##### Міністерство освіти і науки України

Дніпропетровський національний університет імені Олеся Гончара

Факультет прикладної математики

Кафедра математичного забезпечення ЕОМ

**ДИПЛОМНА РОБОТА**

**за ОКР магістра**

|  |  |
| --- | --- |
| Тема: | Розробка програмного забезпечення з локалізації об’єктів на фотозображеннях |

|  |  |
| --- | --- |
| **Виконала**: | студентка гр. ПЗМ-14-1  напряму підготовки  6.050103 Програмна інженерія  Онищенко  Олена Федорівна  “….”………………….  ………………..(підпис) |
|  | |
| **Керівник**: | доц . ПММ, к.ф.-м.н.,  Сорокін В. І.  “….”………………….  .…………….. (підпис) |
|  |  |
|  |  |
| **Рецензент:** | доц. ПОМ, к.ф.-м. н.  Наконечна Т. В.  “….”………………….  …………………….(підпис) |

Дніпропетровськ

2016

# РЕФЕРАТ

Дипломна робота «Розробка програмного забезпечення з локалізації об’єктів на фотозображеннях». Обсяг 49с., 17 рис., 7 табл.,17джерел.

**Об’єкт дослідження:** алгоритми та прийоми, що застосовуються у комп’ютерному зорі для локалізації об’єктів на зображеннях.

**Мета роботи:** оглянути існуючи алгоритми локалізації об’єктів на зображеннях, дослідити метод, заснований на Гафових лісах, та можливість переносу на графічний процесор частини обчислень на етапі прийняття рішень.

**В процесі роботи** було реалізовано програмний продукт, який здатен розпізнавати об’єкти шуканих класів на фотозображеннях. Класифікатор будується на підставі розмічених даних навчальної вибірки, що подаються на вхід програми.

**Реалізація.** Програмне забезпечення реалізовано у середовищі Microsoft Visual Studio 2012 на мові C++ з використанням бібліотеки OpenCV, фреймворку Qt та С++ AMP.

**Область застосування:** результати роботи можуть бути застосовані у окремих прикладних задачах комп’ютерного зору.

**Перелік ключових слів:** ЛОКАЛІЗАЦІЯ ОБ’ЄКТІВ, ГАФОВИЙ ПРОСТІР, ВИПАДКОВІ ЛІСИ, ГІСТОГРАМА НАПРЯМЛЕНИХ ГРАДІЄНТІВ, GPU.

# ANNOTATION

The graduation master’s thesis of the sixth-year student E. Onyshchenko (DNU, Applied Mathematics Department) deals with software development applied for object localization on images.

The object of research are object localization algorithms.

The objective is to examine existing object localization methods, investigate an algorithm based on Hough forests and whether testing-stage computations could be implemented on GPU.

The result of work is implemented software that allows detecting of instances of an object classes on photos. Decision rules are built according to marked training data that were loaded to software.

Software is implemented in the IDE Microsoft Visual Studio 2012 on the language C++ and uses OpenCV library, Qt framework and C++ AMP. Program is supported by operating systems Microsoft Windows 7 with SP1 installed, Windows 8, Windows Server 2008 R2/2012 with DirectX 11 installed.

The results of the work may be used in special applications in computer vision.

Bibliog. 17, ill.17, tab.7.

# ЗМІСТ

[Вступ 5](#_Toc440146428)

[Постановка задачі 5](#_Toc440146429)

[Огляд проблеми 5](#_Toc440146430)

[Розділ 1. Теоретичні положення 8](#_Toc440146431)

[1.1 Дескриптори зображень 8](#_Toc440146432)

[1.1.1 Згортка. Оцінка градієнта зображення 8](#_Toc440146433)

[1.1.2 Гістограма напрямлених градієнтів (HOG) 9](#_Toc440146434)

[1.2 Побудова «візуального словника» (bag of visual words approach) 10](#_Toc440146435)

[1.3 Випадкові ліси 12](#_Toc440146436)

# Вступ

Задача локалізації об’єктів на зображеннях є однією з найбільш інтенсивно досліджуваних проблем в області комп’ютерного зору і включає декілька пов’язаних задач: класифікація зображень, локалізація об’єктів шуканого класу, знаходження конкретного об’єкту, знаходження «схожих» зображень, у тому числі на відео та ін.. Людина вирішує задачу локалізації і класифікації об’єктів напрочуд якісно та швидко, що не є дивним, оскільки вона навчається це робити усе своє життя. За роки існування дисципліна комп’ютерного зору здобула величезний набір технік, що дозволяє покращувати вже існуючі навчальні алгоритми та «заточувати» їх під конкретну предметну область чи розпізнавання однотипних сцен (наприклад, машини, будівлі, пішоходів на вулиці). Деякі з них використані у дипломній роботі.

Дипломна робота містить опис сучасного стану проблеми. У першому розділі розглянуті деякі теоретичні питання, деталі реалізації та використані алгоритми – у другому, у третьому наведені результати досліджень, робота має висновок.

# Постановка задачі

Метою цієї роботи є створення програмного забезпечення, яке на вхід приймало би певну кількість навчальних даних, а на виході отримувало набір правил для локалізації екземплярів об’єктів цільових класів. Навчальними даними є множина , де - чорно-білі або кольорові фотозображення, – кількість каналів зображення; – координати прямокутника з міткою класу, кількість класів. Якщо зображення не містить жодного об’єкту шуканих класів, тобто є негативним прикладом, то c = . Отримані правила мають бути застосованими до множини тестових зображень і встановити, чи є на них екземпляри цільових класів. Якщо є – позначити множину прямокутників з мітками класів, n – кількість знайдених представників, або , якщо нічого не знайдено.

Програмне забезпечення повинно мати дружній до користувача інтерфейс, візуально відображати отримані результати локалізації та мати можливість задавати усі необхідні параметри тренування та тестування.

# Огляд проблеми

Задача локалізації об’єктів не є повністю вирішеною на сьогодні. Під час аналізу зображень комп’ютером виникають складнощі, пов’язані з тим, що об’єкти одного класу можуть сильно відрізнятися, а саме:

* мінливість світлотіньового малюнку, текстур (а)
* мінливість кута зору (б), масштабу (в)
* внутрішньокласова мінливість (в)
* мінливість частин цілого, форм, поз (г)
* перекриття іншими об’єктами (д), тощо.

а б в

г д

Рис. 1 Труднощі, що виникають під час локалізації класів об’єктів

Деякі проблеми вважаються вирішеними, наприклад пошук облич методом Віоли-Джонса[3], чи, наприклад, частково вирішеною є проблема локалізації пішоходів методом Далала-Триггса[4]. Значний прорив у вирішенні задачі локалізації об’єктів відбувся у 2012 р. з появою роботи Крижевського та ін.[5]. Алгоритм базувався на нейронній мережі, для його навчання використовувалися можливості графічних процесорів та велика за обсягом навчальна вибірка. Цей класифікатор був здатен розпізнавати об’єкти 1000 класів з такою точністю, що тільки у 26% випадків метод помилявся і не містив у п’яти найбільш ймовірних гіпотезах правильної відповіді. Згодом ці результати тільки покращувалися іншими авторами, але, не зважаючи на успіх нейронних мереж, вони мають свої недоліки. Зокрема якість роботи іноді може сильно залежати від незначних змін у параметрах. Саме тому спільнота, що займається проблемами комп’ютерного зору, наразі активно експериментує з нейронними мережами: майже кожна корпорація має своїх дослідників. У ході досліджень намагаються отримати від комп’ютера не гірші за людські результати, і за останні роки зроблені вражаючі успіхи у побудові класифікаторів зображень: отримана точність класифікації може бути порівняна з людською[1]. Ці успіхи обумовлені, головним чином, двома факторами:

* розвитком апаратного забезпечення, що призвело до значного збільшення обчислювальних потужностей. Розвиток графічних прискорювачів зробив можливим обробляти величезні об’єми зображень за реальний час.
* появою великих наборів тренувальних даних (більше 14 млн. зображень)[2]. Великі вибірки частково вирішують проблеми, наведені на початку цього розділу.

Для порівняння результатів досліджень є міжнародні конкурси з комп’ютерного зору, які є одночасно базою навчальних та тренувальних вибірок, наприклад Pascal VOC Challenge[[1]](#footnote-2) (2005 – 2012), або ImageNet[2] (з 2010 року).

Деякі приклади програмного забезпечення з локалізації об’єктів можна знайти у відкритому доступі, наприклад у [6][7].

# Розділ 1. Теоретичні положення

## Дескриптори зображень

Дескриптор зображення – це модель, якою можна представити зображення у цифровому вигляді. Наявність великої кількості дескрипторів у комп’ютерному зорі спричинена тим, що візуальне сприйняття людини та комп’ютера кардинально відрізняються ()



Рис. Сприйняття зображення людиною та комп’ютером

Найпоширеніші дескриптори:

* RGB та інші моделі кольорів
* яскравість зображення у відтінках сірого
* результати згорток зображення
* виділені контури (детектор Кенні) та ін.

### 1.1.1 Згортка. Оцінка градієнта зображення

Згортка (англ. Convolution) - це операція, що показує «схожість» однієї функції з відбитою і зсунутою копією іншої. Для зображень згортка – це операція обчислення нового значення заданого пікселя, при якій враховуються значення оточуючих його сусідніх пікселів. Ядро згортки – це матриця довільного розміру і співвідношення сторін; найчастіше використовується квадратна матриця невеликого розміру. Формально дискретна згортка функцій *f(x,y)* та *h(x,y)* розміру визначається (1):

*(*1)

Згортки активно використовуються, зокрема у алгоритмах шумозниження, виділення країв, розпізнавання об’єктів, тощо.

Градієнт – векторна величина, що вказує на напрямок та величину найшвидшого зростання або спадання деякої величини на одиницю простору, аналог похідної для багатовимірного простору:

Градієнт зображення характеризує напрямок найбільшого перепаду яскравості. Оскільки зображення дискретне, застосовують різницеві похідні різних порядків апроксимації, в найпростішому випадку

,

Для знаходження похідних використовують різні оператори, наприклад оператор Собела, Робертса чи Превітта[8]. Оператори роблять згортку зображення ядрами 3х3 або більше для отримання похідних по *x* та *y* у центральній точці. Для оператора Превітта ядро згортки виглядає так:

,

для оператора Собела:

,

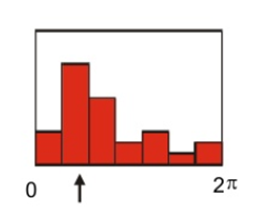
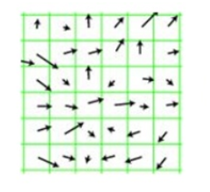
Величина градієнта знаходиться як , напрямок дорівнює . Для прискорення обчислень величину градієнта можна знаходити як суму модулів, а саме

Аналог похідних вищих порядків можна отримати за допомогою згортки попередніх результатів згорток.

### 1.1.2 Гістограма напрямлених градієнтів (HOG)

Мета гістограми напрямлених градієнтів (*Histogram of Oriented Gradients, HOG*)[4] – виділити тенденції в околах точок до зміни яскравості в деякому напрямі. Для її отримання зображення розбивають на невеликі комірки (від 4х4 до 12х12 пікселів) і для кожної комірки виконують алгоритм:

|  |  |  |
| --- | --- | --- |
| Визначити 9 напрямів орієнтації градієнтів  комірки | | |
|  | пікселя у комірці | |
|  |  | визначити напрямок градієнта (а)  визначити номер напряму (б) |
|  | порахувати гістограму напрямків (в) | |



а б в

Рис. Алгоритм пошуку гістограми орієнтації градієнтів

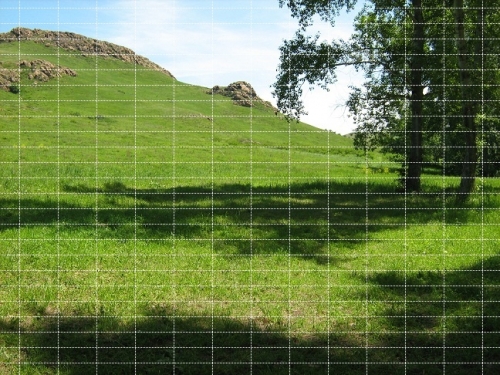
Отримані гістограми об’єднують у блоки (від 1х1 до 4х4 комірок) та нормують. Як правило, блоки перетинаються: одна комірка може входити до декількох блоків. Блоки можуть бути прямокутної або круглої форми, відповідно R-HOG та C-HOG, нормуючі множники отримують теж за різними формулами.

У [4] автори побудували класифікатор на основі метода опорних векторів, який досить добре знаходив пішоходів. Основне припущення полягає у тому, що зовнішній вигляд і форма пішоходів не впливають на напрями змін яскравості. Алгоритм широко використовується для аналізу інших сцен, він є стійким до змін освітлення, масштабування, але є чутливим до зміни положення об’єкту у просторі.

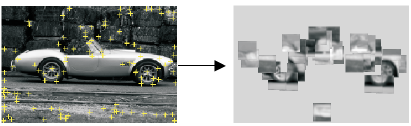
## Побудова «візуального словника» (bag of visual words approach)

Багато алгоритмів локалізації об’єктів використовують розбиття зображень на невеликі шматки певного розміру , де Ii – масив дескрипторів зображення, di – вектор, що містить додаткову інформацію про шматок (наприклад, якщо шматок лежить у границі об’єкту деякого цільового класу, di може містити номер цього класу). Розбиття зображень може відбуватися декілька разів для одного й того ж зображення та за різними критеріями:

1. на зображення накладається рівномірна сітка (а)
2. шматок – окіл деякої «особливої» точки[6]. «Особлива» точка може бути контуром, кутом, точкою з яскравістю, що виділяється та ін..(б)
3. випадково (в)



а. Рівномірна сітка б. Околи «особливих точок»



в. Випадково отримані візуальні слова

Рис. Отримання візуальних слів

Кожному шматку ставиться у відповідність деякий дескриптор для того, щоб шматки можна було порівнювати. Поширеною практикою є опис шматка за допомогою гістограми напрямлених градієнтів. Отримані гістограми кластеризуються одним із відомих методів, найчастіше методом к-середніх ().



Рис. Кластеризація шматків

Кожному центру отриманих кластерів ставиться у відповідність кількість «схожих» шматків, таким чином у вигляді гістограми формується візуальний словник. У такий спосіб зображення можна класифікувати за кількістю візуальних слів, які в ньому зустрічаються, або, навпаки, відсутні.

## 1.3 Випадкові ліси

Бінарне дерево рішень, або дерево класифікації – жадібний алгоритм прийняття рішень, який складається з ієрархії питань у вузлах дерева та множини листя – множини рішень на виході (). Нехай кожний об’єкт, що подається на вхід дерева, характеризується множиною ознак . Якщо дерево рішень ставить у відповідність об’єкту дійсне число, то вирішується задача регресії, якщо мітку класу – класифікації. До переваг цього методу належать простота для розуміння та прозорість ходу прийняття рішення (на відміну від нейронних мереж, в яких пояснити результат складно або неможливо), а також те, що ознаки можуть бути як кількісними, так і якісними. Головними недоліками є проблеми перетренування та складність у побудові оптимального дерева, тому що оптимізація відбувається тільки на поточному кроці. Цю проблему у контексті локалізації об’єктів намагаються вирішити у[13], але отримані результати кардинально не відрізняються від результатів, отриманих алгоритмом, досліджуваним у [15].

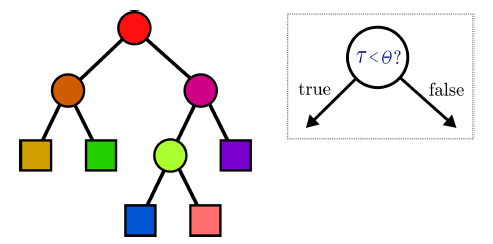


Рис. Зліва: схематичне зображення дерева рішень. Кола містять питання, квадрати – можливі значення на виході. Справа: вузол містить тестову функцію вигляду , де – відповідь деякої ознаки, - поріг для формування відповіді на «так» чи «ні»

Для кожної *fi*можна позначити множину (діапазон) значень, які приймає ознака на множині об’єктів. Нехай *pi* – ймовірність появи деякого значення. Визначимо функцію неоднорідності множини , яка досягає свого мінімуму, якщо на множині об’єктів ознака приймає однакове значення, і максимуму – якщо значення різні. Використовують різні функції, наприклад індекс Гіні(2) або функція ентропії(3):

(2)

(3)

Під час тренування дерева рішень правило для розбиття множини навчальних прикладів обирається таким чином, щоб мінімізувати міру неоднорідності двох отриманих підмножин.

Випадковий ліс[9][12] – множина бінарних дерев рішень. Не зважаючи на назву, кожне дерево є детермінованим. Щоб уникнути кореляції між деревами, тренування має наступні особливості:

1. Дерева рішень тренуються на різних наборах даних. Якщо вибірка обмежена, наприклад, потрібно m вибірок по n елементів у кожній, але є усього n елементів, екземпляри обирають випадковим чином з n елементів з рівною вірогідністю. Якщо n велике, то кількість унікальних елементів дорівнюватиме *(1 — 1/e)* ≈ 63.2%
2. Тренування дерев відбувається на випадкових підмножинах ознак. Є декілька емпіричних правил для визначення кількості ознак, що беруть участь у тренуванні: правило «7 із 32», або .

Під час тестування кожний об’єкт «пропускається» через дерева лісу, і кожне дерево віддає свої голос за приналежність об’єкту до певного класу.

Випадкові ліси широко застосовуються в комп’ютерному зорі[13][14]. У комбінації з методом, описаним у розділі 1.2, вони замінюють словник на множину листя у деревах рішень[10][11].

## 1.4 Перетворення Гафа для виявлення категорій об’єктів

У класичному варіанті перетворення Гафа – алгоритм, що дозволяє виявити прямі лінії на зображенні. Згодом він був трансформований для виявлення кіл, довільних кривих та об’єктів. Перетворення Гафа призначене для пошуку об'єктів, що належать визначеному класу фігур з використанням процедури голосування. Процедура голосування застосовується до простору параметрів, з якого і отримують об'єкти певного класу фігур згідно локальних максимумів в, так званому, накопичувальному просторі (accumulator space), яке будується при обчисленні трансформації Гафа. У досліджуваному алгоритмі[15] накопичувальний простір використовується наступним чином: на етапі тестування шматок зображення пропускається через кожне дерево. Отриманий голос не просто стверджує ймовірність присутності об’єкту, а голосує за центр (), в якому об’єкт може знаходитися. Величина голосу пропорціональна ймовірності присутності об’єкта класу. Після отримання усіх голосів, простір згладжують і знаходять локальні максимуми будь-яким способом.

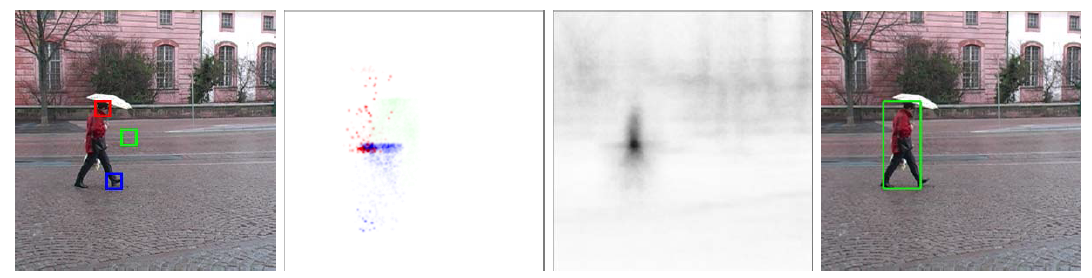


Рис. Зліва направо: окремі шматки, що приймають участь у голосуванні; голоси за центр від усіх дерев; згладжений та інвертований накопичувальний простір

## 1.5 Знаходження локальних максимумів зображення

Є багато алгоритмів знаходження локальних та глобальних максимумів як для одновимірних, так и для багатовимірних даних. Для цієї роботи алгоритм має задовольняти наступним критеріям:

* швидкість роботи
* можливість варіювання порогу, за якого значення може бути взагалі визначене як максимум
* знаходити усі максимуми величиною більшою за поріг, але по одному в околі деякої точки

Був застосований комбінований алгоритм, що використовує сегментацію зображення за порогом та ідею ковзного вікна:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Визначити поріг *t*  Виконати сегментацію за правилом *I(x,y) > t ? I(x,y) : 0*  Визначити розмір ковзного вікна *w\*h* | | | | |
| для , де M – висота зображення | | | | |
|  | для , де N – ширина зображення | | | |
|  |  | якщо *I(x,y) != 0* | | |
|  |  |  | знайти локальний максимум *(xm,ym)* у вікні *[x,y, x+h, y+w]*  якщо (*xm== x+h/2 && ym == y+w/2)* | |
|  |  |  |  | додати *(xm,ym)* до списку максимумів |
|  |  |  | інакше | |
|  |  |  |  | *y += w* |

## 1.6 Вимірювання якості роботи алгоритмів локалізації об’єктів

Для вимірювання якості роботи алгоритмів локалізації об’єктів використовують базові поняття помилки першого та другого роду. Якщо маємо два класи областей на зображенні: об’єкт та фон, тоді в контексті комп’ютерного зору

1. помилка першого роду – якщо об’єкт не було знайдено, тобто об’єкт було класифіковано як фон.
2. помилка другого роду – якщо об’єкт було виявлено там, де його немає, тобто «помилкова тривога»;

У різних додатках менша ймовірність деякої з помилок може бути більш критичною. Наприклад, у системах безпеки намагаються у першу чергу мінімізувати помилку першого роду.

Більшість алгоритмів мають можливість регулювати допустимі помилки, наприклад за деякого параметра зменшувати число помилкових локалізацій. Для оцінки використовується ROC (*receiver operating characteristic*) крива (рис. 5). По вісі *у* відкладають долю об’єктів, що були правильно знайдені (true positive rate, або recall), по *х* – кількість помилково знайдених (false positive rate, FPPW – false positive per window), або величину помилки другого роду. Кожна зміна параметрів алгоритму дає одну точку на кривій. Якість можна оцінити за допомогою площі, обмеженої кривою і віссю х (AUC, *area under ROC curve*). Чим більше ця площа, тим кращим вважається алгоритм. ROC крива, що пролягає під кутом 45ᵒ, рівносильна випадковому рівно ймовірнісному вибору.

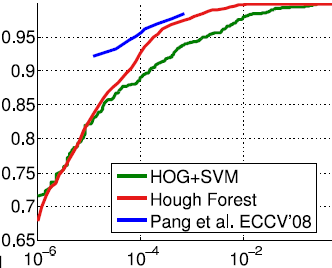


Рис. 5 ROC крива

## 1.7 Оцінка та методи побудови тренувальної вибірки. Boosting, bootstrapping

За різними оцінками, для побудови достатньо гарного класифікатора одного класу об’єктів (тобто такого, що знаходить > 90% відсотків об’єктів та дає досить малу помилку другого роду порядку 10-6), потрібно в середньому 10 тис. позитивних і стільки ж негативних розмічених прикладів в якості тренувальної вибірки[16, л.9].

На властивостях вибірки базується багато алгоритмів локалізації та класифікації, зокрема групи методів Boosting та Bootstrapping.

Ідея методів Boosting полягає у композиції більш слабких простих класифікаторів. Алгоритми є ітеративними і жадібними. На кожній ітерації:

1. обирається найкращий класифікатор, будуються множини об’єктів, що були класифіковані правильно та неправильно.
2. Неправильно класифікованим об’єктам за деяким законом збільшується вага.

Процес повторюється доки точність класифікації не досягне деякого порогу.

Цю ідею використовує, зокрема, метод Віоли-Джонса для пошуку облич.

Bootstrapping (бутстрепінг) – метод, який дозволяє штучним образом збільшити об’єм тренувальної вибірки та підвищити якість локалізації без зміни базового алгоритму. Для кожного тренувального прикладу виконуються незначні, з точки зору людини, перетворення (викривлення):

* незначне масштабування
* невеликі зсуви
* повороти на малу кількість градусів
* дзеркальні відображення (зокрема, горизонтальні)
* незначне розмиття

Таким чином об’єм вибірки можна збільшити в 30-50 разів. В якості негативних прикладів випадковим образом обирають таку ж кількість шматків і тренують алгоритм. Після цього тестують класифікатор на тренувальних даних і виявляють приклади, що були помилково класифіковані, тобто «складні» приклади та ті, що не були знайдені. Ці приклади включають до тренувальних даних. Процес тренування та тестування повторюють декілька разів до досягнення деякого критерію якості[16, л.8].

## 1.8 Технології створення програмного забезпечення

### 1.8.1 Бібліотека OpenCV

OpenCV (англ. *Open Source Computer Vision Library*) — крос платформна бібліотека функцій та алгоритмів комп'ютерного зору, [обробки зображень](https://uk.wikipedia.org/wiki/%D0%9E%D0%B1%D1%80%D0%BE%D0%B1%D0%BA%D0%B0_%D0%B7%D0%BE%D0%B1%D1%80%D0%B0%D0%B6%D0%B5%D0%BD%D1%8C) і чисельних алгоритмів загального призначення з відкритим кодом. Бібліотека написана мовою С, але є інтерфейси для інших мов програмування: С++, Python, Java, Matlab, обгортки для C#, Ruby. OpenCV складається з кількох модулів[17], ось тільки деякі з них:

* **core** – головне ядро, що містить:
  + базові структури даних
  + матричну алгебру
  + алгоритми перетворення типів та ін.
* **imgproc** – модуль обробки зображень, містить низку алгоритмів:
  + Функції для роботи з зображеннями (згортка, масштабування, згладжування, фільтрація, сегментація та ін.)
  + Функції аналізу зображень (пошук контурів, Гафові перетворення, побудову гістограм та ін.)
  + інші алгоритми, наприклад аналізу руху, слідкування за об’єктами, розпізнавання та пошуку об’єктів тощо
* **HighGUI** – модуль для створення інтерфейсу користувача, відповідає за:
  + створення вікон, відображення даних
  + читання/запис зображень у файл
  + та інше, наприклад, захват зображень з камер

Бібліотека також містить низку алгоритмів, адаптованих під виконання на графічних процесорах. Є два модулі, що базуються на OpenCL та CUDA.

Основним контейнером зображень є тип cv::Mat. Матриці можуть бути різних типів даних, від 8-bit unsigned char до 64-bit double і мати до CV\_CN\_MAX=512 каналів. Створити матрицю можна безпосередньо прочитав у неї файл, або за допомогою конструкторів, методу create(), ініціалізувати методом zeros(), тощо. Кожна матриця має заголовок, який містить інформацію: розмір, флаги (глибина, кількість каналів та ін.), покажчик *.data*. Доступ до елементів матриці можливий через метод *.at<тип>(row, col)*, але для пришвидшення рекомендують оперувати з покажчиком. Під час передачі матриці як параметру, присвоювання оператором = та у більшості інших випадків, копіюється тільки заголовок. Для глибокої копії є методи .copyTo() та .clone().

### 1.8.2 Фреймворк Qt

Qt – крос платформний інструментарій розробки програмного забезпечення мовою C++, комплектується IDE QtCreator, яка дозволяє створювати та налагоджувати візуальні інтерфейси. Бібліотека розділена на декілька модулів, для розробки було використано лише декілька за них:

* QtCore — класи ядра бібліотеки для використання іншими модулями;
* QtGui, QtWidgets — компоненти графічного інтерфейсу;

Qt має багато вбудованих класів, що в цілому дублюють та розширюють стандартні класи С++, наприклад: QString, QDateTime, QDir, QFile; контейнери: QList, QLinkedList, QVector, QStack, QMap, QQueue, QSet, QHash та інші. Віджети (Widgets) – це візуальні елементи, з яких складається інтерфейс користувача. Серед них стандартні елементи: QPushButton, QLabel, QLineEdit, QListWidget, QMenu, QMenuBar та інші, усього близько 50 доступних елементів. Звичайно, можна створювати свої елементи керування, якщо є потреба. Структура віджета є ієрархічною, організованою за принципом «частина-ціле». Кожний віджет може містити один чи більше дочірніх віджетів. Така ієрархія немає нічого спільного з ООП. Віджети, що не мають батьків – окремі вікна.

Однією з фундаментальних можливостей в Qt є взаємодія об'єктів за допомогою сигнально-слотових з'єднань. Кожен з об'єктів знаходиться у стані, який визначає сукупність даних, що зберігаються у ньому в даний момент. У відповідь на взаємодію з об'єктом його стан може змінитися і він може повідомити про це інший об'єкт посилаючи йому повідомлення про зміну, або посилаючи сигнал. Візуальні компоненти виробляють сигнали у відповідь на дії користувача (наприклад, натискання кнопки, установки прапорця, редагування тексту в поле введення тощо). Інші об'єкти можуть приєднатися до сигналу слотом - спеціальним методом, який реалізує деяку функціональність і викликається кожного разу, коли був випущений приєднаний до нього сигнал. Для завдання з'єднання сигналів і слотів використовують метод connect() класу QObject. Метод приймає п`ять параметрів: покажчик на об'єкт, який посилає сигнал (sender); назва сигналу (signal), покажчик на об'єкт, який отримує сигнал (receiver); назва слота (slot) і його параметри, тип сигнально-слотового з'єднання (має значення за замовчуванням Qt :: AutoConnection). Один і той же сигнал об'єкта може бути підключений до декількох слотів і навпаки. При цьому послідовність, в якій будуть викликатися приєднані сигнали і слоти, відповідатиме порядку, в якому їх з'єднали.

### 1.8.3 Паралельні обчислення на GPU. C++ AMP

Графічний процесор (*graphics processing unit,* GPU) – окремий пристрій персонального комп’ютера, метою якого є швидка обробка графіки. Може входити до складу відеокарти або гібридного процесору. Організація пам’яті графічних процесорів відрізняється у різних виробників, але загальна схема[18] зображена на .

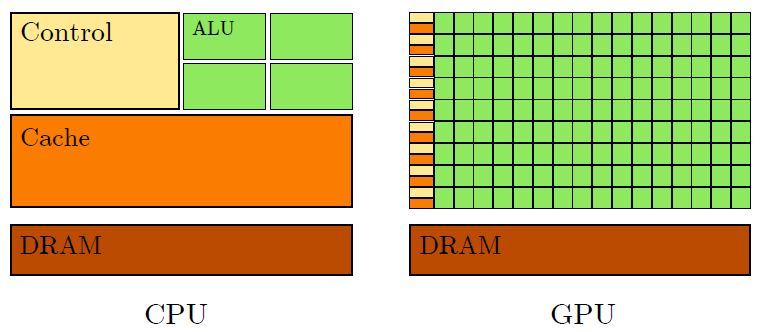


Рис. Схема пам’яті CPU та GPU

Центральний процесор має великий кеш для того, щоб ядру якомога рідше доводилося чекати завершення читання даних з пам'яті. Для ЦП характерне повторне звернення до одних і тих самих даних, що дозволяє отримати від кешування реальний виграш. На відміну від ЦП, графічний процесор оснащений однорівневим обчислювальним конвеєром, невеликим кешем і великим числом ниток (thread), що виконують послідовну вибірку з пам'яті. Отже на GPU швидко виконується велика кількість незалежних один від одного математичних обчислень, проте, на відміну від CPU, більш повільно виконуються умовні оператори, цикли. Інші відмінності стисло наведені у таблиці [19].

*Таблиця 1*

**Відмінності CPU та GPU**

|  |  |  |
| --- | --- | --- |
|  | CPU | GPU |
| Набір команд | GPU має обмежений набір команд | |
| Швидкість | 0,1 Тфлопс[[2]](#footnote-3) (100 Гфлопс) | до 30 Тфлопс |
| Пропускна здатність | ~ 20 Гб/с | ~150Гб/с |
| Енергоспоживання | 1 Гфлопс/вт | 10 Гфлопс/вт |
| Кешування даних | є автоматичне | тільки програмоване |
| Мови програмування та інструменти | багато різних | обмежена кількість |

Особлива архітектура та швидкість математичних обчислень спричинили появу GPGPU (*General-purpose computing for graphics processing units*) – техніки використання обчислювальних можливостей GPU для загальних обчислень, що дозволяє пришвидшити роботу програми у десятки разів. Проте варто зазначити, що такого ефекту досягти вкрай складно, оскільки внесок саме *послідовної* частини в кінцевому підсумку визначає максимальне загальне прискорення, якою би малою не була його частка. Це просте твердження також відоме як закон Амдала[19, с. 33-34].

GPGPU має обмежений набір інструментів, стисле порівняння найпопулярніших з них наведене у табл. .

*Таблиця* *2*

**Порівняння інструментів GPGPU**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Апаратна залежність | Залежність від ОС | Можливість виконання на CPU |
| CUDA | тільки прискорювачі NVIDIA® | Windows, Linux | немає |
| DirectCompute | немає | Windows | немає |
| OpenCL | немає | немає | є |
| С++ AMP | немає | Windows (див. нижче) | є |

C++ AMP (Accelerated Massive Parallelism) – відкритий стандарт та бібліотека на основі DirectX 11 з відкритою специфікацією від компанії Microsoft для реалізації паралельних програм для гетерогенних систем. Для роботи необхідні операційна система Windows 7 з пакетом оновлень SP1 або Windows Server 2008 R2 або вищі, з встановленим DirectX 11. Для розробки ПЗ потрібно середовище Microsoft Visual Studio 2012 або вище. Бібліотека є простою у використанні та апаратно незалежною, на відміну від, наприклад, CUDA. Беззаперечною перевагою є те, що якщо у системі немає графічного процесора, на якому може бути виконано код, то він буде запущений за допомогою центрального процесору і використовувати SSE інструкції. Тобто код буде виконано у будь-якому разі без зайвих зусиль програміста, тому що прискорювачами у C++ AMP є також віртуальні прискорювачі, зокрема, емулятор, встановлюваний разом з Visual Studio, або WARP (прискорювач, реалізований на багатоядерному ЦП). Під час виконання програми, якщо не вказано іншого, обирається кращий з виявлених прискорювачів. Так, якщо є GPU та, наприклад, емулятор, то за замовчуванням буде обраний апаратний прискорювач.[19 с.82-83].

C++ AMP має багато обмежень порівняно з С++: у типах та структурах даних, покажчиках, операціях тощо[20]. Крім того, «вузьким» з точки зору продуктивності місцем є процес копіювання даних с центрального процесора на графічний та навпаки. Тому при переносі алгоритмів важливо враховувати не тільки можливість незалежного виконання потоків, але й кількість даних, що треба скопіювати. Іншим обмеженням є наявність в ОС Windows механізму TDR (*Timeout Detection and Recovery*)[21], що контролює швидкість роботи GPU. Якщо деякий процес виконується надто довго, а саме довше за 2 секунди, екран комп’ютера може перестати перемальовуватися не зважаючи на те, що усі процеси працюють. Коли це трапляється, TDR оновлює стан GPU та відновлює вигляд робочого столу, але усі процеси перериваються. Отже, час виконання є додатковим обмеженням на послідовність операцій, що виконуються в одному thread.

1. **Olga Russakovsky, Jia Deng, Hao Su et al.** ImageNet Large Scale Visual Recognition Challenge / *IJCV,* 2015 – Режим доступу: <http://arxiv.org/abs/1409.0575>
2. IMAGENET [Електронний ресурс] – Режим доступу: <http://image-net.org/>
3. **P. Viola, M. Jones** Robust Real-time Object Detection / *IEEE ICCV Workshop Statistical and Computational Theories of Vision,* July 2001.
4. **N. Dalal, B. Triggs** Histograms of Oriented Gradients for Human Detection / *CVPR (1)*, pp. 886–893, 2005
5. **A. Krizhevsky, I. Sutskever, G. Hinton** ImageNet classification with deep convolutional neural networks / *Neural Information and Processing Systems,* 2012.
6. **Sivic, Zisserman** Video Google: A Text Retrieval Approach to Object Matching in Videos/ *ICCV*, 2003 – <http://www.robots.ox.ac.uk/~vgg/research/vgoogle/index.html>
7. **A. Torralba** A simple object detector with boosting – [Електронний ресурс] – Режим доступу: <http://people.csail.mit.edu/torralba/shortCourseRLOC/boosting/boosting.html>
8. **Р. Гонсалес, Р. Вудс** Цифровая обработка зображений – М.: Техносфера, 2005 – ISBN 5-94836-028-8
9. **L. Breiman**. Random forests. *Machine Learning*, 45(1):5–32, 2001
10. **F. Schroff, A. Criminisi, A. Zisserman** Object class segmentation using random forests. *BMVC*, 2008
11. **J. Shotton, M. Johnson, R.** **Cipolla** Semantic texton forests for image categorization and segmentation. *CVPR*, 2008.
12. **A. Criminisi, J. Shotton, E. Konukoglu** Decision forests: A unified framework for classification, regression, density estimation, manifold learning and semi-supervised learning / *Foundations and Trends in Computer Graphics and Vision*, 2011, pp.81–227
13. **S. Schulter et al.** Alternating Regression Forests for Object Detection and Pose Estimation / *ICCV*, 2013
14. **P. Kontschieder et al**. Context-Sensitive Decision Forests for Object Detection / *Conference on* Neural Information Processing Systems (*NIPS)*, 2012
15. **J. Gall, V. Lempitsky** Class-specific hough forests for object detection // *IEEE Conf. Computer Vision and Pattern Recognition* – 2009
16. **А. Конушин** Введение в компьютерное зрение – Курс лекций / *Лаборатория компьютерной графики ВМК МГУ*, 2012, [Електронний ресурс] – Режим доступу: <https://www.lektorium.tv/course/22847>
17. OpenCV API Reference [Електронний ресурс] – Режим доступу: <http://docs.opencv.org/2.4/modules/refman.html>
18. **А. Григорьев, И. Еремеев, М. Алексеева** Параллельное программирование с использованием технологии CUDA [Електронний ресурс] – Режим доступу: <http://edu.chpc.ru/cuda/mainse2.html#x5-40001.2>
19. **К. Грегори, Э. Миллер** C++ AMP: построение массивно параллельных программ с помощью Microsoft Visual C++. Пер. с англ. Слинкин А. – М.: ДМК Пресс, 2013 – 412с.: ил.
20. **Lingli Zhang**. *restrict(amp) restrictions part 0 of N – introduction* – Режим доступу: <http://blogs.msdn.com/b/nativeconcurrency/archive/2011/12/19/restrict-amp-restrictions-part-0-of-n-introduction.aspx> - Заголовок з екрану
21. Timeout Detection and Recovery (TDR) [Електронний ресурс] – Режим доступу: <https://msdn.microsoft.com/en-us/library/windows/hardware/ff570087%28v=vs.85%29.aspx> – Заголовок з екрану

1. Pascal2. Pattern Analysis, Statistical Modelling and Computational Learning [Електронний ресурс] – Режим доступу: <http://host.robots.ox.ac.uk/pascal/VOC/> [↑](#footnote-ref-2)
2. флопс від англ. ***FL****oating-point* ***O****perations* ***P****er* ***S****econd –* кількість операцій над числом з плаваючою комою за секунду. Відповідно, Гфлопс = 10^9 флопс, ТФлопс = 10^12 флопс [↑](#footnote-ref-3)