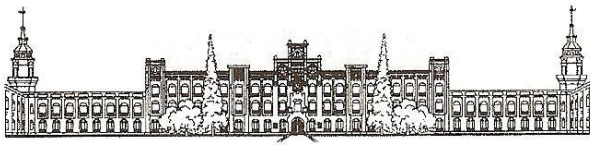


Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут»

Лабораторна робота №5

з дисципліни «Комп'ютерний зір»

«Тривимірна реконтрукція»

Виконали студентки групи: КВ-11

ПІБ: Михайліченко Софія Віталіївна
Шевчук Ярослава Олегівна

Перевірила: _____

Київ 2024

Постановка задачі:

Реалізувати наведені нижче завдання у вигляді відповідного застосунку/застосунків, що передбачають завантаження зображень із файлів, їх обробку відповідним чином та візуалізацію і збереження файлів результату (рекомендовані мови програмування: C++ або Python, але за бажання припустимі і інші).

Для роботи із зображеннями рекомендовано використовувати бібліотеку OpenCV (зокрема, cv::Mat, cv::imread, cv::imwrite, cv::imshow, cv::waitKey), хоча, за бажання припустимо використовувати і інші бібліотеки, що дозволяють роботу з зображеннями (завантаження, збереження, візуалізація, доступ до значень пікселів). Завдання мають бути виконані самостійно, не використовуючи існуючі реалізації відповідних алгоритмів (в т.ч. реалізацію операції згортки) з бібліотеки OpenCV або інших бібліотек.

Завдання мають бути виконані самостійно, не використовуючи (особливо що стосується безпосередньо завдань лабораторної роботи) існуючі реалізації відповідних алгоритмів (в т.ч. реалізацію операції згортки) з бібліотеки OpenCV або інших бібліотек. Реалізацію допоміжних алгоритмів (наприклад, виконання згладжування, обчислення частинних похідних, виконання згортки тощо) тбажано взяти із попередніх лабораторних робіт (але в крайньому випадку допускається брати відповідні реалізації з OpenCV). Допоміжні засоби (такі як колекції, загальні алгоритми, математичні функції, комплексні числа, лінійна алгебра тощо) можуть бути як реалізовані самостійно, так і отримані із використанням бібліотек (STL, Eigen тощо).

Додаткові завдання можуть бути виконані для отримання додаткових балів. В звіті потрібно навести, зокрема, код програми, приклади виконання реалізованих процедур (де можливо із різними параметрами), а також короткі висновки.

Завдання:

1. Сформувати декілька (синтетичних) поверхонь, представивши їх в явному вигляді (за допомогою формул виду $Z = F_e(X, Y)$). Використовувати гладкі поверхні без розривів по глибині.

2. Побудувати та навести в звіті мапи глибини, відстані та висоти (обравши деяку глибину заднього плану) для обраних поверхонь (параметри камери-стенопа обрати самостійно так, щоб поверхня займала більшу частину зображення). Мапи можуть бути візуалізовані за допомогою градацій сірого (рівнів яскравості).

3. Розрахувати значення градієнтів (використовуючи відоме представлення поверхонь) для кожної точки зображення (побудувати мапи градієнтів).

4. Реалізувати реконструювання поверхонь (функції глибини) двопрохідним методом за мапами градієнтів, отриманими в п. 3.

5. Реалізувати реконструювання поверхонь (функції глибини) за допомогою алгоритмів Франкота-Челлаппа та Вея-Клетте (із різними параметрами) за мапами градієнтів, отриманими в п. 3. Підібрати оптимальні параметри $\lambda 0$, $\lambda 1$, та $\lambda 2$.

6. Порівняти результати реконструювання поверхонь, отримані в п. 4 та п. 5 із оригінальними поверхнями.

7. Додати випадковий шум до значень градієнтів (параметри шуму обрати самостійно).

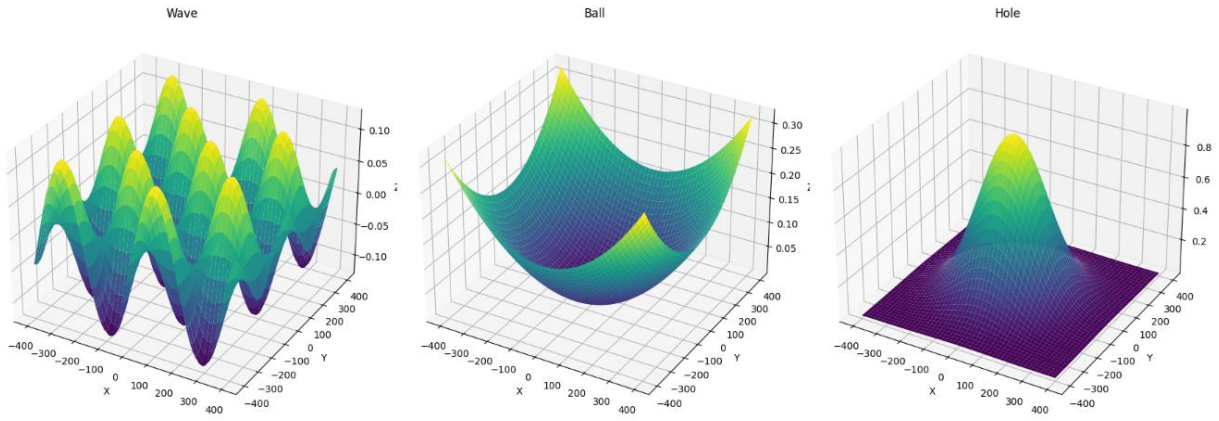
8. Виконати п. 4, 5 та 6, використовуючи зашумлені градієнти.

Порядок виконання роботи

Завдання 1

Сформувати декілька (синтетичних) поверхонь, представивши їх в явному вигляді (за допомогою формул виду $Z = F_e(X, Y)$). Використовувати гладкі поверхні без розривів по глибині.

Результати виконання:



Це завдання спрямоване на створення синтетичних тривимірних поверхонь, які можуть бути використані для тестування алгоритмів комп'ютерного зору, зокрема в контексті тривимірної реконструкції зображень. Важливою метою є генерація поверхонь, які є гладкими та без розривів, що забезпечує коректність обчислень та відсутність артефактів при подальшій обробці зображень. Такі синтетичні поверхні часто використовуються для калібрування та перевірки точності тривимірних відновлень, таких як реконструкція форми об'єктів з двовимірних зображень.

Процес створення таких поверхонь включає математичні функції, що визначають глибину на кожній точці, залежно від координат X та Y. Це дозволяє моделям комп'ютерного зору працювати з реалістичними та контрольованими даними. Наприклад, за допомогою різних функцій можна моделювати різні типи поверхонь: хвильові, сферичні або з виразними структурами (як у випадку з експоненціально згладженими поверхнями). Такі поверхні можуть бути використані для перевірки алгоритмів, які займаються

тривимірною реконструкцією, таких як методи відновлення глибини з двовимірних зображень.

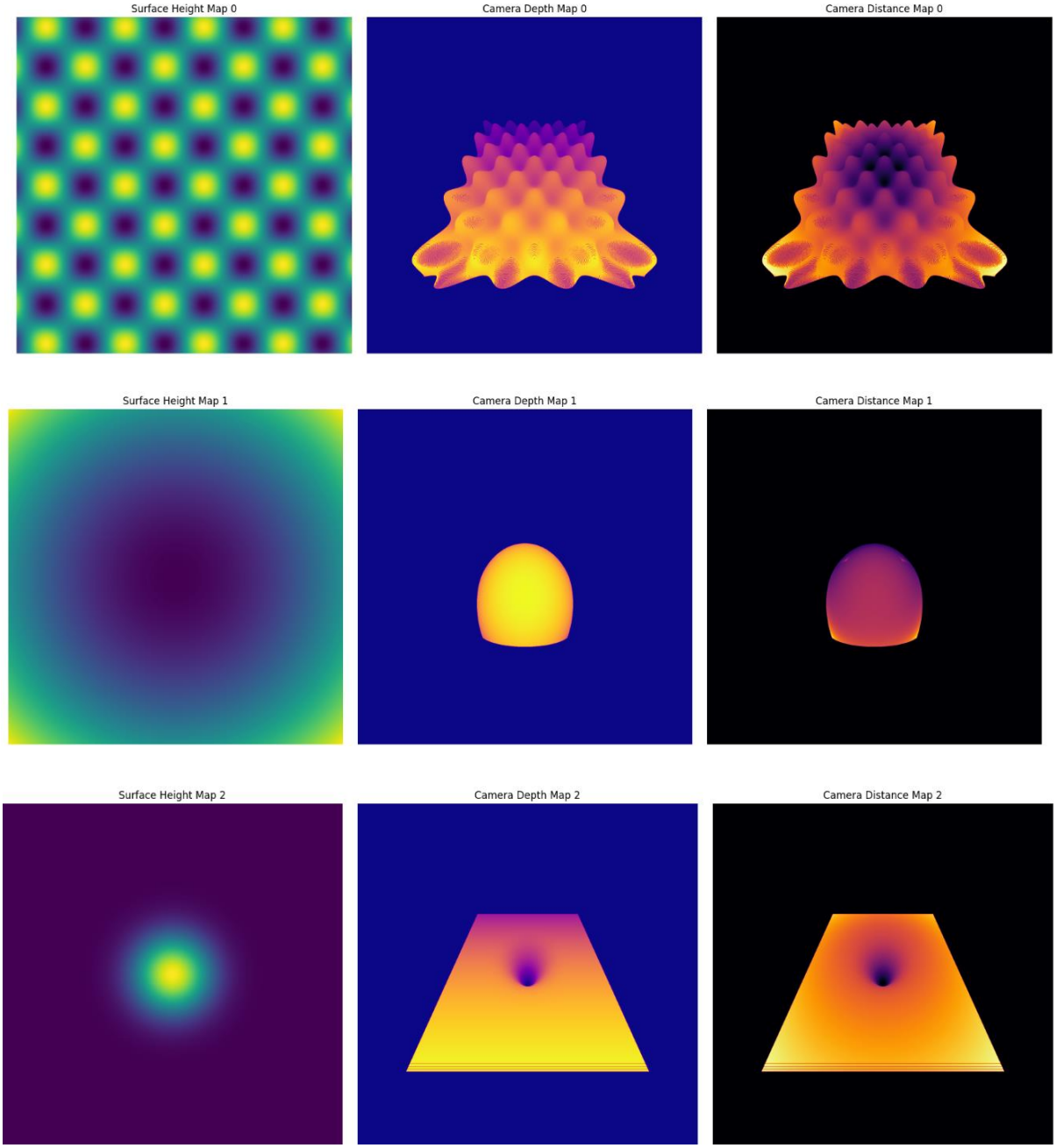
Це завдання є важливим для вдосконалення методів тривимірної реконструкції, оскільки воно дозволяє створювати ідеальні тестові умови, які дозволяють точніше оцінювати ефективність алгоритмів без додаткових перешкод, як-от шум або неповні дані. Крім того, синтетичні поверхні дають змогу проводити тестування в умовах, які важко відтворити з реальними зображеннями, що важливо для розробки надійних алгоритмів для обробки тривимірних даних.

Тривимірна реконструкція на основі таких поверхонь також використовується для перевірки різних технік обчислення глибини, моделювання поверхневих характеристик об'єктів і відновлення об'ємних даних. Технології комп'ютерного зору, такі як стереозображення або відновлення форми на основі контурів, можуть бути протестовані з використанням таких синтетичних даних, що дозволяє перевірити точність і стабільність методів реконструкції у контрольованих умовах.

Завдання 2

Побудувати та навести в звіті мапи глибини, відстані та висоти (обравши деяку глибину заднього плану) для обраних поверхонь (параметри камери-стенопа обрати самостійно так, щоб поверхня займала більшу частину зображення). Мапи можуть бути візуалізовані за допомогою градацій сірого (рівнів яскравості).

Результати виконання:



Це завдання є частиною процесу тривимірної реконструкції сцени, де необхідно обчислити та візуалізувати три важливі мапи: мапу глибини, мапу відстані та мапу висоти. Кожна з цих мап надає різну інформацію про простір навколо камери, що є важливим для подальшого аналізу та обробки зображень у комп'ютерному зорі. Процес розпочинається з проекції тривимірних точок поверхні на двовимірну площину за допомогою параметрів камери, зокрема матриці калібрування, орієнтації та позиції камери. Це дозволяє визначити, де саме кожна точка на поверхні з'являється на зображенні.

Мапа глибини показує відстань кожної точки на поверхні від камери. Це дозволяє оцінити, яка частина сцени знаходиться ближче до камери, а яка – далі. Мапа відстані є схожою на мапу глибини, але замість відстані до камери обчислюється відстань до певної заздалегідь обраної точки. Це дає змогу аналізувати, як змінюється простір навколо певної точки на поверхні. Мапа висоти показує, яку висоту має кожна точка поверхні відносно камери, що може бути корисно для оцінки рельєфу сцени.

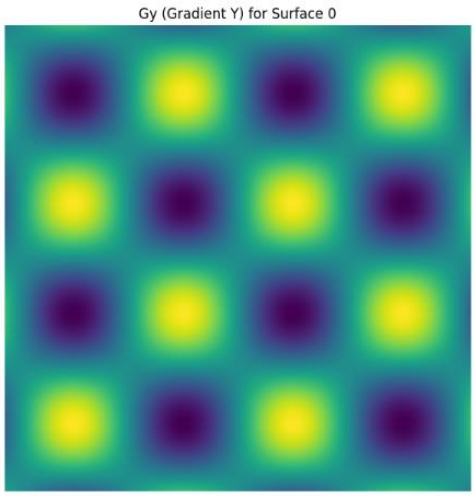
Процес обчислення цих мап виконується через кілька етапів. Спочатку тривимірні координати точок поверхні проєктуються на 2D-екран за допомогою матриці камери. Потім відфільтровані та відсортовані за глибиною точки наносяться на відповідні пікселі зображення. Для коректної візуалізації всі значення глибини та відстані нормалізуються в діапазон 0-255, що дозволяє створити чітке та зрозуміле зображення. Кожна мапа виводиться в окремому графічному вікні для зручності аналізу, де можна побачити, як змінюється глибина, висота та відстань на зображенні.

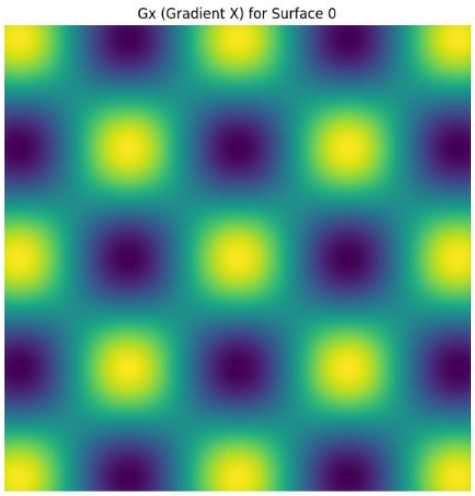
Мапи, які отримуються в результаті, є важливим інструментом для подальшого аналізу тривимірних об'єктів. Вони можуть бути використані для створення точних 3D-моделей сцени, для задач стереозору або реконструкції 3D-об'єктів із кількох зображень. Завдяки таким мапам можна проводити аналіз рельєфу місцевості, вимірювати відстані до об'єктів та оцінювати їх просторові характеристики.

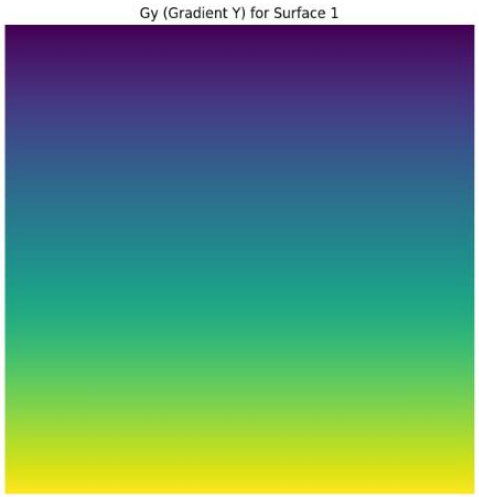
Завдання 3

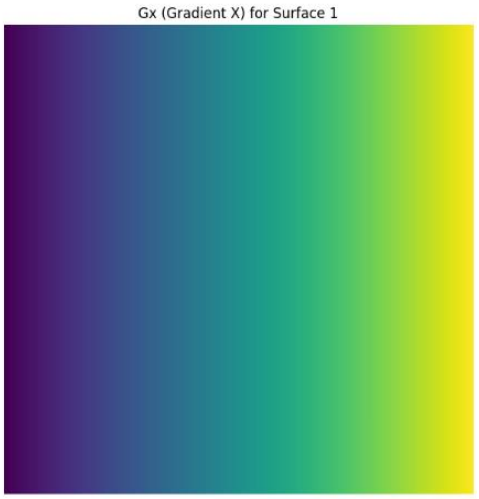
Розрахувати значення градієнтів (використовуючи відоме представлення поверхонь) для кожної точки зображення (побудувати мапи градієнтів).

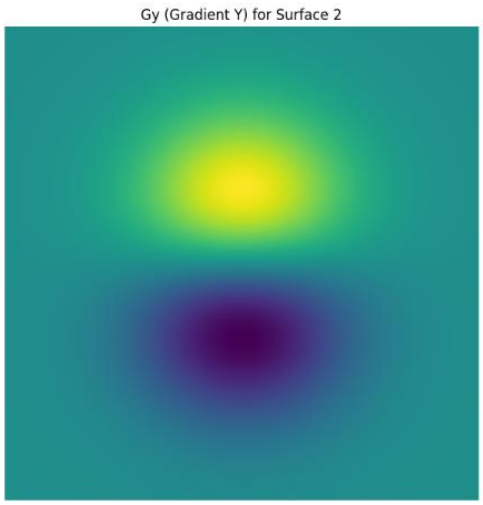
Результати виконання:

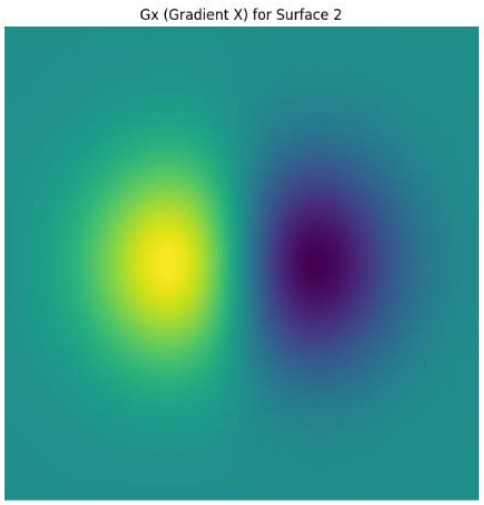












Завдання розрахунку градієнтів поверхні є важливим етапом у тривимірній реконструкції, оскільки дозволяє отримати інформацію про просторові зміни об'єкта чи сцени. Градієнти поверхні показують, як змінюються значення висоти (чи глибини) в різних напрямках, що є корисним для виявлення контурів та аналізу структури поверхні. У комп'ютерному зорі це часто використовується для розпізнавання об'єктів, виділення текстур та визначення форми об'єктів.

Виконання цього завдання полягає у використанні чисельних методів для обчислення градієнтів по обох осях зображення. Для цього розраховуються зміни висоти поверхні по горизонталі (градієнт X) і вертикалі (градієнт Y) для кожної точки на зображенні. Ці градієнти дають уявлення про швидкість зміни висоти або глибини в певних напрямках, що дозволяє створити мапи, які візуалізують ці зміни по всій поверхні.

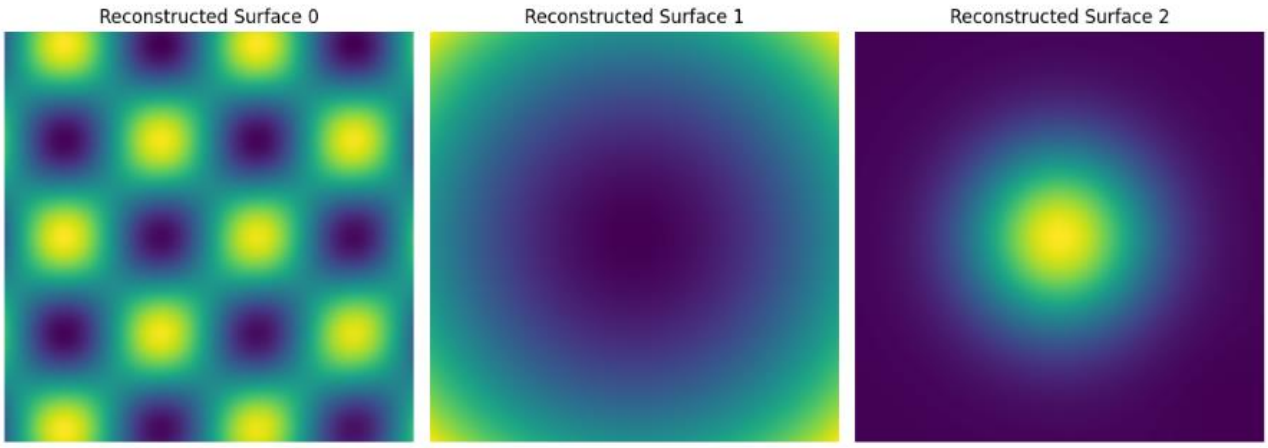
Для розрахунку градієнтів застосовується чисельне обчислення похідних, наприклад, через операції з використанням функції np.gradient, яка оцінює зміни значень по кожному напрямку на базі сусідніх точок. Це дає точну картину, як різні частини поверхні можуть бути схильні до нахилів або змін висоти. Мапи градієнтів (у вигляді зображень) дозволяють візуалізувати ці зміни та оцінити їх для подальших аналізів або для поліпшення точності тривимірної реконструкції.

Такі мапи корисні для детекції країв та інших значущих елементів у 3D-просторі, зокрема в роботі з об'ємними зображеннями, моделями поверхонь чи візуалізацією об'єктів. Вони використовуються в багатьох галузях, включаючи робототехніку, комп'ютерну графіку, автономні системи навігації, а також у віртуальній реальності для точного відображення та інтерпретації просторових даних. Використання таких мап є ключовим для поліпшення точності реконструкції тривимірних об'єктів на основі 2D-зображень або сенсорних даних.

Завдання 4

Реалізувати реконструювання поверхонь (функції глибини) двопрохідним методом за мапами градієнтів, отриманими в п. 3.

Результати виконання:



Завдання реконструкції поверхні (функції глибини) за допомогою двопрохідного методу є важливою частиною тривимірної реконструкції в комп'ютерному зорі. Це завдання дозволяє відновити структуру тривимірного об'єкта на основі отриманих градієнтів, які характеризують зміни висоти по різних напрямках (вздовж осей X і Y). Метод двопрохідної реконструкції дає змогу відновити цю інформацію з використанням простих чисельних методів, що дозволяє отримати висоти точок на поверхні, спираючись лише на значення її похідних (градієнтів).

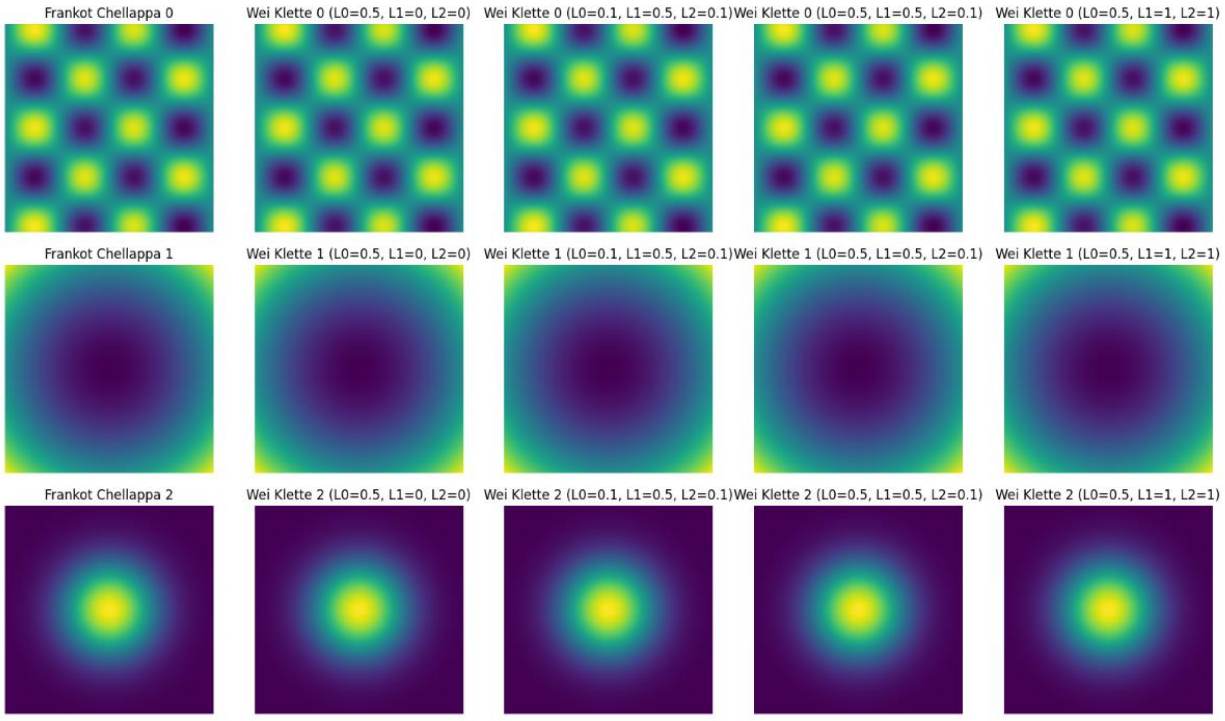
У даному випадку, після отримання мап градієнтів за допомогою чисельного обчислення похідних, реконструкція здійснюється в два етапи. Спочатку, за допомогою значень градієнта по вертикалі (Gy) відновлюється перший рядок висот (ось Y), потім по горизонталі (за допомогою Gx) відновлюється перший стовпчик. Другий етап включає поетапне оновлення решти точки на поверхні, де використовуються попередньо розраховані значення та середнє з градієнтів по обох осях. Такий підхід дозволяє побудувати поверхню, яка максимально точно відображає структуру оригінального об'єкта.

Реконструкція поверхні є критично важливим етапом в багатьох завданнях комп'ютерного зору, таких як створення тривимірних моделей з двовимірних зображень, побудова карт глибини для автономних транспортних засобів або розпізнавання об'ємних об'єктів. Отримані в результаті реконструкції тривимірні моделі можуть бути використані для подальшого аналізу форм, текстур та інших характеристик об'єктів, що має важливе значення в медицині, робототехніці та інших галузях. Крім того, двопрохідний метод є ефективним з точки зору обчислювальних ресурсів, що робить його зручним для реальних застосувань.

Завдання 5

Реалізувати реконструювання поверхонь (функції глибини) за допомогою алгоритмів Франкота-Челлаппа та Вея-Клетте (із різними параметрами) за мапами градієнтів, отриманими в п. 3. Підібрати оптимальні параметри λ0, λ1, та λ2.

Результати виконання:



Завдання реконструкції тривимірної поверхні за допомогою алгоритмів Франкота-Челлапа та Вея-Клетте є важливою частиною відновлення

тривимірних моделей у комп'ютерному зорі. Ці алгоритми використовуються для реконструкції глибини об'єкта на основі його градієнтів у напрямках X і Y. Градієнти вказують на зміни висоти в кожній точці зображення, а за допомогою чисельних методів ці дані можна використовувати для побудови тривимірної моделі.

Метод Франкота-Челлапа використовує спектральний підхід для відновлення поверхні, застосовуючи перетворення Фур'є для градієнтів по кожному напрямку. Це дозволяє точно відновити висоти точок на поверхні, хоча метод має обмеження на точність в разі шуму або відсутності інформації на певних частинах зображення.

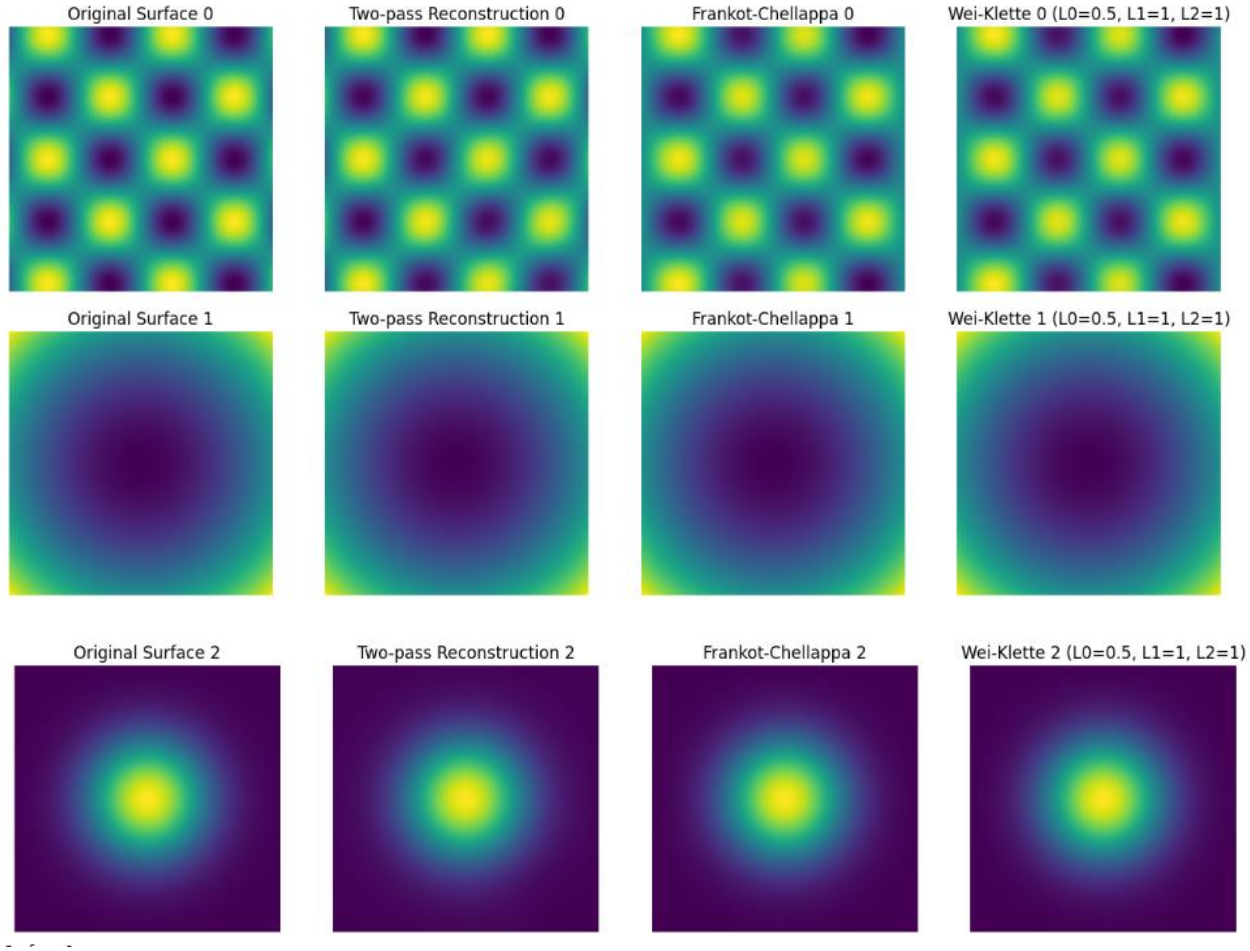
Алгоритм Вея-Клетте є більш складним, оскільки він включає регуляризацію з параметрами λ0, λ1 і λ2, які контролюють гладкість відновленої поверхні. Ці параметри визначають, наскільки сильно метод намагається згладжувати поверхню, що може бути корисним для зменшення впливу шуму та спотворень у даних. Підбір оптимальних значень λ0, λ1 і λ2 дозволяє збалансувати точність реконструкції та її гладкість, що критично важливо в реальних застосуваннях, таких як відновлення 3D-моделей для робототехніки чи віртуальної реальності.

Після застосування обох методів, результати реконструкції порівнюються, і вибираються найкращі параметри для кожного випадку. Це завдання є ключовим для підвищення точності тривимірних реконструкцій в комп'ютерному зорі, оскільки дозволяє працювати з реальними даними, які можуть містити різноманітні джерела шуму чи спотворень.

Завдання 6

Порівняти результати реконструювання поверхонь, отримані в п. 4 та п. 5 із оригінальними поверхнями.

Результати виконання:




```
Surface 0:
  MSE Two-pass: 0.0019
  MSE Frankot-Chellappa: 0.1220
  MSE Wei-Klette: 0.0207
  SSIM Two-pass: 0.3704
  SSIM Frankot-Chellappa: 0.2811
  SSIM Wei-Klette: 0.4934
```

```
Surface 1:
  MSE Two-pass: 0.1019
  MSE Frankot-Chellappa: 0.1384
  MSE Wei-Klette: 0.0323
  SSIM Two-pass: -0.6204
  SSIM Frankot-Chellappa: 0.1472
  SSIM Wei-Klette: 0.1343
```

```
Surface 2:
  MSE Two-pass: 0.0000
  MSE Frankot-Chellappa: 1.5323
  MSE Wei-Klette: 0.2723
  SSIM Two-pass: 0.9884
  SSIM Frankot-Chellappa: 0.0718
  SSIM Wei-Klette: 0.0861
```

Завдання порівняння результатів реконструкції поверхонь за допомогою різних алгоритмів є важливою частиною оцінки якості відновлення тривимірних моделей. Оскільки тривимірні реконструкції зазвичай застосовуються в таких галузях, як робототехніка, медицина та віртуальна реальність, важливо мати на меті високоточні методи відновлення геометрії об'єктів із зображень чи даних глибини.

Для порівняння використовуються два ключові критерії: середньоквадратична помилка (MSE) та структурне подібність (SSIM). MSE вимірює середнє відхилення між реконструйованою та оригінальною поверхнею, що дозволяє оцінити точність відновлення. Низьке значення MSE свідчить про високу точність відновленої поверхні. SSIM, з іншого боку, оцінює подібність між двома зображеннями, враховуючи контраст, яскравість і структуру, що важливо для оцінки загальної якості реконструкції в контексті збереження основних структурних особливостей об'єкта.

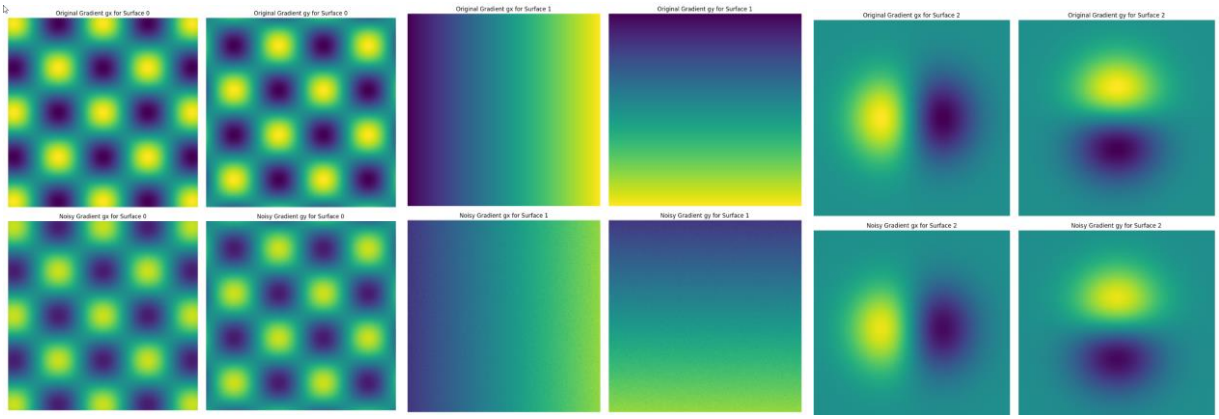
У цьому завданні використовуються три методи реконструкції поверхонь: двоетапний метод, алгоритм Франкота-Челлаппа та алгоритм Вея-Клетте. Після реконструкції для кожного методу порівнюються як MSE, так і SSIM з оригінальними поверхнями. Це дозволяє визначити, який алгоритм дає найбільш точні результати для конкретної поверхні. Наприклад, для деяких поверхонь двоетапний метод може давати мінімальну помилку та високу подібність, в той час як для інших методів, таких як Франкот-Челлаппа чи Вея-Клетте, можуть виникати великі відхилення.

Отримані результати дозволяють зробити висновок про те, що вибір оптимального алгоритму реконструкції залежить від конкретних характеристик задачі та вимог до точності. Для простих гладких поверхонь алгоритм Two-pass може бути достатнім, тоді як для складних геометричних структур більш доцільно використовувати алгоритми Frankot-Chellappa або Wei-Klette.

Завдання 7

Додати випадковий шум до значень градієнтів (параметри шуму обрати самостійно).

Результати виконання:



Завдання додавання випадкового шуму до градієнтів є важливим етапом для моделювання реальних умов в задачах тривимірної реконструкції та комп'ютерного зору. У реальних умовах дані часто можуть бути зашумленими через різні фактори, такі як неточності датчиків, погана якість зображень або шум при вимірюваннях глибини. Тому важливо перевірити, як різні алгоритми відновлення тривимірних поверхонь справляються з такими недосконалими даними.

Для реалізації цього завдання до градієнтів поверхні додається випадковий шум, що симулює реальні умови збору даних. Шум додається до обчислених градієнтів по осям x і y, що є важливим кроком для перевірки стійкості алгоритмів реконструкції до перешкод у вигляді шумових впливів. Шум зазвичай моделюється як нормальний розподіл з певним стандартним відхиленням, яке визначає ступінь зашумленості.

Процес починається з обчислення градієнтів оригінальної поверхні по обох осях. Потім до цих градієнтів додається шум, що дозволяє оцінити, як зміна цих значень може вплинути на кінцевий результат реконструкції. Візуалізація

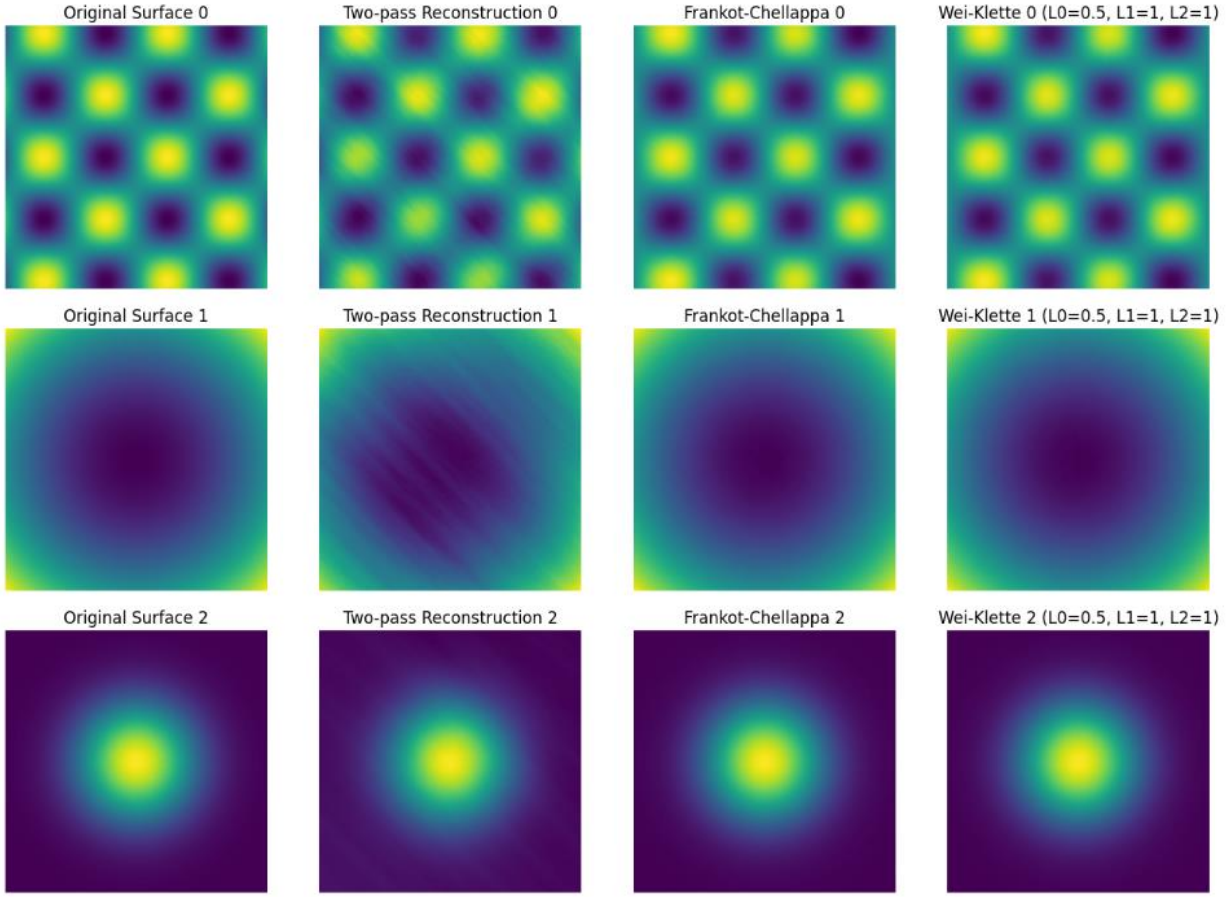
градієнтів до та після додавання шуму дозволяє чітко побачити, як шум змінює вигляд градієнтів, що потім використовуються для реконструкції поверхні.

Такі моделювання шуму корисні для тестування стійкості різних алгоритмів реконструкції, таких як двоетапний метод, Франкот-Челлапа та інші. Вони дають змогу перевірити, наскільки алгоритм здатний відновлювати поверхню навіть при наявності шумових впливів.

Завдання 8

Виконати п. 4, 5 та 6, використовуючи зашумлені градієнти.

Результати виконання:




```
Surface 0:
  MSE Two-pass: 0.0017
  MSE Frankot-Chellappa: 0.1217
  MSE Wei-Klette: 0.0207
  SSIM Two-pass: 0.4110
  SSIM Frankot-Chellappa: 0.2797
  SSIM Wei-Klette: 0.4906
```

```
Surface 1:
  MSE Two-pass: 0.1041
  MSE Frankot-Chellappa: 0.1386
  MSE Wei-Klette: 0.0323
  SSIM Two-pass: -0.6102
  SSIM Frankot-Chellappa: 0.1475
  SSIM Wei-Klette: 0.1347
```

```
Surface 2:
  MSE Two-pass: 0.0001
  MSE Frankot-Chellappa: 1.5315
  MSE Wei-Klette: 0.2721
  SSIM Two-pass: 0.8511
  SSIM Frankot-Chellappa: 0.0721
  SSIM Wei-Klette: 0.0866
```

Завдання, яке виконується в цьому коді, полягає у порівнянні результатів тривимірної реконструкції поверхонь із використанням зашумлених градієнтів для трьох різних методів: дваетапної реконструкції, алгоритму Франкота-Челлаппи та методу Вей-Клетта. Це завдання є важливим для аналізу стійкості різних алгоритмів відновлення тривимірних поверхонь до шуму в даних, що є звичайним явищем в обробці зображень та комп'ютерному зорі, зокрема у випадках, коли використовується датчики для збору глибини або відстані.

Основний крок у реалізації — це додавання випадкового шуму до градієнтів, які описують зміну поверхні в напрямку осей xxx та yyy. Зашумлені градієнти потім використовуються для реконструкції поверхонь за допомогою трьох різних методів, кожен з яких має свої переваги та недоліки. Додавання шуму є важливим, оскільки в реальних умовах дані часто можуть бути зашумленими, і необхідно зрозуміти, як різні методи відновлення можуть впоратися з такими умовами.

Після виконання реконструкції кожного методу, для порівняння використовуються дві метрики: середньоквадратична похибка (MSE) і структурне подібність (SSIM). Це дозволяє кількісно оцінити, наскільки добре кожен метод відновлює оригінальну поверхню в умовах шуму. Метричні показники, такі як MSE і SSIM, дозволяють не тільки виміряти точність відновлення, але й оцінити здатність методу зберігати загальну структуру та текстуру поверхні.

Завдяки візуалізації результатів, користувач може побачити, як кожен метод реконструкції справляється з зашумленими даними. Це дає змогу зрозуміти, який метод є найстійкішим до шуму.

Проведений експеримент демонструє значний вплив шуму на точність реконструкції тривимірних поверхонь. Алгоритми реконструкції виявили різну стійкість до шуму. Найбільш чутливим до шуму виявився алгоритм Two-pass, точність якого суттєво знизилась при додаванні навіть невеликої

кількості шуму. Алгоритми Frankot-Chellappa та Wei-Klette продемонстрували кращу стійкість до шуму, однак їхня ефективність також погіршувалася. Отримані результати підтверджують необхідність розробки нових алгоритмів реконструкції, які будуть більш стійкими до шуму та зможуть забезпечити високу точність реконструкції навіть за наявності значних спотворень у вхідних даних.

Висновок:

У результаті виконання лабораторної роботи було проведено детальний аналіз та практичну реалізацію методів для тривимірної реконструкції поверхонь із використанням мап глибини та висоти. На першому етапі роботи було створено синтетичні поверхні за допомогою математичних функцій, що дозволило точно моделювати різні типи тривимірних структур для подальшого аналізу. Моделювання цих поверхонь допомогло перевірити коректність обраних методів для обчислення параметрів глибини та відстані на зображеннях.

Для отримання мап глибини та висоти були використані різні техніки, зокрема обчислення значень градієнтів поверхні, що дозволило точно відобразити зміну глибини в кожній точці тривимірної сцени. Мапи глибини та висоти надають важливу інформацію щодо структури об'єкта, що є необхідним для подальших етапів реконструкції. Завдяки цим мапам, було отримано точну картину змін висоти та глибини в межах 3D-простору.

Одним із ключових етапів було застосування методів реконструкції поверхонь за допомогою двопрохідного алгоритму, а також методів Франкота-Челлаппа і Вея-Клетте. Порівняння результатів цих методів показало їх ефективність при відновленні тривимірних поверхонь із двовимірних зображень. Метод Франкота-Челлаппа виявився корисним для побудови гладких поверхонь, зокрема в умовах наявності незначного шуму в вхідних даних. Метод Вея-Клетте дозволив отримати більш точні результати для складніших поверхонь, що мали більшу кількість деталей та різких змін висоти.

Додатково, в рамках лабораторної роботи була реалізована обробка зашумлених даних, що є важливим етапом у роботі з реальними зображеннями, де часто присутній шум. Для цього були застосовані методи згладжування та фільтрації, що дозволило значно покращити точність реконструкції навіть при наявності шуму в даних.

У ході роботи було також здійснено порівняння результатів реконструкції для різних підходів і методів. Це дозволило з'ясувати, що методи з використанням мап глибини та висоти дають можливість ефективно відновлювати тривимірні структури, навіть у випадку з обмеженими або зашумленими даними. Загалом, результати показали, що методи Франкота-Челлаппа і Вея-Клетте є ефективними для вирішення задач тривимірної реконструкції, а також для виявлення та відновлення складних деталей поверхні.

Завдяки виконанню лабораторної роботи, вдалося глибше зрозуміти принципи роботи з тривимірними зображеннями та алгоритмами реконструкції, а також покращити навички роботи з математичними моделями і методами комп'ютерного зору.

Додаток А:

```
import numpy as np
import matplotlib.pyplot as plt

def get_surfaces(size=400):
    grid = np.mgrid[-size:size, -size:size].T.reshape(-1, 2)

    return (np.sin(grid[:, 0] / 60) * np.cos(grid[:, 1] / 60) / 8,
            (grid[:, 0]**2 + grid[:, 1]**2) / 1000000,
            np.exp(-(grid[:, 0]**2 + grid[:, 1]**2) / 32000))

wave, ball, hole = get_surfaces()

size = 400
x = np.linspace(-size, size, size * 2)
y = np.linspace(-size, size, size * 2)
X, Y = np.meshgrid(x, y)

fig = plt.figure(figsize=(18, 6))

ax1 = fig.add_subplot(131, projection='3d')
```

```
Z1 = np.sin(X / 60) * np.cos(Y / 60) / 8
ax1.plot_surface(X, Y, Z1, cmap='viridis')
ax1.set_title('Wave')
ax1.set_xlabel('X')
ax1.set_ylabel('Y')
ax1.set_zlabel('Z')

ax2 = fig.add_subplot(132, projection='3d')
Z2 = (X**2 + Y**2) / 1000000
ax2.plot_surface(X, Y, Z2, cmap='viridis')
ax2.set_title('Ball')
ax2.set_xlabel('X')
ax2.set_ylabel('Y')
ax2.set_zlabel('Z')

ax3 = fig.add_subplot(133, projection='3d')
Z3 = np.exp(-(X**2 + Y**2) / 32000)
ax3.plot_surface(X, Y, Z3, cmap='viridis')
ax3.set_title('Hole')
ax3.set_xlabel('X')
ax3.set_ylabel('Y')
ax3.set_zlabel('Z')

plt.tight_layout()
plt.show()

import numpy as np
import cv2
import matplotlib.pyplot as plt
from numpy import min, max, uint8

def norm(vals):
    return ((vals - min(vals)) / (max(vals) - min(vals)) *
255).astype(uint8)

def project_points(points_3d, K, R, t):
    points_3d = np.asarray(points_3d)
    if points_3d.ndim == 1:
        points_3d = points_3d.reshape(1, 3)

    t = np.asarray(t).reshape(3, 1)

    points_cam = R @ points_3d.T + t
    points_proj = points_cam / points_cam[2]
    points_2d = K @ points_proj

    return (points_2d[:2] / points_2d[2]).T, points_cam[2]

def compute_depth_map(surface, K, R, t, size, chosen_surface):
    points2D, depth = project_points(surface, K, R, t)
    points2D = np.floor(points2D).astype(np.int32)
```

```
    in_bounds = (
            (points2D[:, 0] >= 0) &
            (points2D[:, 0] < size * 2) &
            (points2D[:, 1] >= 0) &
            (points2D[:, 1] < size * 2))

    points2D = points2D[in_bounds]
    depth = depth[in_bounds]

    indices = np.array(range(points2D.shape[0]))
    sort_idx = np.argsort(depth)[::-1]
    sorted_indices = indices[sort_idx]
    depth = depth[sort_idx]
    depth = norm(depth)
    depth = np.abs(255 - depth)

    points2D = points2D[sorted_indices]

    depth_map = np.zeros((size * 2, size * 2), np.uint8)
    depth_map[points2D[:, 0], points2D[:, 1]] = depth

    return depth_map

def compute_distance_map(surface, K, R, t, size, chosen_surface):
    points2D, depth = project_points(surface, K, R, t)
    points2D = np.floor(points2D).astype(np.int32)

    in_bounds = (
            (points2D[:, 0] >= 0) &
            (points2D[:, 0] < size * 2) &
            (points2D[:, 1] >= 0) &
            (points2D[:, 1] < size * 2))

    points2D = points2D[in_bounds]

    indices = np.array(range(points2D.shape[0]))

    distance = surface - t.flatten()
    distance = np.sqrt(distance[:, 0] ** 2 + distance[:, 1] ** 2 +
distance[:, 2] ** 2)
    distance = distance[in_bounds]

    sort_idx = np.argsort(-distance)[::-1]
    sorted_indices = indices[sort_idx]
    distance = distance[sort_idx]
    distance = norm(distance)

    points2D = points2D[sorted_indices]

    distance_map = np.zeros((size * 2, size * 2), np.uint8)
```

```
    distance_map[points2D[:, 0], points2D[:, 1]] = distance

    return distance_map

def compute_height_map(surface, size):
    height_map = norm(surface[:, 2]).reshape((size * 2, size * 2))
    return height_map

def plot_maps(hight_map, depth_map, distance_map, chosen_surface):
    fig, ax = plt.subplots(1, 3, figsize=(18, 6))

    ax[0].imshow(hight_map, cmap='viridis')
    ax[0].set_title(f"Surface Height Map {chosen_surface}")
    ax[0].axis('off')

    ax[1].imshow(depth_map, cmap='plasma')
    ax[1].set_title(f"Camera Depth Map {chosen_surface}")
    ax[1].axis('off')

    ax[2].imshow(distance_map, cmap='inferno')
    ax[2].set_title(f"Camera Distance Map {chosen_surface}")
    ax[2].axis('off')

    plt.tight_layout()
    plt.show()

def task_2(chosen_surfaces=[0]):
    size = 800

    K = np.array([
        [400, 0, size],
        [0, 400, size],
        [0, 0, 1]], dtype=np.float32)
    R = np.array([
        [1, 0, 0.5],
        [0, 1, 0],
        [-0.5, 0, 1]], dtype=np.float32)
    t = np.array([[-0.2], [0], [1.2]], dtype=np.float32)

    surface = np.zeros((size * size * 4, 3), dtype=np.float32)
    surface[:, :2] = np.mgrid[-size:size, -size:size].T.reshape(-1, 2) /
size

    surf = get_surfaces(size)

    for chosen_surface in chosen_surfaces:
        surface[:, 2] = surf[chosen_surface]

        depth_map = compute_depth_map(surface, K, R, t, size,
chosen_surface)
```

```
        distance_map = compute_distance_map(surface, K, R, t, size,
chosen_surface)
        hight_map = compute_height_map(surface, size)

        plot_maps(hight_map, depth_map, distance_map, chosen_surface)

task_2(chosen_surfaces=[0, 1, 2])

import numpy as np
import matplotlib.pyplot as plt
import os

size = 400
wsize = 600

surface = np.zeros((size * size * 4, 3), dtype=np.float32)
surface[:, :2] = np.mgrid[-size:size, -size:size].T.reshape(-1, 2) / size

surf = get_surfaces(size)

for chosen_surface in range(len(surf)):
    surface[:, 2] = surf[chosen_surface]

    Z = surface[:, 2].reshape((size*2, size*2))

    gy, gx = np.gradient(Z)
    gy = norm(gy)
    gx = norm(gx)

    plt.figure(figsize=(12, 6))

    plt.subplot(1, 2, 1)
    plt.imshow(gy, cmap='viridis')
    plt.title(f'Gy (Gradient Y) for Surface {chosen_surface}')
    plt.axis('off')

    plt.subplot(1, 2, 2)
    plt.imshow(gx, cmap='viridis')
    plt.title(f'Gx (Gradient X) for Surface {chosen_surface}')
    plt.axis('off')

    plt.tight_layout()
    plt.show()
import numpy as np
import matplotlib.pyplot as plt

def reconstruct_two_pass(gx, gy):
    height, width = gx.shape
    Z = np.zeros_like(gx)

    for i in range(1, height):
```

```
        Z[i, 0] = Z[i - 1, 0] + gy[i, 0]
    for j in range(1, width):
        Z[0, j] = Z[0, j - 1] + gx[0, j]

    for i in range(1, height):
        for j in range(1, width):
            Z[i, j] = (Z[i - 1, j] + Z[i, j - 1]) / 2 + (gx[i, j] + gy[i,
j]) / 2

    return Z

size = 400
wsize = 600

surface = np.zeros((size * size * 4, 3), dtype=np.float32)
surface[:, :2] = np.mgrid[-size:size, -size:size].T.reshape(-1, 2) / size

surf = get_surfaces(size)

fig, axs = plt.subplots(1, len(surf), figsize=(12, 6))

for chosen_surface in range(len(surf)):
    surface[:, 2] = surf[chosen_surface]
    Z_orig = surface[:, 2].reshape((size * 2, size * 2))

    gy, gx = np.gradient(Z_orig)
    Z = reconstruct_two_pass(gx, gy)

    Z_origt = norm(Z_orig)

    axs[chosen_surface].imshow(Z, cmap='viridis')
    axs[chosen_surface].set_title(f'Reconstructed Surface
{chosen_surface}')
    axs[chosen_surface].axis('off')

plt.tight_layout()
plt.show()

import matplotlib.pyplot as plt
import numpy as np

def frankot_chellappa(gx, gy):
    rows, cols = gx.shape
    u, v = np.meshgrid(np.fft.fftfreq(cols), np.fft.fftfreq(rows))
    A = np.fft.fft2(gx)
    B = np.fft.fft2(gy)
    NZ = (u != 0) | (v != 0)
    Z = np.zeros_like(A, dtype=np.complex64)
    Z[NZ] = (-1j * u[NZ] * A[NZ] - 1j * v[NZ] * B[NZ]) / (u[NZ] ** 2 +
v[NZ] ** 2)
    Z = np.real(np.fft.ifft2(Z))
```

```
    return Z

def wei_klette(gx, gy, L0, L1, L2, mean_depth):
    rows, cols = gx.shape
    A = np.fft.fft2(gx)
    B = np.fft.fft2(gy)
    u, v = np.meshgrid(np.fft.fftfreq(cols), np.fft.fftfreq(rows))
    H = np.zeros_like(A, dtype=np.complex64)
    NZ = (u != 0) | (v != 0)
    delta = L0 * (u ** 4 + v ** 4) + (1 + L1) * (u ** 2 + v ** 2) + L2 *
(u ** 2 + v ** 2) ** 2
    H[NZ] = (-1j * (u[NZ] + L0 * u[NZ] ** 3) * A[NZ] - 1j * (v[NZ] + L0 *
v[NZ] ** 3) * B[NZ]) / delta[NZ]
    H[~NZ] = mean_depth
    Z = np.real(np.fft.ifft2(H))
    return Z

def show_images(images, titles, size=6):
    fig, axes = plt.subplots(1, len(images), figsize=(size * len(images),
size))
    if len(images) == 1:
        axes = [axes]
    for ax, img, title in zip(axes, images, titles):
        ax.imshow(img, cmap='viridis')
        ax.set_title(title)
        ax.axis('off')
    plt.show()

def norm(img):
    return ((img - np.min(img)) / (np.max(img) - np.min(img)) *
255).astype(np.uint8)

def get_surfaces(size=400):
    grid = np.mgrid[-size:size, -size:size].T.reshape(-1, 2)
    return (np.sin(grid[:, 0] / 60) * np.cos(grid[:, 1] / 60) / 8,
            (grid[:, 0]**2 + grid[:, 1]**2) / 1000000,
            np.exp(-(grid[:, 0]**2 + grid[:, 1]**2) / 32000))

def task_5_reconstruction():
    size = 400
    surface = np.zeros((size * size * 4, 3), dtype=np.float32)
    surface[:, :2] = np.mgrid[-size:size, -size:size].T.reshape(-1, 2) /
size
    surfaces = get_surfaces(size)

    for i, surf in enumerate(surfaces):
        surface[:, 2] = surf
        Z_orig = surface[:, 2].reshape((size * 2, size * 2))

        gy, gx = np.gradient(Z_orig)
```

```
        Z_frank = frankot_chellappa(gx, gy)
        Z_wei = wei_klette(gx, gy, 0.5, 0, 0, np.mean(Z_orig))
        Z_frank = norm(Z_frank)
        Z_wei = norm(Z_wei)

        images = [Z_frank, Z_wei]
        titles = [f"Frankot Chellappa {i}", f"Wei Klette {i} (L0=0.5,
L1=0, L2=0)"]

        for L0, L1, L2 in [(0.1, 0.5, 0.1), (0.5, 0.5, 0.1), (0.5, 1, 1)]:
            Z_wei = wei_klette(gx, gy, L0, L1, L2, np.mean(Z_orig))
            Z_wei = norm(Z_wei)
            images.append(Z_wei)
            titles.append(f"Wei Klette {i} (L0={L0}, L1={L1}, L2={L2})")

        show_images(images, titles, size=4)

task_5_reconstruction()
from sklearn.metrics import mean_squared_error
from skimage.metrics import structural_similarity as ssim
import numpy as np
import matplotlib.pyplot as plt

def compare_reconstructions(surfaces, size):
    results = []

    for i, surface in enumerate(surfaces):
        Z_orig = surface.reshape((size * 2, size * 2))
        gy, gx = np.gradient(Z_orig)

        Z_two_pass = reconstruct_two_pass(gx, gy)
        Z_two_pass_norm = norm(Z_two_pass)

        Z_frank = frankot_chellappa(gx, gy)
        Z_frank_norm = norm(Z_frank)

        Z_wei = wei_klette(gx, gy, 0.5, 1, 1, np.mean(Z_orig))
        Z_wei_norm = norm(Z_wei)

        mse_two_pass = mean_squared_error(Z_orig.flatten(),
Z_two_pass.flatten())
        mse_frank = mean_squared_error(Z_orig.flatten(),
Z_frank.flatten())
        mse_wei = mean_squared_error(Z_orig.flatten(), Z_wei.flatten())

        ssim_two_pass = ssim(Z_orig, Z_two_pass,
data_range=Z_two_pass.max() - Z_two_pass.min())
        ssim_frank = ssim(Z_orig, Z_frank, data_range=Z_frank.max() -
Z_frank.min())
        ssim_wei = ssim(Z_orig, Z_wei, data_range=Z_wei.max() -
Z_wei.min())
```

```
        results.append({
            "Surface": i,
            "MSE Two-pass": mse_two_pass,
            "MSE Frankot-Chellappa": mse_frank,
            "MSE Wei-Klette": mse_wei,
            "SSIM Two-pass": ssim_two_pass,
            "SSIM Frankot-Chellappa": ssim_frank,
            "SSIM Wei-Klette": ssim_wei
        })

        show_images(
            [norm(Z_orig), Z_two_pass_norm, Z_frank_norm, Z_wei_norm],
            [
                f"Original Surface {i}",
                f"Two-pass Reconstruction {i}",
                f"Frankot-Chellappa {i}",
                f"Wei-Klette {i} (L0=0.5, L1=1, L2=1)"
            ],
            size=4
        )

    for res in results:
        print(f"Surface {res['Surface']}:")
        print(f"  MSE Two-pass: {res['MSE Two-pass']:.4f}")
        print(f"  MSE Frankot-Chellappa: {res['MSE Frankot-
Chellappa']:.4f}")
        print(f"  MSE Wei-Klette: {res['MSE Wei-Klette']:.4f}")
        print(f"  SSIM Two-pass: {res['SSIM Two-pass']:.4f}")
        print(f"  SSIM Frankot-Chellappa: {res['SSIM Frankot-
Chellappa']:.4f}")
        print(f"  SSIM Wei-Klette: {res['SSIM Wei-Klette']:.4f}")
        print()

size = 400
surfaces = get_surfaces(size)
compare_reconstructions(surfaces, size)

import numpy as np
import matplotlib.pyplot as plt

def add_noise_to_gradients(gx, gy, noise_std=0.0001):

    gx_noisy = gx + np.random.normal(0, noise_std, gx.shape)
    gy_noisy = gy + np.random.normal(0, noise_std, gy.shape)

    return gx_noisy, gy_noisy

size = 400
surface = np.zeros((size * size * 4, 3), dtype=np.float32)
surface[:, :2] = np.mgrid[-size:size, -size:size].T.reshape(-1, 2) / size
```

```
surf = get_surfaces(size)

for chosen_surface in range(len(surf)):
    surface[:, 2] = surf[chosen_surface]
    Z_orig = surface[:, 2].reshape((size * 2, size * 2))

    gy, gx = np.gradient(Z_orig)

    gx_noisy, gy_noisy = add_noise_to_gradients(gx, gy)

    plt.figure(figsize=(12, 6))
    plt.subplot(1, 2, 1)
    plt.imshow(gx, cmap='viridis')
    plt.title(f'Original Gradient gx for Surface {chosen_surface}')
    plt.axis('off')

    plt.subplot(1, 2, 2)
    plt.imshow(gy, cmap='viridis')
    plt.title(f'Original Gradient gy for Surface {chosen_surface}')
    plt.axis('off')
    plt.tight_layout()
    plt.show()

    plt.figure(figsize=(12, 6))
    plt.subplot(1, 2, 1)
    plt.imshow(gx_noisy, cmap='viridis')
    plt.title(f'Noisy Gradient gx for Surface {chosen_surface}')
    plt.axis('off')

    plt.subplot(1, 2, 2)
    plt.imshow(gy_noisy, cmap='viridis')
    plt.title(f'Noisy Gradient gy for Surface {chosen_surface}')
    plt.axis('off')
    plt.tight_layout()
    plt.show()

from sklearn.metrics import mean_squared_error
from skimage.metrics import structural_similarity as ssim
import numpy as np
import matplotlib.pyplot as plt

def add_noise_to_gradients(gx, gy, noise_std=0.001):
    gx_noisy = gx + np.random.normal(0, noise_std, gx.shape)
    gy_noisy = gy + np.random.normal(0, noise_std, gy.shape)
    return gx_noisy, gy_noisy

def compare_reconstructions_with_noise(surfaces, size):
    results = []

    for i, surface in enumerate(surfaces):
        Z_orig = surface.reshape((size * 2, size * 2))
```

```
        gy, gx = np.gradient(Z_orig)

        gx_noisy, gy_noisy = add_noise_to_gradients(gx, gy)

        Z_two_pass = reconstruct_two_pass(gx_noisy, gy_noisy)
        Z_two_pass_norm = norm(Z_two_pass)

        Z_frank = frankot_chellappa(gx_noisy, gy_noisy)
        Z_frank_norm = norm(Z_frank)

        Z_wei = wei_klette(gx_noisy, gy_noisy, 0.5, 1, 1, np.mean(Z_orig))
        Z_wei_norm = norm(Z_wei)

        mse_two_pass = mean_squared_error(Z_orig.flatten(),
Z_two_pass.flatten())
        mse_frank = mean_squared_error(Z_orig.flatten(),
Z_frank.flatten())
        mse_wei = mean_squared_error(Z_orig.flatten(), Z_wei.flatten())

        ssim_two_pass = ssim(Z_orig, Z_two_pass,
data_range=Z_two_pass.max() - Z_two_pass.min())
        ssim_frank = ssim(Z_orig, Z_frank, data_range=Z_frank.max() -
Z_frank.min())
        ssim_wei = ssim(Z_orig, Z_wei, data_range=Z_wei.max() -
Z_wei.min())

        results.append({
            "Surface": i,
            "MSE Two-pass": mse_two_pass,
            "MSE Frankot-Chellappa": mse_frank,
            "MSE Wei-Klette": mse_wei,
            "SSIM Two-pass": ssim_two_pass,
            "SSIM Frankot-Chellappa": ssim_frank,
            "SSIM Wei-Klette": ssim_wei
        })

        show_images(
            [norm(Z_orig), Z_two_pass_norm, Z_frank_norm, Z_wei_norm],
            [
                f"Original Surface {i}",
                f"Two-pass Reconstruction {i}",
                f"Frankot-Chellappa {i}",
                f"Wei-Klette {i} (L0=0.5, L1=1, L2=1)"
            ],
            size=4
        )

    for res in results:
        print(f"Surface {res['Surface']}:")
        print(f"  MSE Two-pass: {res['MSE Two-pass']:.4f}")
```

```
        print(f"  MSE Frankot-Chellappa: {res['MSE Frankot-
Chellappa']:.4f}")
        print(f"  MSE Wei-Klette: {res['MSE Wei-Klette']:.4f}")
        print(f"  SSIM Two-pass: {res['SSIM Two-pass']:.4f}")
        print(f"  SSIM Frankot-Chellappa: {res['SSIM Frankot-
Chellappa']:.4f}")
        print(f"  SSIM Wei-Klette: {res['SSIM Wei-Klette']:.4f}")
        print()

size = 400
surfaces = get_surfaces(size)
compare_reconstructions_with_noise(surfaces, size)
```