

Pixelium Képszerkesztő

Dokumentáció



2025. november 3.

Tartalomjegyzék

1. Bevezetés	3
1.1. A Projekt Célja	3
1.2. Főbb Jellemzők	3
2. Telepítés és Futtatás	4
2.1. Előfeltételek	4
2.2. Fordítás a Forráskódból	4
2.2.1. Gyors fordítás	4
2.2.2. Többplatformos fordítás	4
2.3. Támogatott Platformok	4
2.4. Az Alkalmazás Indítása	5
2.4.1. Forráskódból történő indítás	5
2.4.2. Előre buildelt binárisok indítása	5
3. Felhasználói Útmutató	6
3.1. Kezdő lépések	6
3.1.1. Alkalmazás indítása	6
3.1.2. Kép megnyitása	6
3.1.3. Mintaképek betöltése	6
3.2. A Felület Ismertetése	6
3.3. Képszerkesztés és Szűrők Alkalmazása	7
3.3.1. Alapvető Szerkesztések	7
3.3.2. Speciális Szűrők	7
3.4. Hisztogram Analízis	7
3.4.1. Hisztogram megjelenítése	7
3.4.2. Hisztogram kiegyenlítés	8
3.5. Rétegek Kezelése	8
3.5.1. Rétegek létrehozása és kezelése	8
3.5.2. Réteg tulajdonságok	8
3.5.3. Rétegek egyesítése	8
3.6. Visszavonás és Ismétlés (Undo/Redo) műveletek	8
3.7. Billentyűparancsok	9
3.8. Képek Mentése	9
3.8.1. Támogatott formátumok	9
3.8.2. Mentési lehetőségek	9

4. Fejlesztői Dokumentáció	10
4.1. Architektúra és Fő Komponensek	10
4.1.1. Pixelium.Core	10
4.1.2. Pixelium.UI	10
4.2. Kulcs technológiák	10
4.3. Teljesítményoptimalizálások	11
4.3.1. Lookup Table (LUT) Gyorsítótárazás	11
4.3.2. Párhuzamos Feldolgozás (Multithreading)	11
4.3.3. Unsafe Kód és Pointer Műveletek	12
4.4. Képmegjelenítés és Formátumok	13
4.4.1. Belső Formátum	13
4.4.2. Be- és Kimeneti Formátumok	13
4.5. Képfeldolgozási Algoritmusok	15
4.5.1. Pixelenkénti Transzformációk	15
4.5.2. Geometriai Transzformációk	16
4.5.3. Konvolúciós Szűrők	17
4.5.4. Harris Sarokérzékelés	19
4.5.5. Hisztogram Alapú Műveletek	20
4.6. Parancs Minta és Undo/Redo Funkció	21
4.6.1. IImageCommand Interfész	21
4.6.2. CommandHistory	21
4.6.3. FilterCommand	21
5. Képernyőképek	22
5.1. Alkalmazás Felület	22
5.2. Hisztogram Analízis	23
5.3. Élérzékelés	24
5.4. Réteges Szerkesztés	25
5.5. Elmosási Szűrők	26

1. fejezet

Bevezetés

1.1. A Projekt Célja

A Pixelium egy nagy teljesítményű, többplatformos képszerkesztő alkalmazás, amely a .NET 9.0 és Avalonia UI technológiákra épül. Célja, hogy alapvető képszerkesztési funkciókat biztosítson gyors és modern felületen, a párhuzamosság és gyorsítótárok kihasználásával.

1.2. Főbb Jellemzők

- **Nagy teljesítmény:** Párhuzamos feldolgozás és memória-optimalizálás
- **Multiplatform:** Linux, Windows, macOS támogatás
- **Réteges szerkesztés:** Rétegkezelésben alapuló szerkesztés
- **Szűrők:** Gauss elmosás, élérzékelés, sarokdetektálás
- **Hisztogram analízis:** Hisztogram generálás a bemeneti képről
- **Visszavonás/ismétlés:** Undo/Redo művelet

2. fejezet

Telepítés és Futtatás

2.1. Előfeltételek

A Pixelium Fedora 42 Linux rendszeren került fejlesztésre, szem előtt tartva a keretrendszer kiválasztásakor a multiplatform támogatást. A program futtatható Linux, Windows és Mac rendszereken is.

2.2. Fordítás a Forráskódból

A Pixelium fordításához a .NET 9.0 SDK-ra van szükség.

2.2.1. Gyors fordítás

```
1 dotnet build Pixelium.sln  
2  
3 dotnet run --project Pixelium.UI/Pixelium.UI.csproj
```

2.2.2. Többplatformos fordítás

A `build-all.sh` szkript segítségével minden támogatott platformra elkészíthetők a bináris fájlok:

```
1 ./build-all.sh
```

A fordítási eredmények a `builds/` könyvtárban találhatók gzip állományba csomagolva.

2.3. Támogatott Platformok

- **Linux:** x64, ARM64 architektúrák
- **Windows:** 10/11 (x64, ARM64), Önálló egyetlen EXE fájl vagy EXE és DLL-ek
- **macOS:** Intel és Apple Silicon processzorok, (x64, ARM64)

2.4. Az Alkalmazás Indítása

2.4.1. Forráskódból történő indítás

Az alkalmazás forráskódból a következő módon indítható el:

1. Terminálablak megnyitása
2. Navigáció a projekt könyvtárába
3. Parancs futtatása: `dotnet run -project Pixelium.UI`

2.4.2. Előre buildelt binárisok indítása

A kész, buildelt binárisok esetében az alkalmazás elindul a forráskód vagy telepítés nélkül.

3. fejezet

Felhasználói Útmutató

3.1. Kezdő lépések

3.1.1. Alkalmazás indítása

A program indítása után egy beépített mintakép jelenik meg, ahol lehet tesztelni a szűrőket. A későbbiekben másik beépített minta kép is választható, vagy a tallázással megnyitható egy meglévő kép.

3.1.2. Kép megnyitása

A kép megnyitásához több lehetőség is rendelkezésre áll:

- A **Fájl > Megnyitás** menüpont kiválasztása
- A **Ctrl+O** billentyűkombináció használata
- A támogatott képformátumok közül választás (PNG, JPEG, BMP)

3.1.3. Mintaképek betöltése

Az alkalmazás beépített mintaképeket tartalmaz, amelyek megkönnyítik a különböző funkciók tesztelését:

- **Minta 1: Gradiens és szöveg** - Általános szűrőtesztelésre
- **Minta 2: Geometriai formák** - Él- és sarokdetektálás vizsgálatára
- **Minta 3: Színteszt mintázat** - Hisztogram és gamma korrekció tesztelésére

3.2. A Felület Ismertetése

Az alkalmazás felülete több fő területre oszlik:

- **Menüsor:** Az alapvető műveletek és beállítások elérésére szolgál
- **Rétegek panel:** A rétegkezelés és tulajdonságok kezelésére szolgál
- **Munkaterület:** A képek szerkesztésének és megjelenítésének fő területe

3.3. Képszerkesztés és Szűrők Alkalmazása

3.3.1. Alapvető Szerkesztések

- **Szürkeárnyalatos** (Ctrl+G): A kép szürkeárnyalatosra konvertálása
- **Invertálás** (Ctrl+I): A kép színeinek invertálása
- **Tükrözés** (Ctrl+H): A kép vízszintes tükrözése
- **Gamma korrekció**: A kép fényerejének beállítása
- **Logaritmikus transzformáció**: Dinamikatartomány tömörítése

3.3.2. Speciális Szűrők

Elmosási szűrők

Az elmosási szűrők a kép simítására szolgálnak:

- **Box Filter**: Gyors átlagolás alapú elmosás
- **Gaussian Blur**: Sima elmosás testreszabható kernel mérettel (3x3, 5x5, 7x7)

Élérzékelés

Az élérzékelő szűrők a kép éleinek detektálására szolgálnak:

- **Sobel éldetektor**: Sobel operátor alapú élérzékelés
- **Laplace éldetektor**: Laplace kernel alapú élérzékelés

Harris Sarokérzékelés

A Harris sarokdetektálás használata a következő lépésekkel áll:

1. A **Szűrők > Harris Sarokérzékelés** menüpont kiválasztása
2. A paraméterek beállítása:
 - **Érzékenység (k)**: A sarokválasz szabályozása
 - **Küszöbérték**: A minimális sarokerősség (0-255 tartományban)
 - **Gauss szigma**: Az előfeldolgozó elmosás erőssége
3. Az **Alkalmaz** gomb megnyomása a sarkok észleléséhez

3.4. Hisztogram Analízis

3.4.1. Hisztogram megjelenítése

A hisztogram megjelenítése a következőképpen történik:

- A **Hisztogram mutatása** menüpont kiválasztása
- A luminancia hisztogramjának megjelenítése

3.4.2. Hisztogram kiegyenlítés

A hisztogram kiegyenlítés végrehajtása:

1. A **Hisztogram kiegyenlítés** menüpont kiválasztása
2. A művelet automatikusan javítja a kép kontrasztját
3. Az eredmény ellenőrizhető a hisztogram ablak újbóli megnyitásával

3.5. Rétegek Kezelése

3.5.1. Rétegek létrehozása és kezelése

A rétegek kezelése a következő műveleteket tartalmazza:

- **Új réteg:** Filter alkalmazásával egy új réteg jön létre a rétegek panelen
- **Destruktív szerkesztési mód:** Filter alkalmazásakor nem jön létre új réteg, hanem a meglévőben megy végbe a szerkesztés
- **Rétegek egyesítése:** A réteg kijelölése után az "Egyesítés" gomb használata
- **Duplikálás:** A réteg kijelölése után a "Duplikálás" gomb használata
- **Elrejtés:** A rétegen a checkboxra kattintás
- **Törlés:** A réteg kijelölése után a "Törlés" gomb használata

3.5.2. Réteg tulajdonságok

Az egyes rétegek tulajdonságainak kezelése:

- **Láthatóság:** A checkboxra kattintással kapcsolható a réteg mutatása vagy elrejtése

3.5.3. Rétegek egyesítése

A rétegek kombinálásának lehetőségei:

- **Rétegek egyesítése:** A kijelölt rétegek kombinálása
- **Összevonás:** minden látható réteg egyetlen rétegbe fűzése

3.6. Visszavonás és Ismétlés (Undo/Redo) műveletek

A műveletek visszavonásának és ismétlésének lehetőségei:

- **Visszavonás:** Ctrl+Z (maximum 50 műveletig)
- **Ismétlés:** Ctrl+Y
- **Támogatott műveletek:** minden szűrő és rétegművelet visszavonható

3.7. Billentyűparancsok

Billentyűkombináció	Művelet
Ctrl+N	Új projekt létrehozása
Ctrl+O	Kép megnyitása
Ctrl+S	Kép mentése
Ctrl+Shift+S	Mentés másként
Ctrl+Z	Visszavonás
Ctrl+Y	Ismétlés
Ctrl+G	Szürkeárnyalatos konverzió
Ctrl+I	Színek invertálása
Ctrl+H	Vízszintes tükrözés
Ctrl+F	Illesztés a képernyőhöz
Ctrl++	Nagyítás
Ctrl+-	Kicsinyítés
Ctrl+0	100%-os nagyítás

3.1. táblázat. Billentyűparancsok összefoglaló táblázata

3.8. Képek Mentése

3.8.1. Támogatott formátumok

Az alkalmazás a következő kéatformátumok mentését támogatja:

- **PNG** (Portable Network Graphics) - Veszeségmentes formátum
- **JPEG** (Joint Photographic Experts Group) - Tömörített formátum

3.8.2. Mentési lehetőségek

A mentés kétféleképpen történhet:

- **Mentés** (Ctrl+S): Az aktuális fájl felülírása
- **Mentés másként** (Ctrl+Shift+S): Új fájl létrehozása

4. fejezet

Fejlesztői Dokumentáció

4.1. Architektúra és Fő Komponensek

A Pixelium moduláris architektúrája két fő komponensből áll:

4.1.1. Pixelium.Core

A képfeldolgozási könyvtár, amely független a felhasználói felülettől:

- Képfeldolgozási algoritmusok implementációja
- Rétegkezelés logikája
- Hisztogram számítások

4.1.2. Pixelium.UI

Avalonia UI alapú asztali alkalmazás:

- Felhasználói felület komponensek
- MVVM minta implementációja
- Parancskezelés és visszavonás/ismétlés rendszer

4.2. Kulcstechnológiák

Az alkalmazás a következő technológiákra épül:

- **.NET 9.0:** Modern, nagy teljesítményű alkalmazás keretrendszer
- **Avalonia UI:** Keretrendszer-től független UI keretrendszer
- **SkiaSharp:** 2D grafikai könyvtár gyors rendereléshez
- **MVVM minta:** Tiszta választási rétegek megvalósítása
- **Parancs minta:** Visszavonás/ismétlés funkcionálitás implementációja

4.3. Teljesítményoptimalizálások

A Pixelium több teljesítményoptimalizálási technikát alkalmaz a gyors képfeldolgozás érdekében.

4.3.1. Lookup Table (LUT) Gyorsítótárazás

A lookup table-ök (keresési táblázatok) előre kiszámított transzformációs értékeket tárolnak, ami jelentősen felgyorsítja a pixelenkénti műveleteket.

LUT szolgáltatás implementációja

A `LookupTableService` osztály a következő funkciókat biztosítja:

- **Thread-safe cache:** `ConcurrentDictionary<string, Lazy<byte[]>` alapú gyorsítótár
- **Lazy inicializálás:** A LUT-ok csak az első használatkor kerülnek kiszámításra
- **Előre betöltött táblázatok:** A gyakran használt táblázatok (invert, grayscale, gamma) már inicializáláskor létrejönnek
- **256 elemű táblázatok:** minden lehetséges bemeneti érték (0-255) számára előre kiszámított kimenet

LUT alapú szűrők

A következő szűrők használnak LUT-ot a gyors végrehajtáshoz:

- **GrayscaleProcessor:** Három külön LUT-ot használ ($R \times 0.299$, $G \times 0.587$, $B \times 0.114$)
- **InvertProcessor:** Egyszerű 255-érték LUT
- **GammaProcessor:** Gamma érték alapján előre kiszámított táblázat
- **LogarithmicProcessor:** Logaritmikus transzformáció LUT-ja

4.3.2. Párhuzamos Feldolgozás (Multithreading)

A képfeldolgozó algoritmusok intenzíven használják a párhuzamos feldolgozást:

Sor-alapú párhuzamosítás

A képfeldolgozás soronként történik párhuzamosan a `Parallel.For` használatával:

```
1 Parallel.For(0, bitmap.Height, y =>
2 {
3     byte* rowPtr = ptr + (y * bitmap.Width * 4);
4     for (int x = 0; x < bitmap.Width * 4; x += 4)
5     {
6         // Pixelenkenti feldolgozas
7     }
8});
```

Thread-safe adatstruktúrák

- **ConcurrentDictionary:** A LUT gyorsítótár thread-safe tárolására
- **Lazy<T>:** Biztosítja, hogy a LUT-ok inicializálása csak egyszer történjen meg párhuzamos hozzáférés esetén is
- **Független sorok:** minden képsor függetlenül dolgozható fel, nincs szükség szinkronizációra

4.3.3. Unsafe Kód és Pointer Műveletek

A kritikus teljesítményű részeken közvetlen memória-hozzáférés történik unsafe pointer műveletekkel:

Előnyök

- **Gyorsabb hozzáférés:** Közvetlen memória manipuláció tömb indexelés helyett
- **Kevesebb bounds checking:** Nincs automatikus határellenőrzés
- **Cache-barát:** Lineáris memória hozzáférési minta

Implementáció

```
1 var pixels = bitmap.GetPixels();
2 var ptr = (byte*)pixels.ToPointer();
3
4 // Közvetlen memoria hozzaferes
5 byte blue = ptr[offset + 0];
6 byte green = ptr[offset + 1];
7 byte red = ptr[offset + 2];
8 byte alpha = ptr[offset + 3];
```

4.4. Képmegjelenítés és Formátumok

4.4.1. Belső Formátum

Minden belső képfeldolgozási művelet a `SKColorType.Bgra8888` formátumot használja:

BGRA8888 formátum jellemzői

- **4 byte per pixel:** Kék, Zöld, Piros, Alfa csatornák
- **Csatorna sorrend:** B-G-R-A (nem R-G-B-A!)
- **8 bit per csatorna:** minden csatorna 0-255 értéktartományban
- **Premultiplied alpha:** az alfa csatorna előre megszorozva

Memória elrendezés

```
1 Pixel offset: [0] [1] [2] [3] [4] [5] [6] [7] ...
2 Csatorna:      B   G   R   A   B   G   R   A   ...
3           | -- Pixel 1 -- | | -- Pixel 2 -- |
```

4.4.2. Be- és Kimeneti Formátumok

Bemeneti formátumok (Megnyitás)

Az alkalmazás a következő formátumokat képes beolvasni:

- **PNG (Portable Network Graphics)**
 - Veszeségmentes tömörítés
 - Támogatja az alfa csatornát (átlátszóság)
 - minden képtípushoz ajánlott
- **JPEG (Joint Photographic Experts Group)**
 - Veszeséges tömörítés
 - Nincs alfa csatorna támogatás
 - Fotók tárolására optimalizált
- **BMP (Bitmap)**
 - Tömörítetlen vagy RLE tömörített
 - Nagy fájlméret
 - Gyors betöltés

Kimeneti formátumok (Mentés)

Az alkalmazás a következő formátumokba képes menteni:

- **PNG**

- Veszeségmentes mentés
- Alfa csatorna megőrzése
- Minőségi paraméter: 100 (maximális tömörítési szint)

- **JPEG**

- Veszeséges mentés
- Kisebb fájlméret
- Minőségi paraméter: 100 (maximális minőség)
- Az átlátszó területek fehér háttérrel kerülnek mentésre

Formátum konverzió

Minden betöltött kép automatikusan BGRA8888 formátumra konvertálódik:

```
1 using var originalBitmap = SKBitmap.Decode(stream);
2 var bitmap = new SKBitmap(
3     originalBitmap.Width,
4     originalBitmap.Height,
5     SKColorType.Bgra8888,
6     SKAlphaType.Premul
7 );
8 using var canvas = new SKCanvas(bitmap);
9 canvas.DrawBitmap(originalBitmap, 0, 0);
```

4.5. Képfeldolgozási Algoritmusok

4.5.1. Pixelenkénti Transzformációk

Szürkeárnyalatos Konverzió (Grayscale)

A színes kép szürkeárnyalatos képpé alakítása a luminancia számításával:

Képlet:

$$Gray = 0.299 \times R + 0.587 \times G + 0.114 \times B \quad (4.1)$$

LUT implementáció:

- Három külön lookup table (R, G, B csatornákra)
- Előre kiszámított értékek: `redLut[i] = (byte)(i * 0.299)`
- minden pixel: `gray = redLut[r] + greenLut[g] + blueLut[b]`

Invertálás (Invert)

A kép színeinek invertálása minden csatornán:

Képlet:

$$Output = 255 - Input \quad (4.2)$$

LUT implementáció:

```
1 invertLut[i] = (byte)(255 - i); // i: 0-255
```

Gamma Korrekció

A kép fényerejének nem-lineáris beállítása:

Képlet:

$$Output = 255 \times \left(\frac{Input}{255} \right)^{1/\gamma} \quad (4.3)$$

Paraméterek:

- $\gamma < 1.0$: Világosítás
- $\gamma = 1.0$: Nincs változás
- $\gamma > 1.0$: Sötétítés

Logaritmikus Transzformáció

Dinamikatartomány tömörítése logaritmikus függvényel:

Képlet:

$$Output = c \times \log(1 + Input) \quad (4.4)$$

ahol c egy konstans a normalizáláshoz.

4.5.2. Geometriai Transzformációk

Tükrözés (Flip)

A kép tükrözése vízszintesen vagy függőlegesen:

Vízszintes tükrözés:

- minden sor pixeleinek megfordítása
- Idő komplexitás: $O(width \times height)$
- Párhuzamosan futtatható soronként

Függőleges tükrözés:

- A sorok sorrendjének megfordítása
- Szintén párhuzamosítható

4.5.3. Konvolúciós Szűrők

Konvolúció Általános Képlete

A konvolúciós szűrők egy kernelt (maszkot) alkalmaznak a képre:

$$Output(x, y) = \sum_{i=-k}^k \sum_{j=-k}^k Input(x + i, y + j) \times Kernel(i, j) \quad (4.5)$$

Box Filter (Átlagoló Elmosás)

Egyszerű átlagoló szűrő, minden kernelelem értéke azonos:

3×3 kernel:

$$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad (4.6)$$

Gauss Elmosás (Gaussian Blur)

Gauss-eloszlás alapú elmosás, simább eredményt ad:

Gauss függvény:

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \quad (4.7)$$

5×5 kernel példa ($\sigma = 1.0$):

$$\frac{1}{273} \begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix} \quad (4.8)$$

Sobel Élérzékelés

A Sobel operátor két irányt (vízszintes és függőleges) vizsgál:

Vízszintes kernel (G_x):

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad (4.9)$$

Függőleges kernel (G_y):

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \quad (4.10)$$

Eredmény számítás:

$$G = \sqrt{G_x^2 + G_y^2} \quad (4.11)$$

Laplace Élérzékelés

A Laplace operátor második deriváltakat használ:

3×3 kernel:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad (4.12)$$

4.5.4. Harris Sarokérzékelés

A Harris sarokdetektor algoritmus a következő lépésekkel áll:

Algoritmus lépései

1. **Gradiens számítás:** Sobel operátorral I_x és I_y számítása
2. **Gradiens négyzetek:** I_x^2 , I_y^2 , és $I_x \times I_y$ számítása
3. **Gauss simítás:** A négyzetek simítása Gauss szűrővel
4. **Sarokválasz számítás:**

$$R = \det(M) - k \times \text{trace}(M)^2 \quad (4.13)$$

ahol M a struktúra mátrix, k tipikusan 0.04-0.06

5. **Küszöbölés:** Csak a $R > \text{threshold}$ pontok kerülnek megjelölésre
6. **Non-maximum suppression:** Helyi maximumok kiválasztása

Paraméterek

- **Threshold:** Minimális sarokerősség (0-255)
- **k:** Érzékenységi paraméter (0.04-0.06 ajánlott)
- **Sigma:** Gauss simítás erőssége

4.5.5. Hisztogram Alapú Műveletek

Hisztogram Számítás

A hisztogram egy 256 elemű tömb, amely az egyes intenzitásértékek gyakoriságát tárolja:

```
1 var histogram = new int[256];
2 for (int i = 0; i < pixelCount; i++)
3 {
4     byte intensity = pixels[i];
5     histogram[intensity]++;
6 }
```

Luminancia Hisztogram

RGB képeknél a luminancia számítása:

$$L = 0.299 \times R + 0.587 \times G + 0.114 \times B \quad (4.14)$$

Hisztogram Kiegyenlítés

A hisztogram kiegyenlítés növeli a kép kontrasztját:

Algoritmus:

1. Hisztogram számítása
2. Kumulatív eloszlásfüggvény (CDF) számítása:

$$CDF[i] = \sum_{j=0}^i histogram[j] \quad (4.15)$$

3. Normalizálás és LUT létrehozása:

$$Output[i] = \frac{CDF[i] - CDF_{min}}{TotalPixels - CDF_{min}} \times 255 \quad (4.16)$$

4. LUT alkalmazása minden pixelre

4.6. Parancs Minta és Undo/Redo Funkció

A Pixelium a Command Pattern-t alkalmazza a visszavonás/ismétlés funkcióhoz.

4.6.1. IImageCommand Interfész

Minden visszavonható művelet implementálja az IImageCommand interfést:

```
1 public interface IImageCommand
2 {
3     void Execute();
4     void Undo();
5 }
```

4.6.2. CommandHistory

A CommandHistory osztály kezeli a parancsok előzményeit:

- **Maximum 50 parancs**: A legrégebbi parancsok automatikusan törlődnek
- **Stack alapú**: Végrehajtott és visszavont parancsok külön stacken
- **Snapshot**: minden parancs elmenti a művelet előtti állapotot

4.6.3. FilterCommand

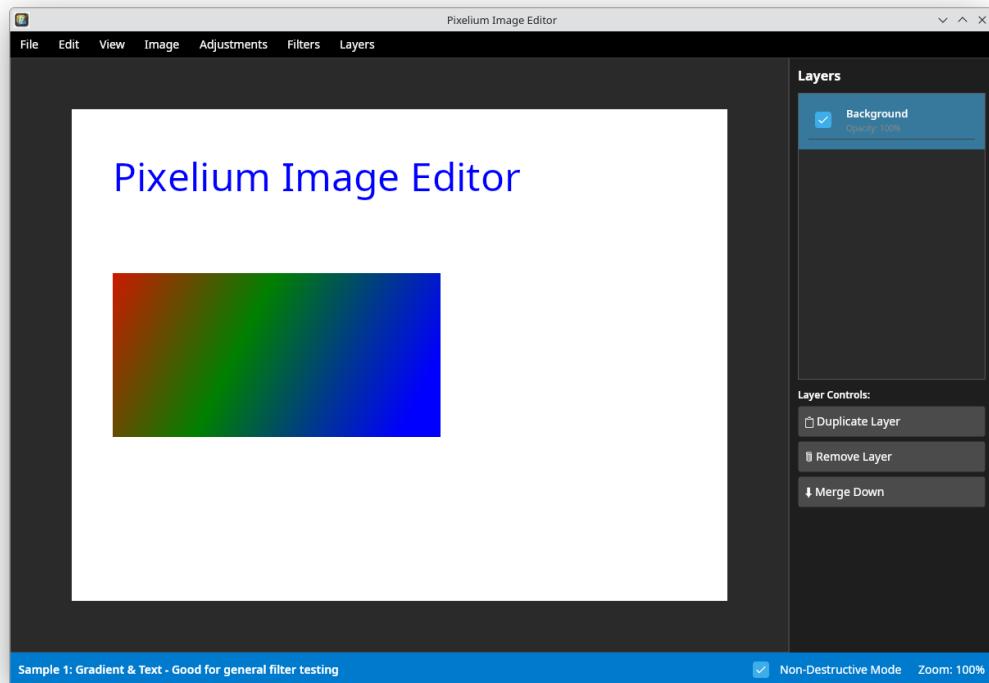
A szűrők alkalmazása előtt másolat készül a képről:

```
1 public void Execute()
2 {
3     _snapshot = _layer.Content.Copy();
4     _processor.Process(_layer.Content);
5 }
6
7 public void Undo()
8 {
9     var temp = _layer.Content;
10    _layer.Content = _snapshot;
11    _snapshot = temp;
12 }
```

5. fejezet

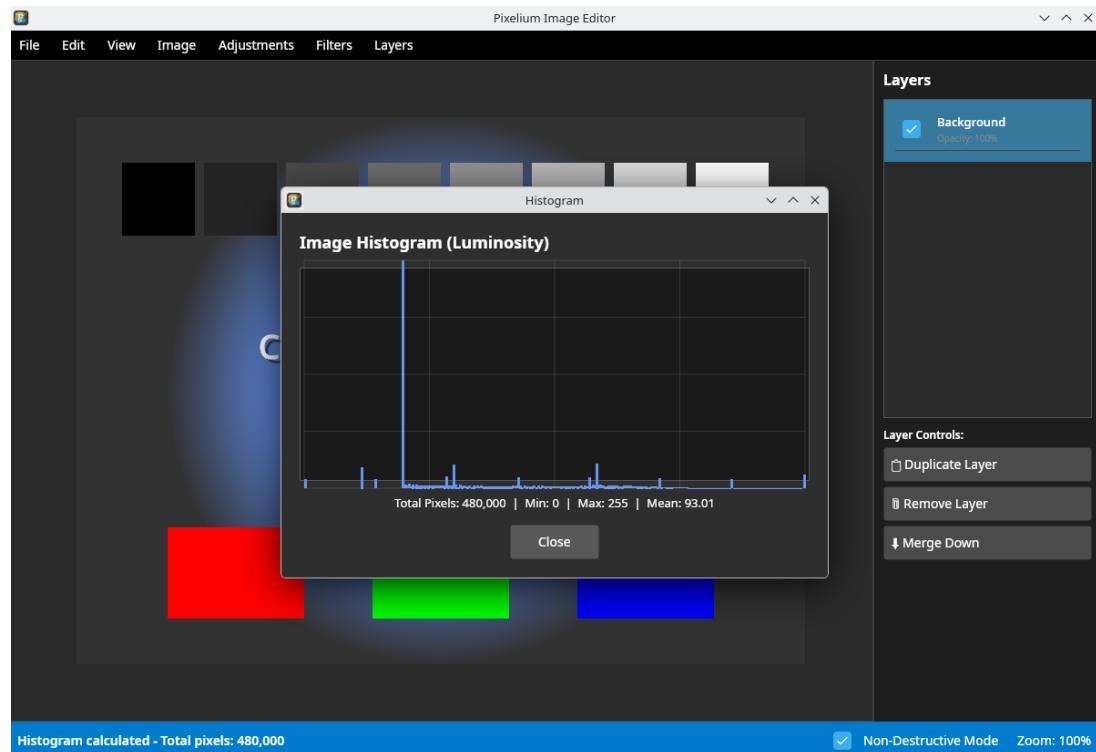
Képernyőképek

5.1. Alkalmazás Felület



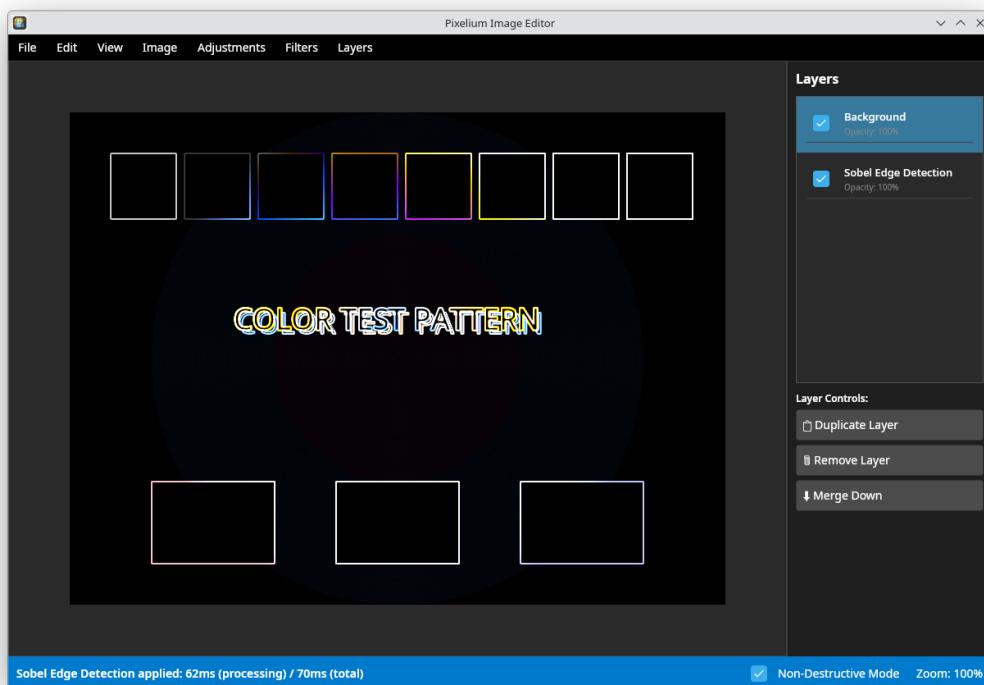
5.1. ábra. Pixelium főképernyő - Alapértelmezett nézetben a mintaképpel és a rétegkezelő panellel

5.2. Hisztogram Analízis



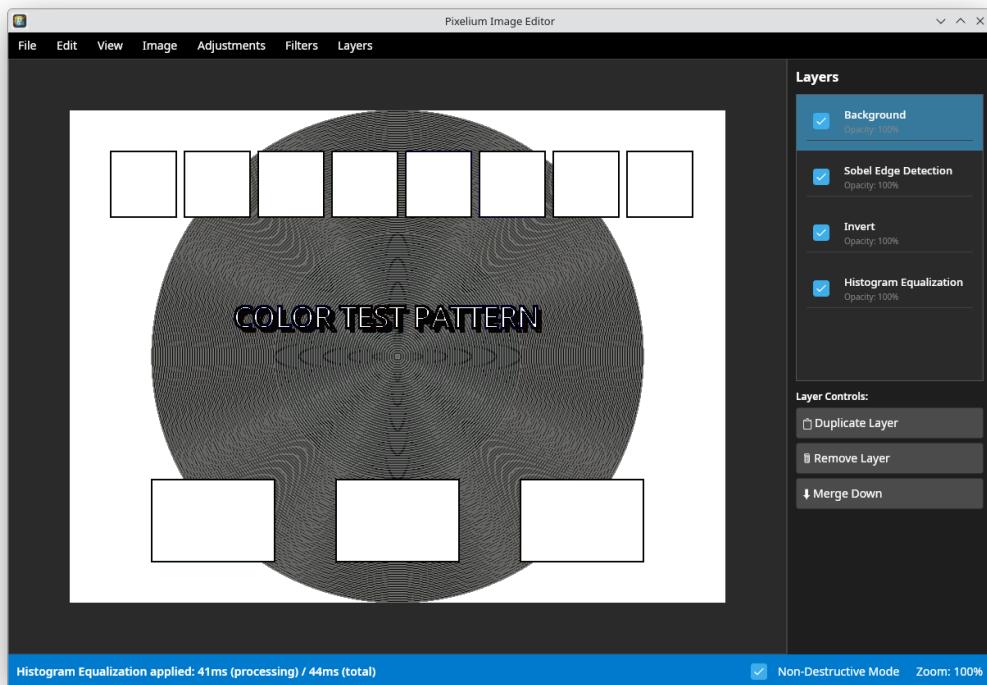
5.2. ábra. Hisztogram megjelenítés - A luminancia eloszlás grafikus ábrázolása

5.3. Élérzékelés



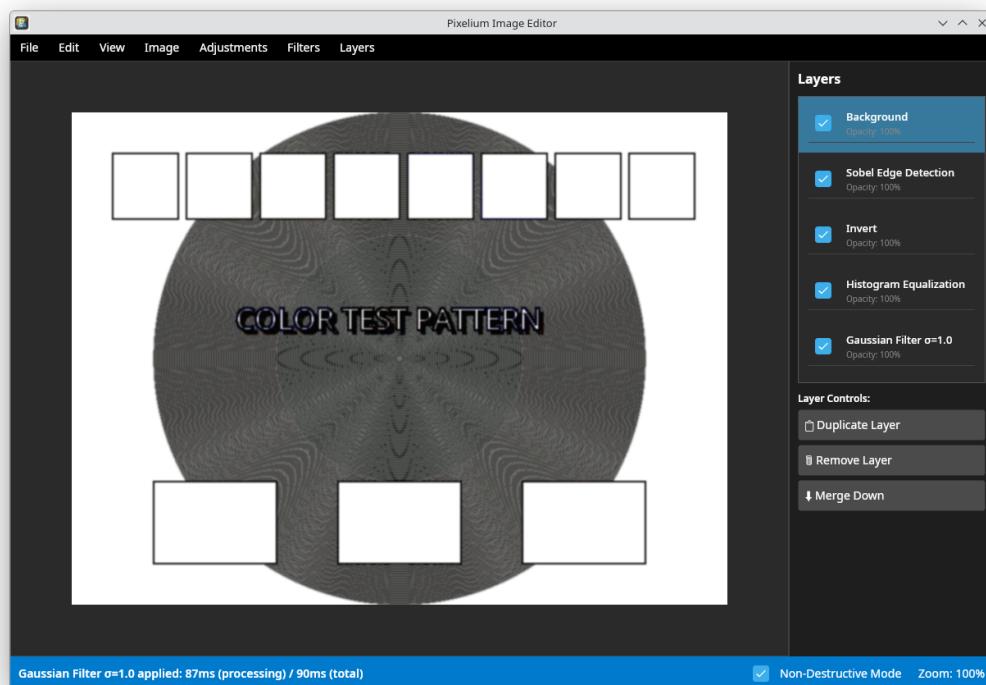
5.3. ábra. Sobel élérzékelés alkalmazása - Az élek kiemelése a képen

5.4. Réteges Szerkesztés



5.4. ábra. Több réteg kezelése - Rétegek panel különböző szűrőkkel

5.5. Elmosási Szűrők



5.5. ábra. Gauss elmosás alkalmazása - A kép simítása Gauss szűrővel