Тема 6. Растрові алгоритми зафарбовування і заповнення областей

6.1. Основні поняття

Область можна задавати двома способами: шляхом виведення ліній контуру (границі області) (рис. 6.1, *а*) або зафарбувавши всі пікселі області (рис. 6.1, *б*).

а *б*

Рис. 6.1. Способи задання областей

Якщо область задається пікселями, що лежать усередині області, то всі пікселі з області мають одне й теж значення *old* і жоден піксель із границі такої області не має значення *old.* Такі області називаються *внутрішньо заданими*. Алгоритми, які працюють з такими областями так, щоб пікселі з атрибутом *old* перевести в пікселі зі значенням *new,* називаються *внутрішньо заповнюючими алгоритмами*.

Області, що задаються своїми замкнутими контурами (границями), називаються *гранично визначеними*. Пікселі, що належать границі, мають значення *bound*, а коди пікселів внутрішньої частини області відмінні від *bound.* Алгоритми заповнення таких областей називаються *гранично заповнюючими алгоритмами*. При цьому пікселі області потрібно зафарбувати в колір *new*, а колір контуру не повинен змінитися.

Користувач може інтерактивно визначати область, задавши границю із пікселів, що прилягають один до одного. Ці області потрібно зафарбувати кольором *new*. Після вибору області для рекурсивного алгоритму зафарбовування користувач повинен указати ще довільну затравочну точку, що належить області. Далі в цьому алгоритмі зафарбовування області потрібно прочитати сусідні пікселі, визначити, чи вони належать області і не зафарбовані, якщо так, то зафарбувати їх. При цьому виникає питання, які пікселі ($x1$, $y1$), ($x2$, $y2$) можна вважати сусідніми.

Важливим поняттям растрових областей є їх зв’язність – можливість з’єднати два пікселі області растровою лінією, тобто послідовним набором пікселів, що належать цій області. Використовуючи зв’язність можна, рухаючись від точки затравки, досягти і зафарбувати всі пікселі області.

За способом доступу до сусідніх пікселів області поділяються на 4-зв’язні та 8-зв’язні.

4-зв'язними називають ті області, в яких кожен піксель області може бути досягнутий з іншого пікселя області за допомогою комбінацій переміщень на один піксель у чотирьох напрямках: горизонтальних (вліво, вправо), або вертикальних (вверх, вниз), тобто якщо координати сусідніх пікселів задовольняють умову

$$|x1 - x2| + |y1 - y2| \le 1.$$

Такі пікселі називають ще околом фон Неймана.

У 8-зв'язних областях до цих напрямків додаються ще і діагональні, тобто *8-зв'язними* називаються області, в яких кожний піксель може бути досягнутий з іншого пікселя цієї ж області за допомогою послідовності переміщень на один піксель у восьми напрямках (рис. 6.2). У цьому випадку сусідніми вважаються пікселі, якщо їх координати відрізняються відповідно не більше ніж на одиницю, тобто

$$|x1 - x2| \le 1 \text{ і } |y1 - y2| \le 1.$$

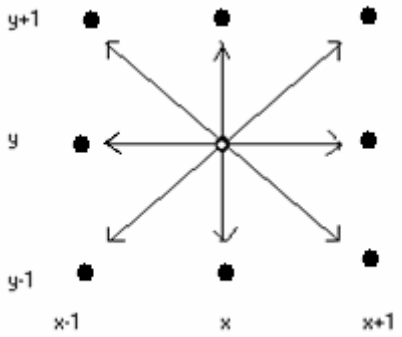


Рис. 6.2. Сусідні пікселі для 8-зв'язної області (окіл Мура)

Із наведених означень випливає, що довільних два 4-зв'язних пікселі є 8-зв'язними, але не навпаки. Тому, в загальному випадку 4-зв'язну область можна заповнити як 4-зв'язними так і 8-зв'язними алгоритмами, але не навпаки, Тобто 8-зв'язну область не завжди можна заповнити 4-зв'язним алгоритмом. Розглянемо два квадрати зображені на рис. 6.3. За алгоритмом заповнення для 4-зв'язних областей буде заповнено лише один квадрат, а алгоритм для 8-зв'язних областей заповнить два квадрати.

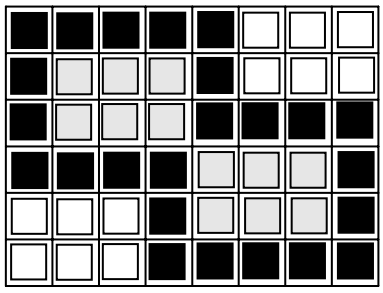


Рис. 6.3. 8-зв'язна область

Методи заповнення областей можна поділити на дві групи: методи заповнення з затравочною точкою та методи растрової розгортки.

У методах заповнення з точкою затравки заповнення починається з деякої точки, що знаходиться всередині замкнутого контуру (затравочна точка). Далі відшуковуються сусідні з нею точки, які розміщені всередині контуру. Якщо така точка знайдена, то вона стає новою затравочною точкою і для неї рекурсивно ведеться пошук нових точок. Подібні алгоритми використовуються тільки для растрових пристроїв.

У методах растрової розгортки в порядку сканування рядків визначають чи точка лежить всередині контуру і якщо так, то її зафарбовують.

6.2. Рекурсивні алгоритми зафарбовування областей

Наведемо найпростіший рекурсивний алгоритм зафарбовування 4-зв'язної гранично заданої області [34].

Pixel_Fill (x, y, bound, new);

// (x,y) – координати початкової точки, з якої почнеться зафарбовування*, bound* – колір границ*i, new* – новий колір області

якщо точка (*x*, *y*) не належить контуру і не зафарбована в колір *new, то*

початок

Зафарбувати точку (*x, y*) у колір *new;*

Pixel_Fill (x + 1*, y, bound, new);*

Pixel_Fill (x, y + 1*, bound, new);*

Pixel_Fill (x, y – 1*, bound, new);*

Pixel_Fill (x – 1*, y, bound, new);*

кінець.

Аналогічний алгоритм маємо для внутрішньо заданої області [34].

Flood_Fill_4 (x, y, old, new);

//(x,y) – координати затравочної точки, *old* – значення атрибуту пікселів, що задають область*, new* – нове значення атрибуту пікселя

якщо getpixel (x, y) = old то

початок

Зафарбувати точку *(x, y)* у колір *new;*

Flood_Fill_4 (x +1*, y, old, new);*

Flood_Fill_4 (x – 1*, y, old, new);*

Flood_Fill_4 (x, y +1*, old, new);*

Flood_Fill_4 (x, y –1*, old, new);*

кінець.

Рекурсивні алгоритми досить прості. Вони абсолютно коректно зафарбовують найскладніші області, але неефективні по тій причині, що функції зафарбовування викликаються до вже зафарбованих пікселів і,

крім цього, вимагають великого стеку через значну глибину рекурсії. Практика показує, що ці алгоритми неприйнятні для областей із тисячею і більше пікселів, оскільки відбувається переповнення стеку.

Тому для розв'язання задачі зафарбування перевагу надають алгоритмам, які обробляють цілі групи пікселів.

6.3. Пострічковий алгоритм зафарбовування із затравкою

Розглянемо версію одного з найпопулярніших алгоритмів зафарбовування з затравкою. Ідея цього алгоритму полягає в тому, що для заданої точки (x, y) з області визначається і заповнюється максимальний відрізок [xl, xr], що лежить усередині області і містить цю точку. Після цього перевіряються відрізки області, що лежать вище і нижче знайденого відрізка. Якщо такі пікселі знаходяться, то рекурсивно викликається функція для їх обробки. Розглянемо алгоритм більш детально.

На кожній горизонталі пікселі, які потрібно зафарбувати, складають горизонтальні інтервали, що лежать усередині області. Ці інтервали відокремлені один від одного інтервалами з пікселів, які належать межі або зовнішності області.

Якщо набір пікселів утворює зв'язний інтервал, який лежить всередині області, то пікселі над і під цим інтервалом є або межовими, або лежать усередині області. Останні можуть служити затравочними пікселями для рядків, що лежать вище та нижче від активного рядка.

Нехай (x, y) – координати початкового пікселя, тоді пострічковий алгоритм має такий вигляд:

Line_Fill (x,y);
початок
Знайти крайню ліву точку xl інтервалу;
Знайти крайню праву точку xr інтервалу;
Заповнити інтервал від xl до xr;
Для всіх x від xl до xr виконати
якщо точка (x, y +1) не належить межі і не зафарбована,
то Line_Fill (x, y +1);
Для всіх x від xl до xr виконати
якщо точка (x, y – 1) не належить межі і не зафарбована,
то Line_Fill (x, y – 1)
кінець.

Цей алгоритм також є рекурсивним, але оскільки функція Line_Fill викликається для лінії в цілому, а не для кожного пікселя, то це зменшує кількість вкладених викликів, а отже, зменшуються витрати стекової пам'яті. Алгоритм ефективно працює і для областей із дірками (рис. 6.4).

<u>Зауваження 1.</u> Для зафарбовування простіших фігур (прямокутник, еліпс) можна використати алгоритми виведення контуру, які розглядались вище. У процесі виконання цих алгоритмів послідовно обчислюються координати пікселів контуру. Для зафарбовування необхідно виводити горизонталі, які з'єднують пари точок на контурі, що розміщуються симетрично відносно вертикальної осі (рис. 6.5).

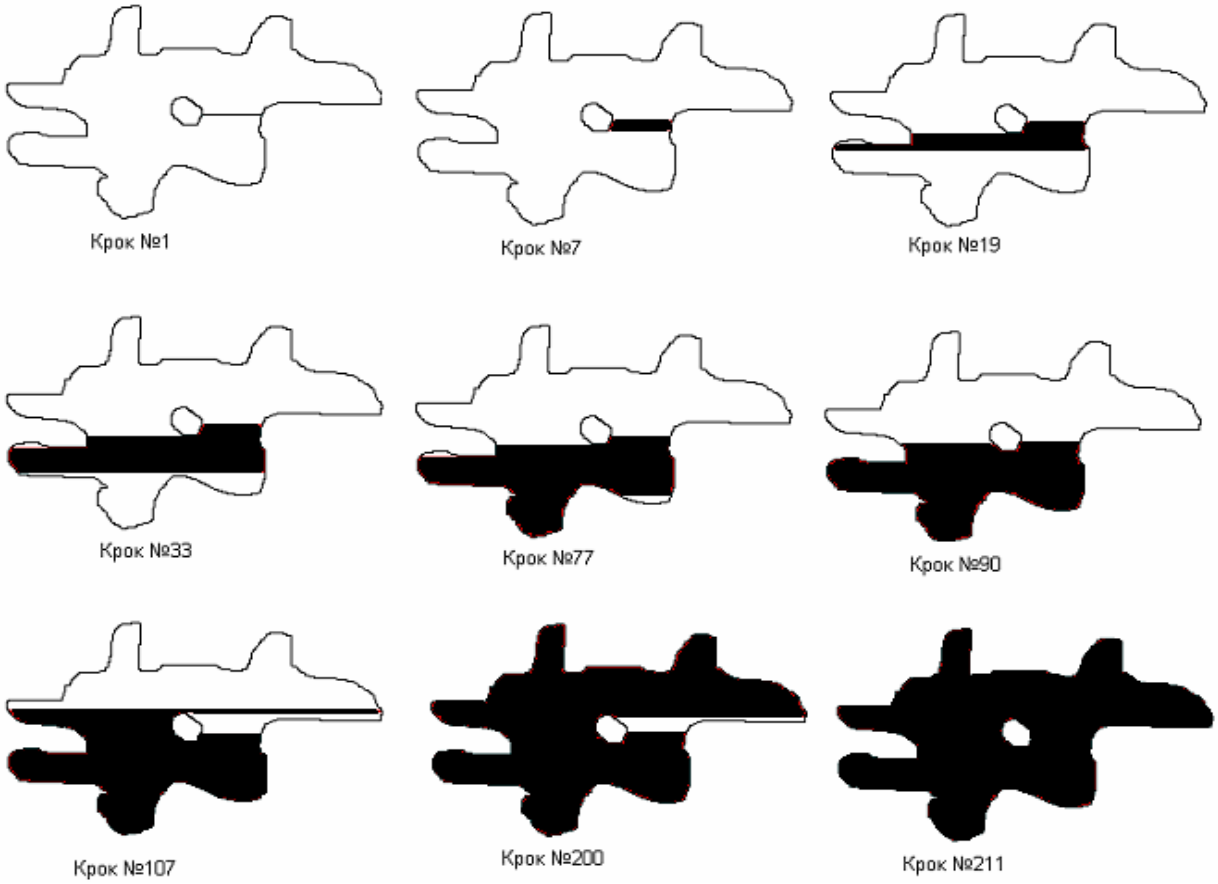


Рис. 6.4. Пострічковий алгоритм зафарбовування областей

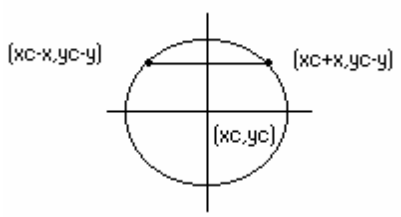


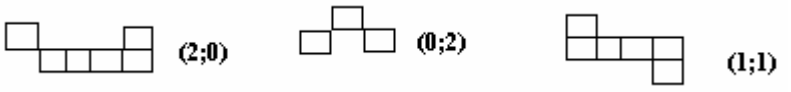
Рис. 6.5. Зафарбовування круга

6.4. Алгоритм зафарбовування області за критерієм парності

Опишемо основну ідею алгоритму. В циклі по *y* проводяться горизонтальні прямі. Точкам перетину дуги контуру з горизонтальною прямою ставиться у відповідність певна структура даних, що визначає характер перетину дуги з горизонтальною лінією. Ця структура даних визначає чи точки горизонталі є внутрішніми чи зовнішніми по відношенню до розглядуваної області. Так стає відомо, які пікселі потрібно зафарбовувати.

Введемо в розгляд змінні *вище, нижче*, які набуватимуть значень, що описують характер перетину горизонтальної прямої з дугою контуру. Точки контуру, що відповідають локальним екстремумам контуру, матимуть порядки (0, 2) або (2, 0) (рис. 6.6, *а*, *б*).

Якщо контур не містить кратних пікселів і точка контура не є екстремумом, то такій точці відповідає структура даних (1, 1) (рис. 6.6, *в*).



а *б* *в*

Рис. 6.6. Структури даних

Поява ознаки помилки при виконанні алгоритму сигналізує про структури даних, які відрізняються від допустимих пар значень (1, 1), (2, 0), (0, 2).

Для визначення порядків граничних точок використовується процедура Link. Уведемо в розгляд лічильник *l,* значення якого вказує на кількість перетинів контуру області з розглядуваною горизонтальною прямою. Крім цього для зменшення обчислювальних затрат визначимо прямокутну оболонку $(x_{min}, y_{min}) - (x_{max}, y_{max})$, що містить нашу область.

Алгоритм заповнення області за критерієм парності має вигляд [22].

Для кожного y від $y = y_{min}$ *до* $y = y_{max}$ *виконати*
початок
Встановити значення лічильника в 0, l=0*;*
Присвоїти x значення лівого краю сітки $x = x_{min}$ *;*
поки $x \le x_{max}$ *виконувати*
початок
якщо (x,y) не належить контуру, *то*
початок
якщо значення лічильника l непарне, *то*
Зафарбувати *(x,y);*
Збільшити x: x:=x+1
кінець
інакше
початок
Link (x,y, вище, нижче);
якщо значення *вище, нижче* = 1, *то* збільшити лічильник l:=l + 1;
якщо вище + нижче ≠ 2 *то* ознака помилки
кінець
кінець
кінець.

Функцію Link визначимо наступним чином.

Link (x,y, вище, нижче)

// *x,y* – вхідна інформація: координати пікселя, що належить контуру

// *вище, нижче* – вихідна інформація (значення верхнього та нижнього порядків точки контуру)

початок

Встановити значення *вище, нижче* в нуль*;*

якщо (*x* – 1, *y* + 1) належить контуру*, то вище: = вище* +1;

якщо (*x* – 1, *y* – 1) належить контуру*, то нижче: = нижче* +1;

поки (*x, y*) належить контуру *виконувати*

початок

якщо (*x, y* +1) належить контуру, а (*x* –1, *y* + 1) не належить контуру*, то вище: = вище* +1;

якщо (*x, y* – 1) належить контуру, а (*x* –1, *y* – 1*)* не належить контуру*, то нижче: = нижче* +1*;*

збільшити x

кінець

якщо (*x* –1, *y* + 1) не належить контуру*,* а (*x, y*+ 1) належить контуру*, то вище = вище* + 1;

якщо (*x* –1, *y* – 1) не належить контуру, а (*x, y* – 1) належить контуру*, то нижче = нижче* + 1

кінець.

Зауваження 2. Наведений алгоритм спрощується для полігонів. Для цього замість процедури Link потрібно визначити характер перетину горизонтальних ліній з ребрами полігону. Якщо точка перетину є вершиною локального екстремуму, то зараховуємо дві точки перетину, інакше – одну.

6.5. Зафарбовування полігонів. YX-алгоритм

Адаптуємо наведені вище алгоритми на випадок, коли область є многокутником (полігоном). Основна ідея алгоритму – зафарбовування полігона відрізками горизонталей.

Нехай контур полігона задається множиною координат вершин $P_i(x_i, y_i)$, $i = 0, 1,..., n$, які послідовно з'єднані відрізками прямих.

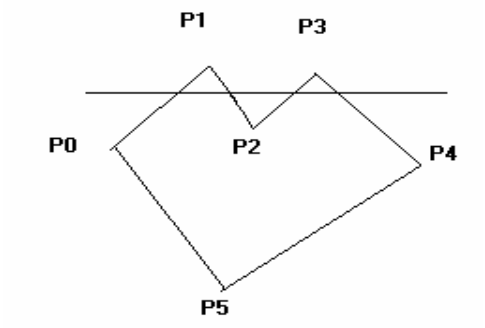


Рис. 6.7. YX-алгоритм зафарбовування полігона

В алгоритмі здійснюється цикл по y, на кожному кроці якого знаходяться точки перетину відрізків контуру полігона (ребер) з відповідними горизонталями (рис. 6.7). Цей алгоритм називається YX-алгоритмом і має такий вигляд:

початок

Знайти $min\ (y_i) = y_{min}$ і $max\ (y_i) = y_{max}$ серед всіх вершин P_i;

Для y від $y = y_{min}$ *до* $y = y_{max}$ *виконувати*

початок

Знайти точки перетину усіх відрізків контуру з горизонталлю y;

Координати точок перетину записати в масив X;

Упорядкувати масив X по зростанню координат x_i;

Вивести горизонтальні відрізки з координатами кінців

$(x_0, y) - (x_1, y)$; $(x_2, y) - (x_3, y)$; . . . ; $(x_{2к}, y) - (x_{2к+1}, y)$

кінець

кінець.

Довільна пряма лінія перетинає замкнений контур парну кількість разів, тобто в масив X записується парна кількість точок перетину. Опуклі фігури з довільною прямою завжди мають дві точки перетину.

При знаходженні точок перетину горизонталі з контуром необхідно брати до уваги особливі точки (вершини полігона). Якщо горизонталь має координату y, що збігається з y_i вершини P_i, то необхідно аналізувати, як горизонталь проходить через вершину P_i. Якщо вершина P_i є локальним максимумом або мінімумом, як, наприклад, P_1, P_2, P_3, P_5 (рис. 6.7), то вершина P_i не записується в масив X або записується 2 рази. Якщо горизонталь перетинає вершину P_i так, що $(y_{i-1} - y_i) * (y_{i+1} - y_i) < 0$ (наприклад, як вершини P_0, P_4 (рис. 6.7)), то в масив X ця вершина записується один раз. Якщо горизонталь зустрічається з горизонтальним відрізком полігона P_iP_{i+1}, то необхідно аналізувати добуток $(y_{i-1} - y_i)*(y_{i+2} - y_i)$.

Цей алгоритм можна оптимізувати, беручи до уваги той факт, що кожна горизонталь у більшості випадків перетинає не всі ребра, а лише невелику кількість ребер контуру. Тому, якщо зробити попередній відбір ребер, що перетинаються з горизонталлю, то можна досягнути зменшення кількості обчислень в алгоритмі.

Наприклад, розділимо діапазон $[y_{min}, y_{max}]$ навпіл точкою $y_{ср}$. Якщо для діапазону $[y_{min}, y_{ср}]$ скласти список вершин (ребер), для яких y-координата належить цьому діапазону, то для горизонталей з $y \in [y_{min}, y_{ср}]$ будемо перевіряти перетин тільки із цим списком ребер (їх буде приблизно у 2 рази менше). Аналогічно для діапазону $[y_{ср}, y_{max}]$ також складаємо список ребер (їх теж буде приблизно вдвічі менше).

Контур можна ділити не навпіл, а на менші частини. Такий спосіб підвищення швидкодії ефективний для великої кількості вершин.

Наведені вище алгоритми можуть бути використані не тільки для зафарбовування фігур, а й для розв'язування інших задач, наприклад, для заповнення фігур, знаходження площі фігури, центру ваги та ін.

6.6. Заповнення фігур. Текстури

Задача заповнення областей полягає в тому, щоб дану область зафарбувати не суцільно, а тільки її частини (можливо, різними кольорами).

При виведенні зображення фігур можуть використовуватись різні стилі заповнення. Для визначення стилю заповнення фігур, відмінного від суцільного зафарбовування, використовують поняття текстури, хоча воно більшою мірою застосовується для тривимірних об'єктів. *Текстура* – це стиль заповнення, що імітує складну поверхню.

Для заповнення фігур певним стилем використовують ті ж самі алгоритми, що і при зафарбовуванні фігур. Суцільне зафарбовування означає, що колір кожного пікселя області однаковий (*color = const*), тобто алгоритми зафарбовування фігур обов'язково містять оператор

виведення пікселя постійного кольору color із координатами (*x, y*).

А для того, щоб фігуру заповнити певним візерунком, необхідно змінювати колір пікселів заповнення, тобто в алгоритмі заповнення фігури перед оператором виведення пікселя (*x*, *y*) кольору *color* потрібно поставити оператор *color = f*(*x*, *y*), який буде визначати колір *color* для кожного пікселя зокрема, а отже, алгоритми заповнення будуть містити оператори: *color = f(x, y); виведення пікселя (x, y) кольору color.*

Функція *f(x,y)*, аргументами якої є координати поточного пікселя, визначає стиль заповнення. Якщо значеня функції *f(x, y)* генерувати випадковим чином, тобто значення *color = random*( ), то можна створити ілюзію матової поверхні (рис. 6.8), а якщо за *f(x, y)* взяти функцію

$$f(x, y) = \begin{cases} C_ш, \text{ якщо } (x+y) \bmod S < T; \\ C_ф, \text{ для решти } x, y; \end{cases},$$

то можна заповнити область широкими штриховими лініями (рис. 6.8) .

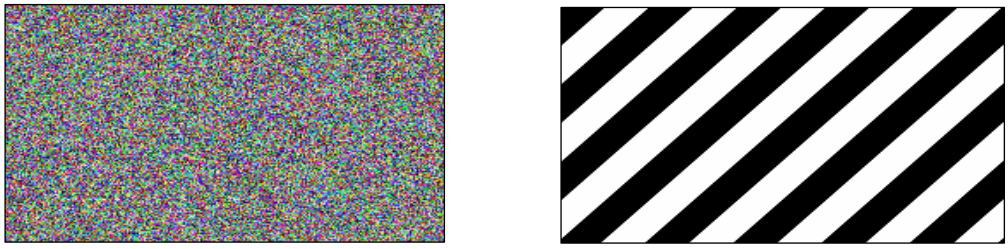


Рис. 6.8. Матова та штрихова фігури

Тут параметр T задає товщину штрихів, S – період нанесення штрихів, $C_ш$ – колір штрихових ліній, $C_ф$ – колір фону. Якщо не задавати колір фону, то можна створити ілюзію напівпрозорої фігури.

Часто при заповненні фігур використовується копіювання невеликих растрових зразків-візерунків (текселів) на всю область фігури. Нехай маємо растр із $m * n$ пікселів, що складає зразок текстури. При цьому координати пікселів структури $x_т$, $y_т$ можуть набувати значень $x_т = 0, 1,..., m - 1$, $y_т = 0, 1,..., n - 1$. Заповнення фігури можна забезпечити шляхом циклічного копіювання фрагмента зразка всередині області заповнення фігури.

Для цього потрібно оператор $color = f(x, y)$ замінити такою послідовністю операцій:

- для координат (x, y) пікселя заповнення обчислити відповідні їм координати $(x_т, y_т)$ зразка заповнення;
- за координатами $(x_т, y_т)$ визначити колір *color* пікселя зразка текстури;
- вивести піксель з координатами (x, y) і з кольором color.

Обчислення координат $(x_т, y_т)$ через координати (x, y) можна здійснити за формулами

$$x_т = x \bmod m, \qquad y_m = y \bmod n,$$

де m, n – розміри растра-зразка, mod – функція, що визначає залишок від ділення. Якщо m та n є степенем 2, тоді функцію mod можна замінити більш швидкодіючими функціями. Приклад заповнення області текселами зображено на рис. 6.9.

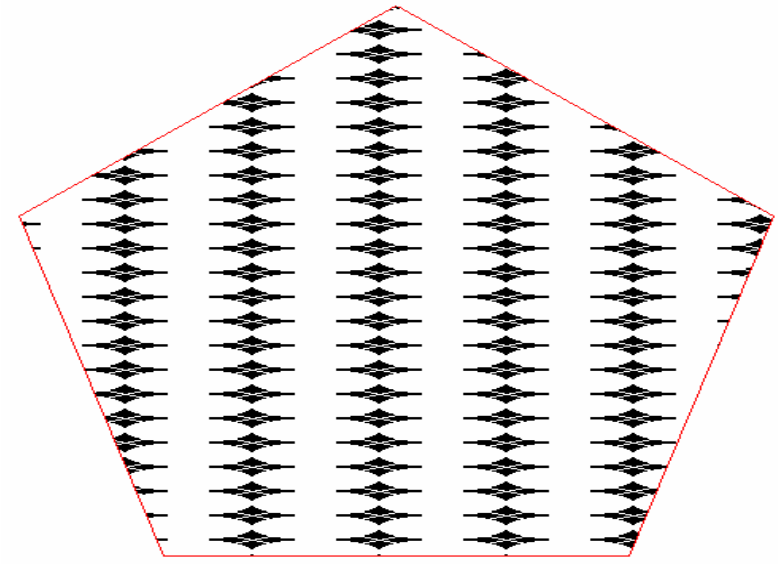


Рис. 6.9. Заповнення області деякими зразками

6.7. Обробка растрових зображень

Обробка растрових зображень передбачає перетворення растрових даних. Існує багато різних варіантів обробки растрових зображень. Можна виділити три основних класи алгоритмів обробки растрових даних [7]:

- точкові алгоритми, в яких значення пікселів змінюються на основі значень самих пікселів та їх координат;
- просторові алгоритми, в яких значення пікселів змінюється на основі вихідних значень самих пікселів і пікселів навколо них;
- алгоритми геометричних перетворень, коли розміщення пікселів в зображенні змінюється на основі геометричних перетворень.

Для кожного з цих класів розроблено цілий ряд алгоритмів, причому застосування цих алгоритмів некомутативне, тому важливим є порядок їх застосування.

Розглянемо деякі поняття, що лежать в основі цих алгоритмів. Нехай a_{ij} елемент зображення, тобто піксель з координатами (i, j), тоді $A = \{ a_{ij} \}$, $i = 0, 1,\ldots, m, j = 0, 1, \ldots, n$ растрове зображення, тобто матриця пікселів.

Якщо $a_{ij} \in \{0, 1\}$, то растрове зображення називається бінарним і складається тільки з чорних та білих пікселів.

Якщо $a_{ij} \in \{0, 1, \ldots, N-1\}$, то растрове зображення називається на півтоновим і кожний піксель може приймати N відтінків сірого (градацій яскравості).

Якщо $a_{ij} \in \left\{ \begin{pmatrix} x_{ij}^1 \\ x_{ij}^2 \\ x_{ij}^3 \end{pmatrix} \right\}$, де $x_{ij}^k \in \{0, 1, \ldots, N-1\}$, то зображення називається кольоровим. Колір кожного пікселя визначається координатами (x^1, x^2, x^3) в просторі кольорів.

Розглянемо алгоритмічні основи обробки бінарних і напівтонових зображень.

Точкові алгоритми. Точкові алгоритми прості і найбільш часто застосовуються в алгоритмах обробки зображень. Вони застосовуються для бінарних і напівтонових зображень. В цих алгоритмах для зміни яскравості пікселя в зображенні використовується тільки вихідна яскравість цього пікселя, інколи координати цього пікселя; ніякі інші значення не використовуються.

Нові значення яскравості обчислюються на основі деякого алгоритму. Точкові алгоритми сканують зображення піксель за пікселем, здійснюючи перетворення точок зображення. Якщо перетворення залежить тільки від яскравості пікселів, то процес обробки зображень краще реалі-

зувати на основі таблиць перетворення. Якщо перетворення в точкових алгоритмах враховує і розміщення пікселів, то воно задається формулою. Точкові алгоритми не можуть модифікувати деталі зображення.

Найпростіші алгоритми цього класу це алгоритми побудови негативних зображень, або алгоритми інверсії яскравості пікселів. Негативне зображення одержується шляхом віднімання вихідного значення яскравості $f(x, y)$ кожного пікселя зображення від максимально можливого значення яскравості f_{max}:

$$g(x, y) = f_{max} - f(x, y),$$

де $g(x, y)$ – нове значення яскравості пікселя (x, y). Негативні зображення необхідні для виділення яскравих частин зображення, оскільки людське око краще сприймає деталі в темній області зображення, ніж в світлій.

Друга важлива група алгоритмів цього класу це алгоритми бінаризації – напівтонових зображень, тобто перетворення їх в чорно-біле (бінарне) зображення. Ці алгоритми базуються на пороговій обробці напівтонових зображень шляхом розділення всіх пікселів зображення на два класи за ознакою яскравості: пікселі об'єкта і фону. Формула перетворення яскравості в алгоритмах бінаризації має вигляд

$$g(x, y) = \begin{cases} f_{\max}, \textit{якщо } f(x, y) \geq f_0; \\ f_b, \textit{якщо } f(x, y) < f_0, \end{cases}$$

де f_0 – деяке порогове значення яскравості (рис. 6.10).

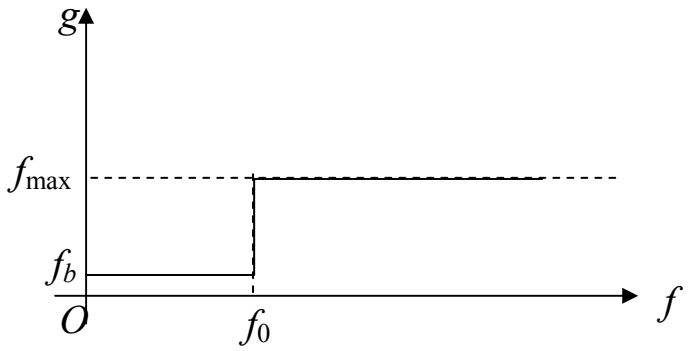


Рис. 6.10.

При цьому важливо правильно вибрати порогове значення f_0, щоб не втратити корисну інформацію. Особливо це суттєво, коли зображення містить деякий шум. Такі перетворення виконують для того щоб в зображенні залишити тільки ту інформацію, яка необхідна для розв'язування задач, наприклад для знаходження контурів об'єкта.

За допомогою точкових алгоритмів можна розв'язати задачу збільшення контрастності зображення. Зображення з низьким контрастом можуть бути як надто світлими так і надто темними. Зображення з високим контрастом мають як темні так і світлі області, тобто використовують

весь діапазон яскравості. Зображення з низьким контрастом складається з тонів обмеженого діапазону яскравості. Задача збільшення контрасту полягає в розтягуванні діапазону яскравості вихідного зображення на всю шкалу [f_b, f_{max}].

Цю задачу можна розв'язати за допомогою точкового перетворення яскравості за формулою

$$g(x, y) = af(x, y) + b$$

де a, b підібрані коефіцієнти.

Просторові алгоритми. В просторових алгоритмах використовується інформація про групи пікселів, на відміну від точкових алгоритмів у яких використовується інформація тільки про один піксель.

Група пікселів зображення, які приймають участь в просторових алгоритмах називається *областю примикання*. Область примикання визначається матрицею значень яскравості пікселів з непарною кількістю рядків і стовпців. Пікселі, в яких старе значення яскравості замінюється на нове, розміщені в центрі області притягання.

Більшість просторових алгоритмів використовують фільтри. Фільтр зручно задавати матрицею чисел малого розміру (3 × 3, або 5 × 5), яку називають *ядром згортки*. Розміри, структура ядра згортки і значення коефіцієнтів, що містяться в ядрі визначають тип просторового алгоритму, дозволяють вплинути на значення пікселя зображення і одержати різні спецефекти.

Більший розмір матриці підвищує гнучкість процесів згортки, однак виникають труднощі з примежовими пікселями, оскільки маска згортки для них виходить за межі зображення. Тому на практиці обмежуються невеликими розмірами матриці, та додатково опрацьовують пікселі на краях зображення.

Щоб перетворити один піксель в зображенні, значення його яскравості множиться на число в центрі ядра, а яскравості пікселів, що оточують центральний піксель множаться на відповідні коефіцієнти ядра. Потім ці всі добутки сумуються і в результаті одержується нове значення яскравості для центрального пікселя.

Цей процес повторюється для кожного пікселя зображення (так фільтрується зображення). Коефіцієнти ядра визначають результат процесу фільтрування. Якщо сума коефіцієнтів більша за одиницю, то яскравість збільшується; якщо сума менша за одиницю, то яскравість зменшується.

Найбільш широко розповсюджені просторові алгоритми обробки растрових зображень – це алгоритми розмивання, збільшення чіткості, тиснення і акварельний ефект.

В *алгоритмі розмивання* в зображенні пом'ягшуються різкі границі за рахунок перерозподілу яскравості, тобто шляхом усереднення швидких змін яскравості (рис. 6.11, *б*). Ядро розмивання складають коефіцієнти менші за одиницю, а їх сума дорівнює одиниці. Це означає, що в результаті фільтрації кожний піксель вбирає інформацію від сусідніх пікселів, але повна яскравість зображення залишається незмінною. Результуюче зображення таким чином буде більш розмитим порівняно з оригіналом. Ядро розмивання, наприклад, має вигляд

$$\begin{pmatrix} 0{,}05 & 0{,}05 & 0{,}05 \\ 0{,}05 & 0{,}6 & 0{,}05 \\ 0{,}05 & 0{,}05 & 0{,}05 \end{pmatrix} \quad \text{або} \quad \begin{pmatrix} 0{,}1 & 0{,}1 & 0{,}1 \\ 0{,}1 & 0{,}2 & 0{,}1 \\ 0{,}1 & 0{,}1 & 0{,}1 \end{pmatrix}.$$

Ступінь розмивання можна змінити

- збільшенням розміру ядра;
- підбором коефіцієнтів ядра зі зменшенням впливу центрального коефіцієнта;
- проведенням багатокрокової фільтрації з ядром розмивання.

Зауважимо, що у випадку кольорових зображень ядро розмивання застосовується до червоної, зеленої і синьої компонент кольору кожного пікселя зображення.

В *алгоритмах збільшення чіткості* застосовується інше ядро, оскільки необхідно збільшити чіткість зображення, тобто підкреслити різницю між яскравостями сусідніх пікселів і виділити непомітні деталі (рис.6.11, *в*).

В ядрі чіткості центральний коефіцієнт більший за одиницю, а решту елементів ядра від'ємні числа, сума яких на одиницю менша, ніж центральний коефіцієнт. Це означає, що при обробці зображення з великими змінами яскравості нове значення яскравості центрального пікселя різко збільшується, тобто великі зміни яскравості збільшуються, а області постійної яскравості залишаються незмінними. Результуюче зображення одержується більш чітким, ніж оригінал. При повторній обробці чіткість знову може бути збільшена, але при цьому ніякі нові деталі з нічого не з'являться.

При обробці пікселів в зображенні застосовуються такі ядра чіткості (високочастотні маски):

$$\begin{pmatrix} -0{,}1 & -0{,}1 & -0{,}1 \\ -0{,}1 & 1{,}8 & -0{,}1 \\ -0{,}1 & -0{,}1 & -0{,}1 \end{pmatrix}, \quad \begin{pmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{pmatrix}, \quad \begin{pmatrix} 1 & -2 & 1 \\ -2 & 5 & -2 \\ 1 & -2 & 1 \end{pmatrix}.$$

Знову, як і раніше у випадку кольорового зображення червона, зелена і синя складові обробляються маскою окремо, а потім об'єднуються, щоб сформувати 24-бітне значення кольору.

Алгоритми тиснення перетворюють зображення так, що об'єкти сцени виглядають видавленими на деякій поверхні (рис. 6.11, *г*). Тиснення виконується за допомогою ядра згортки так, як і в попередніх алгоритмах. Кожен піксель зображення обробляється ядром тиснення розміру 3×3.

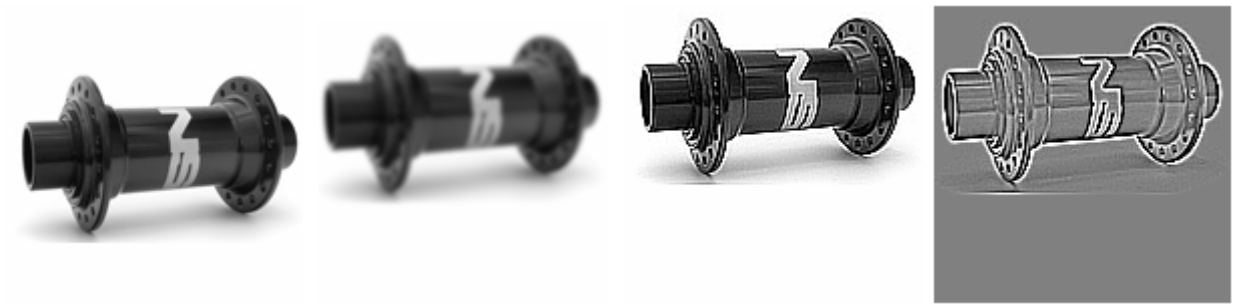
Наведемо приклади ядра тиснення

$$\begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix}, \quad \begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix}.$$

Зауважимо, що сума коефіцієнтів в ядрі тиснення дорівнює нулю. Це означає, що тим пікселям, які не знаходяться на границях переходу від від одного кольору до іншого присвоюються нульові значення, а тим що знаходяться на таких границях ненульові значення. Тобто, якщо всі дев'ять пікселів, що знаходяться в області матриці тиснення мають однакові яскравості, то значення центрального пікселя після перетворення стане рівним нулю (чорний колір).

Після обробки пікселя ядром тиснення до одержаної яскравості (у випадку кольорового зображення до кожної складової R, G, B) добавляють число 128. Суми, що перевищують 255 заокруглюються до 255.

Аналогічно конструюються маски для акварельних зображень.



а *б* *в* *г*

Рис. 6.11. Обробка зображень: *а* – вихідне зображення, *б* – розмивання зображення, *в* – збільшення чіткості, *г* – тиснення

До просторових алгоритмів відносяться ще *медіанні фільтри*, за допомогою яких можна відфільтрувати шумові ефекти. Ці алгоритми не працюють за принципом згортки, тобто нове значення яскравості пікселя не обчислюється шляхом матричного множення, а замість цього всі пікселі в області примикання сортуються в зростаючому порядку за яскравістю пікселів і вибирається середнє значення яскравості для нового значення розглядуваного пікселя. Наприклад, потрібно змінити значення яскравості центрального пікселя в такому вікні

$$\begin{pmatrix} 43 & 75 & 50 \\ 57 & 126 & 50 \\ 24 & 83 & 50 \end{pmatrix}.$$

Відсортувавши значення пікселів, одержимо послідовність 24, 43, 50, 50, 50, 57, 75, 83, 126 в якій 50 є середнім значенням. В результаті середнє значення яскравості 50 замінить 126 у вихідному зображенні, тобто випадковий шум, що міститься в зображенні буде усунено.

Алгоритми геометричних перетворень. Алгоритми геометричних перетворень растрових зображень змінюють місце розміщення і/або структуру пікселів зображення на основі геометричних перетворень. Геометричні перетворення не завжди змінюють яскравість елементів зображення, але вони завжди змінюють розміщення пікселів зображення, тобто значення яскравості пікселів займають нову позицію.

Геометричні перетворення растрових даних широко застосовуються, наприклад при розпізнаванні образів, накладанні текстури, забезпеченні різних ефектів у зображеннях. Геометричні перетворення растрових даних – це переміщення, масштабування, зсув і поворот зображення. Ці перетворення будуть вивчатимуться далі.

Контрольні питання та завдання

1. Які є способи задання областей?
2. Які області називаються 4-зв’язними/8-зв’язними?
3. Як працюють рекурсивні алгоритми заповнення областей?
4. В чому переваги пострічкового алгоритму?
5. Запишіть і поясніть роботу процедури Line_Fill.
6. Які структури даних використовуються в алгоритмі заповнення за критерієм парності?
7. Поясніть роботу процедури Link.
8. В чому полягає основна ідея YX-алгоритму?
9. Чим відрізняється заповнення фігур від зафарбовування?
10. Як заповнити фігуру деяким узором?
11. Як одержати кольоровий негатив графічного зображення?

Вправи і задачі для самостійного виконання

1. Знайти небажані ефекти, які можуть виникнути при заповненні 4-зв'язної області 8-зв'язним алгоритмом?
2. Записати рекурсивні алгоритми зафарбовування 8-зв'язних областей.
3. Визначте на вашому комп'ютері максимальну кількість пікселів області, яку ще вдається заповнити рекурсивними алгоритмами.
4. Знайти порядок зафарбовування простої 4-зв'язної гранично-заданої області в рекурсивному алгоритмі з затравкою.
5. Записати пострічковий алгоритм зафарбовування полігона.
6. Оптимізувати процедуру Link для полігонів.
7. Відобразити растрове зображення з прямокутної області з протилежними вершинами (0, 0), (x_0, y_0) в довільну чотирикутну область з вершинами (u_0,v_0), (u_1,v_1), (u_2,v_2), (u_3,v_3) і навпаки. *Вказівка*. *Закон перетворення прямокутної області в чотирикутну виразити квазілінійним співвідношенням виду:*

$$u=a_1x+b_1xy+c_1y+d_1$$
$$v=a_2x+b_2xy+c_2y+d_2.$$

8. Розробити алгоритм створення ефекту хвиль на чорно-білому зображенні.
9. Розробити алгоритм побудови контурів предметів на чорно-білому зображенні.
10. Розробити алгоритм морфологічного розширення зображень *A* та *B*. Під морфологічним розширенням розуміють операцію

$$A(+)B = \{t\in R^2: t = a + b, a\in A, b\in B\}.$$