Лабораторна робота №2. Інкрементні алгоритми

Завдання

1. Створити програму із зручним інтерфейсом користувача, що буде реалізовувати можливість побудови рисунку відповідно до варіанта завдання.
2. Єдиною графічною операцією, що дозволяється використовувати, є операція побудови точки. Всі інші графічні елементи — лінії, що утворюють багатокутники контурів символів, повинні будуватися із використанням інкрементних алгоритмів Брезенхема.
3. Реалізувати в програмі можливість діалогового режиму вибору кольору зображення.

Варіанти завдань

№ варіанта	**Завдання**
1	Перша літера вашого прізвища та цифра 1
2	Перша літера вашого імені та цифра 2
3	Перша літера вашого прізвища та цифра 3
4	Перша літера вашого імені та цифра 4
5	Перша літера вашого прізвища та цифра 5
6	Перша літера вашого імені та цифра 6
7	Перша літера вашого прізвища та цифра 7
8	Перша літера вашого імені та цифра 8
9	Перша літера вашого прізвища та цифра 9
10	Перша літера вашого імені та цифра 0

Примітка. Літера і цифра повинні бути зображені як такі, що при вирізанні з листкового металу не розпадаються на частини, наприклад, літера Ю, може бути зображена так:

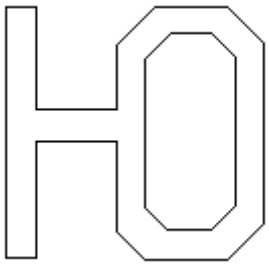

Рис. 2.1 - Приклад зображення літери

Теоретичні відомості

При побудові графічних зображень будь-яка графічна бібліотека використовує операції рисування графічних примітивів типу ”точка”, ”пряма”, ”коло”, ”еліпс” і т.п. Проте, в основі будь-якої графічної бібліотеки лежить операція відображення точки. А інші, більш складні фігури зображуються за допомогою виведення багатьох точок.
Розглянемо, наприклад, як використовуючи лише операцію виведення точки, можна намалювати відрізок прямої лінії (рис.3).

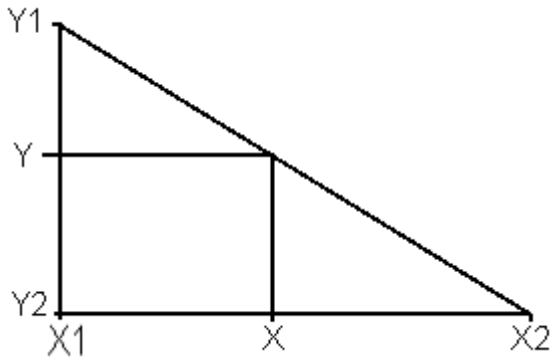

Рис. 2.2 — зображення відрізка прямої ліній

Кожен відрізок визначається координатами початку і кінця: (x1,y1) та (x2,y2). Як відомо, через кожні дві точки проходить єдина пряма, її рівняння можна представити у вигляді $\frac{y-y1}{y2-y1}=\frac{x-x1}{x2-x1}$, або $(x2-x1)(y-y1)=(x-x1)(y2-y1)$, звідки випливає $y=y1+\frac{x-x1}{x2-x1}(y2-y1)$, тобто $y=f(x)$ або $x=x1+(y-y1)\frac{x_2-x_1}{y_2-y_1}$, тобто $x=f1(y)$.

В залежності від кута нахилу прямої виконується цикл вздовж вісі x або y, якщо $|x_2 - x_1| > |y_2 - y_1|$, то виконується цикл по x, якщо ні – то по y.

На основі цих формул можливо побудувати алгоритм виведення прямої лінії, у якому, як легко бачити, буде міститись операція ділення, проте, оскільки на екран треба виводити тільки цілі координати, то доведеться ще використовувати операцію округлення. Крім того, як відомо, операції з дійсними числами на більшості комп'ютерних платформ виконуються повільніше, ніж із цілими числами. Тому, бажано розробити такий алгоритм побудови прямих ліній, який би не використовував операцій ділення та округлення. Оскільки початкові та кінцеві координати відрізка прямої є цілими числами, то всі дії в ньому будуть лише над цілими числами.

Брезенхем запропонував підхід, що дозволяє розробляти так звані інкрементні алгоритми растеризації. Основною метою для розробки таких алгоритмів була побудова циклів обчислення координат на основі тільки операцій з цілими числами – додавання/віднімання, та без застосування множення та ділення. На теперішній час відомі інкрементні алгоритми не тільки для відрізків прямих, але і для кривих ліній.

Інкрементні алгоритми виконуються як послідовне обчислення координат сусідніх пікселів шляхом додавання приростів координат. Прирости обчислюються на основі аналізу функції похибки. В циклі виконуються лише цілочислові операції порівняння і додавання/віднімання. Таким чином досягається збільшення швидкості обчислення кожного пікселя у порівнянні з прямим шляхом на основі формули рівняння прямої.

Наведемо один з варіантів алгоритму Брезенхема для прямої (всі змінні, що використовуються в цьому алгоритмі, лише цілі):

```
xErr = 0; yErr = 0;
dx = x2 – x1; dy = y2 – y1;

Якщо dx > 0 то incX = 1;
     dx = 0 то incX = 0;
     dx < 0 то incX = -1;
Якщо dy > 0 то incY = 1;
     dy = 0 то incY = 0;
     dy < 0 то incY = -1;
dx = |dx|; dy = |dy|;
Якщо dx > dy
  то d = dx
  інакше d = dy;

x = x1; y = y1;
малюватиПіксел(x,y);
Виконати цикл d разів {
     xErr += dx;
```

```
        yErr += dy;
        Якщо xErr > d, то {
              xErr -= d; x += incX;
        }
        Якщо yErr > d, то {
              yErr -= d; y += incY;
        }
        малюватиПіксел(x,y);
}
```

Для реалізації цього алгоритму за допомогою будь якої мови програмування, треба представити інструкції псевдокоду у вигляді інструкцій мови, крім того, треба використати одну з існуючих графічних бібліотек обраної мови програмування. В обраній бібліотеці треба скористатися функцією, що малює точку на екрані та замінити нею інструкцію “малюватиПіксел”.

Крім алгоритмів побудови прямих ліній, Брезенхем запропонував також алгоритми побудови деяких кривих: кола, еліпса та ін.

Наведемо інкрементний алгоритм виведення кола з центром у точці **xc**, **yc** та радіусом **radius**, при цьому будемо вважати, що в нас є визначена така функція sim, що малює точки симетрично відносно точки xc, yc та прямих, що проходять через цю точку паралельно осям координат.

```
sim(int x, int y, Color col) {
   малюватиПіксел(x+xc, y+yc, col);
   малюватиПіксел(x+xc, -y+yc, col);
   малюватиПіксел(-x+xc, -y+yc, col);
   малюватиПіксел(-x+xc, y+yc, col);
   малюватиПіксел(y+xc, x+yc, col);
   малюватиПіксел(y+xc, -x+yc, col);
   малюватиПіксел(-y+xc, -x+yc, col);
   малюватиПіксел(-y+xc, x+yc, col);
}
```

```
d = 3 - 2 * y;
x = 0;
y = r;
поки (x <= y) {
     sim(x,y);
     якщо (d<0)
     то {
          d = d + 4 * x + 6;
     }
     інакше {
          d = d + 4 * (x - y) + 10;
          dec(y);
     }
     x++;
}
```

В цьому алгоритмі використано симетрію кола – в основному циклі обчислюються координати точок кола тільки для одного октанта (між променями OA і OB) і одразу малюються вісім симетрично розташованих пікселів.

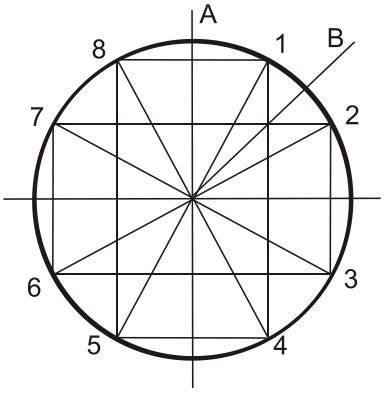

Рис. 2.3 — Симетрія кола