基于 SIMD 与多核编程的电子相册的设计与实现

作者：崔悦 李亮

学号：3118311046 3118311015

指导教师：朱利教授

2018 年 11 月

摘　要

传统的电子相册淡入淡出的实现，对图片按像素进行循环串行处理，当图片分辨率较高时，像素数量可达到百万级别，传统的方法效率比较低，融合速度较慢，甚至会出现视觉感受不流畅的问题。

本文的主要工作是设计和实现一个电子相册应用，主要功能是读取对应路径下图片进行融合处理，以淡入淡出的效果进行显示。文中分析了传统图像融合方法中存在的问题，用不同方法进行加速融合并进行了评测和对比。

本文采用 opencv 开源库图像处理技术和 SSE 和多核编程技术作为基本技术，使用了 vs2017 集成开发环境搭建了系统编码环境，完成了基于 SSE 和多核编程的电子相册的编码，并分别对此应用进行了功能测试和性能测试。测试结果说明，应用满足所有业务需求，实现了所有功能，达到了性能要求。

关 键 词：SEE；多线程；电子相册；淡入淡出

论文类型：应用研究

目录

1 背景和意义

开发一个能够高效进行图像融合的应用是非常有意义的，尤其是对于当期图像和视频分辨率日趋增大的情况。并且 Intel SIMD 指令集直到目前已经发展多代，从最初 MMX 寄存器只有 64 位到现在 AVX 达到 512 位，硬件的加速发展也推动着软件开发的发展，很多编程者对这些指令集在软件中的应用并不多，这也是一种资源的浪费。

2 系统主要相关技术和基础理论

为了保证基于 SSE 和多核编程应用的顺利实现，并在实现过程中减少开发风险，提升代码的可移植行，增加应用的可靠性。本系统将采用 opencv 开源库作为图像处理基本技术，将 opencv 和 SIMD 与多线程编程结合起来，提高效率，加速图像的融合。

2.1 OpenCV 介绍

OpenCV 是一个基于 BSD 许可（开源）发行的跨平台计算机视觉库，可以运行在 Linux、Windows、Android 和 Mac OS 操作系统上。它轻量级而且高效——由一系列 C 函数和少量 C++ 类构成，同时提供了 Python、Ruby、MATLAB 等语言的接口，实现了图像处理和计算机视觉方面的很多通用算法。

OpenCV 的优点

计算机视觉市场巨大而且持续增长，且这方面没有标准 API，如今的计算机视觉软件大概有以下三种：

1、研究代码（慢，不稳定，独立并与其他库不兼容）

2、耗费很高的商业化工具（比如 Halcon, MATLAB+Simulink）

3、依赖硬件的一些特别的解决方案（比如视频监控，制造控制系统，医疗设备）这是如今的现状，而标准的 API 将简化计算机视觉程序和解决方案的开发，OpenCV 致力于成为这样的标准 API。

OpenCV 致力于真实世界的实时应用，通过优化的 C 代码的编写对其执行速度带来了可观的提升，并且可以通过购买 Intel 的 IPP 高性能多媒体函数库

（Integrated Performance Primitives）得到更快的处理速度。右图为 OpenCV 与当前其他主流视觉函数库的性能比较。

2.2 SSE 介绍

SIMD 全称 Single Instruction Multiple Data，单指令多数据流，能够复制多个操作数，并把它们打包在大型寄存器的一组指令集。

支持 SIMD 的处理器在音频解码、视频回放、3D 游戏等应用中显示出优异的性能。

SIMD 技术的大致发展脉络是

1996 MMX 64bit Registers

1999-2008 SSE-SSE4 128bit Register

2010 AVX 256bit Register

SSE(SSE(Streaming SIMD Extensions)是 SIMD 技术的一种，它包括 70 条指令，其中包含单指令多数据浮点计算、以及额外的 SIMD 整数和高速缓存控制指令，可同时对 4 个 32 位单精度浮点数进行运算处理。其优势在于更高分辨率的图像浏览和处理、更高精度和更快响应速度。

2.3 多核编程介绍

通过对多核与单核多线程的比较可更清楚的解释多核编程。

首先介绍并行与并发：并行指两件（多件）事情在同一时刻一起发生；并发：两件（多件）事情在同一时刻只能有一个发生，由 CPU 快速切换，从而给人的感觉是同时进行

使用多线程来实现并行计算来缩短计算时间时，只有在多核 CPU 下才行，单核 CPU 下启用多线程最终总的计算计算一样，因为 CPU 在同一时间，只能服务于一个线程，

在单核 CPU 下运用多线程仅仅能实现快速响应用户的请求，避免因 io 或网络阻塞而导致界面停留卡顿。

3 基于 SSE 和多核编程应用的分析与实现

3.1 应用需求描述

分别为功能需求描述和性能需求描述。

3.1.1 功能需求描述

1）相册获取

程序对相对路径下的若干相册进行读取(每个相册是一个文件夹，里面有若干相片)，相册数量理论上没有限制，每个相册相片数量理论上没有限制。

2) 相片融合

就单独一个相册来说，对其中相片按序进行两两融合并实现淡入淡出显示，要求视觉上没有突兀感。

3) 相册展示

每个相册有单独的窗口进行显示，可拖动大小和位置，方便进行布局。

3.1.2 性能需求描述

预期融合速度接近传统方法的 8 倍：理论依据是由于采用 128 位寄存器进行多数据处理，每次可同时处理 8 个字节数据，相比较传统每次处理一个字节，。

3.2 原理介绍与具体实现

3.2.1 本节内容介绍

电子相册的基本功能是将多幅图片轮流显示，并增加前后两张图片的过渡效果。其中，过渡效果的实现是本实验的重点，所以，本节的主要内容将分为以下几点进行展示：

1 图像融合的基本原理；

2 图像融合的普通实现；

3 在普通实现基础上对循环体进行改造，增大循环步长；

4 在普通实现基础上利用 OPENMP 进行并行加速；

5 使用 SSE 加速图像融合；

6 以上四种方法的比较和总结；

3.2.2 图像融合的基本原理

图像融合的基本原理是将两幅图像相同位置上的 RGB 值进行加权求和，总权值为 1，计算公式如下：

Result = (A-B)*fade+B

其中，A 代表 A 图片的对应像素 RGB 值，B 代表 B 图片对应像素的 RGB 值，fade 代表权重，Result 代表两幅图像对应像素的融合值。本实验中为提高计算速度，将权重扩大 128 倍再除以 128，其中除以 128 可用移位操作代替。公式变为：

Result = ((A-B)*fade)>>7+B

>>7 代表右移 7 位。

3.2.2 图像融合的四种实现

1 普通实现

实现代码如下：

```
void imgFusionNormal(cv::Mat src1, float alpha, cv::Mat src2, cv::Mat & dst)
{
MyRegion
    //将opencv Mat类型转为无符号数组进行处理
    TransMatToBuffer(src1, pBuffer1, nWidth, nHeight, nBandNum, nBPB, nMemSize);
    TransMatToBuffer(src2, pBuffer2, nWidth, nHeight, nBandNum, nBPB, nMemSize);
    //矩阵运算，两个图像融合
    uchar* pBuffer3 = new uchar[nMemSize]();                                    //()表默
    for (int i = 0; i < nMemSize; ++i)
    {
        *(pBuffer3 + i) = uchar((*(pBuffer1 + i) - *(pBuffer2 + i))*alpha + *(pBuffe
    }
    dst = TransBufferToMat(pBuffer3, nWidth, nHeight, nBandNum, nBPB);      //将处理
MyRegion
}
```

图 1

函数使用两个 opencv Mat 类型的图像数据作为输入，对每一个像素值进行加权求和。融合效果的一个展示如下（使用的图片是两张高清照片）：

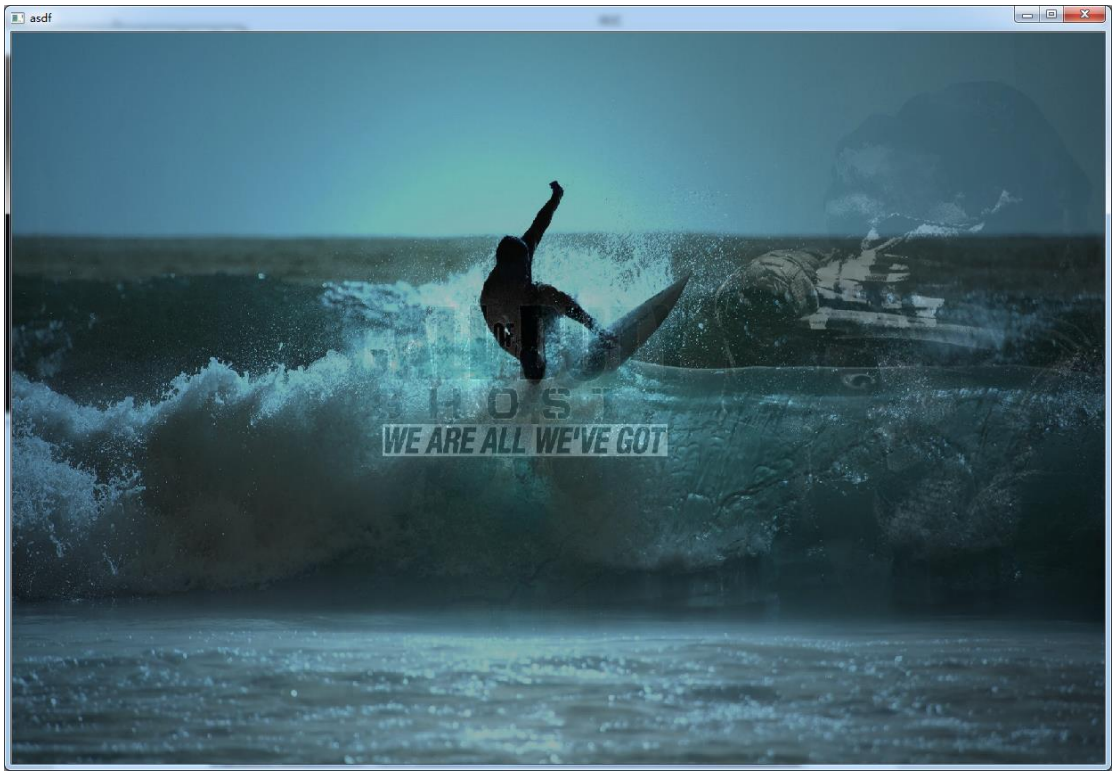


图 2

2 增大循环步长

在进行 SSE4 实验中，发现循环体是同时对 8 个像素值进行加权求和，于是想到可以对普通实现进行修改，将循环体进行优化，每次循环同时对 8 个像素值进行加权求和。实现代码如下：

```
void imgFusionLongStep(cv::Mat src1, float alpha, cv::Mat src2, cv::Mat & dst)
{
MyRegion
    size_t nMemSize = 0;
    //将opencv Mat类型转为无符号数组进行处理
    TransMatToBuffer(src1, pBuffer1, nWidth, nHeight, nBandNum, nBPB, nMemSize);
    TransMatToBuffer(src2, pBuffer2, nWidth, nHeight, nBandNum, nBPB, nMemSize);
    //矩阵运算，两个图像融合
    uchar* pBuffer3 = new uchar[nMemSize]();                                    //()表默认初始化为0
    int nBlocks = nMemSize / 8;                    //按8byte分块数量
    int nBlocksX = nBlocks * 8;
    int nRem = nMemSize % 8;                       //剩余bytes
    for (int i = 0; i < nBlocksX;  i+= 8)
    {
        *(pBuffer3 + i) = uchar((*(pBuffer1 + i) - *(pBuffer2 + i))*alpha + *(pBuffer2 + i));
        *(pBuffer3 + i+1) = uchar((*(pBuffer1 + i + 1) - *(pBuffer2 + i + 1))*alpha + *(pBuffer2 + i + 1));
        *(pBuffer3 + i+2) = uchar((*(pBuffer1 + i + 2) - *(pBuffer2 + i + 2))*alpha + *(pBuffer2 + i + 2));
        *(pBuffer3 + i+3) = uchar((*(pBuffer1 + i + 3) - *(pBuffer2 + i + 3))*alpha + *(pBuffer2 + i + 3));
        *(pBuffer3 + i+4) = uchar((*(pBuffer1 + i + 4) - *(pBuffer2 + i + 4))*alpha + *(pBuffer2 + i + 4));
        *(pBuffer3 + i+5) = uchar((*(pBuffer1 + i + 5) - *(pBuffer2 + i + 5))*alpha + *(pBuffer2 + i + 5));
        *(pBuffer3 + i+6) = uchar((*(pBuffer1 + i + 6) - *(pBuffer2 + i + 6))*alpha + *(pBuffer2 + i + 6));
        *(pBuffer3 + i+7) = uchar((*(pBuffer1 + i + 7) - *(pBuffer2 + i + 7))*alpha + *(pBuffer2 + i + 7));

    }
    for (int i = nBlocksX; i < nMemSize; ++i)
    {
        *(pBuffer3 + i) = uchar((*(pBuffer1 + i) - *(pBuffer2 + i))*alpha + *(pBuffer2 + i));
    }
    dst = TransBufferToMat(pBuffer3, nWidth, nHeight, nBandNum, nBPB);      //将处理完毕的无符号数组转为opencv
MyRegion
```

图 3

可以看见，循环体结构的改变。

4 利用 OPENMP 并行化循环体

使用 openMP 进行并行程序设计，在循环体前加入

#pragma　　omp parallel for num_threads(8)

```
void imgFusionNormal_OMP(cv::Mat src1, float alpha, cv::Mat src2, cv::Mat & dst)
{
MyRegion
    size_t nMemSize = 0;
    //将opencv Mat类型转为无符号数组进行处理
    TransMatToBuffer(src1, pBuffer1, nWidth, nHeight, nBandNum, nBPB, nMemSize);
    TransMatToBuffer(src2, pBuffer2, nWidth, nHeight, nBandNum, nBPB, nMemSize);
    //矩阵运算，两个图像融合
    uchar* pBuffer3 = new uchar[nMemSize]();                                    //()表默认初始化为0
#pragma omp parallel for num_threads(8)
    for (int i = 0; i < nMemSize; ++i)
    {
        *(pBuffer3 + i) = uchar((*(pBuffer1 + i) - *(pBuffer2 + i))*alpha + *(pBuffer2 + i));
    }

    dst = TransBufferToMat(pBuffer3, nWidth, nHeight, nBandNum, nBPB);      //将处理完毕的无符号
MyRegion
}
```

图 4

对 for 循环使用自动进行优化为多谢线程并行进行执行，可实现简单的多线程并行。另外，此种方法必须消除循环间的数据依赖，否则虽然能够加速运行，但是却得到错误的结果。

3 使用 SSE 加速

实现代码如下：

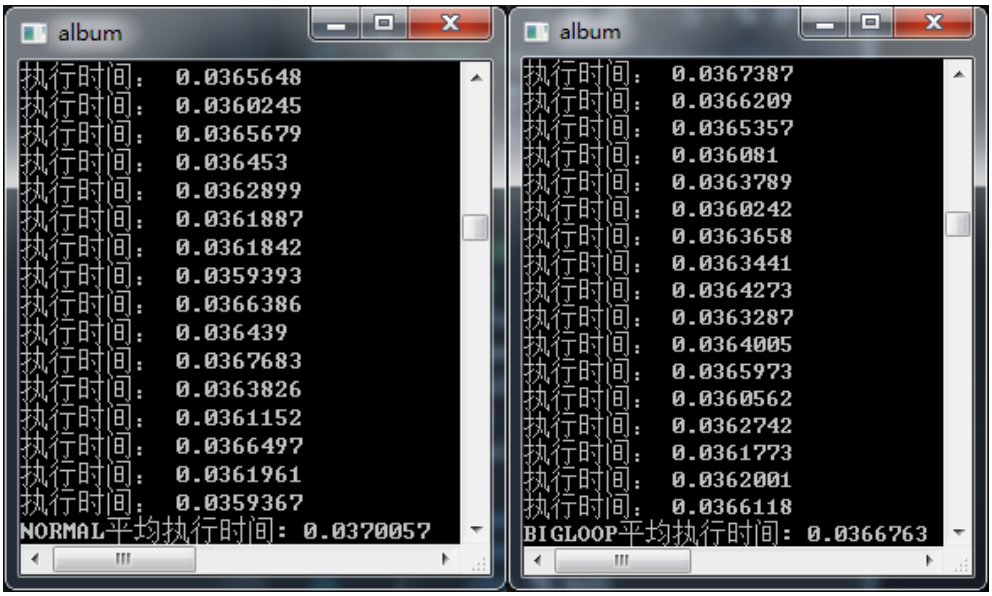
```
void imgFusionSSE(cv::Mat src1, float alpha, cv::Mat src2, cv::Mat& dst)
{
MyRegion
///////////////矩阵运算，两个图像融合////////SSE加速/////////////////////////////////////////////
    uchar* pBuffer3 = new uchar[nMemSize]();                    //()表默认初始化为0
    __m128i xmm0, xmm1, xmm2, xmm3, xmm4, xmm5, xmm6, xmm7, xmm9;
    __m128i xmm8 = _mm_setzero_si128();
    int fade = int(127*alpha);                      //将加权因子转为整型，方便后续运算
    xmm0 = _mm_set_epi16(fade, fade, fade, fade, fade, fade, fade, fade);   //8个fade因子装入寄
    int nBlocks = nMemSize / 8;                     //按8byte分块数量
    int nRem = nMemSize % 8;                        //剩余bytes
    for (int i = 0; i < nBlocks; ++i)               //初始化寄存器
    {
        xmm1 = _mm_loadu_si128((__m128i*)(pBuffer1 + i * 8));   //A的两个像素分量装入寄存器
        xmm1 = _mm_unpacklo_epi8(xmm1, xmm8);       //8个一位解紧缩至16位
        //xmm2 = _mm_loadu_si128((__m128i*)pByte2); //B的两个像素分量装入寄存器
        xmm2 = _mm_loadu_si128((__m128i*)(pBuffer2 + i * 8));   //B的两个像素分量装入寄存器
        xmm2 = _mm_unpacklo_epi8(xmm2, xmm8);       //8个一位解紧缩至16位
        xmm1 = _mm_sub_epi16(xmm1, xmm2);           //A-B
        xmm1 = _mm_mullo_epi16(xmm1, xmm0);         //8个16位乘法
        xmm1 = _mm_srai_epi16(xmm1, 7);             //右移7位，相当于除127
        xmm1 = _mm_add_epi16(xmm1, xmm2);           //加法
        xmm1 = _mm_packus_epi16(xmm1, xmm8);        //16个一位紧缩为8位
        _mm_storel_epi64((__m128i*)(pBuffer3 + i * 8), xmm1);   //把运算结果存到临时内存中
    }
    for (int i = nBlocks * 8; i < nMemSize; ++i)
    {
        *(pBuffer3 + i) = uchar((*(pBuffer1 + i) - *(pBuffer2 + i))*alpha + *(pBuffer2 + i));
    }
```

图 5

在计算两幅图像的加权平均时，将两幅图像的八个 RGB 值一次装入 128 位寄存器，然后相减，在乘以权重，在做一次移位和加法操作，得到 16 位的 RGB 值，对 16 位的 RGB 值进行一次紧缩，得到 8 位的 RGB 值，最后将寄存器中的数据返回到相应的内存地址中进行展示。

3.2.3 以上四种方法的性能评测和比较

分别调用以上四种方法对两幅图像（1920x1080）的融合进行测试。在调用函数前后分别计时，以得到各个方法的实际运行时间（单位为秒）。为了消除由计算机状态影响导致的某一次执行时间的不稳定，所以对以上融合效果分别调用 100 次，求其平均作为性能度量的最终依据。实验结果如下：



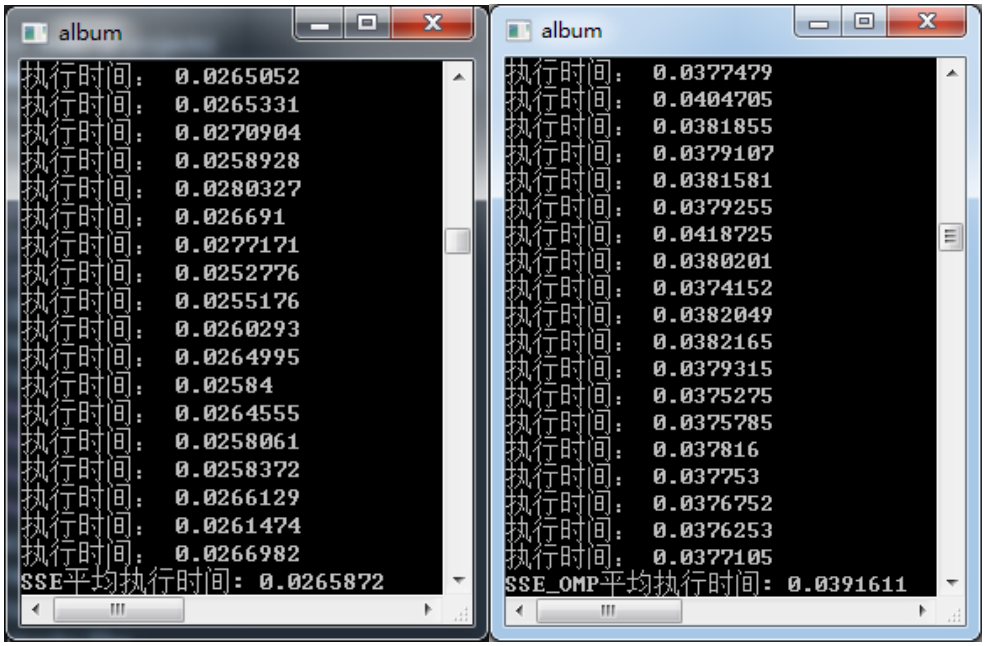


图 6

表 1

使用技术	普通实现 (NORMAL)	循环体改造 (BIGLOOP)	循环体+ OPENMP	SSE
性能提升（以 NORMAL 为基准）	1	-1%	39.2%	27.2%

根据实验结果可以看到，将循环体结构进行改造没有提升性能。使用 SSE 对

图像融合效果进行加速相较于普通方法能明显提高，有 27.2%的性能提升，但是与预期的 8 倍性能提升想去甚远，分析原因是由于其他环节造成了瓶颈，比如需要访存加载和存储数据到 SSE 寄存器需要耗费不少时间。出乎意料的是，使用 openmp 进行多线程加速提升效果最好，达到了 39.2%，分析原因是测试用计算机拥有 8 核处理器，openmp 是基于内核数量进行并行加速的，故效果较好。

3.3 电子相册

本节从程序开发的角度对电子相册的实现进行优化，主要介绍两个方面的工作：

- 使用 UI 选择图片和背景音乐
- 在播放动态电子相册的同时播放背景音乐

3.3.1 使用 UI 选择图片和背景音乐

```
TCHAR szBuffer[MAX_PATH] = {0};
BROWSEINFO bi;
ZeroMemory(&bi, sizeof(BROWSEINFO));
bi.hwndOwner = NULL;
bi.pszDisplayName = szBuffer;
bi.lpszTitle = "选择相册目录";
bi.ulFlags = BIF_RETURNFSANCESTORS;
LPITEMIDLIST idl = SHBrowseForFolder(&bi);
if(NULL == idl)
{
    cout << "异常";
    return 0;
}
SHGetPathFromIDList(idl, szBuffer);
```

图 7

图 7 中代码将打开一个对话框，由用户选择相册文件夹，并将相册文件夹目录名放入 szBuffer 中备用。

```
void getJustCurrentFile(string path, vector<string>& files) {
    //文件句柄
    long hFile = 0;
    //文件信息
    struct _finddata_t fileinfo;
    string p;
    if ((hFile = _findfirst(p.assign(path).append("\\*.bmp").c_str(), &fileinfo)) != -1) {
        do {
            if ((fileinfo.attrib & _A_SUBDIR)) {
            }
            else {
                files.push_back(fileinfo.name);
            }
        } while (_findnext(hFile, &fileinfo) == 0);
        _findclose(hFile);
    }
}
```

图 8

图 8 中代码定义了一个通过路径参数 path 获取该目录下所有 bmp 图片的函数，所有的 bmp 文件的文件名被放入一个 string 向量中返回到调用函数。

3.3.2 在播放动态电子相册的同时播放背景音乐

```
String bgm = path + "\\bgm.wav";
LPCSTR pszSound = bgm.c_str();
PlaySound(pszSound, NULL, SND_FILENAME | SND_ASYNC);
for (int i = 0; i < files.size()-1; i++) {
    Mat img1 = imread(path +"\\"+ files[i]);
    Mat img2 = imread(path +"\\"+files[i+1]);
    fadebysse4andmt(img1, img2);
}
PlaySound(NULL, NULL, SND_FILENAME | SND_ASYNC);
```

图 9

图 9 中代码将路径与 wav 格式的背景音乐名连接得到音乐文件的绝对路径，使用 windows API（PlaySound）函数实现音乐文件的播放，同时指定播放形式为异步，这样就能够在电子相册展示相片的同时播放背景音乐，在相册播放完成后，同样使用 PlaySound 函数停止播放音乐。

使用循环语句依次读取 bmp 图片进行展示，以达到电子相册的效果。

4 结论与展望

从以上的评测和比价可见，SSE 和 openmp 都起到了加速的效果，其中普通循环结合 openmp 加速效果最明显，而使用循环体改造加速效果甚微，甚至起到了负面效果；这启发我们在编程过程中要有意识的编写可以并行处理的代码块，编写出局部性好的代码块，了解机器支持的并行处理指令集，发挥硬件加速的优势，从而开发出高效的软件产品。

图片处理的速度还有优化的空间，一种可行的思路是考虑到相片融合的计算具有可重复利用性，可以事先将每一种 RGB 值融合的结果进行计算，并存储在计算机中，不同的图片进行融合只需要进行查表操作便可以得到融合结果，不再需要计算，从而可以大幅度的减少计算量。