

# 基于 SIMD 扩展指令进行图像处理加速实验报告

姓名：张煌昭  
学号：1400017707  
院系：元培学院  
邮箱：[zhang\\_hz@pku.edu.cn](mailto:zhang_hz@pku.edu.cn)  
手机：17888838127

## 一. 实验要求

### 1. 单幅图像淡入/淡出

从相关文件 demo 目录中读入一幅 YUV420 格式的图像（大小均为  $1920 \times 1080$ ）后进行如下操作：首先进行 YUV420 到 ARGB8888 的转换，Alpha 取值为 3, 6, 9, ..., 255，共生成 84 幅 ARGB8888 格式的图像；再进行 ARGB8888 到 RGB888 的转换，将上一步生成的图像进行 Alpha 混合生成 84 幅 RGB888 格式的图像；最后进行 RGB888 到 YUV420 的转换，将 Alpha 混合所得图像按帧转换为 YUV420 格式并连续保存，最终得到有 84 帧的一段 YUV 格式的视频文件。

### 2. 两幅图像的叠加渐变

从相关文件 demo 目录中读入两幅 YUV420 格式的图像（大小均为  $1920 \times 1080$ ）后进行如下操作：首先进行 YUV420 到 RGB888 的转换，每个图像各生成 84 幅 RGB888 格式的图像；再进行 Alpha 混合，按照上一部分的 Alpha 取值将两图像进行叠加，得到 84 幅 RGB888 格式的图；最后进行 RGB888 到 YUV420 的转换，将图像按帧转换为 YUV420 格式并连续保存，最终得到 84 帧的一段 YUV 格式的视频文件。

### 3. SIMD 指令加速处理

使用 x86 的 MMX, SSE2, AVX 扩展指令对上述两部份进行加速处理，统计加速后处理时间的变化。

## 二. 基础知识

### 1. 图像格式介绍

**YUV420** 图像每个像素具有三个分量，明度 Y，色度 U 和 V，每个分量用一个字节表示。其中 Y 为图像的灰度值，U 和 V 描述图像的色彩和饱和度。YUV 具有 YUV444, YUV422, YUV420 等不同的采样方式。其中 YUV420 采样的 Y 分量对每个像素进行采样，U 和 V 对周围的四点采样，如下左图所示，实心点为 Y 分量采样，空心点为 U 和 V 分量采样。



一个 YUV420 格式的图像，假设大小为  $w \times h$ ，在存放时首先按行连续地存放  $w \times h$  字节的 Y 分量，之后按行连续地存放  $\frac{w}{2} \times \frac{h}{2}$  字节的 U 分量，最后按行连续地存放  $\frac{w}{2} \times \frac{h}{2}$  字节的 V 分量，示意图如上右图。

**ARGB8888** 图像每个像素具有四个分量，透明度 A 和红绿蓝三色灰度 RGB，每个分量用一个字节表示。一个 ARGB8888 图像，假设大小为  $w \times h$ ，在存放时按行连续地存放  $w \times h$  个像素，每个像素按顺序存放 ARGB 分量，因此图像总共需要  $w \times h \times 4$  个字节存放。

**RGB888** 图像每个像素具有三个分量，分别为红绿蓝三色灰度 RGB，每个分量用一个字节表示。一个 RGB888 图像，假设大小为  $w \times h$ ，在存放时按行连续地存放  $w \times h$  个像素，每个像素按顺序存放 RGB 分量，因此图像总共需要  $w \times h \times 3$  个字节存放。

## 2. 图像格式转换

**RGB2YUV** 按照如下公式<sup>[1]</sup>进行转换。

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & -0.001 & 1.402 \\ 1 & -0.344 & -0.714 \\ 1 & 1.772 & 0.001 \end{bmatrix} \begin{bmatrix} Y \\ U - 128 \\ V - 128 \end{bmatrix}$$

**YUV2RGB** 按照如下公式<sup>[1]</sup>进行转换。

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.169 & -0.331 & -0.500 \\ 0.500 & -0.419 & 0.081 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} + \begin{bmatrix} 0 \\ 128 \\ 128 \end{bmatrix}$$

**ARGB2RGB** 按照如下公式进行混合。

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} A/255 \\ A/255 \\ A/255 \end{bmatrix} \circ \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

**图像叠加**按照如下公式进行。

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} A/255 \\ A/255 \\ A/255 \end{bmatrix} \circ \begin{bmatrix} R1 \\ G1 \\ B1 \end{bmatrix} + \begin{bmatrix} (255 - A)/255 \\ (255 - A)/255 \\ (255 - A)/255 \end{bmatrix} \circ \begin{bmatrix} R2 \\ G2 \\ B2 \end{bmatrix}$$

## 3. MMX 扩展指令

MMX 扩展指令使用 64 位寄存器，允许同时打包处理不同的 8 个字节（8 位），或 4 个半字（16 位），或 2 个字（32 位），或 1 个双字（64 位），支持定点加减和乘法，以及位运算等。可以通过引用 `mmintrin.h` 头文件，在 C/C++ 代码中使用封装的 MMX 指令<sup>[2]</sup>。

## 4. SSE 扩展指令

SSE 扩展指令使用 128 位寄存器，允许同时打包处理不同的 4 个单精度浮点数（32 位），

或 2 个双精度浮点数（64 位），支持浮点算术运算等。可以通过引用 `xmmintrin.h` 头文件，在 C/C++ 代码中使用封装的 SSE 指令<sup>[2]</sup>。

SSE2 扩展指令同 SSE 扩展指令类似，使用 128 位寄存器，允许同时打包处理不同的 16 个字节（8 位），或 8 个半字（16 位），或 4 个字（32 位），或 2 个双字（64 位），用于定点运算。可以通过引用 `emmintrin.h` 头文件，在 C/C++ 代码中使用封装的 SSE2 指令<sup>[2]</sup>。

5. AVX 扩展指令

AVX 扩展指令使用 256 位寄存器，允许同时打包处理不同的 8 个单精度浮点数（32 位），或 4 个双精度浮点数（64 位），支持浮点算术运算等。可以通过引用 `immintrin.h` 头文件，在 C/C++ 代码中使用封装的 AVX 指令<sup>[2]</sup>。

AVX2 扩展指令使用 256 位寄存器，允许同时打包处理不同的 32 个字节（8 位），或 16 个半字（16 位），或 8 个字（32 位），或 4 个双字（64 位），用于定点运算。可以通过引用 `immintrin.h` 头文件，在 C/C++ 代码中使用封装的 SSE2 指令<sup>[2]</sup>。

三 . 基准

1. 实验环境

实验均使用相同的 DELL XPS13 笔记本电脑，在 Ubuntu 虚拟机环境下进行，具体的参数如下。

表. 实验环境	
项目	详细指标和参数
处理器型号及相关参数	Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz 8-way set-associative, 64 sets 32KB size, Level-1 D-/I-Cache 4-way set-associative, 1024 sets 256KB size, Level-2 Cache 16-way set-associative, 4096 sets 4096KB size, Level-3 Cache
内存	4GB Main Memory
外存	VBOX HARDDISK 1.0, 10GB device size 512byte logical/physical sector size
操作系统及其版本	Linux 4.10.0-28-generic 16.04.2-Ubuntu x86_64 GNU/Linux
编译器版本	GCC version 5.4.0 20160609 (Ubuntu 5.4.0-6ubuntu1~16.04.4)

2. 播放 YUV420 格式图像/视频

首先编写 python 程序用于将 YUV 格式直接转为 RGB 的 jpg 格式，以及显示转换中间的没有头的 RGB 格式文件。

使用 ffplay 播放 YUV 格式的图像或视频，播放 dem1.yuv（左）和 dem2.yuv（右）的截图效果如下。



### 3. 基准实现

不使用任何的扩展指令加速，对要求的两个功能进行实现，添加"-mavx2"编译选项以便后续使用扩展指令加速处理。详情请见所附代码。

使用"./yuv\_conv ./dem1.yuv ./dem2.yuv ./res1.yuv ./res2.yuv ./res.yuv -no"命令运行yuv\_conv程序进行YUV图像处理工作,读入当前目录下的dem1.yuv和dem2.yuv文件,对其进行淡入淡出处理存放于当前目录下的res1.yuv和res2.yuv文件,进行叠加渐变处理,存放于当前目录下的res.yuv文件。程序的运行结果截图如下图,叠加渐变耗时18.397秒,淡入淡出分别耗时12.897和12.860秒,具体的各步运行时间如下表所示。

[illegible]

表. YUV 图像淡入淡出和叠加渐变处理基准结果

Merge	YUV2RGB / s	RGB Merge / s	RGB2YUV / s	Total / s
dem1.yuv + dem2.yuv	9.118	2.678	6.601	18.397
Fade	YUV2ARGB / s	ARGB2RGB / s	RGB2YUV / s	Total / s
dem1.yuv	3.905	2.268	6.724	12.897
Fade	YUV2ARGB / s	ARGB2RGB / s	RGB2YUV / s	Total / s
dem2.yuv	3.908	2.267	6.685	12.860

叠加渐变的 YUV420 格式视频的播放截图如下图上部；dem1.yuv 淡入淡出的 YUV420 格式视频的播放截图如下图中部；dem2.yuv 淡入淡出的 YUV420 格式视频的播放截图如下图下部。发现图像整体偏红，可能是在转化过程中防止溢出的饱和操作所致。



#### 四．优化处理

##### 1. MMX 扩展指令优化

首先考虑对基准程序进行循环展开来进行优化，计算过程中定点加减法均可以通过 MMX 指令完成，但浮点乘法无法完成，因此考虑将浮点乘法转为一次定点乘法和一次右移操作进行，例如 $0.081 \times B$ 可以转化为 $(\text{unsigned char}(0.081 \times 128) \times B) \gg 7$ 来进行。

将公式改写为上述的定点形式后，即可进行循环展开，由于需要进行 8 位定点乘法，因此使用半字打包，一个寄存器可以装下 4 个数据，因而展开层数为 4。详细情况请见所附代码。

使用“./yuv\_conv ./dem1.yuv ./dem2.yuv ./res1.yuv ./res2.yuv ./res.yuv -mmx”命令运行程序进行处理。程序的运行结果截图如下图，叠加渐变耗时 20.313 秒，淡入淡出分别耗时 12.689 和 13.175 秒，具体的各步运行时间如下表所示。









### 3. AVX2 扩展指令优化

继续采用循环展开优化顶点操作的思路进行 AVX2 扩展指令进行优化处理。优化的思路为利用 AVX2 的 256 位寄存器，将循环展开的层数扩大至 16 层。详细情况请见所附代码。

使用"./yuv\_conv ./dem1.yuv ./dem2.yuv ./res1.yuv ./res2.yuv ./res.yuv -avx"命令运行程序进行处理。程序的运行结果截图如下图，叠加渐变耗时 15.518 秒，淡入淡出分别耗时 9.927 和 9.666 秒，具体的各步运行时间如下表所示。

[illegible]



表. YUV 图像淡入淡出和叠加渐变处理 AVX2 优化结果				
Merge	YUV2RGB / s	RGB Merge / s	RGB2YUV / s	Total / s
dem1.yuv + dem2.yuv	7.637	3.748	4.133	15.518
Fade	YUV2ARGB / s	ARGB2RGB / s	RGB2YUV / s	Total / s
dem1.yuv	3.486	2.274	4.157	9.927
Fade	YUV2ARGB / s	ARGB2RGB / s	RGB2YUV / s	Total / s
dem2.yuv	3.381	2.205	4.079	9.666

叠加渐变的 YUV420 格式视频的播放截图如下图上部；dem1.yuv 淡入淡出的 YUV420 格式视频的播放截图如下图中部；dem2.yuv 淡入淡出的 YUV420 格式视频的播放截图如下图下部。图像与基准相同。



## 五．实验结果和分析

### 1. 实验结果

将上述所有实验结果汇总，得到下表。发现 MMX 没有起到优化效果，SSE2 和 AVX2 均起到一定的优化效果。

表. SMID 扩展指令加速处理实验结果

Merge dem1.yuv and dem2.yuv				
SIMD	YUV2RGB / s	RGB Merge / s	RGB2YUV / s	Total / s
No	9.118	2.678	6.601	18.397
MMX	10.268 (113%)	4.529 (169%)	5.517 (84%)	20.313 (110%)
SSE2	8.233 (90%)	3.690 (138%)	4.618 (70%)	16.541 (90%)
AVX2	7.637 (84%)	3.748 (140%)	4.133 (63%)	15.518 (84%)

  

Fade dem1.yuv				
SIMD	YUV2RGB / s	RGB Merge / s	RGB2YUV / s	Total / s
No	3.905	2.268	6.724	12.897
MMX	4.867 (125%)	2.558 (113%)	5.263 (78%)	12.689 (98%)
SSE2	3.751 (96%)	2.272 (100%)	4.553 (68%)	10.578 (82%)
AVX2	3.486 (89%)	2.274 (100%)	4.157 (62%)	9.927 (77%)

  

Fade dem2.yuv				
SIMD	YUV2RGB / s	RGB Merge / s	RGB2YUV / s	Total / s
No	3.908	2.267	6.685	12.860
MMX	5.017 (128%)	2.678 (118%)	5.481 (82%)	13.175 (102%)
SSE2	3.739 (96%)	2.284 (101%)	4.522 (68%)	10.545 (82%)
AVX2	3.381 (87%)	2.205 (97%)	4.079 (61%)	9.666 (75%)

## 2. 结果分析

SSE2 和 AVX2 均使用了很大的循环展开，因此大多数情况下具有加速效果属于正常现象。

下面分析两个情况：a. MMX 没有起到优化效果；b. Alpha 混合基本不受加速。

a. 由于寄存器大小限制，MMX 只进行了 4 层展开，因此有可能因为展开加速小于展开的开销，使得用时反而增大。深入考虑后，发现其实并没有为了使用定点乘法而使用半字打包，由于最后需要右移取高位，因而实际上只需要定点乘取高位即可，如此便可以省下四个元素，从而进行 8 层展开。

b. Alpha 混合几乎全部是乘除操作，并且本身用时很短，因此不容易被加速，或者不容易被观测到加速。

## 六. 使用说明

运行所附 yuv\_conv 程序,或重新通过 qmake 编译源代码得到 yuv\_conv 程序后再执行。使用的命令为如下形式

```
. /yuv_conv <YUV path 1> <YUV path 2> <Fade path 1> <Fade path 2> <Merge path> <Optimize option>
```

其中，YUV path 1 和 YUV path 2 为需要进行操作的 YUV420 格式的大小为 1920\*1080 的图像的路径，Fade path 1 和 Fade path 2 分别为对应的淡入淡出的 YUV420 格式视频的存储路径，Merge path 为重叠渐变的 YUV420 格式视频的存储路径，Optimize option 为使用的优化选项，“-no”不使用任何 SIMD 优化，“-mmx”使用 MMX 扩展指令，“-sse”使用 SSE2

扩展指令，“-avx”使用 AVX2 扩展指令，不进行设置默认使用“-no”选项。

## 七．参考资料

[1] 维基百科 YUV 词条

[2] Intel 扩展指令手册 <https://software.intel.com/sites/landingpage/IntrinsicsGuide>