基于 SSE2的 YUV与 RGB色彩空间转换

刘云鹏[1),2)] 王树东[1)]
[1)](兰州理工大学, 兰州 730050) [2)](69015部队, 乌鲁木齐 830031)

摘 要 视频处理中需要完成从 YUV色彩空间到 RGB色彩空间的转换。通过分析 YUV格式与 RGB格式间的转换算法, 提出使用整形计算替代浮点运算, 利用整除 256对应右移八位操作, 提高运算速度。结合使用 Intel单指令多数据扩展指令集 SSE2技术进行算法优化后, 显著提高格式转换运算的效率。实验结果表明, 采用此技术可以提高 25倍以上的运算速度, 在视频处理中具有很好的应用性。

关键词 色彩空间转换 YUV RGB SSE2
中图法分类号: TP317. 4 **文献标志码**: A **文章编号**: 1006-8961(2009)01-0045-05

Conversion of Color Space Between YUV and RGB on SSE2 Technique

LIU Yun-lin[1),2)], WANG Shu-dong[1)]
[1)](*College of Petrochemical Engineering, Lanzhou University of Technology, Lanzhou* 730050)
[2)](*PLA* 69015 *Unit, Xijiang Umuqi* 830031)

Abstract The conversion should be completed from the YUV color space to RGB color space in video processing. In this paper, by analyzing the YUV format and RGB format conversion algorithm, floating-point operations is replaced by integer operations and divisible by 256 corresponds to eight right shift operation to increase the speed of calculation. Combine intel single-instruction, multiple-data extension SSE2 instruction set technology optimization algorithm, the efficiency of the different digital video format conversion is significantly improved. Experiments prove that SSE2 technology can increase by 25 times the computational speed and has a good application in the video processing.

Keywords Color Space Conversion, YUV, RGB, SSE2

0 引 言

数字视频处理涉及色彩空间模型, 用来描述各种各样的颜色。对于数字视频领域来说, 我们经常接触的色彩空间的概念, 主要是 RGB和 YUV这两种。RGB是通过与亮度有关的红色、绿色和蓝色的组合来描述颜色, 而 YUV则是按照亮度, 色差的原理来描述颜色。

在数字视频中, CIF (common intermediate format)格式图像基于 YUV色彩空间, 而在一些应用中, 需要 RGB色彩空间的视频帧, 因此, 需要完成从 YUV色彩空间到 RGB色彩空间的转换。而由于数字视频的数据量非常巨大, 不同数字视频格式的转换就需要巨大的运算量, 造成数字视频实时处理不能满足要求。本文给出如何使用数据流单指令多数据扩展指令 2(SSE2)技术实现常用的数字视频格式 YUV411/YUV420与 RGB格式之间的快速转换方法。

1 YUV相关色彩空间模型

1.1 YUV

YUV是被欧洲电视系统所采用的一种颜色编

收稿日期: 2008-09-02; 改回日期: 2008-10-27
第一作者简介: 刘云鹏(1977—)男, 工程师。兰州理工大学电子通信专业硕士研究生。主要研究方向为多媒体信息处理及视频通信。E-mail: liskman@263.net

码方法。YUV主要用于优化彩色视频信号的传输,使其向后兼容老式黑白电视。与RGB视频信号传输相比,它最大的优点在于只需占用极少的带宽(RGB要求3个独立的视频信号同时传输)。其中“Y”表示明亮度(Luminance或Luma),也就是灰阶值;而“U”和“V”表示的则是色度(Chrominance或Chroma),作用是描述影像色彩及饱和度,用于指定像素的颜色。“亮度”是通过将RGB信号的特定部分叠加到一起建立。“色度”定义了颜色的色调与饱和度,YUV色彩空间使用U与V两个色差信号来表示色度,其中,V反映了输入信号红色部分与RGB信号亮度值之间的差异,而U反映的是RGB输入信号蓝色部分与RGB信号亮度值之间的差异。YUV与RGB相互转换的公式如下:

$$Y = 0.299R + 0.587G + 0.114B$$

$$U = -0.147R - 0.289G + 0.436B$$

$$V = 0.615R - 0.515G - 0.100B$$

$$R = Y + 1.407\,5(V - 128)$$

$$G = Y - 0.345\,5(U - 128) - 0.716\,9(V - 128)$$

$$B = Y + 1.779(U - 128)$$

在DirectShow中,常见的YUV格式有YUY2,YUYV,YVYU,UYVY,AYUV,Y41P,Y411,Y211,IF09,IYUV,YV12,YVU9,YUV411,YUV420等。

1.2 YUV411,YUV420格式

YUV411,YUV420格式是常用的两种格式,前者用于NTSC制视频,后者用于PAL制视频。YUV格式通常有两大类:打包格式和平面格式。前者将YUV分量存放在同一个数组中,通常是几个相邻的像素组成一个宏像素;而后者使用3个数组分开存放YUV 3个分量,就像是一个3维平面一样。YUV411,YUV420是平面格式的YUV矢量矩阵。

YUV411为每个像素都提取分量Y,而分量UV在水平方向上每4个像素采样一次(如图1所示)。YUV420并非分量V采样为0,而是跟YUV411相比,在水平方向上提高一倍色差采样频率,在垂直方向上以U/V间隔的方式减小一半色差采样(图2)。

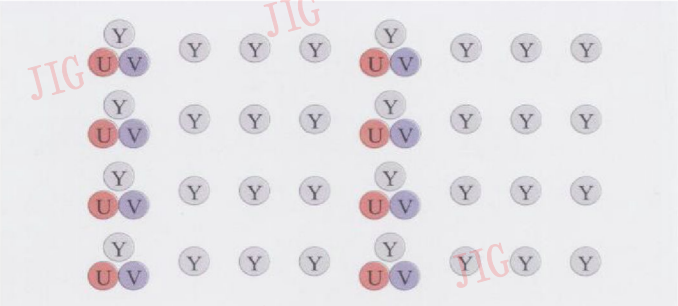

图1 YUV411格式

Fig. 1 YUV411 format

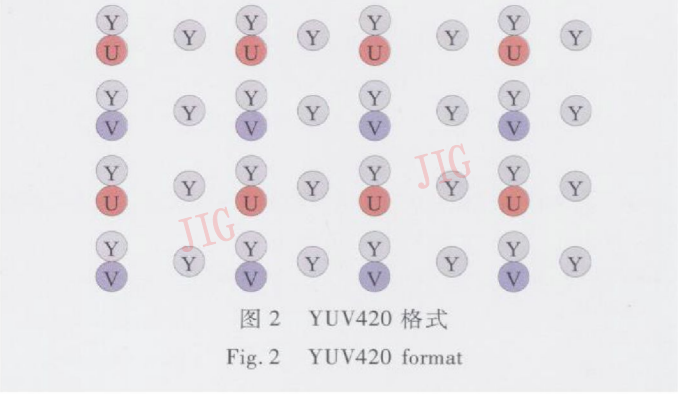

图2 YUV420格式

Fig. 2 YUV420 format

YUV411/YUV420格式一帧CIF分辨率图像数据量为352×288×1.5=152 064字节,RGB24格式一帧CIF分辨率数据量是352×288×3=304 128字节,所以YUV411/YUV420的大小是RGB24格式的一半。

2 SSE2指令集

Intel的奔腾处理器系列给IA-32指令集提供了高级数学处理能力。如果开发需要大量数学处理(比如音频和视频处理)的应用程序,利用高级数学特性能够在很大程度上提高应用程序的性能。SSE2(streaming simd extensions 2,Intel官方称为SMD流技术扩展2或数据流单指令多数据扩展指令集2)指令集是Intel公司在SSE指令集的基础上发展起来的。相比于数据流单指令多数据扩展指令集SSE,SSE2使用了144个新增指令,添加5种新的数据类型,扩展了多媒体扩展指令集MMX技术和数据流单指令多数据扩展指令集SSE技术,这些指令提高了应用程序的运行性能。

SSE2包含处理打包字节、打包字、打包双字和打包四字整数值的指令,为数据处理提供了大量可以利用的能力。每种打包数据类型都包含数学指令,用于使用单一指令并行地处理打包数据类型。

3 YUV格式与 RGB格式快速算法分析

YUV格式与 RGB格式的转换公式牵涉到浮点运算, 而计算结果只需要整数部分。由于在计算机中整型运算的速度快于浮点运算, 移位操作快于除法操作, 所以要实现快速算法, 可以分析运算方法特点, 将计算过程转换为整型运算, 并使用 SSE2指令集的单指令多数据操作来提高计算速度。

3.1 整型算法

计算机中除 2的幂数倍可用移位操作完成, 如除 2可以用右移 1比特代替, 除 4用右移 2比特代替。一个字数据用 2字节表示, 当对该数据除 256时, 可将该数据右移 8比特, 实际操作变为直接取字数据的高字节。通过上述方法对转换公式进行变换, 使之成为全整数操作, 变换方法为将红色分量 R计算公式中的浮点数 1.407 5用 (360/256)代替, 误差为 0.001 25, 然后将除 256操作用移位操作代替。由于结果取值范围 0~255, 最大误差为 255×0.001 25= 0.318 75, 而计算结果只取整数部分, 所以参数替换的误差对计算结果几乎不产生影响。由于移位操作对数据进行了截取, 所以要在计算过程的最后进行, 避免影响计算结果。

$$\begin{aligned} R &= Y + 1.407\,5(V-128) \\ &= Y + (360/256)(V-128) \\ &= Y + 360(V-128)/256 \\ &= Y + (360(V-128)) \gg 8 \end{aligned}$$

同样, 绿色分量 G 计算中将参数 0.345 5用 (88/256)代替, 参数 0.716 9用 (183/256)代替, 蓝色分量 B 计算中将参数 1.779用 (455/256)代替。公式转换为

$$\begin{aligned} G &= Y - ((88(U-128)) \gg 8) - \\ &\quad ((183(V-128)) \gg 8) \\ B &= Y + (455(U-128)) \gg 8 \end{aligned}$$

3.2 结合 SSE2指令集优化

下面使用汇编语言通过 SSE2指令实现格式转换。此汇编代码通过嵌入式代码, 可以方便地在 C/C++程序中使用。该算法实现了 YUV420与 RGB24之间的转换, 能够一次完成 8个像素点格式的计算。

在 SSE2指令操作中, 寄存器 xmm0-xmm7可以保存 128位的数据, 这里用寄存器存放 8个有符号字数据, 定义数据结构体如下:

```
struct _CRT_ALIGN(16) __s128 { __int16 a[8]; };
```

为使用 SSE2指令集, 先对数据进行打包排列, 亮度信号 Y按 $\{Y_0, Y_1, Y_2, Y_3, Y_4, Y_5, Y_6, Y_7\}$保存在长度 128bits结构体中, 每个值占一个字长 (16bits), 色差信号 U按 $\{U_0, U_0, U_1, U_1, U_2, U_2, U_3, U_3\}$保存, 色差信号 V按 $\{V_0, V_0, V_1, V_1, V_2, V_2, V_3, V_3\}$保存。定义 3个结构体 packedR, packedG, packedB保存计算结果。

```
__s128 packedY = {Y0, Y1, Y2, Y3, Y4, Y5, Y6, Y7};
__s128 packedU = {U0, U0, U1, U1, U2, U2, U3, U3};
__s128 packedV = {V0, V0, V1, V1, V2, V2, V3, V3};
__s128 packedR;
__s128 packedG;
__s128 packedB。
```

对计算中使用的固定数值, 直接定义为常数。

```
const __s128 packedConst128 = {-128, -128, -128, -128, -128, -128, -128, -128};
const __s128 packedConst360 = {360, 360, 360, 360, 360, 360, 360, 360};
const __s128 packedConst88 = {88, 88, 88, 88, 88, 88, 88, 88};
const __s128 packedConst183 = {183, 183, 183, 183, 183, 183, 183, 183};
const __s128 packedConst455 = {455, 455, 455, 455, 455, 455, 455, 455}。
```

首选计算 8个像素点 RGB格式中的红色分量 R值: 使用 SSE2指令集的 movdqa指令, 将打包的色差信号 packed$V$装入寄存器 xmm0, 使用 paddsw指令, 装打包好的常数与寄存器 xmm0内的数据进行带符号整数值相加, 完成对寄存器内的 8个值减 128; 使用 pmullw指令, 对 xmm0内的数据进行有符号字相乘, 完成对寄存器内的 8个字乘以 360, 再用 psraw指令对寄存器内的数据按字算术右移 8位, 完成除法; 最后将打包的亮度分量 packedY与 xmm0寄存器中的值相加, 得到 8个像素点的 R值 (R_0-R_7)。使用 movdqa将结果保存在内存中。操作过程如图 3所示。

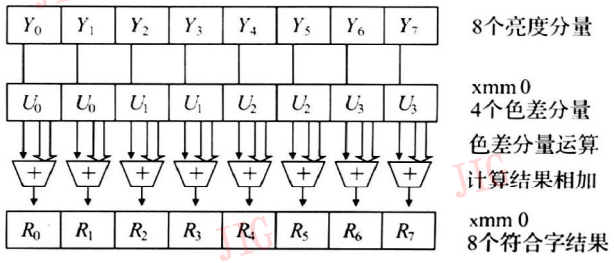

图 3 RGB格式红色分量 R的 SSE2计算

Fig. 3 SSE2 computations of red component on RGB format

计算绿色分量 G 值: 将打包的亮度分量 packed$Y$ 装入寄存器 xmm0, 将打包的色差分量 packed$U$ 装入寄存器 xmm1, 使用 paddsw 对寄存器中的数按字进行减 128, 再用 pmullhw 指令对寄存器 xmm1中的数按字乘 360, 最后将寄存器 xmm1中的数按字算术右移 8bits后, 与寄存器 xmm0相加, 相加结果保存在 xmm0中。接着将打包的色差分量 packed$V$ 装入寄存器 xmm1, 进行运算后, 将结果再与寄存器 xmm0相加。xmm0中即为 8个像素点 RGB格式中的绿色分量 G 值 ($G_0 - G_7$), 使用 movdqa将结果保存在内存中。

计算蓝色分量 B 过程与计算 R 过程一样。最后将计算结果保存在内存中。操作完成后, 使用 emms指令清除 MMX状态。汇编代码如下:

```
__asm{
movdqa xmm0, packedV;
paddsw xmm0, packedConst128;
pmullw xmm0, packedConst360;
psraw xmm0, 8;
paddsw xmm0, packedY;
movdqa packedR, xmm0;
movdqa xmm0, packedY;
movdqa xmm1, packedU;
paddsw xmm1, packedConst128;
pmullw xmm1, packedConst88;
psraw xmm1, 8;
psubw xmm0, xmm1;
movdqa xmm1, packedV;
paddsw xmm1, packedConst128;
pmullw xmm1, packedConst183;
psraw xmm1, 8;
psubw xmm0, xmm1;
mpvdqa packedG, xmm0;
movdqa xmm0, packedU;
paddsw xmm0, packedConst128;
pmullw xmm0, packedConst455;
psraw xmm0, 8;
paddsw xmm0, packedY;
movdqa packedR, xmm0;
emms;
}
```

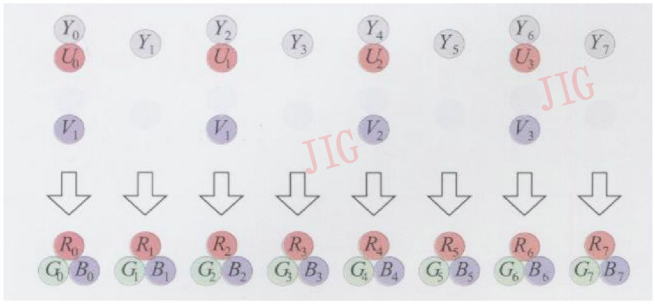

图 4 YUV420转换 RGB

Fig. 4 YUV420 conversion RGB

一次计算, 可完成 8个像素点的 R, G, B 值 (图 4)。完成一次计算后, 将后 8个像素点的 Y 值 (Y_8-Y_{15})、U 值 (U_5-U_7)和 V 值 (V_5-V_7)打包, 再次进行运算, 循环直至所有数据处理完。使用 SSE2指令减少了循环遍历次数, 从而减少了运算时间。

上述代码是计算 YUV420格式的数据, 对 YUV411格式, 只要加载数据时 U 分量按 {U_0, U_0, U_0, U_0, U_1, U_1, U_1, U_1}排列, V 分量按 {V_0, V_0, V_0, V_0, V_1, V_1, V_1, V_1}排列即可。

4 实验结果与结论

在主频为 2.4GHz的 Intel Pentium 4处理器计算机上, 分别通过直接公式计算、公式整数优化计算、SSE2优化计算 3种方式转换一帧 CIF格式图像测试, 计算 1 000次所需要的时间结果如图 5所示:

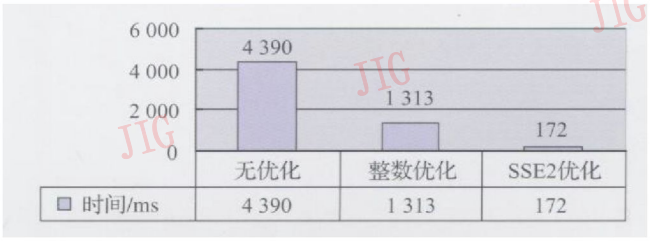

图 5 计算 1 000帧所需时间

Fig. 5 The time of calculation on 1 000 frams

实验结果表明, 使用 SSE2指令集优化后相对直接公式计算能提高 25倍的计算速度。所以, 在 YUV视频格式与 RGB视频格式转换特点和 Intel SSE2多媒体扩展指令集基础上, 利用整型算法代替浮点运算, 使用 Intel SSE2对速整型算法模块的进行优化算法, 可以大大提高了视频格式转换速度。通过图像对比, 优化后的算法不影响图像质量。基于 SSE2指令集, 对算法进行工程优化设计和实现, 是在目前通用计算机上获得较大性能提高的一种有效、可行的方法。

参考文献 (References)

[1] International Telecommunication Union. Draft ITU-T Recommendation H.263 Video Coding for Low Bitrate Communication[S]. ITU-T, 1996

[2] Intel Corporation. Intel 64 and IA-32 Architectures Optimization

Reference Manual[EB/OL]. [2009-09-09]. http://www.intel.com/Assets/PDF/manual/248966.pdf

[3] Joch A, Kossentini F, Nasiopoulos P A. Performance Analysis of the ITU-T Draft H.264 Video Coding Standard[C]//Proceedings of the 12th International Packet Video Workshop. Pittsburg, Pa, USA: [s.n.]2002.

[4] Diefendorff K. Pentium III = Pentium II + SEE, Internet SSE Architecture Boosts Multimedia Performance[EB/OL]. [2009-09-09]. http://www.cs.cmu.edu/afs/cs/academic/class/15740-f03/public/doc/discussions/uniprocessors/media/mpr_p3_mar99.pdf

[5] Raman S K, Pentkovski V, Keshava J. Implementation streaming SIMD extensions on the pentium III processor[J]. IEEE Micro, 2000, 20(4): 47-57.