**BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**TRƯỜNG ĐẠI HỌC NHA TRANG**

**KHOA CÔNG NGHỆ THÔNG TIN**

---🙢🕮🙠---



**BÁO CÁO THỰC TẬP CƠ SỞ CÔNG NGHỆ THÔNG TIN**

**ĐỀ TÀI: SỬ DỤNG CÔNG CỤ ĐỒ HOẠ GRAPHICS.H ĐỂ MÔ PHỎNG THUẬT TOÁN SINH VÀ TÔ MÀU ĐA GIÁC**

**Giáo viên hướng dẫn: Ths. Doàn Vũ Thịnh**

**Sinh viên thực hiện: Trịnh Đăng Khoa**

**Mã số sinh viên: 63134337**

**Lớp: 63.CNTT-4**

**Nha Trang, 12/2023**

**BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**TRƯỜNG ĐẠI HỌC NHA TRANG**

**KHOA CÔNG NGHỆ THÔNG TIN**

---🙢🕮🙠---



**BÁO CÁO THỰC TẬP CƠ SỞ CÔNG NGHỆ THÔNG TIN**

**ĐỀ TÀI: SỬ DỤNG CÔNG CỤ ĐỒ HOẠ GRAPHICS.H ĐỂ MÔ PHỎNG THUẬT TOÁN SINH VÀ TÔ MÀU ĐA GIÁC**

**Giáo viên hướng dẫn: Ths. Doàn Vũ Thịnh**

**Sinh viên thực hiện: Trịnh Đăng Khoa**

**Mã số sinh viên: 63134337**

**Lớp: 63.CNTT-4**

**Nha Trang, 12/2023**

**TRƯỜNG ĐẠI HỌC NHA TRANG**

**Khoa: Công nghệ Thông tin**

**PHIẾU THEO DÕI TIẾN ĐỘ VÀ ĐÁNH GIÁ BÁO CÁO THỰC TẬP CƠ SỞ**

**Tên đề tài:** SỬ DỤNG CÔNG CỤ ĐỒ HOẠ GRAPHICS.H ĐỂ MÔ PHỎNG THUẬT TOÁN SINH VÀ TÔ MÀU ĐA GIÁC

**Giảng viên hướng dẫn:** ThS. Đoàn Vũ Thịnh

**Sinh viên được hướng dẫn:** Trịnh Đăng Khoa

**MSSV:** 63134337

**Khóa:** 63 **Ngành:** Công nghệ Thông tin

|  |  |  |  |
| --- | --- | --- | --- |
| **Lần** | **Ngày** | **Nội dung** | **Nhận xét của GVHD** |
| 1 | 27/11/2023 | Nhận đề tài hướng dẫn và định hướng giải quyết vấn đề. Sinh viên trình bày kế hoạch thực hiện. |  |
| 2 | 04/12/2023 | Sinh viên trình bày việc mô phỏng thuật toán chính dựa trên kiến thức đã được học ở môn kỹ thuật đồ họa và các kiến thức thu nhận được từ Internet để minh họa bài toán đa dạng nhất có thể. |  |
| 3 | 11/12/2023 | Sinh viên thiết kế giao diện chương trình như các nút chạy thuật toán, nút nhập dữ liệu từ bàn phím, từ file, từ chuột, khung hiển thị và các nút thay đổi màu sắc |  |
| 4 | 18/12/2023 | Sinh viên xây dựng các hàm dựa trên ý tưởng thuật toán |  |
| 5 | 25/12/2023 | Sinh viên trình bày bản thảo lần 1 |  |
| 6 | 1/1/2024 | Sinh viên nộp bản thảo lần cuối sau khi đã chỉnh sửa các yêu cầu như đã đề ra. |  |

# LỜI CẢM ƠN

Để có thể hoàn thành đợt thực tập lần này, em xin chân thành cảm ơn đến quý thầy cô khoa Công nghệ Thông tin đã tạo điều kiện hỗ trợ và giúp đỡ em trong quá trính học tập và nghiên cứu đề tài này.

Qua đây, em xin chân thành cảm ơn thầy Đoàn Vũ Thịnh, người đã trực tiếp quan tâm và hướng dẫn chúng em hoàn thành tốt đợt thực tập trong thời gian qua.

Do kiến thức còn hạn chế và thời gian thực hiện còn ngắn nên bài báo cáo của em còn nhiều thiếu sót, kính mong sự góp ý của quý thầy cô.

Em xin chân thành cảm ơn!

**MỤC LỤC**

[LỜI CẢM ƠN ii](#_Toc155558906)

[MỤC LỤC iii](#_Toc155558907)

[MỤC LỤC HÌNH ẢNH v](#_Toc155558908)

[Chương 1. TỔNG QUAN VỀ ĐỀ TÀI 1](#_Toc155558909)

[1.1 THƯ VIỆN GRAPHICS.H 1](#_Toc155558910)

[1.2 CÁC THUẬT TOÁN TÔ MÀU 1](#_Toc155558911)

[1.3 THUẬT TOÁN SCANLINE 1](#_Toc155558912)

[1.3.1 Giới thiệu 1](#_Toc155558913)

[1.3.2 Ý tưởng thuật toán 1](#_Toc155558914)

[1.3.3 Đánh giá thuật toán 2](#_Toc155558915)

[1.4 THUẬT TOÁN FLOOD FILL 3](#_Toc155558916)

[1.4.1 Giới thiệu 3](#_Toc155558917)

[1.4.2 Ý tưởng thuật toán 3](#_Toc155558918)

[1.4.3 Đánh giá thuật toán 4](#_Toc155558919)

[1.5 THUẬT TOÁN BOUNDARY FILL 4](#_Toc155558920)

[1.5.1 Giới thiệu 4](#_Toc155558921)

[1.5.2 Ý tưởng thuật toán 5](#_Toc155558922)

[1.5.3 Đánh giá thuật toán 5](#_Toc155558923)

[Chương 2. PHƯƠNG PHÁP NGHIÊN CỨU 7](#_Toc155558924)

[2.1 TỔNG QUAN VỀ GIAO DIỆN 7](#_Toc155558925)

[2.1.1 Sơ lược về giao diện 7](#_Toc155558926)

[2.1.2 Xây dựng giao diện 7](#_Toc155558927)

[2.2 XÂY DỰNG CÁC HÀM CHÍNH 15](#_Toc155558928)

[2.2.1 Hàm nhập dữ liệu bằng bàn phím 15](#_Toc155558929)

[2.2.2 Hàm nhập dữ liệu từ file 16](#_Toc155558930)

[2.2.3 Hàm vẽ đa giác 16](#_Toc155558931)

[2.2.4 Các hàm của thuật toán Scanline 17](#_Toc155558932)

[2.2.5. Các hàm của thuật toán Flood Fill 19](#_Toc155558933)

[2.2.6 Các hàm của thuật toán Boundary Fill 20](#_Toc155558934)

[2.2.7 Hàm kiểm tra 1 điểm có nằm trong 1 nút không 22](#_Toc155558935)

[2.3. CÁC THAO TÁC TRONG CHƯƠNG TRÌNH 22](#_Toc155558936)

[2.3.1 Các sự kiện khi dùng chuột 22](#_Toc155558937)

[2.3.2 Tạo hiệu ứng khi di chuyển chuột 23](#_Toc155558938)

[2.3.3 Thao tác với các nút nhấn 25](#_Toc155558939)

[Chương 3. KẾT QUẢ THỰC HIỆN 29](#_Toc155558940)

[Chương 4. KẾT LUẬN VÀ ĐÁNH GIÁ BẢN THÂN 34](#_Toc155558941)

[4.1 KẾT QUẢ ĐẠT ĐƯỢC 34](#_Toc155558942)

[4.2 NHƯỢC ĐIỂM 34](#_Toc155558943)

[TÀI LIỆU THAM KHẢO 35](#_Toc155558944)

**MỤC LỤC HÌNH ẢNH**

[Hình 1.1. Ví dụ minh họa về thuật toán Scanline 2](#_Toc155465175)

[Hình 1.2. Ví dụ minh họa về thuật toán Flood Fill 3](#_Toc155465176)

[Hình 1.3. Ví dụ minh họa về thuật toán Boundary Fill 5](#_Toc155465177)

[Hình 2.1. Giao diện khi bắt đầu chương trình.................................................................7](#_Toc155682931)

[Hình 2.2. Các thư viện trong chương trình 7](#_Toc155682932)

[Hình 2.3. Các biến kiểu struct 8](#_Toc155682933)

[Hình 2.4. Các biến toàn cục 8](#_Toc155682934)

[Hình 2.5. Code xây dựng cửa sổ đồ họa BGI 8](#_Toc155682935)

[Hình 2.6. Kích thước cửa sổ đồ họa BGI 9](#_Toc155682936)

[Hình 2.7. Code xây dựng khung hiển thị 9](#_Toc155682937)

[Hình 2.8. Kích thước khung hiển thị 10](#_Toc155682938)

[Hình 2.9. Code hàm button\_color() 10](#_Toc155682939)

[Hình 2.10. Code vẽ các nút màu 11](#_Toc155682940)

[Hình 2.11. Các nút màu 12](#_Toc155682941)

[Hình 2.12. Code hàm button\_exe() 12](#_Toc155682942)

[Hình 2.13. Code vẽ các nút chức năng 13](#_Toc155682943)

[Hình 2.14. Các nút chức năng 14](#_Toc155682944)

[Hình 2.15. Code thông tin tác giả 14](#_Toc155682945)

[Hình 2.16. Thông tin tác giả 15](#_Toc155682946)

[Hình 2.17. Code hàm keyboard() 15](#_Toc155682947)

[Hình 2.18. Code hàm readfile() 16](#_Toc155682948)

[Hình 2.19. Code hàm drawpoly() 16](#_Toc155682949)

[Hình 2.20. Code hàm hesogoc() 17](#_Toc155682950)

[Hình 2.21. Code hàm minmax() 18](#_Toc155682951)

[Hình 2.22. Code hàm scanline() 18](#_Toc155682952)

[Hình 2.23. Code hàm fillright() 19](#_Toc155682953)

[Hình 2.24. Code hàm fillleft() 20](#_Toc155682954)

[Hình 2.25. Code hàm floodfill() 20](#_Toc155682955)

[Hình 2.26. Code hàm boundaryright() 21](#_Toc155682956)

[Hình 2.27. Code hàm boundaryleft() 21](#_Toc155682957)

[Hình 2.28. Code hàm boundaryfill() 22](#_Toc155682958)

[Hình 2.29. Code hàm check\_button() 22](#_Toc155682959)

[Hình 2.30. Code xử lý sự kiện khi di chuyển chuột 23](#_Toc155682960)

[Hình 2.31. Code hiển thị tọa độ khi di chuyển chuột 23](#_Toc155682961)

[Hình 2.32. Hiển thị tọa độ khi di chuyển chuột 24](#_Toc155682962)

[Hình 2.33. Code tạo hiệu ứng hover nút 24](#_Toc155682963)

[Hình 2.34. Hiệu ứng hover các nút 25](#_Toc155682964)

[Hình 2.35. Code xử lý sự kiện nhấn chuột trái 25](#_Toc155682965)

[Hình 2.36. Code nhấn nút Load file 25](#_Toc155682966)

[Hình 2.37. Code nhấn nút Keyboard 26](#_Toc155682967)

[Hình 2.38. Code nhấn nút Mouse 26](#_Toc155682968)

[Hình 2.39. Code nhấn các nút màu 26](#_Toc155682969)

[Hình 2.40. Code nhấn nút Scanline 27](#_Toc155682970)

[Hình 2.41. Code nhấn nút Floodfill 27](#_Toc155682971)

[Hình 2.42. Code nhấn nút Boundaryfill 27](#_Toc155682972)

[Hình 3.1. Vẽ đa giác khi nhấn nút Load file.................................................................29](#_Toc155558852)

[Hình 3.2. Cửa sổ console để nhập dữ liệu 29](#_Toc155558853)

[Hình 3.3. Vẽ đa giác khi nhập dữ liệu 30](#_Toc155558854)

[Hình 3.4. Vẽ các đỉnh của đa giác bằng chuột 30](#_Toc155558855)

[Hình 3.5. Vẽ đa giác bằng chuột 31](#_Toc155558856)

[Hình 3.6. Chạy thuật toán Scanline 31](#_Toc155558857)

[Hình 3.7. Chạy thuật toán Flood Fill khi điểm xuất phát bên trong đa giác 32](#_Toc155558858)

[Hình 3.8. Chạy thuật toán Flood Fill khi điểm xuất phát bên ngoài đa giác 32](#_Toc155558859)

[Hình 3.9. Chạy thuật toán Boundary Fill khi điểm xuất phát bên trong đa giác 33](#_Toc155558860)

[Hình 3.10. Chạy thuật toán Boundary Fill khi điểm xuất phát bên ngoài đa giác 33](#_Toc155558861)

# Chương 1. TỔNG QUAN VỀ ĐỀ TÀI

## 1.1 THƯ VIỆN GRAPHICS.H

Graphics.h hay tên chính xác và đầy đủ của nó là Borland Graphics Interface – còn được biết đến với tên gọi BGI – là một thư viện đồ họa rất phổ biến trên DOS và các máy tính chạy hệ điều hành Windows thời kì đầu như Windows 95, Windows 98.

Thư viện này cung cấp cho người dùng 2 file: graphics.h và graphics.lib để có thể sử dụng được với ngôn ngữ C/C++ cũng như module graph nếu người dùng sử dụng ngôn ngữ Pascal. Bộ thư viện này đi kèm với IDE Borland C++ 3.1 (1992).

Một trong những điểm mạnh của thư viện này là việc khởi tạo cũng như sử dụng rất đơn giản, vì vậy mặc dù ra đời rất lâu nhưng hiện tại vẫn có rất nhiều trường đại học sử dụng cho mục đích giảng dạy.

## 1.2 CÁC THUẬT TOÁN TÔ MÀU

* Một vùng tô thường được xác định bởi một đường khép kín nào đó gọi là đường biên. Dạng đường biên đơn giản thường gặp là đa giác.
* Có hai dạng vùng tô thường gặp: tô bằng một màu thuần nhất (solid fill) và tô theo một mẫu tô (fill - pattern) nào đó.
* Việc tô màu thường được chia làm hai công đoạn :
  + Xác định vị trí các điểm cần tô màu.
  + Quyết định tô các điểm trên bằng màu nào. Công đoạn này thực sự phức tạp khi ta cần tô theo một mẫu tô nào đó chứ không phải tô thuần một màu.
* Có 3 thuật toán chính: thuật toán scanline, thuật toán floodfill và thuật toán boundaryfill.

## 1.3 THUẬT TOÁN SCANLINE

### 1.3.1 Giới thiệu

Thuật toán tô màu Scanline (hay còn gọi là thuật toán tô màu theo dòng quét) được áp dụng để tô màu các đa giác lồi, lõm hay đa giác tự cắt.

Với mỗi dòng quét, ta sẽ xác định phần giao của đa giác và dòng quét, rồi tô màu các pixel thuộc đoạn giao đó. Để xác định các đoạn giao, ta tiến hành việc tìm giao điểm của dòng quét với các cạnh của đa giác, sau đó các giao điểm này sẽ được sắp theo thứ tự tăng dần của hoành độ giao điểm. Các đoạn giao chính là các đoạn thẳng được giới hạn bởi từng cặp giao điểm một.

### 1.3.2 Ý tưởng thuật toán

* Tìm ymin, ymax lần lượt là giá trị nhỏ nhất, lớn nhất của tập các tung độ của các đỉnh của đa giác đã cho.
* Ứng với mỗi dòng quét y=k, k thay đổi từ ymin đến ymax, lặp:
  + Tìm tất cả các hoành độ giao điểm của dòng quét y=k với các cạnh của đa giác.
  + Sắp xếp các hoành độ giao điểm theo thứ tự tăng dần: x0, x1, ….
  + Tô màu các đoạn thẳng y=k trên đường thẳng lần lượt được giới hạn bởi các cặp (x0, x1), (x2, x3), ….(x(2k), x(2k+1)).

**A group of white dots

Description automatically generated**

Hình 1.1. Ví dụ minh họa về thuật toán Scanline

### 1.3.3 Đánh giá thuật toán

#### 1.3.3.1 Độ phức tạp

* Độ phức tạp của thuật toán Scanline phụ thuộc vào số lượng đoạn đường hoặc vùng mà bạn đang xử lý.
* Trong trường hợp đơn giản nhất, khi bạn chỉ cần quét qua mỗi pixel một lần, độ phức tạp của thuật toán là O(n), trong đó n là tổng số pixel trong hình ảnh.
* Tuy nhiên, nếu bạn xem xét cả quét qua mỗi pixel của mỗi đoạn đường hoặc vùng, thì độ phức tạp sẽ tăng lên tùy thuộc vào số lượng đoạn đường hoặc vùng cần xử lý. Điều này có thể là O(m \* n), trong đó m là số lượng đoạn đường hoặc vùng cần xử lý và n là tổng số pixel.

#### 1.3.3.2 Ưu điểm, nhược điểm

* Ưu điểm
  + Thuật toán Scanline có cấu trúc đơn giản, dễ hiểu và thực hiện, làm cho việc triển khai nó trở nên thuận lợi.
  + Hiệu quả khi xử lý các đối tượng có cạnh thẳng và hình dạng đơn giản.
  + Tính toán điểm cắt giữa các cạnh và dòng quét được thực hiện một cách hiệu quả.
* Nhược điểm
  + Không hiệu quả khi xử lý các đối tượng có hình dạng phức tạp, có nhiều đỉnh và cạnh không thẳng.
  + Hiệu suất giảm khi xử lý các đối tượng lớn, có nhiều cạnh và đỉnh, vì cần phải thực hiện nhiều tính toán.
  + Không xử lý tốt khi có đối tượng chồng chéo lên nhau, có thể dẫn đến việc không đúng khi xác định thứ tự vẽ các đối tượng.
  + Yêu cầu bước sắp xếp các cạnh của đối tượng theo chiều tăng dần hoặc giảm dần của giá trị y, tăng độ phức tạp của thuật toán.
  + Có thể gây ra tình trạng overdraw, khi một số pixel được tô màu nhiều lần, dẫn đến tốn tài nguyên tính toán không cần thiết.
  + Có thể mất mát chi tiết trong trường hợp các điểm cắt không đủ để đảm bảo sự chính xác của hình ảnh.

## 1.4 THUẬT TOÁN FLOOD FILL

### 1.4.1 Giới thiệu

Thuật toán Flood Fill (hay còn gọi là thuật toán tô màu loang) là một thuật toán sử dụng trong đồ họa máy tính và xử lý ảnh để tô màu các khu vực liên tục. Mục tiêu của thuật toán là tô màu một khu vực nào đó, bắt đầu từ một điểm cụ thể (gọi là điểm xuất phát) bằng một màu mới, sao cho tất cả các điểm trong khu vực đó có màu giống nhau hoặc có màu gần giống với màu của điểm xuất phát.

### 1.4.2 Ý tưởng thuật toán

* Chọn một điểm bắt đầu (x, y) trong ma trận và lưu màu ban đầu của nó.
* Đặt màu mới cho điểm xuất phát.
* Kiểm tra và tô màu các điểm xung quanh:
  + Kiểm tra màu của các điểm xung quanh điểm xuất phát.
  + Nếu màu của điểm xung quanh giống với màu ban đầu, thì đặt màu mới cho điểm đó và thêm vào danh sách xử lý tiếp theo.
* Lặp lại cho tất cả các điểm xung quanh:
  + Lặp lại bước 3 cho tất cả các điểm xung quanh điểm xuất phát.
  + Các điểm mới được thêm vào danh sách xử lý tiếp theo.
* Lặp lại quá trình kiểm tra và tô màu cho đến khi không còn điểm nào cần xử lý.

A picture containing screen, food, drawing

Description automatically generatedA picture containing screen, drawing

Description automatically generated

Hình 1.2. Ví dụ minh họa về thuật toán Flood Fill

### 1.4.3 Đánh giá thuật toán

#### 1.4.3.1 Độ phức tạp

* Độ phức tạp của thuật toán này thường phụ thuộc vào kích thước của vùng cần tô màu và số lượng pixel trong vùng đó.
* Nếu giả sử có một vùng kín đóng với kích thước là O(n), trong đó n là số lượng pixel trong vùng, và nếu mỗi pixel được xem xét một lần duy nhất, thì độ phức tạp của thuật toán Flood Fill sẽ là O(n). Trong trường hợp tốt nhất, khi chỉ cần tô màu một số ít pixel, độ phức tạp có thể là O(1).

#### 1.4.3.2 Ưu điểm, nhược điểm

* Ưu điểm
  + Flood Fill có cấu trúc đơn giản, dễ hiểu và triển khai. Việc thực hiện thuật toán này không đòi hỏi nhiều phức tạp so với một số thuật toán khác.
  + Hiệu quả khi áp dụng cho các vùng màu đồng nhất, nơi mà một số điểm bắt đầu có thể tô màu cho toàn bộ vùng một cách nhanh chóng.
  + Có thể được triển khai bằng cách sử dụng đệ quy hoặc ngăn xếp (stack), tùy thuộc vào yêu cầu cụ thể của ứng dụng.
  + Người dùng có thể chọn bất kỳ điểm bắt đầu nào trong vùng cần tô màu, làm cho thuật toán linh hoạt và dễ sử dụng.
* Nhược điểm
  + Hiệu suất giảm khi áp dụng cho các vùng màu phức tạp hoặc có nhiều đường biên.
  + Không xử lý tốt trong trường hợp có đối tượng chồng chéo lên nhau, có thể dẫn đến việc không đúng màu ở các khu vực chồng chéo.
  + Nếu không được quản lý đúng, việc sử dụng ngăn xếp có thể dẫn đến việc cần một đệm ngăn xếp lớn khi tô màu một vùng lớn.
  + Trong một số trường hợp, thuật toán có thể tạo ra kết quả màu sắc không đồng đều, đặc biệt là ở các khu vực có biên giới có thể chứa nhiều màu sắc.
  + Yêu cầu dữ liệu hình ảnh phải được biểu diễn đúng để đảm bảo hiệu suất và độ chính xác của thuật toán.

## 1.5 THUẬT TOÁN BOUNDARY FILL

### 1.5.1 Giới thiệu

Thuật toán Boundary Fill (hay còn gọi là thuật toán tô màu theo đường biên, tô lân cận). Đường biên của vùng tô màu ở thuật toán tô loang được xác định bởi tập các đỉnh của 1 đa giác, đường biên trong thuật toán được mô tả bằng một giá trị duy nhất, đó là màu của tất cả các điểm thuộc về đường biên (nói ngắn gọn là chúng ta sẽ tô đường biên một màu riêng).

### 1.5.2 Ý tưởng thuật toán

* Bắt đầu từ một điểm bất kỳ (x, y) trong vùng cần tô màu.
* Nếu màu tại điểm này không phải là màu biên và không phải là màu mới mà chúng ta muốn tô, thì thực hiện các bước sau:
  + Tô màu tại điểm (x, y) bằng màu mới.
  + Gọi đệ quy thuật toán Boundary Fill cho các điểm lân cận (trên, dưới, trái, phải).
* Dừng đệ quy khi màu tại điểm đang xét bằng màu biên hoặc bằng màu mới.
* Nếu điều kiện dừng được đáp ứng (màu tại điểm đang xét là màu biên hoặc màu mới), thì không gọi đệ quy cho các điểm lân cận để tránh "tràn màu".

A black and blue dot pattern with pink and blue dots

Description automatically generated with medium confidence

Hình 1.3. Ví dụ minh họa về thuật toán Boundary Fill

### 1.5.3 Đánh giá thuật toán

#### 1.5.3.1 Độ phức tạp

* Độ phức tạp của thuật toán này thường phụ thuộc vào kích thước của vùng cần tô màu và cách mà thuật toán được triển khai.
* Giả sử vùng cần tô màu có kích thước là O(n), trong đó n là số lượng pixel trong vùng. Nếu mỗi pixel được xem xét một lần và nếu giả sử mọi điều kiện kiểm tra (check) và cập nhật màu được thực hiện trong thời gian O(1) (hằng số), thì độ phức tạp của thuật toán Boundary Fill có thể là O(n).

#### 1.5.3.2 Ưu điểm, nhược điểm

* Ưu điểm
  + Thuật toán Boundary Fill có cấu trúc đơn giản, dễ triển khai và hiểu. Người lập trình có thể thực hiện nhanh chóng.
  + Hiệu quả khi áp dụng cho các vùng có biên đơn giản, không quá phức tạp.
  + Có thể kiểm soát biên của vùng cần tô màu dễ dàng, giúp tránh được tràn màu.
  + Sử dụng tính chất đệ quy giúp thuật toán linh hoạt và dễ mở rộng.
* Nhược điểm
  + Không hiệu quả khi áp dụng cho các vùng có biên phức tạp, có thể dẫn đến việc tràn màu hoặc không tô màu đúng các vùng cần thiết.
  + Yêu cầu màu biên để xác định biên của vùng cần tô màu, điều này làm cho thuật toán không linh hoạt khi màu biên không được xác định một cách chính xác.
  + Đệ quy không kiểm soát có thể dẫn đến tràn đệ quy (stack overflow) nếu không được quản lý đúng.
  + Không xử lý tốt trong trường hợp có đối tượng chồng chéo lên nhau, có thể tô màu các vùng không mong muốn.
  + Với các vùng lớn, thuật toán có thể đối mặt với vấn đề hiệu suất do số lượng lớn lệch đệ quy.
  + Trong một số trường hợp, thuật toán có thể tạo ra kết quả màu sắc không đồng đều, đặc biệt là ở các khu vực có biên giới có thể chứa nhiều màu sắc.

# Chương 2. PHƯƠNG PHÁP NGHIÊN CỨU

## 2.1 TỔNG QUAN VỀ GIAO DIỆN

### 2.1.1 Sơ lược về giao diện

Giao diện chính bao gồm: khung hiển thị, các nút màu, các nút vẽ đa giác (từ file, từ bàn phím hay bằng chuột), các nút chạy thuật toán tô màu (thuật toán Scanline, thuật toán Flood Fill, thuật toán Boundary Fill), nút làm mới màn hình hiển thị và thông tin của tác giả làm chương trình.

A screenshot of a computer

Description automatically generated

Hình 2.1. Giao diện khi bắt đầu chương trình

### 2.1.2 Xây dựng giao diện

#### 2.1.2.1 Khai báo các biến và thư viện của chương trình

* Các thư viện cần dùng:

A screenshot of a computer screen

Description automatically generated

Hình 2.2. Các thư viện trong chương trình

* Các biến kiểu struct:

A screen shot of a computer

Description automatically generated

Hình 2.3. Các biến kiểu struct

* A screen shot of a computer code

  Description automatically generatedCác biến toàn cục:

Hình 2.4. Các biến toàn cục

#### 2.1.2.2 Xây dựng cửa sổ đồ họa BGI

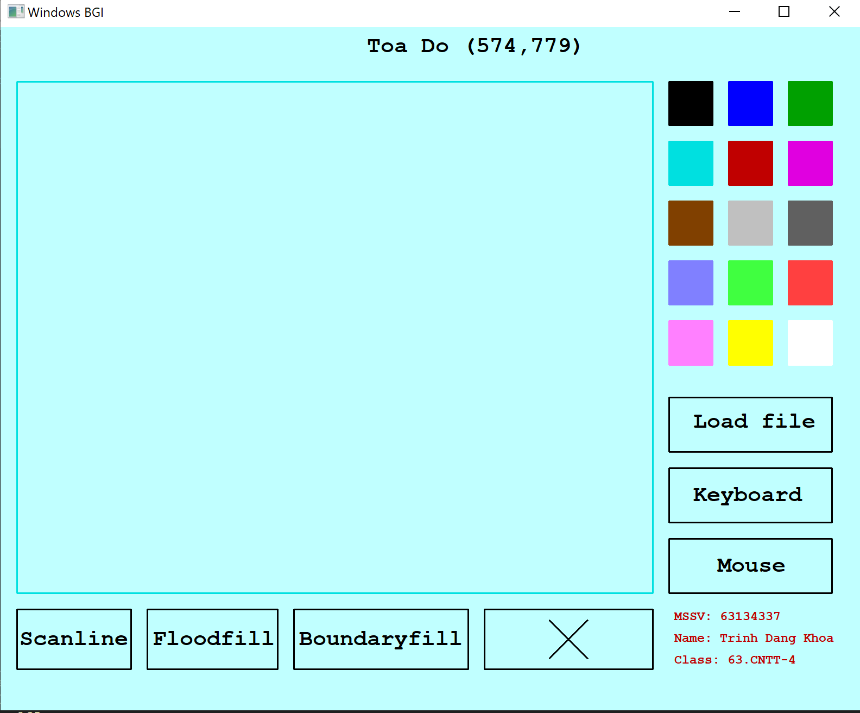
A screen shot of a computer code

Description automatically generated

Hình 2.5. Code xây dựng cửa sổ đồ họa BGI

Sử dụng các hàm trong thư viện graphics.h để xây dựng cửa sổ đồ họa BGI và được viết trong hàm *main()*:

* + *initwindow(1600, 1300)*: tạo cửa sổ đồ họa BGI với chiều dài bằng 1600 và chiều rộng bằng 1300.
  + *setbkcolor(11)*: tạo màu cho cửa sổ đồ họa BGI.



**1600**

**1300**

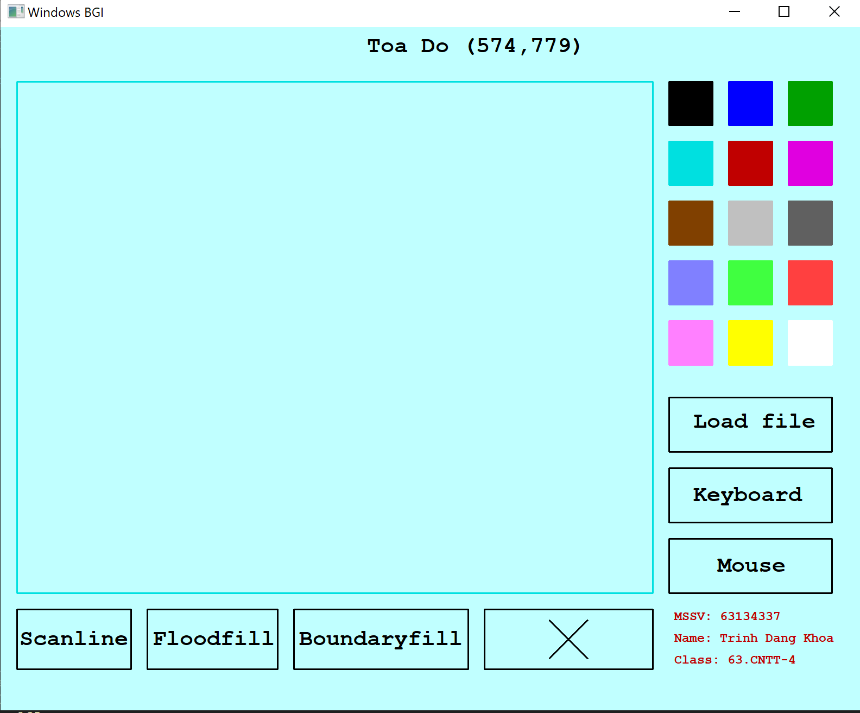
Hình 2.6. Kích thước cửa sổ đồ họa BGI

#### A screen shot of a computer code Description automatically generated2.1.2.3 Tạo khung hiển thị

Hình 2.7. Code xây dựng khung hiển thị

Sử dụng các hàm trong thư viện graphics.h để xây dựng khung hiển thị và được viết trong hàm *GUI\_init()*:

* + *setcolor(3)*: tạo màu cho khung hiển thị.
  + *setlinestyle(0, 0 ,3)*: tham số thứ nhất dùng để dịnh dạng kiểu khung (0 là kiểu solid\_line), tham số thứ 2 có thể bỏ qua vì tham số thứ nhất là 0 (chỉ dùng khi tham số thứ nhất bằng 4), tham số thứ 3 dùng để tạo độ dày của khung (3 là 3px).
  + *rectangle(30, 100, 1200, 1040)*: 2 tham số đầu dùng để chỉ tọa độ đỉnh trái trên (30, 100) của khung và 2 tham số cuối dùng để chỉ tọa độ dưới phải (1200, 1040) của khung. Do đó, chiều dài của khung 1170 (1200 - 30) và chiều rộng là 940 (1040 - 100).



**940**

**1170**

Hình 2.8. Kích thước khung hiển thị

#### 2.1.2.4 Tạo các nút

Sử dụng các hàm trong graphics.h để xây dụng các nút:

* Các nút màu
  + Vì các nút màu có hình dạng giống nhau nên tạo một hàm riêng để vẽ các nút màu với các tham số là btn kiểu button và color kiểu int (màu và vị trí các nút là khác nhau).

A screen shot of a computer code

Description automatically generated

Hình 2.9. Code hàm button\_color()

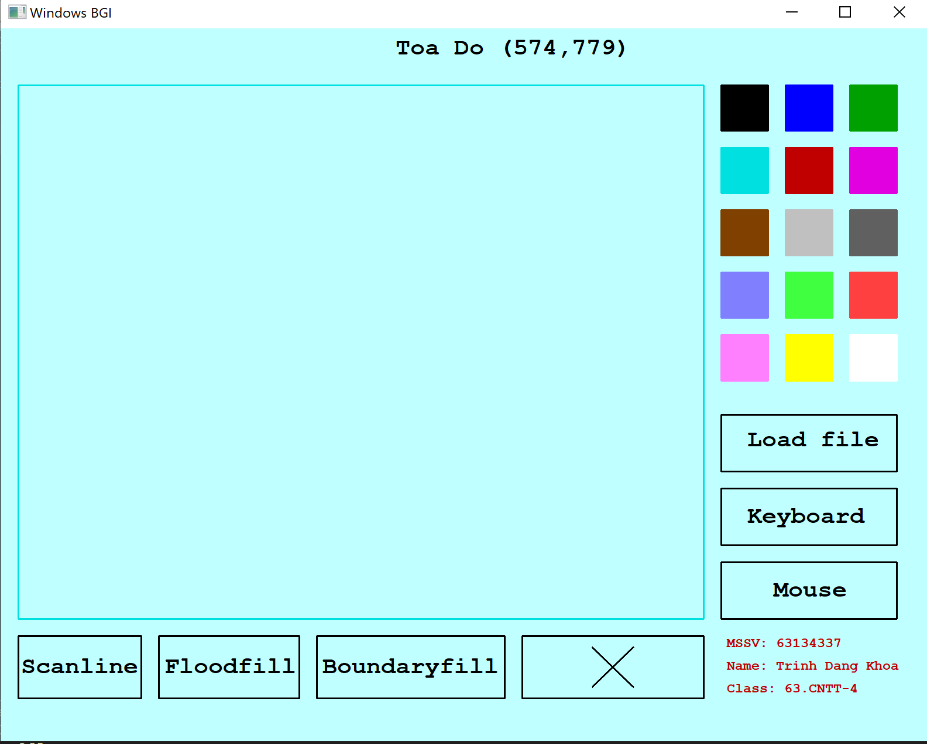
* *setcolor(color)*: tạo màu cho viền nút màu.
* *setfillstyle(1, color)*: tham số đầu tiên để xác định kiểu tô màu (1 là kiểu solid\_fill), tham số thứ 2 để xác định màu cần tô.
* *rectangle(btn.x1, btn.y1, btn.x2, btn.y2)*: tạo nút màu với tọa độ đỉnh trái trên (btn.x1, btn.y1) và tọa độ đỉnh dưới phải (btn.x2, btn.y2).
* *floodfill(btn.x1 + 2, btn.y1 + 2, color)*: tô màu cho nút màu với 2 tham số đầu là tọa độ điểm xuất phát để tô (điểm này phải nằm trong vùng cần tô), tham số cuối dùng để xác định màu đường biên của vùng cần tô màu.
* Vẽ các nút màu trong hàm *GUI\_init()*:

*A computer screen with numbers and symbols

Description automatically generated*

Hình 2.10. Code vẽ các nút màu

* Khai báo 1 mảng các nút màu kiểu button và vị trí ban đầu của nút màu đầu tiên.
* Dùng vòng lặp để vẽ các nút màu với i tăng từ 0 tới 15 (biểu thị cho 16 màu khác nhau) nhưng bỏ qua i bằng 11 (11 trùng với màu cửa sồ BGI). Với mỗi lần lặp tăng vị trí của các nút màu để tạo ra các nút màu ở vị trí khác nhau.



Hình 2.11. Các nút màu

* Các nút chức năng
  + Vì các nút chức năng có hình dạng giống nhau nên tạo một hàm riêng để vẽ các nút chức năng với các tham số là btn kiểu button, color\_btn kiểu int (màu và vị trí các nút là khác nhau), txt kiểu text và color\_txt kiểu int (tên các nút chức năng là khác nhau).

A computer screen shot of a code

Description automatically generated

Hình 2.12. Code hàm button\_exe()

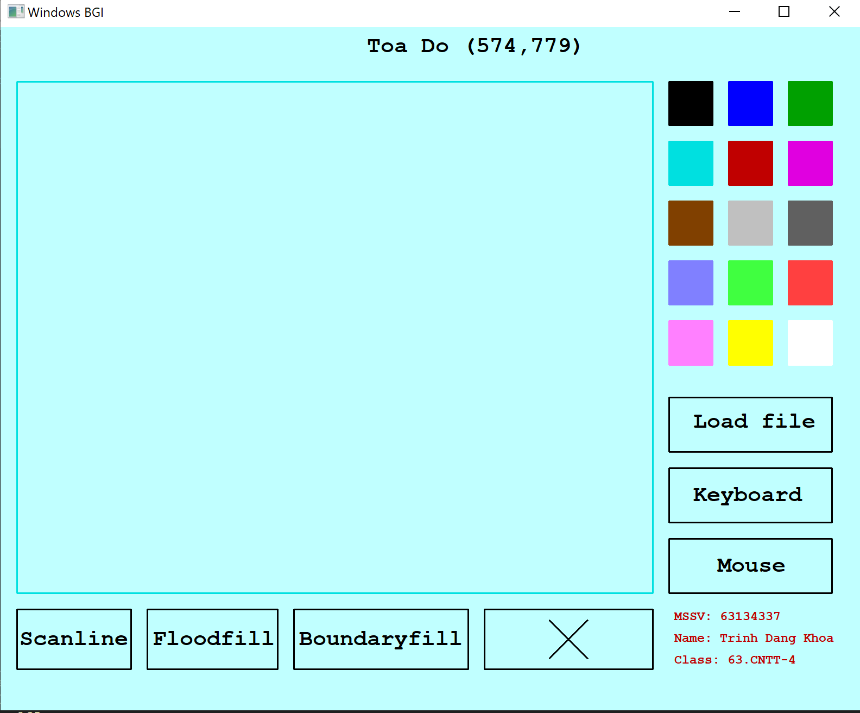
* *setcolor(color\_btn)*: tạo màu cho viền nút chức năng.
* *setlinestyle(0, 0 ,3)*: tham số thứ nhất dùng để dịnh dạng kiểu khung (0 là kiểu solid\_line), tham số thứ 2 có thể bỏ qua vì tham số thứ nhất là 0 (chỉ dùng khi tham số thứ nhất bằng 4), tham số thứ 3 dùng để tạo độ dày của nét vẽ (3 là 3px).
* *rectangle(btn.x1, btn.y1, btn.x2, btn.y2)*: tạo nút chức năng với tọa độ đỉnh trái trên (btn.x1, btn.y1) và tọa độ đỉnh dưới phải (btn.x2, btn.y2).
* *settextstyle(10, 0, 5)*: tham số thứ nhất dùng để xác định kiểu chữ (10 là kiểu bold\_font), tham số thứ 2 dùng để xác định chiều chữ (0 là từ trái sang phải) và tham số cuối dùng để xác định cỡ chữ.
* *setcolor(color\_btn)*: tạo màu cho chữ.
* *outtextxy(txt.x, txt.y, txt.txt)*: 2 tham số đầu để xác định tọa độ chữ cần hiển thị (txt.x, txt.y), tham số thứ 3 dùng để xác định nội dung của chữ.
* Vẽ các nút chức năng trong hàm *GUI\_init()*:

A screen shot of a computer program

Description automatically generated

Hình 2.13. Code vẽ các nút chức năng

* Khai báo các nút chức năng và chữ của các nút rồi truyền các đối số tương ứng vào hàm *button\_exe().*
* Riêng nút xóa sẽ không có chữ mà sẽ vẽ thêm đường gạch chéo nằm bên trong nút
  + *setcolor(0)*: tạo màu cho đường gạch chéo.
  + *line(1010, 1090, 1080, 1160)*: vẽ đoạn thẳng bắt đầu từ điểm có tọa độ (1010, 1090) tới điểm có tọa độ (1080, 1160).
  + *line(1010, 1160, 1080, 1090)*: vẽ đoạn thẳng bắt đầu từ điểm có tọa độ (1010, 1160) tới điểm có tọa độ (1080, 1090).



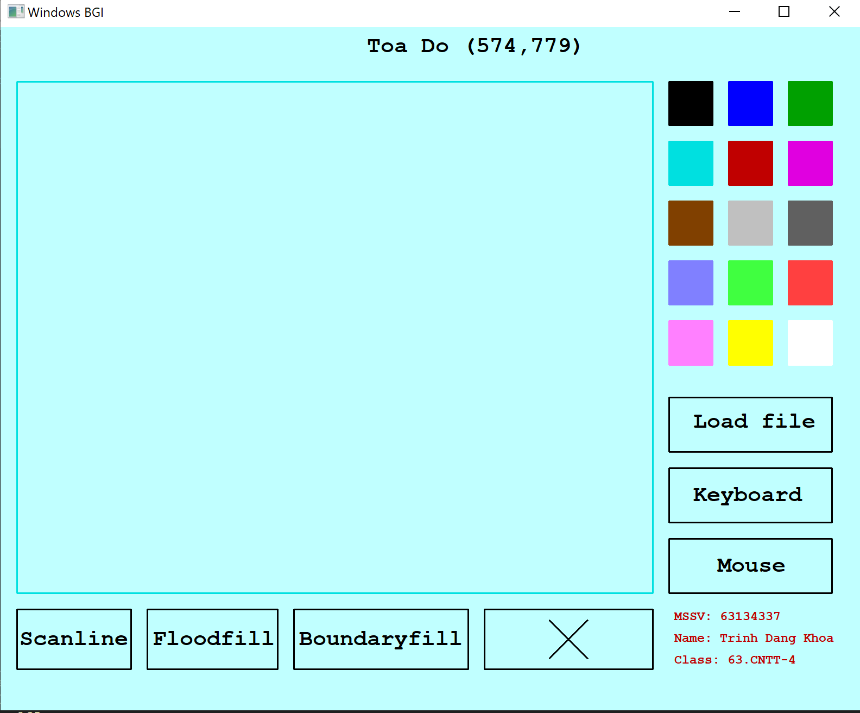
Hình 2.14. Các nút chức năng

#### A screen shot of a computer code Description automatically generated2.1.2.5 Tạo thông tin tác giả

Hình 2.15. Code thông tin tác giả

Sử dụng các hàm trong thư viện graphics.h để tạo thông tin tác giả và được viết trong hàm *GUI\_init()*:

* *settextstyle(10, 0, 3)*: tham số thứ nhất dùng để xác định kiểu chữ (10 là kiểu bold\_font), tham số thứ 2 dùng để xác định chiều chữ (0 là từ trái sang phải) và tham số cuối dùng để xác định cỡ chữ.
* *setcolor(RED)*: tạo màu cho chữ.
* Khai báo các chuỗi và dùng hàm *outtextxy()* để hiển thị các chuỗi với vị trí khác nhau lên cửa sổ.



Hình 2.16. Thông tin tác giả

## 2.2 XÂY DỰNG CÁC HÀM CHÍNH

### 2.2.1 Hàm nhập dữ liệu bằng bàn phím

A computer screen shot of text

Description automatically generated

Hình 2.17. Code hàm keyboard()

* Nhập số đỉnh và các đỉnh của đa giác cần vẽ với điều kiện các đỉnh đó phải nằm trong khung hiển thị (30 < x < 1200, 100 < y < 1040), nếu không đúng điều kiện thì bắt người dùng nhập lại.
* Nối đỉnh đầu và đỉnh cuối của đa giác lại với nhau.
* Hiển thị tọa độ các đỉnh của đa giác lên cửa sổ console.

### 2.2.2 Hàm nhập dữ liệu từ file

A computer screen shot of a program code

Description automatically generated

Hình 2.18. Code hàm readfile()

* Mở file để đọc nếu không tìm thấy file thì in ra màn hình “File not found”.
* Lấy từ file số đỉnh của đa giác và các đỉnh của đa giác.
* Nối đỉnh đầu và đỉnh cuối của đa giác lại với nhau.
* Hiển thị tọa độ các đỉnh của đa giác lên cửa sổ console.

### 2.2.3 Hàm vẽ đa giác

A screen shot of a computer code

Description automatically generated

Hình 2.19. Code hàm drawpoly()

* Sử dụng vòng lặp để vẽ đa giác với i đi từ 0 tới n-1 (biểu thị số đỉnh của đa giác).
* *setcolor(0)*: tạo màu vẽ đa giác.
* *setlinestyle(1, 0 ,3)*: tham số thứ nhất dùng để dịnh dạng kiểu khung (1 là kiểu dotted\_line), tham số thứ 2 có thể bỏ qua vì tham số thứ nhất là 1(chỉ dùng khi tham số thứ nhất bằng 4), tham số thứ 3 dùng để tạo độ dày của nét vẽ (3 là 3px).
* *line(pt[i].x, pt[i].y, pt[i+1].x, pt[i+1].y)*: nối hai đỉnh của đa giác lại với nhau thành 1 đoạn thẳng.

### 2.2.4 Các hàm của thuật toán Scanline

* Hàm *hesogoc()* dùng để tính toán các hệ số góc của các cạnh đa giác:

A computer screen with many colorful text

Description automatically generated with medium confidence

Hình 2.20. Code hàm hesogoc()

* Dùng vòng lặp để tính các hệ số góc với i đi từ 0 tới n-1 (biểu thị số đỉnh của đa giác).
* Công thức để tìm hoành độ giao điểm giữa đường quét y và các cạnh của đa giác là xgd = x(đỉnh 1 hoặc đỉnh 2) + (y – y(đỉnh 1 hoặc đỉnh 2)) / m. Vì công thức có chia cho m nên m không thể bằng 0 do đó nếu tính ra m bằng 0 thì gán m = 1 và nếu m là vô cùng thì gán m bằng 9999 (vì độ phân giải của màn hình là 1600x1200 nên 9999 là vượt khỏi size màn hình).
* Nếu không nằm trong hai trường hợp đó thì tính hệ số góc bình thường với công thức m = (ysau – ytruoc)/(xsau – xtruoc).
* Hiển thị các hệ số góc của các cạnh lên cửa sổ console.
* *A computer screen with text and symbols

  Description automatically generated*Hàm *minmax()* dùng để tìm ymin và ymax của đa giác:

Hình 2.21. Code hàm minmax()

* Gán giá trị ymin và ymax ban đầu là y của đỉnh thứ nhất.
* Dùng vòng lặp để tìm ymin và ymax với i đi từ 0 đến n-1 (biểu thị số đỉnh của đa giác).
* Hiển thị ymin và ymax lên cửa sổ console.
* A computer screen shot of text

  Description automatically generatedHàm *scanline()* chạy thuật toán Scanline:

Hình 2.22. Code hàm scanline()

* Khai báo một mảng kiểu int lưu các hoành độ giao điểm.
* Dùng vòng lặp đi từ ymin đến ymax (biểu thị cho dòng quét y đi từ ymin đến ymax của đa giác).
* Dùng vòng lặp để tính các hoành độ giao điểm của các cạnh đa giác và dòng quét y với i đi từ 0 đến n-1 (biểu thị cho số đỉnh của đa giác).
* Kiểm tra xem dòng quét y có nằm dưới hoặc trên cả 2 tung độ đỉnh của 1 cạnh đa giác không, nếu có thì hiển thị “Không tồn tại giao điểm” lên cửa sổ console.
* Nếu không thì tính hoành độ giao điểm với công thức xgd = x(đỉnh 1 hoặc đỉnh 2) + (y – y(đỉnh 1 hoặc đỉnh 2)) / m.
* Hiển thị các hoành độ giao điểm lên cửa sổ console.
* Sắp xếp các hoành độ giao điểm theo thứ tự tăng dần bằng thuật toán Bubble Sort.
* Nối các đỉnh của hoành độ giao điểm lại với nhau.
* *delay(10)*: để tạo độ trễ giữa mỗi lần quét là 10 mili giây.

### 2.2.5. Các hàm của thuật toán Flood Fill

* Hàm *fillright()* tô màu toàn bộ bên phải của đa giác:

*A computer screen shot of a program code

Description automatically generated*

Hình 2.23. Code hàm fillright()

* Hai tham số cần truyền là tọa độ của đỉnh xuất phát để tô màu.
* Sử dụng đệ quy để tô màu hết các pixel ở phía bên phải.
* Dừng lại khi không tìm thấy pixel có màu trùng với màu ban đầu của pixel xuất phát.
* Hàm *fillleft()* tô màu toàn bộ bên trái của đa giác:

A computer screen shot of a program code

Description automatically generated

Hình 2.24. Code hàm fillleft()

* Hai tham số cần truyền là tọa độ của đỉnh xuất phát để tô màu.
* Sử dụng đệ quy để tô màu hết các pixel ở phía bên trái.
* Dừng lại khi không tìm thấy pixel có màu trùng với màu ban đầu của pixel xuất phát.
* Hàm floodfill() chạy thuật toán Flood Fill:

A screen shot of a computer code

Description automatically generated

Hình 2.25. Code hàm floodfill()

### 2.2.6 Các hàm của thuật toán Boundary Fill

* Hàm *boundaryright()* tô màu toàn bộ bên phải của đa giác:

A computer screen shot of a program code

Description automatically generated

Hình 2.26. Code hàm boundaryright()

* Hai tham số cần truyền là tọa độ của đỉnh xuất phát để tô màu.
* Sử dụng đệ quy để tô màu hết các pixel ở phía bên phải.
* Dừng lại khi không tìm thấy pixel có màu khác màu cần tô và pixel có màu khác với màu đường biên (mặc định màu đường biên là 0).
* Hàm *boundaryleft()* tô màu toàn bộ bên trái của đa giác:

A computer screen shot of a program code

Description automatically generated

Hình 2.27. Code hàm boundaryleft()

* Hai tham số cần truyền là tọa độ của đỉnh xuất phát để tô màu.
* Sử dụng đệ quy để tô màu hết các pixel ở phía bên trái.
* Dừng lại khi không tìm thấy pixel có màu khác màu cần tô và pixel có màu khác với màu đường biên (mặc định màu đường biên là 0).
* Hàm *boundaryfill()* chạy thuật toán Boundary Fill:

A screen shot of a computer code

Description automatically generated

Hình 2.28. Code hàm boundaryfill()

### A computer screen with text Description automatically generated2.2.7 Hàm kiểm tra 1 điểm có nằm trong 1 nút không

Hình 2.29. Code hàm check\_button()

* Các tham số cần truyền là 1 button và tọa độ của 1 điểm (x, y).
* Kiểm tra xem nếu tọa độ của 1 điểm có nằm trong 1 nút không ( btn.x1 <= x <= btn.x2 và btn.y1 <= y <= btn.y2), nếu đúng trả về true, không trả về false.

## 2.3. CÁC THAO TÁC TRONG CHƯƠNG TRÌNH

### 2.3.1 Các sự kiện khi dùng chuột

* Trong thư viện graphics.h, một số sự kiện chuột quan trọng được hỗ trợ. Dưới đây là một số sự kiện chuột cơ bản:
* WM\_LBUTTONDOWN: Sự kiện khi nút trái chuột được nhấn.
* WM\_LBUTTONUP: Sự kiện khi nút trái chuột được nhả.
* WM\_RBUTTONDOWN: Sự kiện khi nút phải chuột được nhấn.
* WM\_RBUTTONUP: Sự kiện khi nút phải chuột được nhả.
* WM\_MBUTTONDOWN: Sự kiện khi nút giữa chuột được nhấn.
* WM\_MBUTTONUP: Sự kiện khi nút giữa chuột được nhả.
* WM\_MOUSEMOVE: Sự kiện khi chuột di chuyển.
* WM\_MOUSEWHEEL: Sự kiện khi bánh xe chuột được cuộn.
* Thư viện còn cung cấp các hàm để xử lý các sự kiện chuột như:
  + *ismouseclick(int kind)*: Hàm này trả về 1 nếu có sự kiện chuột xảy ra với loại nút chuột kind, ngược lại trả về 0.
  + *getmouseclick(int kind, int& x, int& y)*: Hàm này lấy vị trí của sự kiện chuột loại kind và lưu trữ nó vào x và y.
  + *clearmouseclick(int kind)*: Hàm này xóa thông tin về sự kiện chuột loại kind.

### 2.3.2 Tạo hiệu ứng khi di chuyển chuột

* Xử lý xự kiện di chuyển chuột

A black background with colorful text

Description automatically generated

Hình 2.30. Code xử lý sự kiện khi di chuyển chuột

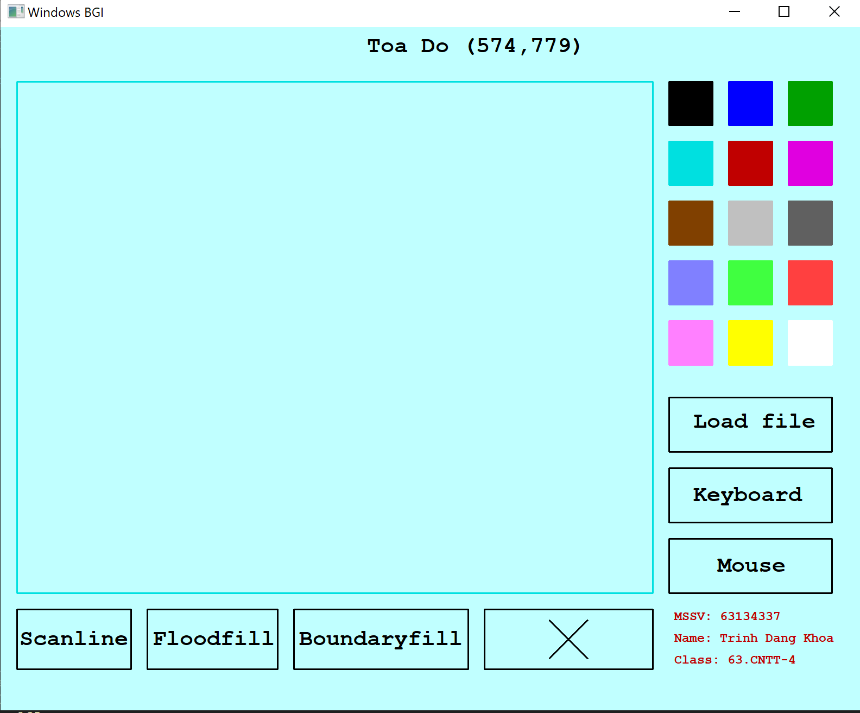
* + *ismouseclick(WM\_MOUSEMOVE)*: trả về 1 nếu đang di chuyển chuột.
  + *getmousclick(WM\_MOUSEMOVE, x, y)*:lấy vị trí của chuột khi di chuyển và gán vào x, y.
* Hiển thị tọa độ khi di chuyển chuột

A black background with multicolored text

Description automatically generated

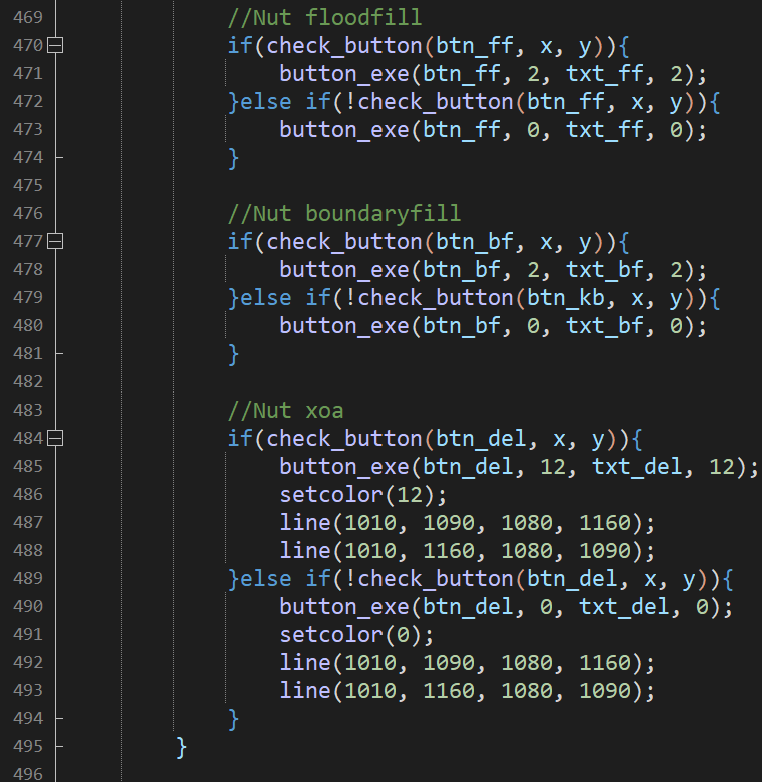
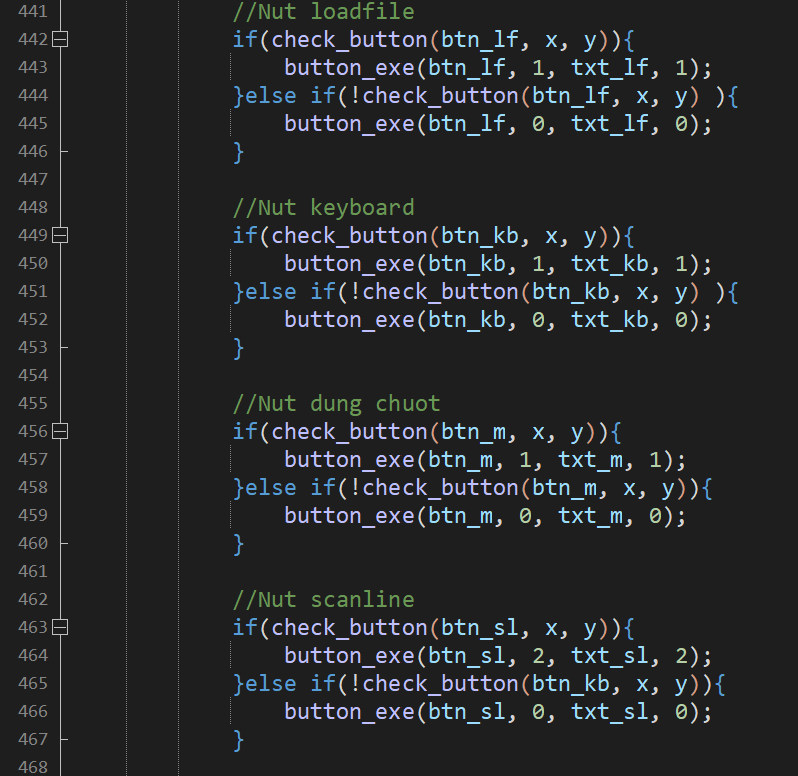
Hình 2.31. Code hiển thị tọa độ khi di chuyển chuột

* *setcolor(0)*: tạo màu cho tọa độ hiển thị
* *sprintf(s, “\t Toa Do (%d,%d) \t”, x, y)*: chuyển tọa độ x, y (vị trí của chuột khi di chuyển) thành chuỗi theo định dạng của nó (%d) và chèn vào chuỗi “\t Toa Do (%d,%d) \t” rồi gán chuỗi đó vào s.
* *outtextxy(650, 10, s)*: hiển thị chuỗi s lên màn hình với vị trí (650, 10).

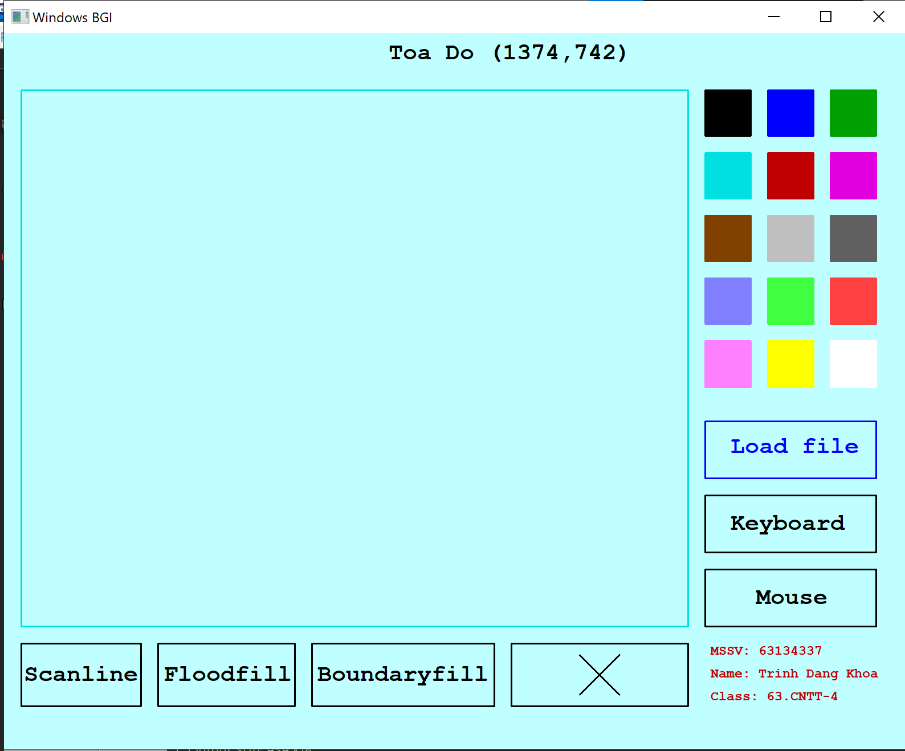


Hình 2.32. Hiển thị tọa độ khi di chuyển chuột

* Tạo hiệu ứng hover các nút. Kiểm tra xem vị trí của chuột (được lấy từ xử lý sự kiện di chuyển chuột) có nằm trong nút đó khi chuột di chuyển không, nếu có thì màu nút sẽ thay đổi còn không thì vẫn giữ nguyên nút ban đầu.



Hình 2.33. Code tạo hiệu ứng hover nút



Hình 2.34. Hiệu ứng hover các nút

### 2.3.3 Thao tác với các nút nhấn

* Xử lý sự kiện nhấn chuột trái

A black background with colorful text

Description automatically generated

Hình 2.35. Code xử lý sự kiện nhấn chuột trái

* + *ismouseclick(WM\_LBUTTONDOWN)*: trả về 1 nếu nhấn chuột trái.
  + *getmousclick(WM\_LBUTTONDOWN, x, y)*:lấy vị trí của chuột khi nhấn chuột trái và gán vào x, y.
* Nhấn nút Load file. Kiểm tra xem vị trí của chuột (được lấy từ xử lý sự kiện nhấn chuột trái) có nằm trong nút Load file khi nhấn chuột trái không, nếu có thì thực hiện hàm *readfile()* và *drawpoly()*.

A screen shot of a computer code

Description automatically generated

Hình 2.36. Code nhấn nút Load file

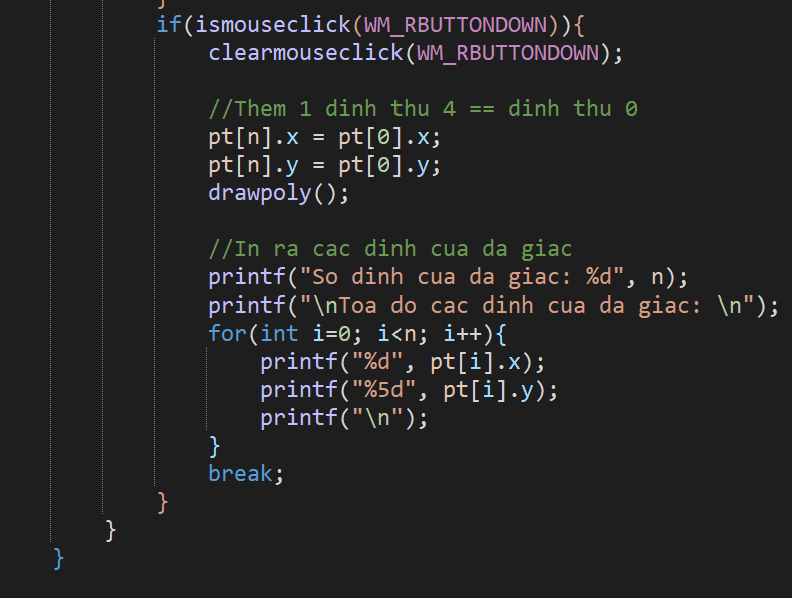
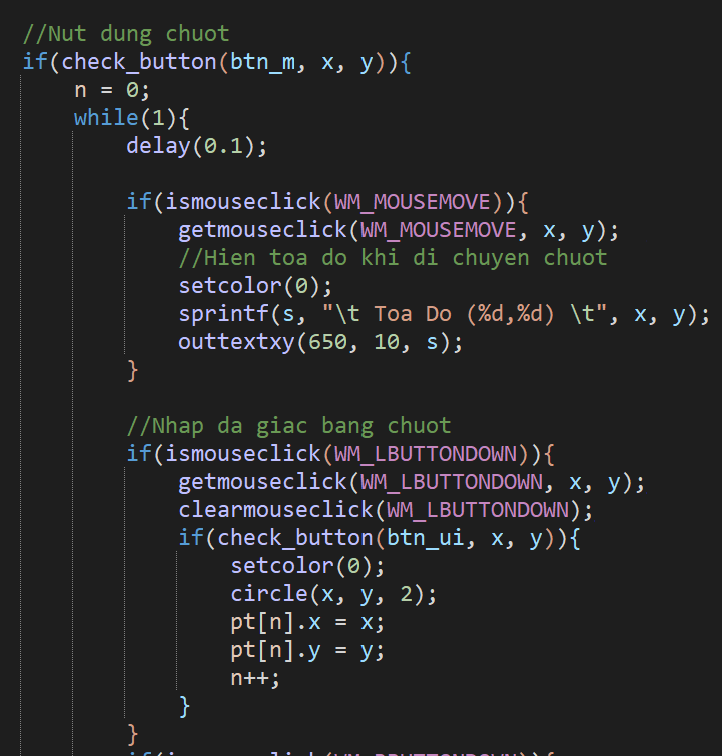
* Nhấn nút Keyboard. Kiểm tra xem vị trí của chuột (được lấy từ xử lý sự kiện nhấn chuột trái) có nằm trong nút Keyboard khi nhấn chuột trái không, nếu có thì thực hiện hàm *keyboard()* và *drawpoly()*.

A screen shot of a computer code

Description automatically generated

Hình 2.37. Code nhấn nút Keyboard

* Nhấn nút Mouse



Hình 2.38. Code nhấn nút Mouse

* + Kiểm tra xem vị trí của chuột (được lấy từ xử lý sự kiện nhấn chuột trái) có nằm trong nút Mouse khi nhấn chuột trái không.
  + Nếu có thì sử dụng sự kiện nhấn chuột trái để vẽ và lấy vị trí các đỉnh của đa giác.
  + Khi vẽ xong các đỉnh thì nhấn chuột phải vào bất kỳ chỗ nào trên cửa sổ BGI thì sẽ vẽ đa giác và hiển thị tọa độ lên cửa sổ console.
* A screen shot of a computer code

  Description automatically generatedNhấn các nút màu

Hình 2.39. Code nhấn các nút màu

* + Dùng vòng lặp với i từ 0 đến 15 (biểu thị cho 16 màu) nhưng bỏ qua i = 11 (11 trùng với màu nền cửa sổ BGI).
  + Kiểm tra xem vị trí của chuột (được lấy từ xử lý sự kiện nhấn chuột trái) có nằm trong nút màu khi nhấn chuột trái không, nếu có thì gán biến toàn cục color bằng i.
* Nhấn nút Scanline. Kiểm tra xem vị trí của chuột (được lấy từ xử lý sự kiện nhấn chuột trái) có nằm trong nút Scanline khi nhấn chuột trái không, nếu có thì thực hiện hàm *scanline().*

A computer code on a black background

Description automatically generated

Hình 2.40. Code nhấn nút Scanline

* A computer screen shot of a program code

  Description automatically generatedNhấn nút Floodfill

Hình 2.41. Code nhấn nút Floodfill

* + Kiểm tra xem vị trí của chuột (được lấy từ xử lý sự kiện nhấn chuột trái) có nằm trong nút Floodfill khi nhấn chuột trái không.
  + Nếu có thì sử dụng sự kiện nhấn chuột trái xác định vị trí và màu ban đầu của điểm xuất phát (điểm xuất phát chỉ được nằm trong khung hiển thị).
  + Tô màu đa giác bằng hàm floodfill().
* Nhấn nút boundaryfill

A computer screen shot of text

Description automatically generated

Hình 2.42. Code nhấn nút Boundaryfill

* + Kiểm tra xem vị trí của chuột (được lấy từ xử lý sự kiện nhấn chuột trái) có nằm trong nút Boundaryfill khi nhấn chuột trái không.
  + Nếu có thì sử dụng sự kiện nhấn chuột trái xác định vị trí của điểm xuất phát (điểm xuất phát chỉ được nằm trong khung hiển thị).
  + Tô màu đa giác bằng hàm boundaryfill().

# Chương 3. KẾT QUẢ THỰC HIỆN

* A screenshot of a computer

  Description automatically generatedKhi nhấn nút Load file

Hình 3.1. Vẽ đa giác khi nhấn nút Load file

* Khi nhấn nút Keyboard thì chuyển sang cửa sổ concole để nhập dữ liệu

A screenshot of a computer

Description automatically generated

Hình 3.2. Cửa sổ console để nhập dữ liệu

A screenshot of a computer

Description automatically generated

Hình 3.3. Vẽ đa giác khi nhập dữ liệu

* Khi nhấn nút Mouse ta có thể nhấn chuột trái để vẽ các đỉnh của đa giác trong khung hiển thị.

A screenshot of a computer

Description automatically generated

Hình 3.4. Vẽ các đỉnh của đa giác bằng chuột

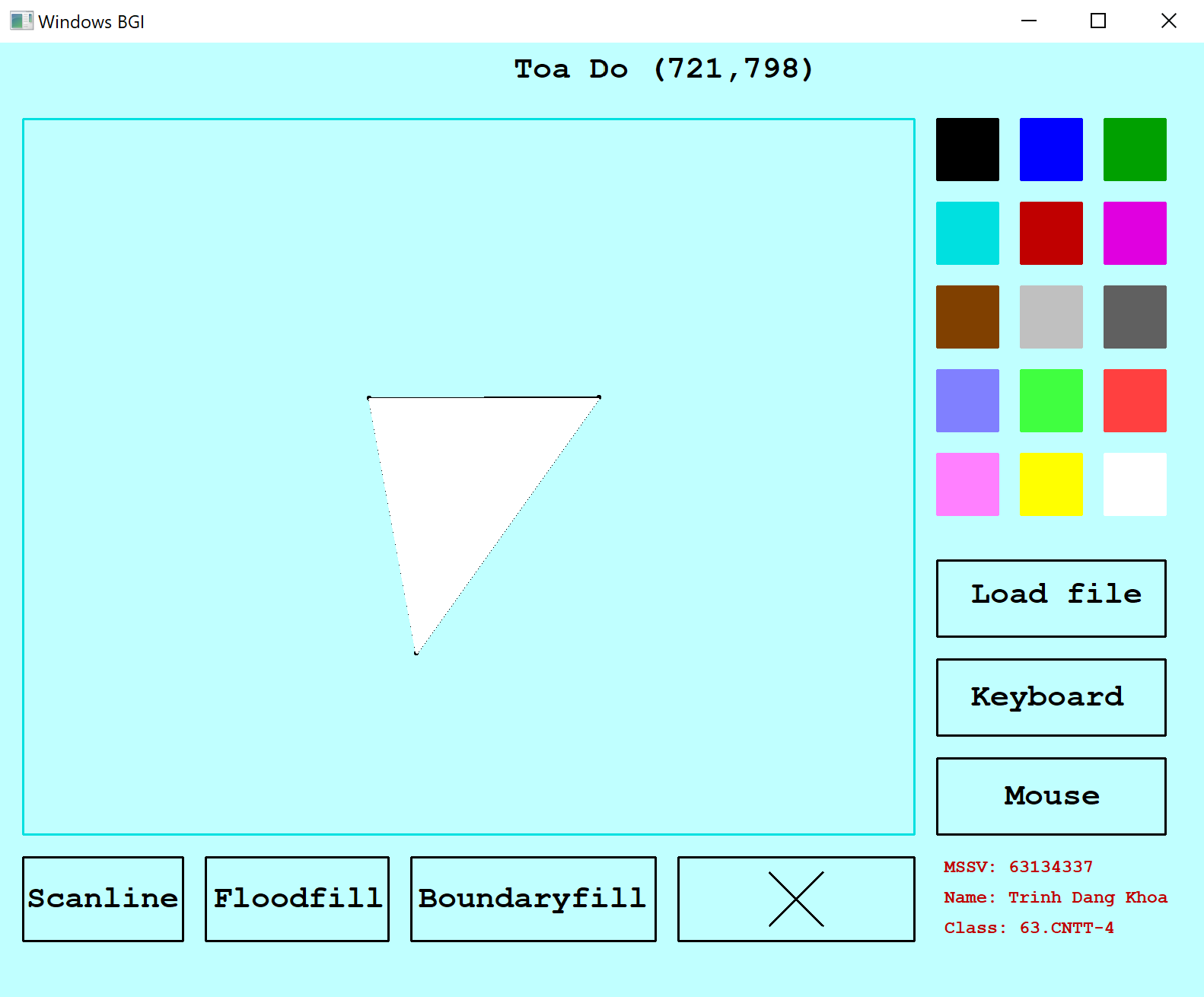
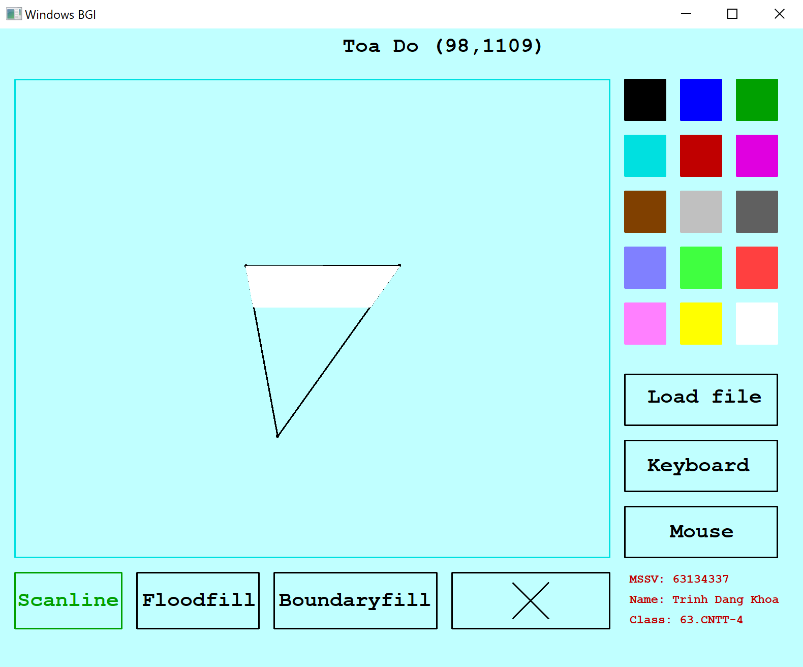
* Sau đó nhấn chuột phải vào bất kỳ điểm nào trong cửa sổ BGI để vẽ đa giác

A screenshot of a computer

Description automatically generated

Hình 3.5. Vẽ đa giác bằng chuột

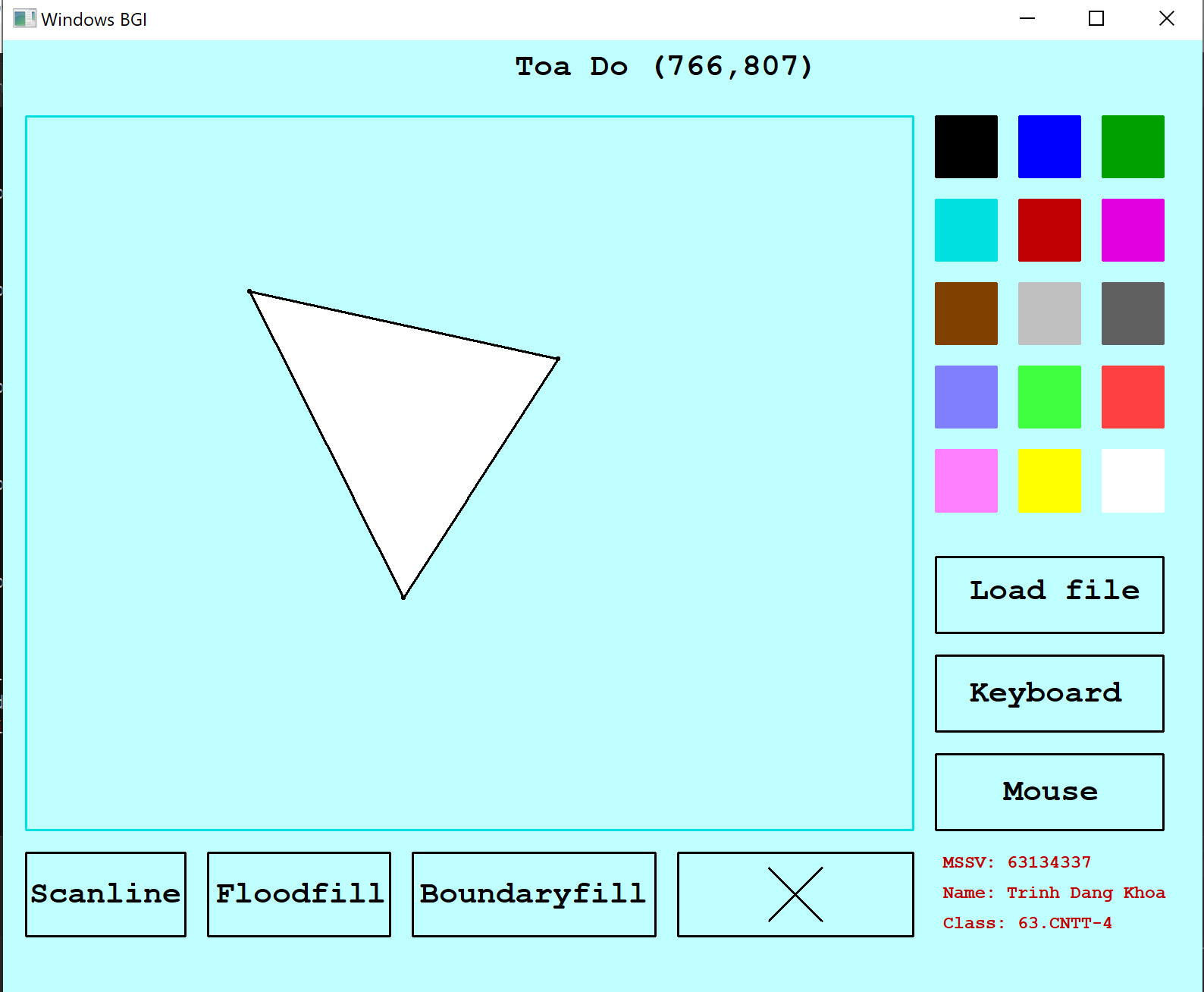
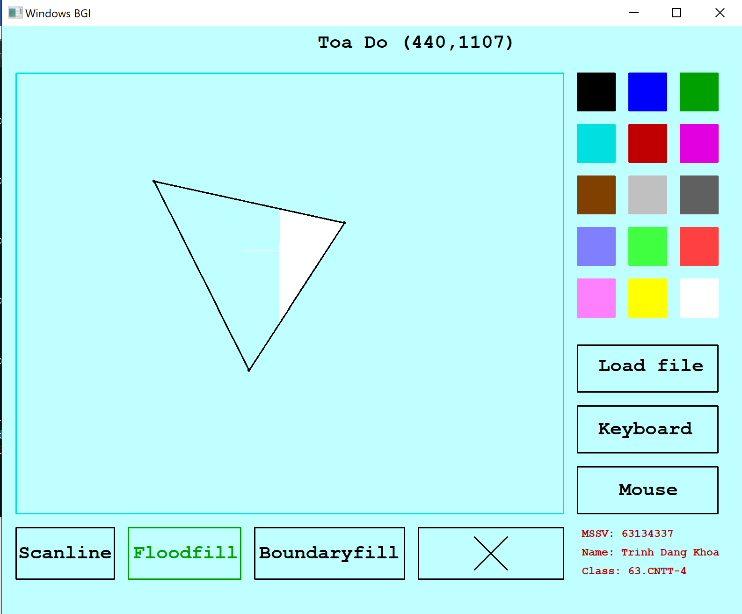
* Khi nhấn nút Scanline



Hình 3.6. Chạy thuật toán Scanline

* Khi nhấn nút Floodfill ta sẽ chọn điểm xuất phát trong đa giác cần tô màu rồi tiến hành tô màu cho đa giác.

Hình 3.7. Chạy thuật toán Flood Fill khi điểm xuất phát bên trong đa giác



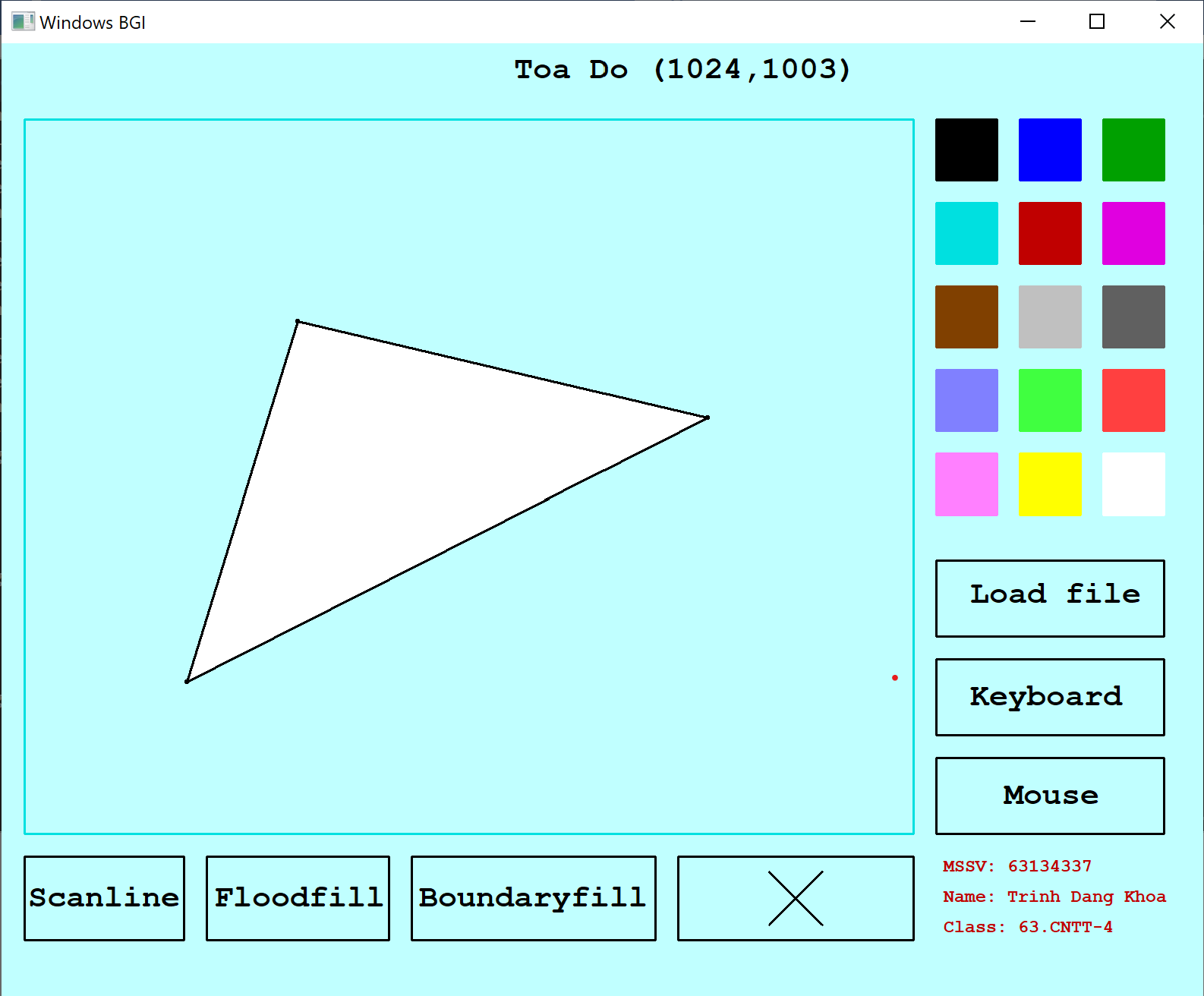
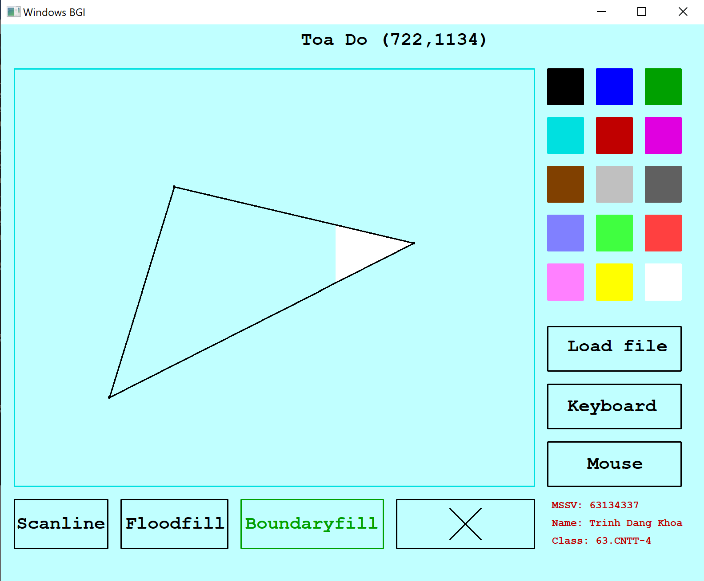
* Nếu điểm xuất phát nằm ngoài đa giác sẽ tô bên ngoài khung hiển thị (nhược điểm của thuật toán Flood Fill)

A screenshot of a computer

Description automatically generated

Hình 3.8. Chạy thuật toán Flood Fill khi điểm xuất phát bên ngoài đa giác

* Khi nhất nút Boundaryfill ta sẽ chọn điểm xuất phát trong đa giác cần tô màu rồi tiến hành tô màu cho đa giác.



Hình 3.9. Chạy thuật toán Boundary Fill khi điểm xuất phát bên trong đa giác

* Nếu điểm xuất phát nằm ngoài đa giác thì sẽ không thể tô màu cho đa giác (nhược điểm của thuật toán Boundary Fill)

A screenshot of a computer

Description automatically generated

Hình 3.10. Chạy thuật toán Boundary Fill khi điểm xuất phát bên ngoài đa giác

# Chương 4. KẾT LUẬN

## 4.1 KẾT QUẢ ĐẠT ĐƯỢC

* Mô phỏng được các thuật toán tô màu bằng thư viện đồ họa graphics.h.
* Giao diện chương trình mô phỏng dễ thao tác, dễ hiểu.
* Rèn luyện thêm các kỹ năng vẽ đồ họa của môn Kỹ thuật đồ họa.
* Hiểu sâu hơn về cách hoạt động của các thuật toán tô màu.

## 4.2 NHƯỢC ĐIỂM

* Khi chạy thuật toán Flood Fill và Boundary Fill nếu điểm xuất phát nằm bên ngoài đa giác thì không thể tô màu cho đa giác.
* Nếu vẽ nhiều đa giác thì thuật toán Scanline chỉ chạy trên đa giác vừa mới vẽ.

# TÀI LIỆU THAM KHẢO

[1] Đoàn Vũ Thịnh, *Bài giảng Kỹ thuật đồ họa*, 2019, Đại học Nha Trang.

[2] Donald Hearn, *Computer Graphics, C Version (2nd Edition)*, 1996.

[3] Đoàn Vũ Thịnh, *Hướng dẫn giải chi tiết và lập trình Kỹ thuật đồ hoạ*, 2021.

[4] Lê Minh Hoàng, *Giải thuật và lập trình*, 2006.