**App.xaml**

<Application x:Class="BOLD.App"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

DispatcherUnhandledException="Application\_DispatcherUnhandledException"

StartupUri="MainWindow.xaml">

<Application.Resources>

</Application.Resources>

</Application>

**App.xaml.cs**

using System.Windows;

namespace BOLD

{

/// <summary>

/// Interaction logic for App.xaml

/// </summary>

public partial class App : Application

{

private void Application\_DispatcherUnhandledException(object sender, System.Windows.Threading.DispatcherUnhandledExceptionEventArgs e)

{

MessageBox.Show(e.Exception.Message, "Exception", MessageBoxButton.OK, MessageBoxImage.Warning);

e.Handled = true;

}

}

}

**MainWindow.xaml**

<Window

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:Avalon="clr-namespace:Utils.Avalon" xmlns:local="clr-namespace:BOLD" xmlns:System="clr-namespace:System;assembly=mscorlib" x:Name="window" mc:Ignorable="d" x:Class="BOLD.MainWindow"

Title="BOLD" Height="350" Width="452.667" MinWidth="430" MinHeight="350" Icon="Pictures/Logo.png"

>

<Window.Resources>

<Avalon:JScriptConverter x:Key="JScriptConverter"/>

<local:MarginConverter x:Key="MarginConverter"/>

</Window.Resources>

<!--<Window.Resources>

<Avalon:JScriptConverter x:Key="JScriptConverter"/>

</Window.Resources>-->

<Grid>

<Menu VerticalAlignment="Top" >

<MenuItem Header="\_File" >

<MenuItem Header="\_New..." Click="new\_Click"/>

<Separator />

<MenuItem Header="\_Add Image..." Click="add\_Click" />

<MenuItem Header="\_Replace Image..." Click="replace\_Click" />

<Separator />

<MenuItem Header="\_Save As..." Click="save\_As\_Click"/>

<MenuItem Header="E\_xport PNG..." Click="save\_PNG\_Click"/>

<MenuItem Header="E\_xport Statistics..." Click="save\_Statistics\_Click"/>

<Separator />

<MenuItem Header="\_Exit" Click="exit\_Click" />

</MenuItem>

<MenuItem Header="\_Colors">

<MenuItem x:Name="checked\_gray" Header="\_Grayscale" IsCheckable="true" Click="MenuItem\_Grayscale" IsChecked="True" />

<MenuItem x:Name="checked\_rb" Header="\_RedBlue" IsCheckable="true" Click="MenuItem\_RedBlue"/>

<MenuItem x:Name="checked\_rgb" Header="R\_GB" IsCheckable="true" Click="MenuItem\_Click\_RGB" />

</MenuItem>

<MenuItem Header="\_Set">

<MenuItem x:Name="minmax" Header="Set Scale Min/Max" IsCheckable="False" Click="minmax\_Click"/>

</MenuItem>

<MenuItem Header="\_Help">

<MenuItem Header="\_About" Click="about\_Click" />

</MenuItem>

</Menu>

<StackPanel x:Name="canv" Margin="10,57,10,50" Orientation="Horizontal">

<Canvas x:Name="drawBox" Width="{Binding ActualHeight, ElementName=canv}" Height="{Binding ActualHeight, ElementName=canv}"

MouseDown="Grid\_MouseDown" MouseUp="Grid\_MouseUp" MouseMove="Grid\_MouseMove">

<Image x:Name="image" Width="{Binding ActualHeight, ElementName=canv}" Height="{Binding ActualHeight, ElementName=canv}" RenderTransformOrigin="0.5,0.5" >

<Image.RenderTransform>

<TransformGroup>

<ScaleTransform ScaleY="-1" ScaleX="1"/>

<RotateTransform Angle="90"/>

</TransformGroup>

</Image.RenderTransform>

</Image>

<Rectangle x:Name="selectionBox" Visibility="Collapsed" Stroke="Black" StrokeThickness="1">

<Rectangle.Fill>

<SolidColorBrush Color="#7F4040D1" Opacity="0.5"/>

</Rectangle.Fill>

</Rectangle>

</Canvas>

<DockPanel Margin="5,0,0,0" >

<GroupBox Header="Opierations" DockPanel.Dock="Top">

<StackPanel Orientation="Vertical">

<Button x:Name="differenceAB" VerticalAlignment="Top" Width="40" Content="A-B" Margin="0,0,0,0" Click="difference\_ClickAB" IsEnabled="False" />

<Button x:Name="differenceAB\_A" VerticalAlignment="Top" Width="60" Content="(A-B)/A" Margin="0,0,0,0" Click="difference\_ClickABA" IsEnabled="False" />

<Button x:Name="differenceBA" VerticalAlignment="Top" Width="40" Content="B-A" Margin="0,0,0,0" Click="difference\_ClickBA" IsEnabled="False" />

<Button x:Name="sum" VerticalAlignment="Top" Width="40" Content="A+B" Margin="0,0,0,0" Click="sum\_Click" IsEnabled="False" />

<Button x:Name="avg3" VerticalAlignment="Top" Width="60" Content="(A+B+C)/3" Margin="0,0,0,0" Click="avg3\_Click" IsEnabled="False" />

</StackPanel>

</GroupBox>

<DockPanel x:Name="color\_panel" DockPanel.Dock="Top">

<Rectangle x:Name="color\_scale\_gray" Stroke="Black" StrokeThickness="2" Width="40" Visibility="Visible">

<Rectangle.Fill>

<LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">

<GradientStop Color="Black" Offset="1"/>

<GradientStop Color="White"/>

<GradientStop Color="Gray" Offset="{Binding DataContext, ConverterParameter=1-values[0], Converter={StaticResource JScriptConverter}, ElementName=zeroBond}"/>

</LinearGradientBrush>

</Rectangle.Fill>

</Rectangle>

<Rectangle x:Name="color\_scale\_rb" Stroke="Black" StrokeThickness="2" Width="40" Visibility="Collapsed">

<Rectangle.Fill>

<LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">

<GradientStop Color="Blue" Offset="1"/>

<GradientStop Color="Black" Offset="{Binding DataContext, ConverterParameter=1-values[0], Converter={StaticResource JScriptConverter}, ElementName=zeroBond}"/>

<GradientStop Color="Red"/>

</LinearGradientBrush>

</Rectangle.Fill>

</Rectangle>

<Rectangle x:Name="color\_scale\_rgb" Stroke="Black" StrokeThickness="2" Width="40" Visibility="Collapsed">

<Rectangle.Fill>

<LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">

<GradientStop Color="Blue" Offset="1"/>

<GradientStop Color="Cyan" Offset="{Binding DataContext, ConverterParameter=1-values[0]/2, Converter={StaticResource JScriptConverter}, ElementName=zeroBond}"/>

<GradientStop Color="Lime" Offset="{Binding DataContext, ConverterParameter=1-values[0], Converter={StaticResource JScriptConverter}, ElementName=zeroBond}"/>

<GradientStop Color="Yellow" Offset="{Binding DataContext, ConverterParameter=1-(0.5+values[0]/2), Converter={StaticResource JScriptConverter}, ElementName=zeroBond}"/>

<GradientStop Color="Red"/>

</LinearGradientBrush>

</Rectangle.Fill>

</Rectangle>

<Grid x:Name="scale" DockPanel.Dock="Left">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="1\*"/>

<ColumnDefinition Width="1\*"/>

</Grid.ColumnDefinitions>

<Label x:Name="upperBond" Content="" VerticalAlignment="Top" />

<Label x:Name="lowerBond" Content="" VerticalAlignment="Bottom" />

<Label x:Name="zeroBond" Content="" VerticalAlignment="Bottom" Visibility="Collapsed" >

<Label.Margin>

<MultiBinding Converter="{StaticResource MarginConverter}">

<Binding Path="ActualHeight" ElementName="color\_panel"></Binding>

<Binding Path="DataContext" ElementName="zeroBond"></Binding>

</MultiBinding>

</Label.Margin>

<Label.DataContext>

<System:Double>0.5</System:Double>

</Label.DataContext>

</Label>

<Slider x:Name="zeroBondSlider" Orientation="Vertical" Grid.Column="1" Maximum="0.999" TickFrequency="0.001"

Value="{Binding DataContext, ElementName=zeroBond}" Minimum="0.001" Visibility="Visible" ValueChanged="zeroBondSlider\_ValueChanged" LargeChange="0.1" SmallChange="0.01" >

<Slider.DataContext>

<System:Boolean>true</System:Boolean>

</Slider.DataContext>

</Slider>

</Grid>

</DockPanel>

</DockPanel>

<GroupBox Header="Calculations" DockPanel.Dock="Top" VerticalAlignment="Top">

<Grid Height="180" VerticalAlignment="Top" Width="120">

<Grid.RowDefinitions>

<RowDefinition Height="1\*"/>

<RowDefinition Height="1\*"/>

<RowDefinition Height="1\*"/>

<RowDefinition Height="1\*"/>

<RowDefinition Height="1\*"/>

<RowDefinition Height="1\*"/>

<RowDefinition Height="1\*"/>

<RowDefinition Height="1\*"/>

<RowDefinition Height="1\*"/>

<RowDefinition Height="1\*"/>

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="37\*"/>

<ColumnDefinition Width="3\*"/>

<ColumnDefinition Width="80\*"/>

</Grid.ColumnDefinitions>

<TextBlock Text="x:" FontWeight="Bold" HorizontalAlignment="Right" VerticalAlignment="Bottom" Grid.ColumnSpan="2" />

<TextBlock Text="y:" Grid.Row="1" FontWeight="Bold" HorizontalAlignment="Right" VerticalAlignment="Bottom" Grid.ColumnSpan="2" />

<TextBox x:Name="xPos" Grid.Column="2" IsEnabled="False" Margin="5,0,0,0" />

<TextBox x:Name="yPos" Grid.Row="1" Grid.Column="2" IsEnabled="False" Margin="5,0,0,0" />

<TextBlock Text="Statistics" Grid.Column="0" Grid.Row="2" Grid.ColumnSpan="3" FontWeight="Bold" HorizontalAlignment="Center" TextDecorations="Underline" Margin="35,0,34,0" />

<TextBlock Text="Avg:" FontWeight="Bold" Grid.Row="3" HorizontalAlignment="Right" VerticalAlignment="Bottom" Grid.ColumnSpan="3" Margin="0,0,79,0" />

<TextBlock Text="Std:" FontWeight="Bold" Grid.Row="4" HorizontalAlignment="Right" VerticalAlignment="Bottom" Grid.ColumnSpan="2" />

<TextBlock Text="Pixels:" FontWeight="Bold" Grid.Row="5" HorizontalAlignment="Right" VerticalAlignment="Bottom" Grid.ColumnSpan="2" />

<TextBox x:Name="avgImg" Grid.Row="3" Grid.Column="2" IsEnabled="False" Margin="5,0,0,0" />

<TextBox x:Name="stdImg" Grid.Row="4" Grid.Column="2" IsEnabled="False" Margin="5,0,0,0" />

<TextBox x:Name="noImg" Grid.Row="5" Grid.Column="2" IsEnabled="False" Margin="5,0,0,0" />

<TextBlock Text="Box" Grid.Column="2" Grid.Row="6" FontWeight="Bold" HorizontalAlignment="Center" TextDecorations="Underline" Margin="9,0,49,0" />

<TextBlock FontFamily="Palatino Linotype" Grid.Row="6" HorizontalAlignment="Right" VerticalAlignment="Bottom" FontWeight="Bold" FontSize="16" Grid.ColumnSpan="2" Grid.RowSpan="2">

<Run>c</Run><Run Typography.Variants="Subscript">x</Run><Run>:</Run>

</TextBlock>

<TextBlock FontFamily="Palatino Linotype" Grid.Row="7" HorizontalAlignment="Right" VerticalAlignment="Bottom" FontWeight="Bold" FontSize="16" Grid.ColumnSpan="3" Margin="0,0,79,0" Grid.RowSpan="2">

<Run>c</Run><Run Typography.Variants="Subscript">y</Run><Run>:</Run>

</TextBlock>

<TextBlock FontFamily="Palatino Linotype" Grid.Row="8" HorizontalAlignment="Right" VerticalAlignment="Bottom" FontWeight="Bold" FontSize="16" Grid.ColumnSpan="2" Grid.RowSpan="2">

<Run>c</Run><Run Typography.Variants="Subscript">z</Run><Run>:</Run>

</TextBlock>

<TextBox x:Name="xSize" Grid.Row="7" Grid.Column="2" IsEnabled="False" Margin="5,0,0,0" />

<TextBox x:Name="ySize" Grid.Row="8" Grid.Column="2" IsEnabled="False" Margin="5,0,0,0" />

<TextBox x:Name="zSize" Grid.Row="9" Grid.Column="2" IsEnabled="False" Margin="5,0,0,0" />

</Grid>

</GroupBox>

</StackPanel>

<StackPanel Orientation="Horizontal" Margin="0,20,0,0" VerticalAlignment="Top">

<Label x:Name="countername" x:FieldModifier="private" Margin="5,5,0,5" Content="slice:" />

<TextBox x:Name="txtNum" x:FieldModifier="private" Margin="0,5,0,5" Width="30" Text="0" TextChanged="txtNum\_TextChanged" />

<Button x:Name="cmdUp" x:FieldModifier="private" Margin="5,5,0,5" Width="20" Click="cmdUp\_Click" >

<StackPanel>

<Image Source="Pictures/circle-up-icon.png" />

</StackPanel>

</Button>

<Button x:Name="cmdDown" x:FieldModifier="private" Margin="0,5,0,5" Width="20" Click="cmdDown\_Click" >

<StackPanel>

<Image Source="Pictures/circle-down-icon.png" />

</StackPanel>

</Button>

<ComboBox x:Name="fileNameBox" x:FieldModifier="private" Margin="5,5,0,5" Width="100" SelectionChanged="fileName\_SelectionChanged"/>

<Label x:Name="resizename" x:FieldModifier="private" Margin="5,5,0,5" Content="resize:" />

<CheckBox x:Name="resize" Margin="0,13,0,5" Checked="resize\_Checked" Unchecked="resize\_UnChecked" />

<Button x:Name="remove" x:FieldModifier="private" Margin="5,5,0,5" Width="25" ToolTip="Delete current slice" Click="remove\_Click">

<StackPanel>

<Image Source="Pictures/delete.png" />

</StackPanel>

</Button>

</StackPanel>

</Grid>

</Window>

**MainWindow.xaml.cs**

using Microsoft.Win32;

using System.Collections.Generic;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Media.Imaging;

using System.IO;

using System.Windows.Input;

using System.Windows.Media;

using System;

using System.Windows.Shapes;

namespace BOLD

{

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

public const int FONT\_HEIGHT = 12;

public MainWindow()

{

InitializeComponent();

txtNum.Text = \_numSlice.ToString();

color\_scale = color\_scale\_gray;

}

// Menu items controlers

private void exit\_Click(object sender, RoutedEventArgs e)

{

Close();

}

private void AddImage(ImageSlice slice, string sliceName)

{

// insert new data image

slice.sliceFileName = sliceName;

\_imageData.Add(slice);

// update combobox

ComboboxItem item = new ComboboxItem();

item.Text = sliceName;

item.Value = \_numImage;

fileNameBox.Items.Add(item);

fileNameBox.SelectedIndex = fileNameBox.Items.Count - 1;

\_numImage++;

if (\_imageData[fileNameBox.SelectedIndex].zeroIntensity == -1)

{

zeroBond.Content = "";

zeroBond.DataContext = 0.5;

}

else

{

zeroBond.Content = Convert.ToString(getZeroPointSlider());

zeroBond.DataContext = \_imageData[fileNameBox.SelectedIndex].zeroIntensity;

}

}

private void ReplaceImage(ImageSlice slice, string sliceName)

{

// replace data image

slice.sliceFileName = sliceName;

\_imageData[fileNameBox.SelectedIndex] = slice;

// update combobox

(fileNameBox.Items[fileNameBox.SelectedIndex] as ComboboxItem).Text = sliceName;

// update Image

NumSlice = \_numSlice;

}

private void add\_Click(object sender, RoutedEventArgs e)

{

OpenFileDialog openFileDialog = new OpenFileDialog();

openFileDialog.Filter = "TXT files (\*.txt)|\*.txt|All files (\*.\*)|\*.\*";

if (openFileDialog.ShowDialog() == true)

{

// load image from file

ImageSlice slice = new ImageSlice(openFileDialog.FileName);

\_numSlice = 1;

// update text field

txtNum.Text = \_numSlice.ToString();

AddImage(slice, System.IO.Path.GetFileNameWithoutExtension(openFileDialog.FileName));

// Activate difference button if at least 2 images are added

activate\_deactivate\_Buttons();

}

}

private void replace\_Click(object sender, RoutedEventArgs e)

{

if (fileNameBox.SelectedIndex == -1)

throw new System.Exception("No image selected, cannot replace!");

OpenFileDialog openFileDialog = new OpenFileDialog();

if (openFileDialog.ShowDialog() == true)

{

// load image from file

ImageSlice slice = new ImageSlice(openFileDialog.FileName);

\_numSlice = 1;

// update text field

txtNum.Text = \_numSlice.ToString();

ReplaceImage(slice, System.IO.Path.GetFileNameWithoutExtension(openFileDialog.FileName));

}

}

private void new\_Click(object sender, RoutedEventArgs e)

{

\_imageData.Clear();

fileNameBox.Items.Clear();

image.Source = null;

activate\_deactivate\_Buttons();

txtNum.Text = "0";

}

private void save\_As\_Click(object sender, RoutedEventArgs e)

{

if (fileNameBox.SelectedIndex == -1)

throw new System.Exception("No image selected, cannot save!");

SaveFileDialog saveFileDialog = new SaveFileDialog();

saveFileDialog.Filter = "TXT files (\*.txt)|\*.txt|All files (\*.\*)|\*.\*";

if (saveFileDialog.ShowDialog() == true)

{

\_imageData[fileNameBox.SelectedIndex].SaveImage(saveFileDialog.FileName);

}

}

private void save\_PNG\_Click(object sender, RoutedEventArgs e)

{

if (fileNameBox.SelectedIndex == -1)

throw new System.Exception("No image selected, cannot save!");

SaveFileDialog saveFileDialog = new SaveFileDialog();

saveFileDialog.Filter = "PNG files (\*.PNG)|\*.png|All files (\*.\*)|\*.\*";

if (saveFileDialog.ShowDialog() == true)

{

using (var fileStream = new FileStream(saveFileDialog.FileName, FileMode.Create))

{

BitmapEncoder encoder = new PngBitmapEncoder();

encoder.Frames.Add(BitmapFrame.Create(transformBitmap() as BitmapSource));

encoder.Save(fileStream);

}

}

}

private void save\_Statistics\_Click(object sender, RoutedEventArgs e)

{

if (fileNameBox.SelectedIndex == -1)

throw new System.Exception("No image loaded, cannot export!");

if (selectionBox.Visibility != Visibility.Visible)

throw new System.Exception("No selection availible!");

SaveFileDialog saveFileDialog = new SaveFileDialog();

saveFileDialog.Filter = "TXT files (\*.TXT)|\*.txt|All files (\*.\*)|\*.\*";

if (saveFileDialog.ShowDialog() == true)

{

using (var fileStream = new StreamWriter(saveFileDialog.FileName))

{

ImageSlice slices = \_imageData[fileNameBox.SelectedIndex];

fileStream.WriteLine("Slice\tAvarage\tStdDev\t#Points\tsizeX({0})\tsizeY({0})\tsizeZ({0})", slices.realUnit);

for (int i = 1; i <= slices.zSize; i++)

{

var r = convertSelectionToRect(slices);

var stats = slices.GetStatistics(r, i);

var dim = slices.GetDimensions(r);

fileStream.WriteLine("{0}\t{1}\t{2}\t{3}\t{4}\t{5}\t{6}\t", i,

stats.Item1, stats.Item2, stats.Item3,

dim.Item1, dim.Item2, dim.Item3);

}

}

}

}

private void about\_Click(object sender, RoutedEventArgs e)

{

var about = new About();

about.Show();

}

private Rectangle color\_scale;

private void MenuItem\_Grayscale(object sender, RoutedEventArgs e)

{

checked\_rb.IsChecked = false;

checked\_rgb.IsChecked = false;

color\_scale\_gray.Visibility = Visibility.Visible;

color\_scale\_rb.Visibility = Visibility.Collapsed;

color\_scale\_rgb.Visibility = Visibility.Collapsed;

color\_scale = color\_scale\_gray;

NumSlice = \_numSlice;

}

private void MenuItem\_RedBlue(object sender, RoutedEventArgs e)

{

checked\_gray.IsChecked = false;

checked\_rgb.IsChecked = false;

color\_scale\_gray.Visibility = Visibility.Collapsed;

color\_scale\_rb.Visibility = Visibility.Visible;

color\_scale\_rgb.Visibility = Visibility.Collapsed;

color\_scale = color\_scale\_rb;

NumSlice = \_numSlice;

}

private void MenuItem\_Click\_RGB(object sender, RoutedEventArgs e)

{

checked\_gray.IsChecked = false;

checked\_rb.IsChecked = false;

color\_scale\_gray.Visibility = Visibility.Collapsed;

color\_scale\_rb.Visibility = Visibility.Collapsed;

color\_scale\_rgb.Visibility = Visibility.Visible;

color\_scale = color\_scale\_rgb;

NumSlice = \_numSlice;

}

// List of images

private List<ImageSlice> \_imageData = new List<ImageSlice>();

private int \_numImage = 0;

// slice changer controler

private int \_numSlice = 0;

public int NumSlice

{

get { return \_numSlice; }

set

{

if (\_imageData.Count == 0)

return;

// check if slice is in range

if (value < 1 )

\_numSlice = 1;

else if (value > \_imageData[fileNameBox.SelectedIndex].zSize)

\_numSlice = \_imageData[fileNameBox.SelectedIndex].zSize;

else

\_numSlice = value;

// update text box

txtNum.Text = \_numSlice.ToString();

// if resize box is checked resize image and insert to image control

CheckedResize = resize.IsChecked.Value;

// change dataContent for Image: max and min intensity

upperBond.Content = (Math.Truncate(\_imageData[fileNameBox.SelectedIndex].maxIntensity \* 100d) / 100d).ToString();

lowerBond.Content = (Math.Truncate(\_imageData[fileNameBox.SelectedIndex].minIntensity \* 100d) / 100d).ToString();

// set DataContext for zeroIntensity

// this will be then used by xaml and MarginConverter

if (\_imageData[fileNameBox.SelectedIndex].zeroIntensity == -1)

{

zeroBond.Visibility = Visibility.Collapsed;

color\_scale\_gray.Visibility = Visibility.Collapsed;

color\_scale\_rb.Visibility = Visibility.Collapsed;

color\_scale\_rgb.Visibility = Visibility.Collapsed;

zeroBondSlider.Visibility = Visibility.Collapsed;

}

else

{

zeroBond.Visibility = Visibility.Visible;

//color\_scale\_rb.Visibility = Visibility.Visible;

//color\_scale\_gray.Visibility = Visibility.Collapsed;

zeroBondSlider.Visibility = Visibility.Visible;

}

statistics(\_imageData[fileNameBox.SelectedIndex]);

}

}

private void cmdUp\_Click(object sender, RoutedEventArgs e)

{

NumSlice++;

}

private void cmdDown\_Click(object sender, RoutedEventArgs e)

{

NumSlice--;

}

private void txtNum\_TextChanged(object sender, TextChangedEventArgs e)

{

if (txtNum == null)

return;

if (int.TryParse(txtNum.Text, out \_numSlice))

NumSlice = \_numSlice;

}

public double getZeroPointSlider()

{

if (\_imageData.Count==0 || \_imageData[fileNameBox.SelectedIndex].zeroIntensity == -1)

return 0;

else

return

Math.Truncate((

(double)(zeroBond.DataContext) \* (

\_imageData[fileNameBox.SelectedIndex].maxIntensity - \_imageData[fileNameBox.SelectedIndex].minIntensity

) + \_imageData[fileNameBox.SelectedIndex].minIntensity)\*100d)/100d;

}

// controler for image resize checkbox

bool \_checkedResize = false;

public bool CheckedResize

{

get { return \_checkedResize; }

set

{

if (\_imageData.Count == 0)

return;

// insert image to image control

ImageSlice slice = \_imageData[fileNameBox.SelectedIndex];

image.Source = slice.GetImage(\_numSlice - 1, Convert.ToInt32(getZeroPointSlider()), color\_scale.Fill as LinearGradientBrush);

// resize if needed

if (resize.IsChecked ?? true)

{

Int32Rect resizedSelection = new Int32Rect(slice.selection.Y, slice.selection.X, slice.selection.Height, slice.selection.Width);

image.Source = new CroppedBitmap(image.Source as BitmapSource, resizedSelection);

}

\_checkedResize = value;

}

}

private void resize\_Checked(object sender, RoutedEventArgs e)

{

CheckedResize = true;

}

private void resize\_UnChecked(object sender, RoutedEventArgs e)

{

CheckedResize = false;

}

// controler for combobox -> loaded image changer

private void fileName\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

if (fileNameBox.SelectedIndex == -1)

return;

if (\_numSlice > \_imageData[fileNameBox.SelectedIndex].zSize)

NumSlice = \_numSlice = \_imageData[fileNameBox.SelectedIndex].zSize;

else

NumSlice = \_numSlice;

}

// buttons of different operation controllers

private void difference\_ClickAB(object sender, RoutedEventArgs e)

{

ImageSlice slice = \_imageData[\_imageData.Count - 2] - \_imageData[\_imageData.Count - 1];

AddImage(slice, slice.sliceFileName);

activate\_deactivate\_Buttons();

}

private void difference\_ClickBA(object sender, RoutedEventArgs e)

{

ImageSlice slice = \_imageData[\_imageData.Count - 1] - \_imageData[\_imageData.Count - 2];

AddImage(slice, slice.sliceFileName);

activate\_deactivate\_Buttons();

}

private void sum\_Click(object sender, RoutedEventArgs e)

{

ImageSlice slice = \_imageData[\_imageData.Count - 2] + \_imageData[\_imageData.Count - 1];

AddImage(slice, slice.sliceFileName);

activate\_deactivate\_Buttons();

}

private void activate\_deactivate\_Buttons()

{

if (\_imageData.Count<2)

{

differenceAB.IsEnabled = false;

differenceBA.IsEnabled = false;

differenceAB\_A.IsEnabled = false;

sum.IsEnabled = false;

avg3.IsEnabled = false;

}

else if (\_imageData.Count == 2)

{

differenceAB.IsEnabled = true;

differenceBA.IsEnabled = true;

differenceAB\_A.IsEnabled = true;

sum.IsEnabled = true;

avg3.IsEnabled = false;

}

else

{

differenceAB.IsEnabled = true;

differenceBA.IsEnabled = true;

differenceAB\_A.IsEnabled = true;

sum.IsEnabled = true;

avg3.IsEnabled = true;

}

}

// remove button

private void remove\_Click(object sender, RoutedEventArgs e)

{

if (fileNameBox.SelectedIndex == -1)

throw new System.Exception("No image selected, cannot remove!");

\_imageData.RemoveAt(fileNameBox.SelectedIndex);

fileNameBox.Items.RemoveAt(fileNameBox.SelectedIndex);

image.Source = null;

activate\_deactivate\_Buttons();

}

// functionality of selection box for active image

bool mouseDown = false; // Set to 'true' when mouse is held down.

Point mouseDownPos; // The point where the mouse button was clicked down.

private void Grid\_MouseDown(object sender, MouseButtonEventArgs e)

{

// Capture and track the mouse.

mouseDown = true;

mouseDownPos = e.GetPosition(image);

mouseDownPos = transformMouse(mouseDownPos);

image.CaptureMouse();

// Initial placement of the drag selection box.

Canvas.SetLeft(selectionBox, mouseDownPos.X);

Canvas.SetTop(selectionBox, mouseDownPos.Y);

selectionBox.Width = 0;

selectionBox.Height = 0;

// Make the drag selection box visible.

selectionBox.Visibility = Visibility.Visible;

}

private void Grid\_MouseUp(object sender, MouseButtonEventArgs e)

{

// Release the mouse capture and stop tracking it.

mouseDown = false;

image.ReleaseMouseCapture();

// Hide the drag selection box.

//selectionBox.Visibility = Visibility.Collapsed;

Point mouseUpPos = e.GetPosition(image);

mouseUpPos = transformMouse(mouseUpPos);

//

// The mouse has been released, calculate all related statistics below

//

ImageSlice slice = \_imageData[fileNameBox.SelectedIndex];

xPos.Text = Math.Round(mouseUpPos.X \* slice.xSize / image.Width).ToString();

yPos.Text = Math.Round(mouseUpPos.Y \* slice.ySize / image.Height).ToString();

statistics(slice);

}

private void statistics(ImageSlice slice)

{

if (selectionBox.Width > 0 && selectionBox.Height > 0)

{

var r = convertSelectionToRect(slice);

var stats = slice.GetStatistics(r, \_numSlice);

var dim = slice.GetDimensions(r);

avgImg.Text = (Math.Truncate(stats.Item1 \* 1000) / 1000).ToString();

stdImg.Text = (Math.Truncate(stats.Item2 \* 1000) / 1000).ToString();

noImg.Text = stats.Item3.ToString();

xSize.Text = (Math.Truncate(dim.Item1 \* 100.0) / 100.0).ToString() +

" " + slice.realUnit;

ySize.Text = (Math.Truncate(dim.Item2 \* 100.0) / 100.0).ToString() +

" " + slice.realUnit;

zSize.Text = dim.Item3.ToString() + " " + slice.realUnit;

}

}

private Int32Rect convertSelectionToRect(ImageSlice slice)

{

var r = new Int32Rect(

Convert.ToInt32(Canvas.GetLeft(selectionBox) \* slice.xSize / image.Width),

Convert.ToInt32(Canvas.GetTop(selectionBox) \* slice.ySize / image.Height),

Convert.ToInt32(selectionBox.Width \* slice.xSize / image.Width),

Convert.ToInt32(selectionBox.Height \* slice.ySize / image.Height)

);

if (CheckedResize)

{

r.Width = Convert.ToInt32(r.Width \* slice.selection.Width / slice.xSize);

r.Height = Convert.ToInt32(r.Height \* slice.selection.Height / slice.ySize);

r.X = Convert.ToInt32(slice.selection.X + r.Width);

r.Y = Convert.ToInt32(slice.selection.Y + r.Height);

}

return r;

}

private void Grid\_MouseMove(object sender, MouseEventArgs e)

{

if (mouseDown)

{

// When the mouse is held down, reposition the drag selection box.

Point mousePos = e.GetPosition(image);

mousePos = transformMouse(mousePos);

if (mouseDownPos.X < mousePos.X)

{

Canvas.SetLeft(selectionBox, mouseDownPos.X);

selectionBox.Width = mousePos.X - mouseDownPos.X;

}

else

{

Canvas.SetLeft(selectionBox, mousePos.X);

selectionBox.Width = mouseDownPos.X - mousePos.X;

}

if (mouseDownPos.Y < mousePos.Y)

{

Canvas.SetTop(selectionBox, mouseDownPos.Y);

selectionBox.Height = mousePos.Y - mouseDownPos.Y;

}

else

{

Canvas.SetTop(selectionBox, mousePos.Y);

selectionBox.Height = mouseDownPos.Y - mousePos.Y;

}

}

}

/// <summary>

/// transforming mouse position invers to transformation of image object

/// </summary>

/// <param name="mousePos">initial position to be transformed</param>

/// <returns>transformed position</returns>

private Point transformMouse(Point mousePos)

{

var rt = new RotateTransform(-90, image.ActualWidth / 2.0, image.ActualHeight / 2.0);

var st = new ScaleTransform(-1, 1, image.ActualWidth / 2.0, image.ActualHeight / 2.0);

mousePos = st.Transform(mousePos);

mousePos = rt.Transform(mousePos);

return mousePos;

}

/// <summary>

/// Function builds image plot into canvas together with color scale and convert them to ImageSource

/// </summary>

/// <returns>ImageSource, which can be then saved as PNG file</returns>

private ImageSource transformBitmap()

{

// build data image

var newimage = new Image();

newimage.Source = image.Source;

newimage.Width = (image.Source as BitmapSource).PixelWidth;

newimage.Height = (image.Source as BitmapSource).PixelHeight;

newimage.RenderTransform = image.RenderTransform;

// build scale

var newscale = new Rectangle();

newscale.Stroke = color\_scale.Stroke;

newscale.StrokeThickness = color\_scale.StrokeThickness;

newscale.Width = color\_scale.Width;

newscale.Height = newimage.Height;

newscale.Fill = color\_scale.Fill;

// build scale labels

var scalelabels = new Grid();

scalelabels.Width = scale.ActualWidth;

scalelabels.Height = newimage.Height;

var label1 = new Label();

var label2 = new Label();

var label3 = new Label();

label1.Content = upperBond.Content;

label1.VerticalAlignment = upperBond.VerticalAlignment;

label2.Content = lowerBond.Content;

label2.VerticalAlignment = lowerBond.VerticalAlignment;

label3.Content = zeroBond.Content;

label3.VerticalAlignment = zeroBond.VerticalAlignment;

label3.Margin = new Thickness(0, 0, 0, (double)zeroBond.DataContext \* scalelabels.Height - FONT\_HEIGHT);

scalelabels.Children.Add(label1);

scalelabels.Children.Add(label2);

scalelabels.Children.Add(label3);

// create container for all objects

Canvas container = new Canvas();

container.Children.Add(newimage);

container.Children.Add(newscale);

container.Children.Add(scalelabels);

Canvas.SetLeft(newscale, newimage.Width);

Canvas.SetLeft(scalelabels, newimage.Width + newscale.Width);

// redraw whole canvas to update children

container.Arrange(new Rect(0,0,newimage.Width+newscale.Width, newimage.Height));

// render the result to a new bitmap.

var target = new RenderTargetBitmap((int)(newimage.Width + newscale.Width + scalelabels.Width), (int)newimage.Height, 96, 96, PixelFormats.Pbgra32);

target.Render(container);

return target;

}

private void zeroBondSlider\_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)

{

if (zeroBondSlider.DataContext!=null)

zeroBond.DataContext = (sender as Slider).Value;

zeroBond.Content = Convert.ToString(getZeroPointSlider());

NumSlice = \_numSlice;

zeroBondSlider.DataContext = false;

}

private void avg3\_Click(object sender, RoutedEventArgs e)

{

ImageSlice slice = (\_imageData[\_imageData.Count-3] + \_imageData[\_imageData.Count - 2] + \_imageData[\_imageData.Count - 1])/3;

AddImage(slice, slice.sliceFileName);

activate\_deactivate\_Buttons();

statistics(slice);

}

private void difference\_ClickABA(object sender, RoutedEventArgs e)

{

ImageSlice slice = (\_imageData[\_imageData.Count - 2] - \_imageData[\_imageData.Count - 1] ) / \_imageData[\_imageData.Count - 2];

AddImage(slice, slice.sliceFileName);

activate\_deactivate\_Buttons();

statistics(slice);

}

private void minmax\_Click(object sender, RoutedEventArgs e)

{

MinMaxDialog minmaxDialog = new MinMaxDialog(lowerBond.Content.ToString(), upperBond.Content.ToString());

if (minmaxDialog.ShowDialog() == true)

{

if (\_imageData.Count>0)

{

\_imageData[fileNameBox.SelectedIndex].minIntensity = Convert.ToDouble(minmaxDialog.MinScale);

\_imageData[fileNameBox.SelectedIndex].maxIntensity = Convert.ToDouble(minmaxDialog.MaxScale);

NumSlice = \_numSlice;

}

}

}

}

}

**ImageSlice.cs**

using System;

using System.IO;

using System.Linq;

using System.Windows.Media.Imaging;

using System.Windows.Media;

using System.Windows;

using System.Globalization;

using System.Collections.Generic;

namespace BOLD

{

[Serializable]

class ImageSlice

{

public int xSize { get; private set; }

public int ySize { get; private set; }

public int zSize { get; private set; }

//private int xSize, ySize, zSize;

public double xRealSize { get; private set; }

public double yRealSize { get; private set; }

public double zRealSize { get; private set; }

public string realUnit { get; private set; }

public string sliceName { get; private set; }

public string sliceFileName { get; set; }

public double[,,] sliceData { get; private set; }

public double minIntensity { get; set; }

public double maxIntensity { get; set; }

public double zeroIntensity { get; private set; }

public Int32Rect selection { get; private set; }

public string header { get; private set; }

/// <summary>

/// Constructor of clase ImageSlice. Loades a file with an image to a object ImageSlice

/// </summary>

/// <param name="filePath">Full path to file to be load</param>

public ImageSlice(string filePath)

{

string data = "";

string[] words = null;

header = "";

try

{ // Open the text file using a stream reader.

using (StreamReader reader = new StreamReader(filePath))

{

data = reader.ReadLine() + "\n";

while (data[0] == '#')

{

header += data;

data = reader.ReadLine() + "\n";

}

data += reader.ReadToEnd();

words = header.Split(new[] { ' ', ' ', '\t', '\r', '\n' }, StringSplitOptions.RemoveEmptyEntries);

}

}

catch (Exception e)

{

MessageBox.Show("The file could not be read: " + e.Message, "ImageSlice", MessageBoxButton.OK, MessageBoxImage.Error);

return;

}

// reading the header

if (words.Count() == 0)

throw new ArgumentException("ImageSlice", "Header data not found");

for (int i = 0; i < words.Count(); i++)

{

if (words[i] == "#")

{

if (words[i + 1] == "x\_dimension:")

xSize = Int32.Parse(words[i + 2]);

if (words[i + 1] == "y\_dimension:")

ySize = Int32.Parse(words[i + 2]);

if (words[i + 1] == "z\_dimension:")

zSize = Int32.Parse(words[i + 2]);

if (words[i + 1] == "File:")

sliceName = words[i + 2];

if (words[i + 1] == "Voxel\_Size:")

{

xRealSize = Double.Parse(words[i + 2], NumberStyles.Any, CultureInfo.InvariantCulture);

yRealSize = Double.Parse(words[i + 3], NumberStyles.Any, CultureInfo.InvariantCulture);

zRealSize = Double.Parse(words[i + 4], NumberStyles.Any, CultureInfo.InvariantCulture);

}

if (words[i + 1] == "Voxel\_Units:")

realUnit = words[i + 2];

}

}

words = data.Split(new[] { ' ', ' ', '\t', '\r', '\n' }, StringSplitOptions.RemoveEmptyEntries);

if (words.Count() == 0)

throw new ArgumentException("ImageSlice", "Image data not found");

// reading the contrast data

sliceData = new double[xSize, ySize, zSize];

maxIntensity = 0;

minIntensity = Int32.MaxValue;

for (int i = 0; i < words.Count(); i += 4)

{

double word = Double.Parse(words[i + 3]);

sliceData[Int32.Parse(words[i]) - 1, Int32.Parse(words[i + 1]) - 1, Int32.Parse(words[i + 2]) - 1] = word;

if (maxIntensity < word)

maxIntensity = word;

if (word != 0 && minIntensity > word)

minIntensity = word;

}

if (minIntensity > 0)

zeroIntensity = 0.5;

else

zeroIntensity = -minIntensity / (double)(maxIntensity - minIntensity);

// initialize selection

selection = new Int32Rect();

int minX, maxX, minY, maxY, slice\_index=0;

do

{

minX = xSize; maxX = 0; minY = ySize; maxY = 0;

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

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

{

if (sliceData[i, j, slice\_index] != 0)

{

if (minX > i)

minX = i;

if (maxX < i)

maxX = i;

if (minY > j)

minY = j;

if (maxY < j)

maxY = j;

}

}

if (maxX < minX)

minX = 0;

if (maxY < minY)

minY = 0;

slice\_index++;

} while (minX == maxX && minY == maxY && slice\_index<zSize+1);

selection = new Int32Rect(minX, minY, maxX - minX, maxY - minY);

}

/// <summary>

/// Converts numeric data from the class with slice to BitmapSource

/// </summary>

/// <param name="i\_slice">number of slice to return</param>

/// <param name="zeroPoint">separation between red and blue color</param>

/// <param name="paletteBrush">color palette taken e.g. from Rectangle.Fill source</param>

/// <returns>BitmapSource with image form this class.</returns>

public BitmapSource GetImage(int i\_slice, int zeroPoint, LinearGradientBrush paletteBrush)

{

// TO DO: implement grayscal and rgb reaction on slider

int stride = ySize + (ySize % 4);

byte[] pixelData = new byte[xSize \* stride];

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

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

{

byte color = sliceData[i, j, i\_slice] == 0 ? (byte)0 :

(byte)(Math.Round((sliceData[i, j, i\_slice] - minIntensity) / (double)(maxIntensity - minIntensity) \* 255.0));

pixelData[j + i \* stride] = color;

}

Color col = GradientStopCollectionExtensions.GetRelativeColor(paletteBrush.GradientStops, 0.25);

col = GradientStopCollectionExtensions.GetRelativeColor(paletteBrush.GradientStops, 0.75);

var myPalette = new BitmapPalette(

Enumerable.Range(0, 256).Select(c => GradientStopCollectionExtensions.GetRelativeColor(paletteBrush.GradientStops, 1-c/255.0)).ToList()

);

var pixelFormat = PixelFormats.Indexed8;

return BitmapSource.Create(xSize, ySize, 96, 96, pixelFormat, myPalette, pixelData, stride);

}

/// <summary>

/// Saves image to file

/// </summary>

/// <param name="filePath">Path to a file as a string.</param>

public void SaveImage(string filePath)

{

try

{ // Open the text file using a stream writer

using (StreamWriter writer = new StreamWriter(filePath))

{

writer.Write(header);

for (int k = 0; k < zSize; k++)

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

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

{

//if (sliceData[i,j,k]>0)

writer.WriteLine((i + 1).ToString() + " " + (j + 1).ToString() + " " + (k + 1).ToString() + " " + sliceData[i, j, k].ToString());

}

}

}

catch (Exception e)

{

MessageBox.Show("The file could not be write: " + e.Message, "SaveImage", MessageBoxButton.OK, MessageBoxImage.Error);

return;

}

}

/// <summary>

/// Calculates average of selected region.

/// </summary>

/// <param name="r">Rectangle object with coordinates of the selected region</param>

/// <param name="i\_slice">number of slice.</param>

/// <returns>Tuple with avarage and standard deviation of the region.</returns>

public Tuple<double, double, int> GetStatistics(Int32Rect r, int i\_slice)

{

var avg = 0.0;

var std = 0.0;

for (int i = r.X < 0 ? 0 : r.X; i < (r.X + r.Width > xSize ? xSize : r.X + r.Width) ; i++)

for (int j = r.Y < 0 ? 0 : r.Y ; j < (r.Y + r.Height > ySize ? ySize : r.Y + r.Height) ; j++)

{

avg += sliceData[i, j, i\_slice-1];

std += (sliceData[i, j, i\_slice-1] \* sliceData[i, j, i\_slice-1]);

}

avg /= r.Width \* r.Height;

std = Math.Sqrt(std / r.Width / r.Height - avg \* avg);

return Tuple.Create(avg, std, r.Width \* r.Height);

}

public Tuple<double, double, double> GetDimensions(Int32Rect r)

{

double x = r.Width \* xRealSize;

double y = r.Height \* yRealSize;

double z = zRealSize;

return Tuple.Create(x, y, z);

}

/// <summary>

/// Summs up two images.

/// </summary>

/// <param name="c1">image 1<param>

/// <param name="c2">image 2</param>

/// <returns>image 1 + image 2</returns>

public static ImageSlice operator +(ImageSlice c1, ImageSlice c2)

{

if (c1.xSize != c2.xSize || c1.ySize != c2.ySize || c1.zSize != c2.zSize)

throw new ArgumentOutOfRangeException("operator +: all sizes have to be the same");

ImageSlice c = ObjectCopier.Clone<ImageSlice>(c1);

c.sliceFileName = "(" + c1.sliceFileName + "+" + c2.sliceFileName + ")"; ;

var tmp = c.minIntensity;

c.minIntensity = c.maxIntensity;

c.maxIntensity = tmp;

for (int i = 0; i < c.xSize; i++)

for (int j = 0; j < c.ySize; j++)

for (int k = 0; k < c.zSize; k++)

{

c.sliceData[i, j, k] += c2.sliceData[i, j, k];

if (c.sliceData[i, j, k] < c.minIntensity)

c.minIntensity = c.sliceData[i, j, k];

if (c.sliceData[i, j, k] > c.maxIntensity)

c.maxIntensity = c.sliceData[i, j, k];

}

if (c.minIntensity > 0)

c.zeroIntensity = 0.5;

else

c.zeroIntensity = -c.minIntensity / (double)(c.maxIntensity - c.minIntensity);

return c;

}

/// <summary>

/// Differentiate two images.

/// </summary>

/// <param name="c1">image 1</param>

/// <param name="c2">image 2</param>

/// <returns>image 1 - image 2</returns>

public static ImageSlice operator -(ImageSlice c1, ImageSlice c2)

{

if (c1.xSize != c2.xSize || c1.ySize != c2.ySize || c1.zSize != c2.zSize)

throw new ArgumentOutOfRangeException("operator -", "all sizes have to be the same");

ImageSlice c = ObjectCopier.Clone<ImageSlice>(c1);

c.sliceFileName = "(" + c1.sliceFileName + "-" + c2.sliceFileName + ")";

var tmp = c.minIntensity;

c.minIntensity = c.maxIntensity;

c.maxIntensity = tmp;

for (int i = 0; i < c.xSize; i++)

for (int j = 0; j < c.ySize; j++)

for (int k = 0; k < c.zSize; k++)

{

c.sliceData[i, j, k] -= c2.sliceData[i, j, k];

if (c.sliceData[i, j, k] < c.minIntensity)

c.minIntensity = c.sliceData[i, j, k];

if (c.sliceData[i, j, k] > c.maxIntensity)

c.maxIntensity = c.sliceData[i, j, k];

}

if (c.minIntensity > 0)

c.zeroIntensity = 0.5;

else

c.zeroIntensity = -c.minIntensity / (double)(c.maxIntensity - c.minIntensity);

return c;

}

public static ImageSlice operator /(ImageSlice c1, double c2)

{

ImageSlice c = ObjectCopier.Clone<ImageSlice>(c1);

c.sliceFileName = "(" + c1.sliceFileName + "/3)";

var tmp = c.minIntensity;

c.minIntensity = c.maxIntensity;

c.maxIntensity = tmp;

for (int i = 0; i < c.xSize; i++)

for (int j = 0; j < c.ySize; j++)

for (int k = 0; k < c.zSize; k++)

{

c.sliceData[i, j, k] = (c.sliceData[i, j, k]/c2);

if (c.sliceData[i, j, k] < c.minIntensity)

c.minIntensity = c.sliceData[i, j, k];

if (c.sliceData[i, j, k] > c.maxIntensity)

c.maxIntensity = c.sliceData[i, j, k];

}

if (c.minIntensity > 0)

c.zeroIntensity = 0.5;

else

c.zeroIntensity = -c.minIntensity / (double)(c.maxIntensity - c.minIntensity);

return c;

}

public static ImageSlice operator /(ImageSlice c1, ImageSlice c2)

{

ImageSlice c = ObjectCopier.Clone<ImageSlice>(c1);

c.sliceFileName = "(" + c1.sliceFileName + "/" + c2.sliceFileName + ")";

var tmp = c.minIntensity;

c.minIntensity = c.maxIntensity;

c.maxIntensity = tmp;

for (int i = 0; i < c.xSize; i++)

for (int j = 0; j < c.ySize; j++)

for (int k = 0; k < c.zSize; k++)

{

if (c2.sliceData[i, j, k]==0)

{

c.sliceData[i, j, k] = 0;

continue;

}

c.sliceData[i, j, k] = (c.sliceData[i, j, k] / c2.sliceData[i, j, k]);

if (c.sliceData[i, j, k] < c.minIntensity)

c.minIntensity = c.sliceData[i, j, k];

if (c.sliceData[i, j, k] > c.maxIntensity)

c.maxIntensity = c.sliceData[i, j, k];

}

if (c.minIntensity > 0)

c.zeroIntensity = 0.5;

else

c.zeroIntensity = -c.minIntensity / (double)(c.maxIntensity - c.minIntensity);

return c;

}

}

}

**About.xaml**

<Window x:Class="BOLD.About"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:BOLD"

mc:Ignorable="d"

Title="About" Height="270" Width="300"

ResizeMode="NoResize" MaxWidth="300" MaxHeight="270" Icon="Pictures/Logo.png">

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="1\*"/>

<RowDefinition Height="4\*"/>

</Grid.RowDefinitions>

<StackPanel Orientation="Horizontal" Grid.Row="0">

<Image Source="Pictures/Logo.png" Height="40" Width="40" Margin="10,0,0,0" />

<TextBlock Text="BOLD" VerticalAlignment="Center" Margin="10,0,0,0" />

</StackPanel>

<Grid Grid.Row="1" Margin="20,10">

<Grid.RowDefinitions>

<RowDefinition Height="1\*" />

<RowDefinition Height="1\*" />

<RowDefinition Height="1\*" />

<RowDefinition Height="1\*" />

<RowDefinition Height="1\*" />

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="1\*" />

<ColumnDefinition Width="2\*" />

</Grid.ColumnDefinitions>

<TextBlock Grid.Row="0" Grid.Column="0" Text="Description:"/>

<TextBlock Grid.Row="0" Grid.Column="1" Text="Program for multiple MRI images manipulation" TextWrapping="Wrap" />

<TextBlock Grid.Row="1" Grid.Column="0" Text="Release:"/>

<TextBlock Grid.Row="1" Grid.Column="1" Text="January 2016" TextWrapping="Wrap" />

<TextBlock Grid.Row="2" Grid.Column="0" Text="Authors:"/>

<TextBlock Grid.Row="2" Grid.Column="1" TextWrapping="Wrap" Text="Agnieszka Boś-Liedke"/>

<TextBlock Grid.Row="3" Grid.Column="0" Text="Contacts:"/>

<TextBlock Grid.Row="3" Grid.Column="1" TextWrapping="Wrap" Text="agnieszkaboss@gmail.com"/>

<Button x:Name="ok" Grid.Row="4" Grid.ColumnSpan="2" HorizontalAlignment="Center" VerticalAlignment="Bottom" Width="60" Height="20" Content="OK" Click="ok\_Click"/>

</Grid>

</Grid>

</Window>

**About.xaml.cs**

using System.Windows;

namespace BOLD

{

/// <summary>

/// Interaction logic for Window1.xaml

/// </summary>

public partial class About : Window

{

public About()

{

InitializeComponent();

}

private void ok\_Click(object sender, RoutedEventArgs e)

{

Close();

}

}

}

**ComboBoxItem.cs**

namespace BOLD

{

public class ComboboxItem

{

public string Text { get; set; }

public int Value { get; set; }

public override string ToString()

{

return Text;

}

}

}

**GradientStopCollectionExtensions.cs**

using System.Linq;

using System.Windows.Media;

namespace BOLD

{

public static class GradientStopCollectionExtensions

{

public static Color GetRelativeColor(this GradientStopCollection gsc, double offset)

{

GradientStop before = gsc.Where(w => w.Offset == gsc.Min(m => m.Offset)).First();

GradientStop after = gsc.Where(w => w.Offset == gsc.Max(m => m.Offset)).First();

foreach (var gs in gsc)

{

if (gs.Offset < offset && gs.Offset > before.Offset)

{

before = gs;

}

if (gs.Offset > offset && gs.Offset < after.Offset)

{

after = gs;

}

}

var color = new Color();

color.ScA = (float)((offset - before.Offset) \* (after.Color.ScA - before.Color.ScA) / (after.Offset - before.Offset) + before.Color.ScA);

color.ScR = (float)((offset - before.Offset) \* (after.Color.ScR - before.Color.ScR) / (after.Offset - before.Offset) + before.Color.ScR);

color.ScG = (float)((offset - before.Offset) \* (after.Color.ScG - before.Color.ScG) / (after.Offset - before.Offset) + before.Color.ScG);

color.ScB = (float)((offset - before.Offset) \* (after.Color.ScB - before.Color.ScB) / (after.Offset - before.Offset) + before.Color.ScB);

return color;

}

}

}

**JScriptConverter.cs**

using System;

using System.Windows.Data;

using System.CodeDom.Compiler;

using System.Reflection;

namespace Utils.Avalon

{

public sealed class JScriptConverter : IMultiValueConverter, IValueConverter

{

private delegate object Evaluator(string code, object[] values);

private static Evaluator evaluator;

static JScriptConverter()

{

string source =

@"import System;

class Eval

{

public function Evaluate(code : String, values : Object[]) : Object

{

return eval(code);

}

}";

CompilerParameters cp = new CompilerParameters();

cp.GenerateInMemory = true;

foreach (Assembly assembly in AppDomain.CurrentDomain.GetAssemblies())

if (System.IO.File.Exists(assembly.Location))

cp.ReferencedAssemblies.Add(assembly.Location);

CompilerResults results = (new Microsoft.JScript.JScriptCodeProvider())

.CompileAssemblyFromSource(cp, source);

Assembly result = results.CompiledAssembly;

Type eval = result.GetType("Eval");

evaluator = (Delegate.CreateDelegate(

typeof(Evaluator),

Activator.CreateInstance(eval),

"Evaluate") as Evaluator);

}

private bool trap = false;

public bool TrapExceptions

{

get { return this.trap; }

set { this.trap = true; }

}

public object Convert(object[] values, System.Type targetType,

object parameter, System.Globalization.CultureInfo culture)

{

try

{

return evaluator(parameter.ToString(), values);

}

catch

{

if (trap)

return null;

else

throw;

}

}

public object Convert(object value, Type targetType,

object parameter, System.Globalization.CultureInfo culture)

{

return Convert(new object[] { value }, targetType, parameter, culture);

}

public object[] ConvertBack(object value, System.Type[] targetTypes,

object parameter, System.Globalization.CultureInfo culture)

{

throw new System.NotSupportedException();

}

public object ConvertBack(object value, Type targetType,

object parameter, System.Globalization.CultureInfo culture)

{

throw new System.NotSupportedException();

}

}

}

**ObjectCopier.cs**

using System;

using System.IO;

using System.Runtime.Serialization;

using System.Runtime.Serialization.Formatters.Binary;

namespace BOLD

{

/// <summary>

/// Reference Article http://www.codeproject.com/KB/tips/SerializedObjectCloner.aspx

/// Provides a method for performing a deep copy of an object.

/// Binary Serialization is used to perform the copy.

/// </summary>

public static class ObjectCopier

{

/// <summary>

/// Perform a deep Copy of the object.

/// </summary>

/// <typeparam name="T">The type of object being copied.</typeparam>

/// <param name="source">The object instance to copy.</param>

/// <returns>The copied object.</returns>

public static T Clone<T>(T source)

{

if (!typeof(T).IsSerializable)

{

throw new ArgumentException("The type must be serializable.", "source");

}

// Don't serialize a null object, simply return the default for that object

if (Object.ReferenceEquals(source, null))

{

return default(T);

}

IFormatter formatter = new BinaryFormatter();

Stream stream = new MemoryStream();

using (stream)

{

formatter.Serialize(stream, source);

stream.Seek(0, SeekOrigin.Begin);

return (T)formatter.Deserialize(stream);

}

}

}

}

**MarginConverter.cs**

using System.Windows;

using System.Windows.Data;

namespace BOLD

{

class MarginConverter : IMultiValueConverter

{

public object Convert(object[] values, System.Type targetType, object parameter, System.Globalization.CultureInfo culture)

{

if (values[1] == null || (double)values[1] == -1)

return new Thickness(0, 0, 0, 0);

else

return new Thickness(0, 0, 0, System.Convert.ToDouble((double)values[0] \* (double)values[1] - MainWindow.FONT\_HEIGHT));

}

public object[] ConvertBack(object value, System.Type[] targetType, object parameter, System.Globalization.CultureInfo culture)

{

return null;

}

}

}

**MinMaxDialog.xaml**

<Window x:Class="BOLD.MinMaxDialog"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:BOLD"

mc:Ignorable="d"

SizeToContent="WidthAndHeight"

WindowStartupLocation="CenterScreen"

ContentRendered="Window\_ContentRendered"

Title="MinMaxDialog" >

<Grid Margin="15">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto" />

<ColumnDefinition Width="\*" />

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="Auto" />

<RowDefinition Height="Auto" />

<RowDefinition Height="Auto" />

<RowDefinition Height="Auto" />

</Grid.RowDefinitions>

<Label Name="lblMin" Grid.Column="0" Grid.Row="1" MinWidth="50">Min:</Label>

<TextBox Name="minScale" Grid.Column="1" Grid.Row="1" MinWidth="100">0</TextBox>

<Label Name="lblQuestion" Grid.Column="0" Grid.Row="2" >Max:</Label>

<TextBox Name="maxScale" Grid.Column="1" Grid.Row="2" >0</TextBox>

<WrapPanel Grid.Row="3" Grid.ColumnSpan="2" HorizontalAlignment="Right" Margin="0,15,0,0">

<Button IsDefault="True" Name="btnDialogOk" Click="btnDialogOk\_Click" MinWidth="60" Margin="0,0,10,0">\_Ok</Button>

<Button IsCancel="True" MinWidth="60">\_Cancel</Button>

</WrapPanel>

</Grid>

</Window>

**MinMaxDialog.xaml.cs**

using System;

using System.Windows;

namespace BOLD

{

/// <summary>

/// Interaction logic for MinMaxDialog.xaml

/// </summary>

public partial class MinMaxDialog : Window

{

public MinMaxDialog(string sMin="0", string sMax="0")

{

InitializeComponent();

MinScale = sMin;

MaxScale = sMax;

}

private void btnDialogOk\_Click(object sender, RoutedEventArgs e)

{

this.DialogResult = true;

}

private void Window\_ContentRendered(object sender, EventArgs e)

{

minScale.SelectAll();

minScale.Focus();

}

public string MinScale

{

get { return minScale.Text; }

set { minScale.Text = value; }

}

public string MaxScale

{

get { return maxScale.Text; }

set { maxScale.Text = value; }

}

}

}