Міністерство освіти і науки України
Кам’янець-Подільський національний університет імені Івана Огієнка
Фізико-математичний факультет
Кафедра комп’ютерних наук

Дипломна робота
магістра

з теми: **«РОЗРОБКА МЕТОДУ ТРИВИМІРНОГО КОДУВАННЯ ДИНАМІЧНИХ ВІДЕОІНФОРМАЦІЙНИХ ПОТОКІВ ДЛЯ ПІДВИЩЕННЯ ЯКОСТІ НАДАННЯ ВІДЕОСЕРВІСІВ»**

Виконав: студент групи KN1-M22
спеціальності 122 Комп’ютерні науки
Горпинчук Вадим Анатолійович

Керівник: **Моцик Р. В.,**
кандидат педагогічних наук, доцент,
доцент кафедри комп’ютерних наук

Рецензент: **Оптасюк С. В.,**
кандидат фізико-математичних наук,
доцент, доцент кафедри фізики

Кам’янець-Подільський – 2023

ЗМІСТ

ВСТУП **3**

РОЗДІЛ 1. ОГЛЯД СИСТЕМ ОБРОБКИ БАГАТОПОТОКОВОЇ ВІДЕОІНФОРМАЦІЇ **5**

1.1. Загальна характеристика обробки відео 5

1.2. Розпізнавання та виявлення відео об’єктів 6

1.3. Виявлення мобільних відеооб’єктів за допомогою тимчасових карт ознак 11

1.4. Кодування відео за допомогою потоків і паралельності 14

1.5. Алгоритм YOLO для виявлення об'єктів 19

Висновки до розділу 1 28

РОЗДІЛ 2. ДОСЛІДЖЕННЯ ЗАСТОСУВАННЯ ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ ОБРОБКИ БАГАТОПОТОКОВОЇ ВІДЕОІНФОРМАЦІЇ 29

2.1. Зйомка фотографій із глибиною 29

2.2. Підготовка до глибинної фотозйомки 30

2.3. Зйомка глибини за допомогою камери LiDAR 33

2.4. Створення 3D-об’єктів з фотографій 38

2.5. Багатопоточність в macOS 43

ВИСНОВКИ **55**

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ **57**

ВСТУП

Передача інформації завжди була ключовою ціллю комунікації і з розвитком інтернету ця потреба завжди збільшувалась спочатку з текстових повідомлень, а потім фото та відео інформація, але виникає ситуація коли відео не в достатньому обсязі описує модель або сцену, саме тому виникла потреба у створенні більш детального опису. Сучасні технології активно вирішують цю проблему, але потрібно велика кількість кадрів та досить довгий час на обробку, тому стає актуальним пришвидшити цей процес.

Більшість сучасних персональних комп'ютерів оснащено програмним забезпеченням, яке дозволяє користувачеві компілювати зображення та відео, редагувати зображення та створювати відео на обмеженому рівні. Розкадровки дозволяють додавати аудіофайли та коригувати візуальні зображення, переходи та аудіофайли, які разом визначають загальну тривалість відео. Відеооператори, інженери-електрики та фахівці з інформатики використовують програми, які здатні виконувати більш широкий спектр функцій. Обробка сигналу зазвичай включає застосування комбінації попередніх, внутрішньофільтрів і постфільтрів.

Метою магістерської роботи є підвищення продуктивності та функціональних показників алгоритмів обробки відеоінформації. Дослідження та аналіз існуючих рішень, що використовуються при обробці багатопотокової інформації, зроблені покращення існуючих методів та аналізу результатів. Розглянуто прискорення ефективності та зменшення показників ресурсозатратності алгоритмів, що використовуються у обробці багатопотокової відеоінформації.

У процесі досягнення поставленої мети вирішувалися такі **задачі**:

- дослідити сучасні методи та алгоритми обробки багатопотокової відеоінформації на основі відеоданих для створення тривимірних об'єктів;
- розробка алгоритму оптимізації часу обробки та ресурсів шляхом використання методів багатопокової обробки;
- перевірка роботи алгоритму з реальними даними;

- аналіз результатів роботи алгоритму у ході проведених дослідів.

Об`єкт дослідження: багатопотокова відеоінформація.

Предмет дослідження: система обробки багатопотокової відеоінформації.

Наукова новизна одержаних у магістерській роботі результатів полягає у покращені продуктивності та ефективності існуючих алгоритмів обробки відеоінформації, а саме – саме зменшення розміру ресурсозатратності обчислювальних ресурсів.

РОЗДІЛ 1. ОГЛЯД СИСТЕМ ОБРОБКИ БАГАТОПОТОКОВОЇ ВІДЕОІНФОРМАЦІЇ

1.1. Загальна характеристика обробки відео

Обробка відео використовує апаратне забезпечення, програмне забезпечення та комбінації обох для редагування зображень і звуку, записаних у відеофайлах. Розширені алгоритми в програмному забезпеченні обробки та периферійному обладнанні дозволяють користувачеві виконувати функції редагування за допомогою різноманітних фільтрів. Бажані ефекти можуть бути створені шляхом редагування кадр за кадром або великими партіями.

Відеофайли отримуються з пристрою запису за допомогою кабелю універсальної стандартної шини або кріплення. Потім файли завантажуються в комп'ютерну програму або периферійний пристрій. Перед застосуванням фільтрів, які використовуються при обробці відео, певним програмам потрібна інформація для оптимізаційної системи. Ця інформація дозволяє програмі розраховувати горизонтальні та вертикальні градієнти зображення, визначати потрібні градієнти фільтра та встановлювати параметри функції.

Попередні фільтри, що використовуються в обробці відео, можуть включати зміни контрасту, відхилення та усунення шуму, а також перетворення розміру пікселя. Зміни контрасту дозволяють процесору виділяти окремі ділянки зображення, змінювати перспективу освітлення, затемнювати або освітлювати зображення. Відхилення виключає рух камери або нерівномірні світлові ефекти, які викликають мерехтіння на відео. Усунення шуму видаляє артефакти, включаючи лінії або інші текстурні ефекти, які знижують чіткість зображення. Використовуючи перетворення розміру, користувачі можуть змінити розмір відео з 720 пікселів на 1080 пікселів, обрізати розмір відео або змінити розташування відео на фоні.

Обробка відео за допомогою внутрішніх фільтрів дозволяє користувачам деблокувати або застосовувати методи, які змінюють якість зображення. Деблокування видаляє блокуючі артефакти, іноді отримані стисненими файлами, які зменшують чіткість зображення. Використовуючи розраховані градієнтні аспекти зображень, фільтри можуть зробити зображення, що не мають фокусу, різкість, застосувати виділення навколо певних ділянок зображення або додати графіку та текст до відео. Фільтри також можуть змінювати цілі колірні схеми або змінювати кольори в межах зображення.

Деінтерлейс - це пост-фільтр, який часто використовується в обробці відео. Коли відеореєстратори записують зображення, зображення можуть накладатися один на одного. Це створює артефакти, які можуть включати розмиті зображення, ефект шахової дошки або лінії, які стають видимими під час відтворення. Програми деінтерлейсу усувають ці проблеми, об'єднуючи кадри та забезпечуючи прогресивне сканування без цих візуальних порушень.

1.2. Розпізнавання та виявлення відео об'єктів

Однією з очевидних причин невеликого дисбалансу є те, що відео, по суті, є послідовністю зображень (кадрів) разом. Однак це визначення не може інкапсулювати всю картину того, що таке обробка відео, тому що обробка відео додає до проблеми новий вимір: часовий вимір. Відео – це не лише послідовність зображень, це скоріше послідовність пов'язаних зображень.

Хоча це виглядає як мінімальна різниця, дослідники можуть використовувати цей вимір багатьма способами, які не стосуються окремих зображень. Крім того, через складність відеоданих (розмір, пов'язані анотації) і дорогі обчислення навчання та висновку, було важче пробитися в цій області. Однак нещодавно, з випуском ImageNet VID та інших масивних

наборів відеоданих у другій половині десятиліття, з’явилося все більше і більше досліджень, які були проведені в області виявлення відео, точніше, як дослідники можуть досліджувати часовий вимір.

Першими методами, які з’явилися, були модифікації, застосовані до етапу постобробки конвеєра виявлення об’єктів. Це тому, що він вимагає менше інфраструктури і не вимагає змін в архітектурі моделі. Методи постобробки все одно будуть процесом визначення за кадром, і, отже, не мають підвищення продуктивності (обробка може зайняти трохи більше часу). Однак це може досягти значного підвищення точності.

Одним з підходів до виявлення відеооб’єктів є розділення відео на складові кадри та застосування алгоритму розпізнавання зображення до кожного кадру. Однак це відкидає всі потенційні переваги виведення тимчасової інформації з відео.

Розглянемо, наприклад, проблему оклюзії та повторного захоплення: якщо рамка розпізнавання відеооб’єкта ідентифікує людину, яку потім на короткий час затьмарює пішохід, що проїжджає повз, системі знадобиться тимчасова перспектива, щоб зрозуміти, що вона «втратила» об’єкт, і визначити пріоритет повторного придбання.

Об’єкт може бути втрачено не тільки через оклюзію, але й через розмиття руху, у випадках, коли рух камери або рух об’єкта (або обидва) спричиняють достатні порушення в кадрі, що об’єкти стають смугами або розфокусовані, і це виходить за межі можливості рамки розпізнавання для ідентифікації (рисунок 1).

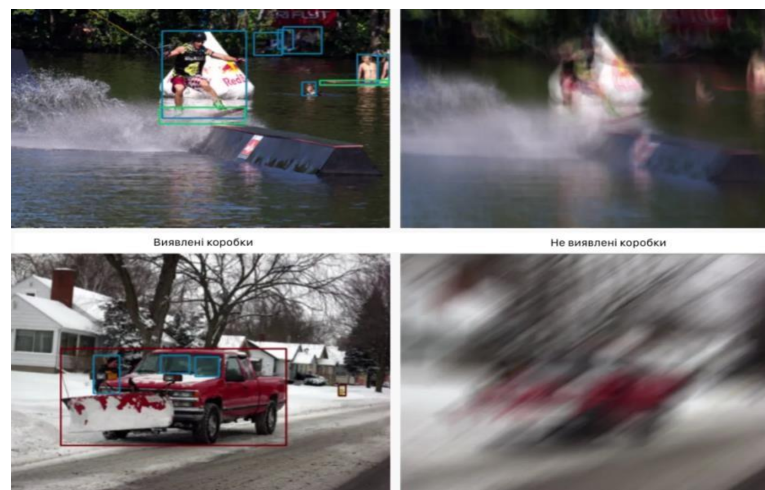


Рисунок 1.1 — Демонстрація проблеми оклюзії

Розмиття в русі руйнує візуальну цілісність об’єкта. Розмитим кадрам знадобиться вручну маркування або надійний алгоритм для підтримки стабільності відстеження на основі попередніх зйомок у послідовності

Якщо система аналізує лише кадри, таке розуміння неможливо, оскільки кожен кадр розглядається як завершений (і закритий) епізод. [3]

Крім цього міркування, для ідентифікації та відстеження об’єкта в обчислювальних термінах інфраструктури виявлення відеооб’єкта коштує менше, ніж багаторазова реєстрація одного і того ж об’єкта для кожного кадру, оскільки, починаючи з кадру №2 і далі, система приблизно знає, що це означає. шукає.

Попередні підходи до тимчасової когерентності покладалися на ідентифікацію після обробки, коли користувач повинен чекати обчислень і коли система отримує достатньо часу обробки (і доступ до щедрих автономних ресурсів) для досягнення високого рівня точності.

Наприклад, Seq-NMS (Sequence Non-Maxmal Suppression), переможець конкурсу ImageNet Large Scale Visual Recognition Challenge 2015, оцінив ймовірність повторного отримання об’єкта на основі вмісту сусідніх кадрів, проаналізованого з коротких послідовностей у вхідному відео (рисунок 1.2).

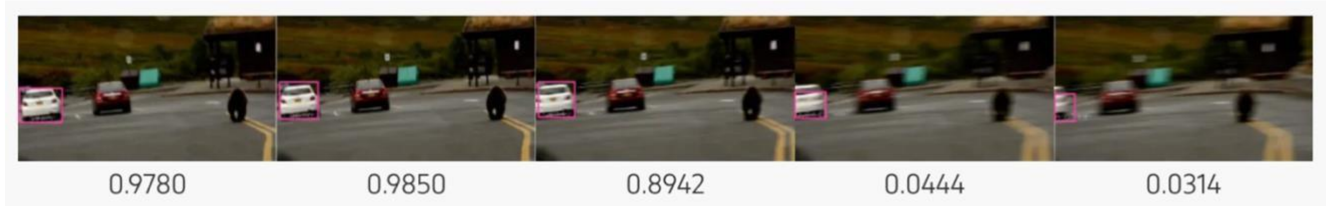


Рисунок 1.2 — Seq-NMS у дії

Нинішній імператив галузі перемістився до розробки систем виявлення в режимі реального часу, з офлайновими фреймворками, які

ефективно перейшли на роль «розробки набору даних» для систем виявлення з низькою або нульовою затримкою.

Оскільки рекуррентні нейронні мережі зосереджені на послідовних даних, вони добре підходять для виявлення відеооб’єктів. У 2017 році співпраця між Технологічним інститутом Джорджії (GIT) і Google Research запропонувала метод виявлення відеооб’єктів на основі RNN, який досягав

швидкості висновку до 15 кадрів в секунду, працюючи на мобільному ЦП (рисунок 1.3).

Рисунок 1.3 — Алгоритм виявлення відеооб’єктів від GIT і Google Research

Дослідження GIT/Google об’єднали швидке виявлення об’єктів окремого зображення зі згортковою короткостроковою пам’яттю для створення алгоритму виявлення відеооб’єктів з високою точністю та низькою затримкою.

Оцінка оптичного потоку генерує двовимірне векторне поле, яке представляє зсув пікселів між одним кадром і сусіднім кадром (попереднім або наступним).

Оптичний потік обчислює траєкторію руху, до якої можна рівномірно застосовувати інші ресурси (рисунок 1.4).

Рисунок 1.4 – Оптичний потік застосований на траєкторію руху автомобілів

Оптичний потік може відображати хід окремих групування пікселів по всій довжині відеоматеріалу, надаючи вказівки, за якими можна виконувати корисні операції. Він використовується протягом тривалого часу в традиційних середовищах із високим вмістом даних, таких як набори для редагування відео, для забезпечення векторів руху, вздовж яких можна застосувати фільтри, і для анімаційних цілей.

З точки зору розпізнавання відеооб'єктів, оптичний потік дозволяє обчислювати розривні траєкторії об'єктів, оскільки він може обчислювати середні шляхи таким чином, що неможливо за допомогою старих методів, таких як підхід Лукаса-Канаде. Останній розглядає постійний потік лише серед згрупованих кадрів і не може сформувати надлишковий зв'язок між кількома групами дій, навіть якщо ці угруповання можуть представляти ту саму подію, переривану такими факторами, як оклюзія та розмиття в русі. (рисунок 1.5)

Рисунок 1.5 — Реалізація проекта оптичниго потоку Лукаса-Канаде

Наскрізна оцінка оптичного потоку, природно, можлива лише як завдання постобробки. Результати цього можуть бути використані для розробки алгоритмів для оцінки оптичного потоку, які можна застосувати до потоків відео в реальному часі.

Багато нових алгоритмів виявлення відео на основі оптичного потоку використовують розріджене розповсюдження ознак, щоб генерувати безперервні оцінки потоку та заповнювати прогалини, де випадковість перервала дію.

Крім того, часто з’являються нові набори даних, що стосуються оптичного потоку, як-от набір даних MPI-Sintel Optical Flow, який містить розширені фрагменти послідовності, розмиття в русі, нефокусовані випадки, атмосферне спотворення, дзеркальні відображення, великі рухи та багато інших складних аспектів для відеооб’єктів. виявлення та розпізнавання.

1.3. Виявлення мобільних відеооб’єктів за допомогою тимчасових карт ознак.

Згорткові нейронні мережі твердо зарекомендували себе як найсучасніший метод виявлення об’єктів з одним зображенням. Однак великі витрати на пам’ять і повільний час обчислень цих мереж обмежили їх практичне застосування. Зокрема, ефективність є основним фактором при

розробці моделей для мобільних і вбудованих платформ. Нещодавно нові архітектури дозволили нейронним мережам працювати на низьких обчислювальних бюджетах з конкурентоспроможною продуктивністю при виявленні об’єктів з одним зображенням. Тим не менш, відеодомен представляє додаткову можливість і проблему використання тимчасових сигналів, і було незрозуміло, як створити порівняно ефективну структуру виявлення для відеосцен. У нашому дослідженні досліджується ідея розбудови цих однокадрових моделей шляхом додавання тимчасової усвідомленості при збереженні їх швидкості та низького споживання ресурсів.

Відео містить різні тимчасові сигнали, які можна використовувати для отримання більш точного та стабільного виявлення об’єктів, ніж на окремих зображеннях. Оскільки відео демонструють тимчасову безперервність, об’єкти в сусідніх кадрах залишаться в подібних місцях, а виявлення не буде істотно відрізнятися. Отже, інформація про виявлення з попередніх кадрів може бути використана для уточнення прогнозів у поточному кадрі. Наприклад, оскільки мережа може переглядати об’єкт у різних позах у кадрах, вона може виявляти об’єкт точніше. Мережа також стане впевненішою щодо прогнозів з часом, зменшуючи проблему нестабільності виявлення об’єктів з одним зображенням.

Останні роботи показали, що ця безперервність поширюється на простір об’єктів, а проміжні карти об’єктів, витягнуті з сусідніх кадрів відео, також мають високу кореляцію. У нашій роботі ми зацікавлені в тому, щоб додати тимчасову обізнаність у простір ознак, а не лише на остаточних виявленнях через більшу кількість інформації, доступної в проміжних шарах. Ми використовуємо безперервність на рівні об’єктів, кондиціонуючи карти ознак кожного кадру відповідними картами об’єктів з попередніх кадрів за допомогою повторюваних мережевих архітектур.

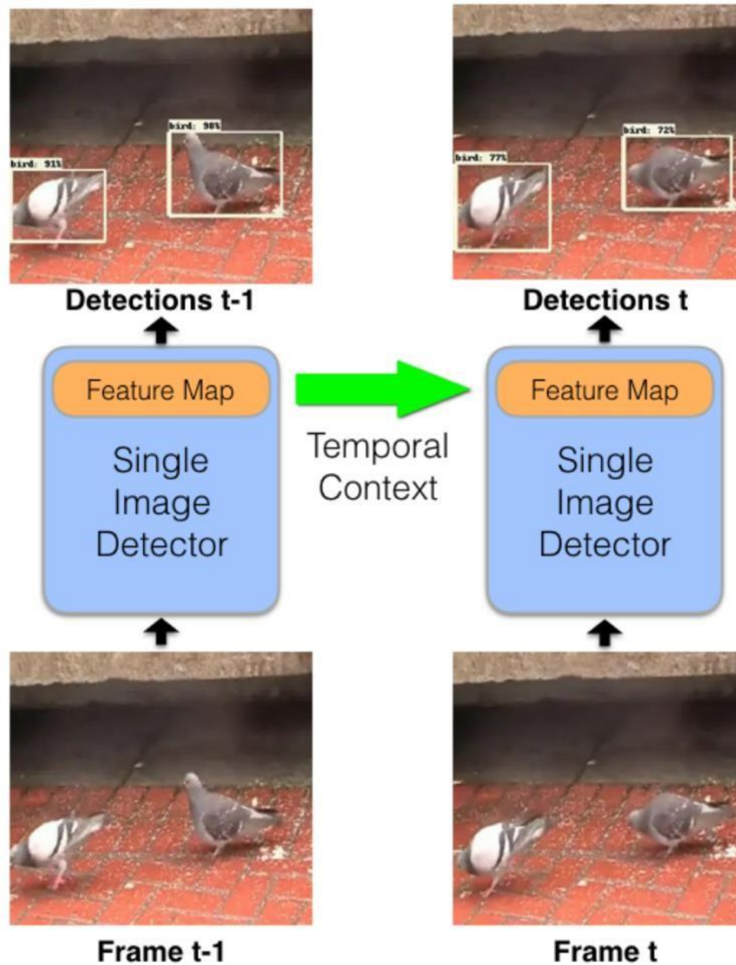


Рисунок 1.6 — Останні методи виявлення відеооб’єктів все ще

Метод генерує карти ознак із спільною згортковою рекурентною одиницею, утвореною шляхом поєднання стандартного згорткового шару зі згортковим LSTM. Мета полягає в тому, щоб згортковий шар видав гіпотезу карти об’єктів, яка потім подається в LSTM і об’єднується з тимчасовим контекстом з попередніх кадрів, щоб вивести витончену, тимчасову карту об’єктів. На малюнку 6 показана ілюстрація нашого методу. Цей підхід дозволяє нам отримати вигоду від досягнення ефективного виявлення об’єктів нерухомого зображення, оскільки ми можемо просто розширити деякі згорткові шари в цих моделях за допомогою наших згортково-рекурентних одиниць. Повторювані шари здатні поширювати тимчасові

сигнали між кадрами, дозволяючи мережі отримати доступ до все більшої кількості інформації під час обробки відео.

Щоб продемонструвати ефективність нашої моделі, ми оцінюємо набір даних Imagenet VID. Наш метод вигідно відрізняється від ефективних однокадрових базових ліній, і ми стверджуємо, що це покращення повинно бути пов'язане з успішним використанням тимчасового контексту, оскільки ми не додаємо жодної додаткової дискримінаційної сили. Ми представляємо варіанти моделей від 200M до 1100M множення-додавання та від 1 до 3,5 мільйонів параметрів, які легко розгорнути на широкому спектрі мобільних і вбудованих платформ. Наскільки нам відомо, наша модель є першою мобільною нейронною мережею для виявлення відеооб'єктів.

1.4. Кодування відео за допомогою потоків і паралельності

Розпаралелювання з використанням потоків на кількох логічних процесорах є привабливим та ефективним способом оптимізації програмного забезпечення. Оскільки технології моделювання кількох процесорів (наприклад, Hyper Threading) і процесорів, що містять кілька ядер, стають стандартом для обчислень навіть на рівні споживача, важливість розпаралелювання стає очевидною.

Однак, щоб правильно розпаралелювати програмне забезпечення, важливо досить добре зрозуміти алгоритм, щоб визначити, чи краще підійдуть дані чи функціональна декомпозиція. Прекрасним прикладом, що демонструє переваги паралелізації, є кодування відео за допомогою кодера H.264.

Оскільки новий стандарт кодеків стає складнішим, процеси кодування та декодування вимагають набагато більшої обчислювальної потужності, ніж більшість існуючих стандартів. Стандарт H.264 включає ряд нових функцій і вимагає набагато більше обчислень, ніж більшість існуючих стандартів, таких як MPEG-2 і MPEG-4.

Навіть після оптимізації медіа-інструкцій кодер H.264 з роздільною здатністю CIF все ще недостатньо швидкий, щоб задовольнити очікування обробки відео в реальному часі. Таким чином, використання паралелізму на рівні потоків для підвищення продуктивності кодерів H.264 стає все більш привабливим.

Як показано в цьому прикладі з оптимізації дизайну відеокодера H.264 з використанням потоків і паралельності, багатопотокова робота на основі моделі програмування OpenMP є простим, але ефективним способом використання паралельності, який вимагає лише кількох додаткових прагм у серійний номер.

Розробники можуть покладатися на те, що компілятор автоматично перетворює послідовний код у багатопотоковий код за допомогою додавання прагм OpenMP. Результати продуктивності показали, що код, згенерований компілятором Intel, забезпечує оптимально збільшену швидкість порівняно з добре оптимізованим послідовним кодом в архітектурі з технологією HyperThreading, часто підвищуючи продуктивність на 20 відсотків у порівнянні з власними паралельними прискореннями, ~4x без HT в цьому випадку з дуже невеликими додатковими витратами.

Використання паралелізму на рівні потоків є привабливим підходом до підвищення продуктивності мультимедійних програм, які працюють на багатопоточних процесорах загального призначення. Враховуючи нові двоядерні та нові багатоядерні процесори, чим раніше ви почнете розробляти багатопоточність, тим краще.

Як ви побачите, одна реалізація, яка використовує модель черги завдань, трохи повільніша за оптимальну продуктивність, але прикладну програму легше читати. Інший метод стосується швидкості. Результати показали збільшення швидкості в діапазоні від 3,97x до 4,69x в порівнянні з добре оптимізованою продуктивністю послідовного коду в системі з чотирьох процесорів Intel Xeon з технологією HT.

H.264 (ISO/IEC 2002) — це новий стандарт кодування відео, який був запропонований Об'єднаною групою відео (JVT). Новий стандарт спрямований на якісне кодування відеоконтенту з дуже низькими бітрейтами. H.264 використовує ту саму модель для гібридної блочної компенсації руху та кодування трансформації, яка використовується в існуючих стандартах, наприклад, для H.263 і MPEG-4 (ISO/IEC 1998).

Крім того, ряд нових функцій і можливостей H.264 покращують продуктивність коду. Оскільки стандарт стає складнішим, процес кодування вимагає набагато більшої обчислювальної потужності, ніж більшість існуючих стандартів. Отже, вам потрібен ряд механізмів для підвищення швидкості кодера.

Одним із способів підвищити швидкість програми є паралельна обробка завдань. Чжоу і Чен продемонстрували, що використання технології MMX/SSE/SSE2 підвищило продуктивність декодера H.264 в діапазоні від двох до чотирьох. Корпорація Intel застосувала ту саму техніку до еталонного кодера H.264, досягнувши результатів у рисунку 1, використовуючи лише оптимізацію SIMD.

Module	Speedup
SAD Calculation	3.5x
Hadamard Transform	1.6x
Sub-Pel Search	1.3x
Integer Transform and Quantization	1.3x
¼ Pel Interpolation	2.0x

Рисунок 1.7 — Прискорення ключових модулів кодера H.264 лише з SIMD

Хоча кодер працює в два-три рази швидше завдяки оптимізації SIMD, швидкість все ще недостатня, щоб відповідати очікуванням обробки відео в реальному часі.

Крім того, оптимізований послідовний код не може використовувати переваги технології Hyper-Threading і багатопроцесорного розподілу навантаження, двох ключових прискорювачів продуктивності, які підтримуються архітектурою Intel. Іншими словами, ви все ще можете значно покращити продуктивність кодера H.264, використовуючи паралельність на рівні потоків.

Паралелізація кодера H.264. Використовуючи паралелізм на рівні потоків на різних рівнях, ви можете скористатися потенційними можливостями підвищення продуктивності. Щоб досягти найбільшого збільшення швидкості в порівнянні з добре налаштованим послідовним кодом на процесорі з технологією HT, ви повинні враховувати наступні характеристики, коли ви перепроектуєте кодер H.264 для паралельного програмування:

- Критерії вибору розділів даних або завдань;
- Судження про зернистість потоку;
- Як перша реалізація використовує дві черги зрізів;
- Як друга реалізація використовує одну чергу завдань;

Декомпозиція області завдань і даних може розділити процес кодування H.264 на кілька потоків за допомогою функціональної декомпозиції або розкладання домену даних.

Функціональна декомпозиція. Кожен кадр повинен проходити ряд функціональних етапів: оцінка руху, компенсація руху, інтегральне перетворення, квантування та ентропійне кодування. Система відліку також потребує зворотної кваліфікації, зворотного інтегрального перетворення та фільтрації. Ці функції можна було б дослідити для того, щоб зробити ці завдання паралельними.

Декомпозиція домену даних. Як показано на рисунку 2 нижче, кодер H.264 обробляє відеопослідовність як багато груп зображень (GOP). Кожен GOP містить ряд кадрів. Кожен кадр поділений на фрагменти. Кожен

фрагмент є одиницею кодування і не залежить від інших фрагментів у тому самому фреймі.

Зріз можна далі розкласти на макроблок, який є одиницею оцінки руху та ентропійного кодування. Нарешті, макроблок можна розділити на блоки та субблоки. Усі можливі місця для розпаралелювання кодера.

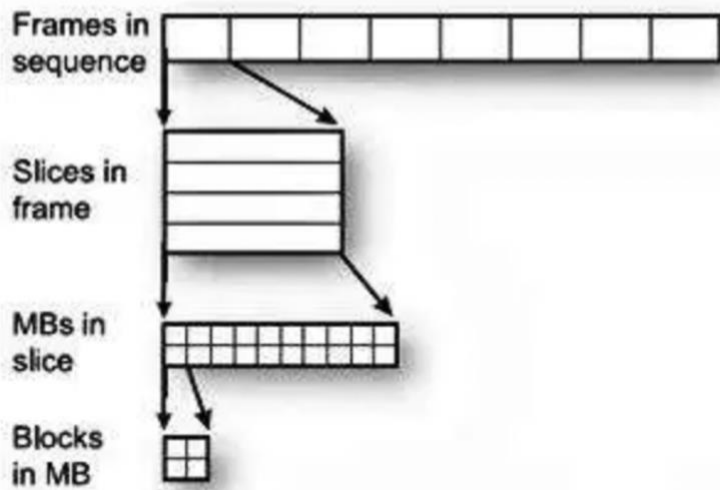


Рисунок 1.8 — Ієрархія декомпозиції домену даних у кодері H.264

Щоб вибрати оптимальну схему завдання або розподілу даних, порівняйте переваги та недоліки двох схем нижче:

Масштабованість. У декомпозиції домену даних, щоб збільшити кількість потоків, ви можете зменшити розмір блоку обробки кожного потоку. Через ієрархічну структуру групових груп, кадрів, фрагментів, макроблоків і блоків у вас є багато варіантів розміру блоку обробки, завдяки чому досягається хороша масштабованість.

У функціональній декомпозиції кожен потік виконує різну функцію. Щоб збільшити кількість потоків, розділіть функцію на два або більше потоків, якщо функція не є непорушною.

Баланс навантаження. При декомпозиції домену даних кожен потік виконує одну і ту ж операцію з різним блоком даних, який має однаковий розмір. Теоретично, без промахів кешу або інших недетермінованих факторів

усі потоки повинні мати однаковий час обробки. З іншого боку, важко досягти гарного балансу навантаження між функціями, оскільки обраний алгоритм визначає час виконання кожної функції.

1.5. Алгоритм YOLO для виявлення об'єктів

YOLO – це алгоритм, який використовує нейронні мережі для виявлення об’єктів у реальному часі. Цей алгоритм популярний завдяки своїй швидкості та точності. Його використовували в різних програмах для виявлення сигналів світлофора, людей і тварин.

Виявлення об’єктів – це явище в комп’ютерному баченні, яке включає виявлення різних об’єктів у цифрових зображеннях або відео. Деякі з виявлених об’єктів включають людей, автомобілі, стільці, каміння, будівлі та тварин. Це явище намагається відповісти на два основних питання:

Виявлення об’єктів складається з різних підходів, таких як швидкий R-CNN, Retina-Net і Single-Shot MultiBox Detector (SSD). Хоча ці підходи вирішили проблеми обмеження даних і моделювання при виявленні об’єктів, вони не можуть виявити об’єкти за один запуск алгоритму. Алгоритм YOLO набув популярності завдяки своїй вищій продуктивності порівняно з вищезгаданими методами виявлення об'єктів.

YOLO — це абревіатура від терміну «Ти дивишся лише раз». Це алгоритм, який виявляє та розпізнає різні об’єкти на зображенні (у режимі реального часу). Виявлення об’єктів у YOLO виконується як проблема регресії та надає ймовірності класів виявлених зображень.

Алгоритм YOLO використовує згорткові нейронні мережі (CNN) для виявлення об’єктів у режимі реального часу. Як випливає з назви, алгоритм вимагає лише одного прямого поширення через нейронну мережу для виявлення об’єктів.

Це означає, що прогнозування всього зображення виконується за один запуск алгоритму. CNN використовується для одночасного прогнозування різних ймовірностей класів і обмежувальних квадратів.

Алгоритм YOLO складається з різних варіантів. Деякі з поширених включають крихітні YOLO та YOLOv3.

Алгоритм YOLO важливий з наступних причин:

Швидкість: цей алгоритм покращує швидкість виявлення, оскільки він може передбачати об’єкти в режимі реального часу.

Висока точність: YOLO – це методика прогнозування, яка забезпечує точні результати з мінімальними фоновими помилками.

Можливості навчання: Алгоритм має чудові можливості навчання, які дозволяють йому вивчати представлення об’єктів і застосовувати їх при виявленні об’єктів.

Алгоритм YOLO працює, використовуючи наступні три методи:

- Залишкові блоки;
- Регресія граничної рамки;
- Intersection Over Union (IOU);

Спочатку зображення розбивається на різні сітки. Кожна сітка має розмір S x S. На наступному зображенні показано, як вхідне зображення поділяється на сітки.

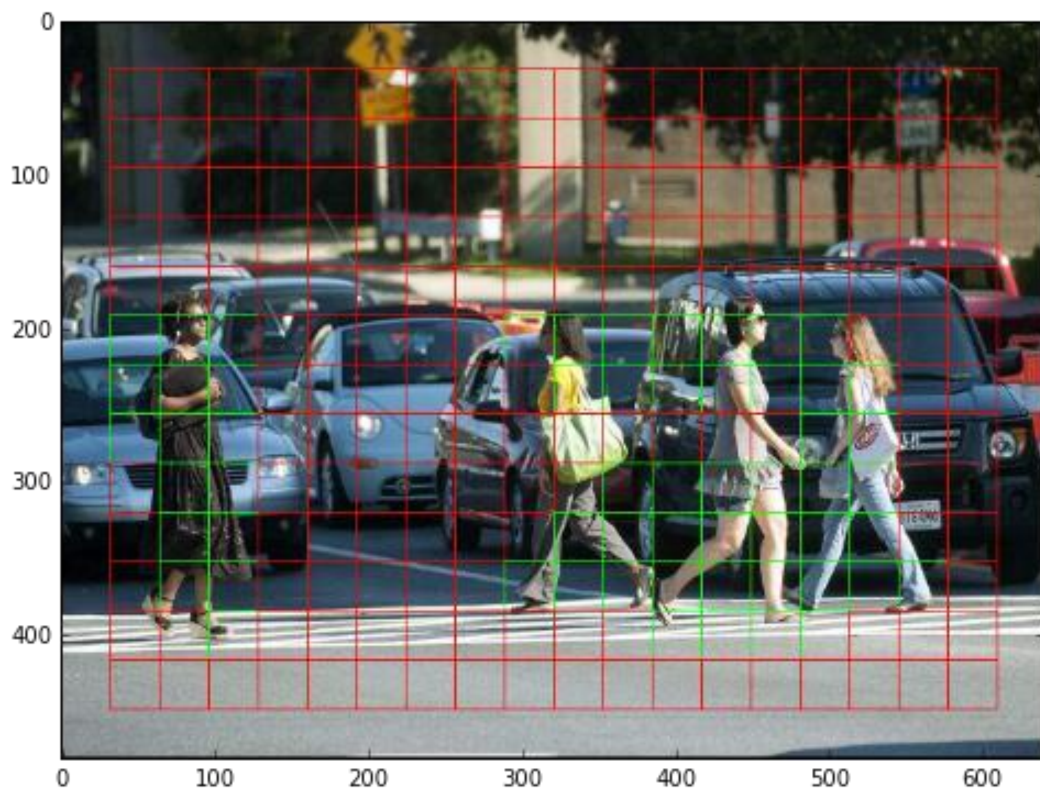


Рисунок 1.9 — Клітинна сітка

На зображенні вище є багато клітинок сітки однакового розміру. Кожна клітинка сітки буде виявляти об'єкти, які з'являються всередині них (Рисунок 9). Наприклад, якщо в певній комірці сітки з’являється центр об’єкта, то ця клітинка відповідатиме за його виявлення. На зображенні вище є багато клітинок сітки однакового розміру. Кожна клітинка сітки буде виявляти об'єкти, які з'являються всередині них. Наприклад, якщо в певній комірці сітки з’являється центр об’єкта, то ця клітинка відповідатиме за його виявлення.

Регресія граничної рамки

Обмежувальна рамка — це контур, який виділяє об’єкт на зображенні.

Кожна обмежувальна рамка на зображенні складається з таких атрибутів:

- Ширина (ч.в.);
- Зріст (bh);
- Клас (наприклад, людина, автомобіль, світлофор тощо) - це позначається буквою c;
- Центр обмежувальної рамки (bx,by);

На наступному зображенні показано приклад рамки. Обмежувальна рамка зображена жовтим контуром.

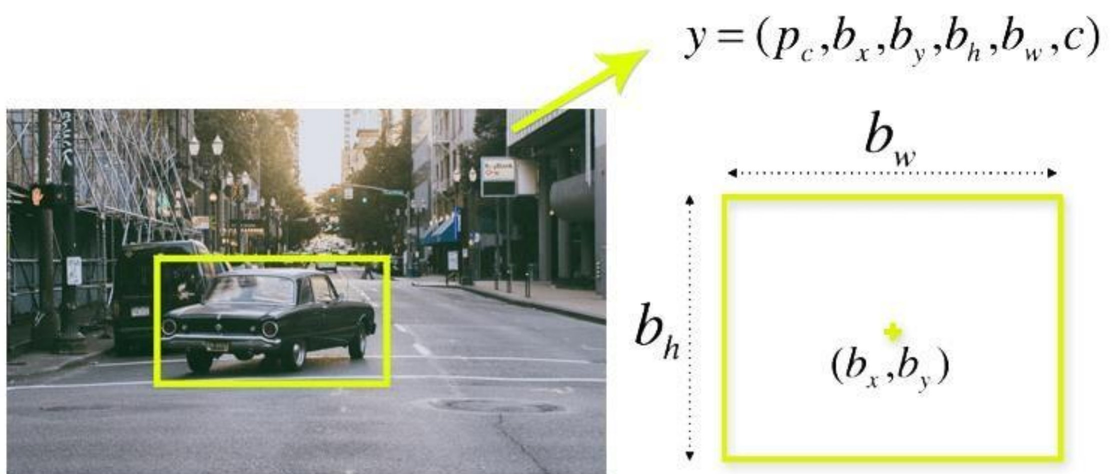


Рисунок 1.10 — Обмежувальна рамка

YOLO використовує регресію однієї рамки, щоб передбачити висоту, ширину, центр і клас об’єктів. На зображенні вище відображає ймовірність появи об’єкта в обмежувальній рамці.

Перетин через об’єднання (IOU) — це явище у виявленні об’єктів, яке описує, як блоки перекриваються. YOLO використовує IOU, щоб створити вихідне поле, яке ідеально оточує об’єкти.

Кожна клітинка сітки відповідає за прогнозування обмежувальних рамок та їх показників довіри. IOU дорівнює 1, якщо передбачена обмежувальна рамка збігається з реальним квадратом. Цей механізм усуває обмежувальні квадрати, які не дорівнюють справжньому прямокутнику.

На наступному зображенні наведено простий приклад роботи IOU.

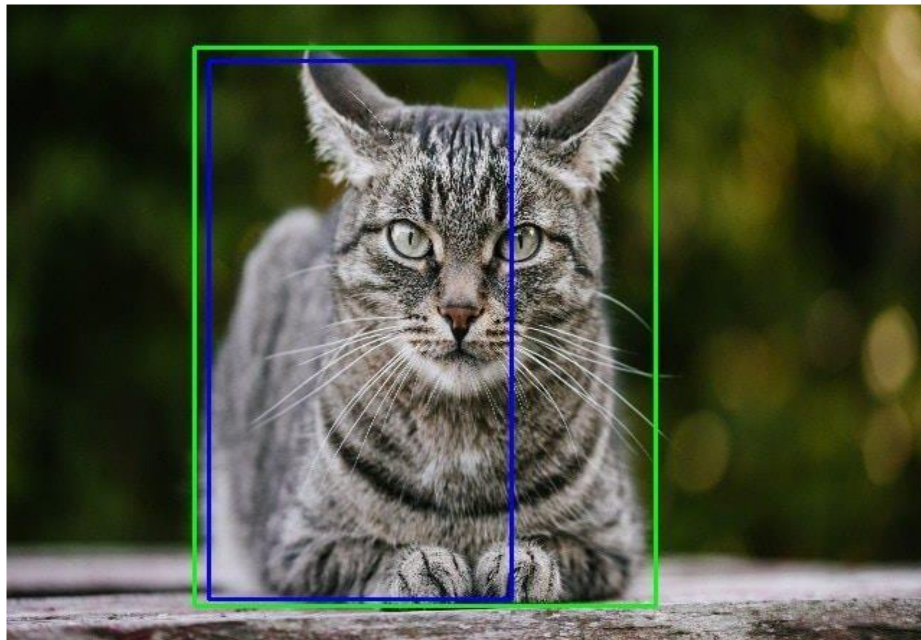


Рисунок 1.11 — Обмежувальні рамка

На зображенні вище є дві обмежувальні прямокутники, одна зелена, а інша синя. Синій прямокутник – це передбачена коробка, а зелена – реальна коробка.

YOLO гарантує, що дві обмежувальні рамки рівні. Поєднання трьох технік.

На наступному зображенні показано, як три методи застосовуються для отримання остаточних результатів виявлення.

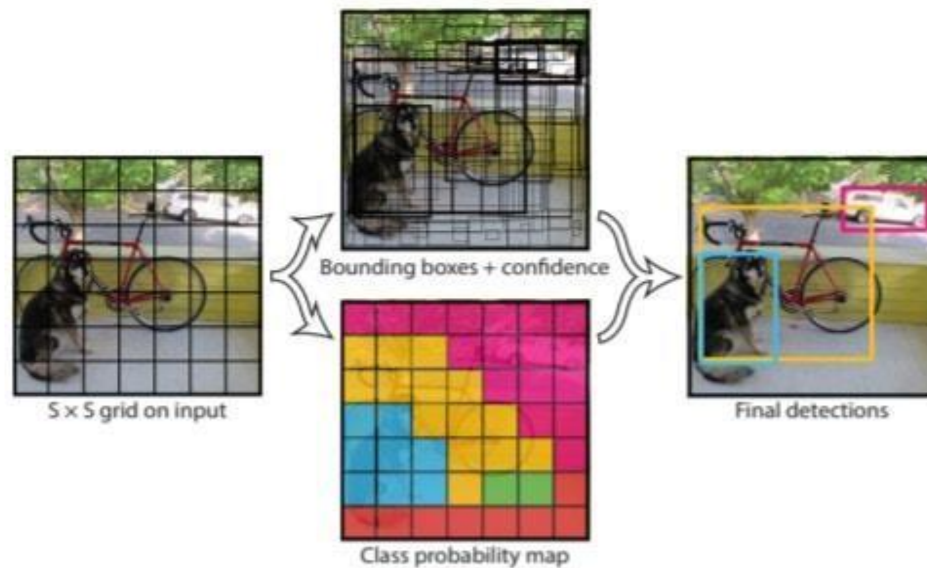


Рисунок 1.12 — Клітинні сітки

Спочатку зображення розбивається на клітинки сітки. Кожна клітинка сітки прогнозує B обмежувальних квадратів і надає їхні оцінки довіри. Комірки передбачають ймовірності класу для встановлення класу кожного об'єкта.

Наприклад, ми можемо помітити щонайменше три класи об’єктів: автомобіль, собака та велосипед. Усі передбачення виконуються одночасно за допомогою однієї згорткової нейронної мережі.

Перетин через об'єднання гарантує, що передбачені обмежувальні прямокутники дорівнюють реальним квадратам об'єктів. Це явище усуває непотрібні обмежувальні рамки, які не відповідають характеристикам об’єктів (наприклад, висота та ширина). Остаточне виявлення буде складатися з унікальних обмежуючих рамок, які ідеально підходять до об’єктів.

Наприклад, автомобіль оточений рожевою обмежувальною рамкою, а велосипед — жовтою. Собаку виділено за допомогою синьої обмежувальної рамки.

Алгоритм YOLO можна застосувати в таких полях:

- Автономне водіння: алгоритм YOLO можна використовувати в автономних автомобілях для виявлення об'єктів навколо автомобілів, таких як транспортні засоби, люди та сигнали паркування. Виявлення об'єктів в автономних автомобілях робиться, щоб уникнути зіткнення, оскільки жоден водій не керує автомобілем.
- Дика природа: цей алгоритм використовується для виявлення різних видів тварин у лісах. Цей тип виявлення використовується рейнджерами дикої природи та журналістами для ідентифікації тварин на відео (як записаних, так і в реальному часі) та зображеннях. Деякі з тварин, яких можна виявити, включають жирафів, слонів і ведмедів.
- Безпека: YOLO також можна використовувати в системах безпеки для забезпечення безпеки в зоні. Припустимо, що людям заборонено проїзд через певну територію з міркувань безпеки. Якщо хтось проходить через заборонену зону, алгоритм YOLO виявить його/її, що вимагатиме від співробітників служби безпеки вжити подальших дій.

Середня точність розраховується як площа під кривою точності та відкликання для набору передбачень.

Відкликання розраховується як відношення загальних прогнозів, зроблених моделлю для класу із загальною кількістю наявних міток для класу.

З іншого боку, точність відноситься до відношення істинних позитивних результатів до загального прогнозу, зробленого моделлю.

Площа під кривою точності та відкликання дає нам середню точність для класу для моделі. Середнє значення цього значення, прийняте для всіх класів, називається середньою точністю (mAP).

При виявленні об'єктів точність і відкликання призначені не для передбачення класів, а для передбачення граничних блоків для вимірювання

ефективності рішення. Значення IoU > 0,5. приймається як позитивний прогноз, тоді як значення IoU < 0,5 є негативним прогнозом.

Відмінності: YOLO, YOLOv2, YOLO9000, YOLOv3, YOLOv4+

YOLOv2 був запропонований для вирішення основних проблем YOLO — виявлення малих об’єктів у групах та точності локалізації.

YOLOv2 збільшує середню середню точність мережі, вводячи пакетну нормалізацію.

Пакетна норма збільшує значення mAP на цілих 2 відсотки.

Набагато більш ефектним доповненням до алгоритму YOLO, запропонованого YOLOv2, було додавання блоків прив'язки. YOLO, як ми знаємо, прогнозує один об’єкт на комірку сітки. Хоча це спрощує вбудовану модель, вона створює проблеми, коли одна клітинка містить більше одного об’єкта, оскільки YOLO може призначити клітині лише один клас.

YOLOv2 позбавляється від цього обмеження, дозволяючи передбачати кілька обмежуючих прямокутників з однієї клітинки. Це досягається завдяки тому, що мережа передбачає 5 обмежуючих прямокутників для кожної клітинки.

Число 5 емпірично отримано як таке, що має хороший компроміс між складністю моделі та продуктивністю прогнозування. DarkNet-19, що містить загалом 19 згорткових шарів і 5 шарів максимального об’єднання, використовується як основа архітектури YOLOv2.

Використовуючи подібну архітектуру мережі, як YOLOv2, YOLO9000 був запропонований як алгоритм для виявлення більшої кількості класів, ніж COCO, оскільки набір даних виявлення об’єктів міг зробити можливим.

Набір даних виявлення об’єктів, на якому навчалися ці моделі (COCO), має лише 80 класів у порівнянні з мережами класифікації, такими як ImageNet, яка має 22 000 класів.

Щоб уможливити виявлення багатьох інших класів, YOLO9000 використовує мітки як із ImageNet, так і з COCO, ефективно об’єднуючи завдання класифікації та виявлення, щоб виконувати лише виявлення.

Оскільки деякі класи COCO можна назвати класами надмножин деяких класів ImageNet, YOLO9000 використовує алгоритм на основі ієрархічної класифікації, натхненний WordNet, де класи та їх підкласи представлені деревоподібним способом.

Хоча YOLO9000 забезпечує нижчу середню точність порівняно з YOLOv2, він здатний виявляти понад 9000 класів, що робить його потужним алгоритмом.

Хоча YOLOv2 — це надшвидка мережа, на сцену також з’явилися різні альтернативи, які пропонують кращу точність, наприклад детектори одиночних пострілів. Хоча вони набагато повільніше, вони випереджають YOLOv2 і YOLO9000 з точки зору точності.

Для покращення YOLO за допомогою сучасних CNN, які використовують залишкові мережі та пропускають з’єднання, був запропонований YOLOv3.

У той час як YOLOv2 використовує DarkNet-19 як архітектуру моделі, YOLOv3 використовує набагато складнішу DarkNet-53 як основу моделі — 106шарову нейронну мережу в комплекті із залишковими блоками та мережами підвищення дискретизації.

Архітектурна новизна YOLOv3 дозволяє йому прогнозувати в 3 різних масштабах, при цьому карти об’єктів витягуються на шарах 82, 94 і 106 для цих передбачень.

Виявляючи ознаки в 3 різних масштабах, YOLOv3 компенсує недоліки YOLOv2 і YOLO, особливо у виявленні менших об’єктів. Завдяки архітектурі, яка дозволяє об’єднати вихідні дані шару з підвищеною дискретизацією з функціями попередніх шарів, дрібнозернисті елементи, які були вилучені, зберігаються, що полегшує виявлення менших об’єктів.

YOLOv3 передбачає лише 3 обмежувальні прямокутники на клітинку (порівняно з 5 в YOLOv2), але він робить три передбачення в різних масштабах, у сумі до 9 прив’язних блоків.

YOLOv4 був запропонований Bochkovskiy et. ін. у 2020 році як удосконалення YOLOv3. Алгоритм досягає найсучасніших результатів із середньою точністю 43,5 %, працюючи зі швидкістю 65 кадрів в секунду на GPU Tesla v100.

Ці результати досягаються шляхом включення комбінації змін в архітектурний дизайн та методи навчання YOLOv3.

YOLOv4 пропонує додавання зважених залишкових зв’язків, перехресної міні-пакетної нормалізації, міжетапних часткових з’єднань, самоконкурентної підготовки та активації Mish як методологічні зміни серед сучасних методів регуляризації та збільшення даних. Автори також пропонують версію YOLOv4 Tiny, яка забезпечує швидше виявлення об’єктів і вищий FPS, роблячи компроміс у точності передбачення.

YOLOv5 — це проект з відкритим вихідним кодом, який складається із сімейства моделей виявлення об’єктів і методів виявлення на основі моделі YOLO, попередньо навченої на наборі даних COCO. Він підтримується Ultralytics і представляє дослідження організації з відкритим вихідним кодом щодо майбутнього робіт Computer Vision.

YOLACT (You Only Look At Coefficients), запропонований Bolya, є застосуванням принципу YOLO для сегментації екземплярів у реальному часі.

Іншими словами, YOLACT пропонує наскрізну згорткову мережу, наприклад сегментацію, яка досягає середньої точності 29,8 при 33,5 FPS на одному Titan Xp, що значно швидше, ніж інші алгоритми сегментації екземплярів.

YOLACT виконує сегментацію екземплярів, генеруючи набір масокпрототипів і коефіцієнтів маски для кожного екземпляра. Лінійна

комбінація двох кроків виконується для створення масок кінцевого екземпляра.

Висновки до розділу 1

Даний розділ був присвячений високорівневому огляду предметної області, що досліджується в даній роботі.

Комп’ютерний зір також відіграє важливу роль у програмах розпізнавання облич, технології, яка дозволяє комп’ютерам узгоджувати зображення облич людей з їхніми особами. Алгоритми комп’ютерного зору визначають риси обличчя на зображеннях і порівнюють їх з базами даних профілів обличчя. Споживчі пристрої використовують розпізнавання облич для автентифікації особи своїх власників. Додатки соціальних мереж використовують розпізнавання облич для виявлення та позначення користувачів. Правоохоронні органи також покладаються на технологію розпізнавання облич, щоб ідентифікувати злочинців у відеоканалах.

Комп’ютерний зір також відіграє важливу роль у доповненій та змішаній реальності, технології, яка дозволяє комп’ютерним пристроям, таким як смартфони, планшети та розумні окуляри, накладати та вбудовувати віртуальні об’єкти на зображення реального світу. Використовуючи комп’ютерний зір, обладнання AR виявляє об’єкти в реальному світі, щоб визначити розташування на дисплеї пристрою для розміщення віртуального об’єкта. Наприклад, алгоритми комп’ютерного зору можуть допомогти додаткам AR виявляти такі площини, як стільниці, стіни та підлоги, що є дуже важливою частиною встановлення глибини та розмірів та розміщення віртуальних об’єктів у фізичному світі.

РОЗДІЛ 2. ДОСЛІДЖЕННЯ ЗАСТОСУВАННЯ ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ ОБРОБКИ БАГАТОПОТОКОВОЇ ВІДЕОІНФОРМАЦІЇ

2.1. Зйомка фотографій із глибиною

На пристроях iOS із задньою подвійною камерою або фронтальною камерою TrueDepth система захоплення може записувати інформацію про глибину. Карта глибини схожа на зображення (рисунок 2.1); однак, замість того, щоб кожен піксель забезпечує колір, він вказує відстань від камери до цієї частини зображення (або в абсолютному вираженні, або відносно інших пікселів на карті глибини).

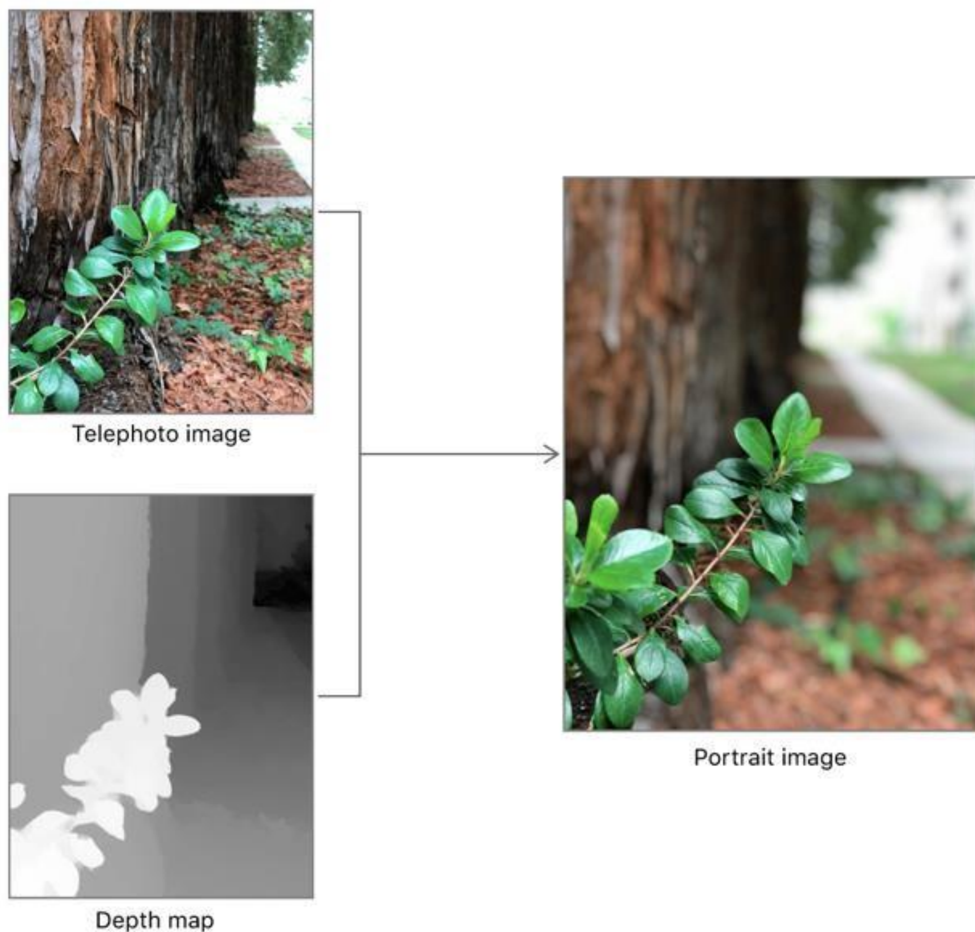


Рисунок 2.1 – Портретний режим з використанням карти глибини

Можливе використання карти глибини разом із фотографією, щоб створити ефекти обробки зображень, які по-різному обробляють передній і задній план фотографії, як-от портретний режим у програмі iOS Camera

(рисунок 2.1). зберігаючи дані про колір і глибину окремо, навіть можливе застосовування та змінювання цих ефектів після зйомки.

Можна додати захоплення глибини до багатьох інших робочих процесів фотозйомки.

2.2. Підготовка до глибинної фотозйомки

Щоб отримати карти глибини, потрібно спочатку вибрати вбудовану камеру InDualCamera або вбудовану камеру InTrueDepthCamera як відеовхід сеансу.

Навіть якщо пристрій iOS має подвійну камеру або камеру TrueDepth, вибір задньої або передньої камери за замовчуванням не вмикає захоплення глибини.

Захоплення глибини також вимагає внутрішньої реконфігурації конвеєра захоплення, ненадовго затримуючи захоплення та перериваючи будь-які поточні захоплення.

Перш ніж зробити першу фотографію глибини, переконайтеся, що конвеєр налаштований належним чином, увімкнене захоплення глибини на об’єкті AVCapturePhotoOutput.

Приклад реалізації режиму захоплення глибини можна побачити на рисунку 2.2.

```
// Select a depth-capable capture device.
guard let videoDevice = AVCaptureDevice.default(.builtInWideAngleCamera,
    for: .video, position: .unspecified)
    else { fatalError("No dual camera.") }
guard let videoDeviceInput = try? AVCaptureDeviceInput(device: videoDevice),
    self.captureSession.canAddInput(videoDeviceInput)
    else { fatalError("Can't add video input.") }
self.captureSession.beginConfiguration()
self.captureSession.addInput(videoDeviceInput)

// Set up photo output for depth data capture.
let photoOutput = AVCapturePhotoOutput()
photoOutput.isDepthDataDeliveryEnabled = photoOutput.isDepthDataDeliverySupported
guard self.captureSession.canAddOutput(photoOutput)
    else { fatalError("Can't add photo output.") }
self.captureSession.addOutput(photoOutput)
self.captureSession.sessionPreset = .photo
self.captureSession.commitConfiguration()
```

Рисунок 2.2 – Режим захоплення глибини через AVCapturePhotoOutput

Коли вихідні фотографії будуть готові до зйомки глибини, ви можете попросити, щоб будь-які окремі фотографії зняли карту глибини разом із кольоровим зображенням.

Створений об'єкт AVCapturePhotoSettings, вибравши формат кольорового зображення. Потім увімкнення захоплення глибини та вихід глибини (та будь-які інші параметри, потрібні для цієї фотографії) і виклик

```
let photoSettings = AVCapturePhotoSettings(format: [AVVideoCodecKey: AVVideoCodecTy
photoSettings.isDepthDataDeliveryEnabled = photoOutput.isDepthDataDeliverySupportec

// Shoot the photo, using a custom class to handle capture delegate callbacks.
let captureProcessor = PhotoCaptureProcessor()
photoOutput.capturePhoto(with: photoSettings, delegate: captureProcessor)
```

метода capturePhoto(with:delegate:) як показано на рисунку 2.3.

Рисунок 2.3 – Зйомка з глибиною

Після зйомки вихідні фотографії викликають метод Photo Output (did Finish Processing Photo:error:) делегата, надаючи фотографію та дані про глибину як об'єкт AVCapturePhoto.

Якщо планується негайно використати зняті дані про глибину, наприклад, для відображення попереднього перегляду ефекту обробки зображень на основі глибини, ви можете знайти їх у властивості deepData фотооб'єкта.

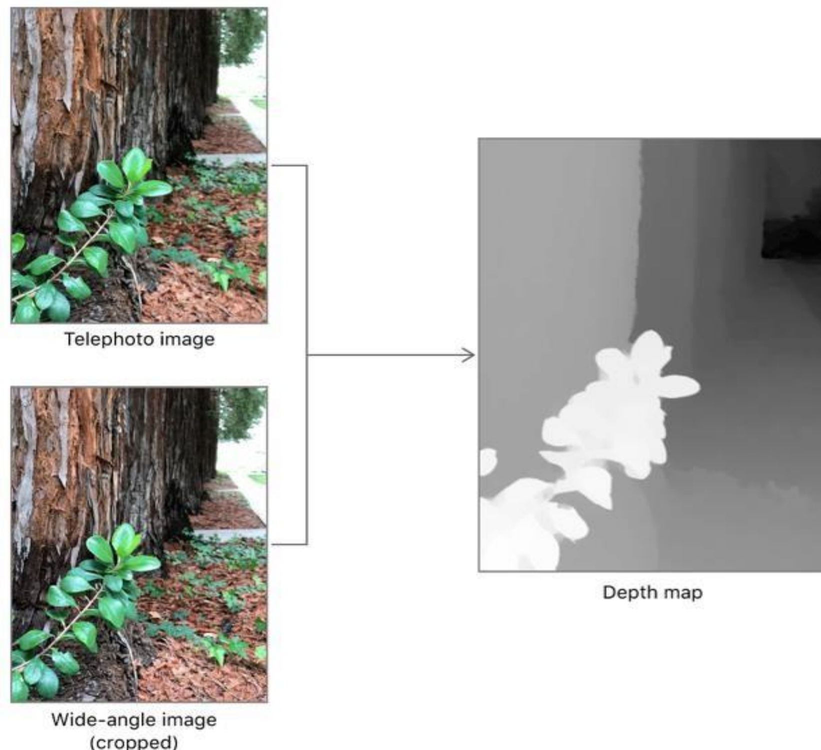
В іншому випадку вихідні да ні зйомки вбудовують дані про глибину та пов'язані з глибиною метадані, коли використовується метод file Data Representation для створення даних файлу для збереження фотографії.

Якщо додати отриманий файл до бібліотеки «Фотографії», інші програми (включаючи системну програму «Фотографії») автоматично розпізнають дані про глибину і можуть застосовувати ефекти обробки зображень на основі глибини.

Коли ввімкнете зйомку глибини за допомогою подвійної задньої камери на сумісних пристроях (див. Довідник щодо сумісності пристроїв iOS), система знімає зображення за допомогою обох камер.

Оскільки дві паралельні камери розташовані на невеликій відстані один від одного на задній панелі пристрою, подібні ознаки, виявлені на обох зображеннях, показують зсув паралакса: об’єкти, які знаходяться ближче до камери, зміщуються на більшу відстань між двома зображеннями.

Система захоплення використовує цю різницю або невідповідність, щоб визначити відносні відстані від камери до об’єктів на зображенні, як



показано нижче на рисунку 2.4.

Рисунок 2.4 – Зйомка з глибиною

Кожна точка на карті глибини, знята пристроєм з подвійною камерою, вимірює диспропорцію в одиницях 1/метр і пропонує відносну точність AVDepthData.Accuracy. Тобто окрема точка не є хорошою оцінкою реальної відстані, але різниця між точками є достатньою для використання для ефектів обробки зображень на основі глибини. [4]

Камера TrueDepth проектує візерунок інфрачервоного світла перед камерою та зображує цей візерунок за допомогою інфрачервоної камери. Спостерігаючи, як шаблон спотворюється об'єктами в сцені, система захоплення може обчислити відстань від камери до кожної точки на зображенні.

Камера TrueDepth створює карти невідповідності за замовчуванням, тому отримані дані глибини подібні до даних, отриманих пристроєм з подвійною камерою. Однак, на відміну від пристрою з подвійною камерою, камера TrueDepth може безпосередньо вимірювати глибину (у метрах) з абсолютною точністю AVDepthData.Accuracy.

Щоб захопити глибину замість диспропорції, встановіть ActiveDepthDataFormat пристрою захоплення перед початком сеансу

```
// Select a depth (not disparity) format that works with the active color format.
let availableFormats = captureDevice.activeFormat.supportedDepthDataFormats

let depthFormat = availableFormats.filter { format in
    let pixelFormatType =
        CMFormatDescriptionGetMediaSubType(format.formatDescription)

    return (pixelFormatType == kCVPixelFormatType_DepthFloat16 ||
            pixelFormatType == kCVPixelFormatType_DepthFloat32)
}.first

// Set the capture device to use that depth format.
captureSession.beginConfiguration()
captureDevice.activeDepthDataFormat = depthFormat
captureSession.commitConfiguration()
```

захоплення, як показано на рисунку 2.5.

Рисунок 2.5 – Налаштування камери TrueDepth

2.3 Зйомка глибини за допомогою камери LiDAR

AVFoundation представив збір даних про глибину для фотографій і відео в iOS 11. Дані, які він надає, підходять для багатьох програм, але можуть не відповідати потребам тих, яким потрібна більша точність глибини.

Починаючи з iOS 15.4, ви можете отримати доступ до камери LiDAR на допоміжному обладнанні, яке пропонує високоточні дані про глибину, які підходять для таких випадків, як сканування та вимірювання кімнат.

Зразок програми показує, як захопити та відтворити дані про глибину з камери LiDAR. Він запускається в режимі потокового передавання, який демонструє, як записувати синхронізоване відео та дані про глибину.

Натискання кнопки камери у верхньому лівому куті екрана перемикає програму в режим фото, який ілюструє, як робити фотографії з даними глибини. В обох режимах програма надає кілька візуалізацій глибини та зображень на основі металу.

Зразок класу CameraController програми надає код, який налаштовує та керує сеансом захоплення, а також обробляє доставку нового відео та даних про глибину. Він починає свою конфігурацію з отримання камери LiDAR (рисунок 2.6).

Він викликає метод класу за замовчуванням (_:for:position:) пристрою захоплення, як показано нижче, передаючи йому новий тип пристрою built In

```
// Look up the LiDAR camera.
guard let device = AVCaptureDevice.default(.builtInLiDARDepthCamera, for: .video,
    throw ConfigurationError.lidarDeviceUnavailable
}
```

Li DAR Depth Camera, доступний в iOS 15.4 і новіших версіях.

Рисунок 2.6 – Отримання камери LiDAR

Після отримання пристрою програма налаштовує його на певний формат відео та глибини. Він запитує в пристрої підтримувані формати та знаходить найкращий неколірний, повнодіапазонний колірний формат YUV, який відповідає бажаній ширині зразка програми та підтримує захоплення глибини.

Нарешті, він встановлює активні формати на пристрої, як показано на

```
// Find a match that outputs video data in the format the app's custom Metal view
guard let format = (device.formats.last { format in
    format.formatDescription.dimensions.width == preferredWidthResolution &&
    format.formatDescription.mediaSubType.rawValue == kCVPixelFormatType_420YpCbC
    !format.isVideoBinned &&
    !format.supportedDepthDataFormats.isEmpty
}) else {
    throw ConfigurationError.requiredFormatUnavailable
}

// Find a match that outputs depth data in the format the app's custom Metal view
guard let depthFormat = (format.supportedDepthDataFormats.last { depthFormat in
    depthFormat.formatDescription.mediaSubType.rawValue == kCVPixelFormatType_Dep
}) else {
    throw ConfigurationError.requiredFormatUnavailable
}

// Begin the device configuration.
try device.lockForConfiguration()

// Configure the device and depth formats.
device.activeFormat = format
device.activeDepthDataFormat = depthFormat

// Finish the device configuration.
device.unlockForConfiguration()
```

рисунку 2.7.

Рисунок 2.7 – Встановлення активних форматів

Додаток працює в режимі потокового або фото. Щоб увімкнути потоковий вихід, він створює екземпляр AVCaptureVideoDataOutput і AVCaptureDepthDataOutput для захоплення буферів зразків відео та даних про глибину відповідно. Він налаштовує їх, як показано в наступному прикладі (рисунок 2.8).

```
// Create an object to output video sample buffers.
videoDataOutput = AVCaptureVideoDataOutput()
captureSession.addOutput(videoDataOutput)

// Create an object to output depth data.
depthDataOutput = AVCaptureDepthDataOutput()
depthDataOutput.isFilteringEnabled = isFilteringEnabled
captureSession.addOutput(depthDataOutput)

// Create an object to synchronize the delivery of depth and video data.
outputVideoSync = AVCaptureDataOutputSynchronizer(dataOutputs: [depthDataOutput
outputVideoSync.setDelegate(self, queue: videoQueue)
```

Рисунок 2.8 – Захоплення буферів AVCaptureDepthDataOutput

Оскільки потік даних відео та глибини від окремих вихідних об’єктів, зразок використовує AVCaptureDataOutputSynchronizer для синхронізації доставки з обох виходів до одного зворотного виклику.

Клас CameraController приймає протокол AV Capture Data Output Synchronizer Delegate синхронізатора і відповідає на доставку нового відео та даних про глибину.

Для обробки фотографій програма також створює екземпляр AV Capture Photo Output. Він оптимізує вихід для високоякісного захоплення та додає вихід до сеансу захоплення, як показано нижче на рисунку 2.9.

```
// Create an object to output photos.
photoOutput = AVCapturePhotoOutput()
photoOutput.maxPhotoQualityPrioritization = .quality
captureSession.addOutput(photoOutput)

// Enable delivery of depth data after adding the output to the capture session.
photoOutput.isDepthDataDeliveryEnabled = true
```

Рисунок 2.9 – Створення AVCapturePhotoOutput

Після того, як він додає вихідні дані до сеансу, він дозволяє доставку даних про глибину, що налаштовує конвеєр захоплення належним чином. Він може ввімкнути доставку глибини лише після додавання виводу до сеансу захоплення, оскільки вихід повинен знати, чи конвеєр налаштований для його доставки. Коли входи та виходи сеансу захоплення налаштовані

відповідно до потреб, програма готова почати запис даних. Програма запускається в режимі потокового передавання, який використовує вихідні дані відео та глибини та AVCaptureDataOutputSynchronizer для синхронізації доставки їхніх даних. Додаток приймає протокол делегата синхронізатора і реалізує його метод dataOutputSynchronizer(_:didOutput:) для обробки

```
func dataOutputSynchronizer(_ synchronizer: AVCaptureDataOutputSynchronizer,
                            didOutput synchronizedDataCollection: AVCaptureSynchronizedDataCollection) {
    // Retrieve the synchronized depth and sample buffer container objects.
    guard let syncedDepthData = synchronizedDataCollection.synchronizedData(for: depthDataOutput) as? AVCaptureSynchronizedDepthData,
          let syncedVideoData = synchronizedDataCollection.synchronizedData(for: videoDataOutput) as? AVCaptureSynchronizedSampleBuffer

    guard let pixelBuffer = syncedVideoData.sampleBuffer.imageBuffer,
          let cameraCalibrationData = syncedDepthData.depthData.cameraCalibrationData else { return }

    // Package the captured data.
    let data = CameraCapturedData(depth: syncedDepthData.depthData.depthDataMap.texture(withFormat: .r16Float, planeIndex: 0, addToCach
                                  colorY: pixelBuffer.texture(withFormat: .r8Unorm, planeIndex: 0, addToCache: textureCache),
                                  colorCbCr: pixelBuffer.texture(withFormat: .rg8Unorm, planeIndex: 1, addToCache: textureCache),
                                  cameraIntrinsics: cameraCalibrationData.intrinsicMatrix,
                                  cameraReferenceDimensions: cameraCalibrationData.intrinsicMatrixReferenceDimensions)

    delegate?.onNewData(capturedData: data)
}
```

доставки, як показано нижче на рисунку 2.10.

Рисунок 2.10 – Режим потокового передавання

Програма отримує об’єкти-контейнери, які зберігають синхронізовані дані з AVCaptureSynchronizedDataCollection. Потім він розгортає основний буфер відеопікселів і дані про глибину та упаковує їх для відображення Metal Views додатка. Натискання кнопки камери програми у верхньому лівому куті інтерфейсу користувача перемикає програму в режим фотозйомки. Коли це відбувається, програма викликає свій метод capturePhoto(), який створює об’єкт налаштувань фотографії, запитує для нього передачу глибини та ініціює зйомку фотографії, як показано нижче (рисунок 2.11).

```
func capturePhoto() {
    var photoSettings: AVCapturePhotoSettings
    if  photoOutput.availablePhotoPixelFormatTypes.contains(kCVPixelFormatType_420YpCbCr8BiPlanarFullRange) {
        photoSettings = AVCapturePhotoSettings(format: [
            kCVPixelBufferPixelFormatTypeKey as String: kCVPixelFormatType_420YpCbCr8BiPlanarFullRange
        ])
    } else {
        photoSettings = AVCapturePhotoSettings()
    }

    // Capture depth data with this photo capture.
    photoSettings.isDepthDataDeliveryEnabled = true
    photoOutput.capturePhoto(with: photoSettings, delegate: self)
}
```

Рисунок 2.11 – Режим зйомки

Коли фреймворк завершує зйомку фотографії, він викликає метод делегата виводу фотографії та передає йому об’єкт AVCapturePhoto, який містить дані зображення та глибини.

Зразок отримує дані з фотографії, зупиняє потік, доки користувач не повернеться в режим потокового передавання, і, подібно до випадку з відео, пакує отримані дані для доставки на рівень інтерфейсу користувача програми

```
func photoOutput(_ output: AVCapturePhotoOutput, didFinishProcessingPhoto photo: AVCapturePhoto, error: Error?) {

    // Retrieve the image and depth data.
    guard let pixelBuffer = photo.pixelBuffer,
          let depthData = photo.depthData,
          let cameraCalibrationData = depthData.cameraCalibrationData else { return }

    // Stop the stream until the user returns to streaming mode.
    stopStream()

    // Convert the depth data to the expected format.
    let convertedDepth = depthData.converting(toDepthDataType: kCVPixelFormatType_DepthFloat16)

    // Package the captured data.
    let data = CameraCapturedData(depth: convertedDepth.depthDataMap.texture(withFormat: .r16Float, planeIndex: 0, addToCache: 
                                  colorY: pixelBuffer.texture(withFormat: .r8Unorm, planeIndex: 0, addToCache: textureCache),
                                  colorCbCr: pixelBuffer.texture(withFormat: .rg8Unorm, planeIndex: 1, addToCache: textureCache
                                  cameraIntrinsics: cameraCalibrationData.intrinsicMatrix,
                                  cameraReferenceDimensions: cameraCalibrationData.intrinsicMatrixReferenceDimensions)

    delegate?.onNewPhotoData(capturedData: data)
}
```

(рисунок 2.12).

Рисунок 2.12 – Передача даних до інтерфейсу користувача

2.4. Створення 3D-об’єктів з фотографій

Можливо створювати тривимірні об’єкти з фотографій, використовуючи процес, який називається фотограмметрією.

Надання RealityKit Object Capture серію добре освітлених фотографій, зроблених з різних ракурсів. (рисунок 2.13) аналізується область перекриття між різними зображеннями, щоб узгодити орієнтири, а потім створюється 3D-модель сфотографованого об’єкта.

Щоб створити найкраще 3D-подання в процесі створення об’єктів, потрібно надати RealityKit високоякісні фотографії з високою роздільною здатністю, які не містять жорстких тіней або яскравих відблисків.

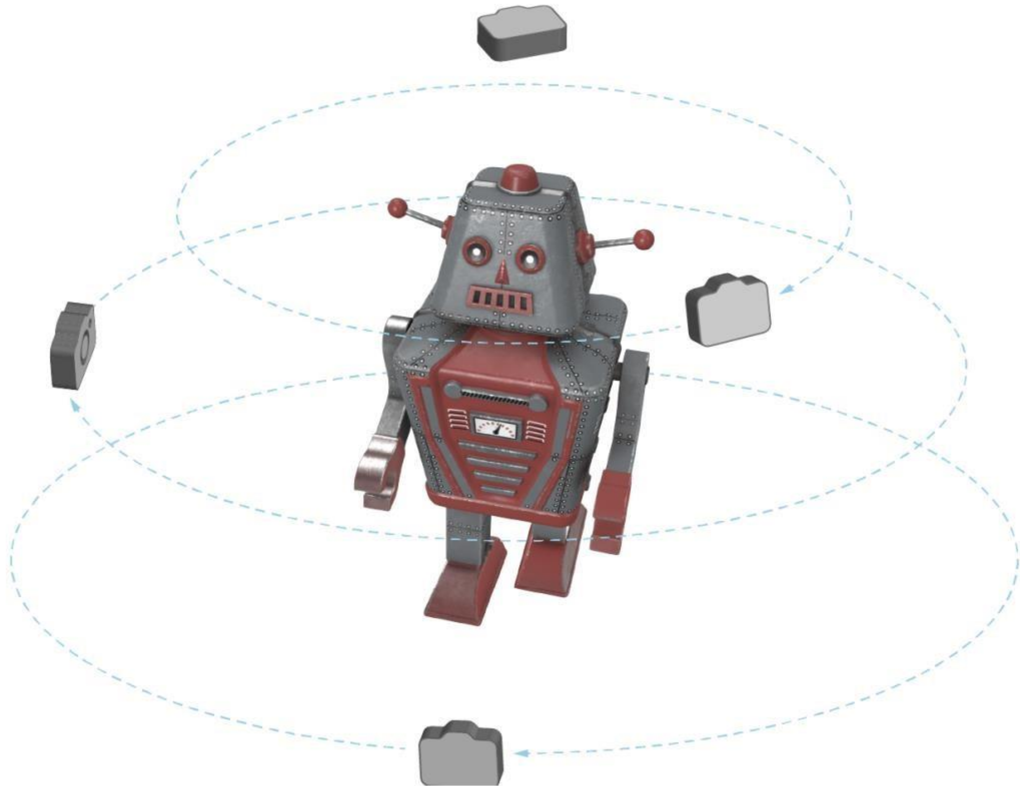


Рисунок 2.13 – Процес фотозйомки об’єкта з різних ракурсів

Потрібно обирати статичні об’єкти, які не згинаються і не деформуються під час фотографування. Можливо переміщати об’єкт між знімками, щоб сфотографувати всі сторони, але м’який, шарнірний або гнучкий об’єкт, який змінює форму під час його переміщення, може поставити під загрозу здатність RealityKit зіставляти орієнтири між різними зображеннями, що може призвести до невдачі або спровокувати зйомку об’єкта. неякісні результати.

Потрібно уникати дуже тонких в одному вимірі об’єктів, які мають високу відбивну здатність, прозорі або напівпрозорі. Крім того, об’єкти, які є одноколірними або мають дуже гладку поверхню, можуть не надавати достатньо даних, необхідних для алгоритму створення об’єктів для побудови тривимірної форми.

Можете малювати або малювати на поверхні об’єкта, щоб додати колір або текстуру, а потім зіставити вихідний колір або текстуру на створеному 3D-об’єкті з інспектором матеріалів у засобі перегляду 3D-моделей Xcode або інспектором властивостей у Reality Composer.

Можете підійти до фотографування для створення об’єкта двома способами: або переміщати камеру навколо об’єкта, фотографуючи під різними кутами на різній висоті, або покласти об’єкт на поворотний стіл і повертати об’єкт під час фотографування. [5]

Використовуючи програвач, робіть знімки перед добре освітленим однотонним фоном, щоб мінімізувати зайві дані зображення, які можуть заважати процесу створення об’єктів, і використовуйте штатив, якщо він доступний. Якщо об’єкт стоїть по різні боки під час фотозйомки на поворотному столі, можна охопити весь об’єкт без щілин і отворів. Кількість зображень, необхідних RealityKit для створення точного 3D-уявлення, залежить від складності та розміру об’єкта, але сусідні знімки повинні мати значне перекриття. Розташуйте послідовні зображення так, щоб вони перекривали 70% або більше.

Все, що менше ніж на 50% перекривається між сусідніми знімками, і процес створення об’єкта може бути невдалим або призвести до низької якості відтворення.

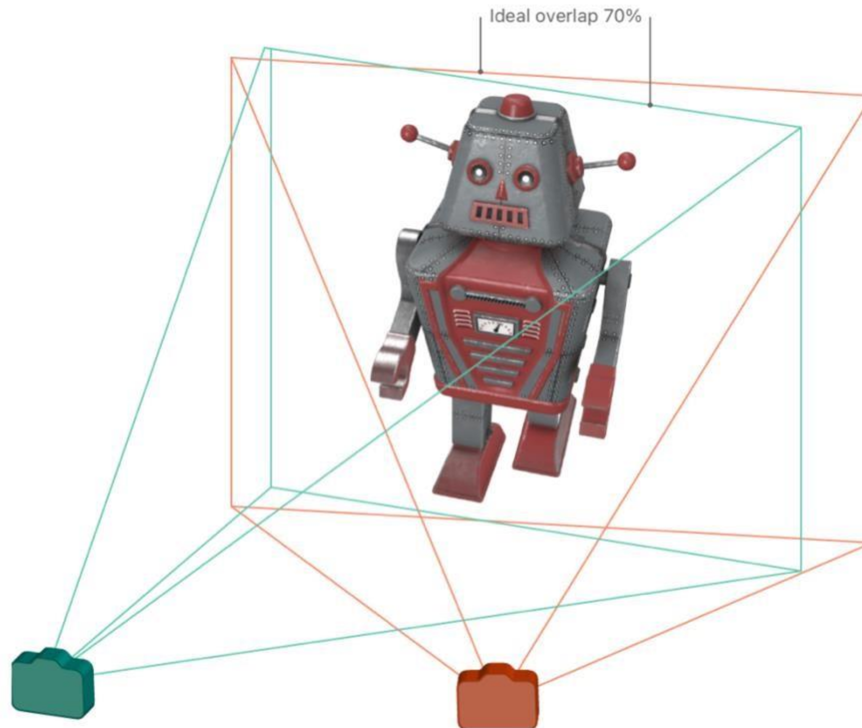


Рисунок 2.14 – Процес фотозйомки об’єкта з різних ракурсів

Створення об’єктів RealityKit приймає зображення, зроблені будь-якою цифровою камерою, включаючи камери на iPhone або iPad, DSLR або бездзеркальну камеру або навіть дрон з камерою. Якщо вихідні зображення містять дані про глибину, RealityKit використовує їх для обчислення реального розміру відсканованого об’єкта. RealityKit також може створювати об’єкти із зображень без даних про глибину, але, можливо, вам доведеться масштабувати об’єкт, коли розміщуєте його у сцені AR. Щоб отримати додаткові відомості про зйомку даних про глибину зображення, див. Зйомка фотографій із глибиною. [10]

Розташуйте об’єкт так, щоб він заповнював якомога більшу частину кадру камери, не виключаючи та не відрізаючи жодної частини. Використовуйте досить вузьке налаштування діафрагми, щоб зберегти чіткий фокус. Знімайте з найвищою роздільною здатністю, яку підтримує ваша камера, і, якщо можливо, використовуйте формат RAW. [6]

Зйомка з вузькою діафрагмою вимагає більше світла, але надає вашим фотографіям більшу глибину різкості, збільшуючи площу об’єкта, який знаходиться у фокусі. Якщо у вас недостатньо світла, щоб використовувати

вузьку діафрагму, подумайте про збільшення значення ISO вашої камери або додайте більше світла.

В умовах слабкого освітлення камери з автофокусом можуть мати труднощі з пошуком та підтримкою фокусування. Фокусуйтеся вручну, якщо навколишнього світла недостатньо для блокування фокусу, і поставте камеру на штатив, щоб переконатися, що ви тримаєте її рівно. Використовуйте дистанційний тригер, наприклад додаток Apple Watch Camera Remote, щоб переконатися, що камера не рухається або не тремтить, коли ви натискаєте кнопку, щоб зробити фотографію.

По можливості використовуйте розсіяне освітлення. Жорстке світло, як-от спалах камери, пряме сонячне світло або оголена лампочка, може спричинити проблеми під час створення об’єкта. Цей тип освітлення відкидає жорсткі тіні, які можуть заплутати алгоритм фотограмметрії. Замість цього відбивайте світло від рефлектора, стіни чи стелі або помістіть між джерелом світла та об’єктом розсіювальний матеріал, наприклад абажур або тонку білу тканину. Також можна використовувати світломодифікатори або легкий намет, призначений для фотографування об’єктів.

Під час фотографування об’єкта на відкритому повітрі, по можливості не допускайте попадання сонця на зображення. Знімайте опівдні, щоб сонце було достатньо високо в небі, щоб його не було видно на жодному з ваших зображень, або знімайте в похмурий день.

Щоб гарантувати, що RealityKit може порівнювати орієнтири між фотографіями, що перекриваються, підтримуйте налаштування камери якомога послідовними від кадру до знімка. Якщо можливо, не змінюйте жодних налаштувань камери під час зйомки, зокрема фокусної відстані (масштабування), діафрагми, витримки або ISO.

Якщо ваша камера пропонує ручне керування будь-якими налаштуваннями, установіть ці значення, щоб вони були однаковими між знімками. Під час зйомки камерою на iPhone або iPad ви можете заблокувати

налаштування експозиції та фокусування, утримуючи і утримуючи вікно перегляду зображення, доки у верхній частині екрана не з'явиться «Блокування AE/AF».

Крім того, маскування об'єктів у фоновому режимі, щоб зображення містило лише об'єкт, який ви знімаєте, видаляє непотрібні дані, які можуть заплутати алгоритм фотограмметрії.

Бувають випадки, коли ви не можете зробити зображення в ідеальних умовах, наприклад, коли фотографуєте великий об'єкт у людному місці. Налаштуйте недоліки зображення за допомогою редактора зображень. Змініть контраст, яскравість, різкість або експозицію фотографії, щоб компенсувати проблеми з оригінальною фотографією.

2.5. Багатопоточність в macOS

Swift має вбудовану підтримку структурованого написання асинхронного та паралельного коду. Асинхронний код можна призупинити та відновити пізніше, хоча одночасно виконується лише одна частина програми. Призупинення та відновлення коду у вашій програмі дозволяє їй продовжувати виконувати короткострокові операції, як-от оновлення інтерфейсу користувача, одночасно продовжуючи працювати над тривалими операціями, такими як отримання даних через мережу або аналіз файлів. Паралельний код означає, що кілька фрагментів коду виконуються одночасно — наприклад, комп'ютер із чотирьохядерним процесором може одночасно виконувати чотири фрагменти коду, при цьому кожне ядро виконує одне із завдань. Програма, яка використовує паралельний та асинхронний код, виконує декілька операцій одночасно; він призупиняє операції, які очікують на зовнішню систему, і полегшує написання цього коду безпечним способом.

Додаткова гнучкість планування з паралельного або асинхронного коду також супроводжується підвищеною складністю. Swift дозволяє висловити свої наміри таким чином, щоб уможливити певну перевірку під час компіляції, наприклад, ви можете використовувати акторів для

безпечного доступу до змінюваного стану. Однак додавання паралельності до повільного чи помилкового коду не є гарантією того, що він стане швидким чи правильним. Насправді, додавання паралельності може навіть ускладнити налагодження вашого коду. Однак використання підтримки на рівні мови Swift для паралельності в коді, який має бути одночасним, означає, що Swift може допомогти вам уловити проблеми під час компіляції.

В іншій частині цієї глави використовується термін паралельність для позначення цієї загальної комбінації асинхронного та паралельного коду.

Якщо ви раніше писали одночасний код, можливо, ви звикли працювати з потоками. Модель паралельності в Swift побудована на основі потоків, але ви не взаємодієте з ними безпосередньо. Асинхронна функція в Swift може відмовитися від потоку, в якому вона працює, що дозволяє іншій асинхронній функції виконуватися в цьому потоці, поки перша функція заблокована.

Хоча можна писати одночасний код без підтримки мови Swift, цей код, як правило, важче читати. Наприклад, код, що наведений на рисунку 2.15 завантажує список назв фотографій, завантажує першу фотографію в цьому списку та показує це фото користувачеві.

```
1  listPhotos(inGallery: "Summer Vacation") { photoNames in
2      let sortedNames = photoNames.sorted()
3      let name = sortedNames[0]
4      downloadPhoto(named: name) { photo in
5          show(photo)
6      }
7  }
```

Рисунок 2.15 – Програмний код без використання багатопотоковості

Навіть у цьому простому випадку, оскільки код має бути написаний як серія обробників завершення, ви в кінцевому підсумку пишете вкладені

замикання. У цьому стилі більш складний код із глибокою вкладеністю може швидко стати громіздким.

Асинхронна функція або асинхронний метод — це особливий тип функції або методу, який можна призупинити, поки він виконується. Це на відміну від звичайних, синхронних функцій і методів, які або виконуються до завершення, видають помилку або ніколи не повертаються. Асинхронна функція або метод все ще виконує одну з цих трьох речей, але також може призупинятися в середині, коли чогось чекає. Усередині тіла асинхронної функції або методу ви позначаєте кожне з цих місць, де виконання може бути призупинено.

Щоб вказати, що функція або метод є асинхронними, ви пишете ключове слово async у його декларації після параметрів, подібно до того, як ви використовуєте throws для позначення функції викидання. Якщо функція або метод повертає значення, ви пишете async перед стрілкою повернення (->). Наприклад, можна отримати назви фотографій у галереї так, як показано на рисунку 2.16.

```
1  func listPhotos(inGallery name: String) async -> [String] {
2      let result = // ... some asynchronous networking code ...
3      return result
4  }
```

Рисунок 2.16 – Асинхронна обробка данизх

Для функції або методу, який одночасно є асинхронним і кидає, ви пишете асинхронно перед виклаками.

Під час виклику асинхронного методу виконання призупиняється, доки цей метод не повернеться. Ви пишете очікування перед дзвінком, щоб позначити можливу точку призупинення. Це як написати спробу під час виклику функції викидання, щоб позначити можливу зміну ходу програми,

якщо є помилка. Усередині асинхронного методу потік виконання призупиняється лише тоді, коли ви викликаєте інший асинхронний метод — призупинення ніколи не буває неявним або випереджальним — що означає, що кожна можлива точка призупинення позначена await.

Наприклад, наведений на рисунку 2.17 код отримує назви всіх зображень у галереї, а потім показує перше зображення.

```
1  let photoNames = await listPhotos(inGallery: "Summer Vacation")
2  let sortedNames = photoNames.sorted()
3  let name = sortedNames[0]
4  let photo = await downloadPhoto(named: name)
5  show(photo)
```

Рисунок 2.17 –Клієнтський код багатопотокового завантаження зображень

Оскільки функції listPhotos(inGallery:) і downloadPhoto(named:) повинні виконувати мережеві запити, їх виконання може зайняти відносно багато часу. Зробивши їх обидва асинхронними, написавши async перед стрілкою повернення, решта коду програми продовжує працювати, поки цей код чекає, поки зображення буде готове.

Щоб зрозуміти сумісний характер наведеного вище прикладу, ось один можливий порядок виконання:

Код починає виконуватися з першого рядка і виконується до першого await. Він викликає функцію listPhotos(inGallery:) і призупиняє виконання, поки очікує повернення цієї функції.

Поки виконання цього коду призупинено, виконується деякий інший одночасний код у тій же програмі. Наприклад, можливо, тривала фонова задача продовжує оновлювати список нових фотогалерей. Цей код також працює до наступної точки призупинення, позначеної як await, або до її завершення.

Після повернення listPhotos(inGallery:) цей код продовжує виконуватися, починаючи з цього моменту. Він призначає значення, яке було повернуто до photoNames.

Рядки, що визначають sortedNames та name, є звичайним, синхронним кодом. Оскільки на цих лініях нічого не позначено, то немає жодних можливих точок підвіски.

Наступне очікування означає виклик функції downloadPhoto(named:). Цей код знову призупиняє виконання, доки ця функція не повернеться, надаючи можливість запустити інший паралельний код.

Після повернення downloadPhoto(named:) його повернуте значення призначається фотографії, а потім передається як аргумент під час виклику show(_:).

Можливі точки призупинення у вашому коді, позначені як await, вказують на те, що поточний фрагмент коду може призупинити виконання під час очікування повернення асинхронної функції або методу. Це також називається вихідним потоком, оскільки за лаштунками Swift призупиняє виконання вашого коду в поточному потоці і замість цього запускає інший код у цьому потоці. Оскільки код з await повинен мати можливість призупинити виконання, лише певні місця у вашій програмі можуть викликати асинхронні функції або методи:

- код в тілі асинхронної функції, методу або властивості;
- код у статичному методі main() структури, класу або перерахування, позначених символом @main;
- код у неструктурованому дочірньому завданні

Метод Task.sleep(nanoseconds:) корисний під час написання простого коду, щоб дізнатися, як працює паралельність. Цей метод нічого не робить, але чекає принаймні задану кількість наносекунд, перш ніж повернеться. Ось

```
1  func listPhotos(inGallery name: String) async throws -> [String] {
2      try await Task.sleep(nanoseconds: 2 * 1_000_000_000)  // Two
     seconds
3      return ["IMG001", "IMG99", "IMG0404"]
4  }
```

версія функції listPhotos(inGallery:), яка використовує режим сну (наносекунд:) для імітації очікування операції мережі (рисунок 2.18).

Рисунок 2.18 – Режим сну при роботі з багатопотоковістю

Функція listPhotos(inGallery:) у попередньому розділі асинхронно повертає весь масив одразу після того, як усі елементи масиву будуть готові. Інший підхід — очікування по одному елементу колекції за раз за допомогою асинхронної послідовності. На рисунку 2.19 показано, як виглядає ітерація

```
1  import Foundation
2
3  let handle = FileHandle.standardInput
4  for try await line in handle.bytes.lines {
5      print(line)
6  }
```

асинхронної послідовності.

Рисунок 2.19 – Одна ітерація асинхронної послідовності

Замість використання звичайного циклу for-in у наведеному вище прикладі записується для з await після нього. Як і коли ви викликаєте асинхронну функцію або метод, запис await вказує на можливу точку призупинення. Цикл for-await-in потенційно призупиняє виконання на початку кожної ітерації, коли очікує доступу наступного елемента.

Так само, як ви можете використовувати власні типи в циклі for-in, додавши відповідність до протоколу Sequence, ви можете використовувати власні типи в циклі for-await-in, додавши відповідність до протоколу AsyncSequence.

Виклик асинхронної функції з await запускає лише один фрагмент коду за раз. Поки виконується асинхронний код, абонент чекає завершення

```
1  let firstPhoto = await downloadPhoto(named: photoNames[0])
2  let secondPhoto = await downloadPhoto(named: photoNames[1])
3  let thirdPhoto = await downloadPhoto(named: photoNames[2])
4
5  let photos = [firstPhoto, secondPhoto, thirdPhoto]
6  show(photos)
```

цього коду, перш ніж перейти до виконання наступного рядка коду. Наприклад, щоб отримати перші три фотографії з галереї, ви можете дочекатися трьох викликів функції downloadPhoto(named:) як показано на рисунку 2.20.

Рисунок 2.20 – Багатопотокове послідовне завантаження трьох зображень

Цей підхід має важливий недолік: хоча завантаження є асинхронним і дозволяє виконувати інші роботи під час його виконання, одночасно виконується лише один виклик downloadPhoto(named:). Кожна фотографія завантажується повністю, перш ніж почнеться завантаження наступної. Однак для цих операцій немає необхідності чекати — кожну фотографію можна завантажувати окремо або навіть одночасно.

Щоб викликати асинхронну функцію і дозволити їй працювати паралельно з кодом навколо неї, напишіть async перед let, коли ви визначаєте константу, а потім пишіть await щоразу, коли ви використовуєте константу

```
1  async let firstPhoto = downloadPhoto(named: photoNames[0])
2  async let secondPhoto = downloadPhoto(named: photoNames[1])
3  async let thirdPhoto = downloadPhoto(named: photoNames[2])
4
5  let photos = await [firstPhoto, secondPhoto, thirdPhoto]
6  show(photos)
```

(рисунок 2.21).

Рисунок 2.21 – Багатопотокове паралельне завантаження трьох зображень

У цьому прикладі всі три виклики downloadPhoto(named:) починаються, не чекаючи завершення попереднього. Якщо системних ресурсів достатньо, вони можуть працювати одночасно. Жоден з цих викликів функцій не позначений символом await, оскільки код не призупиняється, щоб чекати результату функції. Натомість виконання триває до тих пір, поки не буде визначено рядок, де визначено фотографії — на цьому етапі програмі потрібні результати цих асинхронних викликів, тому ви

пишете await, щоб призупинити виконання, доки всі три фотографії не закінчать завантаження. [9]

Викликати асинхронні функції з await, коли код у наступних рядках залежить від результату цієї функції. Це створює роботу, яка виконується послідовно.

Викликайте асинхронні функції за допомогою async-let, коли результат вам не потрібен пізніше у вашому коді. Це створює роботу, яку можна виконувати паралельно.

І await, і async-let дозволяють іншим кодам виконуватися, поки вони призупинені.

В обох випадках ви позначаєте можливу точку призупинення за допомогою await, щоб вказати, що виконання буде призупинено, якщо необхідно, доки не повернеться асинхронна функція. Також можливо змішувати обидва ці підходи в одному коді.

Завдання — це одиниця роботи, яку можна виконувати асинхронно як частину програми. Весь асинхронний код виконується як частина певного завдання. Синтаксис async-let, описаний у попередньому розділі, створює для вас дочірнє завдання. Ви також можете створити групу завдань і додати дочірні завдання до цієї групи, що дає вам більше контролю над пріоритетом і скасуванням, а також дозволяє створювати динамічну кількість завдань.

Завдання розташовані в ієрархії. Кожне завдання в групі завдань має те саме батьківське завдання, і кожне завдання може мати дочірні завдання. Через явний зв’язок між завданнями та групами завдань цей підхід називається структурованою паралельністю. Хоча ви берете на себе частину відповідальності за правильність, явні зв’язки між завданнями «батько-дочірня» дозволяють Swift обробляти деякі дії, як-от поширення скасування

```
1  await withTaskGroup(of: Data.self) { taskGroup in
2      let photoNames = await listPhotos(inGallery: "Summer Vacation")
3      for name in photoNames {
4          taskGroup.addTask { await downloadPhoto(named: name) }
5      }
6  }
```

за вас, і дозволяє Swift виявляти деякі помилки під час компіляції (рисунок 2.22).

Рисунок 2.22 – Реалізація “структурованої паралельності”

На додаток до структурованих підходів до паралельності, Swift також підтримує неструктуровану паралельність. На відміну від завдань, які є частиною групи завдань, неструктуроване завдання не має батьківського завдання. Ви маєте повну гнучкість, щоб керувати неструктурованими

```
1  let newPhoto = // ... some photo data ...
2  let handle = Task {
3      return await add(newPhoto, toGalleryNamed: "Spring Adventures")
4  }
5  let result = await handle.value
```

завданнями будь-яким способом, який потребує ваша програма, але ви також несете повну відповідальність за їх правильність. Щоб створити неструктуроване завдання, яке виконується на поточному акторі, викличте ініціализатор Task.init(priority:operation:). Щоб створити неструктуроване завдання, яке не є частиною поточного актора, точніше відомого як відокремлене завдання, викличте метод класу Task.detached (priority:operation:). Обидві ці операції повертають дескриптор завдання, який дозволяє взаємодіяти із завданням, наприклад, чекати його результату або скасувати (рисунок 2.23).

Рисунок 2.23 – Реалізація “неструктурованої паралельності”

Swift concurrency використовує модель кооперативного скасування. Кожне завдання перевіряє, чи було воно скасовано на відповідних етапах його виконання, і відповідає на скасування будь-яким прийнятним способом. Залежно від роботи, яку ви виконуєте, це зазвичай означає одне з наступного:

- видає помилку, наприклад CancellationError;
- повернення нуля або порожньої колекції;

- повернення частково виконаної роботи.

Щоб перевірити скасування, або викличте Task.checkCancellation(), який генерує CancellationError, якщо завдання було скасовано, або перевірте значення Task.isCancelled і обробіть скасування у вашому власному коді. Наприклад, для завдання, яке завантажує фотографії з галереї, може знадобитися видалити часткові завантаження та закрити мережеві з’єднання. Щоб розповсюдити скасування вручну, викликайте Task.cancel().

Подібно до класів, актори є довідковими типами, тому порівняння типів значень і типів посилань у Classes Are Reference Types застосовується як до акторів, так і до класів. На відміну від класів, актори дозволяють лише одній задачі одночасно отримати доступ до змінюваного стану, що робить безпечним взаємодію коду в кількох завданнях з одним і тим же екземпляром актора.

Наприклад, на рисунку 2.24 показано актора, який записує температуру.

```
1  actor TemperatureLogger {
2      let label: String
3      var measurements: [Int]
4      private(set) var max: Int
5
6      init(label: String, measurement: Int) {
7          self.label = label
8          self.measurements = [measurement]
9          self.max = measurement
10     }
11 }
```

Рисунок 2.24 – Реалізація актора, що моніторингу температури

Актор вводиться за допомогою ключового слова actor, а потім його визначення в парних дужках. Актор TemperatureLogger має властивості, до яких може отримати доступ інший код за межами актора, і обмежує властивість max, тому лише код всередині актора може оновлювати максимальне значення.

Ви створюєте екземпляр актора, використовуючи той самий синтаксис ініціалізації, що й структури та класи. Коли ви отримуєте доступ до властивості або методу актора, ви використовуєте await, щоб позначити потенційну точку призупинення (рисунок 2.25).

```
1  let logger = TemperatureLogger(label: "Outdoors", measurement: 25)
2  print(await logger.max)
3  // Prints "25"
```

Рисунок 2.25 – Клієнтський код для актора

У цьому прикладі доступ до файла logger.max є можливою точкою призупинення. Оскільки актор дозволяє лише одній задачі одночасно отримати доступ до свого змінюваного стану, якщо код іншого завдання вже взаємодіє з реєстратором, цей код призупиняється, поки він очікує доступу до властивості.

На відміну від цього, код, який є частиною актора, не пише await під час доступу до властивостей актора. Наприклад, на рисунку 2.26 зображено

```
1  extension TemperatureLogger {
2      func update(with measurement: Int) {
3          measurements.append(measurement)
4          if measurement > max {
5              max = measurement
6          }
7      }
8  }
```

метод, який оновлює TemperatureLogger новою температурою.

Рисунок 2.26 – Метод оновлення температури

Метод update(with:) уже запущено на акторі, тому він не позначає доступ до властивостей, як-от max, за допомогою await. Цей метод також показує одну з причин, чому актори дозволяють лише одному завданню взаємодіяти зі своїм змінним станом: деякі оновлення стану актора тимчасово порушують інваріанти.

Актор TemperatureLogger відстежує список температур і максимальну температуру, а також оновлює максимальну температуру, коли ви записуєте нове вимірювання. У середині оновлення, після додавання нового вимірювання, але перед оновленням max, реєстратор температури перебуває в тимчасовому непостійному стані. Запобігання одночасному взаємодії кількох завдань з одним і тим же екземпляром запобігає таким проблемам, як така послідовність подій:

Код викликає метод update(with:). Спочатку він оновлює масив вимірювань. Перш ніж ваш код зможе оновити максимум, код в іншому місці зчитує максимальне значення та масив температур. Не вдається отримати доступ до logger.max без запису await, оскільки властивості актора є частиною ізольованого локального стану цього актора. Swift гарантує, що лише код всередині актора може отримати доступ до локального стану актора. Ця гарантія відома як ізоляція акторів.

ВИСНОВКИ

В ході написання магістерської роботи було проведено дослідження та проаналізовані алгоритми обробки багатопотокової інформації.

В першому розділі було високорівневово проаналізовано предметну область, що досліджується в даній роботі. Комп'ютерний зір також відіграє важливу роль у програмах розпізнавання облич, технології, яка дозволяє комп'ютерам узгоджувати зображення облич людей з їхніми особами. Алгоритми комп'ютерного зору визначають риси обличчя на зображеннях і порівнюють їх з базами даних профілів обличчя.

В другому розділі було детально описані технології та алгоритми на основі яких буде відбуватися безпосередня реалізація. Сюди увійшов аналіз та вибір технічних засобів, які необхідні для розробки. Перш за все, було визначено, технологію фотограмметрії яка буде вважатися як початковою точкою в оптимізації алгоритму. По-друге було обрано засіб реалізація багато поточності, який стане основою в покращені алгоритму. Також було обрано формат зберігання, який має переваги в зручності та подальшому редагуванню. Проведено наліз та вибір технічних засобів, які необхідні для розробки додатка для обробки відеоінформації, а також мобільного додатка, який надає відеоінформацію та іншу допоміжну інформацію.

Було детально описано процес реалізації мобільного додатку для відеозапису. Платформою реалізація мобільного додатку була обрана iOS. Пристрої на операційній системі з iOS мають необхідні можливості для реалізації даного завдання, а саме потужні процесори з об'єднаною процесорною та графічною пам'яттю, а також модуль камери який складається з необхідних підмодулів.

Також було визначено технічні характеристики камери, такі як:

- Можливість зчитування глибини кадру, яка досягається модулем камери LiDAR.

- Можливість захоплення нерухомого об’єкта з високою розподільною здатністю.
- Можливість отримання руху пристрою, доступною системою відліку, яка надає можливість відслідкувати зміну положення під час запису відео.

Для перевірки працездатності алгоритму було проведено експеримент із розділення відеофайлу на 2 два набори даних, перший містить 60 кадрів, а другий містить 30 кадрів. У збільшеному наборі даних можемо помітити більш чіткі контури, а також циліндр більш правильну форму.

Важливо помітити, що у моделі яка була створена з більшого об’єму даних можливо прочитати текст, що у своєму значно покращує якість всієї моделі. На отриманих фотографіях чітко видно якість контурів кришки пластикової банки.

Згідно результатів дослідження маємо результати, які демонструють результати роботи і з вихідних замірів то можемо помітити, що оптимізований алгоритм показав кращі результати, так як ресурси машини які використовуються під час обробки були зменшені. В результаті отриманих оптимізацій ми можемо оцінити, що багатопотокова обробка відеоінформації пришвидшує швидкодію виконання задачі даного алгоритму.

Отже, поставлена мета виконана в повному обсязі. Функціональні показники та продуктивність обробки багатопотокової відеоінформації, покращені.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Everything You Ever Wanted To Know About Computer Vision [Електронний ресурс]: Технічний блог – Назва з екрана Доступ: https://towardsdatascience.com/everything-youever-wanted-to-know-about-computer-vision-heres-a-look-why-it-s-so-awesome-e8a58dfb641e

2. What is computer vision? [Електронний ресурс]: Технічний блог – Назва з екрана Доступ: https://www.ibm.com/topics/computer-vision

3. Video object recognition and Video object recognition and detection [Електронний ресурс]: Технічний блог – Назва з екрана Доступ: https://www.itransition.com/blog/video-objectrecognition-detection

4. Capturing Photos with Depth [Електронний ресурс]: Технічний блог – Назва з екрана Доступ: https://developer.apple.com/documentation/avfoundation/additional_data_capture/capturing_photos_with_depth

5. Capturing Depth Using the LiDAR Camera [Електронний ресурс]: Технічний блог – Назва з екрана Доступ: https://developer.apple.com/documentation/avfoundation/additional_data_capture/capturing_depth_usi ng_the_lidar_camera

6. Capturing Photographs for RealityKit Object Capture [Електронний ресурс]: Технічний блог – Назва з екрана Доступ: https://developer.apple.com/documentation/realitykit/capturing-photographs-for-realitykit-objectcapture

7. Usdz File Format Specification Depth [Електронний ресурс]: Технічний блог – Назва з екрана Доступ: https://graphics.pixar.com/usd/release/spec_usdz.html

8. Metal framework Depth [Електронний ресурс]: Технічний блог – Назва з екрана Доступ: https://developer.apple.com/documentation/metal/

9. Swift Concurrency [Електронний ресурс]: Технічний блог – Назва з екрана Доступ: https://docs.swift.org/swift-book/LanguageGuide/Concurrency.html

10. RealityKit [Електронний ресурс]: Технічний блог – Назва з екрана Доступ: https://developer.apple.com/documentation/realitykit