САНКТ–ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО

С. А. МОЛОДЯКОВ
А. В. ПЕТРОВ

АРХИТЕКТУРА ЭВМ
ПРОГРАММИРОВАНИЕ ПЕРИФЕРИЙНЫХ УСТРОЙСТВ

Учебное пособие

Санкт-Петербург
Издательство Политехнического университета
2020

УДК 004.42

Молодяков С. А., Петров А. В. **АРХИТЕКТУРА ЭВМ. ПРОГРАММИРОВАНИЕ ПЕРИФЕРИЙНЫХ УСТРОЙСТВ.** С.А.Молодяков, А.В.Петров, - Изд-во Политехн. ун-та, 2020. – 117 с.

В учебном пособии рассмотрены вопросы разработки программ, предназначенных для работы с системным и периферийным оборудованием ЭВМ. Основное внимание уделено использованию библиотек программ для работы с мультимедийными устройствами (FMod, OpenCV, и др.) и библиотек, обеспечивающих поддержку методов параллельного выполнения команд (OpenMP, Dvec, MPI, CUDA и др.).

Учебное пособие предназначено для студентов, обучающихся по направлениям подготовки 09.03.04 «Программная инженерия» и 02.03.02 «Фундаментальная информатика и информационные технологии», изучающих дисциплины «Архитектура ЭВМ», «Программирование периферийных устройств» и др.

Печатается по решению редакционно-издательского совета Санкт-Петербургского государственного политехнического университета.

Оглавление

Введение

В учебном пособии по лабораторному практикуму рассмотрены вопросы программирования периферийных устройств, создания встраиваемых приложений, использования системных ресурсов компьютера. Представленная информация и примеры программ дают возможность студентам самостоятельно подготовиться к лабораториям, изучить материал в большем объеме, чем предусматривают лабораторные работы.

Программирование периферийных устройств имеет ряд особенностей, одна из которых состоит в использовании большого числа программных элементов, таких как библиотеки программ, функции и утилиты операционных систем, среды разработки, ассемблерные вставки. Полностью освоить все элементы в рамках одной дисциплины не представляется возможным. Поэтому в лабораторном практикуме рассмотрена лишь группа вопросов, связанных с медийными приложениями. Полученная информация и практические навыки низкоуровневого программирования будут полезны не только в последующих дисциплинах, но и при разработке сложных программных проектов.

Представленные лабораторные работы проводятся во втором семестре при изучении дисциплины «Архитектура ЭВМ». Они являются продолжением рассмотрения вопросов первого семестра, связанных с низкоуровневым программированием ЭВМ на языках C++ и Ассемблера. Эти же работы могут быть выполнены и по дисциплине «Программирование периферийных устройств».

Лабораторная работа №1 "BMP конвертор"

Формат BMP (от Bitmap), будучи одним из самых распространенных форматов хранения растровой графической информации, является стандартным для операционных систем Windows. В формате BMP изображение может храниться как без сжатия, так и со сжатием без потерь. Изображения могут быть монохромными (1 бит/пиксел) или цветными (4,8,16,24 или 32 бита/пиксел).

Цель работы: В лабораторной работе предлагается изучить BMP формат. В качестве инструмента при выполнении лабораторной работы можно использовать любую систему программирования, позволяющую считывать и выводить файлы. Дается шаблон программы на языке C++.

Задание на выполнение работы

1. Напишите программу тестирования входного файла. Выведите размеры и число бит на пиксел (8 - 24 бита).

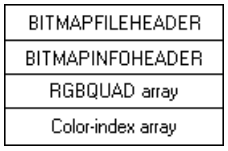
Используйте структуры для тестирования.

2. Напишите программу-конвертор BMP форматов (24-16, 24-8, 24-4 или другое). При бинаризации изображения (24->1) необходимо задание уровня порога. У каждого студента должны быть свои форматы данных

Возможна самостоятельная формулировка других заданий с последующей их оценкой преподавателем. (Преобразование размера файла. Изменение кодирования пикселей в изображении. Подключение таблицы цветов.)

ОПИСАНИЕ BMP ФОРМАТА

Файл в формате BMP состоит из четырех частей [1]: BITMAPFILEHEADER, BITMAPINFOHEADER, RGBQUADS, Pixels.



В BITMAPFILEHEADER и BITMAPINFOHEADER содержатся параметры файла и изображения, в RGBQUADS записывается цветовая палитра, а затем хранятся собственно пикселы изображения (как индексы палитры или как величины красной, зеленой и голубой составляющей цвета).

Формат BITMAPFILEHEADER:

Название поля	Число байт	Комментарий
bfType	2	Тип файла. Должен быть BM.
bfSize	4	Размер файла в байтах
bfReserved1	2	Зарезервировано. Должно быть 0
bfReserved2	2	Зарезервировано. Должно быть 0.
BfOffBits	4	Расстояние в байтах от BITMAPFILEHEADER до пикселов изображения

Формат BITMAPINFOHEADER:

Название поля	Число байт	Комментарий
biSize	4	Размер структуры BITMAPINFOHEADER в байтах
biWidth	4	Ширина изображения в пикселах
biHeight	4	Высота изображения в пикселах
biPlanes	2	Число плоскостей на устройстве вывода. Должно быть 1
biBitCount	**2**	**Число бит на пиксел (1,4,8,16,24,32)**
biCompression	4	Метод хранения пикселов (BI_RGB, BI_RLE8, BI_RLE4, BI_BITFIELDS)
biSizeImage	4	Размер изображения в байтах (Может быть 0, если biCompression=BI_RGB
biXPelsPerMeter	4	Горизонтальное разрешение устройства

		вывода (в пикселах/метр)
biYPelsPerMeter	4	Вертикальное разрешение устройства вывода (в пикселах/метр)
biClrUsed	4	Число цветовых индексов в таблице цветов, которые используются в изображении
biClrImportant	4	Число цветовых индексов, которые считаются важными при выводе изображения

Если величина biHeight положительна, то изображение записано снизу-вверх и начало изображения - левый нижний угол. Если величина biHeight отрицательна, то изображение записано сверху-вниз и начало изображения в левом нижнем углу.

Поле biCompression может принимать следующие значения:

BI_RGB (0)- формат без сжатия.

BI_RLE8 (1)- сжатие длинами серий, 8 бит/пиксел. Каждая запись состоит из 2х байтов, в первом байте хранится число цветовых индексов в серии, во втором байте цветовой индекс.

BI_RLE4 (2) - сжатие длинами серий, 4 бит/пиксел.

BI_BITFIELDS (3) - изображение хранится без сжатия, цветовая таблица состоит из трех четырехбайтовых масок для выделения красной, зеленой и голубой составляющей каждого пиксела. Этот режим используется при 16 и 32 битах/пиксел

RGBQUADS состоит из четверок байт rgbBlue, rgbGreen, rgbRed, rgbReserved, которые определяют голубую, зеленую и красную составляющую цвета. Размер массива RGBQUADS зависит от числа бит на пиксел и метода сжатия.

biBitPerPixel Значения поля:

1 Изображение монохромное. RGBQUADS содержит две четверки, определяющие цветовые компоненты «черных» и «белых» пикселов. В этом случае каждый бит массива задает один пиксел.

4 Изображение содержит до 16 цветов, в RGBQUADS записано до 256 четверок, определяющих палитру изображения.

8 В изображении до 256 цветов. RGBQUADS содержит до 256 четверок, определяющих палитру изображения.

16 До 2^16 цветов. Если biCompression=BI_RGB, то массив RGBQUADS пуст, на каждый пиксел изображения отводится 2 байта, в которых записаны B,G,R цветовые компоненты (5 бит/компоненту, старший бит двухбайтового слова не используется.)

Если biCompression=BI_BITFIELDS, то RGBQUADS состоит из трех четырехбайтовых масок, определяющих R,G,B компоненты.

24 Если biBitPerPixel=24, то массив RGBQUADS пуст и пикселы изображения хранятся в виде троек байт Blue,Green,Red.

32 Аналогично biBitPerPixel=16, только на пиксел отводится 4 байта, три байта на R,G и B, старший байт не используется.

В настоящее время в файлах BMP изображения обычно хранятся без сжатия в формате либо 8 бит/пиксел (с палитрой) либо 24 бит/пиксел. Пикселы изображения хранятся в файле строка за строкой. Представление каждой строки должно быть выравнено на четырехбайтовую границу. Недостающие байты заполняются нулями.

Пример программы

```
#include <windows.h>
#include <iostream>
#include <string>
using namespace std;

void main()
{    string sFileName;
     BITMAPFILEHEADER bmpFileHeader;
    BITMAPINFOHEADER bmpInfoHeader;
    int Width, Height;
    RGBQUAD Palette[256];
    RGBTRIPLE *inBuf;
    BYTE *outBuf;
    HANDLE hInputFile, hOutFile;
    DWORD RW;
     cout << "Enter the full name, please: ";
     cin >> sFileName;
```

```
     hInputFile = CreateFile(sFileName.c_str(), GENERIC_READ,
FILE_SHARE_READ, NULL, OPEN_EXISTING, 0, NULL);
     if (hInputFile == INVALID_HANDLE_VALUE)
          return;

     hOutFile = CreateFile("Result.bmp", GENERIC_WRITE, 0,
NULL, CREATE_NEW, 0, NULL);
    if (hOutFile == INVALID_HANDLE_VALUE)
    {
          CloseHandle (hInputFile);
          return;
     }
     // Считываем инфу
    ReadFile (hInputFile, &bmpFileHeader,
sizeof(bmpFileHeader), &RW, NULL);
    ReadFile (hInputFile, &bmpInfoHeader,
sizeof(bmpInfoHeader), &RW, NULL);

     // Установим указатель на начало растра
    SetFilePointer (hInputFile, bmpFileHeader.bfOffBits, NULL,
FILE_BEGIN);
    Width = bmpInfoHeader.biWidth;
    Height = bmpInfoHeader.biHeight;

    // Выделим память
    inBuf = new RGBTRIPLE [Width];
    outBuf = new BYTE [Width];

    // Заполним заголовки
    bmpFileHeader.bfOffBits = sizeof (bmpFileHeader) + sizeof
(bmpInfoHeader) + 1024;
    bmpInfoHeader.biBitCount = 8;
    bmpFileHeader.bfSize = bmpFileHeader.bfOffBits + Width *
Height + Height * (3*Width % 4);

    // Запишем заголовки
    WriteFile (hOutFile, &bmpFileHeader,
sizeof(bmpFileHeader), &RW, NULL);
    WriteFile (hOutFile, &bmpInfoHeader,
sizeof(bmpInfoHeader), &RW, NULL);
     // Палитра черно-белая
```

```
     for (int i = 0; i < 256; i++)
     {
          Palette[i].rgbBlue = i;
          Palette[i].rgbGreen = i;
          Palette[i].rgbRed = i;
     }
     WriteFile (hOutFile, Palette, 256 * sizeof (RGBQUAD), &RW,
NULL);

    // Начнем преобразовывать
    for (int i = 0; i < Height; i++)
    {
        ReadFile (hInputFile, inBuf, sizeof(RGBTRIPLE) *
Width, &RW, NULL);
        for (int j = 0; j < Width; j++)
            outBuf[j] = 0.3*inBuf[j].rgbtRed +
0.59*inBuf[j].rgbtGreen + 0.11*inBuf[j].rgbtBlue;

        WriteFile (hOutFile, outBuf, sizeof(BYTE) * Width,
&RW, NULL);

        // Пишем мусор для выравнивания
        WriteFile (hOutFile, Palette, (3*Width) % 4, &RW,
NULL);
        SetFilePointer (hInputFile, Width % 4, NULL,
FILE_CURRENT);
    }
    delete[] inBuf;
    delete[] outBuf;
    CloseHandle (hInputFile);
    CloseHandle (hOutFile);
     cout << "Updating has come to the end successfully!";
     system("pause");
}
```

Лабораторная работа №2 "Технологии векторного программирования SIMD"

В 1996 году корпорация Intel внедрила в свои процессоры новую мультимедийную технологию под названием MMX (MultiMedia eXtension), которая давала (со слов фирмы) 400-процентный выигрыш в скорости работы с графикой, звуком и т.п. Особенности MMX: мультимедийный набор 57 команд; регистры MMX MMO - MM7; насыщение при выполнении операций.

Дальнейшее расширение параллельности класса SIMD Single Instruction Multiply Data (одна инструкция - много данных) связано с SSE командами. SSE - Streaming SIMD Extensions. В Pentium реализовано более 70 новых SIMD-инструкций, оперирующих со специальными 128-битными регистрами XMM0-XMM7. Каждый из этих регистров хранит четыре вещественных числа одинарной точности (SSE1). Появились новые команды SSE2- SSE 4.

На следующем этапе развития появились AVX-регистры (256 разрядов) и соответственно AVX-команды. Затем AVX2 (512 разрядов).

Цель работы: В лабораторной работе предлагается изучить и освоить технологии и команды векторного программирования.

Задание на выполнение работы

ЧТО НУЖНО ИЗЧИТЬ

1. Механизм исполнения и особенности команд с векторной (параллельной) обработкой данных.
2. Библиотеки xmmintrin.h , mmintrin.h, immintrin.h.
3. Библиотеку dvec.h и векторные форматы данных.

ЧТО НУЖНО СДЕЛАТЬ

1. Напишите программу «Изучение команд MMX-SSE-AVX», аналогично примеру, но с новыми командами (необходимо использовать минимум две “особенные” команды и команду к AVX-

регистрам). Объясните и покажите в отладчике visual C++ выполнение команд и изменение содержимого регистров.
“Особенными” командами можно считать команды

- с насыщением,
- сравнения,
- перестановок,
- упаковки/ распаковки,
- SSE3 и др.

2. Введите в программу три функции с форматами данных `__m64, __m128, __m256` из библиотек Си (#include <xmmintrin.h>, <mmintrin.h>, <immintrin.h>), которые выполняют операции над описанными ранее массивами типа `char и float.` Покажите в Disassembler-е на регистрах выполнение команд.

`Пример функции:__m64 _mm_add_pi8 (__m64 m1 , __m64 m2);`

Функции у студентов должны отличаться.

3. Напишите программу работы с массивами с использованием их векторного описания и функций библиотеки **dvec**. С применением Disassembler-а определите количество машинных команд, затраченных на реализацию программы.

4. По желанию (на дополнительный плюс). Напишите программу фильтрации (сглаживания) изображения (bmp файл) с использованием средств векторизации и без них (традиционным образом). Сравните время исполнения.

Типы данных

В MMX используется 4 типа данных
- упакованные байты (8 байт в 64-битовом пакете),
- упакованные слова (4 16-битовых слова в 64-битовом пакете),
- упакованные двойные слова (2 32-битовых двойных слова в 64-битовом пакете)
- учетверенное слово (64 бита).

Формат команды

instr [dest, src]

dest – destination; src - source

Суффиксы:

US - команда с непредписанным насыщением (unsigned saturation);

S или SS - команда с предписанным насыщением (signed saturation).

B,W,D,Q - тип данных

Группы команд

Команда	Мнемоника
обмена данными	movd (32), movq (64) movd MM4,mem1
арифметические	padd, … pmadd – слово в двойное слово pmulh– слово – старшая часть pmull– слово – младшая часть
логические	pand, por,pxor,
сдвига	psllw MM4,3 – сдвиг влево psra, psrl - сдвиг вправо с 1/0 или 0
сравнения	pcmpeq - равно, pcmpgt - больше,
преобразования	pack/unpack packuswb M2,M4

Особенности SSE1

В Pentium III реализовано 70 новых SIMD-инструкций, оперирующих со специальными 128-битными регистрами XMM0-XMM7. Каждый из этих регистров хранит четыре вещественных числа одинарной точности. Выполняя операцию над двумя регистрами, SSE фактически оперирует четырьмя парами чисел. Благодаря этому процессор может выполнять до 4-х операций одновременно

Суффиксы: ps - параллельная операция SIMD (maxps xmm1,xmm5)

ss - скалярная операция (операция над 0-31 битами) (addss xmm1,xmm5)

Пример операции перестановки: shufps xmm1,xmm2,9Ch 10(b2)-01(b1)-11(a3)-00(a0) in xmm1

SSE1-команды над числами с плавающей точкой

- Команды пересылки
  - Скалярные типы – MOVSS
  - Упакованные типы – MOVAPS, MOVUPS, MOVLPS, MOVHPS, MOVLHPS, MOVHLPS
- Арифметические команды
  - Скалярные типы – ADDSS, SUBSS, MULSS, DIVSS, RCPSS, SQRTSS, MAXSS, MINSS, RSQRTSS
  - Упакованные типы – ADDPS, SUBPS, MULPS, DIVPS, RCPPS, SQRTPS, MAXPS, MINPS, RSQRTPS
- Команды сравнения
  - Скалярные типы – CMPSS, COMISS, UCOMISS
  - Упакованные типы – CMPPS
- Перемешивание и распаковка
  - Упакованные типы – SHUFPS, UNPCKHPS, UNPCKLPS
- Команды для преобразования типов
  - Скалярные типы – CVTSI2SS, CVTSS2SI, CVTTSS2SI
  - Упакованные типы – CVTPI2PS, CVTPS2PI, CVTTPS2PI
- Битовые логические операции
  - Упакованные типы – ANDPS, ORPS, XORPS, ANDNPS

Команды над целыми числами

- Арифметические команды
  - PMULHUW, PSADBW, PAVGB, PAVGW, PMAXUB, PMINUB, PMAXSW, PMINSW
- Команды пересылки
  - PEXTRW, PINSRW
- Другие

SSE2

У микропроцессора Intel Pentium 4 появилась новая группа инструкций, получивших название SSE2, она дополнила все предыдущие команды MMX SSE. Цифра 2 в названии указывается для того, чтобы отличить новую группу от одноименной, уже поддерживаемой микропроцессором Pentium III. В состав SSE2 входят операции для работы с 64-х разрядными целыми и вещественными (представление с удвоенной точностью) числами. Согласно документации Intel в группу SSE2 входят инструкции, выполняющие 144 новые операции.

Большинство новых инструкций двухадресные. Первый операнд является приемником (dest), а второй источником (src). Приемник, как правило, находится в 128-bit регистре xmm, источник может находиться как в регистре xmm, так и в оперативной памяти (ОЗУ). Исключением являются только инструкции пересылки, у которых приемник может располагаться в ОЗУ. Третий операнд, если он есть, является целым числом, размер которого не превышает одного байта.

При работе с вещественными числами возможно выполнение одной и той же операции над одной или двумя парами чисел. Появляются новые суффиксы.

pd - параллельная операция SIMD, два 64-bit числа расположены в одном 128-bit регистре или в ОЗУ подряд друг за другом.

sd - скалярная операция (операция над 0-63 битами)

SSE2-команды

SQRTPD xmm, xmm/m128 dest = sqrt(src)	- извлечение квадратного корня из двух вещественных чисел источника с записью результата в приемник
MAXPD xmm, xmm/m128 dest = max(dest,src)	- нахождение в каждой паре большего вещественного числа удвоенной точности
MINPD xmm, xmm/m128 dest = min(dest,src)	нахождение в каждой паре меньшего вещественного числа удвоенной точности

SSE3

Набор SSE3 содержит 13 инструкций. Наиболее заметное изменение - возможность горизонтальной работы с регистрами. Если говорить более конкретно, добавлены команды сложения и вычитания нескольких значений, хранящихся в одном регистре. Эти команды упростили ряд DSP и 3D-операций. Существует также новая команда для преобразования значений с плавающей точкой в целые без необходимости вносить изменения в глобальном режиме округления.

Инструкции SSE3

- ADDSUBPD (Add Subtract Packed Double).
- ADDSUBPS (Add Subtract Packed Single).
- HADDPD (Horizontal Add Packed Double).
- HADDPS (Horizontal Add Packed Single).
- HSUBPD (Horizontal Subtract Packed Double).
- HSUBPS (Horizontal Subtract Packed Single).
- FISTTP — преобразование вещественного числа в целое с сохранением целочисленного значения и округлением в сторону нуля.
- LDDQU — загрузка 128bit невыровненных данных из памяти в регистр xmm, с предотвращением пересечения границы строки кеша.

SSE4

SSE4 состоит из 54 инструкций. Добавлены инструкции обработки строк 8/16 битных символов, вычисления CRC32 и др.

Инструкции SSE4.1

1. Ускорение видео
2. Векторные примитивы
3. Вставки/извлечения
4. Скалярное умножение векторов
5. Смешивания
6. Проверки бит
7. Округления
8. Чтение WC памяти

Инструкции SSE4.2

1. Обработка строк

2. Подсчет CRC32
3. Подсчет популяции единичных бит
4. Векторные примитивы
5. Процессоры с SSE4

PHMINPOSUW xmm1, xmm2/m128 — *(Packed Horizontal Word Minimum)*

- Input — { A_0, A_1,… A_7 }
- Output — { MinVal, MinPos, 0, 0… }

Поиск среди 16-ти битных беззнаковых полей A_0…A_7 такого, который имеет минимальное значение (и позицию с меньшим номером, если таких полей несколько). Возвращается 16-ти битное значение и его позиция.

Некоторые функции MMX-SSE

Packed Arithmetic Intrinsics

Intrinsic name	Operation	Signed	Argument and result values/bits	Corresponding instruction
_mm_add_pi8	Adds	Not applicable	8/8, 8/8	PADDB
_mm_add_pi16	Adds	Not applicable	4/16, 4/16	PADDW
_mm_add_pi32	Adds	Not applicable	2/32, 2/32	PADDD
_mm_adds_pi8	Adds	Yes	8/8, 8/8	PADDSB
_mm_adds_pi16	Adds	Yes	4/16, 4/16	PADDSW
_mm_adds_pu8	Adds	No	8/8, 8/8	PADDUSB
_mm_adds_pu16	Adds	No	4/16, 4/16	PADDUSW
_mm_sub_pi8	Subtracts	Not applicable	8/8, 8/8	PSUBB
_mm_sub_pi16	Subtracts	Not applicable	4/16, 4/16	PSUBW

http://msdn.microsoft.com/en-us/library/8cs7e4zs(v=vs.80).aspx

Advanced Vector Extensions (AVX)

Ширина векторных регистров SIMD увеличивается со 128 (XMM) до 256 бит (регистры YMM0 - YMM15). Существующие 128-битные SSE-инструкции будут использовать младшую половину новых YMM-регистров, не изменяя старшую часть. Для работы с YMM-регистрами добавлены новые 256-битные AVX-инструкции. AVX2 (AVX512) расширение векторных регистров SIMD до 512 . (в будущем до 1024) бит.

Набор AVX-инструкций использует трёхоперандный синтаксис (Неразрушающие операции). Например, вместо $a=a+b$ можно использовать $c=a+b$, при этом регистр a остаётся неизмененным. В случаях, когда значение a используется дальше в вычислениях, это повышает производительность, так как избавляет от необходимости сохранять перед вычислением и восстанавливать после вычисления регистр, содержавший a, из другого регистра или памяти.

Набор инструкций AVX содержит в себе аналоги 128-битных SSE инструкций для вещественных чисел. При этом, в отличие от оригиналов, сохранение 128-битного результата будет обнулять старшую половину YMM регистра. 128-битные AVX-инструкции сохраняют прочие преимущества AVX, такие, как новая схема кодирования, трехоперандный синтаксис и невыровненный доступ к памяти. Пример параллельного умножения:

```
vmovapd ymm0, [esi]
vmulpd ymm0, ymm0, [edx]
vmovapd [edi], ymm0
```

Расширение AVX-512 вводит 32 векторных регистра (ZMM), каждый по 512 бит, 8 регистров масок, 512-разрядные упакованные форматы для целых и дробных чисел и операции над ними, тонкое управление режимами округления (позволяет переопределить глобальные настройки), операции broadcast (рассылка информации из одного элемента регистра в другие), подавление ошибок в операциях с дробными числами, операции gather/scatter (сборка и рассылка

элементов векторного регистра в/из нескольких адресов памяти), быстрые математические операции, компактное кодирование больших смещений. AVX-512 предлагает совместимость с AVX, в том смысле, что программа может использовать инструкции как AVX, так и AVX-512 без снижения производительности. Регистры AVX (YMM0-YMM15) отображаются на младшие части регистров AVX-512 (ZMM0-ZMM15), по аналогии с SSE и AVX регистрами.

ПРИМЕР программы: Изучение команд MMX-SSE-AVX

```
#include <stdio.h>
#include <conio.h>
int main(void) {
     char qw1[8] = {1, 1, 1, 1, 1, 1, 1, 1};
     char qw2[8] = {2, 2, 2, 2, 2, 2, 2, 2};
     int a = 1; int b = 2;
     float c[4] = {1, 2, 3, 4};
     float d[4] = {5, 6, 7, 8};
     double f[2]= {16, 4};
     char a128[16] = {1, 18, 3, 19, 5, 21, 7, 23, 9, 25, 11,
27, 13, 29, 15, 31};
     char b128[16] = {17, 2, 19, 4, 21, 6, 23, 8, 25, 10, 27,
12, 29, 14, 31, 16};
     _asm {    //mmx
          movq mm0, qw1
          movq mm1, qw2
          pcmpeqb   mm0, mm1
          movq qw1, mm0
     }
     printf("%s\n", "Summing elements of vectors qw1 + qw2 :");
     for (int i = 0; i < 8; i++)
     { printf("%d ", qw1[i]); }
     printf("\n");

     _asm {    //sse
          movups    xmm0, c
          movups    xmm1, d
          addps     xmm0, xmm1
          movups    c, xmm0
     }
```

```
    printf("%s\n", "Summing elements of vectors c + d :\n");
    for (int i = 0; i < 4; i++)
    { printf("%f ", c[i]); }

    _asm {    //sse2
         movups xmm1, f
         sqrtpd xmm0, xmm1
         movups f, xmm0
    }
    printf("\n%s %f %s %f\n", "Square of ", f[0], "is", f[1]);

    _asm {
         movups    xmm0, a128
         movups    xmm1, b128
         pminub    xmm0, xmm1
         movups    a128, xmm0
    }
    printf("\n%s\n", "Comparing elements :");
    for (int i = 0; i<16; i++)
    { printf("( %d , %d) ; ", a128[i], b128[i]); }
    printf("\n%s\n", "Minimum elements :");
    for (int i = 0; i<16; i++)
    { printf("%d ", a128[i]); }
    return 0;
    getch(0);
}
```

Лабораторная работа №3
"Программирование для многопоточных приложений. OpenMP "

Для работы в многопоточных мультипроцессорных системах необходимы программные средства создания и сопровождения потоков команд. Такими средствами наряду с возможностями операционной системы могут быть функции OpenMP и MPI [2].

OpenMP (Open Multi-Processing) задуман как стандарт для программирования в системах с парадигмой общей памяти. Он был разработан в 1997 г. как API ориентированный для написания портируемых многопоточных приложений. В OpenMP входят спецификации набора директив компилятору, процедур и переменных среды. Разработчик не создает новую параллельную программу, а просто добавляет в текст последовательной программы OpenMP-директивы. При этом система программирования OpenMP предоставляет разработчику большие возможности по контролю над поведением параллельного приложения.

Следует выделить версии OpenMP, начиная с OpenMP 4.0. Они расширяют возможности использования методов параллелизации в направлении применения векторного программирования с использованием SIMD-команд. Часть новых директив OpenMP позаимствованы из Intel CilkPlus. В частности появилась директива #pragma omp simd. OpenMP 4.0 и выше используются в Intel Parallel Studio.

https://pro-prof.com/archives/4335 https://parallel.ru/tech/tech_dev/openmp.html

В 1994 г. был принят стандарт механизма передачи сообщений MPI (Message Passing Interface) MPI - это библиотека функций, обеспечивающая взаимодействие параллельных процессов с помощью механизма передачи сообщений.

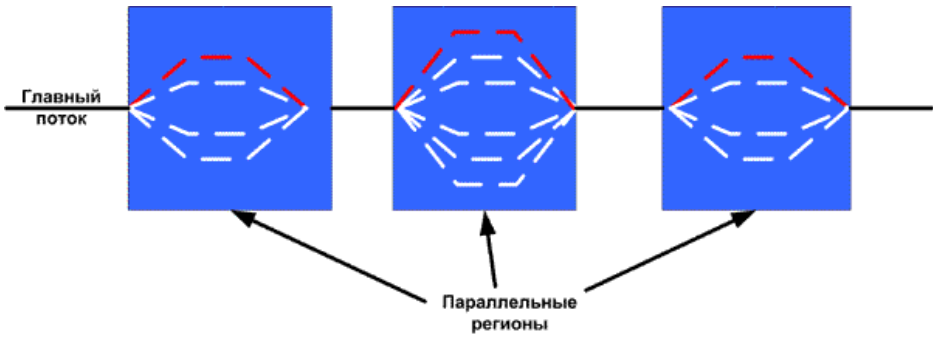
Цель работы: В лабораторной работе предлагается изучить основные директивы OpenMP. Освоить утилиты, связанные с многопоточностью.

Задание на выполнение работы

1. Загрузите тестовую программу. Не забудьте активизировать директивы OpenMP на страницах свойств проекта, выбрав Configuration Properties, C/C++, Language и изменив значение свойства OpenMP Support.
2. Экспериментально подтвердите параллельную работу **двух потоков** при использовании директив *for, sections*. Подтвердите, используя директиву *OMP_GET_NUM_*THREADS. Объясните и покажите использование директивы *barrier*.
3. Напишите многопотоковую программу с применением двух директив (`single, critical, master, atomic, ordered…`) синхронизации потоков (директивы синхронизации у студентов должны отличаться). Программа может выводить на экран номер потока или заполнять массив из двух потоков с использованием директив синхронизации.
4. Напишите многопотоковую программу с применением синхронизации на базе замков *OMP_INIT_LOCK(var), OMP_SET_LOCK, OMP_UNSET_LOCK, OMP_TEST_LOCK.*

Параллельные регионы в OpenMP

Работа OpenMP-приложения начинается с единственного потока — основного. В приложении могут содержаться параллельные регионы, входя в которые, основной поток создает группы потоков (включающие основной поток). В конце параллельного региона группы потоков останавливаются, а выполнение основного потока продолжается. В параллельный регион могут быть вложены другие параллельные регионы, в которых каждый поток первоначального региона становится основным для своей группы потоков. Вложенные регионы могут в свою очередь включать регионы более глубокого уровня вложенности.



Типы директив и функций

1. Директивы реализации параллельной обработки блоков команд
2. Функции изменения и получения параметров исполняющей среды
3. Директивы и функции блокировки/синхронизации
4. Общие и частные данные

OpenMP включает лишь два базовых типа конструкций: директивы pragma и функции исполняющей среды OpenMP. Директивы pragma, как правило, указывают компилятору реализовать параллельное выполнение блоков кода. Все эти директивы начинаются с #pragma omp. Как и любые другие директивы pragma, они игнорируются компилятором, не поддерживающим конкретную технологию — в данном случае OpenMP.

Функции OpenMP служат в основном для изменения и получения параметров среды. Кроме того, OpenMP включает API-функции для поддержки некоторых типов синхронизации. Чтобы задействовать эти функции библиотеки OpenMP периода выполнения (исполняющей среды), в программу нужно включить заголовочный файл omp.h. Если вы используете в приложении только OpenMP-директивы pragma, включать этот файл не требуется.

Директивы pragma имеют следующий формат:

#pragma omp <директива> [раздел [[,] раздел]...]

Формат директивы на C/C++:

<команда для препроцессора> <имя директивы>
<предложение(klausa)>
#pragma omp parallel [clause clause ...] { . . . }

Состав директив и их краткая характеристика приведены в следующей таблице:

Директива	Описание
parallel [параметры]	Директива имеет декларативный характер и не управляет действиями, выполняемыми в параллельном регионе. Она нужна, например, в тех случаях, когда для распараллеливания региона используется несколько директив формирующих нити или выполняющих другие действия.
for [параметры]	Формирует нити, содержащие копии ассоциированного с директивой цикла типа for. Каждая копия будет выполнять свою часть от общего числа итераций, описанных в исходном операторе цикла.
sections [параметры]	Формирует параллельный регион из блоков, расположенных в исходном тексте программы последовательно друг за другом. Перед каждым преобразуемым блоком указывается директива section.
section	Вспомогательная директива, используется только в области действия директивы sections для формирования нитей из ассоциированных блоков.
single [параметры]	Указывает на то, что в регионе должна выполняться только одна нить, содержащая ассоциированный с директивой блок. Такая нить может, например, изменять значения частных переменных, используемых другими нитями региона.
parallel for [параметры]	Сокращенная форма записи для создания параллельного региона, содержащего единственную директиву for. Сочетание двух директив увеличивает количество доступных параметров.
parallel sections[параметры]	Сокращенная форма записи для создания параллельного региона, содержащего единственную директиву sections. Сочетание двух директив

	увеличивает количество доступных параметров.
master	Ассоциированный с директивой блок преобразуется в основную нить, с которой начинается выполнение задачи. Выполнение основной нити продолжается до тех пор, пока не встретится первая распараллеливаемая конструкция.
critical [имя секции]	Критические секции нужны для разграничения доступа к общему ресурсу, например, к памяти. Все нити параллельного региона ждут завершения выполнения критической секции. Если есть несколько критических секций, то им надо присвоить уникальные имена.
barrier	Указывает точку, в которой организуется ожидание окончания исполнения всех нитей параллельного региона. По умолчанию (если не указан параметр nowait) директивы for, sections и single устанавливают барьер в нужной точке региона.
atomic	Директива действует только на один оператор присваивания. При каждом его выполнении новое значение переменной, указанной в левой части принудительно сохраняется в памяти. Это позволяет исключить возможные ошибки при работе с одной переменной в нескольких нитях параллельного региона.
flush[список переменных]	Только указанные в списке, или по умолчанию все общие переменные подвергаются операции "выравнивание" (flush). При этом из кеш в основную память переписываются переменные, значения которых были изменены. Это же касается и переменных находящихся в регистрах процессоров.
threadprivate(список переменных)	Объявляет частными в нитях параллельного региона переменные, описанные во внешнем блоке. Директива указывается во внешнем блоке сразу

	после описания соответствующих переменных и не влияет на работу с ними вне параллельного региона. См. copyin.
ordered	Используется в сфере действия директивы for для выделения блока, в котором повторы цикла будут происходить в естественном порядке (как при обычных последовательных вычислениях). У директивы for должен быть указан одноименный ключ ordered.

Предполагается, что в SMP-системе нити будут распределены по различным процессорам (однако это, как правило, находится в ведении операционной системы). Каким образом между порожденными нитями распределяется работа - определяется директивами DO,SECTIONS и SINGLE. Возможно также явное управление распределением работы (а-ля MPI) с помощью функций, возвращающих номер текущей нити и общее число нитей. По умолчанию (вне этих директив), код внутри PARALLEL исполняется всеми нитями одинаково.

clause: schedule(type[,chink]) ordered private(list) shared(list) firstprivate(list) lastprivate(list) reduction(operator:list) nowait - Определяет параллельный цикл. Клауза schedule определяет способ распределения итераций по нитям: static,m - статически, блоками по m итераций dynamic,m - динамически, блоками по m (каждая нить берет на выполнение первый еще невзятый блок итераций) guided,m - размер блока итераций уменьшается экспоненциально до величины m runtime - выбирается во время выполнения. ordered – (для циклов) реализуется последовательное выполнение витков цикла, как в последовательном алгоритме. lastprivate(list) – переменным присваивается результат последнего витка цикла. По умолчанию, в конце цикла происходит неявная синхронизация; эту синхронизацию можно запретить с помощью nowait

Runtime-процедуры и переменные среды

В целях создания переносимой среды запуска параллельных программ, в OpenMP определен ряд переменных среды, контролирующих поведение приложения.

В OpenMP предусмотрен также набор библиотечных процедур, которые позволяют:

* во время исполнения контролировать и запрашивать различные параметры, определяющие поведение приложения (такие как число нитей и процессоров, возможность вложенного параллелизма); процедуры назначения параметров имеют приоритет над соотвествующими переменными среды.

* использовать синхронизацию на базе замков (locks).

Переменные среды

OMP_SCHEDULE Определяет способ распределения итераций в цикле, если в директиве DO использована клауза SCHEDULE(RUNTIME).

OMP_NUM_THREADS Определяет число нитей для исполнения параллельных областей приложения.

OMP_DYNAMIC Разрешает или запрещает динамическое изменение числа нитей.

OMP_NESTED Разрешает или запрещает вложенный параллелизм.

Процедуры для контроля/запроса параметров среды исполнения

OMP_SET_NUM_THREADS Позволяет назначить максимальное число нитей для использования в следующей параллельной области (если это число разрешено менять динамически). Вызывается из последовательной области программы.

OMP_GET_MAX_THREADS Возвращает максимальное число нитей.

OMP_GET_NUM_THREADS Возвращает фактическое число нитей в параллельной области программы.

OMP_GET_NUM_PROCS Возвращает число процессоров, доступных приложению.

OMP_IN_PARALLEL Возвращает .TRUE., если вызвана из параллельной области программы.

OMP_SET_DYNAMIC / OMP_GET_DYNAMIC
Устанавливает/запрашивает состояние флага, разрешающего динамически изменять число нитей.

OMP_GET_NESTED / OMP_SET_NESTED
Устанавливает/запрашивает состояние флага, разрешающего вложенный параллелизм.

Процедуры для синхронизации на базе замков

В качестве замков используются общие переменные типа INTEGER (размер должен быть достаточным для хранения адреса). Данные переменные должны использоваться только как параметры примитивов синхронизации.

OMP_INIT_LOCK(var) / OMP_DESTROY_LOCK(var)
Инициализирует замок, связанный с переменной var.

OMP_SET_LOCK Заставляет вызвавшую нить дождаться освобождения замка, а затем захватывает его.

OMP_UNSET_LOCK Освобождает замок, если он был захвачен вызвавшей нитью.

OMP_TEST_LOCK Пробует захватить указанный замок. Если это невозможно, возвращает .FALSE.

Классы переменных (PRIVATE,SHARED,REDUCTION, etc.).

Private(list) – список переменных локальных в каждой нити; shared(list) - список переменных общих для каждой нити; firstprivate(list) - список переменных, которые становятся локальными в каждой нити со значениями, ранее присвоиными этим переменным; copyin(list) - список переменных (массивов), которые определены #pragma omp threadprivate(list), и которые создаются в каждой нити; reduction(operator:list) - список переменных, с которыми выполняются операции обобщенно по всем нитям. list: - список переменных.

Пример программы

```
#include <omp.h>
#include <iostream>
#include <windows.h>
#include <stdio.h>
#include <conio.h>
using namespace std;
void func()
      {   for(int i= 0; i < 500000; i++)
  rand();}
int main()
{omp_set_num_threads(2);
 #pragma omp parallel for
      for (int i= 0; i < 100; i++)
 {        cout << 0; func();   }
#pragma omp parallel
{#pragma omp sections nowait
{#pragma omp section
      for (int i= 0; i < 10; i++)
 {        cout << 1; func();   }
#pragma omp section
          for (int i= 0; i < 20; i++)
 {        cout << 2; func();
 }}
#pragma omp barrier
      for (int i= 0; i < 10; i++)
 {        cout << 3;
  func();
 }}
```

Лабораторная работа №4

"3D звук. Библиотека FMod"

Библиотека функций FMOD представляет собой реализацию API верхнего уровня, который включает широкий набор функций для работы со звуковыми файлами различных форматов, обработки звуковых данных и воспроизведения звука через аудиосистему компьютера. Интерес представляют функции для работы с объемным (3D) звуком.

Поставляется в двух версиях — 3.75, и 4.x и поддерживает большинство форматов и платформ. Поддерживаемые платформы:

Win32, Win64, Linux 32-bit, 64-bit, Mac OS X, iPhone, Android

Задание на выполнение работы

ЧТО НУЖНО ИЗЧИТЬ

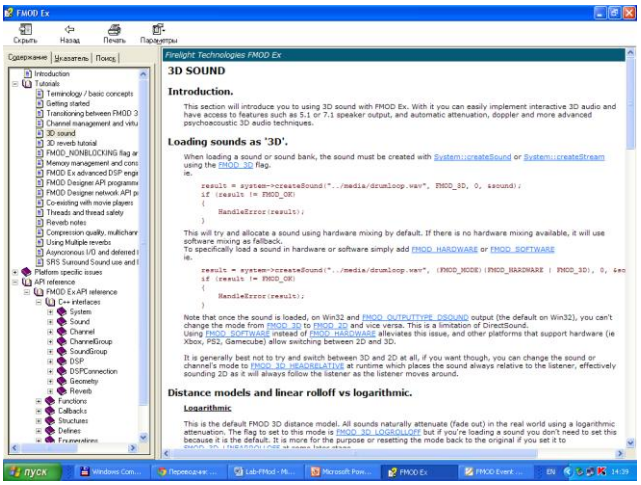
1. Программирование с использованием библиотеки **FMOD**.
2. 3D-звук

ЧТО НУЖНО СДЕЛАТЬ

1. Загрузите и запустите программу, которая проигрывает *.mp3 файл с использованием библиотеки FMOD 4.x.
2. Познакомьтесь с 3D-моделью проигрывания звука и с набором 3D-функций из библиотеки FMOD 4. Используйте описание FMOD API, установленное на компьютере. (Version 4.44.11 Built on Mar 26, 2013) (см. рисунок)
3. Напишите программу проигрывания файла со звуковым 3D-эффектом перемещения звука по объему.
4. Напишите программу проигрывания файла со звуковым эффектом. Звуковые эффекты у студентов должны отличаться.

Список электронных ресурсов

1. Запись звука с использованием API FMOD
http://www.tiflocomp.ru/games/design/sound_games/fmod_rec.php
2. Основной сайт www.FMOD.org.



Программирование. 3D-модель. Основные функции.

1. Первая программа проигрывания файла

(Нужны fmodex.dll и папки inc и lib)

```
    #include "stdafx.h"
    #include <conio.h>
    #include <windows.h>
    #include "inc\fmod.hpp"
    #include "inc\fmod_errors.h"
    #include <iostream>
    #pragma comment(lib, "fmodex_vc.lib")
    using namespace std;

    int _tmain(int argc, _TCHAR* argv[])
    { // FMOD 4
    FMOD::System * system;
    FMOD::System_Create(&system);
    system->init(16, FMOD_INIT_NORMAL, 0);
    FMOD::Sound * sound; // sound
    FMOD::Channel * channel; // sound channel
    system->createSound("jules.mp3",       FMOD_SOFTWARE       |
FMOD_LOOP_OFF, 0, &sound); // creating sound
    system->playSound(FMOD_CHANNEL_FREE,      sound,      false,
&channel);// playing sound (assigning it to a channel)
    channel->setPaused(false); // actually play sound
    getch();
```

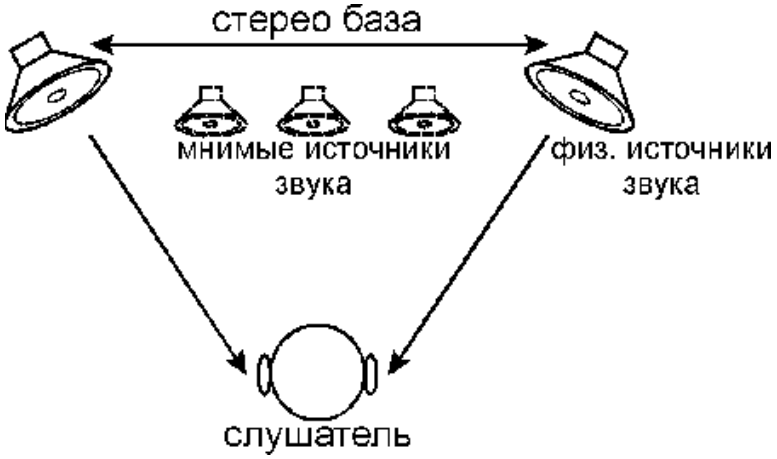
2. 3D-модель

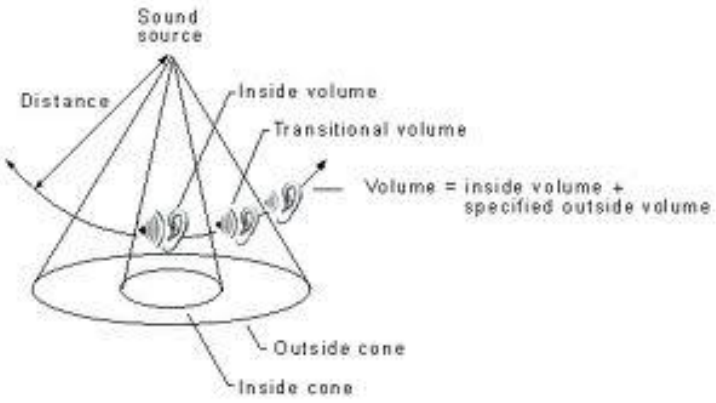
Любая точка в пространстве задается своими координатами, которые записываются в последовательности X, Y, Z. Моделируется набор звуковых эффектов. Используется левосторонняя декартова система координат, состоящая из трех ортогональных координатных осей. Ось X направлена вправо; ось Y направлена вверх; ось Z направлена вперед (то есть в монитор, если сидеть лицом к нему). Расстояние измеряется в метрах. Координаты могут принимать как положительные, так и отрицательные значения.

Координатную триаду XYZ, характеризующую положение точки в пространстве, можно рассматривать как вектор, начало которого находится в начале отсчета, то есть в точке с координатами (0, 0, 0), и конечной точкой с координатами (X, Y, Z).

Кроме векторов положения, в FMOD используются векторы скорости, необходимые для вычисления доплеровского смещения в спектре звука движущихся источников. Вектор скорости задается тремя координатами своей конечной точки (начальной точкой вектора скорости считается точка (0, 0, 0)).

FMOD рассчитывает статическую звуковую картину, которую программист может сделать динамичной для слушателя, меняя положение источников звука.





3D источник звука является каналом, который имеет положение и скорость. Когда канал 3D играет, его громкость, расположение колонок и шаг будут зависеть автоматически от положения слушателя.

Слушатель имеет положение, скорость как у источника звука, но он также имеет ориентацию. Громкость определяется расстоянием между слушателем и источником. Скорость перемещения источника относительно слушателя определяется эффектом Доплера. Ориентации слушателя к источнику определяет панорамирование или размещение динамиков.

3D звук может вызвать любая функция со словом 3D в имени функции.

Некоторые эффекты, поддерживаемые в FMOD:

1. Реверберация — процесс постепенного уменьшения интенсивности звука при его многократных отражениях. Иногда под реверберацией понимается имитация данного эффекта с помощью реверберatoров.

2. Эхо - отражённый звук. Обычно эхо замечают, если слышат также прямой звук от источника, когда в одной точке пространства можно несколько раз услышать звук из одного источника, пришедший по прямому пути и отражённый (возможно несколько раз) от окружающих предметов.

3. Тремоло - многократное быстрое повторение одного звука либо быстрое чередование 2 несоседних звуков, 2 созвучий (интервалов, аккордов), отдельного звука и созвучия.

и др.

3. Основные функции для работы со звуком

System::createSound (Loads a sound into memory, or opens it for streaming.)

C++ Syntax

```
FMOD_RESULT System::createSound(
  const char *  name_or_data,
  FMOD_MODE  mode,
  FMOD_CREATESOUNDEXINFO *  exinfo,
  FMOD::Sound **  sound
);
```

C Syntax

```
FMOD_RESULT FMOD_System_CreateSound(
  FMOD_SYSTEM *  system,
  const char *  name_or_data,
  FMOD_MODE  mode,
  FMOD_CREATESOUNDEXINFO *  exinfo,
  FMOD_SOUND **  sound
);
```

FSOUND_GetVersion() - возвращает версию библиотеки FMOD, установленной на компьютере. Возвращаемое значение следует сравнить с константой FMOD_VERSION, которая хранит номер версии FMOD, для которой была скомпилирована программа.

FSOUND_SetOutput () / FSOUND_GetOutput () - выбрать/получить базовую звуковую систему (Windows Multimedia, DirectSound, A3D и т.п.).

FSOUND_SetDriver () / FSOUND_GetDriver () - выбрать/получить номер устройства вывода (звуковой карты).

FSOUND_SetMixer () /FSOUND_GetMixer () - выбрать/получить тип цифрового микшера.

FSOUND_Init() - инициализирует звуковую систему FMOD.

FSOUND_Sample_Load () - загружает в память и декодирует звуковой файл (поддерживаются .wav, .mp2, .mp3, .ogg, .raw и др.).

FSOUND_PlaySoundEx () - проигрывает звуковой файл, загруженный в память, через звуковой канал.

FSOUND_SetPaused () - приостанавливает / возобновляет воспроизведение звука в канале.

4. Некоторые функции для работы с 3d звуком

FSOUND_3D_SetDistanceFactor () - позволяет установить единицы измерения длин, отличные от метров.

FSOUND_3D_SetDopplerFactor () - позволяет установить доплеровское смещение. Базовое значение (1.0) соответствует скорости звука 340 м/с.

FSOUND_3D_SetRolloffFactor () - позволяет установить уровень потерь энергии звуковой волны (затухания).

FSOUND_3D_SetAttributes () / FSOUND_3D_GetAttributes () - установить/получить вектор положения и вектор скорости источника звука.

FSOUND_3D_Listener_SetAttributes () / FSOUND_3D_Listener_GetAttributes () - установить/ получить вектор положения, вектор скорости и векторы ориентации слушателя.

FSOUND_3D_SetMinMaxDistance () / FSOUND_3D_GetMinMaxDistance () установить / получить минимальное и максимальное расстояние слышимости источника звука. Минимальное расстояние от источника звука до слушателя - при уменьшении которого громкость звука больше не возрастает, а остается на том значении, которого она достигла на минимальном расстоянии. Устанавливая разные минимальные расстояния, например, для самолета и шмеля, можно сделать их одинаково заметными на слух, несмотря на то, что гул мотора будет

восприниматься как более мощный звук. Максимальным называется такое расстояние от источника звука до слушателя, начиная с которого громкость звука больше не уменьшается, а остается на уровне, который она достигла на максимальном расстоянии. Это означает, что как бы далеко не находился источник звука, он будет слышен.

5. Примеры из описания FMOD 4.x

Loading sounds as '3D'.

When loading a sound or sound bank, the sound must be created with System::createSound or System::createStream using the FMOD_3D flag. ie.

```
    result = system->createSound("../media/drumloop.wav",
FMOD_3D, 0, &sound);
    if (result != FMOD_OK)
    {
        HandleError(result);
    }
```

Set the default ambient reverb

In this section will we look at setting the default ambient reverb settings. This is important, as FMOD Ex will use these settings when the listener is not standing within an area affected by any of the 3D reverbs. In this example, we will use the reverb present FMOD_PRESET_OFF.

FMOD_REVERB_PROPERTIES prop1 = FMOD_PRESET_OFF;
system->setReverbAmbientProperties(&prop1);

Create a 3D Reverb

We will now create a virtual reverb, using the call System::createReverb, then set the characteristics of the reverb using Reverb::setProperties.

```
FMOD::Reverb  *reverb;
result = system->createReverb(&reverb);
FMOD_REVERB_PROPERTIES prop2 = FMOD_PRESET_CONCERTHALL;
reverb->setProperties(&prop2);
```

Set 3D Attributes

The 3D attributes of the reverb must now be set. The method Reverb::set3DAttributes allows us to set the origin position, as well as the area of coverage using the minimum distance and maximum distance.

```
FMOD_VECTOR pos = { -10.0f, 0.0f, 0.0f };
float mindist = 10.0f;
float maxdist = 20.0f;
reverb->set3DAttributes(&pos, mindist, maxdist);
```

As the 3D reverb uses the position of the listener in its weighting calculation, we also need to ensure that the location of the listener is set using System::set3dListenerAtrributes.

```
FMOD_VECTOR  listenerpos  = { 0.0f, 0.0f, -1.0f };
system->set3DListenerAttributes(0, &listenerpos, 0, 0, 0);
```

FMOD_DSP_ECHO

Parameter types for the FMOD_DSP_TYPE_ECHO filter.

Enumeration

```
typedef enum {
  FMOD_DSP_ECHO_DELAY,
  FMOD_DSP_ECHO_DECAYRATIO,
  FMOD_DSP_ECHO_MAXCHANNELS,
  FMOD_DSP_ECHO_DRYMIX,
  FMOD_DSP_ECHO_WETMIX
} FMOD_DSP_ECHO;
```

Values

```
FMOD_DSP_ECHO_DELAY
Echo delay in ms. 10 to 5000. Default = 500.
FMOD_DSP_ECHO_DECAYRATIO
Echo decay per delay. 0 to 1. 1.0 = No decay, 0.0 = total
decay (ie simple 1 line delay). Default = 0.5.
FMOD_DSP_ECHO_MAXCHANNELS
Maximum channels supported. 0 to 16. 0 = same as fmod's
default output polyphony, 1 = mono, 2 = stereo etc. See
remarks for more. Default = 0. It is suggested to leave at 0!
FMOD_DSP_ECHO_DRYMIX
Volume of original signal to pass to output. 0.0 to 1.0.
Default = 1.0.
FMOD_DSP_ECHO_WETMIX
Volume of echo signal to pass to output. 0.0 to 1.0. Default =
1.0.
```

FMOD_DSP_TREMOLO

Parameter types for the FMOD_DSP_TYPE_TREMOLO filter.

Enumeration

```
typedef enum {
  FMOD_DSP_TREMOLO_FREQUENCY,
  FMOD_DSP_TREMOLO_DEPTH,
  FMOD_DSP_TREMOLO_SHAPE,
  FMOD_DSP_TREMOLO_SKEW,
  FMOD_DSP_TREMOLO_DUTY,
  FMOD_DSP_TREMOLO_SQUARE,
  FMOD_DSP_TREMOLO_PHASE,
  FMOD_DSP_TREMOLO_SPREAD
} FMOD_DSP_TREMOLO;
Values
FMOD_DSP_TREMOLO_FREQUENCY
LFO frequency in Hz. 0.1 to 20. Default = 4.
FMOD_DSP_TREMOLO_DEPTH
Tremolo depth. 0 to 1. Default = 0.
FMOD_DSP_TREMOLO_SHAPE
LFO shape morph between triangle and sine. 0 to 1. Default =
0.
FMOD_DSP_TREMOLO_SKEW
Time-skewing of LFO cycle. -1 to 1. Default = 0.
FMOD_DSP_TREMOLO_DUTY
LFO on-time. 0 to 1. Default = 0.5.
FMOD_DSP_TREMOLO_SQUARE
Flatness of the LFO shape. 0 to 1. Default = 0.
FMOD_DSP_TREMOLO_PHASE
Instantaneous LFO phase. 0 to 1. Default = 0.
FMOD_DSP_TREMOLO_SPREAD
Rotation / auto-pan effect. -1 to 1. Default = 0.
```

Use the following code as a basis for your Windows start up sequence:

```
FMOD::System    *system;
FMOD_RESULT      result;
unsigned int     version;
int              numdrivers;
FMOD_SPEAKERMODE speakermode;
FMOD_CAPS        caps;
char             name[256];
 /*     Create a System object and initialize. */
```

```
result = FMOD::System_Create(&system);
ERRCHECK(result);
result = system->getVersion(&version);
ERRCHECK(result);
if (version < FMOD_VERSION)
{
    printf("Error!  You are using an old version of FMOD %08x.
This program requires %08x\n",
version, FMOD_VERSION);
    return 0;
}
 result = system->getNumDrivers(&numdrivers);
ERRCHECK(result);
 if (numdrivers == 0)
{
    result = system->setOutput(FMOD_OUTPUTTYPE_NOSOUND);
    ERRCHECK(result);
}
else
{  result = system->getDriverCaps(0, &caps, 0, 0,
&speakermode);
    ERRCHECK(result);
     /*
        Set the user selected speaker mode.
    */
    result = system->setSpeakerMode(speakermode);
    ERRCHECK(result);
     if (caps & FMOD_CAPS_HARDWARE_EMULATED)
    {
        /*  The user has the 'Acceleration' slider set to off!
This is really bad
            for latency! You might want to warn the user about
this.
        */
        result = system->setDSPBufferSize(1024, 10);
        ERRCHECK(result);
    }
     result = system->getDriverInfo(0, name, 256, 0);
    ERRCHECK(result);
     if (strstr(name, "SigmaTel"))
    {
```

```
        /*  Sigmatel sound devices crackle for some reason if
the format is PCM 16bit.
            PCM floating point output seems to solve it.
        */
        result = system->setSoftwareFormat(48000,
FMOD_SOUND_FORMAT_PCMFLOAT, 0,0, FMOD_DSP_RESAMPLER_LINEAR);
        ERRCHECK(result);
    }
}
 result = system->init(100, FMOD_INIT_NORMAL, 0);
if (result == FMOD_ERR_OUTPUT_CREATEBUFFER)
{   /*        Ok, the speaker mode selected isn't supported by
this soundcard.  Switch it
        back to stereo...
    */
    result = system->setSpeakerMode(FMOD_SPEAKERMODE_STEREO);
    ERRCHECK(result);
     /*         ... and re-init.
    */
    result = system->init(100, FMOD_INIT_NORMAL, 0);
}
ERRCHECK(result);
```

Лабораторная работа №5"
Потоковое видео. Библиотека OpenCV"

OpenCV (Open Source Computer Vision Library) — библиотека алгоритмов компьютерного зрения, обработки изображений и численных алгоритмов общего назначения с открытым кодом. Она содержит более 500 функций для обработки, реконструкции и очистки изображений, распознания образов, захвата видео, слежения за объектами, калибровки камер и др. Библиотека была разработана Intel, а сейчас поддерживается Willow Garage Inc. и Itseez Ltd. OpenCV распространяется под лицензией BSD, а значит, бесплатна, как для учебных, так и для коммерческих целей. Реализована на C++, C, Python и Java, поддерживает Windows, Linux, Mac OS, iOS и Android.

Задание на выполнение работы

ЧТО НУЖНО ИЗЧИТЬ

1. Программирование с использованием библиотеки OpenCV.

ЧТО НУЖНО СДЕЛАТЬ

1. Напишите программу, которая захватывает видео с камеры и записывает на диск, используя 6 Захват видео с камеры и 7 Запись в AVI файл.
2. Объедините программу из 6 Захват видео с камеры с функцией обработки (фильтрация, наложение статической картинки, определение движения…). Параметры функции должны меняться ползунком. У каждого студента своя функция.
3. По желанию. Напишите программу поиска в потоке заданного изображения. Предложите свой вариант алгоритмов отображения и обработки.

Познакомьтесь с информацией на сайте `http://robocraft.ru/page/opencv/` с которого взяты описания лабораторных работ.

1. Первая программа - Вывод картинки

OpenCV предоставляет средства для чтения огромного количества типов изображений, в том числе и из видео файлов и камер. Эти утилиты часть модуля HighGUI включённого в OpenCV. Мы можем написать простую программу для вывода изображения из файла на экран используя эти средства:

```
#include <highgui.h>
int main()
{
IplImage* img = cvLoadImage("D:\\foto.jpg"); // Загружаем
изображение
cvNamedWindow("Example1", CV_WINDOW_AUTOSIZE); // Создаём
окно
cvShowImage("Example1", img); // Выводим картинку в окно
cvWaitKey(0); // Ждём
cvReleaseImage(&img); // Освобождаем память из под картинки
cvDestroyWindow("Example1"); // Удаляем окно
return 0; }
```

После компиляции и запуска это программы она выведет на экран изображение.

Давайте теперь разберём каждую строчку программы:

#include <highgui.h>

Подключение модуля HighGUI

IplImage *img = cvLoadImage("D:\\foto.jpg");

cvLoadImage загружает картинку из файла (BMP, DIB, JPEG, JPE, PNG, PBM, PGM, PPM, SR, RAS, TIFF) и возвращает указатель на неё. IplImage это и есть картинка, в OpenCV она используется для всех видов картинок: одноканальные, многоканальные, 8, 16, 32 битные и т.д.

cvNamedWindow ("Example 1", CV_WINDOW_AUTOSIZE);

Создаёт окно, первый параметр – имя окна, второй - параметры окна. Второй параметр обычно равен 0 или CV_WINDOW_AUTOSIZE что одно и тоже, если второй параметр равен 0 то окно будет таких же размеров что и изображение которое в него выводится.

cvShowImage(“Example1”, img);

Выводит картинку в окно, первый параметр имя окна в которое выводится изображение, второй – указатель на структуру IplImage.

cvWaitKey(0);

Ожидает нажатии клавиши заданное кол-во миллисекунд, если кол-во мс равно 0 то бесконечно ждёт нажатие любой клавиши. Также возвращает код нажатой клавиши.

cvReleaseImage(&img);

Освобождает память, выделенную под картинку, и устанавливает указатель в NULL.

cvDestroyWindow(“Example1”);

Закрывает созданное ранее окно.

2. Вторая программа - AVI Видео

Вывод видео с помощью OpenCV. Проблема: цикл для чтения каждого кадра; нам также нужно условие выхода из цикла, если фильм окажется слишком скучным.

```
#include “highgui.h”
int main() {
    cvNamedWindow( “Example2”, CV_WINDOW_AUTOSIZE ); //
Создаем окошко
    CvCapture* capture = cvCreateFileCapture( “D:\\film.avi”
); // Открываем файл
    IplImage* frame; // Здесь будет кадр
    while(1) {
        frame = cvQueryFrame( capture ); // Читаем кадр из
файла
        if( !frame ) break; // Если кадров больше нет -
выходим
        cvShowImage( “Example2”, frame ); // Выводим кадр
        char c = cvWaitKey(33); // Ждем 33мс
   if( c == 27 ) break; // Если нажали Esc - выходим
    }
    cvReleaseCapture( &capture ); // Закрываем файл
    cvDestroyWindow( “Example2” ); // И окно
}
```

3. Создание ползунка

Наша следующая задача – создать ползунок, позволяющий перемотать видео.

HighGUI позволяет создавать множество простых инструментов для работы с изображениями и видео. Один из особенно полезных инструментов – это ползунок, позволяющий легко решить данную задачу. Для его создания используется функция cvCreateTrackbar.

```
#include “cv.h”
#include “highgui.h”
int         g_slider_position = 0; // Позиция ползунка
CvCapture* g_capture = NULL; // Для захвата видео файла
void onTrackbarSlide(int pos) { // Эта функция будет
вызыватся каждый раз при изменении положения ползунка
    cvSetCaptureProperty( // Установка свойств видеозахвата
        g_capture,
        CV_CAP_PROP_POS_FRAMES, // Номер кадра для захвата
        pos );
}
int main() {
    cvNamedWindow( “Example3”, CV_WINDOW_AUTOSIZE ); //
Создаём окошко
    g_capture = cvCreateFileCapture("D:\\film.avi");
    int frames = (int) cvGetCaptureProperty( // Получаем
количетсво кадров
        g_capture, CV_CAP_PROP_FRAME_COUNT );
    if( frames!= 0 ) {
      cvCreateTrackbar( // Создаём ползунок
          “MyTrack”, // Имя ползунка
          “Example3”, // Окно в которое его выводить
          &g_slider_position, // Начальная позиция
          frames, // Максимальная позиция
          onTrackbarSlide // Функция обработчик
      );
    }
    IplImage* frame; // Кадр
while(1) {
        frame = cvQueryFrame( g_capture ); // Получаем кадр
        if( !frame ) break; // Если кадры закончились -
выходим
```

```
            cvShowImage( “Example3”, frame ); // Выводим кадр
            char c = cvWaitKey(33); // Ждём 33мс
            if( c == 27 ) break;  // Если нажали Esc - выходим
        }
        cvReleaseCapture(&g_capture);
        return(0);
    }
```

4. Простое преобразование

Теперь с помощью OpenCV Вы можете создать собственный видеоплеер, который не будет сильно отличатся от множества уже существующих плееров. Но мы ведь интересуемся компьютерным зрением, поэтому хотим чего-то большего, чем видеоплеер. Большинство задач компьютерного зрения требуют применения к видео потоку различных фильтров и преобразований. Мы немного изменим нашу программу, вместо видео используем простое изображение и размоем его.

Одна из самых простых операций это – сглаживание изображения, например по Гауссу. С помощью OpenCV это очень просто сделать. В следующем примере мы начинаем с создания окна куда мы можем вывести результат операции. Затем мы выводим в первое окно исходное изображение, размываем второе и выводим его во второе окно.

```
#include “cv.h”
#include “highgui.h”

int main()
{
IplImage *original =
cvLoadImage("D:\\Programming\\OpenCV\\img.jpg"); // Создаём
и загружаем изображение
IplImage *result = cvCreateImage(cvGetSize(original),
IPL_DEPTH_8U, 3); // Создаём изображение
cvNamedWindow("original", CV_WINDOW_AUTOSIZE); // Создаём
окошко для оригинала
cvNamedWindow("result", CV_WINDOW_AUTOSIZE); // Создаём
окошко для результата
cvShowImage("original", original); // Выводим оригинал
```

```
cvSmooth(original, result, CV_GAUSSIAN, 3, 3);//Размываем по
Гауссу
cvShowImage("result", result); // Выводим результат
cvWaitKey(); // Ждём нажатия клавиши
cvDestroyAllWindows(); // Убиваем все окна
cvReleaseImage(&original); // Освобождаем память
cvReleaseImage(&result);
}
```

5. Более сложные преобразования

В прошлом примере мы создавали новую структуру и затем записывали в неё результат преобразования. Как уже упоминалось, мы можем применить преобразование так, что результат “затрёт” оригинал, но это не очень хорошая идея. В частности некоторые операторы создают изображения, отличающиеся от исходных размером, глубиной и числом каналов. Часто нам будет необходимо производить последовательность преобразований над изображением. В таких случаях удобнее выносить все эти действия в отдельную функцию, и работать с изображением уже внутри неё.

Для примера давайте напишем функцию для уменьшения размера изображения в два раза. Это можно сделать с помощью функции cvPyrDown() которая сначала размывает изображение по Гауссу а затем удаляет лишние части изображения. Это функция будет нам очень полезная в будущем.

```
IplImage* doPyrDown(
  IplImage* in,
  int       filter = IPL_GAUSSIAN_5x5
) {
    assert( in->width%2 == 0 && in->height%2 == 0 ); //
Убеждаемся в том что входное
изображение можно ужать в 2 раза
    IplImage* out = cvCreateImage( // Создаём изображение с
размером в 2 раза меньшим исходного
        cvSize( in->width/2, in->height/2 ),
        in->depth,
        in->nChannels
    );
    cvPyrDown( in, out ); // Выполняем преобразование
```

```
    return( out );
};
```

В этой функции мы убеждаемся в том, что изображение можно уменьшить в 2 раза, создаём новое изображения, используя параметры входного. А затем с помощью функции cvPyrDown выполняем преобразование. Также тут стоит отметить, что в OpenCV все типы данных созданы в виде структур, в которых не существует private данных! Теперь давайте рассмотрим более сложный пример с функцией cvCanny() позволяющей выделить края.

```
IplImage* doCanny(
    IplImage* in,
    double    lowThresh,
    double    highThresh,
    double    aperture
) {
    If(in->nChannels != 1)
return(0);//cvCanny() работает только с одноканальными
изображениями
    IplImage* out = cvCreateImage(
        cvSize( cvGetSize( in ),
        IPL_DEPTH_8U,
        1
    );
    cvCanny( in, out, lowThresh, highThresh, aperture );
    return( out );};
```

6. Захват видео с камеры

OpenCV, а точнее модуль HighGUI, предоставляет нам инструменты для анализировать видео в реальном времени, например с видеокамеры. Способ аналогичен захвату видео, только вместо cvCreateFileCapture нужно вызвать cvCreateCameraCapture. Эта функция вместо имени файла принимает ID камеры, но это имеет смысл только при наличии нескольких камер. Значение по умолчанию равно -1. cvCreateCameraCapture возвращает указатель на структуру CvCapture, с которой мы уже знакомы. Конечно много работы происходит “за кулисами” для того чтобы изображение с камеры

просматривать как видео, но все эти проблемы скрыты от пользователя. Мы можем просто захватить изображение с камеры, когда нам это нужно. Смотрим на пример:

```
#include “highgui.h”
#include “cv.h”
int main()
{
CvCapture *capture = cvCreateCameraCapture(0); // Думаю тут
всё понятно
if(capture == NULL) // Если камер не обнаружено - выходим
        return 0;
IplImage *frame = NULL; // Кадр
cvNamedWindow("camera", CV_WINDOW_AUTOSIZE); // Окошко
while(1)
{
        frame = cvQueryFrame(capture); // Получаем кадр, так
же как и из видео файла
        cvShowImage("camera", frame); // Выводим
        char c = cvWaitKey(33); // Ждём
        if(c == 27)break; // Если Esc - выходим
}
cvReleaseCapture(&capture);
return 0;
}
```

7. Запись в AVI файл

Во многих приложениях нам потребуется запись потокового видео в файл, и OpenCV предоставляет простые средства для этого. Точно также, как мы создавали устройство захвата, мы можем создать и устройство записи, и затем после получения кадра записать его в видео файл. Функция позволяющая сделать это называется cvCreateVideoWriter. После вызова этой функции мы можем воспользоваться cvWriteFrame() для записи кадра в файл, а затем cvReleaseFileCapture когда запись завершена. В следующем примере приведён код программы, которая открывает файл, получает кадр, конвертирует его в полярный вид(то-что видит глаз на самом деле описано в главе 6), и записывает в другой файл.

```
// argv[1]: Имя входного файла
```

```
 // argv[2]: Имя выходного файла, будет создан
#include “cv.h”
#include “highgui.h”
main( int argc, char* argv[] ) {
CvCapture* capture = 0; // Для захвата видео файла
capture = cvCreateFileCapture( argv[1] ); // Открываем файл
    if(!capture){  // Если не получилось - выходим
        return -1; }
    IplImage *bgr_frame=cvQueryFrame(capture); //
Инициализируем чтение видео файла
    double fps = cvGetCaptureProperty ( // Получаем частоту
кадров
        capture,
        CV_CAP_PROP_FPS
    );
    CvSize size = cvSize( // Получаем размер
       (int)cvGetCaptureProperty( capture,
CV_CAP_PROP_FRAME_WIDTH),
       (int)cvGetCaptureProperty( capture,
CV_CAP_PROP_FRAME_HEIGHT)
    );
    CvVideoWriter *writer = cvCreateVideoWriter( // Создаём
файл для записи
        argv[2],
        CV_FOURCC(‘M’,‘J’,‘P’,‘G’),
        fps,
        size
    );
    IplImage* logpolar_frame = cvCreateImage( // Кадр в
полярных координатах
        size,
        IPL_DEPTH_8U,
        3
    );
    while( (bgr_frame=cvQueryFrame(capture)) != NULL ) { //
Читаем кадры, пока они не закончатся
        cvLogPolar( bgr_frame, logpolar_frame, //
Преобразовываем
                    cvPoint2D32f(bgr_frame->width/2,
                    bgr_frame->height/2),
                    40,
```

```
                    CV_INTER_LINEAR+CV_WARP_FILL_OUTLIERS );
        cvWriteFrame( writer, logpolar_frame ); //
Записываем
    }
    cvReleaseVideoWriter( &writer ); // Освобождаем ресурсы
    cvReleaseImage( &logpolar_frame );
    cvReleaseCapture( &capture );
    return(0);
}
```

В этой программе есть ещё незнакомые Вам элементы. Мы открываем видео файл, начинаем читать его с помощью cvQueryFrame(), считываем свойства видео файла, затем с помощью cvGetCaptureProperty() узнаём размер видео потока. Затем открываем файл на запись, конвертируем кадр в полярный вид, и записываем его в другой файл. Вызов CvCreateVideoWriter() содержит несколько параметров которые мы должны понимать. Первый это имя файла. Второй это видео кодек с помощью которого будет сжиматься видео поток. В данном случае мы выбрали популярный кодек MJPG, мы сообщаем об этом OpenCV с помощью макроса CV_FOURCC(), который принимает четыре символа в качестве аргументов. Список кодеков включённых в OpenCV:

'X','V','I','D' - кодек XviD
'P','I','M','1' - MPEG-1
'M','J','P','G' - motion-jpeg
'M', 'P', '4', '2' - MPEG-4.2
'D', 'I', 'V', '3' - MPEG-4.3
'D', 'I', 'V', 'X' - MPEG-4
'U', '2', '6', '3' - H263
'I', '2', '6', '3' - H263I
'F', 'L', 'V', '1' - FLV1

Следующие два параметра – это частота кадров, и размер видео, в данном случае мы получаем эти параметры из исходного файла.

8. Загрузка и установка

Последнюю версию OpenCV можно загрузить с Sourceforge.net. После того как вы скачаете библиотеку вы должны её установить.

Windows

Скачайте установочный файл и запустите его. Программа установки установит OpenCV, зарегистрирует фильтр DirectShow, и выполнит все необходимые процедуры. Теперь Вы готовы к началу использования OpenCV. Открывайте VC++, создайте проект Console Application Win32, зайдите в Проект->Свойства:

Допишите в “Каталоги включения”:

OpenCV_Folder\include\opencv

“Каталоги библиотек”:

OpenCV_Folder\lib

“Каталоги исходного кода”:

OpenCV_Folder\src\ml

OpenCV_Folder\src\highgui

OpenCV_Folder\src\cxcore

OpenCV_Folder\src\cvaux

OpenCV_Folder\src\cv

Затем зайдите в Компоновщик->Ввод->Дополнительные зависимости и допишите туда:

cxcore210.lib

cv210.lib

highgui210.lib

cvaux210.lib

Затем желательно зайти в Диспетчер Конфигураций и поставить конфигурацию Release по умолчанию, чтобы Ваша программа могла запускаться на других компьютерах вместе с ней нужно будет скопировать следующие DLL:

cxcore210.dll

cv210.dll

highgui210.dll

cvaux210.dll
из папки OpenCV_Folder\bin

Пример

```
#include "opencv\cv.h"
#include "opencv\highgui.h"
#pragma comment(lib, "opencv_highgui231.lib")
#pragma comment(lib, "opencv_core231.lib")
#pragma comment(lib, "opencv_objdetect231.lib")
#include <iostream>
using namespace std;
char* gsp(IplImage *image, int x, int y)
{return image->imageData + y*image->width*3 + x*3;
}
void Frame(IplImage *image, CvRect *r)
{    CvPoint pt1 = { r->x, r->y };
     CvPoint pt2 = { r->x + r->width, r->y + r->height };
     cvRectangle(image, pt1, pt2, CV_RGB(0,255,0), 3, 4, 0);
}
void Invert(IplImage *image, CvRect *r)
{    int w = r->x+r->width, h = r->y+r->height, i, j;
     for (j=r->y; j<=h-1; j++)
     {
          for (i=r->x; i<=w-1; i++)
          {
               char* p = gsp(image, i, j);
               p[0] = 0xFF - p[0];
               p[1] = 0xFF - p[1];
               p[2] = 0xFF - p[2];
          } } }
void GrayScale(IplImage *image, CvRect *rn)
{    int w = rn->x+rn->width, h = rn->y+rn->height, i, j, r, g,
b;
     for (j=rn->y; j<=h-1; j++)
     {
          for (i=rn->x; i<=w-1; i++)
          {
               char *temp = gsp(image, i, j);
               r = temp[0]; g = temp[1]; b = temp[2];
```

```
                r = g = b = (unsigned
char)(0.299*(double)r+0.587*(double)g+0.114*(double)b);
                temp[0] = r; temp[1] = g; temp[2] = b;
          } } }
void main()
{ // Вывод видео http://locv.ru/wiki/
     cvNamedWindow( "Example2", CV_WINDOW_AUTOSIZE ); //
Создаем окошко
    CvCapture* capture = cvCreateFileCapture( "D:\\77.avi" );
// Открываем файл
    IplImage* frame; // Здесь будет кадр
    while(1) {
frame = cvQueryFrame( capture ); // Читаем кадр из файла
if( !frame ) break; // Если кадров больше нет - выходим
        cvShowImage( "Example2", frame ); // Выводим кадр
        char c = cvWaitKey(33); // Ждем 33мс
        if( c == 27 ) break; // Если нажали Esc - выходим
    }
    cvReleaseCapture( &capture ); // Закрываем файл
    cvDestroyWindow( "Example2" ); // И окно
```

Лабораторная работа №6 "Технология CUDA"

(Compute Unified Device Architecture)

(4 часа в классе и самостоятельная работа)

Технология CUDA появилась в 2006 году и представляет из себя программно-аппаратный комплекс производства компании Nvidia, позволяющий эффективно писать программы под графические адаптеры. Компания Nvidia обещает, что все графические адаптеры их производства независимо от серии будут иметь сходную архитектуру, которая полностью поддерживает программную часть технологии CUDA. Программная часть, в свою очередь, содержит в себе всё необходимое для разработки программы: расширения языка C, компилятор, API для работы с графическими адаптерами и набор библиотек [5].

CUDA SDK позволяет программистам реализовывать на специальном упрощённом диалекте языка программирования Си алгоритмы, выполнимые на графических процессорах NVIDIA. Графический процессор организует аппаратную многопоточность, что позволяет задействовать все ресурсы графического процессора.

CUDA и язык C. Сама технология CUDA (компилятор nvcc.exe) вводит ряд дополнительных расширений для языка C, которые необходимы для написания кода для GPU:

1.Спецификаторы функций, которые показывают, как и откуда буду выполняться функции.

2. Спецификаторы переменных, которые служат для указания типа используемой памяти GPU.

3. Спецификаторы запуска ядра GPU.

4. Встроенные переменные для идентификации нитей, блоков и др. параметров при исполнении кода в ядре GPU .

5. Дополнительные типы переменных.

Задание на выполнение работы

ЧТО НУЖНО ИЗЧИТЬ

1. Структуры процессоров, использующих технологию CUDA
2. Программирование с использованием технологии CUDA.

ЧТО НУЖНО СДЕЛАТЬ

1. Загрузите и запустите программу, которая тестирует адаптер монитора.
2. Познакомьтесь с вычислительной моделью CUDA (см. рисунки). Напишите программу сложения двух массивов в GPU.
3. Включите в вашу программу функции синхронизации.
4. Для шустрых. Напишите программу улучшения (фильтрации) видеоизображения с применением функций OpenCV и CUDA.
5. Для шустрых. Напишите программу работы с видео, в которой используются функции библиотек Vulkan или Direct3D.

Список электронных ресурсов, которые желательно посмотреть:

1. Начало знакомства http://ru.wikipedia.org/wiki/CUDA
2. Первая программа http://habrahabr.ru/post/54330/
3. Короткое описание http://habrahabr.ru/post/54707/
4. Документация http://docs.nvidia.com/cuda/index.html

1. Первая программа тестирования адаптера

Что потребуется для работы:

1. Видеокарта из серии nVidia GeForce 8xxx/9xxx или более современная
2. CUDA Toolkit v.2.1 (скачать можно здесь: www.nvidia.ru/object/cuda_get_ru.html)
3. CUDA SDK v.2.1 (скачать можно там же где Toolkit)
4. Visual Studio 2008
5. CUDA Visual Studio Wizard (скачать можно здесь: sourceforge.net/projects/cudavswizard/)и др.

Создание CUDA проекта:

После установки всего необходимого в VS появиться новый вид проекта для C++ с названием CUDA WinApp. В данном типе проекта

доступны дополнительные настройки для CUDA, позволяющие настроить параметры компиляции под GPU, в зависимости от типа GPU и т.д. Обычно создается чистый проект (Empty Project), так как Precompiled Headers навряд ли пригодиться для CUDA.

Важно отметить, как собирается CUDA приложение. Файлы с расширением *.cpp обрабатываются компилятором MS C++ (cl.exe), а файлы с расширением *.cu компилятором CUDA (nvcc.exe), который в свою очередь определяет, какой код будет работать на GPU, а какой на CPU. Код из *.cu, работающий на CPU, передается на компиляцию MS C++, эту особенность удобно использовать для написания динамических библиотек, которые будут экспортировать функции, использующие для расчетов GPU.

Текст программы, которая выводит на экран информацию об аппаратных возможностях GPU.:

```
    #include <stdio.h>
#include <cuda_runtime_api.h>

int main()
{  int deviceCount;
  cudaDeviceProp deviceProp;
  //Сколько устройств CUDA установлено на PC.
  cudaGetDeviceCount(&deviceCount);
  printf("Device count: %d\n\n", deviceCount);
  for (int i = 0; i < deviceCount; i++)
  {
    //Получаем информацию об устройстве
    cudaGetDeviceProperties(&deviceProp, i);
    //Выводим иформацию об устройстве
    printf("Device name: %s\n", deviceProp.name);
    printf("Total global memory: %d\n",
deviceProp.totalGlobalMem);
    printf("Shared memory per block: %d\n",
deviceProp.sharedMemPerBlock);
    printf("Registers per block: %d\n",
deviceProp.regsPerBlock);
    printf("Warp size: %d\n", deviceProp.warpSize);
```

```
    printf("Memory pitch: %d\n", deviceProp.memPitch);
    printf("Max threads per block: %d\n",
deviceProp.maxThreadsPerBlock);
    printf("Max threads dimensions: x = %d, y = %d, z = %d\n",
      deviceProp.maxThreadsDim[0],
      deviceProp.maxThreadsDim[1],
      deviceProp.maxThreadsDim[2]);
      printf("Max grid size: x = %d, y = %d, z = %d\n",
      deviceProp.maxGridSize[0],
      deviceProp.maxGridSize[1],
      deviceProp.maxGridSize[2]);
    printf("Clock rate: %d\n", deviceProp.clockRate);
    printf("Total constant memory: %d\n",
deviceProp.totalConstMem);
    printf("Compute capability: %d.%d\n", deviceProp.major,
deviceProp.minor);
    printf("Texture alignment: %d\n",
deviceProp.textureAlignment);
    printf("Device overlap: %d\n", deviceProp.deviceOverlap);
    printf("Multiprocessor count: %d\n",
deviceProp.multiProcessorCount);
    printf("Kernel execution timeout enabled: %s\n",
      deviceProp.kernelExecTimeoutEnabled ? "true" : "false");
  }
  return 0;}
```

2. Вычислительная модель

Основные термины.

Хост(Host) - центральный процессор, управляющий выполнением программы.

Устройство(Device)—видеоадаптер, выступающий в роли сопроцессора центрального процессора.

Грид(Grid)—объединение блоков, которые выполняются наодном устройстве.

Блок(Block)—объединение тредов, которое выполняется целиком на одномSM. Имеет свой уникальный идентификатор внутри грида.

Тред(Thread,поток)—единица выполнения программы. Имеетсвой уникальный идентификатор внутри блока.

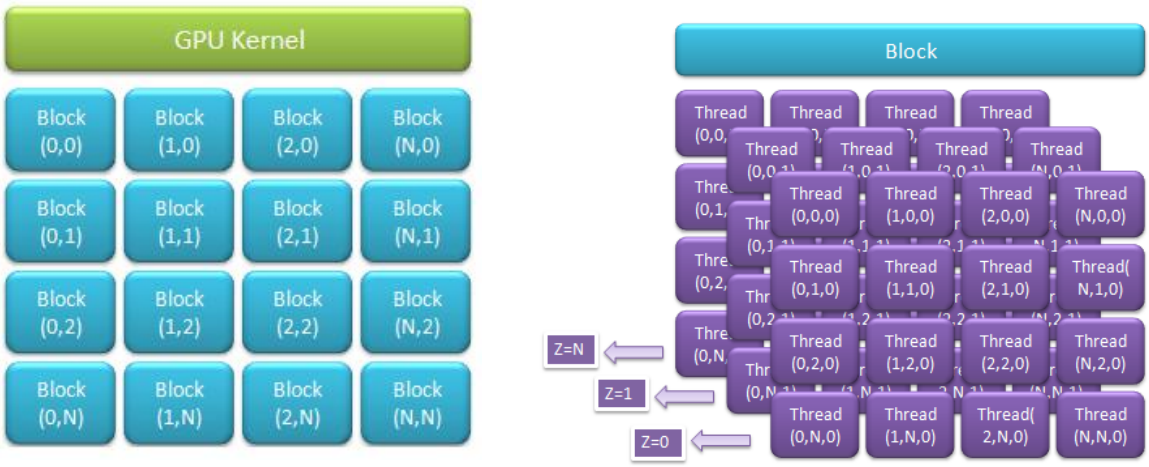
Варп(Warp)—32 последовательно идущих треда, выполняется физически одновременно.

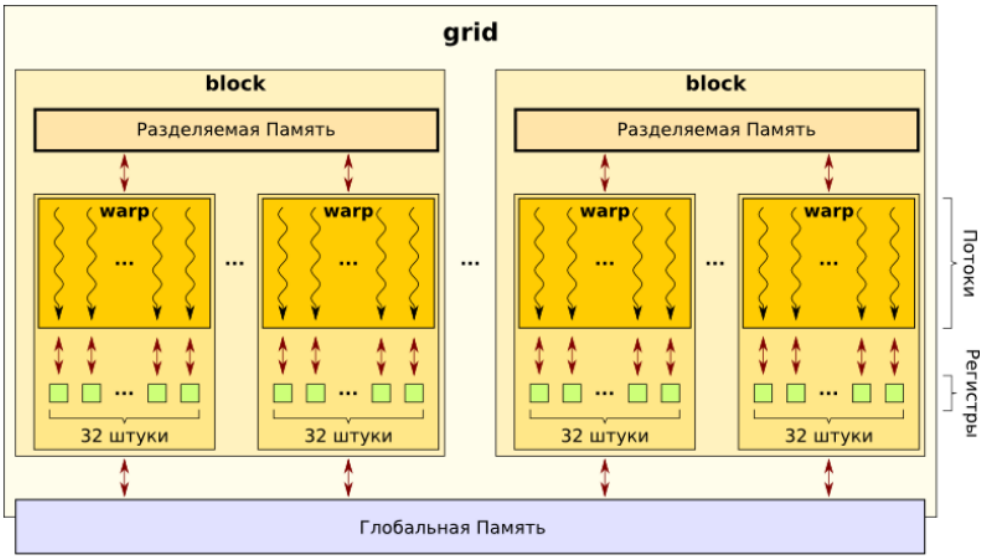
Ядро(Kernel)—параллельная часть алгоритма, выполняется на гриде.

Вычислительная модель GPU

Верхний уровень ядра GPU состоит из блоков, которые группируются в сетку или грид (grid) размерностью N1 * N2 * N3. Размерность сетки блоков можно узнать с помощь функции cudaGetDeviceProperties, в полученной структуре за это отвечает поле maxGridSize.

Любой блок в свою очередь состоит из нитей (threads), которые являются непосредственными исполнителями вычислений. Нити в блоке сформированы в виде трехмерного массива (рис. 2), размерность которого так же можно узнать с помощью функции cudaGetDeviceProperties, за это отвечает поле maxThreadsDim.





3. Основы программирования

CUDA API (application programming interface)

В CUDA есть два уровня API:низкоуровневый драйвер-API и высокоуровневый runtime-API. Runtime-API реализован через драйвер-API.

Runtime-API обладает меньшей гибкостью, но более удобен для написания программ. Оба API не требуют явной инициализации, и для использования дополнительных типов и других расширений языка C не требуется подключать дополнительные заголовочные файлы.

Все функции драйвер-API начинаются с приставки cu, все функции runtime-API начинаются с приставки cuda. Практически все функциии обоих API возвращают значение типа t_cudaError, которое принимает значение cudaSuccess в случае успеха. CUDA host API является связующим звеном между CPU и GPU.

В CUDA runtime API входят следующие группы функций:

Device Management – включает функции для общего управления GPU (получение информации о возможностях GPU, переключение между GPU при работе SLI-режиме и т.д.).

Thread Management – управление нитями.

Stream Management – управление потоками.

Event Management – функция создания и управления event’ами.

Execution Control – функции запуска и исполнения ядра CUDA.

Memory Management – функции управлению памятью GPU.

Texture Reference Manager – работа с объектами текстур через CUDA.

OpenGL Interoperability – функции по взаимодействию с OpenGL API.

Direct3D 9 Interoperability – функции по взаимодействию с Direct3D 9 API.

Direct3D 10 Interoperability – функции по взаимодействию с Direct3D 10 API.

Error Handling – функции обработки ошибок.

Спецификаторы функций определяют, как и откуда буду вызываться функции

__host__ — выполнятся на CPU, вызывается с CPU (в принципе его можно и не указывать).

__global__ — выполняется на GPU, вызывается с CPU.

__device__ — выполняется на GPU, вызывается с GPU.

Спецификаторы запуска ядра служат для описания количества блоков, нитей и памяти, которые вы хотите выделить при расчете на GPU.

myKernelFunc<<<gridSize, blockSize, sharedMemSize, cudaStream>>>(float* param1,float* param2), где

gridSize – размерность сетки блоков (dim3), выделенную для расчетов,

blockSize – размер блока (dim3), выделенного для расчетов,

sharedMemSize – размер дополнительной памяти, выделяемой при запуске ядра,

cudaStream – переменная cudaStream_t, задающая поток, в котором будет произведен вызов.

Встроенные переменные:

gridDim – размерность грида, имеет тип dim3. Позволяет узнать размер грида, выделенного при текущем вызове ядра.

blockDim – размерность блока, так же имеет тип dim3. Позволяет узнать размер блока, выделенного при текущем вызове ядра.

blockIdx – индекс текущего блока в вычислении на GPU, имеет тип uint3.
threadIdx – индекс текущей нити в вычислении на GPU, имеет тип uint3.
warpSize – размер warp'a, имеет тип int.

4. Пример разработки программы и некоторые функции для работы с CUDA

Задача. Требуется вычислить сумму двух векторов размерностью N элементов. Нам известна максимальные размеры нашего блока: 512*512*64 нитей. Так как вектор у нас одномерный, то пока ограничимся использованием x-измерения нашего блока, то есть задействуем только одну полосу нитей из блока. Заметим, что x-размерность блока 512, то есть, мы можем сложить за один раз векторы, длина которых N <= 512 элементов.

В самой программе необходимо выполнить следующие этапы:

1. Получить данные для расчетов.
2. Скопировать эти данные в GPU память.
3. Произвести вычисление в GPU через функцию ядра.
4. Скопировать вычисленные данные из GPU памяти в ОЗУ.
5. Посмотреть результаты.
6. Высвободить используемые ресурсы.

Для выделения памяти на видеокарте используется функция cudaMalloc, которая имеет следующий прототип:

cudaError_t cudaMalloc(void** devPtr, size_t count), где
devPtr – указатель, в который записывается адрес выделенной памяти,
count – размер выделяемой памяти в байтах.
Возвращает:
cudaSuccess – при удачном выделении памяти
cudaErrorMemoryAllocation – при ошибке выделения памяти

Для копирования данных в память видеокарты используется cudaMemcpy, которая имеет следующий прототип:

cudaError_t cudaMemcpy(void* dst, const void* src ,size_t count, enum cudaMemcpyKind kind), где

dst – указатель, содержащий адрес места-назначения копирования,

src – указатель, содержащий адрес источника копирования,

count – размер копируемого ресурса в байтах,

cudaMemcpyKind – перечисление, указывающее направление копирования (может быть cudaMemcpyHostToDevice, cudaMemcpyDeviceToHost, cudaMemcpyHostToHost, cudaMemcpyDeviceToDevice).

Возвращает:cudaSuccess – при удачном копировании

cudaErrorInvalidValue – неверные параметры аргумента (например, размер копирования отрицателен)

cudaErrorInvalidDevicePointer – неверный указатель памяти в видеокарте

cudaErrorInvalidMemcpyDirection – неверное направление (например, перепутан источник и место-назначение копирования)

Программа

```
    // Функция сложения двух векторов
__global__ void addVector(float* left, float* right, float*
result)
{  //Получаем id текущей нити.
  int idx = threadIdx.x;
  //Расчитываем результат.
  result[idx] = left[idx] + right[idx];
}
    #define SIZE 512
__host__ int main()
{  //Выделяем память под вектора
  float* vec1 = new float[SIZE];
  float* vec2 = new float[SIZE];
  float* vec3 = new float[SIZE];
  //Инициализируем значения векторов
  for (int i = 0; i < SIZE; i++)
  {   vec1[i] = i;
    vec2[i] = i;
  }  //Указатели на память видеокарте
  float* devVec1;
```

```
  float* devVec2;
  float* devVec3;
    //Выделяем память для векторов на видеокарте
  cudaMalloc((void**)&devVec1, sizeof(float) * SIZE);
  cudaMalloc((void**)&devVec2, sizeof(float) * SIZE);
  cudaMalloc((void**)&devVec3, sizeof(float) * SIZE);
      //Копируем данные в память видеокарты
  cudaMemcpy(devVec1, vec1, sizeof(float) * SIZE,
cudaMemcpyHostToDevice);
  cudaMemcpy(devVec2, vec2, sizeof(float) * SIZE,
cudaMemcpyHostToDevice);
    //Непосредственный вызов ядра для вычисления на GPU.
dim3 gridSize = dim3(1, 1, 1);    //Размер используемого грида
dim3 blockSize = dim3(SIZE, 1, 1); //Размер используемого
блока
//Выполняем вызов функции ядра
addVector<<<gridSize, blockSize>>>(devVec1, devVec2, devVec3);
```

Определять размер грида и блока необязательно, так как используем всего один блок и одно измерение в блоке, поэтому код выше можно записать:

```
addVector<<<1, SIZE>>>(devVec1, devVec2, devVec3); …
```

Остается скопировать результат расчета из видеопамяти в память хоста. Но у функций ядра при этом есть особенность – асинхронное исполнение, то есть, если после вызова ядра начал работать следующий участок кода, то это ещё не значит, что GPU выполнил расчеты. Для завершения работы заданной функции ядра необходимо использовать средства синхронизации, например event’ы. Поэтому, перед копированием результатов на хост выполняем синхронизацию нитей GPU через event.

Код после вызова ядра:

```
  //Выполняем вызов функции ядра
  addVector<<<blocks, threads>>>(devVec1, devVec2, devVec3);
  //Хендл event'a
  cudaEvent_t syncEvent;
  cudaEventCreate(&syncEvent);    //Создаем event
  cudaEventRecord(syncEvent, 0);  //Записываем event
  cudaEventSynchronize(syncEvent);  //Синхронизируем event
  //Только теперь получаем результат расчета
```

```
  cudaMemcpy(vec3, devVec3, sizeof(float) * SIZE,
cudaMemcpyDeviceToHost);
//Выводим результат на экран и чистим выделенные ресурсы.
  for (int i = 0; i < SIZE; i++)
  {printf("Element #%i: %.1f\n", i , vec3[i]);
  }
  // Высвобождаем ресурсы
    cudaEventDestroy(syncEvent);
  cudaFree(devVec1);
  cudaFree(devVec2);
  cudaFree(devVec3);
  delete[] vec1; vec1 = 0;
  delete[] vec2; vec2 = 0;
  delete[] vec3; vec3 = 0;
```

Пример использования функции удаления фона Mog2, встроенной в OpenCV, с применением технологии CUDA

```
#include < time.h>
#include < opencv2\opencv.hpp>
#include < opencv2\gpu\gpu.hpp>
#include < string>
#include < stdio.h>

#define RWIDTH 800
#define RHEIGHT 600

using namespace std;
using namespace cv;

int main()
{
	////////////////////////////////////////////////////////////////////////
	gpu::MOG2_GPU pMOG2_g(30);
	pMOG2_g.history = 3000; //300;
	pMOG2_g.varThreshold = 64; //128; //64; //32;//;
	pMOG2_g.bShadowDetection = true;
	Mat Mog_Mask;
	gpu::GpuMat Mog_Mask_g;
	////////////////////////////////////////////////////////////////////////
	VideoCapture cap(0);//0);
	Mat o_frame;
	gpu::GpuMat o_frame_gpu;
	////////////////////////////////////////////////////////////////////////

	cap >> o_frame;
	if (o_frame.empty())
		return 0;

	////////////////////////////////////////////////////////////////////////

	unsigned long AAtime = 0, BBtime = 0;
	float sum = 0;
```

```
	int i = 0;

	//Mat rFrame;
	Mat showMat_r;
	namedWindow("origin");
	namedWindow("mog_mask");

	while (1)
	{

	//////////////////////////////////////////////////////////////////////////
		cap >> o_frame;
		if (o_frame.empty())
			return 0;
		o_frame_gpu.upload(o_frame);
		AAtime = getTickCount();

		//
		pMOG2_g.operator()(o_frame_gpu, Mog_Mask_g, -1);
		//
		Mog_Mask_g.download(Mog_Mask);
		BBtime = getTickCount();
		float pt = (BBtime - AAtime) / getTickFrequency();
		//float fpt = 1 / pt;
		sum += pt;
		i++;
		printf("gpu %.4lf \n", pt);
		o_frame_gpu.download(showMat_r);
		imshow("origin", showMat_r);
		imshow("mog_mask", Mog_Mask);


	//////////////////////////////////////////////////////////////////////////

		if (waitKey(10) > 0)
			break;
	}
	printf("average %.4lf \n", sum / i);
	system("Pause");
}
```

Лабораторная работа №7
«Работа в графической среде GraphEdit»

GraphEdit утилита Microsoft, входящая в состав DirectShow SDK. Может пропускать полученный с помощью одного из установленных в системе фильтров сигнал через любой другой кодек или фильтр, установленный и зарегистрированный в системе [3, 4].

Фильтр – единица операции в DirectShow, т.е. каждый фильтр выполняет одну операцию над медиа-потоком: будь то видеозахват с ТВ-тюнера или камеры или перекодирование видео-потока. На физическом уровне каждый фильтр представляет собой COM-компонент. Фильтры могут иметь входные и выходные контакты (pins). При соединении фильтров в определенном порядке посредством контактов, получается граф, который выполняет над медиа-потоком набор последовательных действий.

Существует три основных типа фильтров: фильтры-источники (source), фильтры-преобразователи (transform) и фильтры рендеринга (renderer). Фильтры-источники – фильтры для захвата видео или аудио с внешнего устройства (ТВ-тюнера, камеры, микрофона) и из медиа-файлов. Фильтры-преобразователи – фильтры для преобразования поступающих на них данных. Среди фильтров-преобразователей часто выделяют ещё 2 вида: фильтры Splitter для разделения на медиа-потока на несколько потоков (например, разделение медиа-потока на видео-поток и аудио-поток) фильтры MUX для совмещения медиа-потоков в единый медиа-поток. Фильтры рендеринга – фильтры для вывода медиа-потока на устройство (звуковую или видеокарту) или в файл.Часть фильтров поставляется с Windows, часть устанавливается вместе со сторонним программным обеспечением. Также в DirectShow есть возможность написания своих фильтров, для этого будут необходимы знания технологии COM.

Аудио-видео **контейнеры** - файл, в котором сохраняется видеозапись и служебная информация. Помимо собственно видеоряда и звуковой дорожки он должен содержать служебную информацию: какой формат применён для сжатия видео и звука, так называемый индекс (index, блок данных, который содержит адреса расположения конкретных участков записи — он используется во время перемотки), текстовые описатели (тэги, tags — название записи, автор, информация об авторских правах и прочее). Традиционный контейнер для видеозаписей — это AVI (Audio and Video Interleaved).

Задание на работу

1. Изучите функции и работу GraphEdit (GraphEditPlus) по ниже приведенным примерам.
2. Создайте свой проект 1: Запись видео и аудио потоков с цифровой камеры в один файл с одновременным выводом изображения и звука. Видео-поток должен быть сжат.
3. Создайте свой проект 2: Перекодировка форматов файлов, например конвертирование MP3 -> WAV.

Технология работы и примеры

1. Технология работы на примере проигрывания видеофайла

Жмем File->Render Media File ... - появляется окно с приглашением выбрать media-файл. Выбрали файл с именем new.avi и нажимаем OK. Получилось вот что:

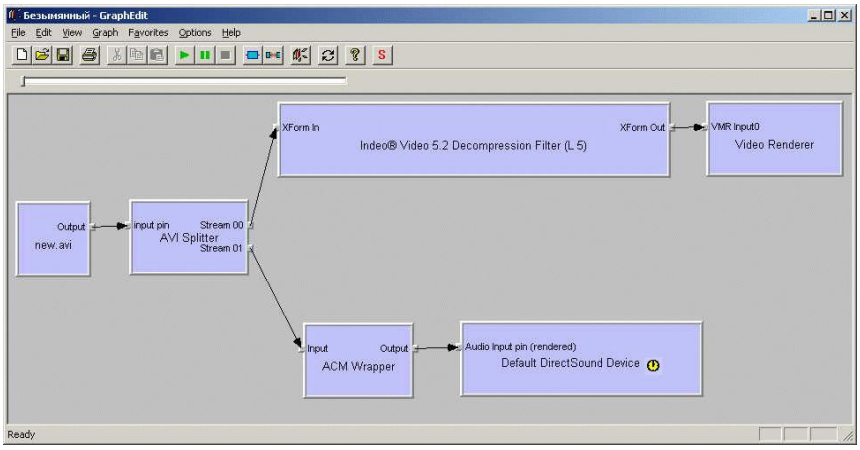


Рис. 1. Автоматически построенный граф фильтров

Первый прямоугольник символизирует наш исходный файл. Далее, по ходу стрелочки, видим следующий прямоугольник, в котором написано "AVI Splitter",- судя по названию, а также по двум исходящим из него стрелкам, он предназначен для того, чтобы взять данные от первого прямоугольника и разделить их на два потока аудио- и видео- потоки. Входы и выходы в прямоугольнички в терминологии DirectShow представляют собой входящие и исходящие контакты (InputPin и OutputPin), а сами прямоугольники - собственно фильтры. Вся эта схема есть так называемым графом фильтров. Выберем пункты меню Graph->Play и посмотрим наше видео-аудио. Остановим просмотр (Graph->Stop или красненький квадратик на панели инструментов).

На втором этапе повторим то, что мы уже сделали, но вручную. Для этого выберем Graph->Insert Filters:

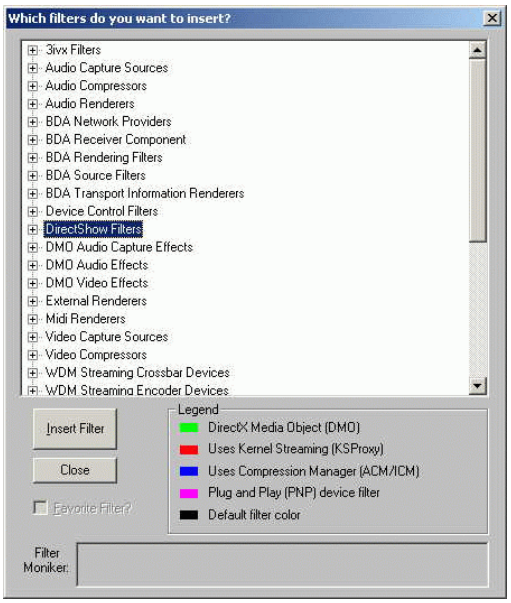


Рис. 2 Окно выбора фильтра

Нас пока будут интересовать DirectShow фильтры. Поэтому распахнем узел дерева "DirectShow Filters", найдем пункт "File Source (Async.)" и совершим на этом пункте двойной щелчок мышью (т.е. выберем этот фильтр). В появившемся окне диалога выбора медиа-файла укажем тот же, что и раньше, файл (new.avi). Появится прямоугольник с именем нашего файла. Если в той области, где

находится надпись "Output" щелкнуть правой кнопкой мышки, увидим меню:

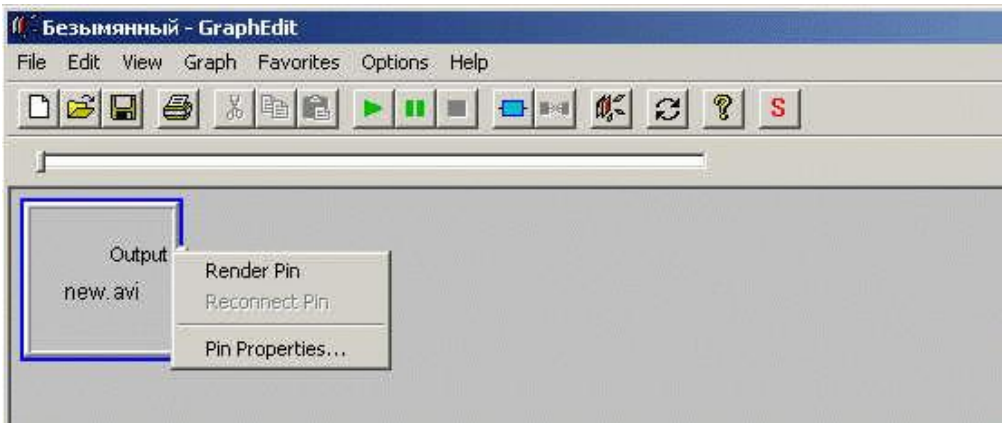


Рис.3 Автоматическое построение графа фильтров

Заметим, что выбор пункта "Render Pin" приведет к построению графа, с которым мы уже знакомы. Дальше из той же категории "DirectShow Filters" выберем "Avi Splitter Filter" фильтр и соединим наши два имеющиеся в данный момент фильтра, протянув мышкой от исходящего контакта первого фильтра к входящему второго:

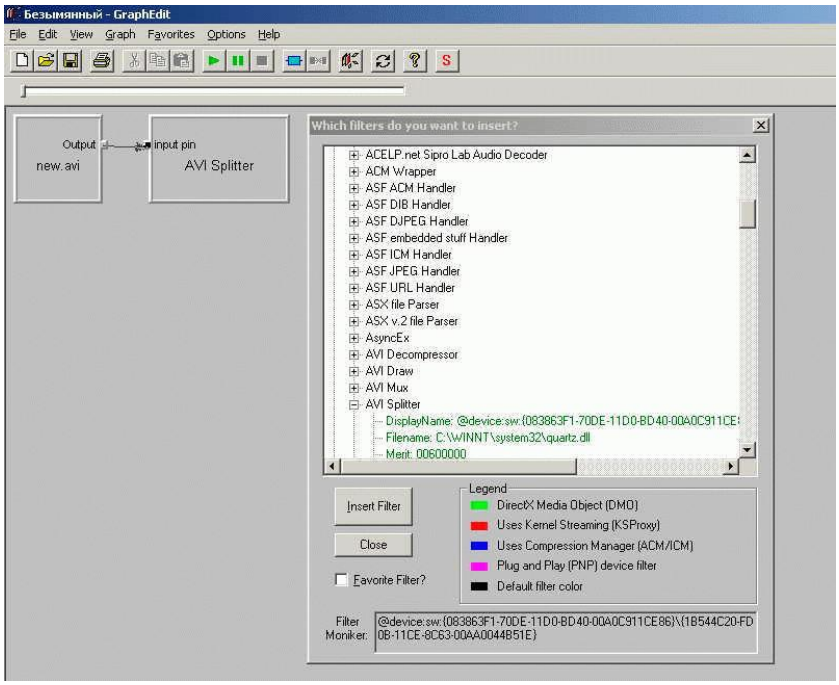


Рис.4 Построение графа фильтров вручную

Остальное очевидно. Смотрим на Рис.1, выбираем соответствующие фильтры и соединяем их. Понятно, что GraphEdit, когда его просишь проиграть медиа-файл, каким-то образом узнает,

какие нужны для этого фильтры и какие их контакты должны быть подключены. Разберемся с этим. Тем более, что это пригодится при написании собственных фильтров в том случае, если мы хотим сделать их доступными в процессе автоматического построения графа фильтров - утилита GraphEdit всего лишь вызывает API-функции для построения такого графа.

Каким образом GraphEdit узнает, какие нужны фильтры для проигрывания медиа-файла? Для этого используется специальный механизм - Intelligent Connect. Он описан в справке DirectShow. Intelligent connect затрагивает следующие методы интерфейсаIGraphBuilder:

Render

AddSourceFilter

RenderFile

Connect

Метод Render строит подсекцию графа. Он начинает обработку, начиная с первого несоединенного исходящего контакта и добавляет новые фильтры по мере необходимости. Стартовый фильтр (самый первый, т.е. - для нашего случая - File Source (Async)) уже должен быть в графе. На каждом новом шаге метод Render ищет фильтр, который можно соединить с текущим фильтром.

Для соединения каждого исходящего контакта метод Render проделывает следующие операции:

Если контакт поддерживает интерфейс IStreamBuilder, менеджер графа фильтра делегирует весь процесс методу контактаIStreamBuilder::Render. Предоставляя этот интерфейс, контакт принимает на себя ответственность за построения остатка графа, вплоть до собственно рендеринга. Впрочем, немногие контакты поддерживают этот интерфейс.

Менеджер фильтра графа пытается использовать фильтры, кешированные в памяти, если такие есть. На протяжении всего процесса "интеллектуального соединения" менеджер графа фильтра

пытается использовать кешированные фильтры из предыдущих шагов процесса.

Если граф фильтра содержит фильтры с несоединенными входными контактами, менеджер графа фильтра попытается соединить их потом. Можно принудительно вызвать метод Render для попытки соединения с каким-то специфическим фильтром перед вызовом метода Render.

И последнее, менеджер графа фильтра ищет в реестре, используя метод IFilterMapper2::EnumMatchingFilters.Он пытается сопоставить исходящим контактам представленных медиа-типов медиа-типы, находящиеся в реестре.

Каждый фильтр регистрируется с неким показателем (merit), числовой величиной, показывающей предпочтение фильтра в отношении других фильтров. Метод EnumMatchingFilters возвращает фильтры в порядке их этого показателя, с минимальным значением MERIT_DO_NOT_USE + 1. Игнорируются фильтры с показателем MERIT_DO_NOT_USE или меньшим. Фильтры также группируются в категории, определяемые посредством GUID. Эти категории имеют определенные показатели, и метод EnumMatchingFilters игнорирует любые из них с показателем MERIT_DO_NOT_USE или меньше, даже если фильтры в этой категории имеют и более высокие показатели.

Метод Render пытается соединить фильтры следующим образом:

Используя IStreamBuilder.

Пытаясь использовать кешированные фильтры.

Пытаясь использовать фильтры в графе.

Просматривая фильтры в реестре.

Продолжим, однако, рассмотрение методов, к которым имеет отношение Intelligent Connect.

Метод AddSourceFilter добавляет фильтр источника, который может произвести рендеринг указанного файла. Он просматривает реестр и сопоставляет расширению файла соответствующий протокол

или набор предопределенных проверочных байтов (check bytes), которые задают определенный шаблон. Словом, этот метод находит подходящий фильтр источника, создает экземпляр этого фильтра, добавляет его в граф и вызывает метод IFileSourceFilter::Load с соответствующим именем файла.

Метод RenderFile строит граф, отталкиваясь от имени файла. Внутри себя он использует AddSourceFilter для поиска корректного фильтра источника и Render для построения оставшейся части графа.

Метод Connect соединяет исходящие и входящие контакты. Этот метод добавляет, если необходимо, промежуточные фильтры, используя вариации алгоритма, используемого для метода Render:

Пытается напрямую соединить фильтры, без промежуточных фильтров.

Пытается использовать кешированные фильтры.

Пытается использовать фильтры в графе

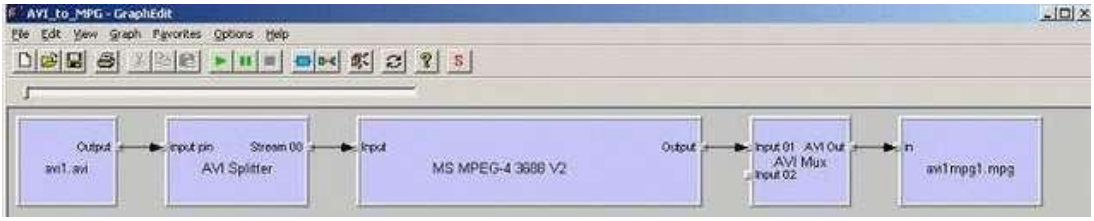
Просматривает фильтры в реестре.

2. Примеры

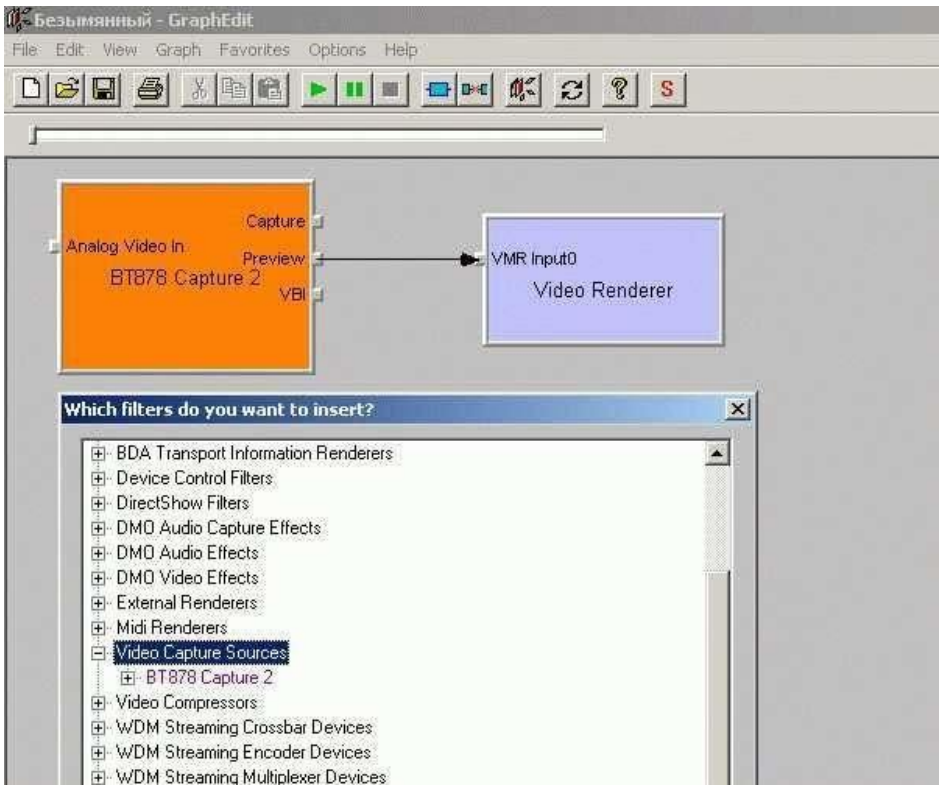
Конвертирование WAV -> MP3

Запускаем GraphEdit, выбираем из DirectShow фильтров фильтр FileSource (Async.), в качестве входного файла указываем ему, например, "Вход в Windows.wav", дальше выбираем фильтры Wave Parser, LAME MPEG Layer III Audio Encoder (если каких-то из предыдущих или последующих фильтров вы у себя не найдете, ищите их в наборах кодеков по ссылкам) и, наконец, фильтр Dump, который используется для записи,- он предложит задать имя файла, в который предполагается произвести конвертацию. Картинка будет иметь приблизительно такой вид:

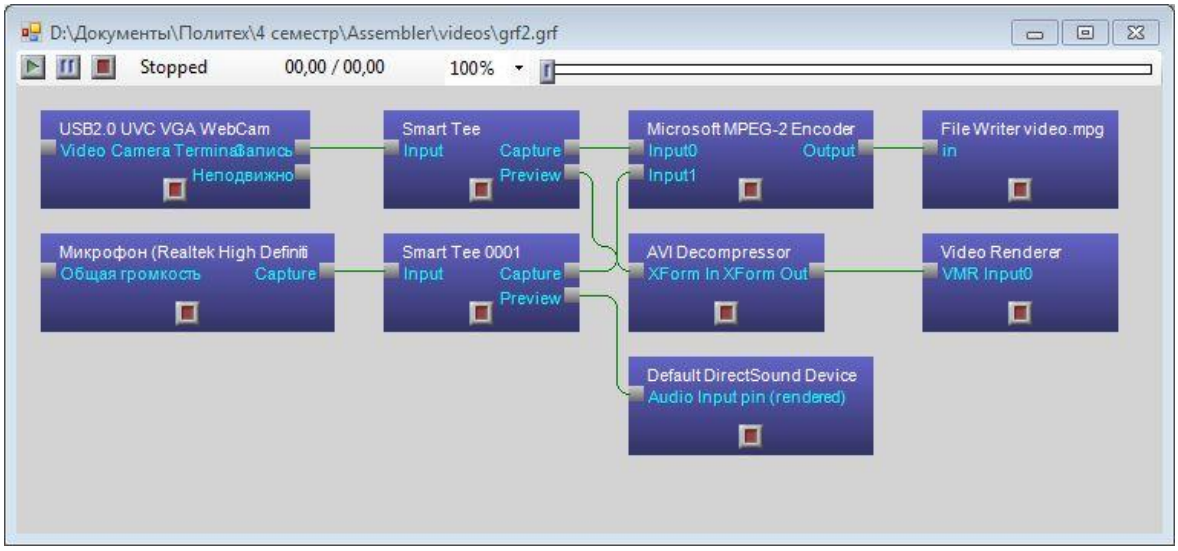
Конвертирование AVI->MP4



Захват видео



Запись со сжатием данных с веб камеры и микрофона в файл MPEG-2.



Можно продолжать эксперименты, по-разному соединять фильтры, накладывать эффекты, конвертировать файлы и т.д.

Лабораторная работа №8
"Захват, воспроизведение и запись видео файлов. Библиотека DirectShow/FFmpeg "

(4 часа в классе и самостоятельная работа)

DirectShow – мультимедийный фреймворк и интерфейс программирования приложений (API) – это универсальная библиотека от Microsoft для работы с аудио и видео [3]. Предоставляет широкий набор возможностей по вводу/выводу и редактированию аудио- и видео-потоков. Является одним из интерфейсов семейства DirectX, входит в Windows SDK.

Фреймворк (англ. Framework — каркас, структура) - структура (каркас) программной системы; программное обеспечение, облегчающее разработку и объединение разных компонентов большого программного проекта.

DirectShow основан полностью на технологии COM. Программирование DirectShow связано с графами. Графы строятся из элементов (фильтров), которыми являются кодеки, декомпрессоры и т.д. У каждого элемента есть каналы, входные и выходные, по которым элементы соединяются между собой. Несколько соединенных между собой элементов образуют граф.

Любая программа, которая использует DirectShow интерфейс, автоматически сможет использовать другие DirectShow компоненты, установленные в систему: декодеры видео, декодеры звука, splitter’ы (демультиплексоры) для извлечения аудио/видеопотоков из разных форматов файлов-контейнеров, фильтры для их обработки (например, для наложения субтитров). Это позволяет каждому пользователю конструировать подсистему для работы с видео из различных «кубиков» по своему вкусу — за их применение по назначению и автоматическое соединение в цепочки отвечает ОС.

Описание технологии построения графов фильтров приведено в предыдущей лабораторной работе при рассмотрении GraphEdit.

Ffmpeg (http://ffmpeg.org/) — набор свободных библиотек с открытым исходным кодом, которые позволяют записывать, конвертировать и передавать цифровые аудио- и видеозаписи в различных форматах. Название происходит от названия экспертной группы MPEG и FF, означающего «fast forward».

Ffmpeg состоит из следующих основных компонентов:

- Ffmpeg — утилита командной строки для конвертирования видеофайла из одного формата в другой. С её помощью можно также захватывать видео в реальном времени с TV-карты.
- Ffplay — простой медиаплеер.
- Ffprobe — консольная утилита, позволяющая собирать и отображать информацию о медиафайлах (как MediaInfo) и мультимедиа потоках, доступных устройствах, кодеках, форматах, протоколах и др.
- Libavcodec — библиотека со всеми аудио/видеокодеками.
- Libavformat — библиотека с мультиплексорами и демультиплексорами для различных аудио- и видеоформатов.
- Libavutil — вспомогательная библиотека со стандартными общими подпрограммами для различных компонентов ffmpeg.
- libpostproc — библиотека стандартных подпрограмм обработки видео.
- libswscale — библиотека для масштабирования видео.

Задание на выполнение работы

1. Напишите (загрузите готовую) программу, которая проигрывает (видео и аудео) *.avi файл с использованием библиотеки DirectShow.

2. Напишите программу, которая соответствует графу, полученному вами при работе с GraphEdit (захват, просмотр и запись видео и звука).

3*. Напишите программу, отображающую на экран график (сечение) указанной строки изображения с камеры. (Выполнение задания имеет высокий рейтинг)

4. Альтернативно, можно написать программу (проигрывание видео со звуком и с эффектом или другое) с использованием возможностей современных библиотек, например ffmpeg.

Список электронных ресурсов

1. DirectShow по-русски http://directshow.wonderu.com
2. DirectShow Теория http://2developers.net/cat/cpp
3. MSDN http://msdn.microsoft.com/ru-ru/library/windows/desktop/dd375454(v=vs.85).aspx
4. Книга "DirectShow и телевидение" http://dslev.narod.ru/dshow.html
5. Официальный сайт http://ffmpeg.org/

1. Основные элементы Direct Show

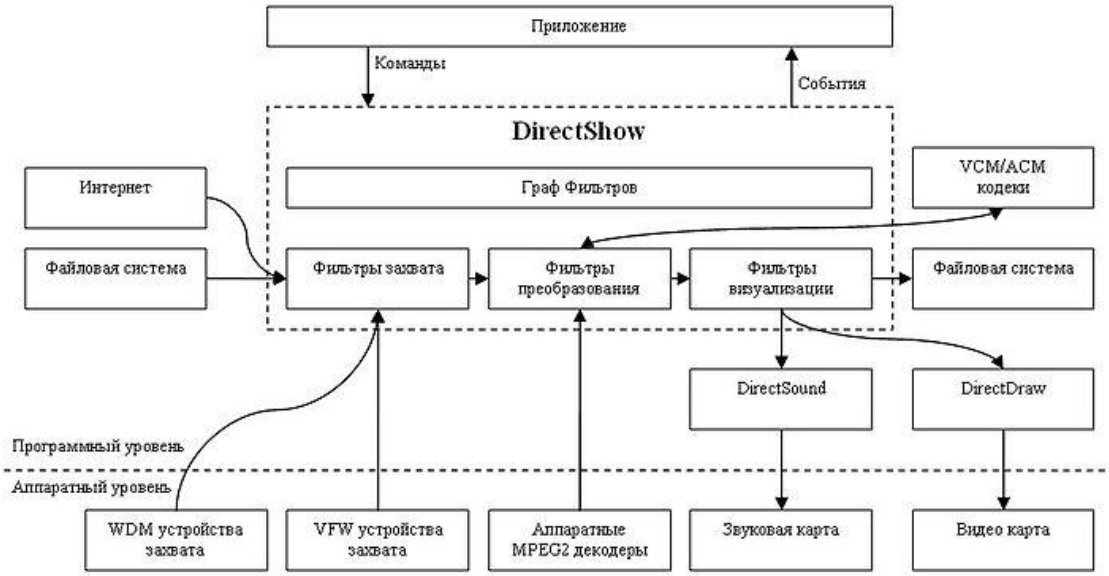
Filter. Фильтры выполняют работу по переработке данных, получаемых от входных *соединений*, и передают обработанные данные на выходные *соединения*.

Pin. Соединения отвечают за соединения *фильтров* между собой, указывая пути прохождения потоков.

Filter graph. Граф фильтров это совокупность активно существующих (как объектов) фильтров для некоторой программы. Фильтры могут как быть, так и не быть соединенными. Они просто являются узлами графа. Для организации графа используется специальный программный компонент - *Filter Graph Manager*.

Media Sample. Порция данных представляет обрабатываемые данные. Она имеет определенный *тип, формат*, объект ее представляющий со своим интерфейсом, обеспечивающим доступ к данным. Понятия тип и формат поясняются в описании реализации фильтра для передачи данных.

Allocator. Объект, отвечающий за выделение блоков памяти для хранения порций данных. При соединении фильтров от одного соединителя к другому должен быть выбран такой объект (создан или использован имеющийся от других соединений).



2. Component Object Model (COM)

Кратко технологию COM можно описать следующим образом:

Имеется некоторая функция, позволяющая по идентификатору *класса* (CLSID) - 128-битному числу (GUID) создать *объект* и вернуть указатель на *интерфейс* этого объекта. Интерфейс представляет список адресов *методов* объекта. Объект может иметь несколько интерфейсов. Для указания, какой именно из них следует получить, используется *идентификатор интерфейса* (IID) - тоже 128 битное число (GUID). Получив указатель на один интерфейс. далее можно получать и другие, так как все они должны, по крайней мере, реализовывать три метода - получение другого интерфейса по IID, добавления счетчика использования, и освобождения объекта при достижении счетчиком нулевого значения или уменьшения счетчика на 1. Считается, что эти три метода в списке идут первыми.

В языках программирования высокого уровня для использования этой технологии имеются специальные типы данных или даже специальные синтаксические конструкции: интерфейсы представляются *типизированными* записями указателей на методы. Тогда доступ к методам выполняется по именам.

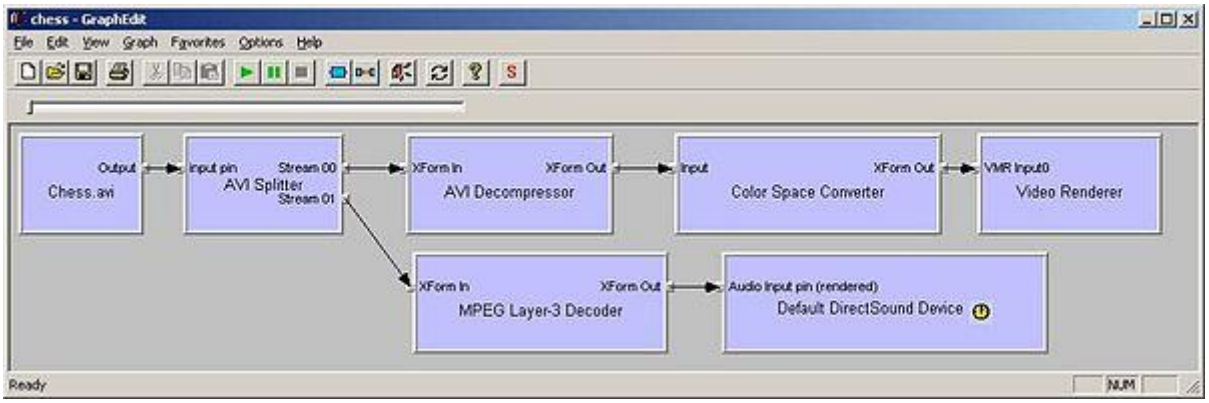
Связи между CLSID и объектами хранятся в реестре. В частности, там записаны пути и имена файлов библиотек,

содержащих код объектов. Реализованный объект оформляется в виде DLL библиотеки (одна библиотека может реализовывать создание разных объектов), экспортирующей функцию создания объекта. Если код только использует готовые объекты, нет необходимости знать эти детали.

Для использования готовых компонентов перечисленных функций достаточно. Для создания новых компонентов требуется реализовать объект с определенным поведением. Для этого в состав SDK включена библиотека классов, реализующих типовые операции для поведения фильтров и их соединений и примеры построений фильтров на ее основе.

Фильтры DirectShow разделены на три типа, соответственно и возможности DirectShow можно классифицировать соответствующим образом.

Фильтры захвата — предназначены для ввода мультимедиа данных в поток программы с различных физических устройств. В роли устройства могут быть как различного рода видео устройства (портативные видео камеры, веб-камеры, TV-тюнеры), так и аудио устройства (микрофон, модемная линия), а также данные могут быть получены и из файла (AVI, MPEG, MP3). DirectShow позволяет одновременно использовать несколько фильтров захвата, например: для одновременного захвата видео с веб-камеры и звука с микрофона. Количество одновременно используемых фильтров захвата ограничено лишь мощностью используемого компьютера.



Пример графа фильтров для воспроизведения AVI файла

Фильтры преобразования — предназначены для обработки поступающих данных из потока программы и последующей отправки преобразованных данных назад в поток к следующему типу фильтров. Этот тип фильтров может производить анализ данных, может полностью манипулировать аудиовидеоданными для создания сложных визуальных эффектов, или, просто объединять (или разъединять) аудио и видео каналы. В стандартной поставке вместе с операционной системой Windows корпорация Microsoft предоставляет небольшое количество готовых фильтров: кодеки (MPEG-1, MP3, WMA, WMV, MIDI), контейнеры (AVI, ASF, WAV), несколько сплитеров (или демультиплексоров) и мультиплексоров.[8] Другие же популярные фильтры: кодеки (MPEG-4, AAC, H.264, Vorbis) и контейнеры (Ogg, .mov, MP4) устанавливаются с различными сторонними программами.

Фильтры визуализации (рендеринга) — предназначены для вывода данных из потока в стандартное физическое устройство вывода, например, на монитор, на звуковую карту или в файл. По аналогии с фильтрами захвата фильтров визуализации также может быть несколько, например, для одновременного отображения видео на экране и записи этого же видео в файл.

Фильтры соединяются между собой через каналы, образуя поток, через который идут данные. Совокупность соединенных между собой фильтров образуют граф.

Синхронизация фильтров. Однако, просто правильно составленного графа не достаточно. Граф должен иметь возможность управлять потоком, например, поставить воспроизведение на паузу, остановить поток или напротив, воспроизвести его. Кроме того, нужно как-то синхронизировать фильтры, потому, что звук и видео при воспроизведении видео фильма делятся на два потока, которые могут один опережать другой. Для синхронизации DirectShow предоставляет программные часы, которые доступны для всех

фильтров графа. Часы работают с точностью до 100 нс. (Точность часов так же зависит от используемого вами оборудования). Когда программист посылает одну из трех основных команд стоп или пауза, она передается каждому фильтру графа в отдельности, фильтры в свою очередь должны быть способны обработать команду.

Интеллектуальное соединение (Intelligent Connect). Это механизм, который DirectShow использует для автоматического построения графа.

1) Если фильтр поддерживает интерфейс IStreamBulder, менеджер графа фильтра делегирует весь процесс ему. Реализуя этот интерфейс, фильтр принимает на себя ответственность за построение оставшееся части графа в низ, до самого рендерера.
2) Менеджер графа фильтра пытается использовать фильтры, кэшируемые в памяти. В процессе соединения фильтров менеджер графа может кэшировать фильтры соединенные на более ранних шагах.
3) Если менеджер графа фильтров содержит фильтры со свободными каналами, он перебирает их. Можно так же указать ему фильтр в ручную.
4) Если на предыдущих шагах дело закончилось ни чем, производится поиск в реестре.

3. Программные интерфейсы DirectShow.

В DirectShow каждый фильтр представляется COM-объектом. Граф фильтров – также COM-объект (его GUID – CLSID_FilterGraph). Для его построения можно пользоваться еще одним COM-объектом (построитель графа фильтров) (CaptureGraphBuilder2).

Объект “Граф фильтров” поддерживает многие необходимые при работе с видео интерфейсы:

IBasicVideo, позволяющий получить или установить некоторые настройки отображения видео (например, ширина и высота видео при захвате и при выводе на экран)

IMediaControl – управление работой всего графа (например, метод Run() выполняет действие аналогичное тому, которое выполняет пункт меню Graph\Play в программе GraphEdit)

IVideoWindow – интерфейс, содержащий методы управления окном вывода данных

IGraphBuilder – интерфейс построения графа фильтров.

4. Пример использования DirectShow

В следующем примере создается граф фильтров: файл видео – декомпрессор – вывод в окно. Для соединения фильтров используется метод интерфейса ICaptureGraphBuilder2 RenderStream, позволяющий соединить несколько фильтров в графе. Для добавления фильтра “Файл видео” – метод AddSourceFilter интерфейса IGraphBuilder.

Для простоты, большинство проверок удачного завершения функций опущено.

```
// Объявление типов
IGraphBuilder* pGraphBuilder = NULL; //Базовый интерфейс графа фильтров
ICaptureGraphBuilder2* pCaptureGraphBuilder2 = NULL;
//Интерфейс построителя графа фильтров
IMediaControl* pMediaControl = NULL; //Интерфейс управления потоком данных
IVideoWindow* pVideoWindow = NULL; // Интерфейс доступа к окну вывода видео
IBasicVideo* pBasicVideo = NULL; // Интерфейс доступа к параметрам видео
IBaseFilter *pSourceFile = NULL; //Фильтр, представляющий файл видео данных
long NativeVideoWidth=0,NativeVideoHeight=0; //Переменные служащие для хранения исходных размеров видео
HRESULT hr = CoCreateInstance(CLSID_FilterGraph, NULL,
CLSCTX_INPROC, IID_IGraphBuilder,
(void**)&pGraphBuilder);//Создание графа, запрос интерфейса построения
if FAILED(hr) MessageBox((CString)"Failed!");    //проверка
CoCreateInstance(CLSID_CaptureGraphBuilder2, NULL,
CLSCTX_INPROC, IID_ICaptureGraphBuilder2,
(void**)&pCaptureGraphBuilder2); //Создание построителя графа
```

```
pCaptureGraphBuilder2->SetFiltergraph(pGraphBuilder);
//Связывание построителя графа и самого графа
//добавление фильтра файла видео
pGraphBuilder->AddSourceFilter("clock.avi", L"",
(IBaseFilter**)&pSourceFile);
//соединение фильтров, фильтр декомпрессора добавиться
автоматически – в этом //преимущество метода RenderStream
перед аналогичными
pCaptureGraphBuilder2->RenderStream(NULL, NULL, pSourceFile,
NULL, NULL);
pGraphBuilder->QueryInterface(IID_IMediaControl,
(void**)&pMediaControl);
pGraphBuilder->QueryInterface(IID_IVideoWindow,
(void**)&pVideoWindow);
//установка окна, в котором будет выводиться видео
pVideoWindow->put_Owner((OAHWND)hVideoWnd);
pVideoWindow->put_WindowStyle(WS_CHILD);
pGraphBuilder->QueryInterface(IID_IBasicVideo,
(void**)&pBasicVideo);
pBasicVideo->GetVideoSize(&NativeVideoWidth,
&NativeVideoHeight);
pMediaControl->Run();
```

5. Практические указания по компиляции проекта с DirectShow

Для компиляции проектов в среде Visual Studio, необходимо включить файл <dshow.h>. Перед этим необходимо настроить среду компиляции (Tools\Options – Project and Solutions\VC++ Directories):

- В директории подключаемых файлов (Include) добавить папку Include из библиотеки DirectShow
- Аналогично в Library
- В свойствах текущего проекта (Proprties\Linker\Input\Additional Depencies дописать strmiids.lib)

Работа с устройством захвата видео

В случае, если необходимо использовать устройство захвата видео (например, Веб-камеру), необходимо получить список таких устройств, установленных в системе. Для этого служат специальные объекты – перечислители (*Enumerator*). Используя их методы, можно

получить имена устройств захвата, их свойства. Затем, получить указатель на фильтр, соответствующий данному устройству, добавить его в граф фильтров, соединить граф.

Полное описание см. DirectSow SDK.

Пример программы проигрывания *.avi файла

http://cadzone.ru/content/view/672/35/ - здесь дополнительное описание

```
#include <dshow.h>
#include <iostream>
#include <stdio.h>
#include <string>
#include <strmif.h>
#include <control.h>
using namespace std;
int main() {
     bool keyEnd=false;
     string str;
     IMediaSeeking *        pMediaSeeking;
    IGraphBuilder *    pGraphBuilder;
    IMediaControl *    pMediaControl;
    IMediaEvent *      pMediaEvent;
    CoInitialize( NULL );
    char s[]="Logo.avi";
     //Create a new COM-object
     //pGraphBuilder - new object
    CoCreateInstance( CLSID_FilterGraph, NULL, CLSCTX_INPROC,
IID_IGraphBuilder, (LPVOID *)&pGraphBuilder );
     //Получение интерфейса управления
    pGraphBuilder->QueryInterface( IID_IMediaControl, (LPVOID
*)&pMediaControl );
     //Второй интерфейс управления
     pGraphBuilder->QueryInterface( IID_IMediaSeeking, (LPVOID
*)&pMediaSeeking );
     //Получение интерфейса сообщений
    pGraphBuilder->QueryInterface( IID_IMediaEvent, (LPVOID
*)&pMediaEvent );
    pMediaControl->RenderFile( L"Logo.avi" );
     while(keyEnd!=true) {
          //system("clear");
          cout << "Please, enter the command\n";
          str="";
```

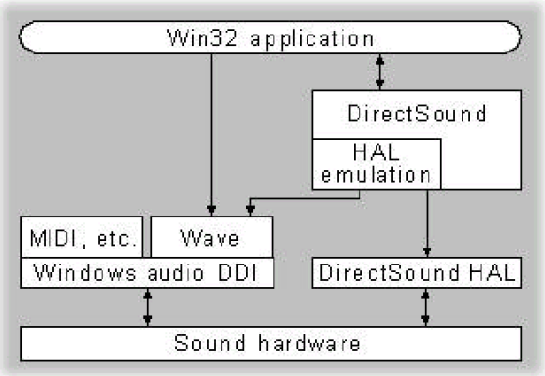
```
        cin >> str;
        if(str == "play") {
             pMediaControl->Run();
        }    if(str == "pause") {
             pMediaControl->Pause();
        }    if(str == "stop") {
             pMediaControl->Stop();
        }    if(str == "end") {
             keyEnd=true;
        }
 }
pMediaControl->Release();
pGraphBuilder->Release();
CoUninitialize();
return 0;
                              }
```

Лабораторная работа по теме №9 "Проигрывание звуковых файлов. Формат WAV. Библиотека DirectSound"

(4 часа в классе и самостоятельная работа)

Подсистема DirectSound обеспечивает приложениям практически непосредственный доступ к аппаратуре звукового адаптера через hardware-abstraction layer (HAL) - интерфейс, который осуществляется драйвером аудио устройства. DirectSound HAL предоставляет следующие возможности:

- Получение и утрата контроля над аудио аппаратурой
- Описывает возможности аудио аппаратуры
- Выполняет указанные операции, когда есть доступ к аппаратуре



- Выполняет сообщение об ошибке, когда аппаратура недоступна

Приложению не приходится вникать в детали программирования того или иного адаптера - это остается прерогативой HAL. Вместо этого приложению предоставляется модель современного звукового адаптера, предельно приближенная к реальности, с минимальным уровнем абстракции. С уровня приложения имеются следующие основные возможности:

1. Смешивание сигналов. DirectSound позволяет приложению просто задавать несколько источников звука, которые будут проиграны одновременно. Количество таких источников ограничено только доступной памятью и быстродействием аппаратуры.
2. Объемный звук. Работа расширенной базовой модели, DirectSound3D, основана на другой концепции, позволяющей размещать монофонические источники звука в пространстве. Для каждого источника, а также для самого слушателя, задаются

координаты в пространстве, ориентация, направление и скорость перемещения. DirectSound3D обрабатывает все источники и создает для слушателя объемную и реалистичную звуковую картину с учетом интерференции, затухания, направленности, эффекта Доплера и т.п.

3. Звуковые буферы. DirectSound предоставляет приложению почти прямой доступ к аппаратным буферам адаптера. Определяют первичный и вторичные буферы. Если в архитектуре адаптера один из аппаратных буферов является основным, его называют первичным (primary). Остальные буферы, занимающие подчиненное положение, называются вторичными (secondary). Вторичные звуковые буфера содержат одиночный сэмпл или аудио поток Обычно звуки из вторичных буферов смешиваются воедино в первичном буфере, откуда и поступают на ЦАП адаптера.

Задание на работу

1. Изучите функции DirectSound и технологию их применения в среде Microsoft Visual C++. В первую очередь используйте SDK.
2. Загрузите проект проигрывания wav файла из примеров SDK.
3. Добавьте в проект новые функции оригинального проигрывания, например, эхо, одновременное проигрывание нескольких файлов или др. У каждого студента свои функции.
4. Напишите программу, которая изменяет заголовок WAV-файла (вносит ваше имя в заголовок).

Введение в DirectSound

DirectSound—программный интерфейс, входящий в семейство «мультимедийных» интерфейсов DirectX. Эта библиотека была разработана специально для поддержки высокопроизводительных мультимедиа приложений и включает в себя средства для работы с видеокартами, 3D ускорителями, звуковыми картами и MIDI устройствами, с графическими, аудио и видео компрессорами,

устройствами пользовательского ввода, а также сетевыми компонентами системы.

Название DirectX трактуется буквально — «прямой, непосредственный интерфейс X». Классические системные интерфейсы с видео-, звуковыми и игровыми адаптерами относятся к Windows первых версий, когда внутренняя организация большинства адаптеров была достаточно разношерстной, многие решения находились в стадии отработки, а приложениям требовалось в первую очередь отображать прямоугольные окна, текст и графики, проигрывать длительные непрерывные звукозаписи и т.п. С массовым переходом производителей игр на платформу Windows и развитием видеотехнологий выяснилось, что большинство классических интерфейсов слишком абстрактны, поэтому при работе с конкретным устройством возникают заметные накладные расходы, ощутимо снижающие быстродействие; при частой и хаотичной перерисовке элементов экрана, выводе коротких одновременных звуков выполняется множество лишних операций. Семейство DirectX было разработано именно для того, чтобы приблизить аппаратуру к приложению, предоставив эффективный интерфейс.

DX состоит из нескольких компонентов:

1. DirectX Graphics комбинирует в себе компоненты DirectDraw и Direct3D предыдущих версий DX в один API, который может быть использован для программирования всей графики.

2. DirectX Audio комбинирует компоненты DirectSound и DirectMusic предыдущих версий DX в один API, который может быть использован для программирования звука и музыки

3. DirectInput предоставляет поддержку множества устройств ввода

4. DirectPlay предоставляет поддержку сетевых игр

5. DirectShow предоставляет высококачественный захват и воспроизведение мультимедийных потоков

6. DirectSetup просто API, предоставляющий установку компонентов DirectX

Назначение и структура DirectSound

Подсистема DirectSound обеспечивает приложениям практически непосредственный доступ к аппаратуре звукового адаптера. Этот факт вовсе не означает, что приложению приходится вникать в детали программирования того или иного адаптера. Вместо этого приложению предоставляется модель современного звукового адаптера, предельно приближенная к реальности, с минимальным уровнем абстракции. При таком подходе общение приложения с адаптером сопровождается минимальными накладными расходами, и в то же время избавляет приложение от излишних подробностей программирования аппаратуры.

Подсистема DirectSound построена по объектно-ориентированному принципу в соответствии с моделью COM (Component Object Model — модель объектов-компонентов, или составных объектов) и состоит из набора интерфейсов. Каждый интерфейс отвечает за объект определенного типа — устройство, буфер, службу уведомления и т.п. По сути, интерфейс представляет собой обычный набор управляющих функций, или методов, организованных в класс объектно-ориентированного языка.

DirectSound не поддерживает звуковые форматы, отличные от PCM (Pulse Code Modulation). PCM - формат звуковых данных в "чистом" виде, т.е. его значения показывают амплитуду звука, полученную при определённой частоте записи. Назначение DirectSound — исключительно эффективный вывод звука, а операции по преобразованию форматов остаются в ведении приложений, которые могут использовать для этого подсистему сжатия звука (ACM).

Основы воспроизведения в DirectSound

Для воспроизведения в DirectSound используются вторичные буферы, микшер и первичный буфер.

Вторичным буфером (secondary buffer) называется объект, содержащий звуковую информацию. В буфере может располагаться как вся информация, которую предполагается воспроизвести (в этом случае говорят, что он является статическим - static buffer), так и её часть (буфер является потоковым - streaming buffer). Преимущество потоковых буферов заключается в том, что, поскольку они содержат только часть воспроизводимого звука, то, записывая в них информацию по мере воспроизведения, можно проиграть звуковые данные любого размера. Статические же буферы, в отличие от потоковых содержат всё, что предполагается воспроизвести, поэтому заполняются только один раз, и, следовательно, более легки в использовании. Независимо от того, каким является буфер - потоковым или статическим - информация в нём должна быть записана в формате PCM (Pulse Code Modulation), т.к. это единственный формат, поддерживаемый DirectSound (для остальных форматов приходится использовать дополнительные средства, чтобы привести их к типу PCM).

Микшер DirectSound (DirectSound mixer) - механизм, отвечающий за совмещение информации, поступающей из проигрываемых вторичных буферов, в одно целое и отправке её в первичный буфер. Микшировать можно любое количество вторичных буферов. Единственным ограничением является свободное процессорное время.

Первичный буфер (Primary buffer)- объект, содержащий готовую для передачи на устройство вывода звуковую информацию. В отличие от вторичных буферов, первичный не может создаваться программистом и всегда присутствует в единственном экземпляре.

Вторичные буферы по мере воспроизведения микшируются и приводятся к формату первичного буфера.

Связь между вышеперечисленными элементами представлена на рисунке (из расчёта, что запись в первичный буфер программистом не ведётся - в противном случае микшер в процессе участвовать не будет

и программист будет ответственен за микширование звуков вручную).

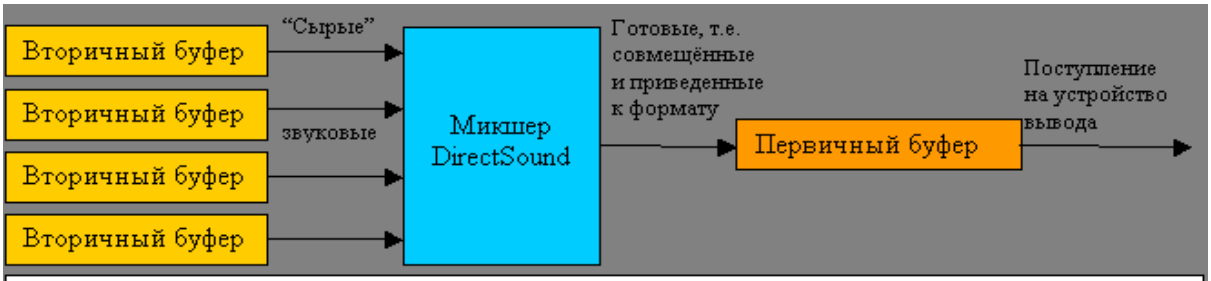


Рис. 1. Передача звуковых данных при использовании DirectSound

Основы программирования

Для того, чтобы использовать функции DirectSound в вашем проекте необходимо совершить следующие шаги:

1. Необходимо включить в ваш проект файл заголовка “Dmusici.h”. Включение этого заголовка ведет к включению 3-х следующих файлов:

- Dmusicc.h – основные функции DirectSound
- Dmerr.h – возвращаемые значения функций DirectSound
- Dsound.h – DirectSound API

2. В свойствах проекта добавить библиотеки: dsound.lib winmm.lib

3. Если это потребуется, то указать в свойствах проекта пути к заголовочным файлам и библеотекам DirectX.

Поскольку в наборе функций DirectSound нет функций, которые реализуют открытие Wave файла, то можно воспользоваться классом CWaveFile, который находится в файле Dsutil.h. Но в этом заголовочном файле есть еще много полезных классов, которые реализуют все функции DirectSound, но предоставляют для разработчика более удобную среду, по этому можно пользоваться именно ими. Для этого нам нужно подключить к проекту следующие файлы: Dsutil.cpp и Dxutil.cpp. Находятся они в следующей папке: [путь, где установлен DirectX SDK]\Samples\C++\Common\.

Рассмотрим по подробнее эти классы:

1. CWaveFile – класс используется для того, чтобы читать и писать в WAV файлы, а также читать ресурсы WAV или WAV, находящиеся в памяти. Объект этого класса создется CSoundManager при создании объекта CSound.

```
    HRESULT Open(LPTSTR strFileName, WAVEFORMATEX* pwfx, DWORD
dwFlags);
    HRESULT OpenFromMemory(BYTE* pbData, ULONG ulDataSize,
        WAVEFORMATEX* pwfx, DWORD dwFlags);
    HRESULT Close();
    HRESULT Read(BYTE* pBuffer, DWORD dwSizeToRead, DWORD*
pdwSizeRead);
    HRESULT Write(UINT nSizeToWrite, BYTE* pbData, UINT*
pnSizeWrote);
    DWORD   GetSize();
    HRESULT ResetFile();
    WAVEFORMATEX* GetFormat() { return m_pwfx; };
```

Метод класса	Описание
Close	Закрытие файла. Если в файл производилась запись, то сначала сохранит данные.
GetFormat	Возвращает структуру WAVEFORMATEX, которая описывает wave файл.
GetSize	Возвращает размер загруженного файла.
Open	Открывает или создает файл или загружает ресурс, загружая информацию о нем. Параметр strFileName – имя файла или ресурса. Параметр pwfx – информация об WAVE объекте Параметр dwFlags - WAVEFILE_READ, для чтения и WAVEFILE_WRITE для записи.
OpenFromMemory	Делает так, чтобы последующие запросы чтения происходили из памяти, а не файла. Параметр pbData указывает на начало данных в памяти.
Read	Читает данные из файла, ресурса или памяти и копирует в адреса, полученные при выполнении

	IDirectSoundBuffer8::Lock. Параметр dwSizeToRead – желаемое кол-во прочитанных данных. Параметр pdwSizeToRead – реальное кол-во прочитанных данных.
ResetFile	Повторно устанавливает указатель на начало файла.
Write	Записывает данные в файла из адресов, полученные при выполнении IDirectSoundBuffer8::Lock.

2. CSound – типовой класс, который представляет звук в одном или более статических буферах. Объект этого класса может использоваться, чтобы проиграть множество образцов звуковых эффектов. Объект обычно создается через класс CSoundManager.

```
    CSound(LPDIRECTSOUNDBUFFER* apDSBuffer, DWORD
dwDSBufferSize,
        DWORD dwNumBuffers, CWaveFile* pWaveFile, DWORD
dwCreationFlags);
    virtual ~CSound();

    HRESULT Get3DBufferInterface(DWORD dwIndex,
        LPDIRECTSOUND3DBUFFER* ppDS3DBuffer);
    HRESULT FillBufferWithSound(LPDIRECTSOUNDBUFFER pDSB,
        BOOL bRepeatWavIfBufferLarger);
    LPDIRECTSOUNDBUFFER GetFreeBuffer();
    LPDIRECTSOUNDBUFFER GetBuffer(DWORD dwIndex);
    HRESULT Play(DWORD dwPriority = 0, DWORD dwFlags = 0,
        LONG lVolume = 0, LONG lFrequency = 0);
    HRESULT Play3D(LPDS3DBUFFER p3DBuffer, DWORD dwPriority =
0, DWORD dwFlags = 0, LONG lFrequency = 0);
    HRESULT Stop();
    HRESULT Reset();
    BOOL    IsSoundPlaying();
```

Метод класса	Описание
FillBufferWithSound	Заполняет указанный буфер данными от WAV файла, который передавали к объектному

	конструктору. Если буфер является больше чем данные, данные могут произвольно быть повторены; иначе остальная часть буфера заполнена тишиной.
Get3DBufferInterface	Возвращает объект IDirectSound3DBuffer8 для указанного буфера, если буфер имеет трехмерные возможности.
GetBuffer	Возвращает объект IDirectSoundBuffer для указанного буфера.
GetFreeBuffer	Возвращает объект IDirectSoundBuffer первого буфера, который не играет. Если все буфера играют, метод возвращает случайно выбранный буфер.
IsSoundPlaying	Возвращает переменную, которая указывает, играет ли указанный буфер.
Play	Включает проигрывание буфера. Флажок DSBPLAY_LOOPING ставят, чтобы повторить звук; Параметр *lVolume* в сотых децибелов (где 0 – полная громкость, а -10 000 – тишина) Параметр *lFrequency* - частота в герцах, которая добавляется к основной частоте звука.
Play3D	Включает проигрывание трехмерного буфера. Этот метод подобен методу Play, но приложение также передает в DS3DBUFFER структуру, описывающую трехмерные параметры, которые будут установлены.
Reset	Устанавливает указатель на всех буферах на начало.
Stop	Останавливает воспроизведение всех буферов.

3. CSoundManager – класс нужен для того, чтобы создать и инициализировать DirectSound, обращаться к первичному буферу, и создавать вторичные буфера.

```
    HRESULT Initialize( HWND hWnd, DWORD dwCoopLevel );
    inline  LPDIRECTSOUND8 GetDirectSound() { return m_pDS; }
    HRESULT SetPrimaryBufferFormat( DWORD dwPrimaryChannels,
        DWORD dwPrimaryFreq, DWORD dwPrimaryBitRate );
    HRESULT Get3DListenerInterface(LPDIRECTSOUND3DLISTENER*
ppDSListener );
    HRESULT Create( CSound** ppSound, LPTSTR strWaveFileName,
        DWORD dwCreationFlags = 0, GUID guid3DAlgorithm =
GUID_NULL, DWORD dwNumBuffers = 1 );
    HRESULT CreateFromMemory( CSound** ppSound, BYTE* pbData,
ULONG ulDataSize, LPWAVEFORMATEX pwfx, DWORD dwCreationFlags =
0, GUID guid3DAlgorithm = GUID_NULL, DWORD dwNumBuffers = 1 );
    HRESULT CreateStreaming( CStreamingSound**
ppStreamingSound, LPTSTR strWaveFileName, DWORD
dwCreationFlags, GUID guid3DAlgorithm, DWORD dwNotifyCount,
DWORD dwNotifySize, HANDLE hNotifyEvent );
```

Метод класса	Описание
Create	Создает объект CSound, который ассоциирован с WAV файлом и содержит указанное число буферов. Значения параметров *dwCreationFlags* и *guid3DAlgorithm* совпадают с таковыми в DSBUFFERDESC.
CreateFromMemory	Создает объект CSound, который ассоциирован с участком памяти и содержит указанное число буферов. Значения параметров *dwCreationFlags* и *guid3DAlgorithm* совпадают с таковыми в DSBUFFERDESC.
CreateStreaming	Создает объект CStreamingSound, который ассоциирован с WAV файлом и содержит единственный потоковый буфер.
Get3DListenerInterface	Возвращает объект IDirectSound3DListener8.
GetDirectSound	Возвращает объект IDirectSound8, созданный при выполнении функции Initialize.
Initialize	Создает объект DirectSound и устанавливает

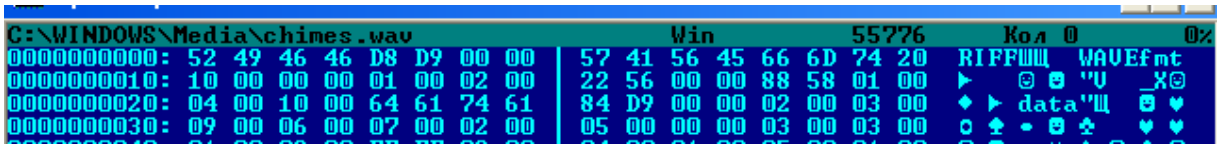
	его уровень доступа к звуковой карте.
SetPrimaryBufferFormat	устанавливает формат первичного буфера. Этот метод не должен использоваться большинством приложений.

Формат WAV файла.

WAV файл - это звуковой файл формата RIFF. WAV-файл состоит из трех блоков – двух заголовочных и одного блока звуковых данных. Первый блок имеет идентификатор "RIFF", за которым в 4-х байтах следует размер файла (без учета первых 8 байтов). В следующих 4-х байтах стоит идентификатор "WAVE", указывающий тип RIFF-файла. (первый заголовок 12 байт)

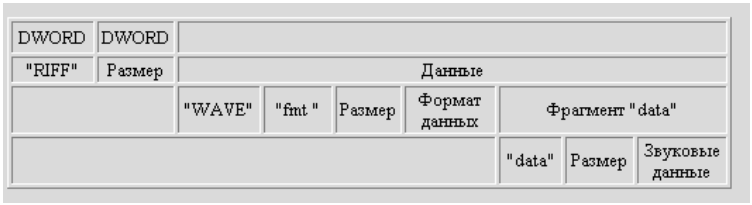
Следующий заголовочный блок содержит байты в следующем порядке:

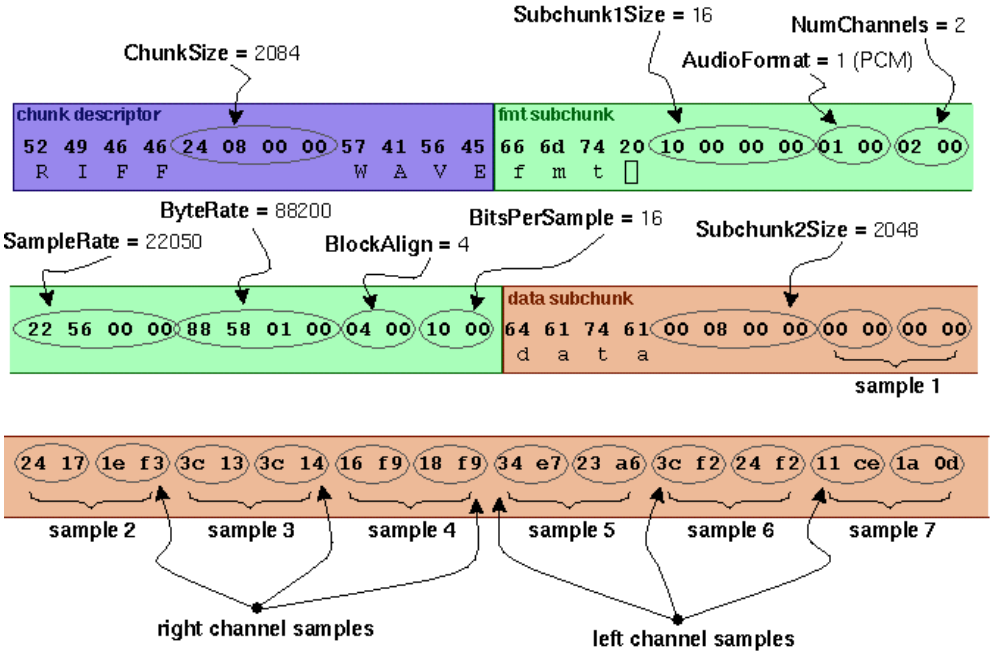
4 байта – идентификатор "fmt_"
4 байта – число 16 + (размер данных блока)
2 байта – информация о кодеке (1 для PCM) – код сжатия
2 байта – число каналов (1 – моно, 2 – стерео)
4 байта – частота дискретизации (при 44100 Гц – AC44H)
4 байта – число байтов в секунду
2 байта – число байтов на один отсчет
2 байта – число битов на выборку В (если В не кратно 8, то выборки дополняются нулями до целого числа байтов).



Последний блок – данные отсчетов – начинается идентификатором "data", за которым расположены 4 байта – размер блока, а затем сами данные. В случае стереоотсчета данные для обоих каналов следуют друг за другом: сначала выборка для левого канала, затем выборка для правого, и т.д.

Файл может дополнительно содержать фрагменты других типов, поэтому не следует думать, что заголовок wav-файла имеет фиксированный формат. Например, в файле может присутствовать фрагмент "LIST" или "INFO", содержащий информацию о правах копирования и другую дополнительную информацию.





INFO - специальный ListType для хранения информации о содержимом файла. INFO не влияет на то, как программы работают с файлом, эта информация (большей частью) показывается пользователю.

Список chunk'ов для INFO:

IARL (Archival Location) - место архивного хранения документа

IART (Artist) - список авторов произведения (стандартный тэг, отображается практически во всех плеерах).

ICMS (Commissioned) - список лиц, предоставивших содержимое файла.

ICMT (Comments) - комментарий. (отображается практически по всех плеерах)

ICOP (Copyright) - информация об авторских правах.

ICRD (Creation date) - Дата создания оригинального произведения. Формат YYYY-MM-DD.

ICRP (Cropped) - данные об обрезке произведения.

DIM (Dimensions) - Физические размеры оригинала.

IENG (Engineer) - фамилии, создававших файл.

IGNR (Genre) - жанр. (частично поддерживается)

Проигрыватель Wav файлов.

Теперь на основе этих классов не сложно написать проигрыватель Wav файлов.

```
CSoundManager* g_pSoundManager;
CSound*        g_pSound;
g_pSoundManager = new CSoundManager(); //Создание объекта
g_pSoundManager->Initialize( NULL, DSSCL_PRIORITY ) ;
//Инициализация
g_pSoundManager->SetPrimaryBufferFormat( 2, 22050, 16 ) ;
//Устанавливается
// первичный буфер (2 – кол-во каналов (стерео); 22050 –
частота дискретизации; 16 – “битность”)
//Будем считать что в strFileName уже храниться имя Wav файла.
g_pSoundManager->Create( &g_pSound, strFileName, 0, GUID_NULL
);
// Создаем буфер в g_pSound, который уже ассоциирован с файлом
g_pSound->Play( 0, DSBPLAY_LOOPING);
// Проигрываем открытый файл.
//  Здесь ждем действий пользователя, и когда он прекратит
прослушивание…
g_pSound->Stop();
// … останавливаем воспроизведение.
// Естественно, что когда программа будет завершаться надо
освободить занимаемую память.
delete g_pSound;
delete g_pSoundManager;
```

Лабораторная работа №10 "MPI. Захват и обработка видео"

В данной работе (4 часа в классе и самостоятельная работа) предлагается изучить основы технологии MPI (Message Passing Interface – Интерфейс передачи сообщений). Изучение проходит на примере разработки программы, которая захватывает и обрабатывает видео с применением директив OpenCV. В проекте используется MPI фирмы Microsoft.

Задание на выполнение работы

ЧТО НУЖНО ИЗЧИТЬ

1. Технологию использования MPI в Visual Studio.
2. Основные функции MPI.

ЧТО НУЖНО СДЕЛАТЬ

1. Загрузите и запустите первую программу Hello World (Приложение 1).
2. Загрузите и запустите программу с функциями MPI и OpenCV (Приложение 3).
3. Модифицируйте программу, включите в нее директивы синхронизации потоков.
4. (Для шустрых) Напишите программу не только с функциями MPI и OpenCV, но и директивами CUDA (используйте возможности OpenCV).

Message Passing Interface (MPI). Основные понятия

MPI - библиотека функций, предназначенная для поддержки работы параллельных процессов в терминах передачи сообщений.

Каждый процесс в рамках MPI выполняется в своем адресном пространстве. В результате передачи данных от одного процесса другому, производится их копирование из одного адресного пространства в другое. В рамках передачи также гарантируется синхронизация процессов.

Номер процесса - целое неотрицательное число, являющееся уникальным атрибутом каждого процесса.

Атрибуты сообщения - номер процесса-отправителя, номер процесса-получателя и идентификатор сообщения. Для них заведена структура *MPI_Status*, содержащая три поля: *MPI_Source* (номер процесса отправителя), *MPI_Tag* (идентификатор сообщения), *MPI_Error* (код ошибки); могут быть и добавочные поля.

Процессы объединяются в *группы*, могут быть вложенные группы. Внутри группы все процессы перенумерованы. С каждой группой ассоциирован свой *коммуникатор*. Поэтому при осуществлении пересылки необходимо указать идентификатор группы, внутри которой производится эта пересылка. Все процессы содержатся в группе с предопределенным идентификатором *MPI_COMM_WORLD*.

Основные процедуры, использующиеся в программе:

- **MPI_Init** - инициализация параллельной части приложения. Реальная инициализация для каждого приложения выполняется не более одного раза, а если MPI уже был инициализирован, то никакие действия не выполняются и происходит немедленный возврат из подпрограммы. Все оставшиеся MPI-процедуры могут быть вызваны только после вызова MPI_Init. Возвращает: в случае успешного выполнения - MPI_SUCCESS, иначе - код ошибки.
- **MPI_Finalize** - завершение параллельной части приложения. Все последующие обращения к любым MPI-процедурам, в том числе к *MPI_Init*, запрещены. К моменту вызова *MPI_Finalize* некоторым процессом все действия, требующие его участия в обмене сообщениями, должны быть завершены.
- **MPI_Comm_size** - Определение общего числа параллельных процессов в группе *comm.*
- **MPI_Comm_rank** - Определение номера процесса в группе *comm*. Значение, возвращаемое по адресу *&rank*, лежит в диапазоне от 0 до *size_of_group-1*.
- **int MPI_Send(void* buf, int count, MPI_Datatype datatype, int dest, int msgtag, MPI_Comm comm)**
  - *buf* - адрес начала буфера посылки сообщения
  - *count* - число передаваемых элементов в сообщении
  - *datatype* - тип передаваемых элементов
  - *dest* - номер процесса-получателя
  - *msgtag* - идентификатор сообщения
  - *comm* - идентификатор группы

  Блокирующая посылка сообщения с идентификатором msgtag, состоящего из count элементов типа datatype, процессу с номером dest. Все элементы сообщения расположены подряд в буфере buf. Значение count может быть

нулем. Тип передаваемых элементов datatype должен указываться с помощью предопределенных констант типа.

- **int MPI_Recv(void* buf, int count, MPI_Datatype datatype, int source, int msgtag, MPI_Comm comm, MPI_Status *status)**
  - OUT *buf* - адрес начала буфера приема сообщения
  - *count* - максимальное число элементов в принимаемом сообщении
  - *datatype* - тип элементов принимаемого сообщения
  - *source* - номер процесса-отправителя
  - *msgtag* - идентификатор принимаемого сообщения
  - *comm* - идентификатор группы
  - OUT *status* - параметры принятого сообщения

  Прием сообщения с идентификатором msgtag от процесса source с блокировкой. Число элементов в принимаемом сообщении не должно превосходить значения count. Если число принятых элементов меньше значения count, то гарантируется, что в буфере buf изменятся только элементы, соответствующие элементам принятого сообщения.

Пример простой программы с описанием

```
#include <mpi.h>
#include <stdio.h>

int main(int argc, char** argv) {
    // Initialize the MPI environment
    MPI_Init(NULL, NULL);

    // Get the number of processes
    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);

    // Get the rank of the process
    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

    // Get the name of the processor
    char processor_name[MPI_MAX_PROCESSOR_NAME];
    int name_len;
    MPI_Get_processor_name(processor_name, &name_len);

    // Print off a hello world message
    printf("Hello world from processor %s, rank %d out of %d
processors\n",
           processor_name, world_rank, world_size);

    // Finalize the MPI environment.
    MPI_Finalize();
}
```

Первым шагом к созданию программы MPI является включение заголовочных файлов MPI с помощью #include <mpi.h>. После этого среду MPI необходимо инициализировать с помощью:

```
MPI_Init(    int* argc,    char*** argv)
```

Во время MPI_Init создаются все глобальные и внутренние переменные MPI. Например, коммуникатор формируется вокруг всех процессов, которые были порождены, и каждому процессу присваиваются уникальные ранги. В настоящее время MPI_Init принимает два аргумента, которые не нужны, и дополнительные параметры просто оставляются как дополнительное пространство на случай, если в будущих реализациях они могут понадобиться.

После MPI_Init, есть две основные функции, которые вызываются. ти две функции используются почти во всех программах MPI, которые вы напишите.

```
MPI_Comm_size(    MPI_Comm communicator,    int* size)
```

MPI_Comm_size возвращает размер коммуникатора. В нашем примере MPI_COMM_WORLD(который построен для нас MPI) охватывает все процессы в задании, поэтому этот вызов должен возвращать количество процессов, которые были запрошены для задания.

```
MPI_Comm_rank(    MPI_Comm communicator,    int* rank)
```

MPI_Comm_rank возвращает ранг процесса в коммуникаторе. Каждому процессу внутри коммуникатора присваивается дополнительный ранг, начиная с нуля. Ранги процессов в основном используются для идентификации при отправке и получении сообщений.

```
MPI_Get_processor_name(    char* name,    int* name_length)
```

MPI_Get_processor_name получает фактическое имя процессора, на котором выполняется процесс.

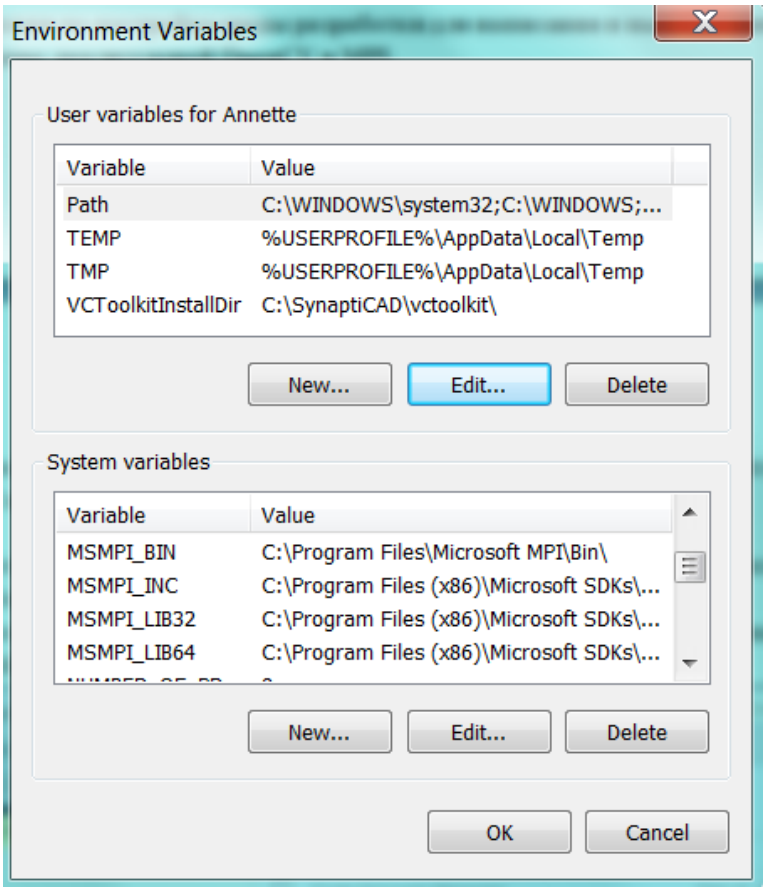
```
MPI_Finalize()
```

MPI_Finalize используется для очистки среды MPI. После этого больше нельзя делать вызовы MPI.

Инструкции по настройке среды разработки и выполнения программы, реализующей OpenCV и MPI

После распаковки библиотек MS-MPI, необходимо провести настройку проекта.

Для MPI все переменные записываются в Переменные среды автоматически.



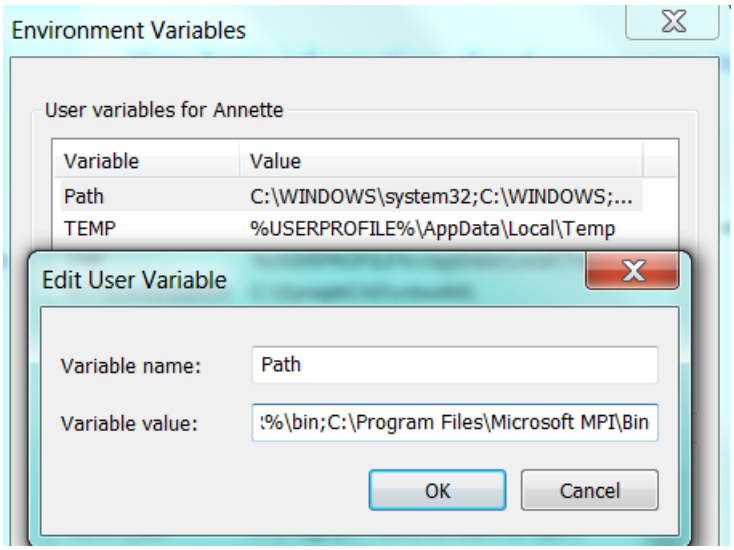
Если этого не произошло, можно добавить переменные самостоятельно.

MSMPI_BIN – путь к папке bin.

MSMPI_INC – папка include.

MSMPI_LIB32/MSMPI_LIB64 – папки Lib\x86\ и Lib\x64\.

Также нужно добавить в Путь C:\Program Files\Microsoft MPI\Bin.



Для библиотеки OpenCV также нужно добавить путь и переменную.

Path: %OPENCV_DIR%\bin

System variable: Name: OPENCV_DIR

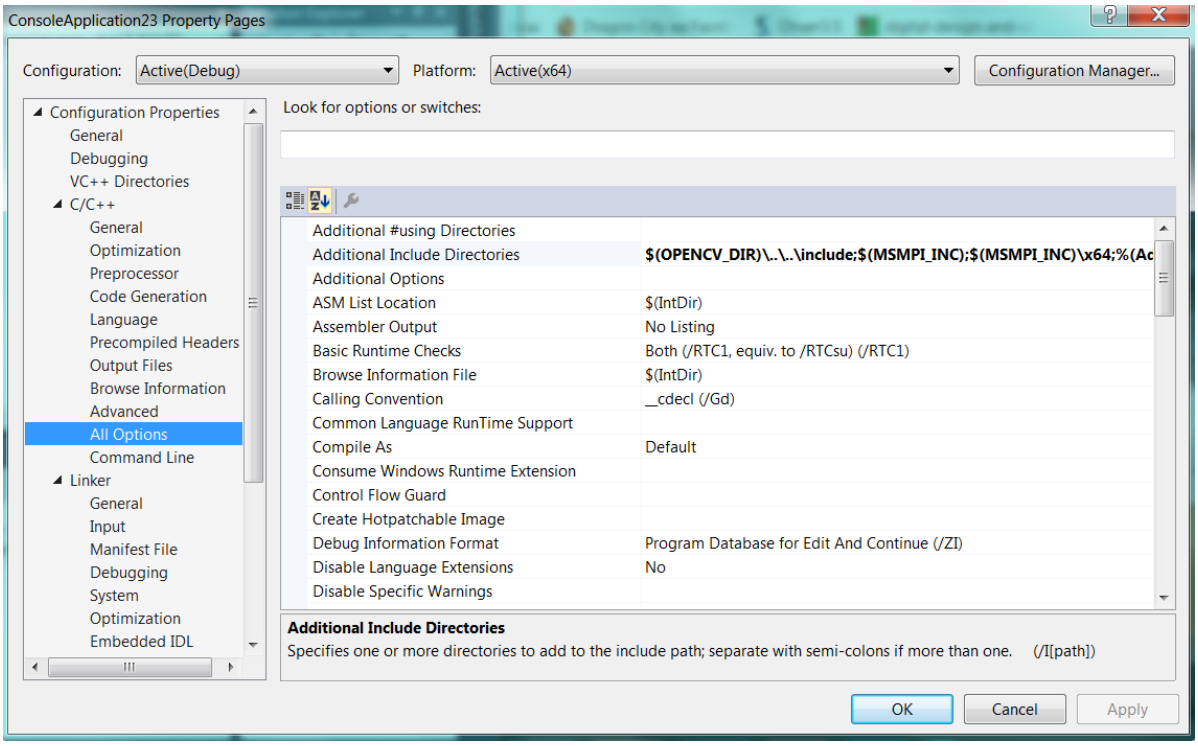
Value: C:\opencv\build\x64\vc14

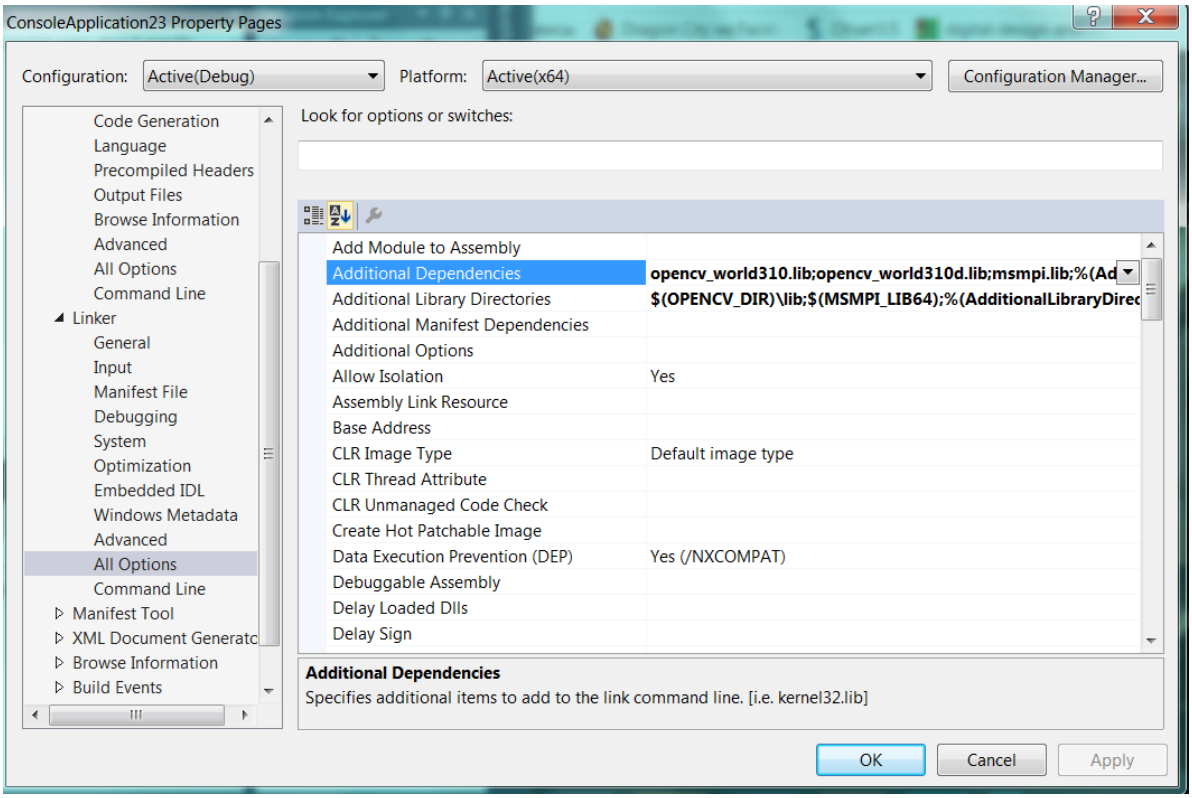
Для работы приложения нужно добавить путь к переменным в Свойствах проекта в среде разработки Visual Studio.

В C/C++ -> All options -> Additional Include Directories добавляем $(OPENCV_DIR)\..\..\include;$(MSMPI_INC);$(MSMPI_INC)\x64;

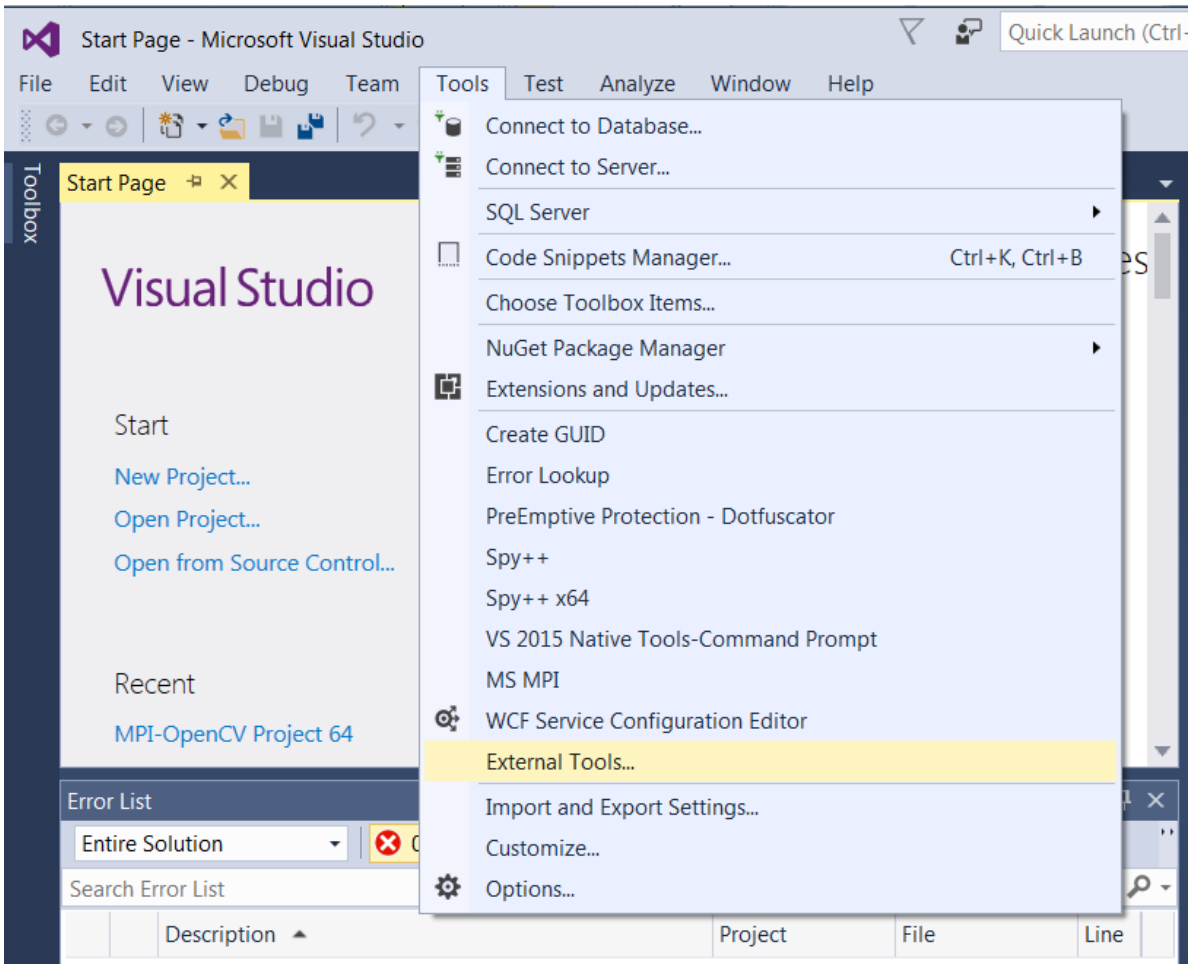
В Linker -> All options -> Additional Dependencies добавляем opencv_world310.lib;opencv_world310d.lib;msmpi.lib;

В Linker -> All options -> Additional Library Directories - $(OPENCV_DIR)\lib;$(MSMPI_LIB64);





Для выполнения программы нам нужен доступ к командной строке. Для вызова cmd из Visual Studio добавим наш собственный инструмент в разделе External Tools.



Для этого достаточно нажать в окне Extrnal Tools кнопку Add и заполнить следующие поля.

Title: MS MPI

Command: $(MSMPI_BIN)mpiexec.exe

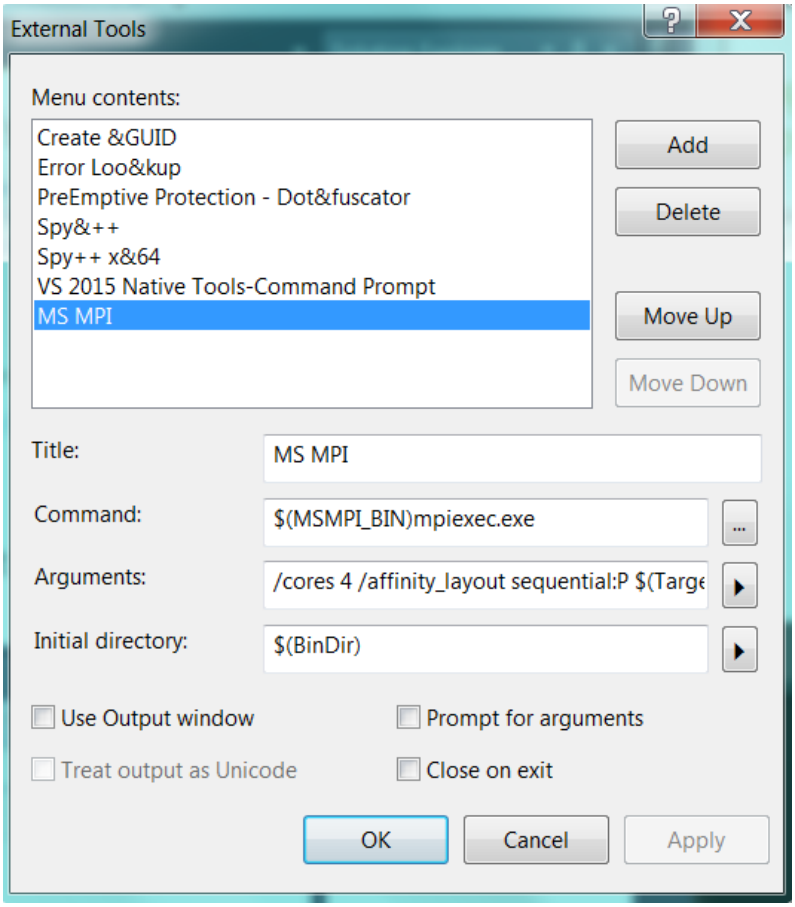
Arguments: /cores 4 /affinity_layout sequential:P $(TargetName). Exe

Initial Directory: $(BinDir)

Для выполнения приложения с MPI на нескольких ядрах, необходимо запускать программу командой mpiexec.exe. При запуске стандартными инструментами VS, будет создаваться всего один процесс.

Перед именем приложения мы также указали ключи /cores n (для явного указания количества физических ядер) и affinity_layout sequential:P (последовательная привязка процессов к физическим ядрам).

Также важно снять флажок с окошка Close on exit.



Теперь для запуска приложения достаточно будет выбрать наш инструмент MS MPI.

Приложение 1. https://mpitutorial.com/tutorials/mpi-hello-world/

```
#include <mpi.h>
#include <stdio.h>

int main(int argc, char** argv) {
    // Initialize the MPI environment
    MPI_Init(NULL, NULL);

    // Get the number of processes
    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);

    // Get the rank of the process
    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

    // Get the name of the processor
    char processor_name[MPI_MAX_PROCESSOR_NAME];
    int name_len;
```

```
    MPI_Get_processor_name(processor_name, &name_len);

    // Print off a hello world message
    printf("Hello world from processor %s, rank %d out of %d
processors\n",
           processor_name, world_rank, world_size);

    // Finalize the MPI environment.
    MPI_Finalize();
}
```

Приложение 2. Простая MPI программа

```
#include<mpi.h>
#include<iostream>
using namespace std;
int main(int argc, char **argv)
{int rank, size;
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
cout << "The number of processes: " << size << " my number is " << rank << endl;
MPI_Finalize();
return 0;
}
```

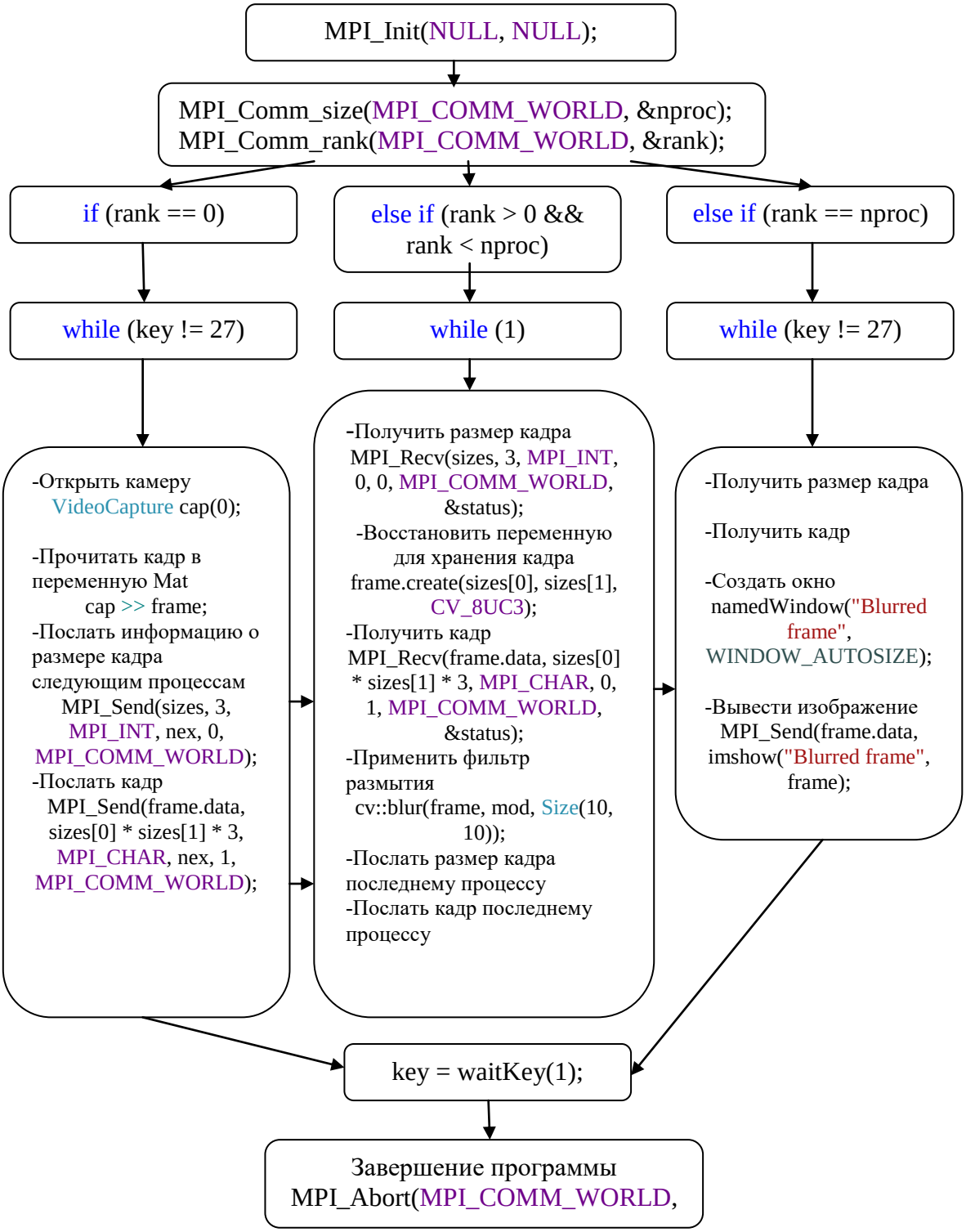
Приложение 3. Программа с функциями MPI и OpenCV

```
#include "mpi.h"
#include <opencv2/opencv.hpp>
#include <ctime>

using namespace cv;
using namespace std;

int main(int argc, char* argv[]) {
      int rank, nproc;
      int nex = 0, key = 0;
      int j = 0;
      Mat frame, mod;
      double start, end;
      double time_diff=0;
      MPI_Status status;
      int number=0;

      int sizes[3];

      int rc = MPI_Init(NULL, NULL);

      if (rc != MPI_SUCCESS) {
            printf("Error \n");
            MPI_Abort(MPI_COMM_WORLD, rc);
      }
```

```
	MPI_Comm_size(MPI_COMM_WORLD, &nproc);
	MPI_Comm_rank(MPI_COMM_WORLD, &rank);
	nproc -= 1;

	if (rank == 0) {

		VideoCapture cap(0);
		//VideoCapture cap("C:\\Users\\Public\\Videos\\Sample
Videos\\Wildlife.wmv");
		if (!cap.isOpened())
		{
			std::cout << "Cannot open the video file. \n";
			return -1;
		}

		while (key != 27) {
			//std::cout << "Process 0 reading frame" << endl;

			cap >> frame;

			if (!cap.read(frame))
			{
				std::cout << "\n Cannot read the video file. \n";
				break;
			}
			namedWindow("Test frame", CV_WINDOW_AUTOSIZE);
			imshow("Test frame", frame);

			std::cout << "Frame size is " << sizeof(frame) << endl;
			nex++;
			if (nex == nproc) nex = 1;

			if (j == 0) {
				sizes[2] = frame.elemSize();
				Size s = frame.size();
				sizes[0] = s.height;
				sizes[1] = s.width;
				MPI_Send(sizes, 3, MPI_INT, nex, 0,
MPI_COMM_WORLD);
			}

			start = clock();
			MPI_Send(&start, 1, MPI_DOUBLE, nex, 1, MPI_COMM_WORLD);
			MPI_Send(frame.data, sizes[0] * sizes[1] * 3, MPI_CHAR,
nex, 1, MPI_COMM_WORLD);

			key = waitKey(1);
			if (nex > (nproc - 2)) j++;
		}
		//for (int j = 1; j < nproc; j++) {
			//MPI_Isend(&j, 1, MPI_INT, j, 11, MPI_COMM_WORLD);

		//}
		MPI_Abort(MPI_COMM_WORLD, rc);
	}
	else if (rank > 0 && rank < nproc) {
		while (1) {
			if (j == 0) {
```

```
                        MPI_Recv(sizes, 3, MPI_INT, 0, 0, MPI_COMM_WORLD,
&status);
                        frame.create(sizes[0], sizes[1], CV_8UC3);
                  }

                  MPI_Recv(&start, 1, MPI_DOUBLE, 0, 1, MPI_COMM_WORLD,
&status);
                  MPI_Recv(frame.data, sizes[0] * sizes[1] * 3, MPI_CHAR,
0, 1, MPI_COMM_WORLD, &status);

                  end = clock();
                  time_diff = (end-start) / (double)CLOCKS_PER_SEC;;
                  std::cout << " 0 send time " << time_diff << endl;

                  cv::blur(frame, mod, Size(10, 10));

                  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
                  //std::cout << "Process " << rank << "blurring frame" <<
endl;
                  if (j == 0) {
                        MPI_Send(sizes, 3, MPI_INT, nproc, 0,
MPI_COMM_WORLD);
                  }

                  MPI_Send(mod.data, sizes[0] * sizes[1] * 3, MPI_CHAR,
nproc, 1, MPI_COMM_WORLD);
                  j++;
            }
            }
      else if (rank == nproc) {
            while (key != 27) {
                  nex++;
                  if (nex == nproc) nex = 1;
                  if (j == 0) {
                        MPI_Recv(sizes, 3, MPI_INT, nex, 0, MPI_COMM_WORLD,
&status);
                        frame.create(sizes[0], sizes[1], CV_8UC3);
                  }

                  MPI_Recv(frame.data, sizes[0] * sizes[1] * 3, MPI_CHAR,
nex, 1, MPI_COMM_WORLD, &status);
                  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
                  //std::cout << "Process " << rank << "showing frame" <<
endl;
                  namedWindow("Blurred frame", WINDOW_AUTOSIZE);
                  imshow("Blurred frame", frame);
                  key = waitKey(1);
                  j++;
            }
            MPI_Abort(MPI_COMM_WORLD, rc);
      }
      destroyAllWindows();
      MPI_Finalize();
      return 0;
}
```

Приложение 4. Алгоритм работы программы приложения 3



Темы итоговых индивидуальных заданий

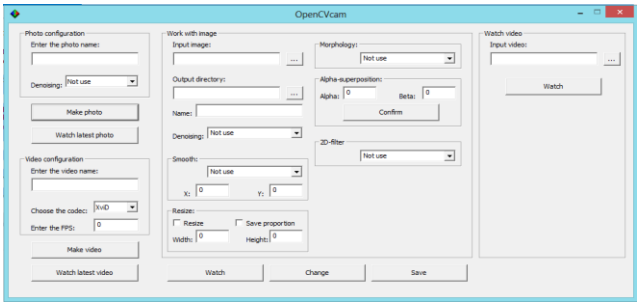
1. Программа превращения USB-камеры в камеру для задач измерений. Используется графический интерфейс, библиотека DirectShow. Выводятся на экран указанные строка/столбец, указанная часть видео кадров, записывается потоковое видео (avi-файл) и статические/динамические(с указанным интервалом) строка/столбец в текстовом файле.

2. Многофункциональный проигрыватель видео файлов с графическим интерфейсом и с использованием библиотеки DirectShow (со звуком).

3. Захват, воспроизведение и запись видео (используется камера) с функциями zoom и subarray . Библиотека DirectShow.

4. Программа превращения USB-камеры в камеру для задач измерений. Используется графический интерфейс, библиотека OpenCV.

5. Многофункциональный интерфейс с функциями OpenCV.



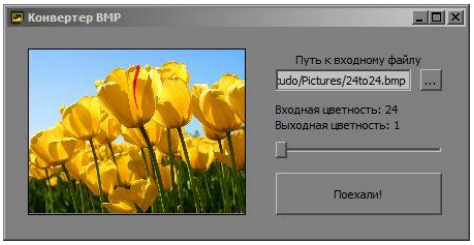
6. Проигрыватель FMOD с графическим интерфейсом. Программа воспроизводит mp3 файл и дает возможности: поставить воспроизведение на паузу, возобновить, и изменить громкость звука.

7 3D проигрыватель в FMOD с графическим интерфейсом. Программа позволяет не только воспроизводит mp3 и др. файл в 3D-

режиме, но и задать параметры 3D-режима, и выбрать функции (эхо, реверберация…).

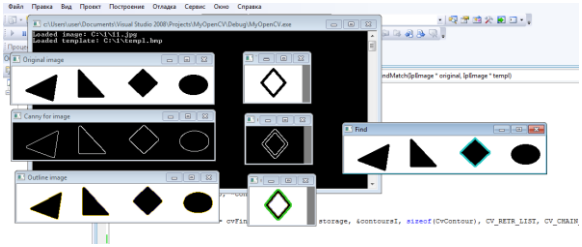
8. Многофункциональный проигрыватель DirectSound с графическим интерфейсом.

9. Конвертор BMP в BMP с графическим интерфейсом. Программа по выбору конвертирует изображения из 24 бит на пиксель в 1, 4, 8, 16, 24 бит на пиксель.



10. Конвертор BMP в другие форматы. Данная программа с графическим интерфейсом конвертирует изображения в различные форматы (такие как JPEG, BMP, PNG и др.), а так же позволяет просмотреть записанные изображения.

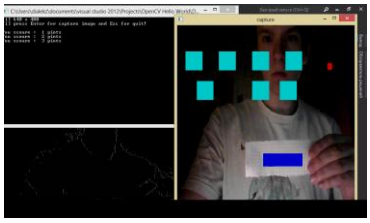
11. OpenCV Поиск образца в изображении путем сравнения контуров.



12. Фильтры в OpenCV. Программа имеет графический интерфейс, который позволяет: запустить поток видео с веб-камеры, и применять различные фильтры к этому потоку в реальном времени.

13. Игра в OpenCV. Работа осуществляется с веб-камерой. Формируется мячик и платформа. Платформой управляем через веб-камеру. При контакте с границами игрового поля или с платформой, управляемой игроком, мячик меняет траекторию передвижения и отскакивает, в зависимости от места столкновения. При столкновении с блоками он так же отскакивает, но при этом блок уничтожается, а игрок получает очко. Задача игрока набрать как можно большее

количество очков за наименьшее время.

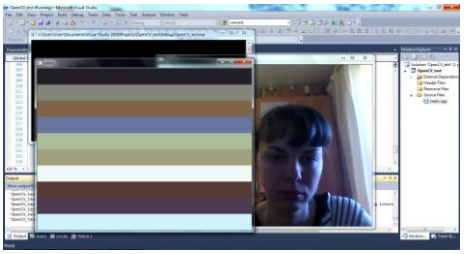


14. Игра в OpenCV. Ловим мячики.

15. Рисование в OpenCV. Считывание жестов пользователя с веб-камеры и рисование соответствующих объектов в другом рабочем окне. Необходима возможность редактирования кисти, с помощью которой осуществляется рисование.

16. Сравнение в OpenCV. Сравнение изображений и генерация картинки отличий.

17. Зум в OpenCV. Программа записывает потоковое видео в avi-файл. Отображает, используя зум и другие возможности...



18. Мультимедиа плеер с графическим интерфейсом в DirectShow. Программа проигрывает файл с расширением .avi. Пользователь может выбрать файл для воспроизведения и остановить или продолжить показ видео. При воспроизведении накладывается дополнительный эффект.

19. Запись движения в OpenCV. Программа считывает и отображает данные с камеры. Если программа находит отличия между текущим и предыдущим кадром, то она пишет видео в avi файл.

20. Применение векторного программирования (SIMD) для ускорения работы с изображением (поиск в потоке тестового изображения).

21. Преобразователь картинки в графическом процессоре CUDA с вычислением времени преобразования и изменением параметров параллельного вычисления.

22. MPI. Захват и расширенная обработка видео с вычислением времени преобразования в последовательном и параллельном режимах.

Авторы благодарны студентам, которые предоставили скриншоты программ в качестве иллюстраций к заданиям.

Библиографический список

1. Петров М.Н. Компьютерная графика. Учебник для вузов. 3-е изд. 2011, 544 с. ISBN 978-5-459-00809-8.

2. Антонов А.С. Параллельное программирование с использованием технологии OpenMP: Учебное пособие.-М.: Изд-во МГУ, 2009. – 77 с. ISBN 978-5-211-05702-9

3. Mark D. Pesce Programming Microsoft Direct Show for Digital Video/Television Microsoft Press. 2003. ISBN: 0735618216

4. Горнаков С.Г. DirectX 9. Уроки программирования на C++ БХВ-Петербург, 2005.- 400 с. ISBN: 5-94157-482-7

5. Боресков А.В., Харламов А.А. Основы работы с технологией CUDA. ДМК Пресс, 2010.- 234с.

6. Арнольд Роббинс. Unix. Справочник / Пер. с англ.-М.: КУДИЦ-ПРЕСС, 2007.-864 с.

7. Молодяков С.А. Проектирование специализированных цифровых видеокамер. / СПб. : Изд-во Политехн. ун-та, 2016. 286 с.

Электронные ресурсы

1. Учебник по ADOBE PHOTOSHOP 6.0. http://e-dok.narod.ru/adobephotoshop/book/index.html

2. http://msdn.microsoft.com

3. http://www.nersc.gov/nusers/help/tutorials/openmp/

4. http://scv.bu.edu/tutorials/OpenMP/

5. Начало знакомства с CUDA http://ru.wikipedia.org/wiki/CUDA

6. Григорьев А.В. Еремеев И.С., Алексеева М.И. Параллельное программирование с использованием технологии CUDA. edu.chpc.ru/

7. Документация nvidia. http://docs.nvidia.com/cuda/index.html

8. Программирование звука в DirectSound http://www.helloworld. ru/ texts/comp/games/dsound/dsound/index.htm

9. Запись звука с использованием API FMOD http://www.tiflocomp.ru/games/design/sound_games/fmod_rec.php

10. Основной сайт FMOD www.FMOD.org.

Молодяков Сергей Александрович,
Петров Александр Владимирович

АРХИТЕКТУРА ЭВМ ПРОГРАММИРОВАНИЕ ПЕРИФЕРИЙНЫХ УСТРОЙСТВ

Учебное пособие

Лицензия ЛР № 020593 от 07.08.97

Налоговая льгота – Общероссийский классификатор продукции
ОК 005-93, т. 2; 95 3005 – учебная литература

Подписано в печать _____2013. Формат 60×84/16. Печать цифровая
Усл. печ. л. ___. Уч.-изд. л. _____. Тираж . Заказ

Отпечатано с готового оригинал-макета, предоставленного авторами
в цифровом типографском центре Издательства Политехнического университета:
195251, Санкт-Петербург, Политехническая ул., 29.
Тел. (812) 540-40-14
Тел./факс: (812) 927-57-76