

Sprawozdanie

Zbieranie i Analiza Danych

Labolatorium nr 1 – Zbieranie danych w kamery



Oskar Kwidziński

S156013

Fizyka Techniczna

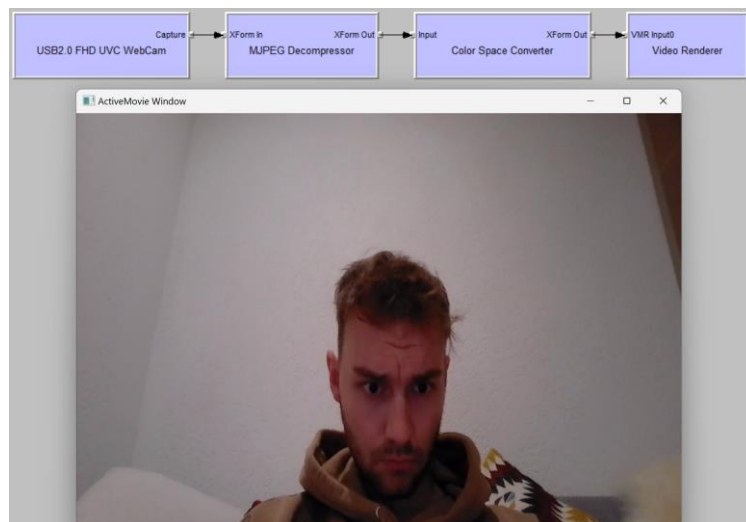
Semestr zimowy 2022/2023

Wstęp

Wykonano zadanie, w którym należało stworzyć program pozwalający na przechwytywanie obrazu z kamery. Rozwiązanie oparto o bibliotekę directshownet. Zmodyfikowany kod pozwala na zapisywanie ramek na dysku twardym w postaci plików o rozszerzeniu .jpg oraz .bmp.

Zadanie 1, 2

Zestawiono testowy graf w programie graphedit.



W podanym grafie skonfigurowano połączenie pomiędzy kamerą laptopa oraz video rendererem. Pomiędzy obydwoma blokami umieszczono dodatkowo dekompresor, niezbędny do rozkodowania ramek MJPEG oraz Color Space Converter konwertujący obraz na rzeczywiste barwy.

Zadanie 3, 4

Zmieniono implementację zdarzenia `toolBar_ButtonClick (Grab)`:

```
private void toolBar_ButtonClick(Object sender, System.Windows.F
{
    Trace.WriteLine("!!BTN: toolBar_ButtonClick");

    int hr;
    if (sampGrabber == null)
        return;

    if (e.Button == toolBarBtnGrab)
    {
        //Trace.WriteLine( "!!BTN: toolBarBtnGrab" );

        if (!capturing)
        {
            if (savedArray == null)
            {
                int size = videoInfoHeader.BmiHeader.ImageSize;
                if ((size < 1000) || (size > 16000000))
                    return;
                savedArray = new byte[size + 64000];
            }

            /*toolBarBtnSave.Enabled = false;
            Image old = pictureBox.Image;
            pictureBox.Image = null;
            if( old != null )
                old.Dispose();

            toolBarBtnGrab.Enabled = false;
            captured = false;*/
            toolBarBtnGrab.Text = "End capture";
            capturing = true;
            hr = sampGrabber.SetCallback(this, 1);
        }
        else
        {
            toolBarBtnGrab.Text = "Start capture";
            capturing = false;
            hr = sampGrabber.SetCallback(null, 0);
        }
    }
}
```

wywołującego odpowiednią implementację interfejsu ISampleGrabber i zainicjowanie bufora. Funkcja przechwytyjąca - ISampleGrabberCB.BufferCB, do której trafiają ramki obrazu, została rozbudowana o pomiar fps:

```
int ISampleGrabberCB.BufferCB(double SampleTime, IntPtr pBuffer, int BufferLen)
{
    //implementacja pseudokodu do pomiaru fps
    bool upd = false;
    int updFPS = 0;
    if (!czas.IsRunning)
    {
        czas.Start();
    } else
    {
        long ile_czasu_uplynelo = czas.ElapsedMilliseconds - czas_poprzedni;

        FPS++;
        if (ile_czasu_uplynelo > 1000)
        {
            czas_poprzedni = czas.ElapsedMilliseconds;
            upd = true;
            Trace.WriteLine(FPS);
            updFPS = FPS;
            FPS = 0;
        }
    }
    if (last_cap + 2 > czas.ElapsedMilliseconds) //blokujemy do 333 fps
    {
        return 0;
    }
    last_cap = czas.ElapsedMilliseconds;
}
```

zaś samo złapanie ramki kopiowane do bufora newBytes:

```
captured = true;
bufferedSize = BufferLen;
//Trace.WriteLine( "!!CB: ISampleGrabberCB.BufferCB
if ((pBuffer != IntPtr.Zero) && (BufferLen > 1000))
{
    int size = videoInfoHeader.BmiHeader.ImageSize;
    byte[] newBytes = new byte[size + 64000];
    Marshal.Copy(pBuffer, newBytes, 0, BufferLen);
    this.OnCaptureDone(newBytes);
    if (upd)
    {
        this.BeginInvoke(new CaptureDone(delegate
        {
            fps_counter.Text = "FPS: " + updFPS;
            //this.OnCaptureDone(newBytes);
        }));
    }
}
```

Następnie wywoływane jest zdarzenie OnCaptureDone. W zdarzeniu tym dokonywane jest obliczenie rozmiaru bitmapy, tworzenie bitmapy z bufora i zapisywanie do pliku:

```

void OnCaptureDone(byte[] bytes)
{
    //fps_counter.Text = "FPS: " + FPS;
    Trace.WriteLine("!!DLG: OnCaptureDone");
    if (!capturing)
    {
        sampGrabber.SetCallback(null, 0);
        return;
    }
    try {
        //toolBarBtnGrab.Enabled = true;
        int hr;
        if (sampGrabber == null)
            return;
        //hr = sampGrabber.SetCallback( null, 0 );

        int w = videoInfoHeader.BmiHeader.Width;
        int h = videoInfoHeader.BmiHeader.Height;
        if ((w & 0x03) != 0) || (w < 32) || (w > 4096) || (h < 32) || (h > 4096))
            return;
        int stride = w * 3;

        GCHandle handle = GCHandle.Alloc(bytes, GCHandleType.Pinned);
        long scan0 = (long)handle.AddrOfPinnedObject(); //zmiana z inta na longa aby
        scan0 += (h - 1) * stride;
        Bitmap b = new Bitmap(w, h, -stride, PixelFormat.Format24bppRgb, (IntPtr)scan0);

        b.Save($"frame{frameCount}.jpg");
        b.Save($"frame{frameCount++}.bmp");
        Trace.WriteLine($"Saved frame {frameCount - 1}");

        handle.Free();
        /*savedArray = null;
        Image old = pictureBox.Image;
        pictureBox.Image = b;
        if (old != null)
            old.Dispose();
        toolBarBtnSave.Enabled = true;*/
    }
}

```

Ponadto, zmieniono funkcję SetupGraph, tak aby pozwalała z poziomu okna ustawić pożądaną rozdzielczość:

```

bool SetupGraph()
{
    int hr;
    try
    {
        Guid streamConfigId = typeof(IAMStreamConfig).GUID;
        object videoConfigObj;

        hr = capGraph.FindInterface(PinCategory.Capture,
            MediaType.Video,
            capFilter, streamConfigId, out videoConfigObj);

        videoConfig = videoConfigObj as IAMStreamConfig;
        videoConfig.GetNumberOfCapabilities(out int pCount, out int pSize);
        mtypes = new AMMediaType[pCount/2];

        IntPtr cc = Marshal.AllocCoTaskMem(pSize);
        for (int x = 0; x < pCount; x+=2) {

            videoConfig.GetStreamCaps(x, out AMMediaType ppmt, cc);
            mtypes[x/2] = ppmt;
            if (!resListInit)
            {
                VideoInfoHeader tvh = (VideoInfoHeader)Marshal.PtrToStructure(ppmt.formatPtr, typeof(VideoInfoHeader));
                resolutionList.Items.Add(tvh.BmiHeader.Width + "x" + tvh.BmiHeader.Height);
            }

            //Trace.WriteLine(mtypes[x / 2].BmiHeader.Width + "x" + mtypes[x / 2].BmiHeader.Height);
        }

        videoConfig.SetFormat(mtypes[videoMode]);
    }
}

```

Podsumowanie

Ramki udało się przechwytywać i zapisywać na dysk w czasie rzeczywistym:

frame0.bmp	1/12/2023 9:25 PM	BMP File	50 KB
frame0.jpg	1/12/2023 9:25 PM	JPG File	50 KB
frame1.bmp	1/12/2023 9:25 PM	BMP File	50 KB
frame1.jpg	1/12/2023 9:25 PM	JPG File	50 KB
frame2.bmp	1/12/2023 9:25 PM	BMP File	49 KB
frame2.jpg	1/12/2023 9:25 PM	JPG File	49 KB
frame3.bmp	1/12/2023 9:25 PM	BMP File	50 KB
frame3.jpg	1/12/2023 9:25 PM	JPG File	50 KB

Sample Grabber for .NET

Tune

End capture

Save

176x144

FPS: 20

Przeprowadzono analizę ilości fps, którą udało się przechwycić dla każdej z rozdzielczości:

Sample Grabber for .NET


Tune

End capture

Save

1920x1080

FPS: 5



Sample Grabber for .NET


Tune

End capture

Save

800x600

FPS: 17



Sample Grabber for .NET


Tune

End capture

Save

652x388

FPS: 20



Sample Grabber for .NET


Tune

End capture

Save

176x144

FPS: 20



Nie udało się zwiększyć ilości fps ponad wyniki wskazane powyżej. Największą wydajność osiągnięto dla niskich rozdzielczości. Nie odnotowano zmian fps przy testach wydajności kamery np. zbliżania do źródła światła lub przy wyłączonym świetle (poza chwilowymi w trakcie zmian oświetlenia). Przypuszcza się, że powodem niskiej ilości fps jest niewydajność dysku twardego, na którym były zapisywane pliki.