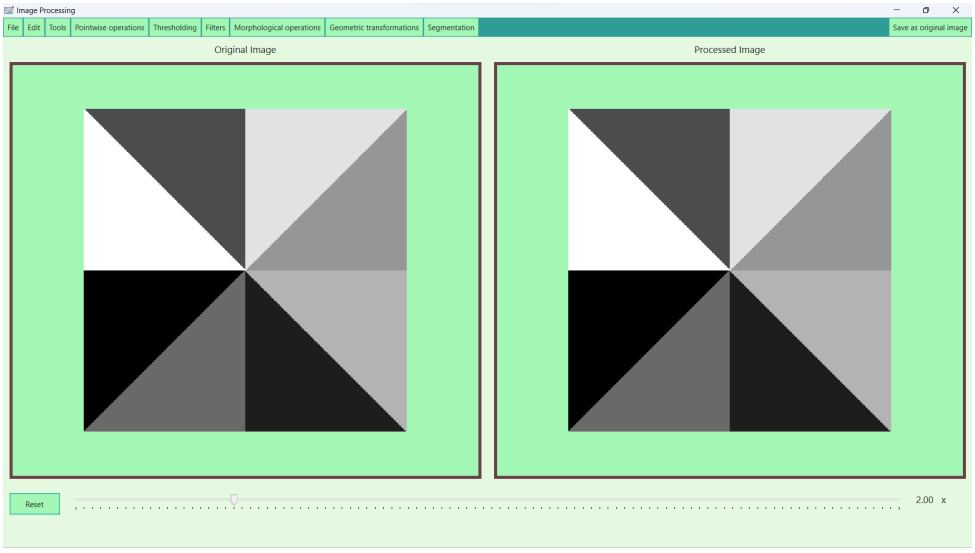


- Transform to gray image - această opţiune este disponibilă doar pentru imaginea color, oferind posibilitatea de a transforma pixelii RGB în pixeli gri, pentru o prelucrare ulterioară a imaginii rezultate.

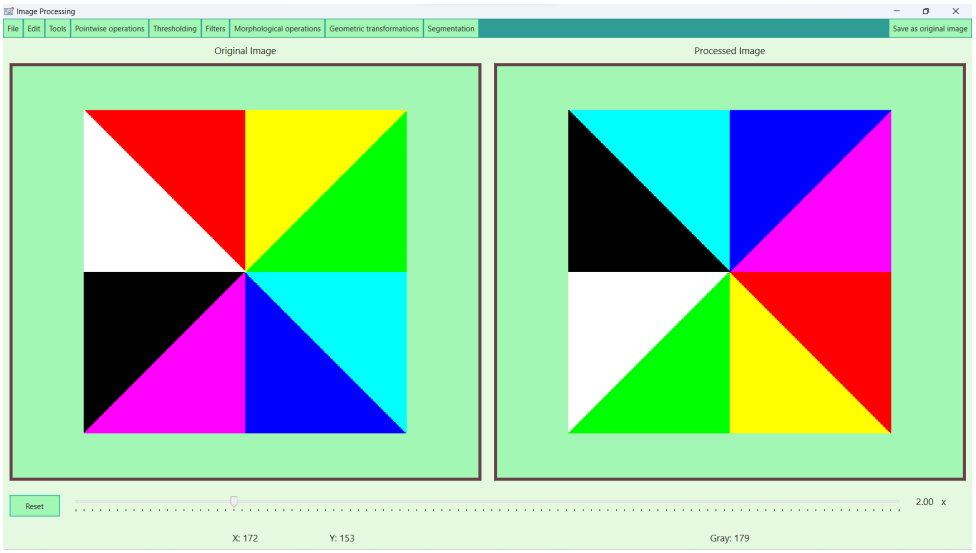


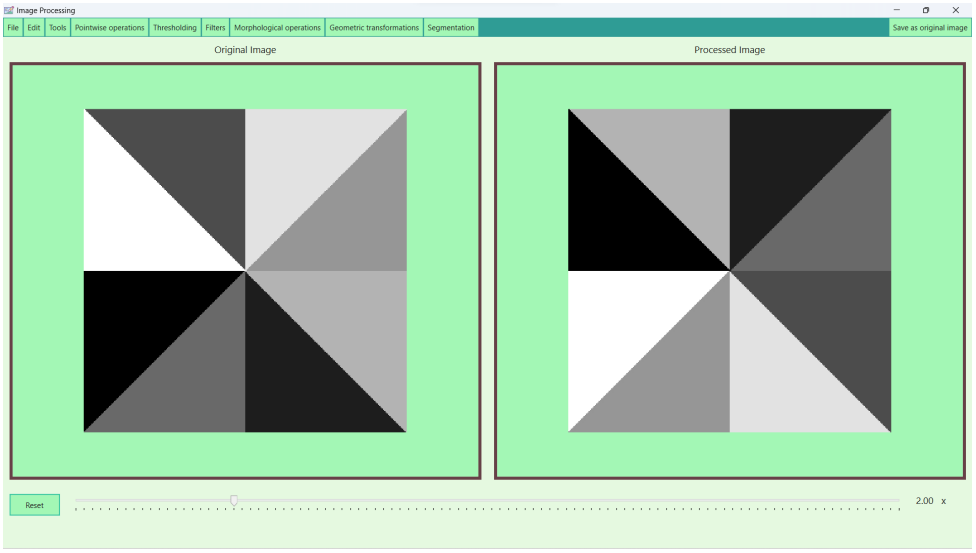
- Copy - copiază imaginea originală şi o plasează ca fiind imagine procesată;



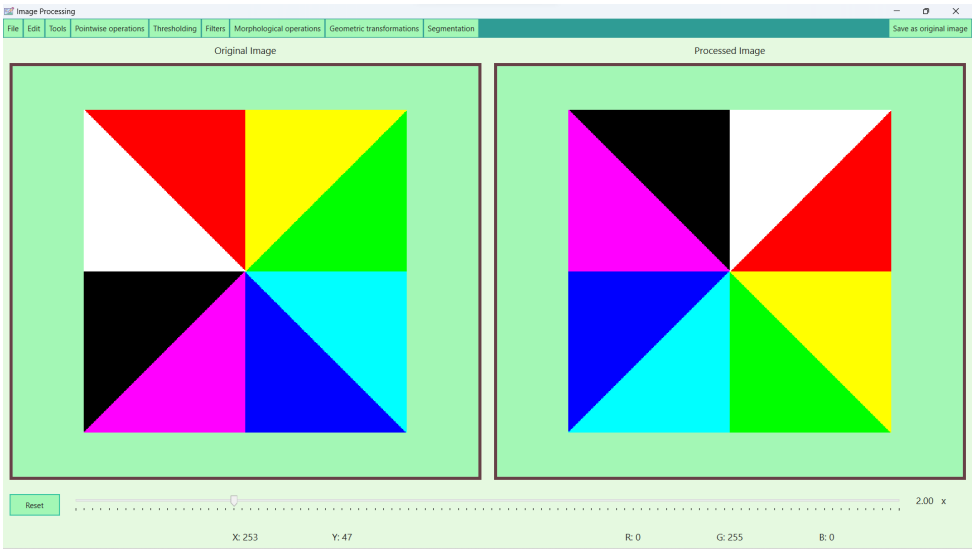


- Invert - imaginea procesată va avea valoarea fiecărui pixel egală cu 255 minus valoarea pixelului respectiv din imaginea originală;

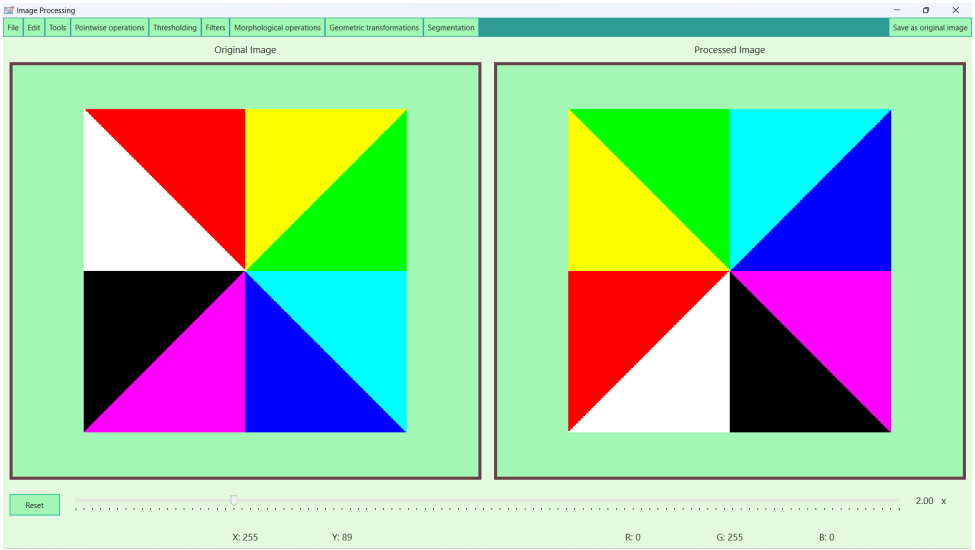




- Rotate image
  - Clockwise - roteşte imaginea în sensul acelor de ceasornic cu 90 de grade;



  - Anti-clockwise - roteşte imaginea în sensul opus acelor de ceasornic cu 90 de grade.



4. Următoarele şase meniuri (Pointwise operations → Segmentation) reprezintă zonele unde utilizatorul va adăuga butoane cu care se vor putea aplica algoritmii implementaţi.

5. Save as original image - acest buton are ca rezultat mutarea imaginii procesate pe canvasul imaginii originale, pentru a putea fi aplicaţi în continuare alţi algoritmi asupra imaginii modificate anterior.

6. Reset - acest buton setează dimensiunea imaginii la dimensiunea originală.

Capitolul 2

Folosirea framework-ului

2.1 Adăugarea unui algoritm

Pentru a putea adăuga un nou algoritm trebuie să parcurgem următoarele etape:

1. Trebuie să adăugăm o metodă ce conţine implementarea algoritmului în proiectul Algorithms, folder-ul Sections, clasa corespunzătoare din care face parte acesta. Metoda trebuie să fie publică şi statică pentru a putea fi accesată oricând şi oriunde în interiorul proiectului Framework.

2. Trebuie să ne ducem în proiectul Framework, fereastra MainWindow, fişierul MainWindow.xaml şi să căutăm meniul.

```
36   <Menu Background="{Binding Path=Theme.MenuBackgroundColor}"
37         Grid.Row="0"
38         Grid.ColumnSpan="2">
39       <Menu.ItemsPanel...>
44
45       <MenuItem Header="File"...>
80
81       <MenuItem Header="Edit"...>
122
123      <MenuItem Header="Tools"...>
183
184      <MenuItem Header="Pointwise operations"...>
189
190      <MenuItem Header="Thresholding"...>
195
196      <MenuItem Header="Filters"...>
208
209      <MenuItem Header="Morphological operations"...>
214
215      <MenuItem Header="Geometric transformations"...>
220
221      <MenuItem Header="Segmentation"...>
226
227      <MenuItem Header="Save as original image"...>
239  </Menu>
```

3. Este necesar să adăugăm un nou MenuItem în secţiunea dorită.

   Acest lucru se poate face cu ajutorul următoarei secvenţe de cod:
   <MenuItem Header="My Algorithm"
   Command="{Binding Path=MenuCommands.MyCommand}"/>

4. În momentul de faţă avem: implementarea algoritmului nostru şi butonul cu care vom executa algoritmul. Totuşi, lipseşte conexiunea dintre buton şi algoritm. Pentru

a realiza această conexiune trebuie să implementăm o comandă în clasa MenuCommands din subfolder-ul Commands al folder-ului părinte ViewModel.

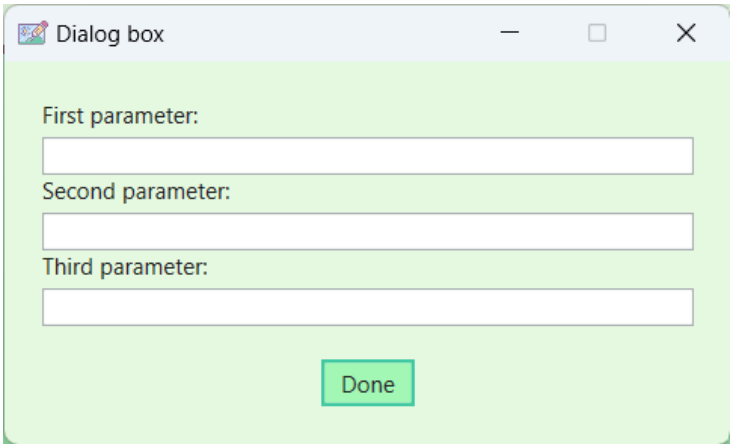
Pentru a face acest lucru vom adăuga în clasa MenuCommands un membru privat *MyCommand*, o metodă care va apela algoritmul şi o proprietate publică de tipul ICommand cu numele *MyCommand*, ales la pasul anterior.

```
private ICommand _myCommand;
0 references
public ICommand MyCommand
{
    get
    {
        if (_myCommand == null)
            _myCommand = new RelayCommand(MyMethod);
        return _myCommand;
    }
}

1 reference
private void MyMethod(object parameter)
{

}
```

După toţi aceşti paşi algoritmul adăugat ar trebui să funcţioneze. Trebuie acordată o atenţie sporită atunci când implementăm *MyMethod*, pentru a nu crea bug-uri în cazul în care avem un algoritm pentru imagini alb-negru, dar care nu funcţionează pentru imagini color, sau chiar invers. Aceste situaţii pot fi uşor tratate cu ajutorul unei structuri decizionale simple unde verificăm care dintre cele două tipuri de imagini este nenulă şi ce algoritm dorim să aplicăm asupra acestora.

2.2 Adăugarea unei casete de dialog

Pentru algoritmii realizaţi există posibilitatea să avem nevoie de a folosi parametri. În acest sens, în proiectul Framework am creat în folder-ul View, o altă fereastră numită DialogBox, folosită pentru a putea introduce unul sau mai mulţi parametri ce pot fi folosiţi pentru algoritmi.



Pentru a putea folosi această fereastră se va crea în comanda pentru algoritm un obiect de tipul DialogBox, de unde se va putea extrage valoarea sau valorile introduse în căsuţele text. Imaginea de mai jos ajută la înţelegerea modului de folosire a casetei de dialog. Textul introdus va reprezenta parametrul care va fi folosit ulterior pentru algoritm.

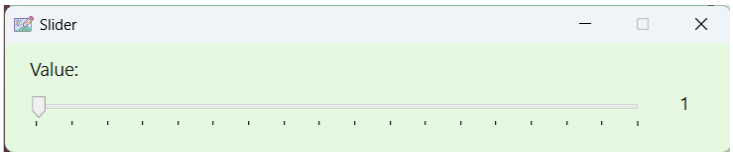
```
List<string> parameters = new List<string>
{
    "First parameter: ",
    "Second parameter: ",
    "Third parameter: ",
};

DialogBox window = new DialogBox(_mainVM, parameters);
window.ShowDialog();

List<double> values = window.GetValues();
if (values != null)
{
    /* Do something with these values */

    double stParam = values[0];
    double ndParam = values[1];
    double rdParam = values[2];
}
```

Constructorul clasei DialogBox primeşte lista de parametri, de tipul *List<string>*, cu ajutorul căreia creăm dinamic căsuţele pentru fiecare parametru. Fiecare string va genera un TextBlock şi un TextBox, unde TextBlock-ul va reprezenta numele parametrului, iar TextBox-ul va fi parametrul. Pentru a putea lua valorile acestor parametri am conceput metoda *GetValues()*, care va întoarce un *List<string>*, reprezentând parametrii doriţi. Aceştia vor veni în ordinea în care a fost dată lista cu numele parametrilor.

2.3 Adăugarea unei bare de glisare

Contextul de utilizare este asemănător cu cel al casetei de dialog descrise anterior. Dacă algoritmul implementat depinde de un singur parametru, cu ajutorul acestei bare de glisare putem să generăm valori pe care metoda noastră să le poată utiliza.

Avantajul folosirii acestei ferestre este că imaginea procesată se actualizează în timp real, în funcţie de mutarea la stânga sau la dreapta a cursorului. Astfel, dacă dorim să vizualizăm în mod rapid cum este influenţată imaginea rezultat în funcţie de valorile alese, folosirea acestei ferestre este opţiunea cea mai bună.



Pentru a putea folosi această fereastră trebuie creat un obiect de tipul *SliderWindow* în metoda corespunzătoare comenzii pe care dorim să o executăm. Prin constructorul ferestrei putem transmite sub formă de string numele parametrului.

Metoda *ConfigureSlider* ajută la setarea proprietăţilor de care are nevoie Slider-ul pentru a putea fi funcţional. Parametrii *minimumValue* şi *maximumValue* reprezintă capătul inferior, respectiv superior ale barei de glisare, *value* este valoarea la care vom

plasa cursorul, iar *frequency* determină frecvenţa cu care valoarea se modifică atunci când se execută defilarea cursorului la stânga sau la dreapta. Dacă nu oferim aceste valori ca argumente metodei descrise, vor fi utilizate valorile implicite ale acestora.

Cu ajutorul metodei *SetWindowData* trebuie să setăm imaginea originală, dar şi algoritmul care se aplică asupra imaginii atunci când se modifică valoarea glisorului.

În imaginea de mai jos este prezentat un exemplu de utilizare al acestei ferestre:

```
if (SliderOn == true) return;

if (InitialImage == null)
{
    MessageBox.Show("Please add an image!");
    return;
}

SliderWindow sliderWindow = new SliderWindow(_mainVM, "Value: ");
sliderWindow.ConfigureSlider(
    minimumValue: 0,
    maximumValue: 255,
    value: 1,
    frequency: 15);

if (GrayInitialImage != null)
{
    sliderWindow.SetWindowData(
        image: GrayInitialImage,
        algorithm: Filters.AlgorithmToApply);
}
else if (ColorInitialImage != null)
{
    sliderWindow.SetWindowData(
        image: ColorInitialImage,
        algorithm: Filters.AlgorithmToApply);
}

sliderWindow.Show();
```

Observăm că funcţia transmisă ca argument metodei *SetWindowData* este un filtru intitulat AlgorithmToApply. Pentru a putea utiliza această fereastră este obligatoriu ca algoritmul creat să primească ca parametri o imagine (Bgr sau Gray), o valoare de tip double, iar ca tip returnat trebuie să fie o imagine (Bgr sau Gray).

```
2 references
public class Filters
{
    1 reference
    public static Image<Bgr, byte> AlgorithmToApply(Image<Bgr, byte> image, double value)
    {
        throw new NotImplementedException();
    }

    1 reference
    public static Image<Gray, byte> AlgorithmToApply(Image<Gray, byte> image, double value)
    {
        throw new NotImplementedException();
    }
}
```

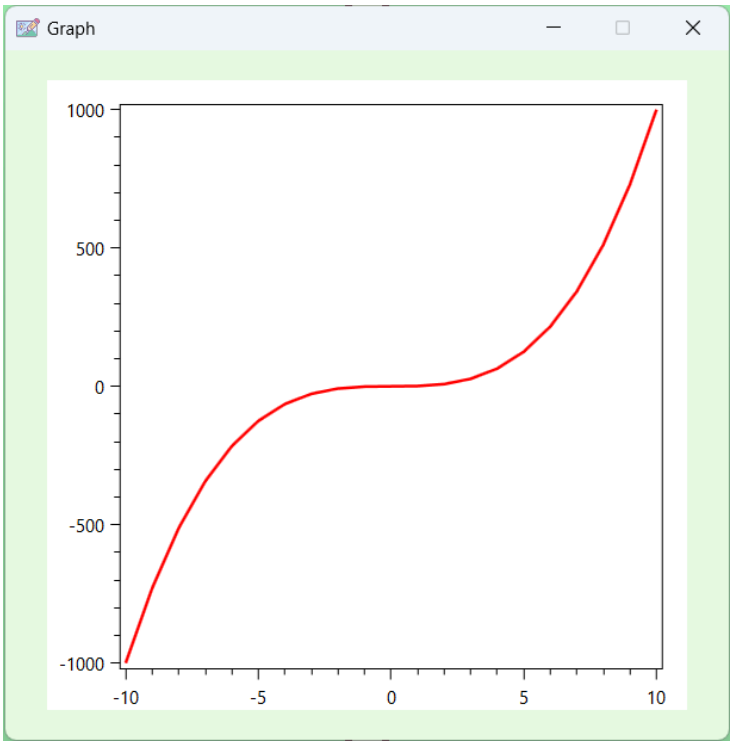
2.4 Adăugarea unei ferestre de plotare a unui vector

În cazul în care dorim să adăugăm o nouă fereastră în care desenăm un grafic al unui vector, vom folosi clasa *GraphWindow*. Această clasă primeşte ca parametru colecţia de puncte, de tipul *PointCollection*, şi le desenează. Pentru a declara o astfel de colecţie de puncte, este nevoie să scriem la începutul fişierului *using PointCollection = System.Windows.Media.PointCollection;*.

Pentru a crea o fereastră de plotare a unui vector vom realiza o comandă precum am folosit şi în cazul casetei de dialog. Un astfel de exemplu este prezentat în imaginea de mai jos:

```
PointCollection points = new PointCollection();

for (double index = -10; index <= 10; ++index)
    points.Add(new Point(index, System.Math.Pow(index, 3)));

GraphWindow window = new GraphWindow(_mainVM, points);
window.Show();
```

Rezultatul secvenţei de cod poate fi văzut în imaginea următoare:



2.5 Folosirea clasei DrawingHelper

Prin intermediul clasei *DrawingHelper* putem adăuga forme geometrice pe imagini. Astfel, avem următoarele metode:

1. *public static Line DrawLine(Canvas canvas, Point start, Point end, int thickness, Brush color, double scaleValue)* - permite desenarea unei linii de la punctul de start la punctul de end;

```
Point start = new Point(20, 20);
Point end = new Point(100, 20);

DrawLine(initialCanvas, start, end, 2, Brushes.Red, ScaleValue);
DrawLine(processedCanvas, start, end, 2, Brushes.Red, ScaleValue);
```

2. *public static Rectangle DrawRectangle(Canvas canvas, Point leftTop, Point rightBottom, int thickness, Brush color, double scaleValue)* - permite desenarea unui dreptunghi bazându-ne pe punctul stânga sus *leftTop* şi punctul dreapta jos *rightBottom*;

```
Point start = new Point(20, 20);
Point end = new Point(100, 100);

DrawRectangle(initialCanvas, start, end, 2, Brushes.Red, ScaleValue);
DrawRectangle(processedCanvas, start, end, 2, Brushes.Red, ScaleValue);
```

3. *public static Ellipse DrawEllipse(Canvas canvas, Point center, double width, double height, int thickness, Brush color, double scaleValue)* - permite desenarea unei elipse de dimensiuni *width* şi *height* cu centrul *center*;

```
Point center = new Point(100, 100);
double width = 50, height = 70;

DrawEllipse(initialCanvas, center, width, height, 2, Brushes.Red, ScaleValue);
DrawEllipse(processedCanvas, center, width, height, 2, Brushes.Red, ScaleValue);
```

4. *public static Polygon DrawPolygon(Canvas canvas, PointCollection vectorOfPoints, int thickness, Brush color, double scaleValue)* - permite desenarea unui poligon pornind de la colecţia *vectorOfPoints*.

```
DrawPolygon(initialCanvas, VectorOfMousePosition, 2, Brushes.Red, ScaleValue);
DrawPolygon(processedCanvas, VectorOfMousePosition, 2, Brushes.Red, ScaleValue);
```

Scriind doar o linie de cod, putem desena pe canvas ce formă geometrică dorim. Însă, cu toate acestea, observăm că *initialCanvas* şi *processedCanvas* sunt elemente specifice interfeţelor grafice. În primă fază pare că principiul şablonului de proiectare MVVM nu este respectat. Existenţa elementelor de interfaţă în ViewModel nu este permisă. Adevărul este că ne folosim de parametrul metodei lansate prin executarea comenzii pentru a prelua informaţii specifice View.

```
private ICommand _myCommand;
0 references
public ICommand MyCommand
{
    get
    {
        if (_myCommand == null)
            _myCommand = new RelayCommand(MyMethod);
        return _myCommand;
    }
}

1 reference
private void MyMethod(object parameter)
{
    var canvases = (object[])parameter;
    var initialCanvas = canvases[0] as Canvas;
    var processedCanvas = canvases[1] as Canvas;

    DrawPolygon(initialCanvas, VectorOfMousePosition, 2, Brushes.Red, ScaleValue);
    DrawPolygon(processedCanvas, VectorOfMousePosition, 2, Brushes.Red, ScaleValue);
}
```

Atunci când dorim să trimitem parametri prin intermediul unei comenzi în WPF, putem utiliza *CommandParameter*. Cu toate acestea, CommandParameter poate accepta doar un singur obiect ca parametru.

Dacă dorim să trimitem mai multe valori, de exemplu două valori, putem folosi un converter pentru a le transforma într-un singur obiect. Converter-ul este o clasă care implementează interfaţa *IMultiValueConverter*. Aceasta oferă două metode, *Convert* şi *ConvertBack*, care sunt folosite pentru a face conversie între tipurile de date. De obicei, utilizăm metoda Convert pentru a converti mai multe valori într-un singur obiect, care poate fi apoi transmis prin CommandParameter.

```
<MenuItem Header="My Algorithm"
          Foreground="#1E1E1E"
          Command="{Binding Path=MenuCommands.MyCommand}">
    <MenuItem.CommandParameter>
        <MultiBinding Converter="{StaticResource arrayMultiValueConverter}">
            <Binding ElementName="canvasOriginalImage"/>
            <Binding ElementName="canvasProcessedImage"/>
        </MultiBinding>
    </MenuItem.CommandParameter>
</MenuItem>
```

Utilizarea unui converter ne permite să trimitem mai multe valori prin intermediul CommandParameter, ceea ce face mai uşor de transmis date între View şi ViewModel în WPF. Pot fi vizualizate exemple de utilizare a celor discutate chiar în framework (comenzile specifice meniului *Edit*, etc).

VectorOfMousePosition este un vector care păstrează toate poziţiile selectate de mouse cu click stânga. Pentru a elibera valorile reţinute în vectorul de poziţii vom apăsa click dreapta în interiorul unei imagini, indiferent dacă aceasta este procesată sau nu.

Capitolul 3

Ce se întâmplă dacă întâmpinăm probleme legate de framework?

Pe parcursul lucrului cu framework-ul prezentat, dacă se întâlnesc dificultăţi în adăugarea unui algoritm sau se produc diverse erori la compilare, este recomandat ca prim pas să se trimită un report pe repo, iar apoi să se trimită un mesaj la unul dintre autori, în care să se descrie situaţia dorită şi care este scenariul prin care s-a ajuns la problemă.