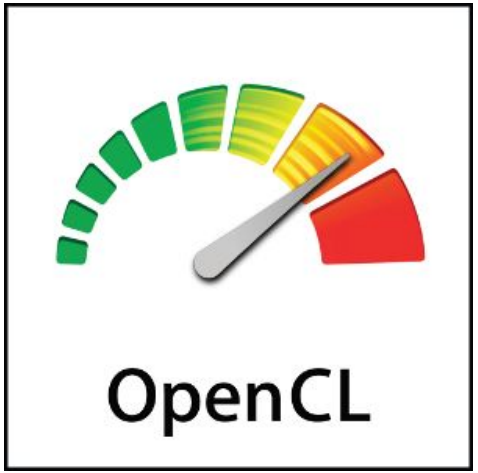
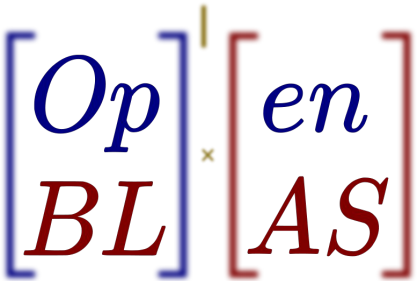
СПбГЭТУ «ЛЭТИ»
ПЕРВЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ

Фреймворки и библиотеки: OpenCV, OpenCL, OpenBLAS и др.

Содержание занятия

1. Введение.
2. OpenCV
3. OpenCL.
4. OpenBLAS.
5. Полезные ссылки.





Введение

Задачи, решаемые OpenCV, OpenCL, OpenBLAS:

- Высокая производительность за счёт реализации оптимизированных алгоритмов, использования распараллеливания и SIMD-инструкций.
- Кроссплатформенность, что позволяет разрабатывать решения для разных операционных систем и аппаратных архитектур, следуя единым стандартам API.
- Широкую область применения: от автоматизации в медицине (анализ снимков) и промышленного машинного зрения до робототехники (отслеживание объектов) и научного моделирования.

OpenCV

OpenCV (Open Source Computer Vision Library) — это открытая библиотека для задач компьютерного зрения и обработки изображений. Её широко применяют в таких областях, как робототехника, системы видеонаблюдения, автоматическая диагностика и автоматизация промышленных процессов.

OpenCV

Ключевые задачи, решаемые с помощью OpenCV:

- Обнаружение и распознавание объектов
- Обработка и улучшение изображений
- Трекинг объектов в видео
- Калибровка камер и 3D-реконструкция

Установка OpenCV (Windows)

- **Открыть консоль MSYS2 (через которую уже устанавливался C++ на занятии по IDE) и ввести команду:**

  pacman -S mingw-w64-ucrt-x86_64-opencv

- **После чего необходимо установить еще и базовый qt6 (для отображение окон через imshow):**

  pacman -S mingw-w64-ucrt-x86_64-qt6-base

Установка OpenCV (Windows)

- **Для подсветки синтаксиса** необходимо в .vscode/c_cpp_properties.json добавить путь до файлов h библиотеки. То есть добавить в "includePath" путь "C:/msys64/ucrt64/include/opencv4" (если msys2 был установлен по пути C:/msys64).
- **Пример файла:**

```
{
  "configurations": [
    {
      "name": "Win32",
      "includePath": [
        "${default}",
        "C:/msys64/ucrt64/include/opencv4"
      ],
      "defines": ["_DEBUG", "UNICODE", "_UNICODE"],
      "windowsSdkVersion": "10.0.26100.0",
      "cStandard": "c17",
      "cppStandard": "c++17",
      "intelliSenseMode": "windows-gcc-x64",
      "compilerPath": "C:/msys64/ucrt64/bin/g++.exe"
    }
  ],
  "version": 4
}
```

Установка OpenCV (Windows)

- **Для сборки OpenCV** исходников необходимо добавить в конец списка аргументов "args" tasks.json следующие аргументы "-IC:/msys64/ucrt64/include/opencv4", "-lopencv_core", "-lopencv_imgproc", "-lopencv_highgui", "-lopencv_imgcodecs", "-lopencv_videoio"(путь до файла с исходниками opencv и необходимые для работы библиотеки).
- **Пример части файла:**

```
"args": [
  "-fdiagnostics-color=always",
  "-g",
  "${file}",
  "-o",
  "${fileDirname}\\${fileBasenameNoExtension}.exe",
  "-IC:/msys64/ucrt64/include/opencv4",
  "-lopencv_core",
  "-lopencv_imgproc",
  "-lopencv_highgui",
  "-lopencv_imgcodecs",
  "-lopencv_videoio"
],
```

Установка OpenCV (Linux)

- **Открыть консоль и ввести команду**

  sudo apt-get install -y libopencv-dev

Установка OpenCV (Linux)

- **Для подсветки синтаксиса** необходимо в .vscode/c_cpp_properties.json добавить путь до файлов h библиотеки. То есть добавить в "includePath" путь "/usr/include/opencv4".
- **Пример файла:**

```
{
    "configurations": [
        {
            "name": "Linux",
            "includePath": [
                "${workspaceFolder}/**",
                "/usr/include/opencv4"
            ],
            "defines": [],
            "compilerPath": "/usr/bin/gcc",
            "cStandard": "c17",
            "cppStandard": "gnu++17",
            "intelliSenseMode": "linux-gcc-x64",
            "compilerArgs": [
                "/usr/include/opencv4"
            ]
        }
    ],
    "version": 4
}
```

Установка OpenCV (Linux)

- **Для сборки OpenCV** исходников необходимо добавить в конец списка аргументов "args" tasks.json следующие аргументы "`pkg-config", "--cflags", "--libs", "opencv4`" (Они нужны для автоматического подбора параметров для библиотеки через pkg-config).
- **Пример части файла:**

```
"args": [
        "-fdiagnostics-color=always",
        "-g",
        "${file}",
        "-o",
     "${fileDirname}/${fileBasenameNoExtension}",
        "`pkg-config", "--cflags", "--libs", "opencv4`",
    ],
```

Установка OpenCV (Linux, распространенная ошибка)

- Ошибка при работе со snap:

  “symbol lookup error: /snap/core20/current/lib/x86_64-linux-gnu/libpthread.so.0: undefined symbol…”

- Необходимо с помощью Ctrl+Shift+P открыть панель ввода команд VS Code и там ввести и выбрать Preferences: Open User Settings (JSON). После чего откроется settings.json, куда в конец необходимо добавить:

```
"terminal.integrated.env.linux": {
        "GTK_PATH": null,
        "GIO_MODULE_DIR": null,
    },
```

- После этих действий перезапустите VS Code.

Базовые классы OpenCV

- Mat (Matrix) — основной класс OpenCV для работы с изображениями и матрицами. Он обеспечивает хранение и обработку многомерных числовых массивов.
- Основные характеристики:
  - Данные: хранит пиксели изображений или числовые массивы.
  - Гибкость: поддерживает различные глубины цветов и количество каналов.
  - Удобство: предоставляет интерфейс для работы с изображениями как с двумерными массивами.

Базовые классы OpenCV

- Создание объектов Mat:

```
// Пустая матрица указанного размера
Mat zero(480, 640, CV_8UC3); // 640×480, 3 канала, 8 бит на канал

// Из файла изображения
Mat img = imread("image.jpg");

// С предопределенными значениями
Mat grayMat = Mat::zeros(100, 100, CV_8UC1); // Черная матрица 100×100
Mat whiteMat = Mat::ones(50, 50, CV_32FC1);  // Белая матрица 50×50

// Заполнение существующей матрицы
img.setTo(Scalar(0, 255, 0)); // Заливка зелёным цветом
```

Базовые классы OpenCV

- Структура Mat, основные поля:
  - data: указатель на область памяти с данными пикселей.
  - rows, cols: количество строк и столбцов (для 2D-матриц).
  - dims: количество измерений (2 для обычных изображений).
  - type(): тип данных (глубина и количество каналов), например CV_8UC3
  - step: размер шага между строками в байтах.

- Управление памятью:
  - Mat использует систему подсчёта ссылок. Память автоматически освобождается, когда все ссылки на объект уничтожаются. Для создания полной копии данных используйте метод .clone().

Базовые классы OpenCV

- Point: представление 2D-точки (x, y).
- Size: размеры объекта (width, height).
- Rect: прямоугольная область (x, y, width, height).
- Scalar: 4-х элементный вектор для хранения значений.
- Vec: шаблонный класс для малых векторов.
- Matx: шаблонный класс для небольших матриц фиксированного размера.

Эти классы интегрированы с Mat и обеспечивают единообразную работу с геометрическими примитивами и данными в OpenCV

Базовые функции OpenCV

- Загрузка изображений

```
// Загрузка изображения
Mat img = imread("image.jpg", IMREAD_COLOR);
if (img.empty()) {
    std::cerr << "Не удалось загрузить изображение"<< std::endl;
    return -1;
}
// Отображение изображения
imshow("Original Image", img);
waitKey(0);
```

Базовые функции OpenCV

- imread()
  - Назначение: Загрузка изображения из файла.
  - Параметры:
  - const String& filename - путь к файлу изображения.
  - int flags = IMREAD_COLOR - флаг загрузки:
  - IMREAD_COLOR - цветное изображение (3 канала).
  - IMREAD_GRAYSCALE - grayscale (1 канал).
  - IMREAD_UNCHANGED - как есть (с альфа-каналом).
  - Возвращает: объект Mat с загруженным изображением.

Базовые функции OpenCV

- imshow()
  - Назначение: отображение изображения в окне.
  - Параметры:
  - const String& winname - имя окна.
  - InputArray mat - изображение для отображения.
- waitKey()
  - Назначение: ожидание нажатия клавиши.
  - Параметры:
  - int delay = 0 - время ожидания в миллисекундах (0 - бесконечно).
  - Возвращает: код нажатой клавиши

Базовые функции OpenCV

- empty()
  - Назначение: проверка матрицы на пустоту.
  - Возвращает: true если матрица пустая.

Базовые функции OpenCV

- Геометрические преобразования

```
// Изменение размера
Mat resized;
resize(img, resized, Size(640, 480), 0, 0, INTER_LINEAR);

// Поворот изображения
Point2f center(img.cols/2.0F, img.rows/2.0F);
Mat rotMat = getRotationMatrix2D(center, 45, 1.0);
Mat rotated;
warpAffine(img, rotated, rotMat, img.size(), INTER_LINEAR, BORDER_CONSTANT);
```

Базовые функции OpenCV

- resize()
  - Назначение: изменение размеров изображения.
  - Параметры:
  - InputArray src - исходное изображение.
  - OutputArray dst - результат.
  - Size dsize - новый размер.
  - double fx = 0 - коэффициент масштабирования по X.
  - double fy = 0 - коэффициент масштабирования по Y.
  - int interpolation = INTER_LINEAR - метод интерполяции:
    - INTER_NEAREST - ближайший сосед.
    - INTER_LINEAR - билинейная.
    - INTER_CUBIC - бикубическая.

Базовые функции OpenCV

- getRotationMatrix2D()
  - Назначение: создание матрицы поворота.
  - Параметры:
    - Point2f center - центр поворота.
    - double angle - угол поворота в градусах.
    - double scale - коэффициент масштабирования.
  - Возвращает: Матрицу 2×3 аффинного преобразования.

Базовые функции OpenCV

- warpAffine()
  - Назначение: применение аффинного преобразования.
  - Параметры:
    - InputArray src - исходное изображение.
    - OutputArray dst - результат.
    - InputArray M - матрица преобразования 2×3.
    - Size dsize - размер выходного изображения.
    - int flags = INTER_LINEAR - метод интерполяции.
    - int borderMode = BORDER_CONSTANT - метод заполнения границ.
    - const Scalar& borderValue = Scalar() - цвет границ.

Базовые функции OpenCV

- Фильтрация и улучшение изображений

```
// Размытие по Гауссу
Mat blurred;
GaussianBlur(img, blurred, Size(5,5), 1.5, 1.5);

// Медианный фильтр
Mat medianFiltered;
medianBlur(img, medianFiltered, 5);

// Двусторонняя фильтрация
Mat bilateralFiltered;
bilateralFilter(img, bilateralFiltered, 9, 75, 75);
```

Базовые функции OpenCV

- GaussianBlur()
  - Назначение: гауссово размытие для уменьшения шума.
  - Параметры:
    - InputArray src - исходное изображение.
    - OutputArray dst - результат.
    - Size ksize - размер ядра (должен быть нечётным).
    - double sigmaX - стандартное отклонение по X.
    - double sigmaY = 0 - стандартное отклонение по Y.
    - int borderType = BORDER_DEFAULT - тип границ.

Базовые функции OpenCV

- medianBlur()
  - Назначение: медианная фильтрация (эффективна против импульсного шума).
  - Параметры:
    - InputArray src - исходное изображение.
    - OutputArray dst - результат.
    - int ksize - размер апертуры (должен быть нечётным > 1).

Базовые функции OpenCV

- bilateralFilter()
  - Назначение: сохранение границ при сглаживании.
  - Параметры:
    - InputArray src - исходное изображение.
    - OutputArray dst - результат.
    - int d - диаметр окрестности пикселя.
    - double sigmaColor - фильтрация в цветовом пространстве.
    - double sigmaSpace - фильтрация в координатном пространстве.

Базовые функции OpenCV

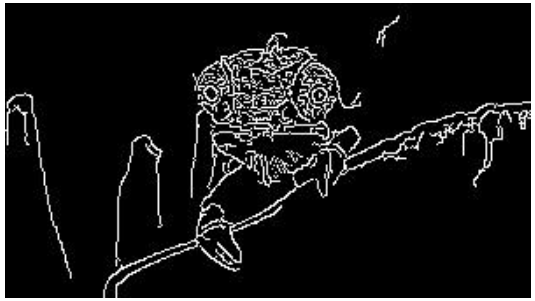
- Анализ изображений

```
// Преобразование в градации серого
Mat gray;
cvtColor(img, gray, COLOR_BGR2GRAY);

// Выделение границ
Mat edges;
Canny(gray, edges, 50, 150, 3);

// Поиск и отрисовка контуров
std::vector<std::vector<Point>> contours;
std::vector<Vec4i> hierarchy;
findContours(edges, contours, hierarchy, RETR_TREE, CHAIN_APPROX_SIMPLE);

// Визуализация контуров
for (size_t i = 0; i < contours.size(); i++) {
    drawContours(img, contours, (int)i, Scalar(0, 255, 0), 1);
}
```



Базовые функции OpenCV

- cvtColor()
  - Назначение: преобразование цветового пространства
  - Параметры:
    - InputArray src - исходное изображение.
    - OutputArray dst - результат.
    - int code - код преобразования:
      - COLOR_BGR2GRAY - BGR → Grayscale
      - COLOR_BGR2HSV - BGR → HSV
      - COLOR_BGR2RGB - BGR → RGB

Базовые функции OpenCV

- Canny()
  - Назначение: детектор границ Канни.
  - Параметры:
    - InputArray image - исходное изображение.
    - OutputArray edges - карта границ.
    - double threshold1 - нижний порог гистерезиса.
    - double threshold2 - верхний порог гистерезиса.
    - int apertureSize = 3 - размер апертуры Собеля.
    - bool L2gradient = false - использование L2-нормы.

Базовые функции OpenCV

- findContours()
  - Назначение: поиск контуров в бинарном изображении.
  - Параметры:
    - InputArray image - бинарное изображение.
    - OutputArrayOfArrays contours - найденные контуры.
    - OutputArray hierarchy - иерархия контуров.
    - int mode - метод извлечения:
      - RETR_EXTERNAL - только внешние контуры.
      - RETR_LIST - все контуры без иерархии.
      - RETR_TREE - все контуры с полной иерархией.
      - int method - метод аппроксимации:
      - CHAIN_APPROX_NONE - все точки.
      - CHAIN_APPROX_SIMPLE - сжатие контуров.

Базовые функции OpenCV

- drawContours()
  - Назначение: визуализация найденных контуров.
  - Параметры:
    - InputOutputArray image - изображение для рисования.
    - InputArrayOfArrays contours - контуры для отрисовки.
    - int contourIdx - индекс контура (-1 для всех).
    - const Scalar& color - цвет.
    - int thickness = 1 - толщина линии.
    - int lineType = LINE_8 - тип линии.

Базовые функции OpenCV

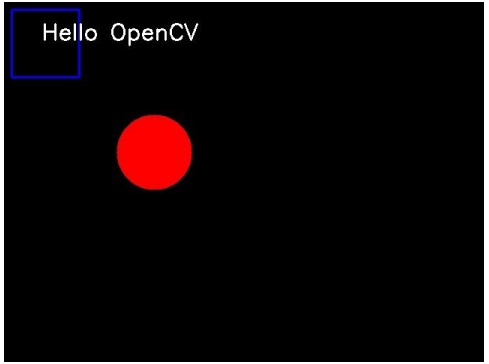
- Некоторые функции записи в изображения

```
// Сохранение изображения
imwrite("output.jpg", img, {IMWRITE_JPEG_QUALITY, 95});

// Создание пустого изображения
Mat blank = Mat::zeros(480, 640, CV_8UC3);

// Рисование примитивов
rectangle(blank, Point(10, 10), Point(100, 100), Scalar(255, 0, 0), 2);
circle(blank, Point(200, 200), 50, Scalar(0, 0, 255), -1);

// Текст на изображении
putText(blank, "Hello OpenCV", Point(50, 50),
        FONT_HERSHEY_SIMPLEX, 1, Scalar(255, 255, 255), 2);
```



Базовые функции OpenCV

- imwrite()
  - Назначение: сохранение изображения в файл.
  - Параметры:
    - const String& filename - имя файла.
    - InputArray img - изображение для сохранения.
    - const std::vector<int>& params = {} - параметры сжатия.

Базовые функции OpenCV

- rectangle()
  - Назначение: рисование прямоугольника.
  - Параметры:
    - InputOutputArray img - изображение.
    - Point pt1 - левый верхний угол.
    - Point pt2 - правый нижний угол.
    - const Scalar& color - цвет.
    - int thickness = 1 - толщина линии (-1 для заливки).
    - int lineType = LINE_8 - тип линии.

Базовые функции OpenCV

- putText()
  - Назначение: добавление текста на изображение.
  - Параметры:
    - InputOutputArray img - изображение.
    - const String& text - текст.
    - Point org - позиция нижнего левого угла текста.
    - int fontFace - шрифт (FONT_HERSHEY_...).
    - double fontScale - масштаб шрифта.
    - Scalar color - цвет текста.
    - int thickness = 1 - толщина.
    - int lineType = LINE_8 - тип линии.

Базовые функции OpenCV

- Работа с масками
- Маска — это двоичный (черно-белый) изображение, которое служит шаблоном или фильтром, выделяющим интересующую область исходного изображения. Маска нужна для ограничение области обработки и выделению конкретных участков для фильтрации, поиска, сегментации и тд.
- В маске пиксели бывают только двух типов:
  - Белые (255) — означают, что соответствующий участок исходного изображения обрабатывается или выбирается.
  - Черные (0) — участок игнорируется.

Базовые функции OpenCV

```
// Создание чёрной маски
Mat mask_example = Mat::zeros(480, 640, CV_8UC1);

// Создание маски по условию (выделение ярких областей)
Mat img = imread("image.jpg");
Mat gray;
cvtColor(img, gray, COLOR_BGR2GRAY);

Mat mask;
threshold(gray, mask, 100, 255, THRESH_BINARY);
// mask содержит 255 где яркость > 100, и 0 — иначе

// Применение маски к изображению
Mat maskedImage;
img.copyTo(maskedImage, mask);
```

Базовые функции OpenCV

- threshold()
  - Назначение: бинаризация изображения для создания маски.
  - Параметры:
    - InputArray src - исходное изображение.
    - OutputArray dst - результат бинаризации.
    - double thresh - пороговое значение.
    - double maxval - значение для пикселей выше порога.
    - int type - тип бинаризации:
      - THRESH_BINARY - бинарная.
      - THRESH_BINARY_INV - инвертированная бинарная.

Базовые функции OpenCV

- copyTo()
  - Назначение: копирование изображения с применением маски.
  - Параметры:
    - OutputArray dst - целевое изображение.
    - InputArray mask - маска операции.

Базовые функции OpenCV

- Работа с видео и камерами

```
// Захват видео с веб-камеры
VideoCapture cap(0);
if (!cap.isOpened()) {
    std::cerr << "Ошибка подключения к камере"<< std::endl;
    return -1;
}

Mat frame;
while (true) {
    cap >> frame;
    if (frame.empty()) break;

    // Обработка кадра
    imshow("Webcam", frame);
    if (waitKey(30) >= 0) break;
}

// Работа с видеофайлом
VideoCapture video("video.mp4");
VideoWriter writer("output.avi",
                   VideoWriter::fourcc('M','J','P','G'),
                   30, Size(640, 480));
```

Базовые функции OpenCV

- VideoCapture()
  - Назначение: создание объекта для захвата видео.
  - Параметры:
    - int index - индекс камеры (0 - первая камера).
    - const String& filename - путь к видеофайлу.
- isOpened()
  - Назначение: проверка успешного подключения к источнику видео.
  - Возвращает: true если источник доступен.

Базовые функции OpenCV

- VideoWriter()
  - Назначение: создание объекта для записи видео.
  - Параметры:
    - const String& filename - имя файла для записи.
    - int fourcc - кодек для сжатия.
    - double fps - количество кадров в секунду.
    - Size frameSize - размер кадра.

Базовые функции OpenCV

- fourcc()
  - Назначение: создание FourCC кода для кодеков.
  - Параметры:
    - char c1, c2, c3, c4 - символы кодека.
  - Примеры:
    - 'M','J','P','G' - Motion-JPEG
    - 'X','2','6','4' - H.264

Мастер-класс

Базовые операции в OpenCV

```
#include <opencv2/opencv.hpp>
#include <iostream>

using namespace cv;
using namespace std;
```

Базовые операции в OpenCV

```
int main()
{
    // 1. Загрузка изображения
    Mat image = imread("image.jpg");
    if (image.empty())
    {
        cout << "Image load failed." << endl;
        return -1;
    }

    cout << "=== BASIC IMAGE PROCESSING ===" << endl;
    cout << "Size: " << image.cols << "x" << image.rows << endl;
    cout << "Channels: " << image.channels() << endl;

    imshow("1. Original Image", image);
    waitKey(0);
```

Базовые операции в OpenCV

```
// 2. Конвертация в оттенки серого
Mat gray;
cvtColor(image, gray, COLOR_BGR2GRAY);
imshow("2. Grayscale", gray);
waitKey(0);

// 3. Размытие по Гауссу
Mat blurred;
GaussianBlur(image, blurred, Size(15, 15), 0);
imshow("3. Gaussian Blur", blurred);
waitKey(0);

// 4. Детектор границ Кэнни
Mat edges;
Canny(image, edges, 50, 150);
imshow("4. Canny Edges", edges);
waitKey(0);
```

Базовые операции в OpenCV

```
// 5. Изменение размера
Mat resized;
resize(image, resized, Size(), 2, 2); // Увеличение в 2 раза
imshow("5. Resized Image", resized);
waitKey(0);

// 6. Поворот изображения
Mat rotated;
Point2f center(image.cols / 2.0, image.rows / 2.0);
Mat rotation_matrix = getRotationMatrix2D(center, 135, 1.0); // Поворот на 45°
warpAffine(image, rotated, rotation_matrix, image.size());
imshow("6. Rotated 135 degrees", rotated);
waitKey(0);

// 7. Обрезка изображения
Rect crop_region(50, 50, 200, 100); // x, y, width, height
Mat cropped = image(crop_region);
imshow("7. Cropped Image", cropped);
waitKey(0);
```

Базовые операции в OpenCV

```
    // 8. Изменение яркости и контраста
    Mat adjusted;
    image.convertTo(adjusted, -1, 1.5, 50); // alpha=1.5 (контраст), beta=50 (яркость)
    imshow("8. Brightness and Contrast", adjusted);
    waitKey(0);

    // 9. Бинаризация
    Mat binary;
    threshold(gray, binary, 128, 255, THRESH_BINARY);
    imshow("9. Binary Image", binary);
    waitKey(0);

    // 10. Сохранение результата
    imwrite("processed_image.jpg", adjusted);
    cout << "Processed image saved!" << endl;

    destroyAllWindows();

    return 0;
}
```

СПбГЭТУ «ЛЭТИ»
ПЕРВЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ

Мастер-класс

Сопоставление изображений в OpenCV

```
#include <opencv2/opencv.hpp>
#include <opencv2/features2d.hpp>
#include <iostream>

using namespace cv;
using namespace std;
```

Сопоставление изображений в OpenCV

```
// Вспомогательная функция для обрезки черных областей
Rect findNonZeroBoundary(const Mat &image) {
    Mat gray;
    cvtColor(image, gray, COLOR_BGR2GRAY);

    vector<Point> nonZeroPoints;
    findNonZero(gray, nonZeroPoints);

    if (nonZeroPoints.empty()) {
        return Rect(0, 0, image.cols, image.rows);
    }

    return boundingRect(nonZeroPoints);
}
```

Сопоставление изображений в OpenCV

```
int main() {
    cout << "=== IMAGE STITCHING WITH KEYPOINTS ==="<< endl;

    Mat img1 = imread("left.jpg");
    Mat img2 = imread("right.jpg");

    if (img1.empty() || img2.empty()) {
        cout << "Failed to load images..." << endl;
        exit(1);
    }

    if (img1.size() != img2.size()){
        resize(img2, img2, img1.size());
    }

    cout << "Image 1: " << img1.cols << "x" << img1.rows << endl;
    cout << "Image 2: " << img2.cols << "x" << img2.rows << endl;
```

Сопоставление изображений в OpenCV

```
imshow("1. Original Image 1", img1);
waitKey(0);

imshow("2. Original Image 2", img2);
waitKey(0);

// 2. Нахождение ключевых точек и дескрипторов
Ptr<ORB> orb = ORB::create(2000); // Увеличили количество features
vector<KeyPoint> keypoints1, keypoints2;
Mat descriptors1, descriptors2;

orb->detectAndCompute(img1, noArray(), keypoints1, descriptors1);
orb->detectAndCompute(img2, noArray(), keypoints2, descriptors2);

cout << "3. Keypoints found: " << endl;
cout << "   - Image 1: " << keypoints1.size() << endl;
cout << "   - Image 2: " << keypoints2.size() << endl;

// Визуализация ключевых точек
Mat img_kp1, img_kp2;
drawKeypoints(img1, keypoints1, img_kp1, Scalar(0, 255, 0), DrawMatchesFlags::DRAW_RICH_KEYPOINTS);
drawKeypoints(img2, keypoints2, img_kp2, Scalar(0, 255, 0), DrawMatchesFlags::DRAW_RICH_KEYPOINTS);
```

Сопоставление изображений в OpenCV

```
imshow("3. Keypoints 1", img_kp1);
waitKey(0);

imshow("4. Keypoints 2", img_kp2);
waitKey(0);

// 3. Сопоставление ключевых точек
Ptr<DescriptorMatcher> matcher;
if (descriptors1.type() == CV_32F){
    matcher = DescriptorMatcher::create(DescriptorMatcher::FLANNBASED);
} else {
    matcher = DescriptorMatcher::create("BruteForce-Hamming");
}

vector<vector<DMatch>> knn_matches;
vector<DMatch> good_matches;
```

Сопоставление изображений в OpenCV

```
// Используем KNN matching с фильтрацией по Lowe's ratio test
if (!descriptors1.empty() && !descriptors2.empty()){
    matcher->knnMatch(descriptors1, descriptors2, knn_matches, 2);

    // Фильтрация по Lowe's ratio test
    const float ratio_thresh = 0.7f;
    for (size_t i = 0; i < knn_matches.size(); i++){
        if (knn_matches[i].size() >= 2){
            if (knn_matches[i][0].distance < ratio_thresh * knn_matches[i][1].distance){
                good_matches.push_back(knn_matches[i][0]);
            }
        }
    }
}

cout << "5. Total matches: " << (knn_matches.empty() ? 0 : knn_matches.size()) << endl;
cout << "6. Good matches: " << good_matches.size() << endl;
```

Сопоставление изображений в OpenCV

```
if (good_matches.size() < 4){
    cout << "Not enough good matches for stitching!"  << endl;
    return -1;
}

// Визуализация совпадений
Mat img_matches;
drawMatches(img1, keypoints1, img2, keypoints2, good_matches, img_matches,
            Scalar::all(-1), Scalar::all(-1), vector<char>(),
            DrawMatchesFlags::NOT_DRAW_SINGLE_POINTS);

imshow("5. Keypoint Matches", img_matches);
waitKey(0);

// 4. Совмещение изображений с помощью найденных сопоставлений
vector<Point2f> points1, points2;
for (const auto &match : good_matches){
    points1.push_back(keypoints1[match.queryIdx].pt);
    points2.push_back(keypoints2[match.trainIdx].pt);
}
```

Сопоставление изображений в OpenCV

```
    // Вычисление матрицы гомографии
    Mat H = findHomography(points2, points1, RANSAC, 3.0);
    cout << "7. Homography matrix computed" << endl;
    cout << "Homography matrix:\n" << H << endl;

    Mat result;
    warpPerspective(img2, result, H, Size(img1.cols + img2.cols, img1.rows * 2));

    // Копирование первого изображения в результат
    Mat roi(result, Rect(0, 0, img1.cols, img1.rows));
    img1.copyTo(roi);

    // Обрезка черных областей
    Rect bbox = findNonZeroBoundary(result);
    Mat final_result = result(bbox);

    imshow("6. Stitching Result", final_result);
    waitKey(0);

    cout << "8. Result size: " << final_result.cols << "x" << final_result.rows << endl;

    destroyAllWindows();

    return 0;
}
```

OpenCL

OpenCL (Open Computing Language) — это открытый, кроссплатформенный стандарт для параллельного программирования, разработанный консорциумом Khronos Group. Его ключевая идея заключается в создании единой программной модели для использования разнородных вычислительных ресурсов в рамках одной системы.

OpenCL

OpenCL был создан как ответ на проблему конфликтов кода написанного на NVIDIA коду написанному на AMD или INTEL, предложив универсальный стандарт, который:

- Абстрагирует программиста от аппаратных особенностей конкретных устройств.
- Гарантирует переносимость кода между различными типами процессоров и платформами.
- Устраняет vendor lock-in, позволяя писать один код для всего парка вычислительных устройств.

OpenCL

OpenCL находит применение в областях, требующих интенсивных параллельных вычислений:

- Обработка изображений и видео: высокоскоростная фильтрация, цветокоррекция, декодирование.
- Машинное обучение: ускорение работы и тренировки моделей.
- Научные вычисления: молекулярное моделирование, вычислительная физика и химия.
- Финансовый анализ: сложные математические расчеты для риск-менеджмента и алгоритмической торговли.

OpenCL

Ключевые преимущества:

- Доступ к гетерогенным архитектурам
- Массовый параллелизм
- Аппаратная независимость
- Высокая производительность

OpenCL

Сравнение с классическим многопоточным программированием

Аспект	**Многопоточность (POSIX Threads)**	**OpenCL**
Цель	Параллелизм на уровне ядер **CPU**	Параллелизм на уровне любых вычислительных устройств (CPU, GPU, FPGA)
Масштаб параллелизма	Десятки потоков (по числу логических ядер CPU)	Десятки тысяч потоков на massively-parallel архитектурах (GPU)
Абстракция	Работа с потоками ОС, требует знания архитектуры CPU	Абстракция от железа через единую модель программирования (kernel functions)
Основное применение	Ускорение CPU-ограниченных задач, асинхронность	Ускорение data-parallel задач (одинаковые операции над большим массивом данных)

Установка OpenCL (Windows)

- **Открыть консоль MSYS2 (через которую уже устанавливался C++ на занятии по IDE) и ввести команду:**

  pacman -S mingw-w64-ucrt-x86_64-opencl-headers mingw-w64-ucrt-x86_64-opencl-icd

Установка OpenCL (Windows)

- **Для сборки OpenCL** исходников необходимо добавить в конец списка аргументов "args" tasks.json следующий аргумент "-lOpenCL".
- **Пример части файла:**

```
"args": [
        "-fdiagnostics-color=always",
        "-g",
        "${file}",
        "-o",
        "${fileDirname}\\${fileBasenameNoExtension}.exe",
        "-lopencl"
      ],
```

Установка OpenCL (Linux)

- **Открыть консоль и ввести команду**

  sudo apt-get install -y ocl-icd-libopencl1 opencl-headers clinfo

- **Также необходимо установить драйвера специфичные для модели GPU/CPU установленной в системе:**
  - Для NVIDIA GPU:
    - sudo apt-get install -y nvidia-opencl-dev
  - Для AMD GPU:
    - sudo apt-get install -y opencl-amdgpu-pro
  - Для Intel CPU/GPU:
    - sudo apt-get install -y intel-opencl-icd
- **Проверить установку можно с помощью команды:**

  clinfo

- **Она должна показать количество доступных платформ для вычисления с использованием OpenCL.**

Установка OpenCL (Linux)

- **Для сборки OpenCL** исходников необходимо добавить в конец списка аргументов "args" tasks.json следующий аргумент "-lOpenCL".
- **Пример части файла:**

```
"args": [
    "-fdiagnostics-color=always",
    "-g",
    "${file}",
    "-o",
    "${fileDirname}/${fileBasenameNoExtension}",
    "-lOpenCL",
],
```

Описание работы программы на OpenCL

- **Подготовка к выполнению**
  - Контекст (cl_context)
    - Создается окружение, которое связывает одно или несколько устройств (GPU/CPU) с ресурсами памяти и командами. Это среда выполнения для всех последующих операций.
  - Очередь команд (cl_command_queue)
    - Для каждого устройства в контексте создается очередь. Через нее host-программа отправляет команды на конкретное устройство. Команды могут выполняться по порядку или вне очереди.

Описание работы программы на OpenCL

- **Работа с кодом и данными**
  - Программа (cl_program)
    - Исходный код ядер, написанный на OpenCL C, (о данном коде будет позже, сейчас рассматривается сама программа для запуска вычислений и обработки результатов) загружается в виде строки или из файла. Затем он компилируется для конкретных устройств, указанных в контексте. При этом проверяется синтаксис и оптимизируется код под архитектуру целевого устройства.
  - Буферы памяти (cl_mem)
    - Выделяются области памяти на устройстве (глобальная память) для хранения входных данных и результатов. Указывается тип буфера: только для чтения, только для записи или для чтения и записи.

Описание работы программы на OpenCL

- **Запуск вычислений**
  - Ядро (cl_kernel)
    - Из скомпилированной программы извлекается функция-ядро, которую нужно выполнить. Для нее устанавливаются аргументы: указатели на буферы данных и другие параметры.
  - Передача данных (clEnqueueWriteBuffer)
    - Исходные данные копируются из оперативной памяти хоста в буферы на устройстве. Эта операция ставится в командную очередь.
  - Запуск ядра (clEnqueueNDRangeKernel)
    - Команда на выполнение ядра ставится в очередь. Указывается размерность и общее количество рабочих элементов, которые создаются для параллельной обработки данных.

Описание работы программы на OpenCL

- **Завершение работы**
  - Чтение результатов (clEnqueueReadBuffer)
    - После завершения вычислений результаты копируются из буферов устройства обратно в оперативную память хоста для дальнейшей обработки.
  - Синхронизация (clFinish)
    - Команда блокирует выполнение host-программы до тех пор, пока все операции в командной очереди не будут полностью завершены.
  - Освобождение ресурсов
    - Явно освобождаются все созданные объекты OpenCL (контекст, очередь, программа, буферы, ядра) во избежание утечек памяти.

Структура стандартной программы на OpenCL

- **Инициализация платформы и устройства**
  - Перед созданием контекста необходимо получить платформу и вычислительное устройство.

```
// Получение платформы
cl_int err;
cl_platform_id platform;
err = clGetPlatformIDs(1, &platform, nullptr);

// Получение устройства
cl_device_id device;
err = clGetDeviceIDs(platform, CL_DEVICE_TYPE_GPU, 1, &device, nullptr);
```

Структура стандартной программы на OpenCL

- **clGetPlatformIDs()**
  - Назначение: получает список доступных платформ opencl.
  - Параметры:
    - cl_uint num_entries — максимальное количество записей для возврата.
    - cl_platform_id *platforms — указатель на массив для хранения платформ.
    - cl_uint *num_platforms — возвращает фактическое количество платформ.
  - Возвращает: CL_SUCCESS при успешном выполнении.

Структура стандартной программы на OpenCL

- **clGetDeviceIDs()**
  - Назначение: получает список устройств указанного типа для выбранной платформы.
  - Параметры:
    - cl_platform_id platform — идентификатор платформы.
    - cl_device_type device_type — тип устройства:
      - CL_DEVICE_TYPE_GPU — графические процессоры.
      - CL_DEVICE_TYPE_CPU — центральные процессоры.
      - CL_DEVICE_TYPE_ACCELERATOR — специализированные ускорители.
      - CL_DEVICE_TYPE_ALL — все доступные устройства.
    - cl_uint num_entries — максимальное количество записей для возврата.
    - cl_device_id *devices — указатель на массив для хранения устройств.
    - cl_uint *num_devices — возвращает фактическое количество устройств.
  - Возвращает: CL_SUCCESS при успешном выполнении.

Структура стандартной программы на OpenCL

- **Создание контекста (cl_context)**
  - Контекст связывает выбранное устройство (gpu или cpu) с ресурсами, управляя их совместной работой.

```
// создание контекста
cl_context context = clCreateContext(nullptr, 1, &device, nullptr, nullptr, &err);
```

Структура стандартной программы на OpenCL

- **clCreateContext()**
  - Назначение: создает контекст для выполнения программ на одном или нескольких устройствах.
  - Параметры:
    - const cl_context_properties *properties — свойства контекста (или nullptr по умолчанию).
    - cl_uint num_devices — количество устройств, входящих в контекст.
    - const cl_device_id *devices — массив идентификаторов устройств.
    - void (CL_CALLBACK *pfn_notify)(...) — функция обратного вызова при ошибках (необязательно).
    - void *user_data — пользовательские данные для функции обратного вызова.
    - cl_int *errcode_ret — код возврата (0 при успешном создании).
  - Возвращает: объект контекста (cl_context).

Структура стандартной программы на OpenCL

- **Создание командной очереди (cl_command_queue)**
  - Командная очередь позволяет устройству выполнять задачи последовательно или асинхронно.

```
// создание командной очереди
cl_command_queue queue = clCreateCommandQueue(context, device, 0, &err);
```

Структура стандартной программы на OpenCL

- **clCreateCommandQueue()**
  - Назначение: создает очередь команд для управления выполнением операций на устройстве.
  - Параметры:
    - cl_context context — контекст, к которому привязана очередь.
    - cl_device_id device — устройство, для которого создается очередь.
    - cl_command_queue_properties properties — флаги для задания поведения очереди.
    - cl_int *errcode_ret — код ошибки выполнения.
  - Возвращает: объект очереди команд (cl_command_queue).

Структура стандартной программы на OpenCL

- **Создание и компиляция программы (cl_program)**
  - Создается объект программы из исходного кода opencl c и компилируется для выбранного устройства.

```
// создание программы из исходного кода
cl_program program = clCreateProgramWithSource(context, 1, &kernelSource, nullptr, &err);

// компиляция программы
clBuildProgram(program, 1, &device, nullptr, nullptr, nullptr);
```

Структура стандартной программы на OpenCL

- **clCreateProgramWithSource()**
  - Назначение: создает объект программы из исходного текста ядра.
  - Параметры:
    - cl_context context — контекст выполнения.
    - cl_uint count — количество строк исходного кода.
    - const char **strings — массив строк исходного кода.
    - const size_t *lengths — массив длин строк (или nullptr).
    - cl_int *errcode_ret — код возврата.
  - Возвращает: объект программы (cl_program).

Структура стандартной программы на OpenCL

- **clBuildProgram()**
  - Назначение: компилирует программу для выбранных устройств.
  - Параметры:
    - cl_program program — объект программы.
    - cl_uint num_devices — количество устройств.
    - const cl_device_id *device_list — список устройств.
    - const char *options — опции компиляции (или nullptr).
    - void (CL_CALLBACK *pfn_notify)(...) — функция обратного вызова (опционально).
    - void *user_data — пользовательские данные.
  - Возвращает: CL_SUCCESS при успешной компиляции, иначе код ошибки.

Структура стандартной программы на OpenCL

- **Создание ядра (cl_kernel)**
  - Ядро — это функция, которая будет выполняться параллельно на устройстве.

```
// создание ядра
cl_kernel kernel = clCreateKernel(program, "vectorAdd", &err);
```

Структура стандартной программы на OpenCL

- **clCreateKernel()**
  - Назначение: создает объект ядра из скомпилированной программы.
  - Параметры:
    - cl_program program — скомпилированная программа.
    - const char *kernel_name — имя функции ядра в исходном коде.
    - cl_int *errcode_ret — код возврата.
  - Возвращает: объект ядра (cl_kernel).

Структура стандартной программы на OpenCL

- **Создание буферов памяти (cl_mem)**
  - Буферы памяти используются для хранения входных и выходных данных на устройстве.

```
// создание буфера
cl_mem bufA = clCreateBuffer(context, CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR, sizeof(float)*N, A.data(),
&err);
```

Структура стандартной программы на OpenCL

- **clCreateBuffer()**
  - Назначение: создает буфер памяти на устройстве.
  - Параметры:
    - cl_context context — контекст выполнения.
    - cl_mem_flags flags — флаги доступа к памяти (например, CL_MEM_READ_ONLY).
    - size_t size — размер буфера в байтах.
    - void *host_ptr — указатель на данные хоста (опционально).
    - cl_int *errcode_ret — код возврата.
  - Возвращает: объект буфера (cl_mem).

Структура стандартной программы на OpenCL

- **Передача данных и запуск ядра**
  - Перед запуском ядра необходимо передать данные на устройство, установить аргументы и запустить вычисления.

```
// передача данных на устройство
clEnqueueWriteBuffer(queue, bufA, CL_TRUE, 0, sizeof(float)*N, A.data(), 0, nullptr, nullptr);

// установка аргументов ядра
clSetKernelArg(kernel, 0, sizeof(cl_mem), &bufA);
clSetKernelArg(kernel, 1, sizeof(cl_mem), &bufB);
clSetKernelArg(kernel, 2, sizeof(cl_mem), &bufC);
clSetKernelArg(kernel, 3, sizeof(int), &N);

// запуск ядра
size_t globalSize = N;
clEnqueueNDRangeKernel(queue, kernel, 1, nullptr, &globalSize, nullptr, 0, nullptr, nullptr);
```

Структура стандартной программы на OpenCL

- **clEnqueueWriteBuffer()**
  - Назначение: копирует данные из памяти хоста в память устройства (существует также clEnqueueReadBuffer, которая копирует данные с памяти устройства в память хоста).
  - Параметры:
    - cl_command_queue command_queue — очередь команд.
    - cl_mem buffer — буфер на устройстве.
    - cl_bool blocking_write — блокирующая запись (CL_TRUE) или неблокирующая (CL_FALSE).
    - size_t offset — смещение в буфере устройства.
    - size_t size — размер данных для копирования.
    - const void *ptr — указатель на данные в памяти хоста.
    - cl_uint num_events_in_wait_list — количество событий в списке ожидания.
    - const cl_event *event_wait_list — список событий ожидания.
    - cl_event *event — возвращаемое событие.

Структура стандартной программы на OpenCL

- **clSetKernelArg()**
  - Назначение: устанавливает значение аргумента для ядра.
  - Параметры:
    - cl_kernel kernel — объект ядра.
    - cl_uint arg_index — индекс аргумента (начинается с 0).
    - size_t arg_size — размер аргумента в байтах.
    - const void *arg_value — указатель на данные аргумента.

Структура стандартной программы на OpenCL

- **clEnqueueNDRangeKernel()**
  - Назначение: ставит ядро в очередь на выполнение.
  - Параметры:
    - cl_command_queue command_queue — очередь команд.
    - cl_kernel kernel — объект ядра.
    - cl_uint work_dim — размерность пространства задач (1, 2 или 3).
    - const size_t *global_work_offset — смещение глобального ID.
    - const size_t *global_work_size — общее количество рабочих элементов.
    - const size_t *local_work_size — размер рабочих групп.
    - cl_uint num_events_in_wait_list — количество событий в списке ожидания.
    - const cl_event *event_wait_list — список событий ожидания.
    - cl_event *event — возвращаемое событие.

Структура стандартной программы на OpenCL

- **Ожидание и освобождение ресурсов**
  - После завершения вычислений необходимо дождаться завершения всех команд и освободить ресурсы.

```
// ожидание завершения всех команд
clFinish(queue);

// освобождение ресурсов
clReleaseMemObject(bufA);
clReleaseMemObject(bufB);
clReleaseMemObject(bufC);
clReleaseKernel(kernel);
clReleaseProgram(program);
clReleaseCommandQueue(queue);
clReleaseContext(context);
```

Структура стандартной программы на OpenCL

- **clFinish()**
  - Назначение: блокирует выполнение до завершения всех команд в очереди.
  - Параметры:
    - cl_command_queue command_queue — очередь команд для ожидания.
- Другие функции clReleaseMemObject(), clReleaseKernel() и т.д. нужны для освобождения ресурсов потраченных на инициализацию разных видов специфичных объектов OpenCL.

Описание исходного кода программы ядра OpenCL

Исходный код ядра OpenCL — это функция (или набор функций), написанная на языке OpenCL C (основанном на C99), которая компилируется не во время создания приложения, а во время его выполнения на целевом устройстве (например, на видеокарте). Эта функция выполняется каждым из тысяч параллельных вычислительных ядер (work-items) на устройстве.

Принципы написания исходного кода программы ядра OpenCL

1. Принцип Модели выполнения OpenCL (Execution Model)

- Это самый важный принцип. Вычисления организуются в виде N-мерного пространства индексов (обычно 1D, 2D или 3D).
- Work-Item: это один экземпляр ядра. У каждого ворк-айтема есть свой уникальный глобальный ID, который говорит ему, с каким элементом данных ему работать.
- Work-Group: это группа ворк-айтемов, которые выполняются вместе. Они могут общаться через локальную память (очень быстрая) и синхронизироваться.

Принципы написания исходного кода программы ядра OpenCL

1. Принцип Модели выполнения OpenCL (Execution Model)

Пример (умножение векторов):

```
// __kernel - ключевое слово, указывающее, что это функция ядра
// global - указатель на данные в глобальной памяти
// const - указывает, что это входные данные (только для чтения)

__kernel void vector_mult(__global const float* a,
                          __globalconst float* b,
                          __globalfloat* result)
{
    // Получаем глобальный индекс для этого work-item'a
    int gid = get_global_id(0);

    // Каждый work-item выполняет одну операцию над своим элементом
    result[gid] = a[gid] * b[gid];
}
```

Принципы написания исходного кода программы ядра OpenCL

2. Принцип Учета Иерархии Памяти (Memory Hierarchy)

Эффективный код максимально использует быстрые виды памяти:

- Глобальная память (Global Memory): большая, но медленная (как оперативная память видеокарты). Используется для ввода/вывода.
- Локальная память (Local Memory): маленькая, но очень быстрая память, разделяемая внутри work-group.
- Частная память (Private Memory): самая быстрая, но уникальная для каждого work-item'а (как регистры процессора).

Принципы написания исходного кода программы ядра OpenCL

2. Принцип Учета Иерархии Памяти (Memory Hierarchy)

Пример (сумма элементов вектора с использованием локальной памяти):

```
__kernel void vector_sum_reduction(
    __global const float* a,
    __global float* result,
    __local float* local_cache)
{
    int gid = get_global_id(0);
    int lid = get_local_id(0);
    int group_size = get_local_size(0);
    int group_id = get_group_id(0);

    // 1. Каждый work-item загружает свой элемент в локальный кеш
    local_cache[lid] = a[gid];

    // 2. Синхронизация: ждем пока все в группе загрузят данные
    barrier(CLK_LOCAL_MEM_FENCE);
```

Принципы написания исходного кода программы ядра OpenCL

2. Принцип Учета Иерархии Памяти (Memory Hierarchy)

Продолжение (сумма элементов вектора с использованием локальной памяти):

```
    // 3. Вычисление: Находим сумму всех элементов в рабочей группе
    // Используем алгоритм параллельного сокращения (parallel reduction)
    // Начинаем с полного размера группы и уменьшаем вдвое на каждом шаге
    for (int stride = group_size / 2; stride > 0; stride >>= 1) {
        if (lid < stride) {
            // Складываем элементы на расстоянии stride
            local_cache[lid] += local_cache[lid + stride];
        }
        barrier(CLK_LOCAL_MEM_FENCE); // Синхронизируем на каждом шаге
    }
    // 4. Теперь local_cache[0] содержит сумму всех элементов группы
    // Первый work-item в группе записывает сумму группы в результат
    if (lid == 0) {
        result[group_id] = local_cache[0];
    }
}
```

Принципы написания исходного кода программы ядра OpenCL

3. Принцип Векторизации (Vectorization)

OpenCL C поддерживает векторные типы данных (float4, int8, float16 и т.д.). Одна операция над вектором float4 эквивалентна четырем операциям над float.

Пример:

```
__kernel void vector_mult_float4(__global const float4* a,
                                 __globalconst float4* b,
                                 __global float4* result)
{
    int gid = get_global_id(0);
    result[gid] = a[gid] * b[gid]; // Выполняет 4 умножения за одну операцию
}
```

Принципы написания исходного кода программы ядра OpenCL

4. Принцип Минимизации Ветвления (Minimizing Branching)

GPU плохо обрабатывают условные операторы (if, else, switch), особенно если внутри work-group ворк-айтемы начинают выполнять разные ветки кода ("branch divergence"). Это приводит к тому, что все ветки выполняются последовательно для всех ворк-айтемов в группе.
Поэтому необходимо стараться организовывать алгоритмы так, чтобы все ворк-айтемы в группе шли по одному и тому же пути выполнения. Если ветвление неизбежно, старайтесь, чтобы оно было на уровне всего work-group, а не отдельных work-items.

Принципы написания исходного кода программы ядра OpenCL

Итог:

1. Хост-программа (на C++):
   - Управляет устройствами OpenCL.
   - Выделяет память на устройстве.
   - Копирует данные из RAM в память устройства.
   - Компилирует и запускает исходный код ядра.
   - Копирует результаты обратно.

2. Исходный код ядра (на OpenCL C):
   - Это та программа, которую выполняет устройство.
   - Пишется с учетом массового параллелизма.
   - Должна эффективно использовать иерархию памяти.
   - Стремится к векторизации и минимизации ветвления.

СПбГЭТУ «ЛЭТИ»
ПЕРВЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ

Мастер-класс

Умножение матриц с OpenCL

```
#include <pthread.h>
#include <CL/cl.h>
…

#define WG_SIZE 32
#define PROGRAM_FILE "matrix_multiply.cl"
#define KERNEL_FUNC "matrix_mul"
```

Умножение матриц с OpenCL

Весь код примера на OpenCL можно найти в материалах к данной лекции.
Из-за его объемности разместить его на слайдах презентации без сильной перегруженности не удалось.
Однако код OpenCL C программы ядра для вычислений будет представлен начиная со следующего слайда.

Умножение матриц с OpenCL

```
#define WG_SIZE 32


__kernel void matrix_mul (
    __global int const * first_mat, __global int const * second_mat,
    __global int * result_mat, int side
) {
    // Выделяем области локальной памяти.
    __local float first_fragment[WG_SIZE][WG_SIZE];
    __local float second_fragment[WG_SIZE][WG_SIZE];

    // Получаем значения строки и столбца, которые нужно вычислить у result_mat.
    int col = get_global_id(0);
    int row = get_global_id(1);
    int l_col = get_local_id(0);
    int l_row = get_local_id(1);

    float result_element = 0;
```

Умножение матриц с OpenCL

```
    for (int i = 0; i < (side / WG_SIZE); i++) {
        // Заполняем 1 элемент нового фрагмента умножаемых матриц (остальные заполнят другие work item).
        first_fragment[l_row][l_col] = first_mat[row*side + (i*WG_SIZE + l_col)];
        second_fragment[l_row][l_col] = second_mat[(i*WG_SIZE + l_row)*side + col];

        // Синхронизируем, чтобы first_fragment и second_fragment полностью загрузились в локальную память.
        barrier(CLK_LOCAL_MEM_FENCE);

        // Считаем частичный результат, который получается из фрагментов матриц first_mat и second_mat.
        for (int j = 0; j < WG_SIZE; j++) {
            result_element += first_fragment[l_row][j] * second_fragment[j][l_col];
        }
        // Ждем, пока все work item-ы закончат использовать фрагменты матриц в локальной памяти.
        barrier(CLK_LOCAL_MEM_FENCE);
    }

    // Записываем вычисленное значение.
    result_mat[row*side + col] = result_element;
}
```

OpenBLAS

OpenBLAS (Open Basic Linear Algebra Subprograms) — это высоко оптимизированная библиотека с открытым исходным кодом, реализующая стандарт BLAS (Basic Linear Algebra Subprograms) и его расширения, включая LAPACK (Linear Algebra PACKage).

Ключевые характеристики:

- Архитектура: поддерживает множество процессорных архитектур (x86, ARM, RISC-V и др.).
- Оптимизация: использует аппаратно-специфичные оптимизации под разные CPU.
- Параллелизм: внутренняя многопоточность через pthreads/OpenMP.

OpenBLAS

Области использования:

- Научные вычисления и вычислительная физика.
- Машинное обучение и искусственный интеллект.
- Обработка сигналов и изображений.
- Финансовое моделирование и количественный анализ.

Причины использования:

- Производительность
- Стандартизация
- Кроссплатформенность
- Автоматическая параллелизация
- Простота интеграции

OpenBLAS vs OpenCL

ТАБЛИЦА стр 43-44

Установка OpenBLAS (Windows)

- **Открыть консоль MSYS2 (через которую уже устанавливался C++ на занятии по IDE) и ввести команду:**

  pacman -S mingw-w64-ucrt-x86_64-openblas

Установка OpenBLAS (Windows)

- **Для подсветки синтаксиса** необходимо в .vscode/c_cpp_properties.json добавить путь до файлов h библиотеки. То есть добавить в "includePath" путь "C:/msys64/ucrt64/include/openblas" (если msys2 был установлен по пути C:/msys64).
- **Пример файла:**

```
{
  "configurations": [
    {
      "name": "Win32",
      "includePath": [
        "${default}",
        "C:/msys64/ucrt64/include/openblas"
      ],
      "defines": ["_DEBUG", "UNICODE", "_UNICODE"],
      "windowsSdkVersion": "10.0.26100.0",
      "cStandard": "c17",
      "cppStandard": "c++17",
      "intelliSenseMode": "windows-gcc-x64",
      "compilerPath": "C:/msys64/ucrt64/bin/g++.exe"
    }
  ],
  "version": 4
}
```

Установка OpenBLAS (Windows)

- **Для сборки OpenBLAS** исходников необходимо добавить в конец списка аргументов "args" tasks.json следующие аргументы "-IC:/msys64/ucrt64/include/openblas", "-lopenblas" (путь до файла с исходником openblas и необходимая для работы библиотека).
- **Пример части файла:**

```
"args": [
  "-fdiagnostics-color=always",
  "-g",
  "${file}",
  "-o",
  "${fileDirname}\\${fileBasenameNoExtension}.exe",
  "-IC:/msys64/ucrt64/include/openblas",
  "-lopenblas"
],
```

Установка OpenBLAS (Linux)

- **Открыть консоль и ввести команду**

  sudo apt-get install -y libopenblas-dev

Установка OpenBLAS (Linux)

- **Для сборки OpenBLAS** исходников необходимо добавить в конец списка аргументов "args" tasks.json следующие аргументы "`pkg-config", "--cflags", "--libs", "openblas`" (Они нужны для автоматического подбора параметров для библиотеки через pkg-config).
- **Пример части файла:**

```
"args": [
    "-fdiagnostics-color=always",
    "-g",
    "${file}",
    "-o",
    "${fileDirname}/${fileBasenameNoExtension}",
    "`pkg-config", "--cflags", "--libs", "openblas`",
],
```

Структура стандартной программы на OpenBLAS

❖ BLAS Level 1 - векторные операции

```
// умножение вектора на скаляр и сложение: y = alpha * x + y
void cblas_daxpy(const int N, const double alpha,
                 const double *X, const int incX,
                 double *Y, const int incY);

// скалярное произведение векторов: result = x· y
double cblas_ddot(const int N, const double *X, const int incX,
                  const double *Y, const int incY);

// норма вектора (L2): result = ||x||
double cblas_dnrm2(const int N, const double *X, const int incX);
```

Основные функции OpenBLAS

- **cblas_daxpy()**
  - Назначение: умножает вектор на скаляр и добавляет к другому вектору.
  - Параметры:
    - const int N — количество элементов в векторах.
    - const double alpha — скалярный множитель.
    - const double *X — указатель на исходный вектор.
    - const int incX — шаг между элементами вектора X.
    - double *Y — указатель на вектор результата (обновляется на месте).
    - const int incY — шаг между элементами вектора Y.

Основные функции OpenBLAS

- **cblas_ddot()**
  - Назначение: вычисляет скалярное произведение двух векторов.
  - Параметры:
    - const int N — количество элементов в векторах.
    - const double *X — указатель на первый вектор.
    - const int incX — шаг между элементами вектора X.
    - const double *Y — указатель на второй вектор.
    - const int incY — шаг между элементами вектора Y.
  - Возвращает: скалярное произведение векторов.

Основные функции OpenBLAS

- **cblas_dnrm2()**
  - Назначение: вычисляет евклидову норму (L2) вектора.
  - Параметры:
    - const int N — количество элементов в векторе.
    - const double *X — указатель на вектор.
    - const int incX — шаг между элементами вектора.
  - Возвращает: евклидова норма вектора.

Структура стандартной программы на OpenBLAS

❖ BLAS Level 2 - матрично-векторные операции

```
// умножение матрицы на вектор: y = alpha * A * x + beta * y
void cblas_dgemv(CBLAS_LAYOUT layout, CBLAS_TRANSPOSE TransA,
                 const int M, const int N, const double alpha,
                 const double *A, const int lda,
             const double *X, const int incX,
                 const double beta, double *Y, const int incY);

// внешнее произведение векторов: A = alpha * x * y□ + A
void cblas_dger(CBLAS_LAYOUT layout, const int M, const int N,
                const double alpha, const double *X, const int incX,
                const double *Y, const int incY, double *A, const int lda);
```

Основные функции OpenBLAS

- **cblas_dgemv()**
  - Назначение: умножает матрицу на вектор с масштабированием.
  - Параметры:
    - CBLAS_LAYOUT layout — расположение матрицы:
      - CblasRowMajor — строки расположены последовательно.
      - CblasColMajor — столбцы расположены последовательно.
    - CBLAS_TRANSPOSE TransA — транспонирование матрицы A.
    - const int M — количество строк матрицы.
    - const int N — количество столбцов матрицы.
    - const double alpha — скалярный множитель.
    - const double *A — указатель на матрицу.
    - const int lda — ведущая размерность матрицы.
    - const double *X — указатель на вектор.
    - const int incX — шаг между элементами вектора X.
    - const double beta — скалярный множитель для вектора Y.
    - double *Y — указатель на вектор результата.
    - const int incY — шаг между элементами вектора Y.

Основные функции OpenBLAS

- **cblas_dger()**
  - Назначение: выполняет внешнее произведение векторов и добавляет к матрице.
  - Параметры:
    - CBLAS_LAYOUT layout — расположение матрицы.
    - const int M — количество строк матрицы.
    - const int N — количество столбцов матрицы.
    - const double alpha — скалярный множитель.
    - const double *X — указатель на первый вектор.
    - const int incX — шаг между элементами вектора X.
    - const double *Y — указатель на второй вектор.
    - const int incY — шаг между элементами вектора Y.
    - double *A — указатель на матрицу (обновляется на месте).
    - const int lda — ведущая размерность матрицы.

Структура стандартной программы на OpenBLAS

❖ BLAS Level 3 - матрично-матричные операции

```
// умножение матриц: C = alpha * op(A) * op(B) + beta * C
void cblas_dgemm(CBLAS_LAYOUT layout, CBLAS_TRANSPOSE TransA,
                 CBLAS_TRANSPOSE TransB, const int M, const int N, const int K,
                 const double alpha, const double *A, const int lda,
                 const double *B, const int ldb, const double beta,
                 double *C, const int ldc);

// умножение симметричной матрицы на матрицу: C = alpha * A * B + beta * C
void cblas_dsymm(CBLAS_LAYOUT layout, CBLAS_SIDE Side, CBLAS_UPLO Uplo,
                 const int M, const int N, const double alpha,
                 const double *A, const int lda, const double *B, const int ldb,
                 const double beta, double *C, const int ldc);
```

Основные функции OpenBLAS

- **cblas_dgemm()**
  - Назначение: выполняет умножение матриц с масштабированием (самая используемая функция).
  - Параметры:
    - CBLAS_LAYOUT layout — расположение матриц.
    - CBLAS_TRANSPOSE TransA — транспонирование матрицы A.
    - CBLAS_TRANSPOSE TransB — транспонирование матрицы B.
    - const int M — количество строк матрицы C и op(A).
    - const int N — количество столбцов матрицы C и op(B).
    - const int K — количество столбцов op(A) и строк op(B).
    - const double alpha — скалярный множитель.
    - const double *A — указатель на матрицу A.
    - const int lda — ведущая размерность матрицы A.
    - const double *B — указатель на матрицу B.
    - const int ldb — ведущая размерность матрицы B.
    - const double beta — скалярный множитель для матрицы C.
    - double *C — указатель на матрицу результата.
    - const int ldc — ведущая размерность матрицы C.

Основные функции OpenBLAS

- **cblas_dsymm()**
  - Назначение: умножает симметричную матрицу на матрицу.
  - Параметры:
    - CBLAS_LAYOUT layout — расположение матриц.
    - CBLAS_SIDE Side — сторона умножения:
      - CblasLeft — A * B
      - CblasRight — B * A
    - CBLAS_UPLO Uplo — использование верхней или нижней треугольной части.
    - const int M — количество строк матрицы C.
    - const int N — количество столбцов матрицы C.
    - const double alpha — скалярный множитель.
    - const double *A — указатель на симметричную матрицу A.
    - const int lda — ведущая размерность матрицы A.
    - const double *B — указатель на матрицу B.
    - const int ldb — ведущая размерность матрицы B.
    - const double beta — скалярный множитель для матрицы C.
    - double *C — указатель на матрицу результата.
    - const int ldc — ведущая размерность матрицы C.

Структура стандартной программы на OpenBLAS

❖ LAPACK функции

```
// решение системы линейных уравнений: A * X = B
int LAPACKE_dgesv(int matrix_layout, int N, int NRHS,
                  double *A, int lda, int *ipiv,
                  double *B, int ldb);

// LU-разложение: A = P * L * U
int LAPACKE_dgetrf(int matrix_layout, int M, int N,
                   double *A, int lda, int *ipiv);

// обращение матрицы: A = A¹
int LAPACKE_dgetri(int matrix_layout, int N, double *A,
                   int lda, const int *ipiv);

// собственные значения симметричной матрицы: A * v = λ * v
int LAPACKE_dsyev(int matrix_layout, char jobz, char uplo, int N,
                  double *A, int lda, double *W);
```

Основные функции OpenBLAS

- **LAPACKE_dgesv()**
  - Назначение: решает систему линейных уравнений.
  - Параметры:
    - int matrix_layout — расположение матрицы.
    - int N — порядок матрицы A.
    - int NRHS — количество правых частей.
    - double *A — указатель на матрицу коэффициентов.
    - int lda — ведущая размерность матрицы A.
    - int *ipiv — массив перестановок.
    - double *B — указатель на матрицу правых частей.
    - int ldb — ведущая размерность матрицы B.
  - Возвращает: 0 при успешном выполнении.

Основные функции OpenBLAS

- **LAPACKE_dgetrf()**
  - Назначение: выполняет LU-разложение матрицы.
  - Параметры:
    - int matrix_layout — расположение матрицы.
    - int M — количество строк матрицы.
    - int N — количество столбцов матрицы.
    - double *A — указатель на матрицу для разложения.
    - int lda — ведущая размерность матрицы.
    - int *ipiv — массив перестановок.
  - Возвращает: 0 при успешном выполнении.

Основные функции OpenBLAS

- **LAPACKE_dgetri()**
  - Назначение: вычисляет обратную матрицу после LU-разложения.
  - Параметры:
    - int matrix_layout — расположение матрицы.
    - int N — порядок матрицы.
    - double *A — указатель на матрицу (вход: LU-разложение, выход: обратная).
    - int lda — ведущая размерность матрицы.
    - const int *ipiv — массив перестановок из LU-разложения.
  - Возвращает: 0 при успешном выполнении.

Основные функции OpenBLAS

- **LAPACKE_dsyev()**
  - Назначение: вычисляет собственные значения и векторы симметричной матрицы.
  - Параметры:
    - int matrix_layout — расположение матрицы.
    - char jobz — вычислять собственные векторы ('N' - нет, 'V' - да).
    - char uplo — использование верхней/нижней треугольной части ('U'/'L').
    - int N — порядок матрицы.
    - double *A — указатель на симметричную матрицу.
    - int lda — ведущая размерность матрицы.
    - double *W — массив для собственных значений.
  - Возвращает: 0 при успешном выполнении.

СПбГЭТУ «ЛЭТИ»
ПЕРВЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ

Мастер-класс

Умножение матриц с OpenBLAS

```
#include <iostream>
#include <vector>
#include <random>
#include <chrono>
#include <cmath>
#include <cblas.h>
```

Умножение матриц с OpenBLAS

```
// Генерация случайных матриц
void generateRandomMatrix(std::vector<double> &matrix, int size) {
    std::random_device rd;
    std::mt19937 gen(rd());
    std::uniform_real_distribution<double> dis(1.0, 2.0);

    for (int i = 0; i < size * size; i++) {
        matrix[i] = dis(gen);
    }
}
```

Умножение матриц с OpenBLAS

```
// 1. Последовательное умножение матриц (базовая версия)
void sequentialMatrixMultiply(const std::vector<double> &A,
                              const std::vector<double> &B,
                              std::vector<double> &C,
                              int size) {
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            double sum = 0.0;
            for (int k = 0; k < size; k++) {
                sum += A[i * size + k] * B[k * size + j];
            }
            C[i * size + j] = sum;
        }
    }
}
```

Умножение матриц с OpenBLAS

```
// 2. Умножение матриц с использованием OpenBLAS (double precision)
void openblasMatrixMultiply(const std::vector<double> &A,
                            const std::vector<double> &B,
                            std::vector<double> &C,
                            int size)
{
    // Инициализируем результат нулями
    std::fill(C.begin(), C.end(), 0.0);

    // dgemm - Double-precision GEneral Matrix Multiply
    cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans,
                size, size, size, 1.0,
                A.data(), size,
                B.data(), size, 0.0,
                C.data(), size);
}
```

Умножение матриц с OpenBLAS

```
// Проверка корректности результатов
bool verifyResults(const std::vector<double> &C1,
                   const std::vector<double> &C2,
                   int size) {
    int error_count = 0;
    const double tolerance = 1e-12; // Высокая точность для double

    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            int idx = i * size + j;
            double diff = std::abs(C1[idx] - C2[idx]);
            double max_val = std::max(std::abs(C1[idx]), std::abs(C2[idx]));
```

Умножение матриц с OpenBLAS

```
            // Проверяем относительную ошибку
            if (diff > tolerance && diff > tolerance * max_val) {
                if (error_count < 3) { // Показываем только первые 3 ошибки
                    std::cout << "Mismatch at [" << i << "," << j << "]: "
                              << C1[idx] << " vs " << C2[idx]
                              << " (diff: " << diff << ")" << std::endl;
                }
                error_count++;
            }
        }
    }

    if (error_count > 0) {
        std::cout << "Total errors: " << error_count << std::endl;
        return false;
    }

    return true;
}
```

Умножение матриц с OpenBLAS

```
int main(int argc, char *argv[]){
    std::cout << "=== MATRIX MULTIPLICATION WITH OPENBLAS (DOUBLE PRECISION) ===" << std::endl;

    // Парсинг аргументов командной строки
    if (argc != 2){
        std::cout << "Usage: " << argv[0] << " <matrix_size>" << std::endl;
        std::cout << "Using default size: 1024" << std::endl;
    }

    int matrix_size = (argc == 2) ? std::stoi(argv[1]) : 1024;

    if (matrix_size <= 0) {
        std::cout << "Matrix size must be positive! Using 1024." << std::endl;
        matrix_size = 1024;
    }

    std::cout << "Matrix size: " << matrix_size << "x" << matrix_size << std::endl;
    std::cout << "Total elements: " << matrix_size * matrix_size << std::endl;
```

Умножение матриц с OpenBLAS

```
// Инициализация матриц
std::vector<double> A(matrix_size * matrix_size);
std::vector<double> B(matrix_size * matrix_size);
std::vector<double> C_seq(matrix_size * matrix_size);
std::vector<double> C_openblas(matrix_size * matrix_size);

std::cout << "\nGenerating random matrices..." << std::endl;
generateRandomMatrix(A, matrix_size);
generateRandomMatrix(B, matrix_size);

std::cout << "\n--- PERFORMANCE RESULTS ---" << std::endl;

// 1. Последовательное умножение
std::cout << "1. Sequential multiplication..." << std::endl;
auto start_seq = std::chrono::high_resolution_clock::now();
sequentialMatrixMultiply(A, B, C_seq, matrix_size);
auto end_seq = std::chrono::high_resolution_clock::now();
auto duration_seq = std::chrono::duration_cast<std::chrono::milliseconds>(end_seq - start_seq);
std::cout << "   Time: " << duration_seq.count() << " ms" << std::endl;
```

Умножение матриц с OpenBLAS

```
    // 2. Умножение с OpenBLAS
    std::cout << "2. OpenBLAS multiplication..." << std::endl;
    auto start_openblas = std::chrono::high_resolution_clock::now();
    openblasMatrixMultiply(A, B, C_openblas, matrix_size);
    auto end_openblas = std::chrono::high_resolution_clock::now();
    auto duration_openblas =
std::chrono::duration_cast<std::chrono::milliseconds>(end_openblas - start_openblas);
    std::cout << "   Time: " << duration_openblas.count() << " ms" << std::endl;

    // Сравнение результатов
    std::cout << "\n--- VERIFICATION ---" << std::endl;

    bool exact_match = verifyResults(C_seq, C_openblas, matrix_size);
    std::cout << "Exact element-wise match: " << (exact_match ? "YES" : "NO") <<
std::endl;
```

Умножение матриц с OpenBLAS

```
    // Сводка производительности
    std::cout << "\n--- PERFORMANCE SUMMARY ---" << std::endl;
    double speedup = (double)duration_seq.count() / duration_openblas.count();
    std::cout << "Speedup OpenBLAS over Sequential: " << speedup << "x" << std::endl;

    if (speedup > 1.0) {
        std::cout << "OpenBLAS is " << speedup << " times faster than sequential
implementation!" << std::endl;
    } else {
        std::cout << "Sequential implementation is " << (1.0 / speedup) << " times
faster than OpenBLAS!" << std::endl;
    }

    return 0;
}
```

Полезные ссылки

- Документация OpenCV: https://docs.opencv.org/4.x/index.html
- Документация OpenCL: https://developer.nvidia.com/opencl
- Документация OpenBLAS: https://www.openmathlib.org/OpenBLAS/docs/user_manual/

Вопросы?