码客说

YUV和RGB存储规则

2023-06-04 | yuv

YUV和RGB存储规则

前言

我们开发平常图片的数据都是RGB，但是涉及视频相关的都会使用到YUV格式，为什么呢?

- YUV是电视信号的格式，为了同时兼容黑白和彩色，只有Y就是黑白，加上UV就是彩色。
- YUV采样可以大大降低传输数据的大小，以YUV420为例就是RGB的一半。

换算

1byte就是1个字节,等于8位(bits)。

常用的值

```
2^24=16777216
2^16=65536
2^8=256
2^6=64
2^5=32
```

RGB/BGR

每一个点都是由三个byte组成，分别存储R、G、B，值范围是[0-255]。

需要注意的是在C#中我们获取到的是按BGR排序的。

如果要包含透明通道，就有RGBA、BGRA、ARGB或者ABGR这四种方式，所以要注意我们所需的格式。

图像的格式RGB565,RGB555,RGB888,BMP24的区别

- RGB565:16位格式,5位红色,6位绿色,5位蓝色,颜色深度较浅,文件大小较小
- RGB555:16位格式,5位红色,5位绿色,5位蓝色,最后1位未使用,颜色深度比RGB565略浅
- RGB888:24位格式,8位红色,8位绿色,8位蓝色,真彩色格式,颜色深度高,文件大小大
- BMP24:也是24位格式,和RGB888一致,真彩色,文件较大

YUV

YUV的值是怎么来的呢?

直接给公式 ：

（请不要使用其他博客中的浮点数类型的公式，会严重影响精度）

```
y = (( 66 * r + 129 * g + 25  * b + 128) >> 8) + 16  ;
u = ((-38 * r - 74  * g + 112 * b + 128) >> 8) + 128 ;
v = ((112 * r - 94  * g - 18  * b + 128) >> 8) + 128 ;
```

这时大家可能就有疑问了，我们明明已经有了RGB可以表示这个像素点了，为什么还需要再使用YUV来进行表示。

说白一点就是RGB三个分量一个都不可少才能表示出一个像素点。

而YUV可以通过不同的采样方式来减少一些U、V分量，从而减小所需的存储空间。而恢复为RGB的时候可以几个Y分量共用U、V分量来恢复为RGB。

这样全采样的YUV其实跟RGB所需存储空间一样了，而这种采样方式就是 **YUV 4:4:4** 。

采样方式

YUV 4:4:4

这种方式也就是上文所说的，YUV分量全部进行采样

YUV 4:2:2

在所有像素上，Y分量全部采样。

在同行的像素上， **U** 和 **V** 分量分别 **交替** 进行采样；

YUV 4:2:0【重点】

在所有像素上，Y分量全部采样。

在（偶数行）， **U** 分量 **间隔** 进行采样，而不采样V分量。

在（奇数行）， **V** 分量 **间隔** 进行采样，而不采样U分量。

以上面的8个像素为例，那么我们采集到的数组长度则分别为：

YUV 4:4:4

8 + 8 + 8 长度为24

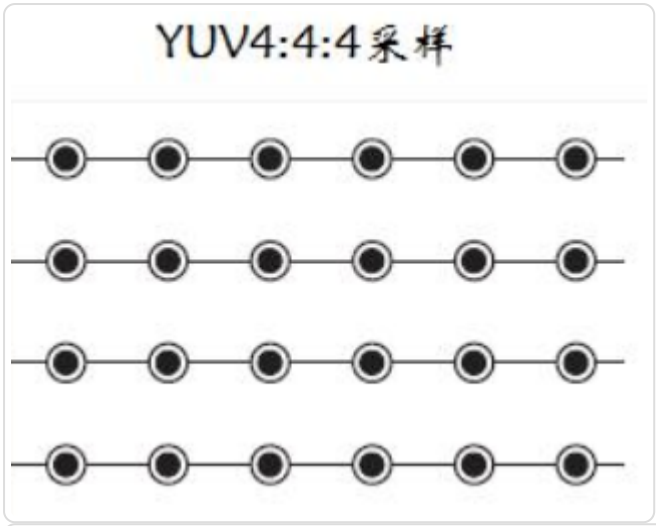
YUV 4:2:2

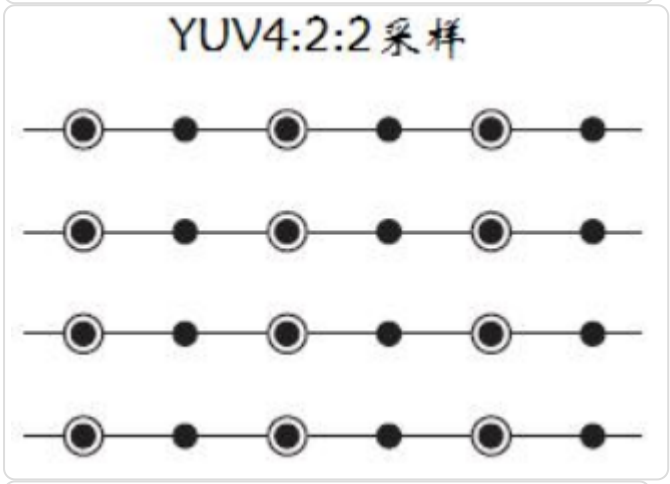
8 + 4 + 4 长度为16，是第一种的 三分之二

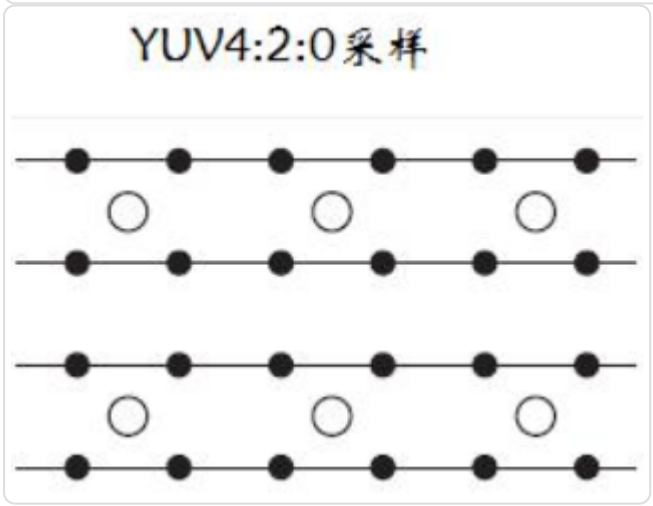
YUV 4:2:0

8 + 2 + 2 长度为12，是第一种的 二分之一

所以使用420的采样方式，所需的存储空间会大大减小。







存储方式

我们4x2的图片为例，共8个像素，使用YUV420存储的话，对应的数组就会是这样：

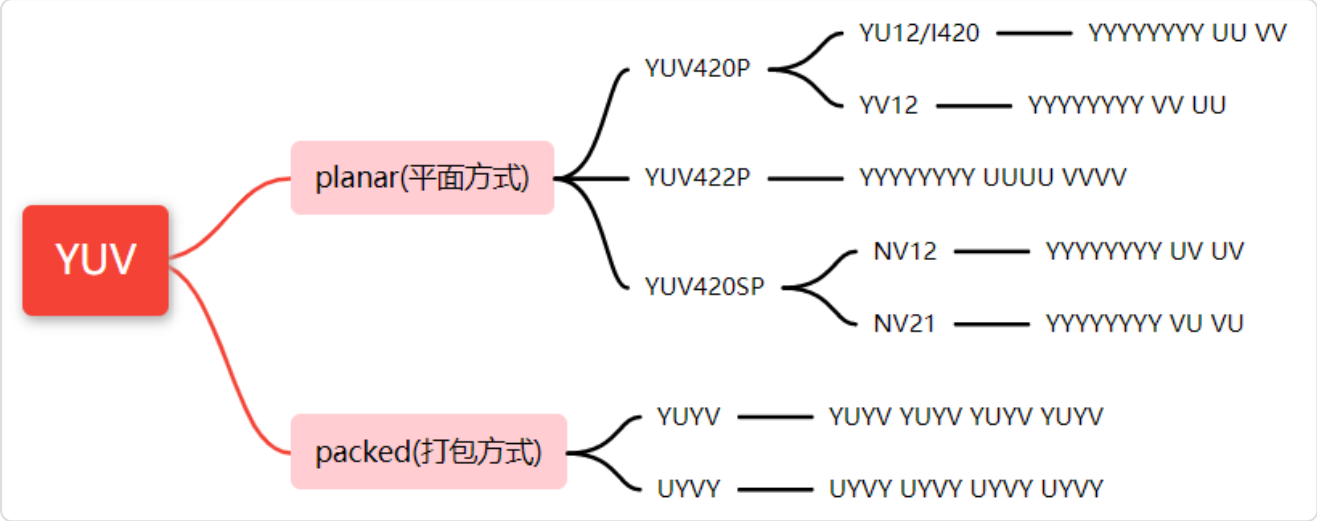
```
Y数组: [Y, Y, Y, Y, Y, Y, Y, Y]
U数组: [U, U]
V数组: [V, V]
```

存储方式分为

- planar（平面方式） 现存Y，在存UV

- packed（打包方式）YUV交替存储

整体



平面模式

顺序可以是 先存Y，再存U，最后存V 。也可以是 先存Y，再存V，再存U 。我们这里把前者称为 **YU的存储方式**，把后者称为 **YV的存储方式**

420采样方式 + YU存储方式 = **YU12**（又叫 **I420** ）

```
YYYYYYYY UU VV
```

420采样方式 + YV存储方式 = **YV12**

```
YYYYYYYY VV UU
```

这两种存储格式呢，又统称为 **YUV420P** 格式。

UV交替存储

UV交替存储的，还有VU交替存储的，那么我们就把前者称为UV存储，把后者称为VU存储，那么总结来了：

420采样方式 + UV存储方式 = NV12

```
YYYYYYYY UV UV
```

420采样方式 + VU存储方式 = NV21

```
YYYYYYYY VU VU
```

上面这两种特殊的平面方式呢，又叫 Semi-Splanar ，所以以上两种格式又称为 YUV420SP 格式。

422采样也可以使用平面存储的方式，如下：

```
Y数组: [Y, Y, Y, Y, Y, Y, Y, Y]
U数组: [U, U, U, U]
V数组: [V, V, V, V]
```

存储方式

```
YYYYYYYY UUUU VVVV
```

422采样方式 + 平面存储方式 = YUV422P(属于YUV422)

打包方式

一般我们使用422采样方式的时候会采用这种存储方式，这种方式就不像上面那种那么直白了，先用数组表示吧，注意是422采样模式，所以U、V数组长度也变化了

```
YUYV YUYV YUYV YUYV
```

如上所示，因为YUV的比例是2:1:1 ，所以取两个Y元素就需要分别取一个U和V元素，后面同理。所以根据上面这种格式：
422采样方式 + YUYV打包存储方式 = YUYV

```
YUYV YUYV YUYV YUYV
```

422采样方式 + UYVY打包存储方式 = UYVY

```
UYVY UYVY UYVY UYVY
```

图片数据

```
using (var bmp = new Bitmap(image))
{
    var data = bmp.LockBits(
        new Rectangle(Point.Empty, image.Size),
        ImageLockMode.ReadWrite,
        PixelFormat.Format24bppRgb
    );

    bmp.UnlockBits(data);
}
```

C#封装的libyuv

https://github.com/jlennox/LibYuvSharp

https://www.nuget.org/packages/Lennox.LibYuvSharp/1.1.2?_src=template

安装

```
Install-Package Lennox.LibYuvSharp -Version 1.1.2
```

注意

> 要求.Net Framework版本net461及以上。
>
> 改库是64位的。

官网

https://www.nuget.org/packages/Lennox.LibYuvSharp/1.1.2?_src=template

推荐程序使用.Net Framework 4.7.2版本

加载BMP

```
private void LoadBmp()
{
    using (var image = Image.FromFile("test.bmp"))
    using (var bmp = new Bitmap(image))
    {
        var data = bmp.LockBits
        (
            new Rectangle(Point.Empty, image.Size),
            ImageLockMode.ReadOnly,
            PixelFormat.Format24bppRgb
        );
        var rgbStride = image.Width * 3;
        var argbStride = image.Width * 4;
        var original = new byte[rgbStride * image.Height];
        var destRgb = new byte[rgbStride * image.Height];
        var destArgb = new byte[argbStride * image.Height];
        unsafe
        {
            fixed (byte* originalPtr = original)
            fixed (byte* destArgbPtr = destArgb)
            fixed (byte* destRgbPtr = destRgb)
            {
                // Put the original 24bit RGB pixel data into an array for
                // later validation.
                Buffer.MemoryCopy
                (
                    (void*)data.Scan0,
```

```
                    originalPtr,
                    original.Length,
                    original.Length
                );

                // Convert the source 24bit RGB pixel data to 32bit ARGB.
                // This conversion is lossless.
                LibYuv.RGB24ToARGB
                (
                    (byte*)data.Scan0,
                    rgbStride,
                    destArgbPtr,
                    argbStride,
                    image.Width,
                    image.Height
                );

                // Convert the newly created 32bit ARGB back to the original
                // 24bit RGB. This conversion is lossless.
                LibYuv.ARGBToRGB24
                (
                    destArgbPtr,
                    argbStride,
                    destRgbPtr,
                    rgbStride,
                    image.Width,
                    image.Height
                );
            }
        }
        bmp.UnlockBits(data);
    }
}
```

Image加载I420数据

I420 => RGB24 => Bitmap => BitmapImage => 加载

I420转RGB24

这里使用了libyuv库

```
int width = obj.Width;
int height = obj.Height;
var original = new byte[width * 3 * height];
unsafe
{
    fixed (byte* originalPtr = original)
```

```
    {
        LibYuv.I420ToRGB24
        (
            (byte*)obj.DataY.ToPointer(),
            obj.StrideY,
            (byte*)obj.DataU.ToPointer(),
            obj.StrideU,
            (byte*)obj.DataV.ToPointer(),
            obj.StrideV,
            originalPtr,
            obj.Width * 3,
            obj.Width,
            obj.Height
        );
    }
}
```

RGB24转Bitmap

```
public static Bitmap RgbToBitmap
(
    byte[] rgbData,
    int width,
    int height
)
{
    Bitmap bitmap = new Bitmap(width, height);
    BitmapData bitmapData = bitmap.LockBits
    (
        new Rectangle
        (
            0,
            0,
            width,
            height
        ),
        ImageLockMode.WriteOnly,
        PixelFormat.Format24bppRgb
    );
    IntPtr ptr = bitmapData.Scan0;
    Marshal.Copy
    (
        rgbData,
        0,
        ptr,
        rgbData.Length
    );
    bitmap.UnlockBits(bitmapData);
```

```
30      return bitmap;
31  }
```

Image加载Bitmap

```
1   Bitmap bitmap = ...;
2   BitmapImage bitmapImage = new BitmapImage();
3   using (MemoryStream memory = new MemoryStream())
4   {
5       bitmap.Save(memory, ImageFormat.Png);
6       memory.Position = 0;
7       bitmapImage.BeginInit();
8       bitmapImage.StreamSource = memory;
9       bitmapImage.CacheOption = BitmapCacheOption.OnLoad;
10      bitmapImage.EndInit();
11  }
12
13  this.MyImg.Source = bitmapImage;
14  bitmap.Dispose();
```

加载图片转RGB/ARGB

```
1   private void LoadBmp()
2   {
3       using (var image = Image.FromFile("test.bmp"))
4       using (var bmp = new Bitmap(image))
5       {
6           var data = bmp.LockBits
7           (
8               new Rectangle(Point.Empty, image.Size),
9               ImageLockMode.ReadOnly,
10              PixelFormat.Format24bppRgb
11          );
12          var rgbStride = image.Width * 3;
13          var argbStride = image.Width * 4;
14          var original = new byte[rgbStride * image.Height];
15          var destRgb = new byte[rgbStride * image.Height];
16          var destArgb = new byte[argbStride * image.Height];
17          unsafe
18          {
19              fixed (byte* originalPtr = original)
20              fixed (byte* destArgbPtr = destArgb)
21              fixed (byte* destRgbPtr = destRgb)
22              {
23                  // Put the original 24bit RGB pixel data into an array for
24                  // later validation.
25                  Buffer.MemoryCopy
```

```
                (
                    (void*)data.Scan0,
                    originalPtr,
                    original.Length,
                    original.Length
                );

                // Convert the source 24bit RGB pixel data to 32bit ARGB.
                // This conversion is lossless.
                LibYuv.RGB24ToARGB
                (
                    (byte*)data.Scan0,
                    rgbStride,
                    destArgbPtr,
                    argbStride,
                    image.Width,
                    image.Height
                );

                // Convert the newly created 32bit ARGB back to the original
                // 24bit RGB. This conversion is lossless.
                LibYuv.ARGBToRGB24
                (
                    destArgbPtr,
                    argbStride,
                    destRgbPtr,
                    rgbStride,
                    image.Width,
                    image.Height
                );
            }
        }
        bmp.UnlockBits(data);
    }
}
```

WWebrtcSharp回显

```
var videoSource = new FrameVideoSource();
videoSource.Frame += Source_Frame;

private void Source_Frame(VideoFrame obj)
{
    int width = obj.Width;
    int height = obj.Height;
    var original = new byte[width * 3 * height];
    unsafe
    {
```

```
            fixed (byte* originalPtr = original)
            {
                LibYuv.I420ToRGB24
                (
                    (byte*)obj.DataY.ToPointer(),
                    obj.StrideY,
                    (byte*)obj.DataU.ToPointer(),
                    obj.StrideU,
                    (byte*)obj.DataV.ToPointer(),
                    obj.StrideV,
                    originalPtr,
                    obj.Width * 3,
                    obj.Width,
                    obj.Height
                );
            }
        }
        var bitmap = ZScreenUtils.RgbToBitmap
        (
            original,
            width,
            height
        );
        this.Dispatcher.Invoke
        (
            () =>
            {
                BitmapImage bitmapImage = new BitmapImage();
                using (MemoryStream memory = new MemoryStream())
                {
                    bitmap.Save(memory, ImageFormat.Png);
                    memory.Position = 0;
                    bitmapImage.BeginInit();
                    bitmapImage.StreamSource = memory;
                    bitmapImage.CacheOption = BitmapCacheOption.OnLoad;
                    bitmapImage.EndInit();
                }
                this.MyImg.Source = bitmapImage;
            },
            DispatcherPriority.Background
        );
        bitmap.Dispose();
    }
```

yuv

Typora使用Mermaid绘制各种图

WPF桌面端开发–音视频录制、获取缩略图(使用OpenCvSharp)