

Visual studio 使用 emguCV

用户手册

2023 年 8 月 23 日

目录

出错处理.....	4
Nugget 联机安装 OpenCVsharp 时，出现“基础连接已经关闭，发送时发生错误”。	4
在 visual studio 2010 配置 EmguCV，编程语言为 C#.....	4
安装 EmguCV	5
配置环境变量.....	6
导入 UI 插件.....	9
添加引用.....	11
测试用例.....	12
visual studio 2022 测试用例	15
spy++ Tool	15
GetWindowText 函数.....	17
窗口捕获.....	19
FindWindow 函数	19
PrintWindow 函数	19
灰度判断.....	22
摄像头捕获.....	24
基础图像处理.....	25
灰度转换.....	25
canny 算子	26
直方图均方化.....	27
高斯滤波.....	28
均值滤波.....	30

Visual studio 2010 配置 OpenCV 环境。编程语言为 C#。

在 C#环境中，选择安装 opencvsharp 版本，该版本能实现 opencv 的功能。

安装方法：工具->NuGet 程序包管理器->管理解决方案的 NuGet 包

但是 visual studio 2010 默认是没有 Nuget 程序包的，所以我们去到网上下载 [NuGet Package Manager - Visual Studio Marketplace](#)

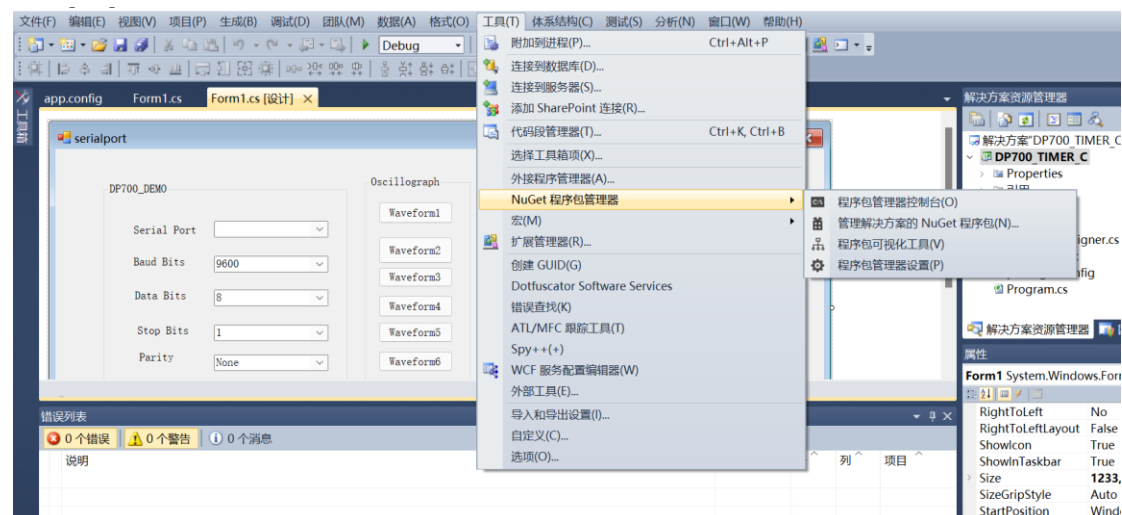
下载好后，直接双击打开，就可以使用 Nuget 插件了。

安装成功后，可以再工具->扩展管理器中看到，如下图



工具栏中此时也有对应的 Nuget 程序包管理器了。

如下图所示：



Visual studio 2010 所支持的 framework 框架最高为 4.0（.NET Framework 4.0），如果想要适配

报错处理：[vs2010 nuget 基础连接已经关闭:发送时发生错误 vs 基础连接已经关闭,发送时](#)

出错处理

Nuget 联机安装 OpenCVsharp 时，出现“基础连接已经关闭，发送时发生错误”。

解决方式：修改注册表，将以下内容使用记事本保存为.reg 格式，随后打开.net4 下的注册表，打开方式为 win+R，随后输入 regedit，打开注册表编辑器，之后导入刚刚保存的 reg 文件，重新启动 visual studio 2010，问题解决。

```
1 Windows Registry Editor Version 5.00
2
3 [HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\.NETFramework\v4.0.30319]
4 "SchUseStrongCrypto"=dword:00000001
5
6 [HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Microsoft\.NETFramework\v4.0.30319]
7 "SchUseStrongCrypto"=dword:00000001
```

Windows Registry Editor Version 5.00

[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\.NETFramework\v4.0.30319]

"SchUseStrongCrypto"=dword:00000001

[HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Microsoft\.NETFramework\v4.0.30319]

"SchUseStrongCrypto"=dword:00000001

在 visual studio 2010 配置 EmguCV，编程语言为 C#

EmguCV 是一个跨平台的图像处理库。它是 [OpenCV](#) 的 .NET 封装的版本。令人惊叹的包装器使得可以从.NET 编程语言调用 OpenCV 函数。支持 C#，VB, IronPython，和 VC++ 等一些语言。EmguCV 可以编译为 Mono，并且可

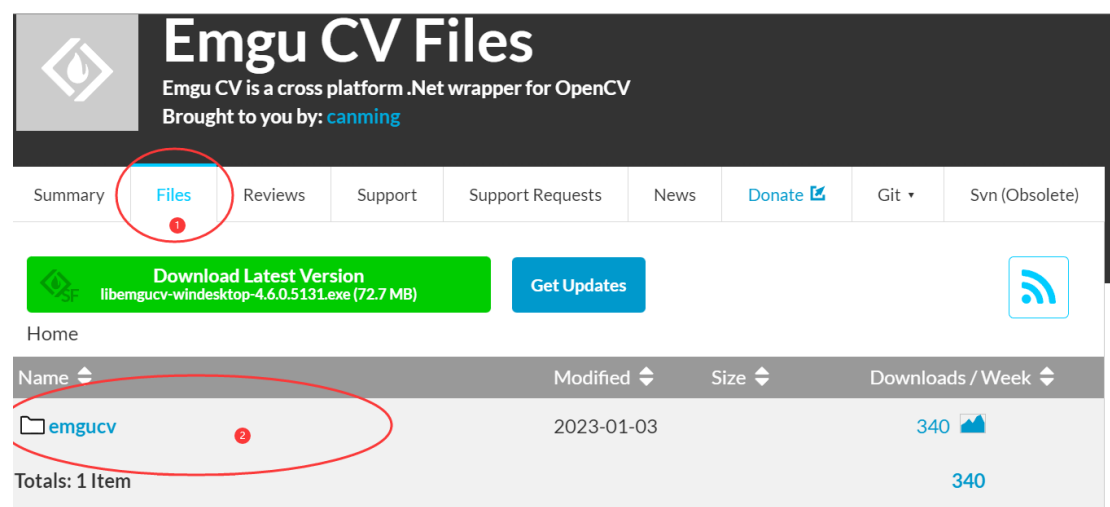
以运行在 Linux, Windows, Mac OS X, 和流行的移动平台, 如 Android, iPhone, iPod Touch, 和 iPad 设备上。

安装 EmguCV

Visual studio 2010 所支持的.netframework 框架最新为 4.0, 所以在安装 emgucv 的时候, 要注意兼容问题, 此处我们选择安装 emgucv3.4 版本, 最好安装此版本及更低的版本。

安装网址: <https://sourceforge.net/projects/emgucv/files/emgucv/>


进入网站后, 依次点击如下图所示:




进入到下面所示界面时, 下载选择对应的版本, 此处我们选择的 3.4.1

4.2.0	2020-02-11	3
4.1.1	2019-10-09	2
4.1.0	2019-05-25	0
4.0.1	2019-04-19	2
3.4.3	2018-10-23	10
3.4.1	2018-04-04	1
3.3	2017-11-11	8
3.2	2017-05-08	6
3.1.0-r16.12	2016-12-16	3

选择对应的版本进去后, 会有两个文件, 如下图所示, 推荐下载第一个可执行文件。下载好后, 双击打开, 安装到自定义的目录, 此处我们选择安装到 D 盘, 路径最好不要带有中文。

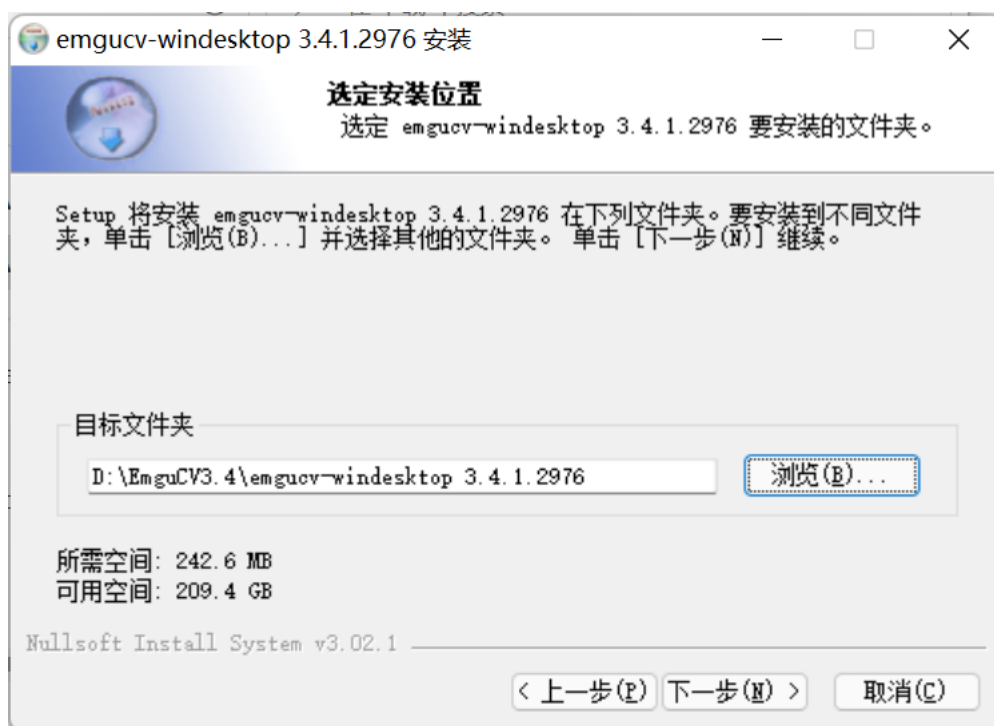
**Download Latest Version**
libemgucv-windesktop-4.6.0.5131.exe (72.7 MB)

Get Updates



Home / emgucv / 3.4.1

Name	Modified	Size	Downloads / Week
Parent folder			
libemgucv-windesktop-3.4.1.2976.exe	2018-04-04	47.8 MB	1
libemgucv-windesktop-3.4.1.2976.zip	2018-04-04	108.8 MB	0



配置环境变量

安装好 EmguCV 后，下一步就是配置环境变量
我们选择右击此电脑->属性->高级系统设置->环境变量



系统 > 关于

机带 RAM

16.0 GB (14.8 GB 可用)

设备 ID

18D36002-3450-4118-A78D-BC6BC31F32DF

产品 ID

00425-00000-00002-AA647

系统类型

64 位操作系统, 基于 x64 的处理器

笔和触控

没有可用于此显示器的笔或触控输入

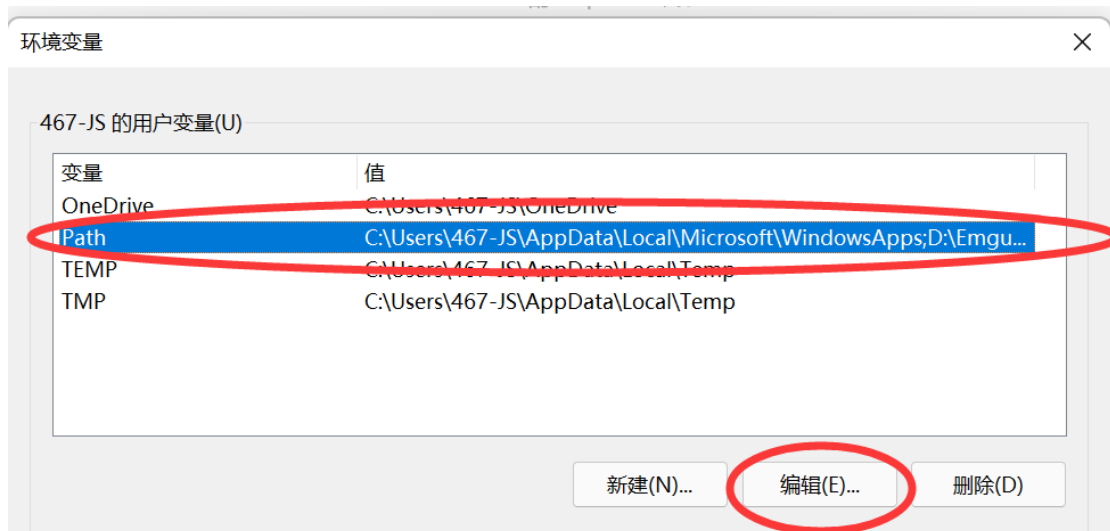
相关链接

[域或工作组](#)

[系统保护](#)

[高级系统设置](#)

2

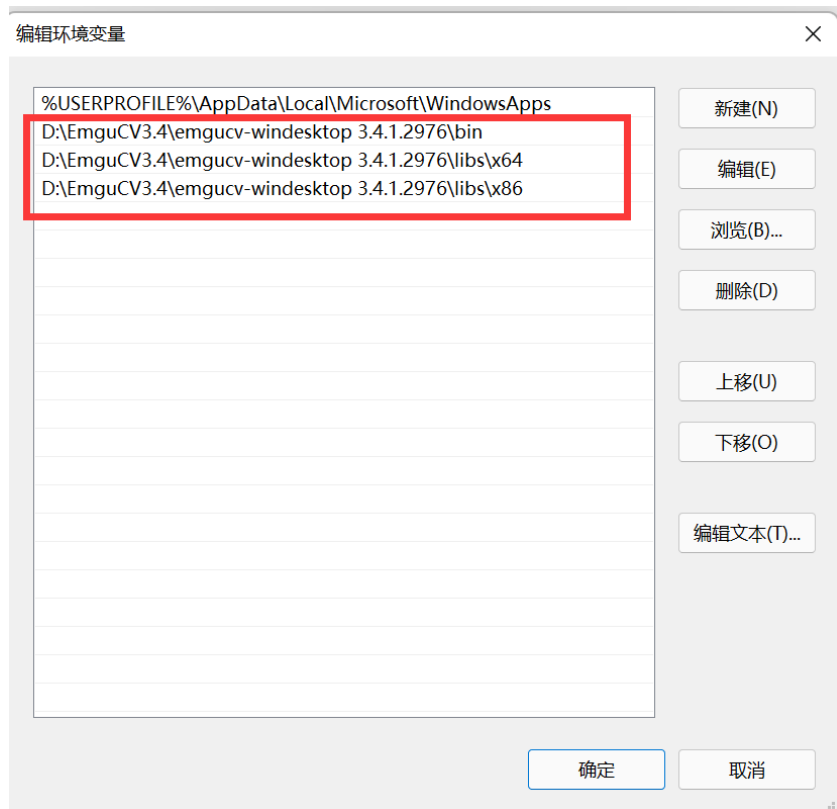


进入环境变量的编辑界面后，将以下路径添加进去（根据自己安装路径决定，不能照抄）

D:\EmguCV3.4\emgucv-windesktop 3.4.1.2976\bin

D:\EmguCV3.4\emgucv-windesktop 3.4.1.2976\libs\x64

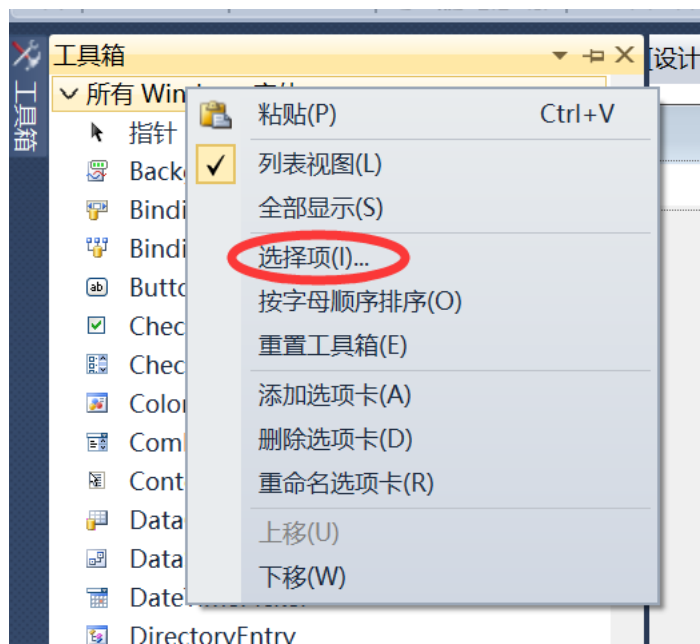
D:\EmguCV3.4\emgucv-windesktop 3.4.1.2976\libs\x86



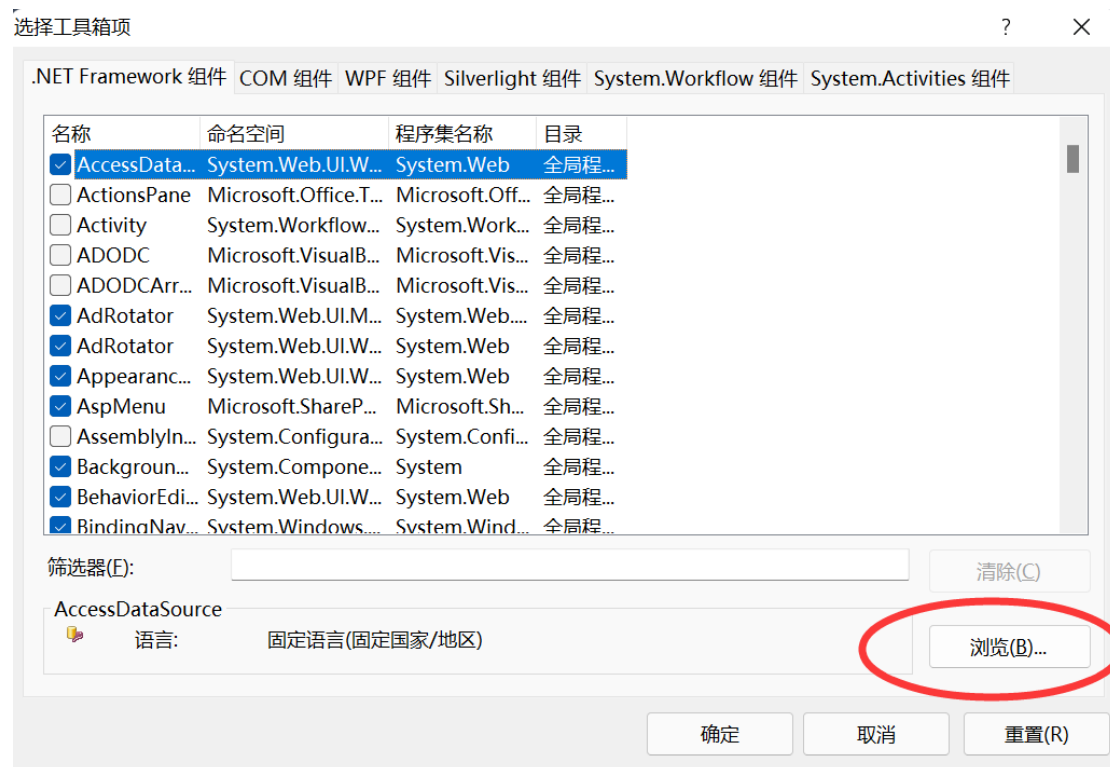
导入 UI 插件

注：在导入 UI 插件和添加引用这两个步骤前，需要手动创建一个窗体应用程序。

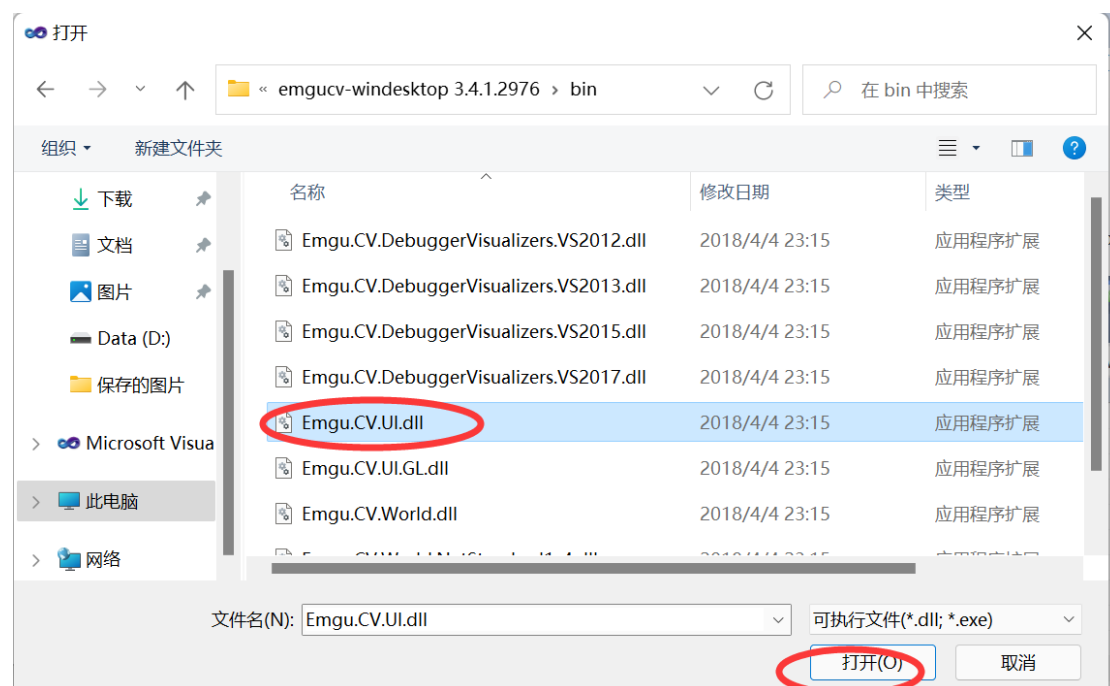
找到工具箱中的所有 Windows 窗体，右击，选择选择项



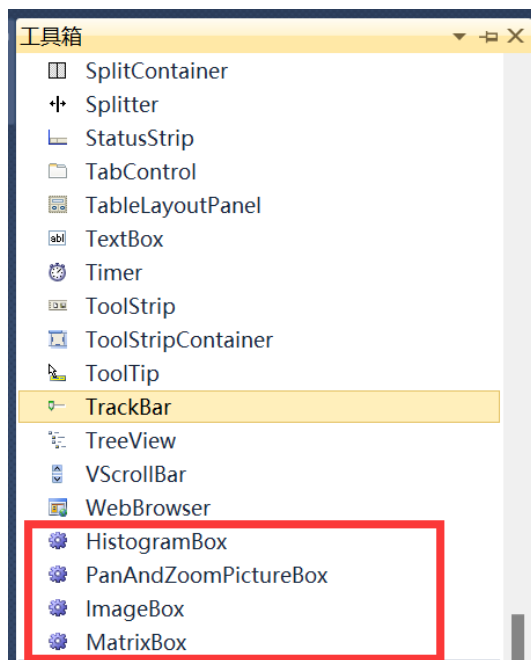
浏览目录



我们找到安装目录下的 bin 目录，选择 Emgu.CV.UI.dll

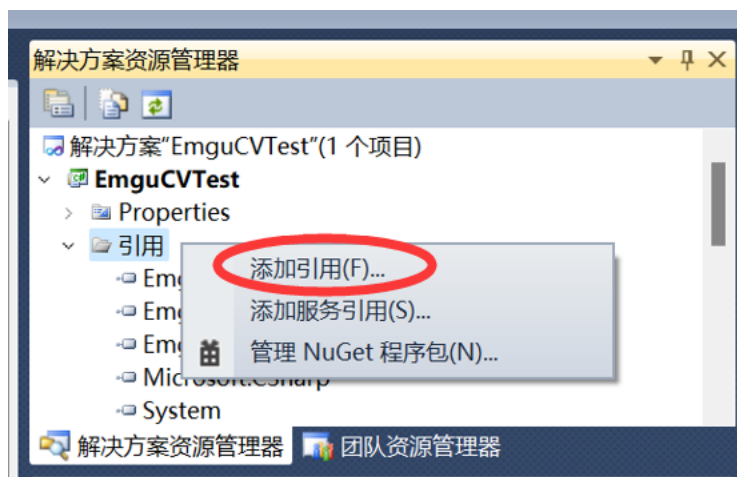


成功 UI 插件后，我们可以在工具箱看到多了几个插件。

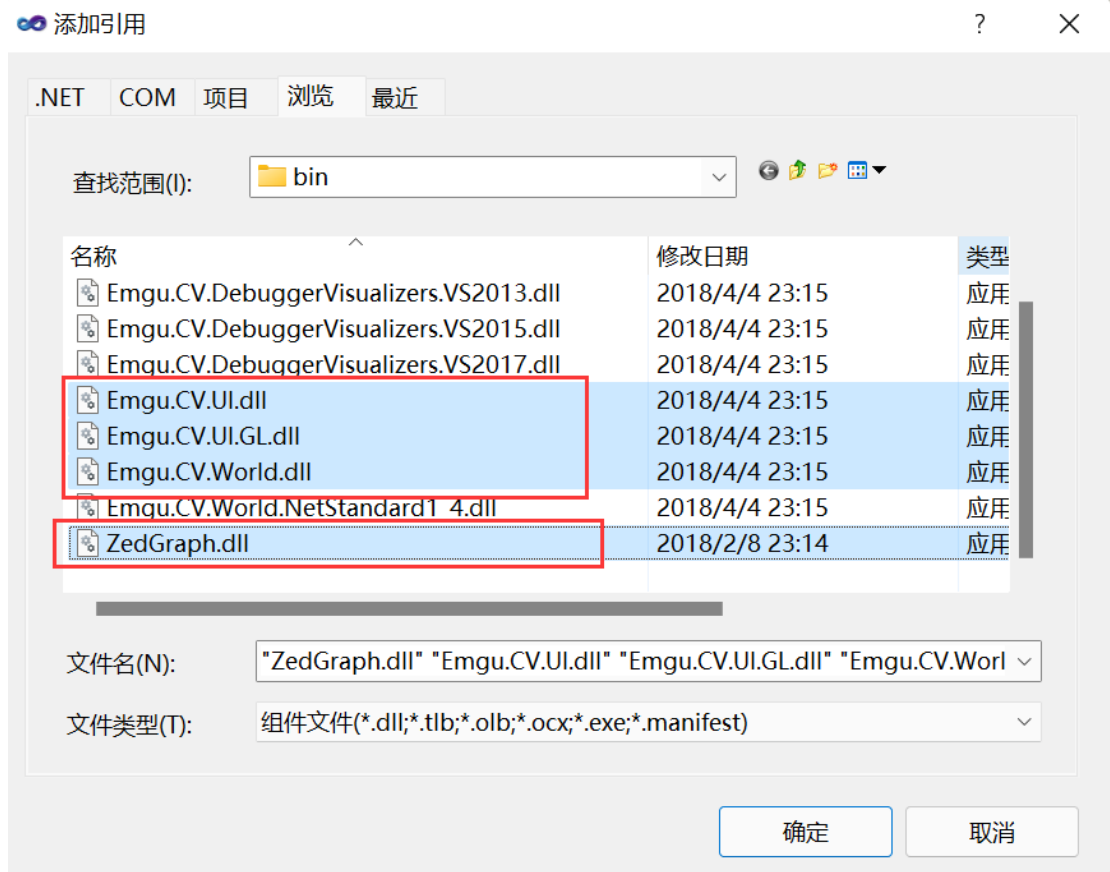


添加引用

在解决方案资源管理器面板中，找到我们项目下的引用选项
右击引用->添加引用



浏览安装目录下的 bin 目录，添加以下四个引用



测试用例

```
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Windows.Forms;  
using System.Text;  
using System.Threading.Tasks;
```

```
using Emgu.CV;  
using Emgu.CV.Structure;  
using Emgu.CV.CvEnum;  
using Emgu.Util;
```

```
namespace EmguCVTest  
{  
    static class Program  
    {  
        [STAThread]  
        static void Main()
```

```

    {

        try
        {
            // 读取图像文件
            string imagePath = "1.jpg";
            Mat srcImg = CvInvoke.Imread(imagePath);

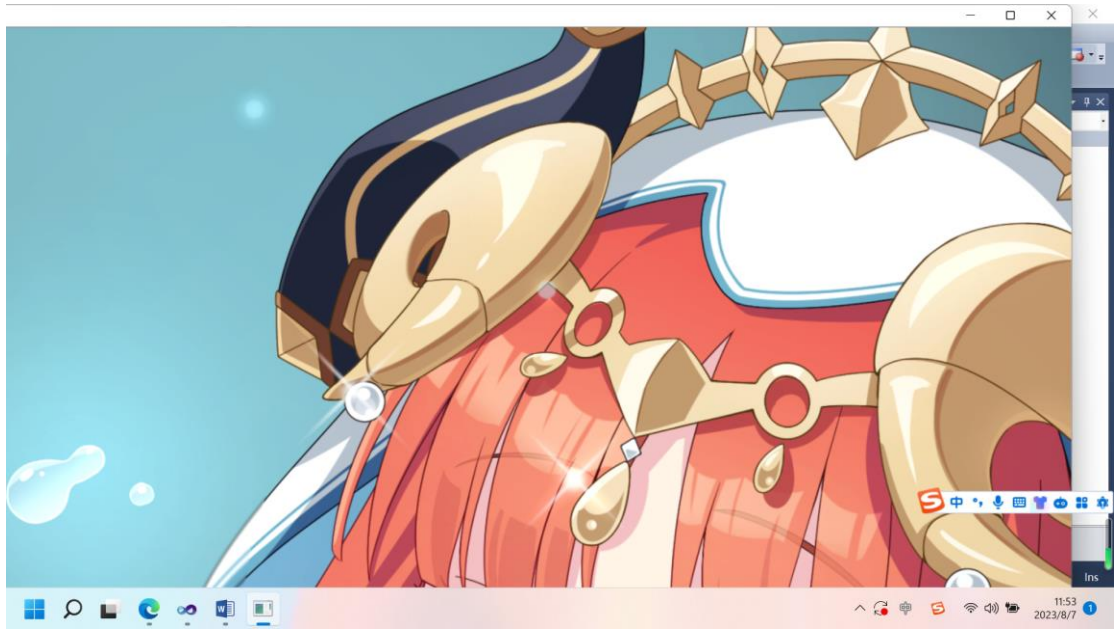
            // 检查图像是否成功读取
            if (srcImg != null && !srcImg.IsEmpty)
            {
                // 显示图像
                CvInvoke.Imshow("Img", srcImg);

                // 等待用户按下任意键
                CvInvoke.WaitKey(0);
            }
            else
            {
                MessageBox.Show("Failed to load the image.");
            }
        }
        catch (Exception ep)
        {
            MessageBox.Show(ep.Message);
        }
    }
}

```

运行程序后，将会加载对应的图片

注：需要将图片放在项目的 Debug 文件下，或者可以使用图片的绝对路径



visual studio 2022 测试用例

spy++ Tool

Spy++（也称为 "Spy++ Tool" 或 "Spy++ Utility"）是一个由 Microsoft 提供的 Windows 开发工具，用于监视和调试 Windows 程序和窗口的行为。它允许开发人员查看和分析正在运行的 Windows 应用程序的窗口层次结构、消息流和其他窗口相关的信息。

Spy++ 可以帮助开发人员了解窗口的层次结构、消息传递、窗口属性、类名、标题和其他与窗口相关的信息。开发人员可以使用 Spy++ 工具来调试和分析窗口应用程序的交互、消息处理、窗口绘制等方面的问题。

通过 Spy++ 工具，开发人员可以：

查看窗口层次结构，包括父窗口、子窗口和兄弟窗口。

监视窗口的消息流，包括窗口接收和发送的消息。

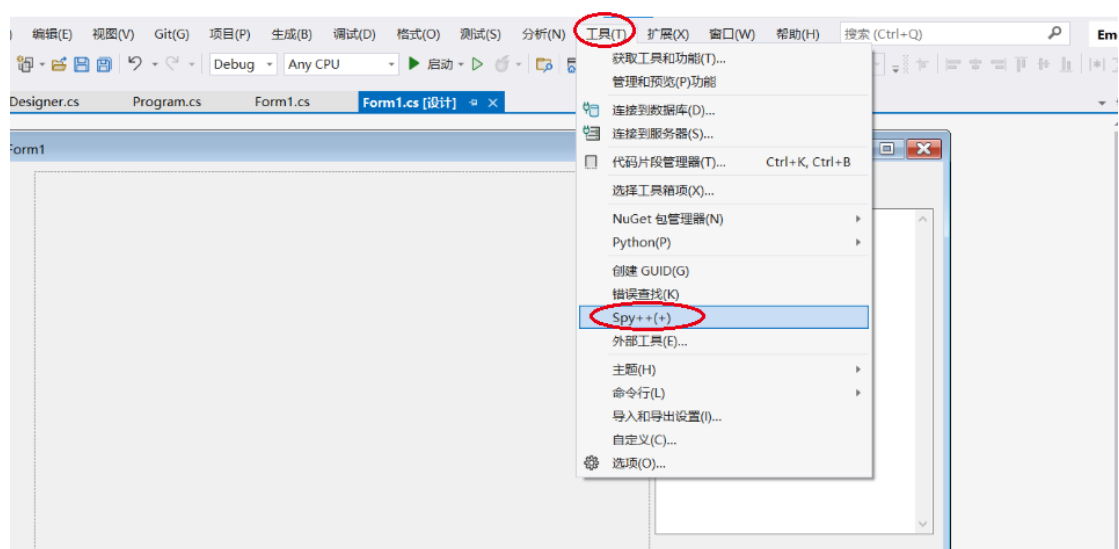
查看窗口的属性，如类名、标题、位置和大小等。

跟踪窗口的鼠标和键盘事件。

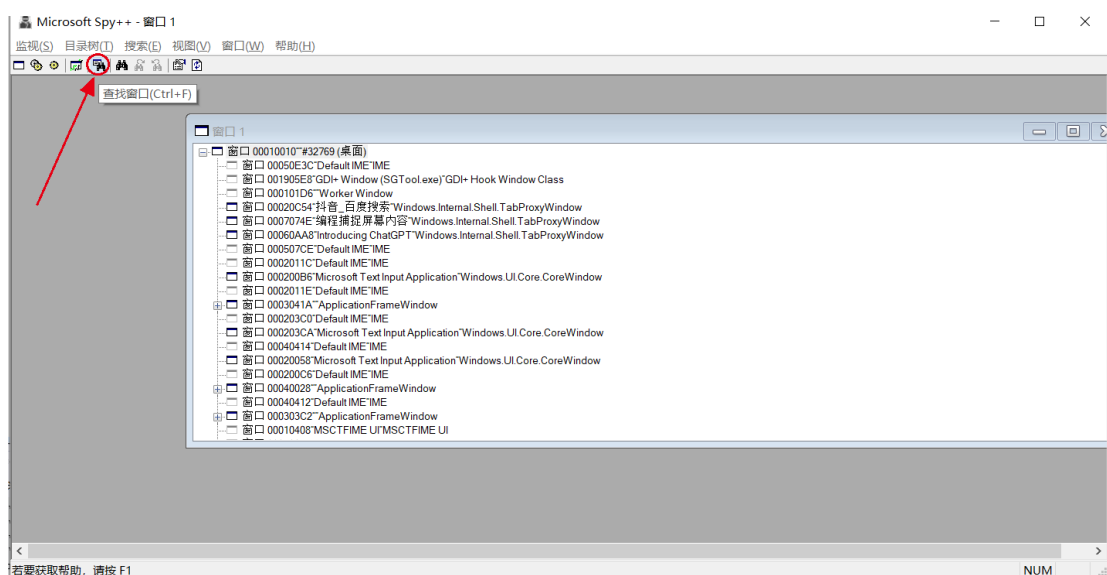
跟踪窗口的绘图和重绘操作。

此处我们使用 spy++来查看窗口的标题，以及窗口的类名，这两者的用途将在后面介绍。

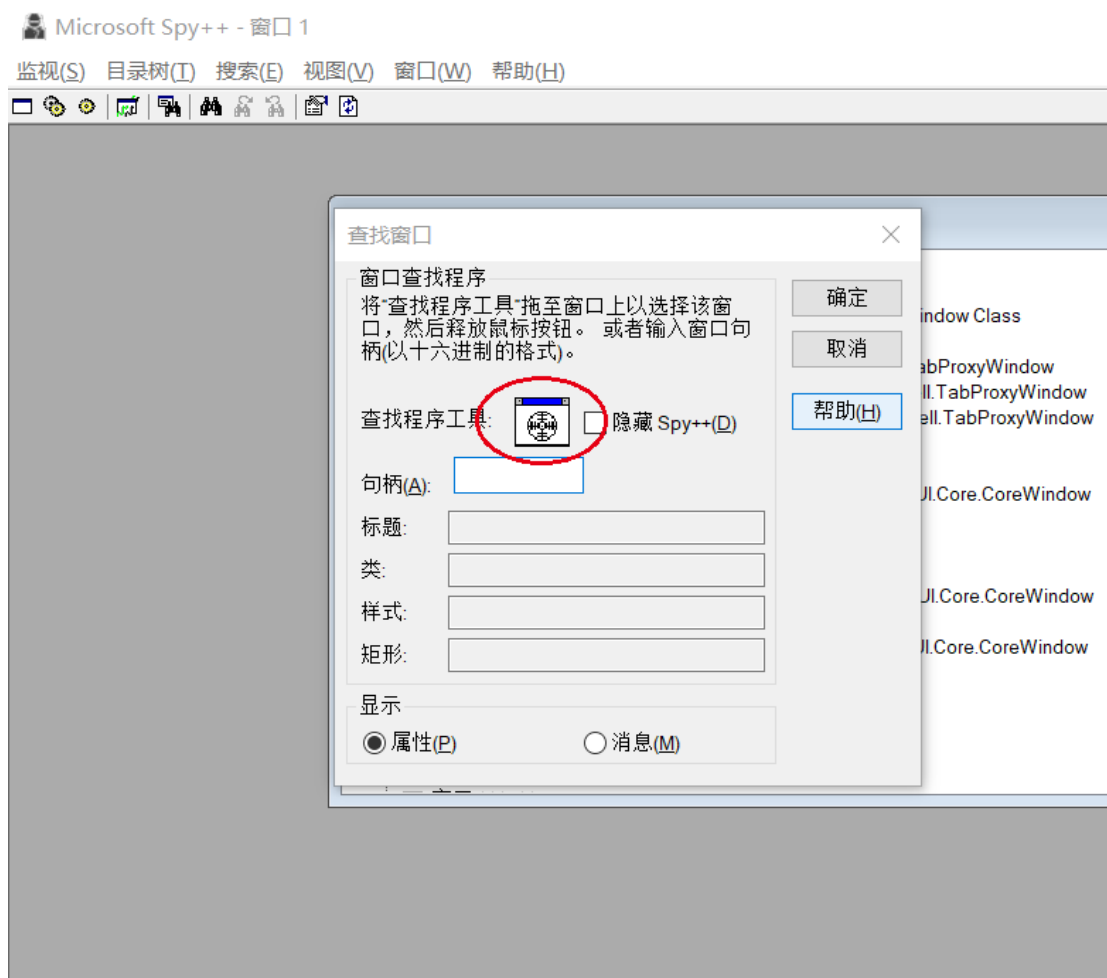
spy++ Tool在 visual studio中就有，我们以 visual studio 2022 为例，在工具栏中找到 spy++，并运行，如下图所示：



进入 spy++后，点击下图所示按钮：



点进图标后，将进入查找窗口界面，拖动查找程序图标到需要查找的窗口中，随后点击确定，即可获取相应的信息，此处以文件资源管理器窗口演示，如下图所示：





可以在属性检查器看到窗口的标题，以及窗口所属的类。

GetWindowText 函数

GetWindowText 函数是 Windows API 中的一个函数，用于获取指定窗口的标题文本（窗口标题）。它可以用于获取窗口的显示名称，通常是窗口标题栏中的文字。

函数原型：int GetWindowText(HWND hWnd, LPTSTR lpString, int nMaxCount);

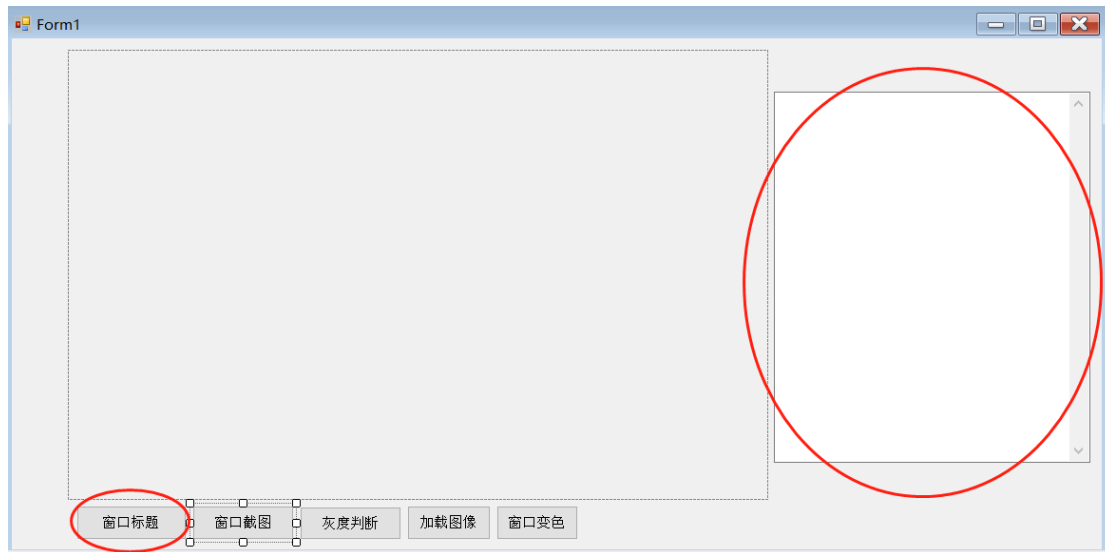
参数说明：

- **hWnd**：窗口句柄，表示要获取标题的目标窗口。
- **lpString**：用于接收窗口标题文本的缓冲区。
- **nMaxCount**：指定 **lpString** 缓冲区的大小（以字符数为单位）。

返回值：

- 如果函数成功，返回复制到缓冲区中的字符数，不包括终止 null 字符。
- 如果窗口没有标题，或者窗口句柄无效，返回 0。

通过此函数，我们在窗口界面设计了一个文本框用于接收返回的窗口标题：



具体的设计代码如下：

```
private void WindowTitle_Click(object sender, EventArgs e)
{
    bool EnumWindowCallback(IntPtr hWnd, IntPtr lParam)
    {
        const int maxWindowTextLength = 256;
        System.Text.StringBuilder windowText = new
System.Text.StringBuilder(maxWindowTextLength);
        int length = GetWindowText(hWnd, windowText, maxWindowTextLength);

        if (length > 0)
        {
            string windowTitle = windowText.ToString();
            WinTitleBox.AppendText("Window Title: " + windowTitle +
Environment.NewLine + "-----" + Environment.NewLine);
        }

        return true; // 继续遍历其他窗口
    }

    EnumWindows(EnumWindowCallback, IntPtr.Zero);
}
```

EnumWindows 函数：这个函数是用来遍历所有的窗口，他有两个参数，第一个参数是一个回调函数，第二个参数一般用不到，此处传递 IntPtr.Zero 即可。

在上面的代码中，我们为窗口标题 button 创建了一个点击事件，在时间函数中定义了一个回调函数 EnumWindowCallback，此回调函数用于将在 EnumWindows 函数调用时，对每个窗口执行这个回调函数，如果回调函数返回 true，则继续遍历执行，将窗口标题信息打印到文本框，打印的时候，手动换行，以及加上标识符，可以更好的区分。代码运行后的实际效果如下：



窗口捕获

FindWindow 函数

FindWindow 是一个 Windows API 函数，用于查找一个顶级窗口的句柄，根据窗口的类名和窗口名称（标题）。这个函数允许你根据窗口的特征来获取对应窗口的句柄，以便后续操作。

函数原型：HWND FindWindow(LPCTSTR lpClassName, LPCTSTR lpWindowName);

参数解释：

- **lpClassName**：窗口类名，一个字符串，用于匹配窗口的类名。这个参数是可选的，可以为 **NULL**，表示不考虑窗口的类名。
- **lpWindowName**：窗口名称，一个字符串，用于匹配窗口的标题。这个参数是可选的，可以为 **NULL**，表示不考虑窗口的标题。

返回值：

- 如果找到匹配的窗口，返回窗口的句柄（**HWND**）。
- 如果未找到匹配的窗口，返回 **NULL**。

PrintWindow 函数

PrintWindow 是一个 Windows API 函数，用于将一个窗口的内容绘制到指定的设备上，通常用于截取或复制窗口的图像。它可以用来获取一个窗口的截图，包括窗口的客户区域和非客户区域。

函数原型：BOOL PrintWindow(HWND hwnd, HDC hdcBlt, UINT nFlags);

参数解释:

- **hwnd**: 窗口的句柄, 表示要绘制的目标窗口。
- **hdcBlt**: 目标设备的句柄, 表示绘制的图像将要绘制到的设备上下文 (Device Context, 简称 DC)。可以是屏幕的 DC, 也可以是一个位图的 DC。
- **nFlags**: 绘制的选项标志, 可以是零或一些特定标志, 控制绘制行为。通常使用零。

返回值:

- 如果函数成功, 返回非零值。
- 如果函数失败, 返回零。可以通过调用 **GetLastError** 函数获取错误码。

完整的代码展示:

```
private void ScreenCapture_Click(object sender, EventArgs e)
{
    string targetWindowTitle = "Microsoft Word 文档"; // 指定目标窗口标题
    //FindWindow函数查找具有类名为 "TXGuiFoundation" 的窗口。IntPtr 是一个指向
    //内存中一个整数的指针, 它在这里用于存储窗口的句柄。
    IntPtr targetWindowHandle = FindWindow(null, targetWindowTitle); // 第一个参
    //数是类名, 第二个参数是窗口标题名

    if (targetWindowHandle != IntPtr.Zero)
    {
        RECT windowRect;
        GetWindowRect(targetWindowHandle, out windowRect); // out 表示 windowRect 是
        // 输出参数

        int width = windowRect.Right - windowRect.Left;
        int height = windowRect.Bottom - windowRect.Top;

        using (Bitmap bitmap = new Bitmap(width, height))
        {
            using (Graphics graphics = Graphics.FromImage(bitmap))
            {
                IntPtr hdc = graphics.GetHdc();
                PrintWindow(targetWindowHandle, hdc, 0);
                graphics.ReleaseHdc(hdc);
            }
            bitmap.Save("C:\\Users\\dhl\\Pictures\\测试图片\\123.png");
        }
    }
}
```

```

    }
    else
    {
        MessageBox.Show("Target window not found.");
    }
}

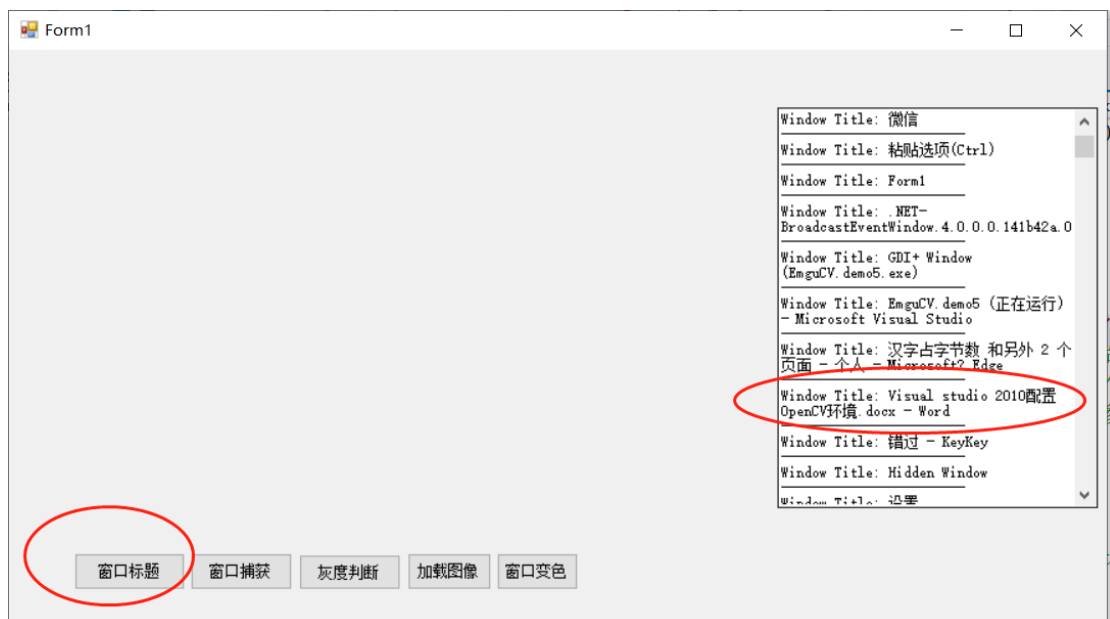
```

这段代码创建了一个点击事件，将其命名为窗口捕获，将需要捕获的窗口名称输入到程序中，即可以实现对顶层窗口的捕获，窗口设计的界面如下图所示，当捕获完成后，我们也可以使用加载图像进行查看刚刚的图像，如下图所示：



演示操作：

1、运行程序，点击窗口标题，此处我们就以 word 为例，可以看到右侧的文本框中，有相应的窗口标题，如下图所示：



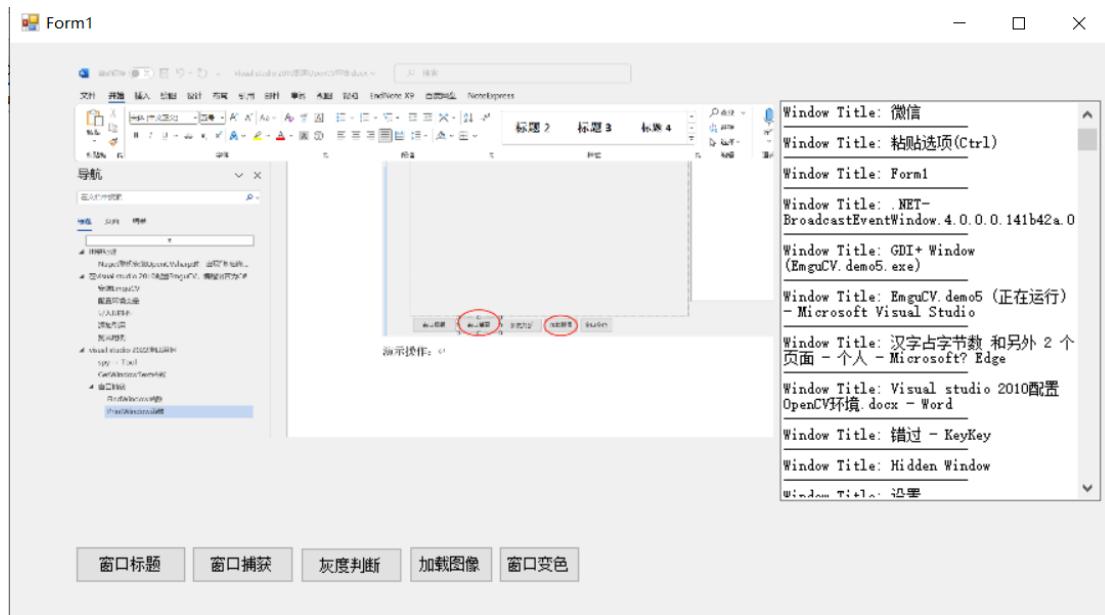
下方的代码语句进行相应的修改：

```

string targetWindowTitle = "Visual studio 2010 配置 OpenCV 环境.docx - Word";

```

2、修改后，重新运行程序，点击窗口加载图像，即可以在窗口的 imageBox 控件中看到刚刚捕获的图像，图像的加载路径，取决于程序中保存的路径，即 `bitmap.Save("C:\\Users\\dhl\\Pictures\\测试图片\\123.png")` 语句指示的路径，加载图像后如下图所示：



灰度判断

此模块主要是用于将加载的图像进行二值化处理，从而判断图像是不是全黑，作出提示，直接看下面的代码：

```
private void GrayCheck_Click(object sender, EventArgs e)
{
    Image<Gray, byte> grayImage = scaledImage.Convert<Gray, byte>();
    double thresholdValue = 20;
    // 创建一个输出图像，用于存储阈值处理后的结果
    Image<Gray, byte> thresholdedImage = new Image<Gray, byte>(grayImage.Width,
    grayImage.Height);

    // 应用阈值处理
    CvInvoke.Threshold(grayImage, thresholdedImage, thresholdValue, 255,
    ThresholdType.Binary);

    // 统计非零像素的数量
    int nonZeroPixelCount = CvInvoke.CountNonZero(thresholdedImage);
    MessageBox.Show(nonZeroPixelCount.ToString());

    // 判断是否全黑，0表示黑色，255则表示白色
    bool isAllBlack = (nonZeroPixelCount <= 1000);

    MessageBox.Show("Is the image all black? " + isAllBlack);
}
```

```
// CvInvoke.Imshow("Thresholded Image", thresholdedImage);
```

```
imageBox1.Image = thresholdedImage;
}
```

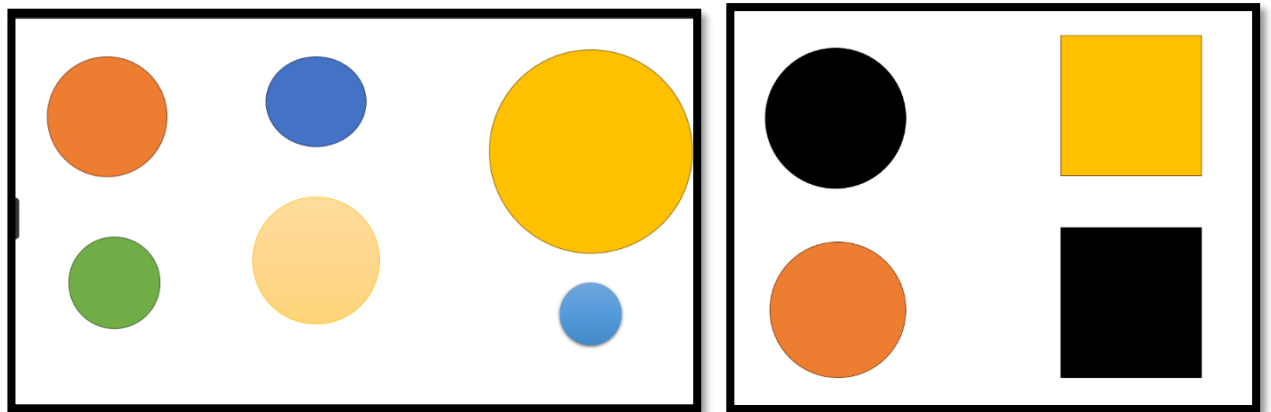
在代码中定义了一个阈值 `thresholdValue`，这个值用于后面进行二值化处理。所谓的二值化处理，是指图像的灰度值大于阈值，则将其直接置为 255，也就是白色；如果小于阈值，则将其置于 0，也就是黑色，这个值可以根据实际情况进行调整。

`CvInvoke.Threshold(grayImage, thresholdedImage, thresholdValue, 255, ThresholdType.Binary)`；这条语句也就是进行了阈值处理。

随后使用 `CvInvoke.CountNonZero(thresholdedImage)` 判断非零像素的数量，我们再次设定一个阈值，此处我们设为 1000，当非零的像素小于 1000 时，可以认为白色占比很少，相当于全黑，在程序中，我们还将非零像素数量在 `messageBox` 中进行打印提醒。

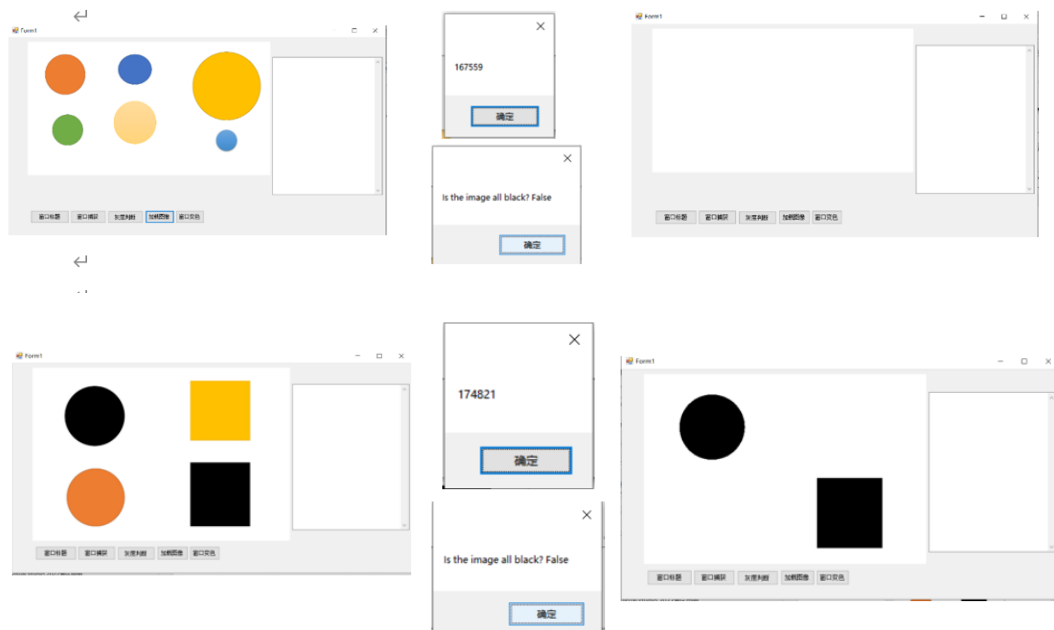
实例演示：

启动程序，加载图像，本例我们选择了两张测试图片，如下图：

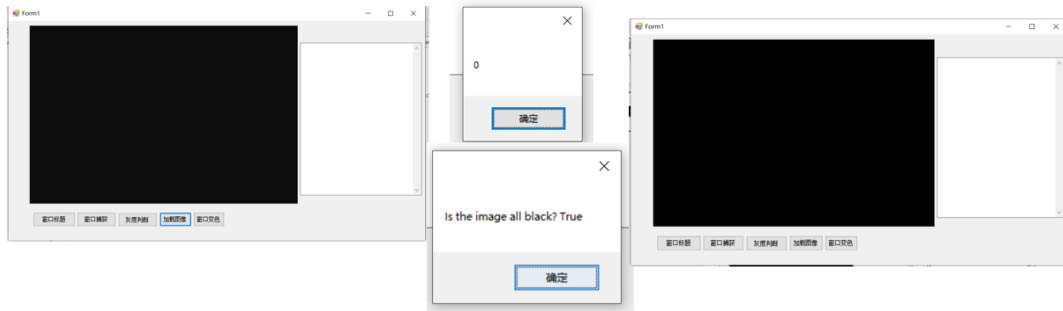


第一张图片没有黑色，第二张图片由一个圆和一个矩形为黑色，按照我们的思路，对于第一张图片，图像中灰度值没有小于 20 的像素，所以最终非零像素的个数很大。进行二值化处理后，将是一片白色。

第二张图片，则将会显示黑色的部分，其余部分全部为白色。具体的效果演示如下图：



我们还可以选取一张全黑的照片作为测试，其测试结果如下图所示：



摄像头捕获

这段是一个基本的图像捕获和处理的示例，使用 **Emgu CV** 库（一个基于 **OpenCV** 的 **.NET** 封装库）来处理图像。这里的目标是捕获视频帧，并在应用程序中显示它。

1. **ImageCapture_Click** 方法：

- 当按钮 "开启摄像头" 被点击时，首先创建一个 **VideoCapture** 对象，该对象用于捕获摄像头的视频流。
- 然后，通过订阅 **Application.Idle** 事件，在每个空闲周期获取一帧图像，并将其显示在名为 **imageBox1** 的图像框中。

2. **ProcessImage** 方法：

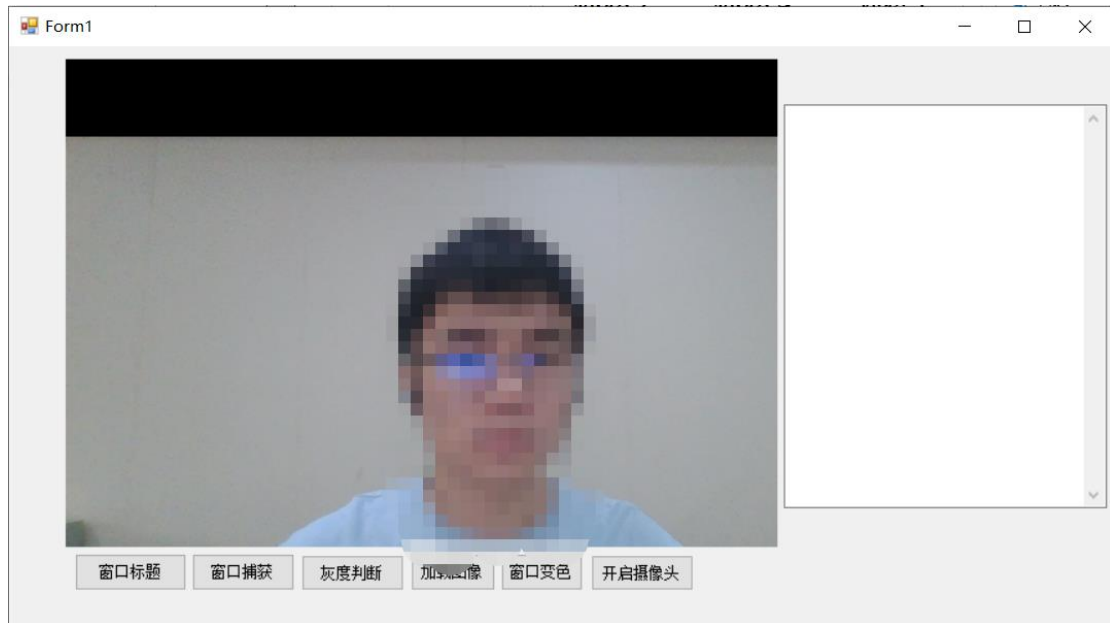
- 这是一个示例的图像处理方法，你可以在其中进行图像处理操作，比如滤波、边缘检测等。
- 本示例中，这个方法只是简单地返回输入的图像，没有进行实际的图像处理。

完整的代码展示：

```
private void ImageCapture_Click(object sender, EventArgs e)
{
    VideoCapture capture = new VideoCapture();
    //imageBox1.Image = capture.QueryFrame();
    Application.Idle += new EventHandler(delegate (object s, EventArgs ea)
    {
        imageBox1.Image = capture.QueryFrame();
    });
}

private Image<Bgr, byte> ProcessImage(Image<Bgr, byte> inputFrame)
{
    // 在这里进行图像处理，例如滤波、边缘检测等
    return inputFrame; // 返回处理后的图像
}
```

实际运行效果展示：



基础图像处理

灰度转换

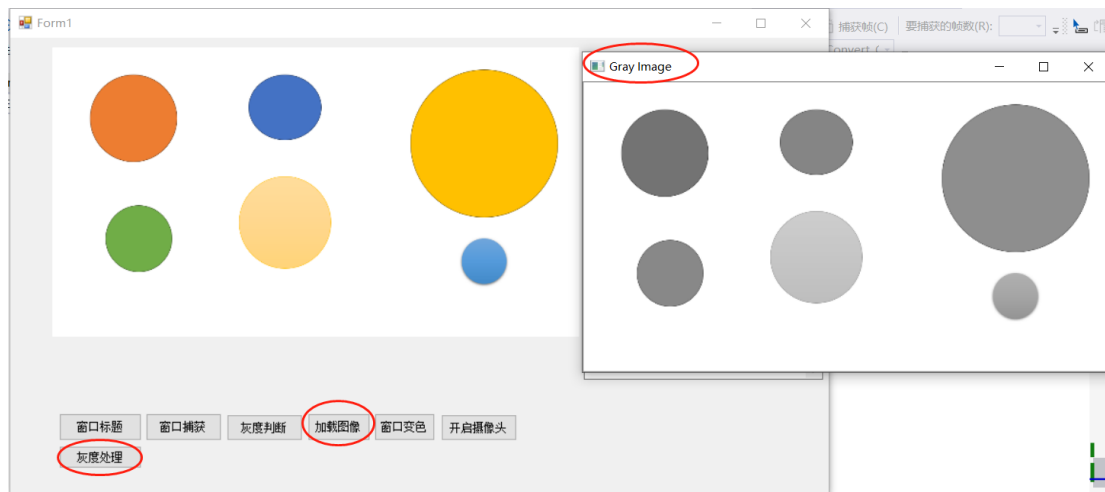
对图像进行灰度处理意味着将彩色图像转换为灰度图像，灰度图像只有一个通道，每个像素的值表示亮度，而不是颜色信息。在灰度图像中，每个像素的值通常在 0 到 255 之间，表示从黑到白的不同亮度级别。

添加点击事件，将加载进来的图像进行灰度转换，随后在另一个窗口中显示。

详细的代码如下：

```
private void GrayConvert_Click(object sender, EventArgs e)
{
    //CvInvoke.Imshow("Jinx", scaledImage);
    Mat graymat = new Mat();
    //转为灰度图像
    CvInvoke.CvtColor(scaledImage, graymat, ColorConversion.Rgb2Gray);
    CvInvoke.Imshow("Gray Image", graymat);
}
```

运行后的效果如下图所示：



canny 算子

Canny 算子是一种用于边缘检测的经典算法，旨在检测图像中的边缘或轮廓。

Canny 算子的工作原理如下：

降噪：首先，对输入图像进行高斯滤波，以去除噪声，使得边缘检测更加准确。

计算梯度：在滤波后的图像中计算每个像素的梯度值和方向。这可以帮助确定图像中的亮度变化。

非极大值抑制：在图像中找到梯度的局部最大值，以提取出细化的边缘。

双阈值处理：根据两个阈值（高阈值和低阈值）将图像中的边缘分为强边缘和弱边缘。强边缘通常是明显的边界，而弱边缘可能是一些噪声或模糊边界。

边缘连接：通过追踪强边缘的方式，将与强边缘连接的弱边缘认定为真正的边缘。

Canny 算子能够有效地检测图像中的边缘，且对噪声具有一定的抵抗能力。它在图像处理中被广泛应用于目标检测、图像分割、特征提取等任务中。

Canny 函数说明：

函数原型：

```
public static void Canny(  
    Mat image,  
    Mat edges,  
    double threshold1,  
    double threshold2,  
    int apertureSize = 3,  
    bool L2gradient = false  
);
```

参数说明：

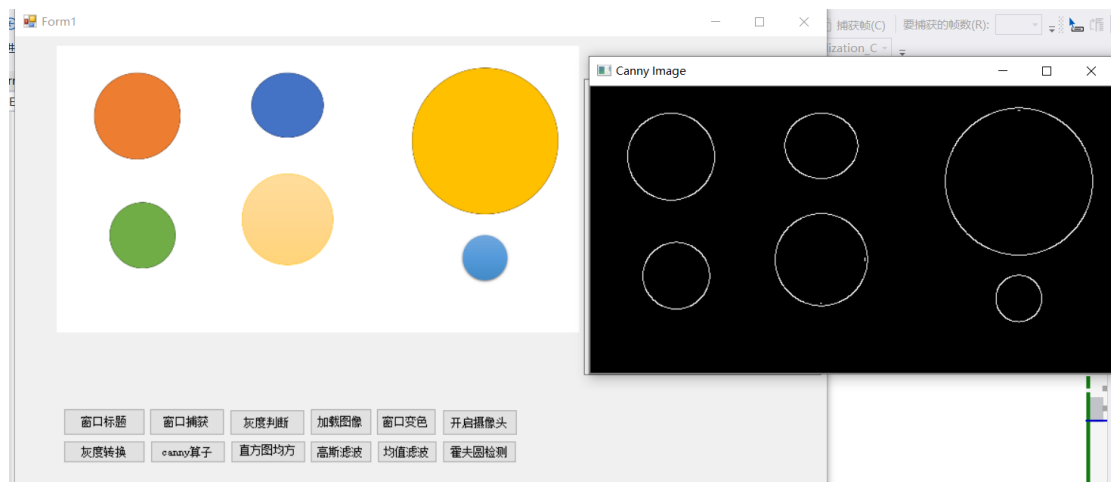
- image**: 输入图像，要求是单通道（灰度）图像。
- edges**: 输出参数，用于存储边缘检测结果的图像。通常为单通道的二值图像，边缘为白色，背景为黑色。
- threshold1**: Canny 算子的低阈值，用于确定弱边缘。

- **threshold2**: Canny 算子的高阈值，用于确定强边缘。
- **apertureSize**: Sobel 算子的孔径大小，默认为 3。
- **L2gradient**: 布尔值，是否使用 L2 范数进行梯度计算，默认为 false。

完整的代码如下：

```
private void canny_Click(object sender, EventArgs e)
{
    Mat grayImage = new Mat();
    //转为灰度图像
    CvInvoke.CvtColor(scaledImage, grayImage, ColorConversion.Rgb2Gray);
    CvInvoke.GaussianBlur(grayImage, grayImage, new Size(3, 3), 3);
    Mat cannyImg = new Mat();
    CvInvoke.Canny(grayImage, cannyImg, 20, 40);
    CvInvoke.Imshow("Canny Image", cannyImg);
}
```

函数运行后效果演示如下图：



可以观察到边缘为白色，背景为黑色，基本能提取到所有的边缘特征。

直方图均方化

直方图均衡化（Histogram Equalization）是一种用于改善图像对比度的图像处理技术。它通过重新分布图像的灰度级，使得图像的像素值在整个灰度范围内更均匀分布，从而增强图像的细节和对比度。

直方图均衡化的基本思想是将原始图像的累积直方图线性映射为均匀分布的累积直方图。这样做的效果是将原始图像中灰度级较少的区域进行拉伸，从而增强了这些区域的对比度，同时压缩了灰度级较多的区域，使得整个图像的对比度更均匀。

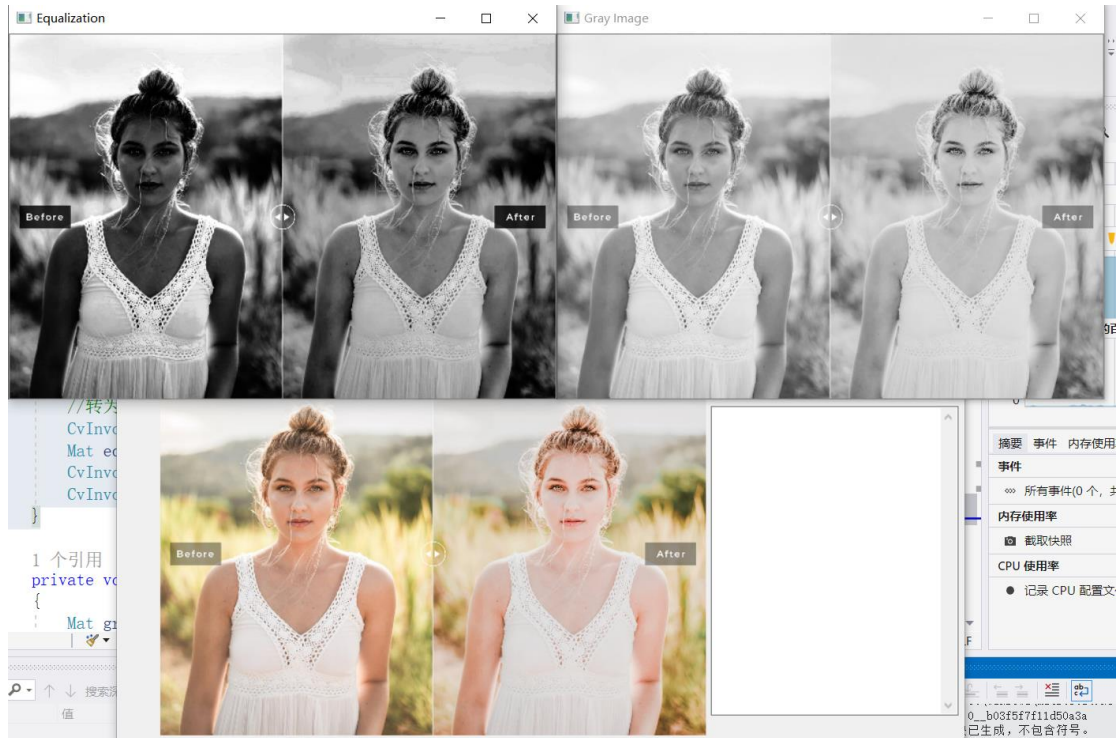
完整代码展示：

```
private void Equalization_Click(object sender, EventArgs e)
{
    Mat grayImage = new Mat();
```

```
//转为灰度图像
```

```
CvInvoke.CvtColor(scaledImage, grayImage, ColorConversion.Rgb2Gray);  
Mat equalizationImage= new Mat(scaledImage.Size, DepthType.Cv8U, 1);  
CvInvoke.EqualizeHist(grayImage, equalizationImage);  
CvInvoke.Imshow("Equalization", equalizationImage);  
}
```

运行效果如下图所示：



右上是普通的灰度图像，正下方是原图，左上方是均方化处理后的图像。

高斯滤波

高斯滤波（Gaussian Filtering）是一种图像处理中常用的平滑滤波技术，用于去除图像中的噪声和细节，以平滑图像并减少图像中的高频成分。

高斯滤波的基本思想是通过对图像中的每个像素点周围的像素进行加权平均，从而得到一个平滑的输出像素值。这种加权平均是通过一个高斯函数来定义的，距离中心像素越远的像素会被赋予较小的权重，而距离越近的像素会被赋予较大的权重。这样做的结果是，高斯滤波在平滑图像的同时，能够保留图像中的边缘和结构特征。

GaussianBlur 函数签名：

```
void GaussianBlur(  
    InputArray src,  
    OutputArray dst,  
    Size ksize,  
    double sigmaX,  
    double sigmaY = 0,  
    BorderType borderType = BorderType.Default);
```

参数说明:

- **src**: 输入图像, 可以是 **Mat** 类型。
- **dst**: 输出图像, 可以是 **Mat** 类型。
- **kernelSize**: 高斯核的大小, 用 **Size** 类型表示。通常是一个正奇数, 例如 **(3, 3)**、**(5, 5)**。
- **sigmaX**: X 方向的标准差。
- **sigmaY**: Y 方向的标准差 (可选, 默认为 0, 如果为 0, 则将其设置为与 **sigmaX** 相同)。
- **borderType**: 边界处理类型, 指定了边界扩展的方式, 默认为 **BorderType.Default**。

完整的代码展示如下:

```
private void GaussianBlur_Click(object sender, EventArgs e)
{
    Mat grayImage = new Mat();
    //转为灰度图像
    CvInvoke.CvtColor(scaledImage, grayImage, ColorConversion.Rgb2Gray);
    CvInvoke.GaussianBlur(grayImage, grayImage, new Size(3, 3), 3);
    CvInvoke.Imshow("GaussianBlur Image", grayImage);
}
```

程序运行效果展示如下:



右边是高斯滤波后的图片, 可以发现图片中的噪声得到了很大的改善。

均值滤波

均值滤波（Mean Filtering）是一种常见的图像平滑滤波方法，也被称为平均滤波。它通过对图像中的像素及其周围像素的灰度值进行平均来实现图像的平滑化。均值滤波可以减少图像中的噪声，并且能够模糊图像中的细节，适用于一些降噪和模糊化的应用场景。

均值滤波的过程如下：

对图像中的每个像素，取其周围邻域区域（通常是一个正方形或矩形窗口）内所有像素的灰度值。

对这些灰度值进行平均，得到一个新的像素值，然后将这个新值赋给中心像素。

均值滤波的核心思想是用局部区域的平均值来代替中心像素的值，从而实现平滑效果。然而，均值滤波也有其局限性，特别是在处理含有较多噪声或边缘信息的图像时，它可能会导致图像细节的丢失。

Blur 函数原型如下：CvInvoke.Blur(Mat src, Mat dst, Size ksize, Point anchor = null, BorderType borderType = BorderType.Default);

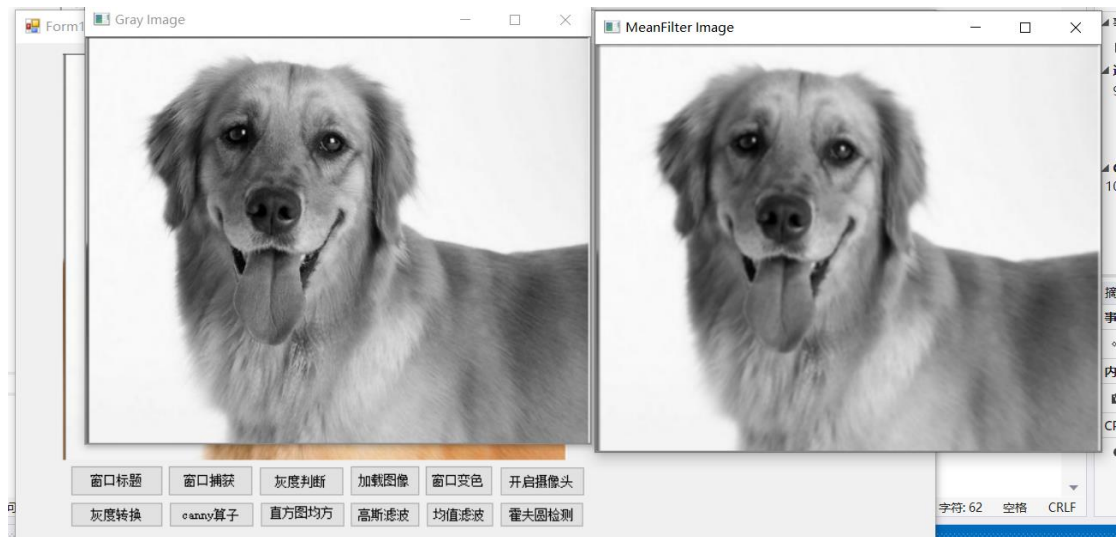
参数说明：

- **src**: 输入图像（源图像），Mat 类型。
- **dst**: 输出图像（目标图像），Mat 类型，用于存储滤波结果。
- **ksize**: 滤波核大小，即局部邻域区域的大小。Size 类型，通常使用 (width, height) 格式进行指定。
- **anchor**: 滤波核的锚点，默认为 null，表示核的中心位置。
- **borderType**: 边界扩展类型，用于处理滤波核在图像边缘区域的情况。

完整的代码展示：

```
private void MeanFilter_Click(object sender, EventArgs e)
{
    Mat grayImage = new Mat();
    //转为灰度图像
    CvInvoke.CvtColor(scaledImage, grayImage, ColorConversion.Rgb2Gray);
    CvInvoke.Blur(grayImage, grayImage, new Size(1, 1), new Point(-1, -1));
    CvInvoke.Imshow("MeanFilter Image", grayImage);
}
```

运行后的展示：

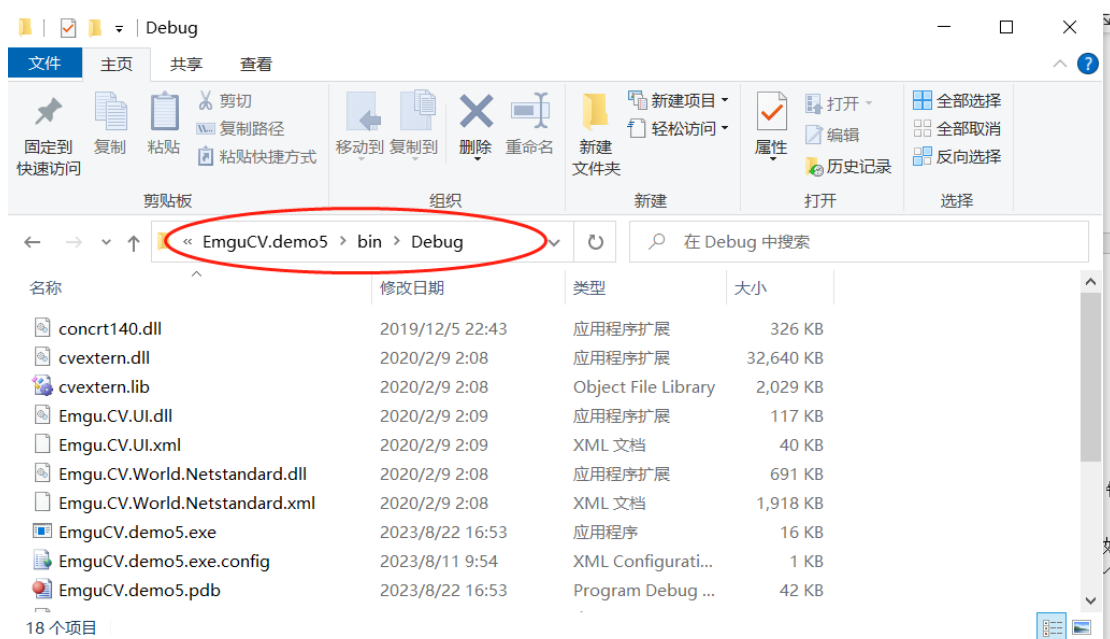
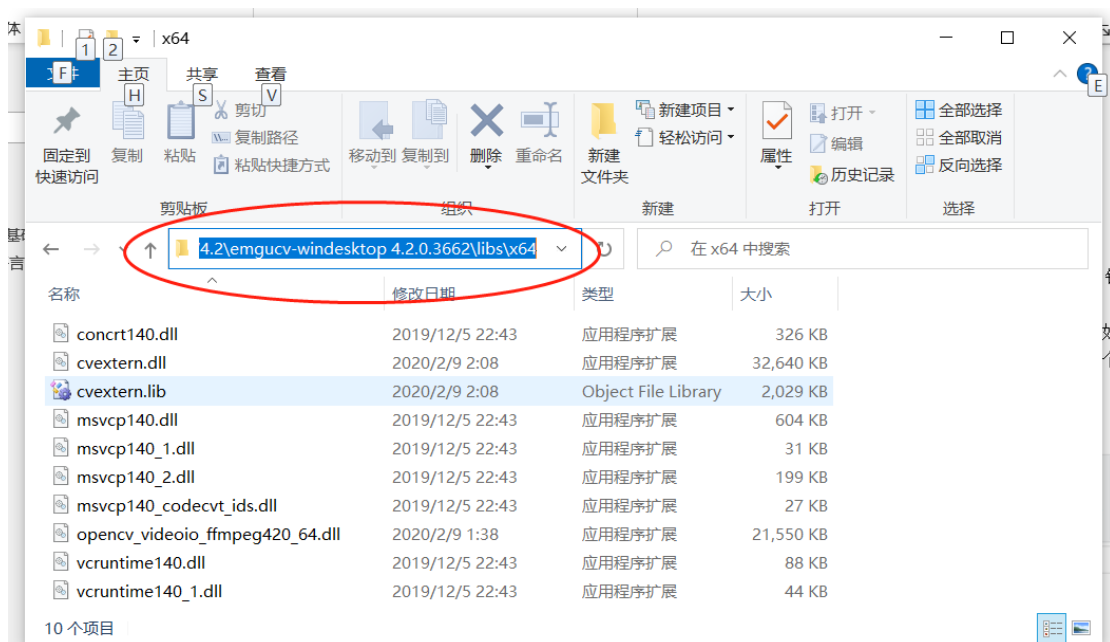


右边为滤波后的图像，左边为原始灰度图像，可以发现均值滤波后的图像变得稍微模糊一点，这是因为均值滤波导致了细节的丢失。

常见错误处理

缺少引用或者找不到相关的定义

在编程过程中，如果出现某个关键字或方法找不到相关的定义（这里主要针对的 **emguCV** 中的一些方法），可以检查是否缺少了相关的 **dll** 文件（在进行次步骤时，一定要确保前面的步骤已经完成，包括安装 **emguCV**，配置环境变量，添加引用，详细步骤可以参考前面的步骤）。此时我们找到 **emguCV** 的安装目录，找到 **libs/x64**（比如我的目录为 **D:\programme\emguCV4.2\emgucv-windesktop 4.2.0.3662\libs\x64**），将这个文件夹中的所有文件复制到程序中的 **bin\Debug** 文件下。



缺少调用函数的声明

可以注意到，我们的整个代码都有很多的这种类似代码：

```
[DllImport("user32.dll")]
private static extern int GetWindowText(IntPtr hWnd, System.Text.StringBuilder
lpString, int nMaxCount);
```

这两行的代码主要是对其他库的函数进行一个声明，如果没有这两条语句，则编译器不知道去哪里寻找代码的出处，比如上面的代码，`[DllImport("user32.dll")]`用于告诉编译器要从名为"user32.dll"的 Windows 动态链接库（也称为动态链接库，DLL）中导入函数。`user32.dll`是包含了许多与用户界面交互相关的 Windows API 函数的标准库。

`private static extern int GetWindowText(IntPtr hWnd, System.Text.StringBuilder lpString, int nMaxCount);`：这是一个函数声明，表示将名为 `GetWindowText` 的函数从 `user32.dll` 导入到 C# 代码中。

如下所示，当我们将上面的两行代码注释掉后，`GetWindowText` 函数会报错，显示“当前上下文不存在名称 `GetWindowText`”。

