

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN



Môn học: Đồ họa máy tính

BÁO CÁO BÀI TẬP THỰC HÀNH 2

Vẽ đối tượng 2D và tô màu

Giảng viên phụ trách: Thầy Trần Thái Sơn - Thầy Võ Hoài Việt



Thành viên

Thứ tự	Họ và tên	MSSV
1	Phạm Long Khánh	21120479



Mục lục

1	Cài đặt thư viện và môi trường	4
2	Xử lý chương trình (Người điều khiển)	5
2.1	Ý nghĩa	5
2.2	Một số hàm chú ý	5
3	Các hình 2D	10
3.1	Hình 2D	10
3.2	Ý tưởng để vẽ	11
4	Thuật toán tô màu	12
4.1	Khai báo đối tượng RGB	12
4.2	Thuật toán BoundaryFill	12
5	Nhận xét	14
6	Demo Hướng dẫn sử dụng	16
6.1	Demo	16
6.2	Hướng dẫn sử dụng	16
7	Các nguồn tham khảo	19



1 Cài đặt thư viện và môi trường

Phần hướng dẫn cài đặt đã được nêu ở báo cáo lần trước.

2 Xử lý chương trình (Người điều khiển)

2.1 Ý nghĩa

- Trong mã nguồn, đây là class **Moderator**, class này có nhiệm vụ điều khiển các hoạt động chính của chương trình, ví dụ như, xử lý các hàm như display(), menuColorCallBack(xử lý trong việc chọn màu), menuCallBack (xử lý của menu chính), mouse (xử lý nhấn chuột), mouseMotion(xử lý rê chuột).
- Ngoài ra nó cũng chứa nhiều biến static nhằm phục vụ cho việc điều khiển chương trình, ví dụ như isColorMode(chế độ tô màu), ShapeStorage (dùng để thêm các Shape vừa vẽ), isSelectionMode(chế độ chọn vật thể) ...
- Hàm main sẽ gọi dùng Moderator này để xử lý mọi tác vụ liên quan đến vẽ, tô và chọn các hình. Điều này không chỉ làm đoạn mã đẹp, dễ quản lý và dễ bảo trì hơn mà còn hạn chế việc sử dụng các biến toàn cục, điều mà chúng ta luôn muốn hạn chế.

2.2 Một số hàm chú ý

- MenuCallBack

```
Title
1 //ShapeStorage.h
2 class ShapeStorage {
3 public:
4     enum ShapeType {
5         LINE, RECTANGLE, TRIANGLE, //Other shapes...
6     };
7
8
9     static Shape* createShape(ShapeType type)
10 {
11     switch (type) {
12         case RECTANGLE:
13             return new Rectangle();
14         case TRIANGLE:
15             return new Triangle();
16         //Other shapes...
17     }
18 }
19 //Moderator.h
20 static void menuCallBack(int value) {
21     if(selectedShape)
22         selectedShape->setSelected(false);
23     selectedShape = NULL;
24     isColorMode = false; // Ensure color mode is off
25     if (value == SELECTION_MODE) {
26         isSelectionMode = true;
27     }
28     else {
29         isSelectionMode = false;
30         createdShape = static_cast<ShapeStorage::ShapeType>(value);
31     }
32 }
```

Đây là hàm để nhận lựa chọn của người dùng sau đó đưa vào ShapeStorage để nó trả về object tương ứng. Ngoài ra nó cũng xử lý logic liên quan đến chương trình. Ví dụ như bởi vì đây là menu chính cho 2 việc là chọn hình khác hoặc vẽ thêm hình mới nên hình đã được chọn sẽ chuyển về NULL và chế độ màu sẽ được tắt đi.

- MenuColorCallBack



```
1 static void menuColorCallback(int value) {  
2     isColorMode = true;  
3     selectedShape = NULL;  
4     switch (value) {  
5         case RED:  
6             fillColor = Colors::RED;  
7             break;  
8         case GREEN:  
9             fillColor = Colors::GREEN;  
10            break;  
11         case BLUE:  
12             fillColor = Colors::BLUE;  
13            break;  
14         //...Other colors  
15     }  
16 }
```

Đây là hàm để nhận lựa chọn màu muốn tô của người dùng sau đó trả về màu tương ứng. Ngoài ra nó cũng xử lý logic liên quan đến chương trình.

- createMenu

```
Title

1 static void createMenu(void) {
2     int colorSubMenu = glutCreateMenu(menuColorCallback);
3     glutSetMenu(colorSubMenu); // Set the current menu to the triangle submenu
4     glutAddMenuEntry("Red", RED);
5     glutAddMenuEntry("Green", GREEN);
6     glutAddMenuEntry("Blue", BLUE);
7     //Other Colors
8
9
10    int triangleSubMenu = glutCreateMenu(menuCallback); // Submenu for triangles
11    glutSetMenu(triangleSubMenu); // Set the current menu to the triangle submenu
12    glutAddMenuEntry("Equilateral Triangle", ShapeStorage::EQUIL_TRIANGLE);
13    glutAddMenuEntry("Isosceles Triangle", ShapeStorage::ISOSCELES_TRIANGLE);
14
15    //..Other Shapes
16
17
18    int menu_id = glutCreateMenu(menuCallback);
19    glutAddSubMenu("Color", colorSubMenu);
20    glutAddMenuEntry("Line", ShapeStorage::LINE);
21    glutAddSubMenu("Triangle", triangleSubMenu);
22    //...
23    glutAddMenuEntry("Selection Mode", SELECTION_MODE);
24    glutAttachMenu(GLUT_RIGHT_BUTTON);
25 }
```

Thêm các màu và các hình vào Menu chính, đây là menu hiển thị các chọn lựa của người dùng. Ngoài ra nó cũng gắn thêm lựa chọn để select object vào Menu.

- mouse

```
1 static void mouse(int button, int state, int x, int y) {
2     if (button == GLUT_LEFT_BUTTON) {
3         if (state == GLUT_DOWN) {
4             if (isColorMode) { // Color Mode
5                 //...
6                 boundaryFill(xi, yi, fillColor, borderColor, newShape);
7                 //...
8             }
9             else if (isSelectionMode) // Selection Mode
10            {
11                //...
12                selectedShape = shapeStorage.selectShapeByPosition(x, y);
13                glutPostRedisplay();
14            }
15            else // Drawing Mode
16            {
17                glColor3f(borderColor.r(), borderColor.g(), borderColor.b());
18                isDragging = true;
19                newShape = ShapeStorage::createShapeType(createdShape, x, y);
20                shapeStorage.addShape(newShape);
21            }
22        }
23    }
24    else if (state == GLUT_UP && isDragging) {
25        isDragging = false; // Set dragging to false here
26        glutPostRedisplay();
27    }
28 }
29 }
```

Đây là hàm để xử lý các thao tác với chuột trong 2 trạng thái: khi được nhấn xuống và đang được giữ (chưa nhấc lên). Trong trạng thái nhấn xuống, hàm sẽ xét thêm 3 trường hợp

- Nếu đang ở chế độ tô màu, hàm sẽ thực thi thuật toán tô màu, và gán màu đó cho hình được click
- Nếu đang chế độ chọn hình, hình được chọn sẽ có đường viền khác với hình còn lại.
- Nếu đang ở chế độ vẽ hình mới, hình mới sẽ được tạo và thêm vào ShapeStorage.

Nếu đang ở trạng thái giữ, hình được vẽ mới sẽ thay đổi khi theo động tác rê chuột.

- mouseMotion

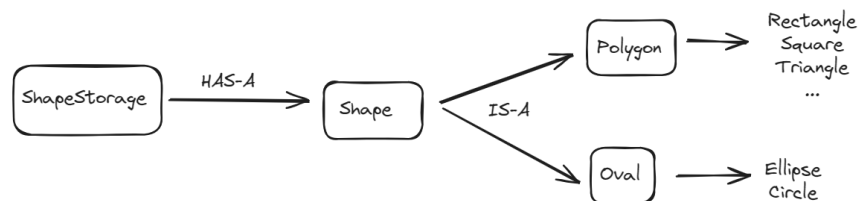

```
Title
1 static void mouseMotion(int x, int y) {
2     if (newShape && isDragging)
3     {
4         newShape->setEnd(x, y);
5         glutPostRedisplay();
6     }
7 }
```

Đây là hàm cho động tác rê chuột, với mỗi lần chuột được rê thì điểm kết thúc của hình sẽ được cập nhật và vẽ lại (khi đó kích thước của hình cũng thay đổi theo).

3 Các hình 2D

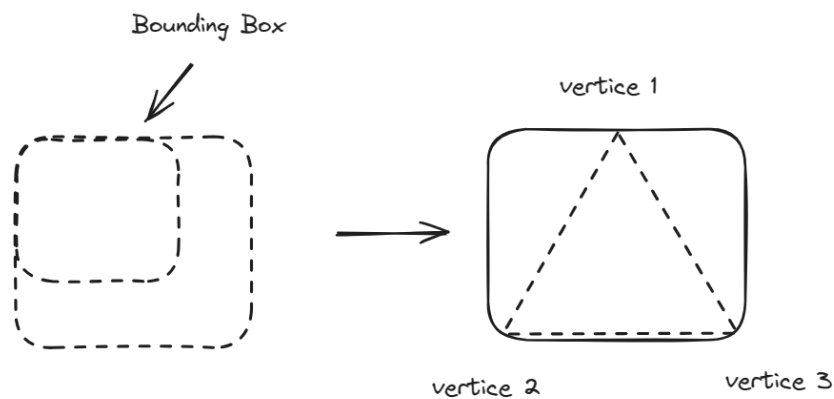
3.1 Hình 2D

- Có tổng cộng 14 hình 2D được yêu cầu vẽ gồm:
 - Hình tam giác vuông cân
 - Hình tam giác đều
 - Hình vuông
 - Hình chữ nhật
 - Hình tròn
 - Hình elip
 - Hình ngũ giác đều
 - Hình lục giác đều
 - Hình mũi tên
 - Hình ngôi sao
 - Hình dấu cộng
 - Hình dấu trừ
 - Hình dấu nhân
 - Hình dấu chia
- Hoàn thành: 14/14 hình.
- Mối quan hệ giữa các class của các hình trong mã nguồn:

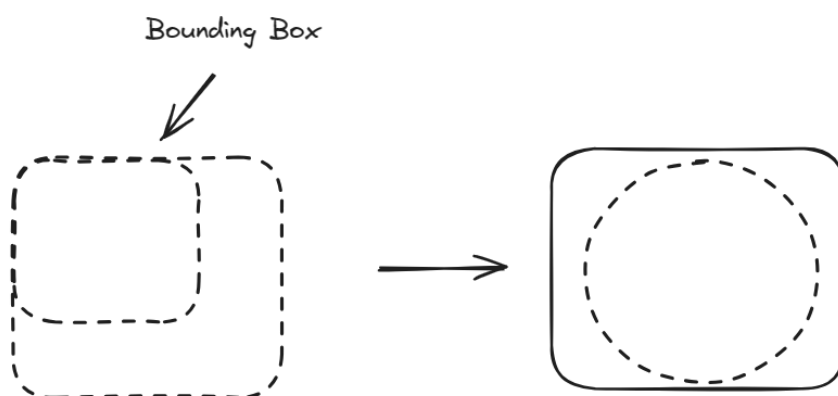


3.2 Ý tưởng để vẽ

- Đầu tiên ta cần có Bounding Box cho các hình, việc rê chuột sẽ thay đổi Bounding Box.
- Sau khi Bounding Box đã được xác định thì ta sẽ xác định các đỉnh dựa trên Bounding Box đó.
- Hàm vẽ của các hình sẽ được thực hiện bằng cách nối các đỉnh của hình bằng các đoạn thẳng được xác định bởi 2 điểm.

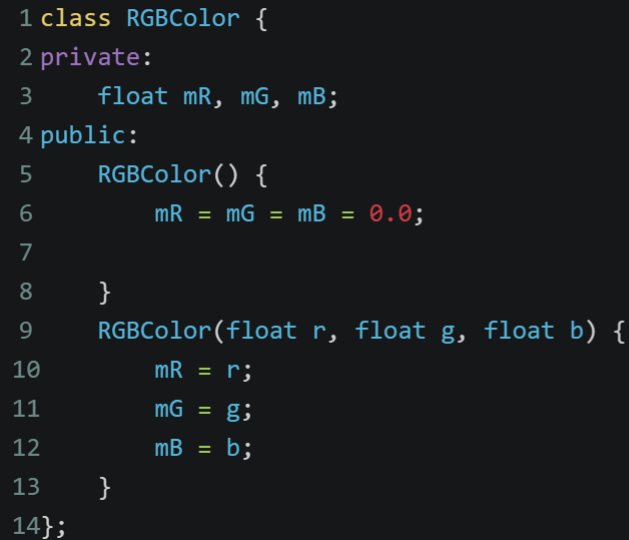


- Đối với hình tròn, nó sẽ xác định tâm và bán kính dựa trên Bounding Box để vẽ.



4 Thuật toán tô màu

4.1 Khai báo đối tượng RGB

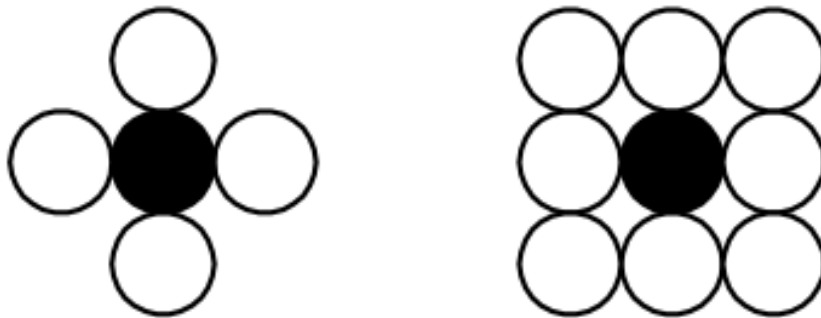


```
1 class RGBColor {
2 private:
3     float mR, mG, mB;
4 public:
5     RGBColor() {
6         mR = mG = mB = 0.0;
7     }
8     RGBColor(float r, float g, float b) {
9         mR = r;
10        mG = g;
11        mB = b;
12    }
13 };
14};
```

- Đây là một class dùng để đại diện cho màu (được biểu diễn bởi hệ màu RGB - red, green, blue).
- Ngoài ra còn có thêm 2 cách khởi tạo class này lần lượt là có tham số và không có tham số.

4.2 Thuật toán BoundaryFill

- Ý tưởng
 - Bắt đầu từ điểm được chọn để tô, thuật toán sẽ kiểm tra các điểm lân cận đã được tô hay chưa và có điểm màu trùng điểm biên hay không, nếu cả hai đều không thì thuật toán sẽ tô màu cho điểm đó.
 - Quá trình này lặp lại cho đến khi không còn điểm để tô nữa thì dừng.
- Có hai cách để thực hiện thuật toán này:
 - Dùng 4 điểm lân cận.
 - Dùng 8 điểm lân cận.



- Minh họa thuật toán như sau:

```
1 void boundaryFill(int x, int y, RGBColor fillColor, RGBColor borderColor,
2   Shape* newShape) {
3   // Non-recursive implementation
4   stack<pair<int, int>> s;
5   s.push(make_pair(x, y));
6   while (!s.empty()) {
7     pair<int, int> it = s.top();
8     s.pop();
9     int x = it.first, y = it.second;
10    RGBColor currentColor = GetPixel(x, y);
11    if (!isInScreen(x, y) || currentColor == borderColor || currentColor ==
12      fillColor)
13    {
14      continue;
15    }
16    PutPixel(x, y, fillColor);
17    Pixel filledPixel(x, y, fillColor);
18    newShape->addFilledPixel(filledPixel);
19    if (isInScreen(x + 1, y))
20      s.push(make_pair(x + 1, y));
21    if (isInScreen(x - 1, y))
22      s.push(make_pair(x - 1, y));
23    if (isInScreen(x, y + 1))
24      s.push(make_pair(x, y + 1));
25    if (isInScreen(x, y - 1))
26      s.push(make_pair(x, y - 1));
27  }
```

- * Trong đoạn mã trên, tạo một stack s để lưu trữ tọa độ các điểm xét. Thêm tọa độ điểm bắt đầu bắt đầu (x, y) vào stack (đây là tọa độ của điểm được click chuột - Do đó nếu người dùng click bên ngoài hình, chương trình sẽ bị tràn bộ nhớ).
- * Lấy các điểm bên trong stack và kiểm tra, cụ thể lấy tọa độ (x, y) và màu của điểm ảnh hiện tại các tọa độ đó. Nếu điểm ảnh nằm ngoài vùng, đã có

màu tô hoặc cùng màu với đường viền, bỏ qua và chuyển sang lần lặp tiếp theo.

- * Nếu không, tô màu điểm ảnh hiện tại bằng fillColor. Thêm các tọa độ lân cận (trên, dưới, trái, phải) vào stack, đảm bảo chúng nằm trong giới hạn màn hình.
 - * Quá trình tiếp tục cho đến khi stack rỗng, cho biết tất cả các điểm ảnh trong phạm vi đường viền đã được tô màu.
- Tự nhận xét: Do thuật toán được cài đặt dựa trên lý thuyết, do đó tính chính xác của nó khá tốt cao.

5 Nhận xét

- Tự nhận xét: độ chính xác của thuật toán tốt vì cài đặt của nó dựa trên lý thuyết.
- So sánh với các thuật toán khác được học:

– Boundary Fill:

- * Ưu điểm: Tích hợp tốt với các hình dạng có biên đều và không quá phức tạp.
Không tô đè lên biên của hình vẽ..
- * Nhược điểm: Yêu cầu xác định điểm bắt đầu và màu sắc giới hạn.
Có thể gặp vấn đề nếu không đặt đúng điểm bắt đầu, có thể dẫn đến tràn bộ nhớ hoặc vấn đề khác.

– Flood Fill

- * Ưu điểm: Tự động tìm điểm bắt đầu bằng cách sử dụng màu sắc tại điểm đó.
Tích hợp tốt với các hình dạng có biên không đều hoặc phức tạp.
- * Nhược điểm: Có thể tô đè lên biên của hình vẽ nếu không được sử dụng cẩn thận.
Cần xử lý kỹ thuật đệ quy hoặc sử dụng ngăn xếp để tránh tràn bộ nhớ.

– Scanline:

- * Ưu điểm: Hiệu suất cao với các hình dạng có các đường biên đơn giản.
Dễ tính toán và thực hiện.



* Nhược điểm: Không hiệu quả cho các hình dạng phức tạp với nhiều lỗ hoặc nhiều biên.

Có thể yêu cầu sắp xếp các điểm theo chiều dọc trước khi thực hiện quét.

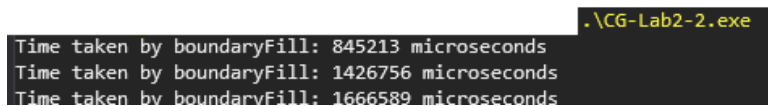
6 Demo Hướng dẫn sử dụng

6.1 Demo

Đây là demo của chương trình: [demo](#).

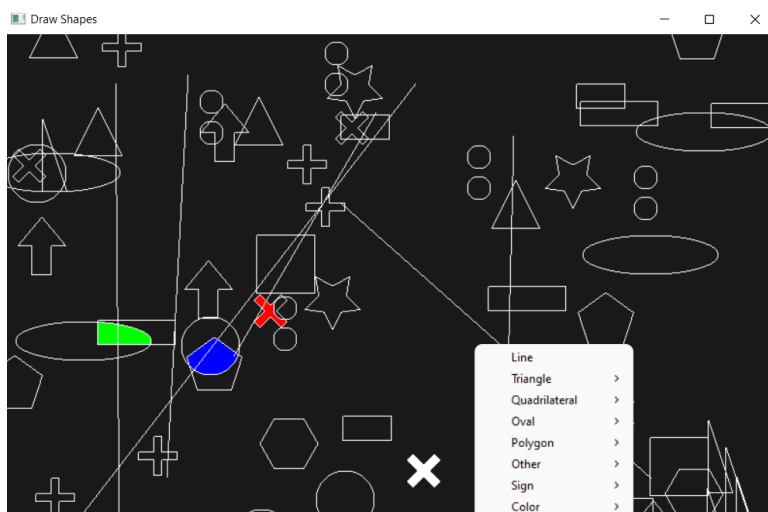
6.2 Hướng dẫn sử dụng

1. Mở folder chứa file build ở dạng Release và chạy file exe.



```
.\CG-Lab2-2.exe
Time taken by boundaryFill: 845213 microseconds
Time taken by boundaryFill: 1426756 microseconds
Time taken by boundaryFill: 1666589 microseconds
```

2. Sau đó ấn chuột phải để mở menu.



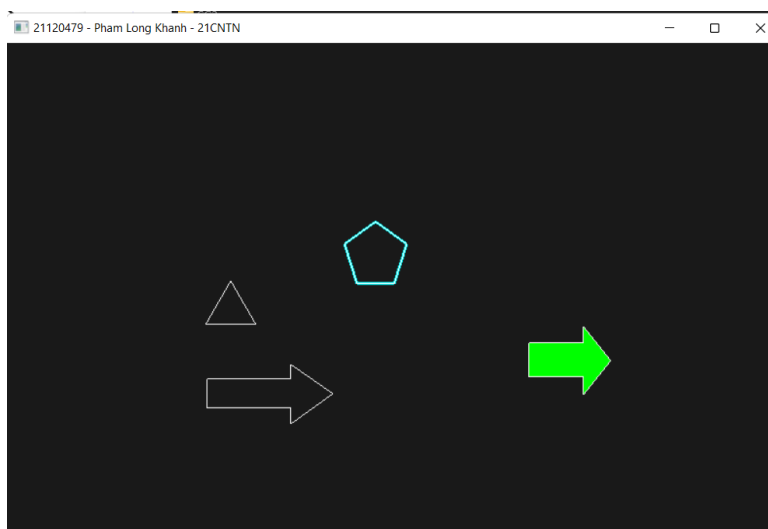
3. Tiếp theo chọn yêu cầu muốn thực hiện.

- Nếu là vẽ hình, chọn hình muốn vẽ sau đó ấn chuột trái để bắt đầu vẽ hình. Giữ chuột trái sau khi nhấn và rê để xác định kích thước của hình đang muốn vẽ
- Nếu là tô màu, chọn điểm bắt đầu tô

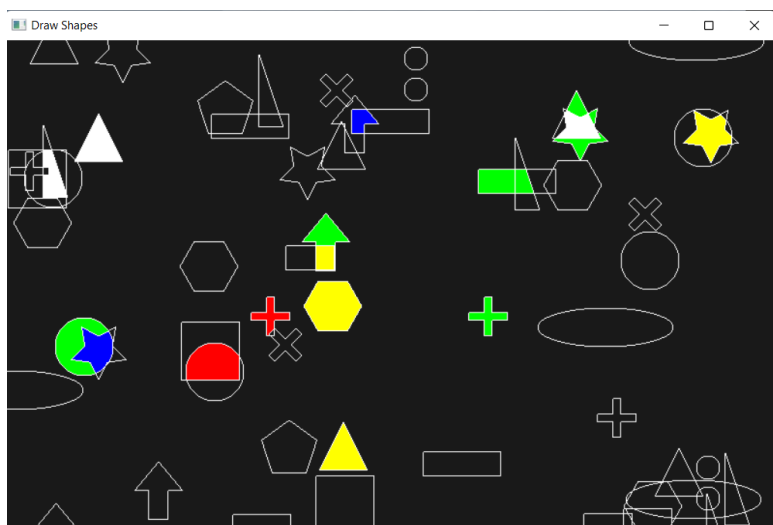
- **Chú ý:** Khi tô màu nên chọn những nơi bị giới hạn diện tích và diện tích không được quá lớn. Nếu không sẽ bị tràn stack, vì thuật toán tô màu có sử dụng đệ quy.
- Ngoài ra, thời gian chạy thuật toán tô màu sẽ được in ở màn hình Console.

```
Time taken by boundaryFill: 631779 microseconds
Time taken by boundaryFill: 1888444 microseconds
Time taken by boundaryFill: 930788 microseconds
Time taken by boundaryFill: 375415 microseconds
Time taken by boundaryFill: 416123 microseconds
Time taken by boundaryFill: 849223 microseconds
Time taken by boundaryFill: 632673 microseconds
Time taken by boundaryFill: 1043448 microseconds
Time taken by boundaryFill: 377066 microseconds
Time taken by boundaryFill: 2215867 microseconds
Time taken by boundaryFill: 1321883 microseconds
Time taken by boundaryFill: 633 microseconds
```

4. Click vào bên trong hình để để chọn hình. Hình được chọn sẽ có đường viền khác với các hình còn lại. Mỗi lần chỉ được chọn 1 hình duy nhất.



Đây là một demo sau khi vẽ các hình và tô màu một số vùng.



7 Các nguồn tham khảo

1. Slide bài giảng của thầy Trần Thái Sơn - Đại học Khoa học Tự nhiên - Đại học Quốc gia Thành phố Hồ Chí Minh (chỉ trên moodle môn học)
2. Slide bài giảng của thầy Trần Võ Hoài Việt - Đại học Khoa học Tự nhiên - Đại học Quốc gia Thành phố Hồ Chí Minh (chỉ trên moodle môn học)
3. Đồ họa máy tính - Dương Anh Đức - - Đại học Khoa học Tự nhiên - Đại học Quốc gia Thành phố Hồ Chí Minh
4. [GeeksForGeeks](#)
5. [Followtutorials](#)
6. [BoundaryFillAlgorithm](#)
7. [OpenGL Programming Guide](#)