Sprawozdanie z przedmiotu:
Grafika komputerowa i wizualizacja

Przekształcenia w obrazach graficznych

Maciej Kurek gr.1

Nr indeksu: 134995

Informatyka

WIMII

Filtracja obrazu to proces przetwarzania obrazu cyfrowego w celu zmniejszenia szumu lub wyostrzenia obrazu. Polega na zastosowaniu specjalnych narzędzi, nazywanych filtrami, które modyfikują piksele obrazu, aby uzyskać pożądany efekt.jest stosowana w wielu dziedzinach, takich jak przetwarzanie obrazów medycznych, analiza obrazów satelitarnych, grafika komputerowa, przetwarzanie obrazów przemysłowych i wiele innych. Filtracja obrazu może być realizowana za pomocą różnych rodzajów filtrów, takich jak filtr dolnoprzepustowy, filtr górnoprzepustowy, filtr medianowy, filtr Laplace'a, filtr Sobela i wiele innych. Każdy z tych filtrów ma swoje własne zastosowanie i wpływ na obraz, a wybór odpowiedniego filtru zależy od konkretnej sytuacji i pożądanego efektu.

Opis kilku wybranych filtrów:

Filtr Laplace’a: Jest to rodzaj filtru stosowanego w przetwarzaniu obrazów cyfrowych do wykrywania krawędzi i zmian w intensywności pikseli. Jest to filtr wykorzystujący operator Laplace'a, który jest drugą pochodną cząstkową względem x i y, co oznacza, że wykrywa on szybkie zmiany w intensywności pikseli na obrazie. Filtr Laplace'a działa przez przepuszczenie obrazu przez macierz filtru Laplace'a. Ta macierz składa się z elementów ułożonych w kształcie krzyża lub kwadratu, z wartością ujemną w środku i wartościami dodatnimi na bokach. Filtr Laplace'a ma wiele zastosowań w przetwarzaniu obrazów, w tym w wykrywaniu krawędzi, wyodrębnianiu szczegółów, wyostrzeniu obrazów i w redukcji szumu. Jednakże, ze względu na jego wysoką czułość na szum, może prowadzić do wykrywania fałszywych krawędzi lub szumów jako krawędzi, co może wymagać zastosowania innych technik filtracji w celu usunięcia niepożądanych artefaktów.

Filtr dolnoprzepustowy: Jest to rodzaj filtru stosowanego w przetwarzaniu obrazów cyfrowych, który przepuszcza tylko niskie częstotliwości i tłumi wysokie częstotliwości. Częstotliwości te odnoszą się do szybkości zmian wartości pikseli w obrazie. Im szybsza zmiana, tym wyższa częstotliwość. Filtr dolnoprzepustowy działa przez wygładzenie obrazu, usuwając szybko zmieniające się detale, które mogą stanowić szum lub fałszywe krawędzie. Dzięki temu, obraz staje się mniej skomplikowany i bardziej ujednolicony.

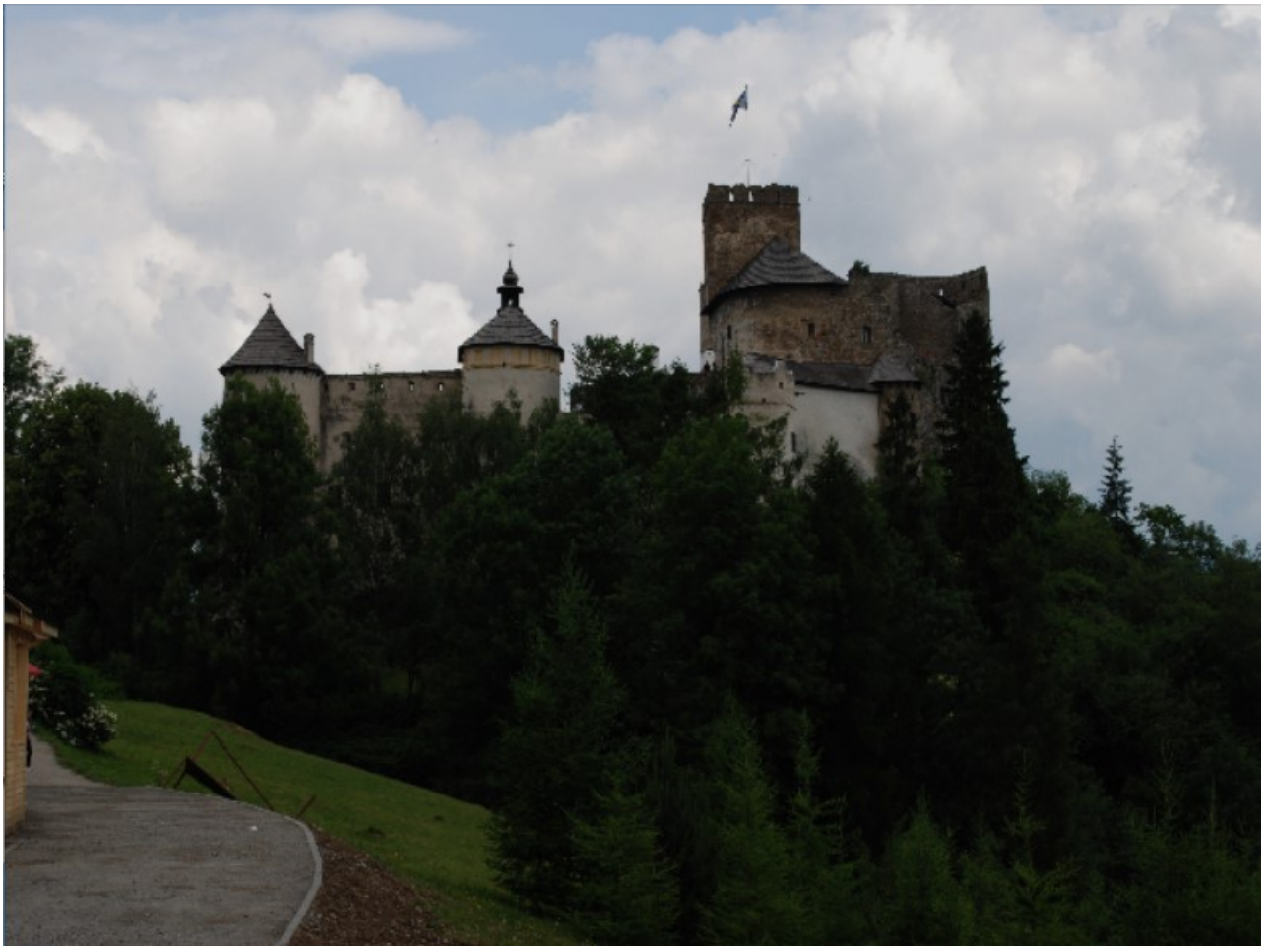
Filtr dolnoprzepustowy działa poprzez przepuszczenie obrazu przez macierz filtru, która jest ustawiona tak, aby przyciemnić piksele na krawędziach i zachować piksele w obszarach o stałym lub powolnym wzroście intensywności. W wyniku tego, obszary o wysokim kontraście i krawędzie są łagodzone lub stają się mniej widoczne.
Filtr dolnoprzepustowy znajduje zastosowanie w wielu dziedzinach, takich jak przetwarzanie obrazów medycznych, grafika komputerowa, analiza obrazów satelitarnych i wiele innych. Jest to również jeden z podstawowych filtrów stosowanych w cyfrowej obróbce obrazów i jest często stosowany wraz z innymi filtrami w celu osiągnięcia pożądanego efektu przetwarzania obrazów.

Filtr Sobel'a: Jest to rodzaj filtru stosowanego w przetwarzaniu obrazów cyfrowych do wykrywania krawędzi. Jest to filtr wykorzystujący dwa oddzielne operatory gradientowe, jeden dla krawędzi wertykalnych (poziomych) i drugi dla krawędzi horyzontalnych.
Filtr Sobel'a działa przez przepuszczenie obrazu przez dwa oddzielne operatory gradientowe, a następnie obliczenie magnitudy gradientu obrazu, czyli pierwiastka kwadratowego sumy kwadratów gradientów poziomych i pionowych. Te dwie operacje pozwalają na wykrycie krawędzi na obrazie, zarówno poziomych, jak i pionowych.
Filtr Sobel'a wykrywa krawędzie przez przetwarzanie obrazu piksel po pikselu i obliczanie gradientu jasności w każdym punkcie. Gradient jest wyznaczany na podstawie różnic w intensywności pikseli wokół danego punktu. Dzięki temu filtr Sobel'a może wykrywać krawędzie o różnych kierunkach i kształtach.
Jest on często stosowany w przetwarzaniu obrazów cyfrowych, zwłaszcza w celu wykrywania krawędzi, wyostrzania obrazów, rozmycia i w redukcji szumu. Jest również stosowany w analizie obrazów medycznych, w odkrywaniu wzorców w danych obrazowych, a także w grafice komputerowej.

Filtr medianowy: Jest to rodzaj filtru stosowanego w przetwarzaniu obrazów cyfrowych, który służy do redukcji szumu. Filtr ten działa poprzez zastąpienie wartości pikseli w obrazie medianą wartości pikseli w otoczeniu danego piksela.
W przeciwieństwie do filtrów dolnoprzepustowych, które obliczają średnią wartość pikseli w otoczeniu, filtr medianowy wykorzystuje medianę - wartość środkową z posortowanej listy wartości pikseli w otoczeniu. Dzięki temu filtr medianowy jest skuteczniejszy w usuwaniu szumu impulsowego, czyli pojedynczych, nietypowych pikseli, które zniekształcają obraz.
Filtr medianowy jest często stosowany w cyfrowej obróbce obrazów, zwłaszcza w przypadku obrazów, w których pojawiają się losowe szumy. Jest to popularny

filtr w zastosowaniach medycznych, gdyż pozwala na usunięcie zakłóceń z obrazów medycznych, takich jak tomografie komputerowe, czy obrazy MRI, a jednocześnie zachowuje ważne informacje diagnostyczne.
Wadą filtru medianowego jest jego złożoność obliczeniowa, która może być większa niż w przypadku innych filtrów redukujących szum. W przypadku dużych obrazów lub złożonych operacji przetwarzania obrazów, filtr medianowy może być zbyt wolny do stosowania w czasie rzeczywistym.

Oryginalne zdjęcie używane do przekształceń podczas zajęć laboratoryjnych:



Kod w C++:

```
/*
 * GLUT Shapes Demo
 *
 * Written by Nigel Stewart November 2003
 *
 * This program is test harness for the sphere, cone
 * and torus shapes in GLUT.
 *
 * Spinning wireframe and smooth shaded shapes are
 * displayed until the ESC or q key is pressed.  The
 * number of geometry stacks and slices can be adjusted
 * using the + and - keys.
 */

#ifdef __APPLE__
#include <GLUT/glut.h>
#else
#include <GL/glut.h>
#include <iostream>
#include <fstream>
#endif

#include <stdlib.h>



static int slices = 16;
static int stacks = 16;

    const int w=1339;
    const int k=2000;


    int Rs[w][k];
    int Gs[w][k];
    int Bs[w][k];
    int Rn[w][k];
```

```
    int Gn[w][k];
    int Bn[w][k];
    int m1[3][3] = {1,1,1,1,1,1,1,1,1};

    float lw,lk;

/* GLUT callback Handlers */
using namespace std;
static void resize(int width, int height)
{
    const float ar = (float) width / (float) height;

    glViewport(0, 0, width, height);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
//    glFrustum(-ar, ar, -1.0, 1.0, 2.0, 100.0);
    glOrtho(0,k,0,w,2.0,100.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity() ;
}

static void display(void)
{
    const double t = glutGet(GLUT_ELAPSED_TIME) / 1000.0;
    const double a = t*90.0;

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glColor3d(1,0,0);


    glPushMatrix();
       glTranslated(0,0,-6);
          glBegin(GL_POINTS);
              for(int i=0;i<lw;++i)
                for(int j=0;j<lk;++j)
                {
                     glColor3f(Rs[i][j]/255.,Gs[i][j]/255.,Bs[i][j]/255.);
                   glVertex3d(j,i,0);
                }


          glEnd();
```

```
    glPopMatrix();



    glutSwapBuffers();
}


static void key(unsigned char key, int x, int y)
{
    switch (key)
    {
        case 27 :
        case 'q':
            ;
            break;

         case 'r':

        for(int i=0;i<lw;++i)
            for(int j=0;j<lk;++j)
            {
                Rs[i][j]=Rn[i][j];
                Gs[i][j]=Gn[i][j];
                Bs[i][j]=Bn[i][j];
            }
        for(int i=0;i<lw;++i)
            for(int j=0;j<lk;++j)
            {
                Gs[i][j]=0;
                Bs[i][j]=0;

            }
            break;

         case 'g':
        for(int i=0;i<lw;++i)
            for(int j=0;j<lk;++j)
            {
                Rs[i][j]=Rn[i][j];
                Gs[i][j]=Gn[i][j];
                Bs[i][j]=Bn[i][j];
```

```
            }
        for(int i=0;i<lw;++i)
            for(int j=0;j<lk;++j)
            {
                Rs[i][j]=0;
                Bs[i][j]=0;
            }
            break;

         case 'b':
        for(int i=0;i<lw;++i)
            for(int j=0;j<lk;++j)
            {
                Rs[i][j]=Rn[i][j];
                Gs[i][j]=Gn[i][j];
                Bs[i][j]=Bn[i][j];
            }
        for(int i=0;i<lw;++i)
            for(int j=0;j<lk;++j)
            {
                Rs[i][j]=0;
                Gs[i][j]=0;
            }
            break;

case 'j':
 for(int i=0; i<lw; ++i)
 for(int j=0; j<lk; ++j)
 {
 Rn[i][j] += 20;
 Gn[i][j] += 20;
 Bn[i][j] += 20;
 }
 for(int i=0; i<lw; ++i)
 for(int j=0; j<lk; ++j)
 {
 R[i][j] = Rn[i][j];
 G[i][j] = Gn[i][j];
 B[i][j] = Bn[i][j];
 }
 break;
 case 'k':
```

```
for(int i=0; i<lw; ++i)
for(int j=0; j<lk; ++j)
{
Rn[i][j] -= 20;
Gn[i][j] -= 20;
Bn[i][j] -= 20;
}
for(int i=0; i<lw; ++i)
for(int j=0; j<lk; ++j)
{
R[i][j] = Rn[i][j];
G[i][j] = Gn[i][j];
B[i][j] = Bn[i][j];
}
break;

      case 'z':
        for(int i=0;i<lw;++i)
         for(int j=0;j<lk;++j)
         {
            Rs[i][j]=Rn[i][j];
            Gs[i][j]=Gn[i][j];
            Bs[i][j]=Bn[i][j];
         }
         break;

      case 'm':
        for(int i=1;i<lw-1;++i)
         for(int j=1;j<lk-1;++j)
         {
            Rs[i][j]=(Rs[i-1][j-1]*m1[0][0]+Rs[i-1][j]*m1[0][1]+Rs[i-1]
[j+1]*m1[0][2]+Rs[i][j-1]*m1[1][0]+Rs[i][j]*m1[1][1]+Rs[i][j+1]*m1[1]
[2]+Rs[i+1][j-1]*m1[2][0]+Rs[i+1][j]*m1[2][1]+Rs[i+1][j+1]*m1[2][2])/9;
            Bs[i][j]=(Bs[i-1][j-1]*m1[0][0]+Bs[i-1][j]*m1[0][1]+Bs[i-1]
[j+1]*m1[0][2]+Bs[i][j-1]*m1[1][0]+Bs[i][j]*m1[1][1]+Bs[i][j+1]*m1[1]
[2]+Bs[i+1][j-1]*m1[2][0]+Bs[i+1][j]*m1[2][1]+Bs[i+1][j+1]*m1[2][2])/9;
            Gs[i][j]=(Gs[i-1][j-1]*m1[0][0]+Gs[i-1][j]*m1[0][1]+Gs[i-1]
[j+1]*m1[0][2]+Gs[i][j-1]*m1[1][0]+Gs[i][j]*m1[1][1]+Gs[i][j+1]*m1[1]
[2]+Gs[i+1][j-1]*m1[2][0]+Gs[i+1][j]*m1[2][1]+Gs[i+1][j+1]*m1[2][2])/9;
         }
         break;
```

```
case 'p':
{
int M[3][3]={{1,2,1},{0,0,0},{-1,-2,-1}};
for(int i=1;i<lw-1;++i)
for(int j=1;j<lk-1;++j)
{
 Rn[i][j]=(R[i-1][j-1]M[0][0]+R[i-1][j]M[0][1]+R[i-1][j+1]M[0][2]+ R[i ][j-1]M[1][0]+R[i ][j]M[1][1]+R[i ][j+1]M[1][2]+ R[i+1][j-1]M[2][0]+R[i+1][j]M[2][1]+R[i+1][j+1]M[2][2]) /( (M[0][0]+M[0][1]+M[0][2]+M[1][0]+M[1][1]+M[1][2]+M[2][0]+M[2][1]+M[2][2])==0 ? 1 : (M[0][0]+M[0][1]+M[0][2]+M[1][0]+M[1][1]+M[1][2]+M[2][0]+M[2][1]+M[2][2]) ); Gn[i][j]=(G[i-1][j-1]M[0][0]+G[i-1][j]M[0][1]+G[i-1][j+1]M[0][2]+ G[i ][j-1]M[1][0]+G[i ][j]M[1][1]+G[i ][j+1]M[1][2]+ G[i+1][j-1]M[2][0]+G[i+1][j]M[2][1]+G[i+1][j+1]M[2][2]) /( (M[0][0]+M[0][1]+M[0][2]+M[1][0]+M[1][1]+M[1][2]+M[2][0]+M[2][1]+M[2][2])==0 ? 1 : (M[0][0]+M[0][1]+M[0][2]+M[1][0]+M[1][1]+M[1][2]+M[2][0]+M[2][1]+M[2][2]) ); Bn[i][j]=(B[i-1][j-1]M[0][0]+B[i-1][j]M[0][1]+B[i-1][j+1]M[0][2]+ B[i ][j-1]M[1][0]+B[i ][j]M[1][1]+B[i ][j+1]M[1][2]+ B[i+1][j-1]M[2][0]+B[i+1][j]M[2][1]+B[i+1][j+1]*M[2][2]) /( (M[0][0]+M[0][1]+M[0][2]+M[1][0]+M[1][1]+M[1][2]+M[2][0]+M[2][1]+M[2][2])==0 ? 1 : (M[0][0]+M[0][1]+M[0][2]+M[1][0]+M[1][1]+M[1][2]+M[2][0]+M[2][1]+M[2][2]) ); } } for(int i=0;i<lw;++i) for(int j=0;j<lk;++j) { R[i][j]=Rn[i][j]; G[i][j]=Gn[i][j]; B[i][j]=Bn[i][j]; } break;


        case '+':
            slices++;
            stacks++;
            break;

        case '-':
            if (slices>3 && stacks>3)
            {
                slices--;
                stacks--;
            }
            break;
```

```
    }

case 'q':
 for(int i=1;i<lw-1;++i)
 for(int j=1;j<lk-1;++j)
 {
 int Rmax=R[i-1][j-1];
 for(int n=-1;n<2;n++)
 for(int m=-1;m<2;m++)
 {
 if(Rmax<R[i+n][j+m])
 Rmax=R[i+n][j+m];
 }
 Rn[i][j]=Rmax;
 int Gmax=G[i-1][j-1];
 for(int n=-1;n<2;n++)
 for(int m=-1;m<2;m++)
 {
 if(Gmax<G[i+n][j+m])
 Gmax=G[i+n][j+m];
 }
 Gn[i][j]=Gmax;
 int Bmax=B[i-1][j-1];
 for(int n=-1;n<2;n++)
 for(int m=-1;m<2;m++)
 {
 if(Bmax<B[i+n][j+m])
 Bmax=B[i+n][j+m];
 }
 Bn[i][j]=Bmax;
 }
 break;

    glutPostRedisplay();
}

static void idle(void)
{
    glutPostRedisplay();
}

const GLfloat light_ambient[]  = { 0.0f, 0.0f, 0.0f, 1.0f };
```

```
const GLfloat light_diffuse[]  = { 1.0f, 1.0f, 1.0f, 1.0f };
const GLfloat light_specular[] = { 1.0f, 1.0f, 1.0f, 1.0f };
const GLfloat light_position[] = { 2.0f, 5.0f, 5.0f, 0.0f };

const GLfloat mat_ambient[]    = { 0.7f, 0.7f, 0.7f, 1.0f };
const GLfloat mat_diffuse[]    = { 0.8f, 0.8f, 0.8f, 1.0f };
const GLfloat mat_specular[]   = { 1.0f, 1.0f, 1.0f, 1.0f };
const GLfloat high_shininess[] = { 100.0f };

/* Program entry point */

int main(int argc, char *argv[])
{



    ifstream plik("C:/Users/student/Desktop/zd5.txt");  // podaæ miejsce pliku
       plik>>lw>>lk;
       cout<<"wiersze="<<lw<<" kolumny="<<lk<<endl;
       for(int i=0;i<lw;++i)
       {


          for(int j=0;j<lk;++j)
          {
             plik>>Rs[i][j];
             plik>>Gs[i][j];
             plik>>Bs[i][j];
          }

       }
       plik.close();
       for(int i=0;i<lw;++i)
       {
           for(int j=0;j<lk;++j)
          {
             Rn[i][j]=Rs[i][j];
             Gn[i][j]=Gs[i][j];
             Bn[i][j]=Bs[i][j];
          }
       }
```

```
    glutInit(&argc, argv);
    glutInitWindowSize(640,480);
    glutInitWindowPosition(10,10);
    glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);

    glutCreateWindow("GLUT Shapes");

    glutReshapeFunc(resize);
    glutDisplayFunc(display);
    glutKeyboardFunc(key);
    glutIdleFunc(idle);

    glClearColor(1,1,1,1);
    glEnable(GL_CULL_FACE);
    glCullFace(GL_BACK);

    glEnable(GL_DEPTH_TEST);
    glDepthFunc(GL_LESS);

    glEnable(GL_LIGHT0);
    glEnable(GL_NORMALIZE);
    glEnable(GL_COLOR_MATERIAL);
    glEnable(GL_LIGHTING);

    glLightfv(GL_LIGHT0, GL_AMBIENT,  light_ambient);
    glLightfv(GL_LIGHT0, GL_DIFFUSE,  light_diffuse);
    glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);
    glLightfv(GL_LIGHT0, GL_POSITION, light_position);

    glMaterialfv(GL_FRONT, GL_AMBIENT,   mat_ambient);
    glMaterialfv(GL_FRONT, GL_DIFFUSE,   mat_diffuse);
    glMaterialfv(GL_FRONT, GL_SPECULAR,  mat_specular);
    glMaterialfv(GL_FRONT, GL_SHININESS, high_shininess);

    glutMainLoop();

    return EXIT_SUCCESS;
}
```

Barwa zielona

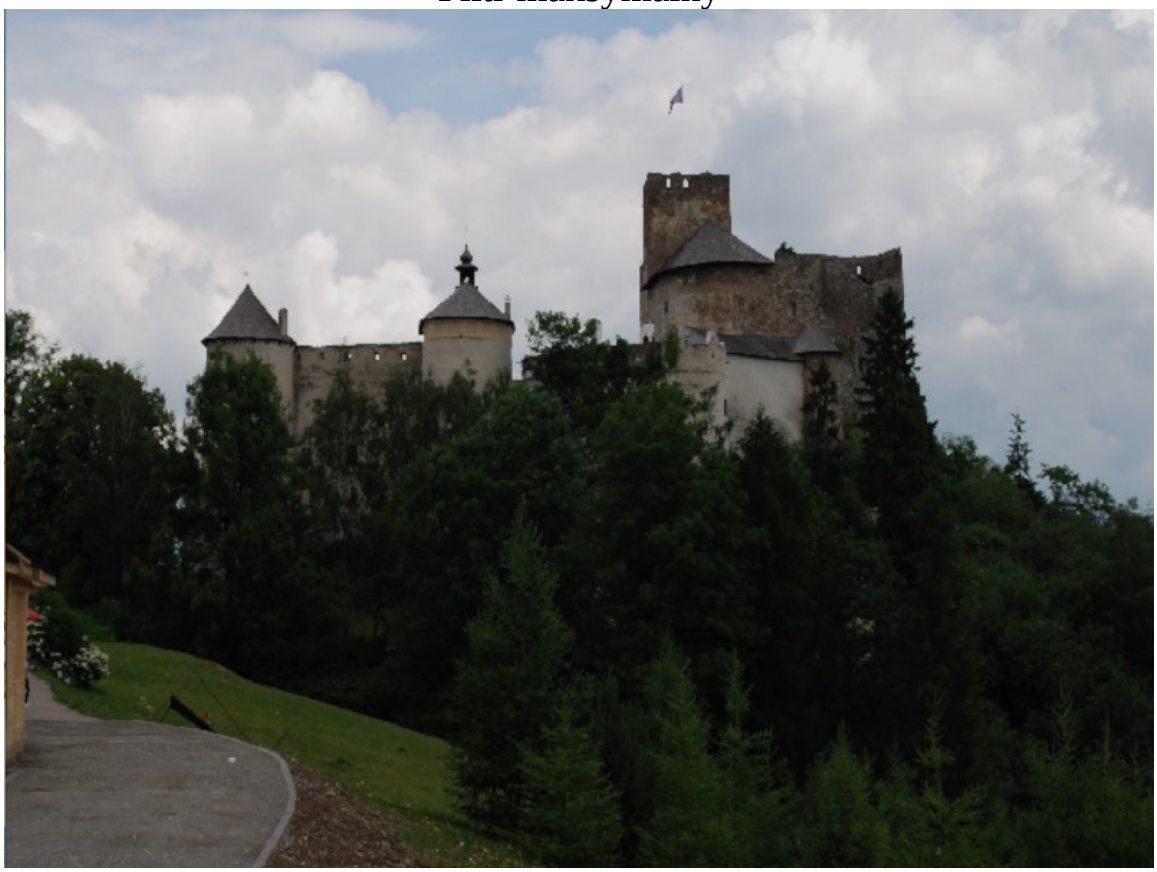
Barwa czerwona

Barwa niebieska

Filtr Sobel’a

Filtr dolnoprzepustowy uśredniający

Filtr maksymalny



Zwiększenie jasności

Zmniejszenie jasności