BÀI THỰC HÀNH ĐỒ HỌA MÁY TÍNH

CÁC ĐỐI TƯỢNG ĐỒ HỌA CƠ BẢN

Họ và tên Sinh viên: Trần Đức Trí
Mã Sinh Viên: 102210996
Nhóm: 21Nh15

Nội dung

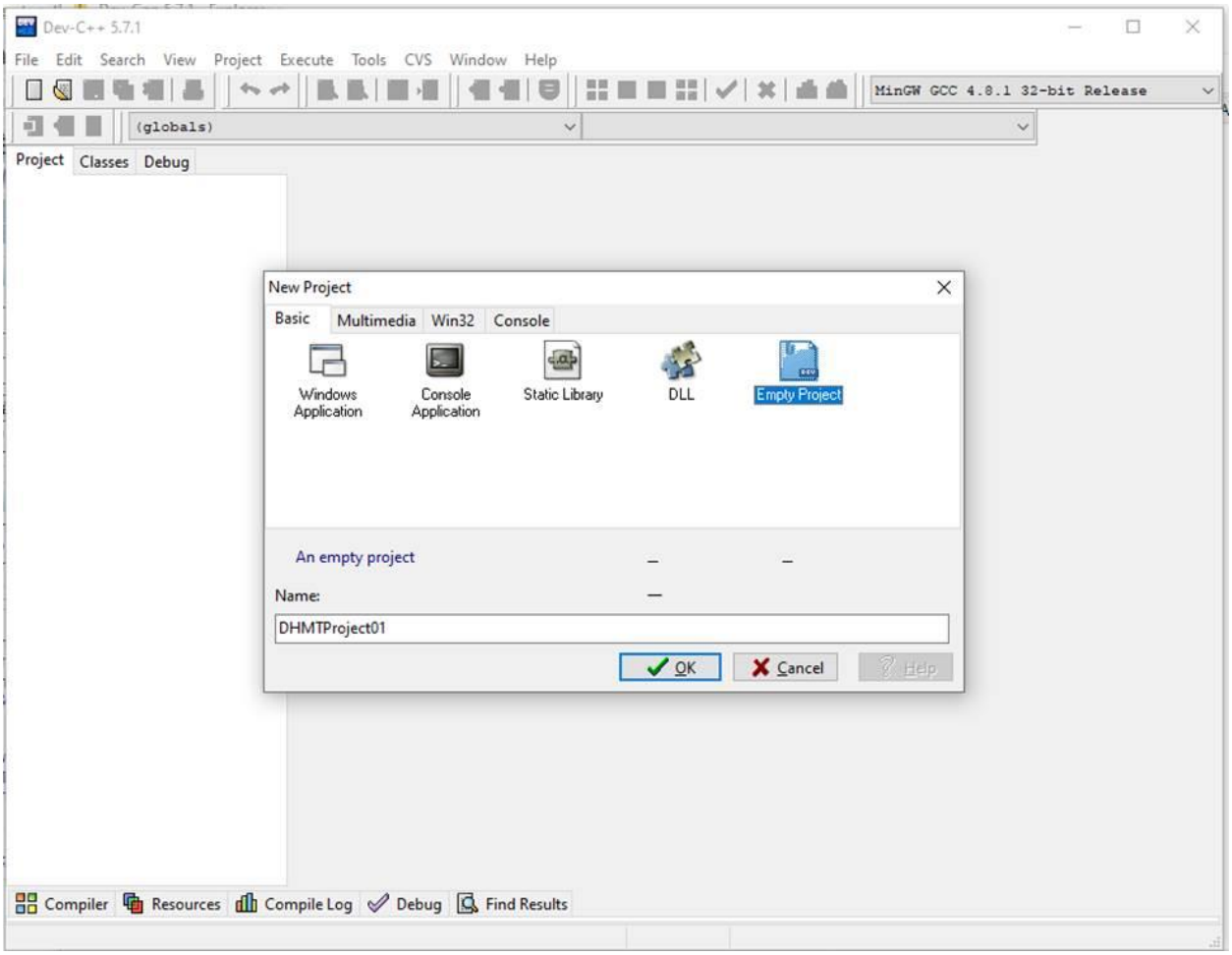
1. Cấu hình Project sử dụng thư viện OpenGL trong Dev C

- Tải file **Dev-Cpp 5.7.1.rar** và giải nén vào thư mục **C:\ Dev-Cpp 5.7.1**
- Vào thư mục Dev-Cpp 5.7.1\libOpenGL: Sao chép các file thư viện **glut32.dll, glut.dll, glut32.dll** vào hai thư mục sau:

  `C:\Windows\SysWOW64\*.*`

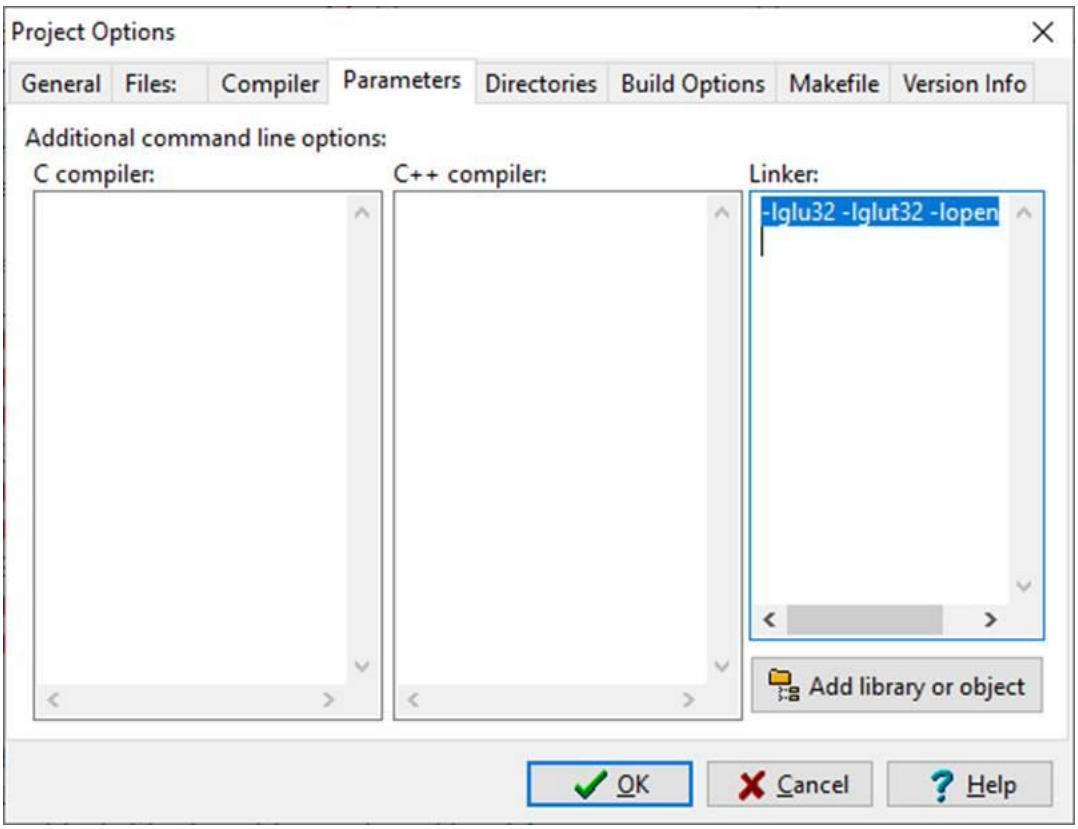
  `C:\Windows\System32\*.*`

- Tạo mới một Project: Menu *File\New\Project* => Chọn *Empty Project*. Nhập tên. OK



- Trong cửa sổ *Project Options* (hoặc nhấn Alt + P): Chọn *tab Parameter* => *Linker* khai báo như sau:

  -lglu32 -lglut32 -lopengl32



Chọn OK.

Trong Project, tạo một tập tin chương trình và lưu Lab*.cpp

2. Lý thuyết - Thư viện đồ họa OpenGL

OpenGL cung cấp một số thành phần đồ hoạ cơ sở gọi là các primitive. Cần xác định các primitive trước khi vẽ trên màn hình.

2.1 Các đối tượng đồ họa cơ sở (primitive) của OpenGL

2.1.1 Điểm : GL_POINTS

```
glBegin(GL_POINTS);
	glVertex3f(1.0f, 0.0f, 0.0f); // A
	glVertex3f(0.0f, 1.0f, 0.0f); // B
	glVertex3f(0.0f, 0.0f, 1.0f); // C
glEnd();
```

2.1.2 Đường thẳng : GL_LINES

```
glBegin(GL_LINES);
	// AB
	glVertex3f(1.0f, 0.0f, 0.0f);
	glVertex3f(0.0f, 1.0f, 0.0f);
	// AC
	glVertex3f(1.0f, 0.0f, 0.0f);
	glVertex3f(0.0f, 0.0f, 1.0f);
glEnd();
```

Hai primitive được xây dựng dựa trên GL_LINES :

- GL_LINE_STRIP : các cạnh được vẽ liên tiếp theo thứ tự các điểm đưa ra bởi glVertex.

```
glBegin(GL_LINES);
       glVertex3f(1.0f, 0.0f, 0.0f); // A
       glVertex3f(0.0f, 1.0f, 0.0f); // B
       glVertex3f(0.0f, 0.0f, 1.0f); // C
       glVertex3f(0.0f, -1.0f, 0.0f); // D
glEnd();
```

- GL_LINE_LOOP : tương tự GL_LINE_STRIP nhưng điểm kết thúc được nối với điểm khởi đầu.
- Các primitive GL_LINE … cho phép vẽ mô hình khung của nhiều hình phức tạp. Muốn vẽ được các đối tượng với các mặt được tô màu, chúng ta phải sử dụng các primitive đa giác.
- Hai mặt của đa giác trong OpenGL có thuộc tính chiều xoay của các đỉnh *(winding)* được xác định theo thứ tự các đỉnh liệt kê khi vẽ đa giác. Mặc định, mặt trước của đa giác tương ứng với chiều xoay ngược kim đồng hồ. Để thiết lập mặt trước / sau của đa giác theo chiều xoay, sử dụng hàm *glFrontFace* :

```
glFrontFace (GL_CW)          // Mặt trước theo chiều kim đồng hồ
glFrontFace (GL_CCW)  // Mặt trước ngược chiều kim đồng hồ
```

2.1.3 Tam giác : GL_TRIANGLES

```
glBegin(GL_TRIANGLES);
     // Tam giác V0V1V2
     glVertex2f(0.0f,0.0f);  // V0
     glVertex2f(1.0f,2.0f);  // V1
     glVertex2f(2.0f,0.0f);  // V2
     // Tam giác V3V4V5
     glVertex2f(-3.0f,0.0f);  // V3
     glVertex2f(-1.0f,0.0f);  // V4
     glVertex2f(-3.0f,2.0f);  // V5
glEnd();
```

Hai primitive xây dựng dựa trên GL_TRIANGLES :

- GL_TRIANGLE_STRIP : vẽ liên tiếp các tam giác. Tam giác thứ *i+1* được tạo thành từ một đỉnh mới và hai đỉnh cũ thuộc tam giác thứ *i*.

```
     //  Tam giác V0V1V2
     glVertex2f ….    // V0
     glVertex2f ….       // V1
     glVertex2f ….       // V2
          //  Tam giác V1V2V3
     glVertex2f ….       // V3
     //  Tam giác V2V3V4
     glVertex2f ….       // V4
glEnd();
```

Lưu ý : Hướng của các cạnh không nhất thiết bảo toàn trong các tam giác khác nhau.

- GL_TRIANGLE_FAN : Vẽ liên tiếp các tam giác xoay quanh đỉnh xác định đầu tiên.

```
glBegin(GL_TRIANGLE_FAN);
     //  Đỉnh tâm
     glVertex2f …..   // V0
     //  Tam giác V0V1V2
     glVertex2f …..       // V1
     glVertex2f …..       // V2
     //  Tam giác V0V2V3
     glVertex2f …..       // V3
     //  Tam giác V0V3V4
     glVertex2f …..       // V4
glEnd();
```

2.1.4 Tứ giác : GL_QUADS

Primitive này được xác định dựa trên 4 đỉnh liên tục được liệt kê, và không có đỉnh nào được phép sử dụng lại cho các tứ giác tiếp theo. Tương tự tam giác, GL_QUAD_STRIP cho phép vẽ nhiều tứ giác liên tiếp nhau, trong đó tứ giác thứ *i+1* được xác định từ 2 đỉnh mới và 2 đỉnh cũ của tứ giác thứ *i*.

2.1.5 Đa giác : GL_POLYGON

```
glBegin(GL_POLYGON);
     // Tam giác V0V1V2
     glVertex2f(0.0f,0.0f);  // V0
     glVertex2f(1.0f,2.0f);  // V1
     glVertex2f(2.0f,0.0f);  // V2
glEnd();
```

2.2 Xóa bộ đệm màu glClear()

Mỗi lần vẽ, nên dùng lệnh xóa bộ đệm màu glClear()

```
glClearColor(0.0, 0.0, 0.0, 0.0); /* xác định màu để xóa color buffer (màu đen) */
glClearDepth(1.0); /* xác định giá trị để xóa depth buffer */
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);  /* xóa color buffer và depth buffer
*/
```

2.3 Thiết lập lại màu vẽ cho đối tượng glColor3f()

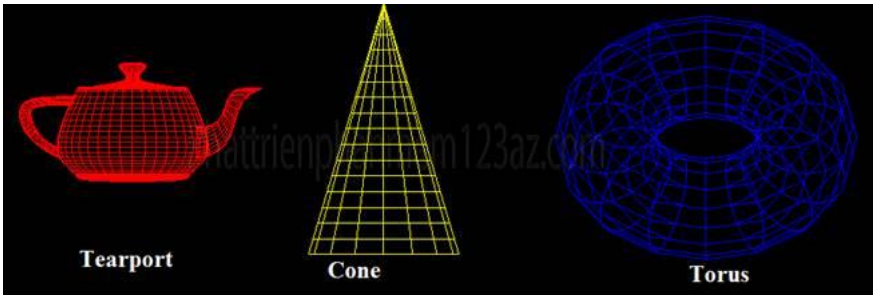
Khi vẽ một đối tượng, OpenGL sẽ tự động sử dụng màu đã được xác định trước đó. Do đó, để vẽ đối tượng với màu mới, cần phải thiết lập lại màu vẽ bằng cách dùng hàm glColor3f()

```
glColor3f(0.0, 0.0, 0.0);          // black
glColor3f(1.0, 0.0, 0.0);          // red
glColor3f(0.0, 1.0, 0.0);          // green
glColor3f(1.0, 1.0, 0.0);          // yellow
glColor3f(0.0, 0.0, 1.0);          // blue
glColor3f(1.0, 0.0, 1.0);          // magenta
glColor3f(0.0, 1.0, 1.0);          // cyan
glColor3f(1.0, 1.0, 1.0);          // white
```

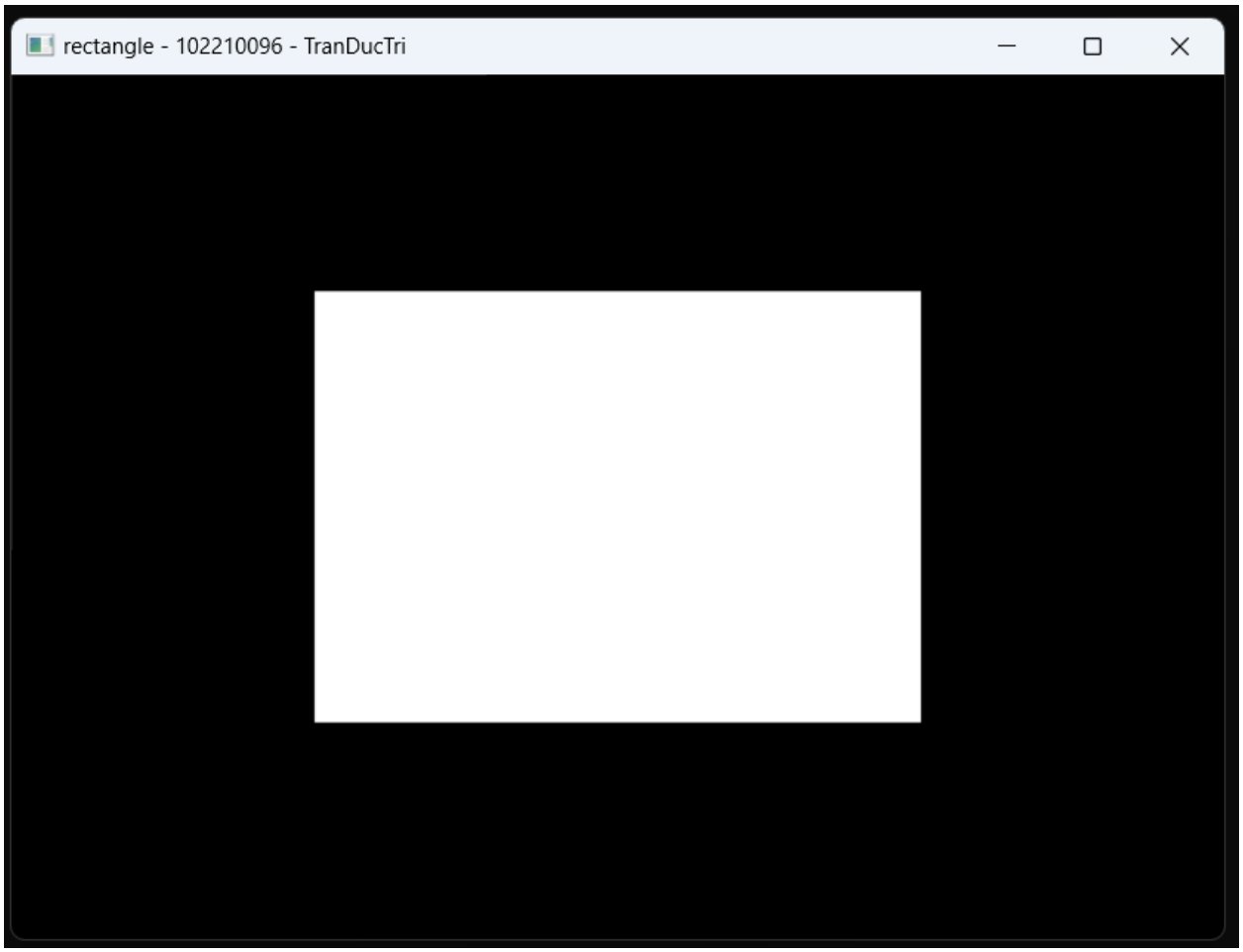
2.4 Các hàm vẽ các đối tượng hình học phức tạp trong thư viện GLUT

- void glutWireSphere(GLdouble radius, GLint slices, GLint stacks);
- void glutSolidSphere(GLdouble radius, GLint slices, GLint stacks);
- void glutWireCube(GLdouble size);
- void glutSolidCube(GLdouble size);

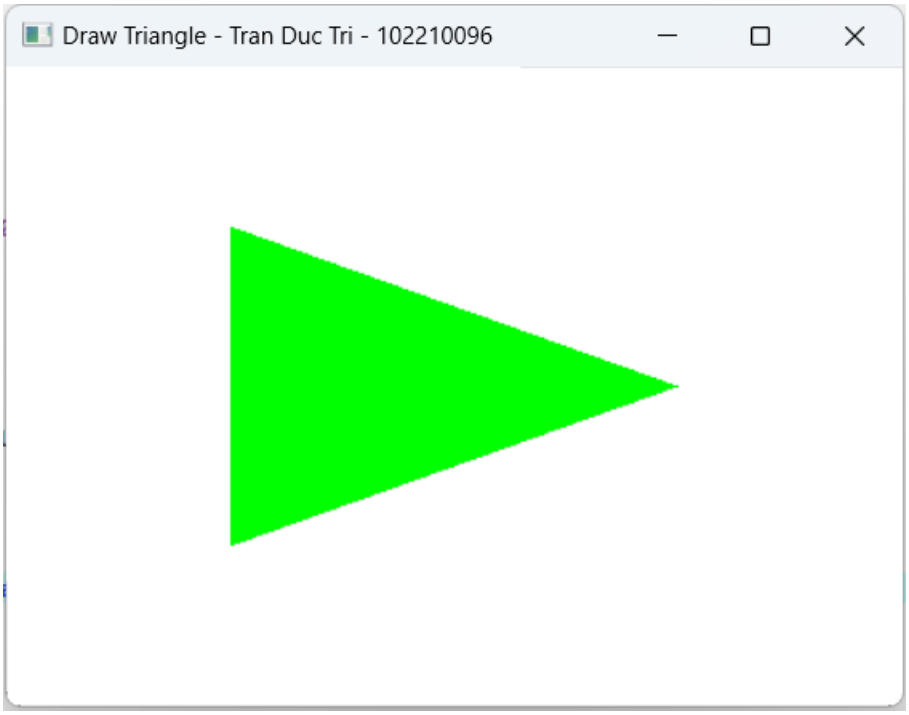
- void glutWireTorus(GLdouble innerRadius, GLdouble outerRadius, GLint nsides, GLint rings);
- void glutSolidTorus(GLdouble innerRadius, GLdouble outerRadius, GLint nsides, GLint rings);
- void glutWireIcosahedron(void);
- void glutSolidIcosahedron(void);
- void glutWireOctahedron(void);
- void glutSolidOctahedron(void);
- void glutWireTetrahedron(void);
- void glutSolidTetrahedron(void);
- void glutWireDodecahedron(GLdouble radius);
- void glutSolidDodecahedron(GLdouble radius);
- void glutWireCone(GLdouble radius, GLdouble height, GLint slices, GLint stacks);
- void glutSolidCone(GLdouble radius, GLdouble height, GLint slices, GLint stacks);
- void glutWireTeapot(GLdouble size);
- void glutSolidTeapot(GLdouble size);



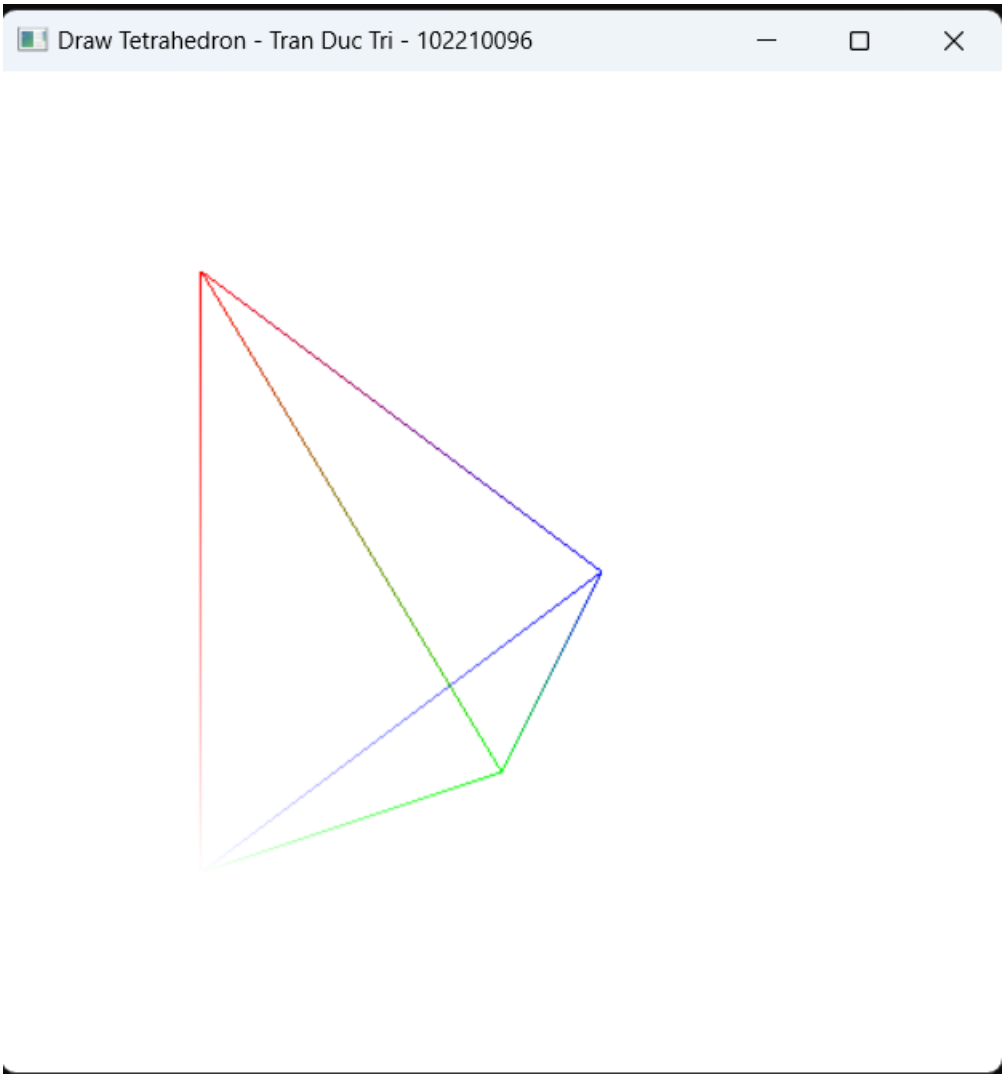
3. Chương trình lab01hcn.cpp: Vẽ một hình chữ nhật màu trắng trên nền đen



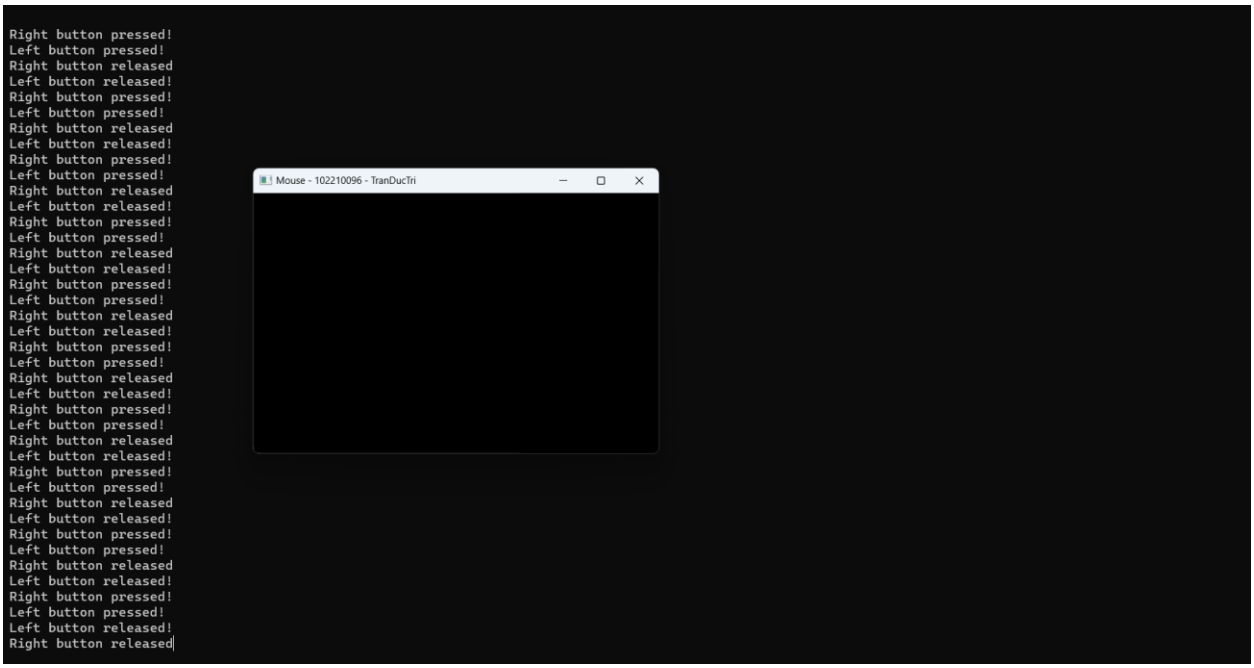
4. Chương trình lab01tamgiac.cpp: Vẽ một tam giác



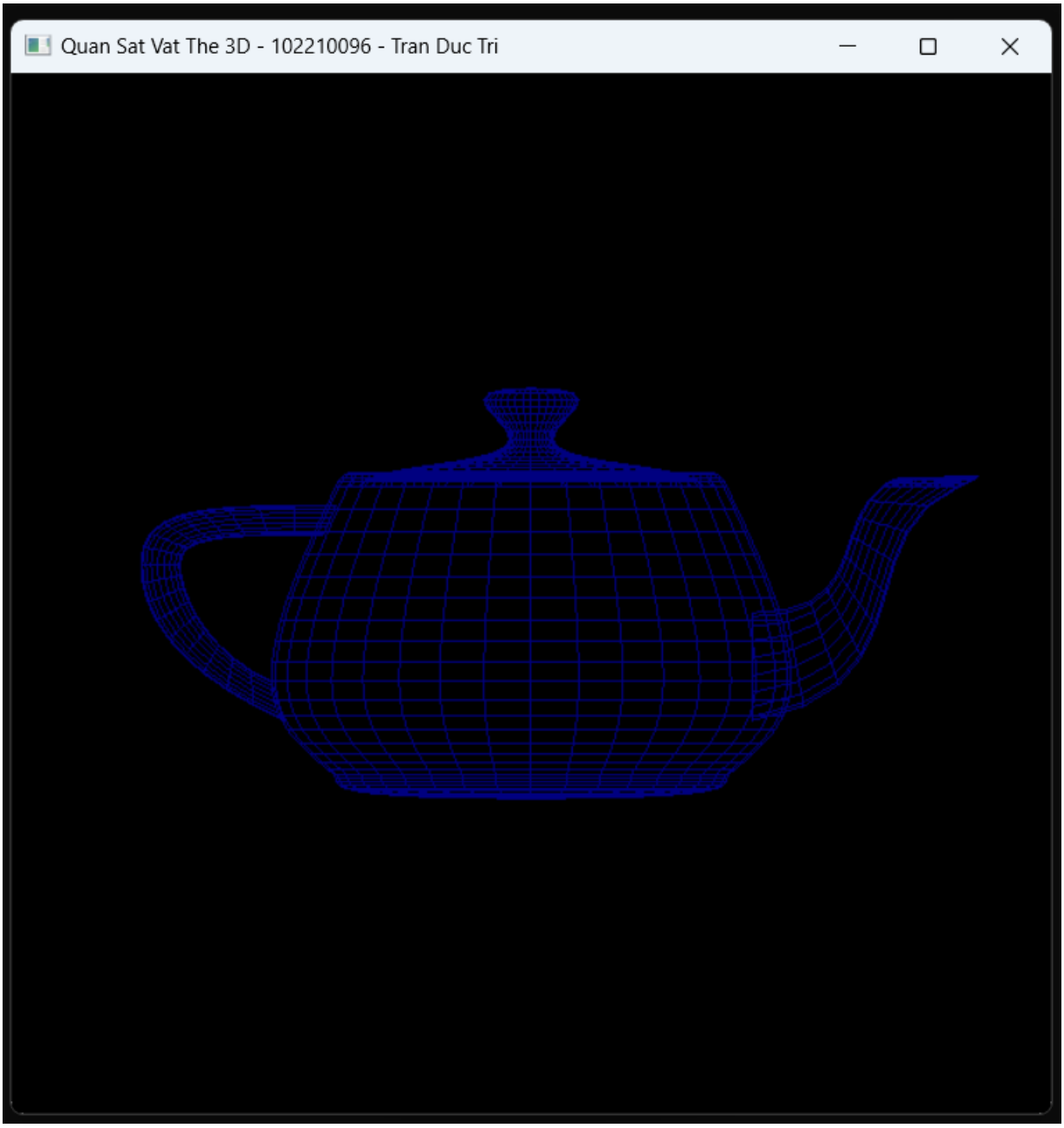
5. Chương trình lab01tudien.cpp: Vẽ một khối tứ diện



6. Chương trình lab01mouse.cpp: Xử lý các sự kiện phím và chuột trong OpenGL

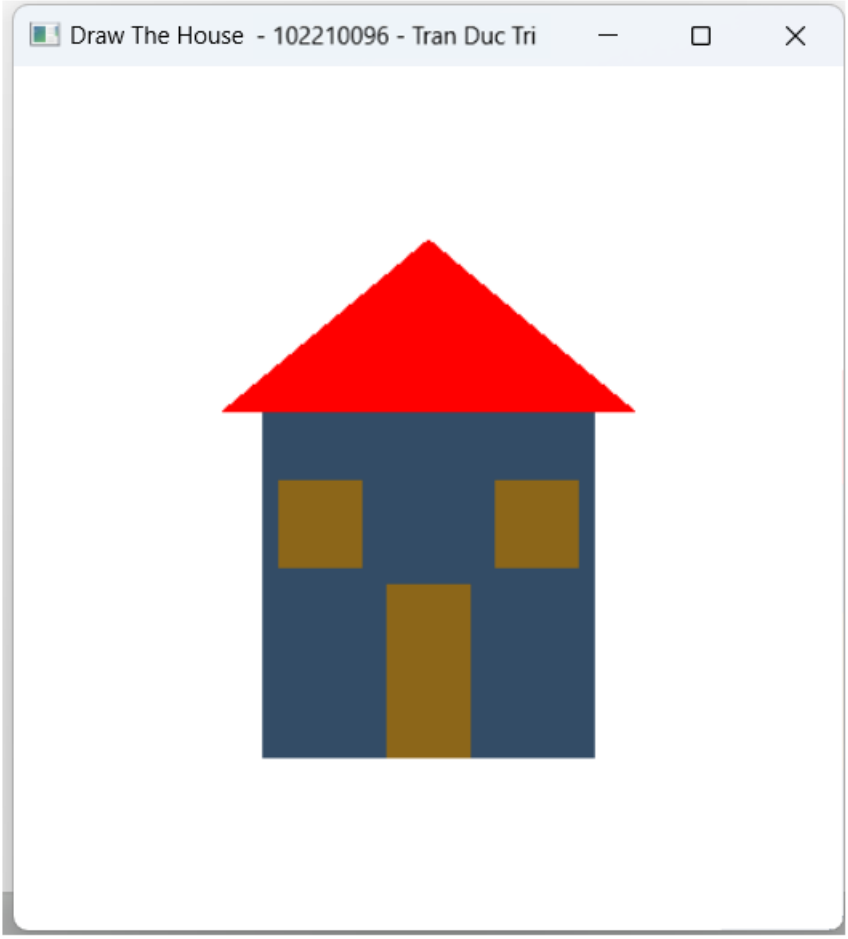


7. Chương trình lab01hinhcau.cpp: Vẽ mặt cầu/bình trà

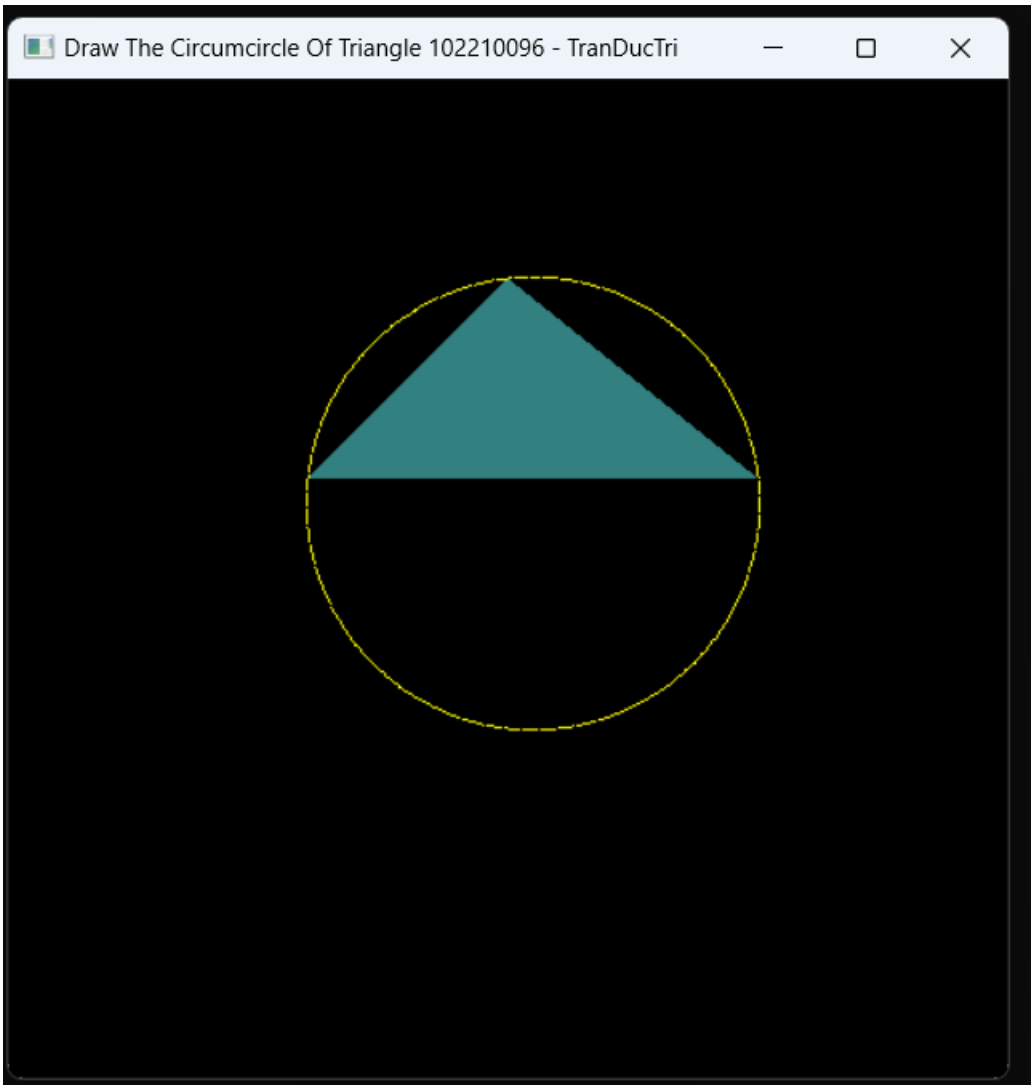


8. Bài tập

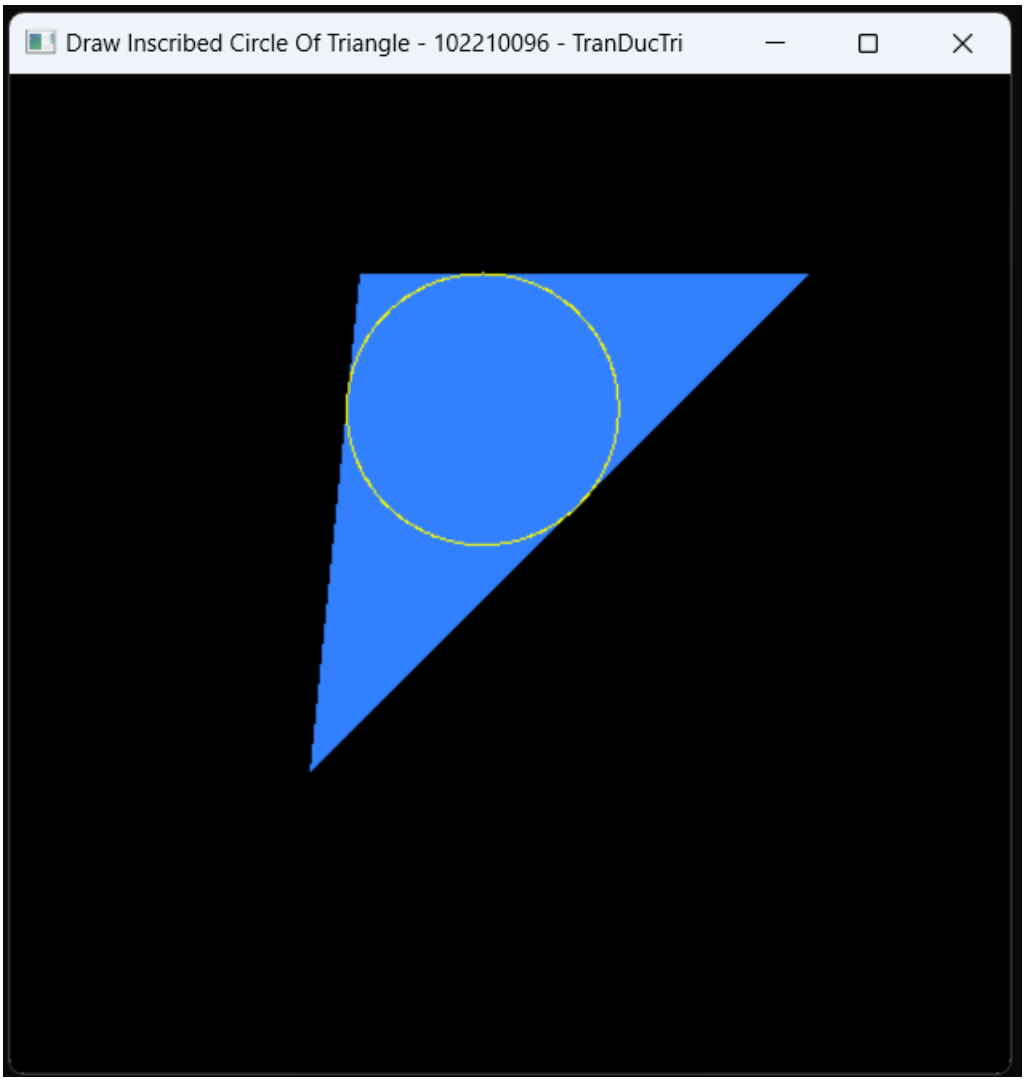
1. Lập trình OpenGL vẽ một đối tượng đơn giản (ngôi nhà, hình hộp chữ nhật, …)



2. Lập trình OpenGL 2D vẽ một đường tròn ngoại tiếp tam giác ABC có các điểm $A(x_a, y_a)$, $B(x_b, y_b)$, $C(x_c, y_c)$ cho trước không thẳng hàng.

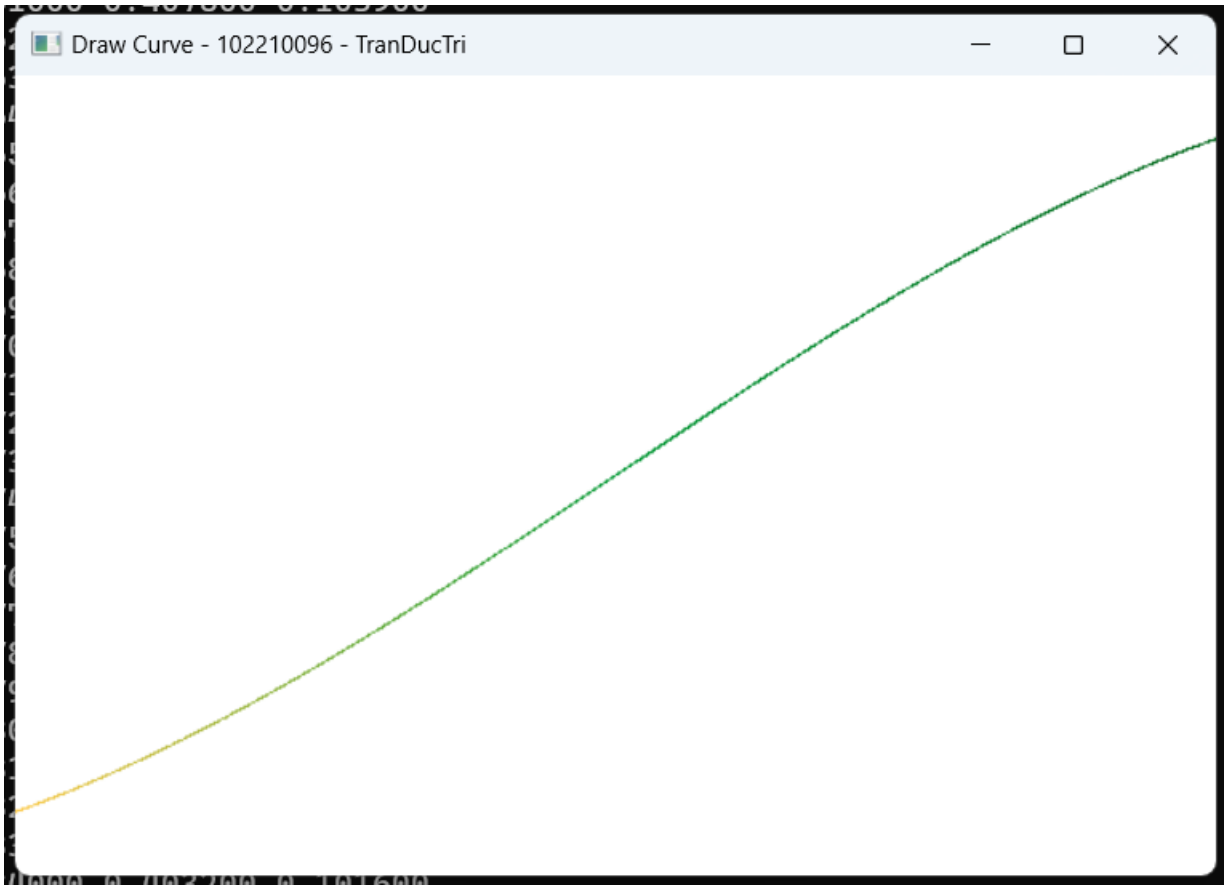


3. Lập trình OpenGL 2D vẽ một đường tròn 2D nội tiếp tam giác ABC có các điểm $A(x_a, y_a)$, $B(x_b, y_b)$, $C(x_c, y_c)$ cho trước không thẳng hàng.

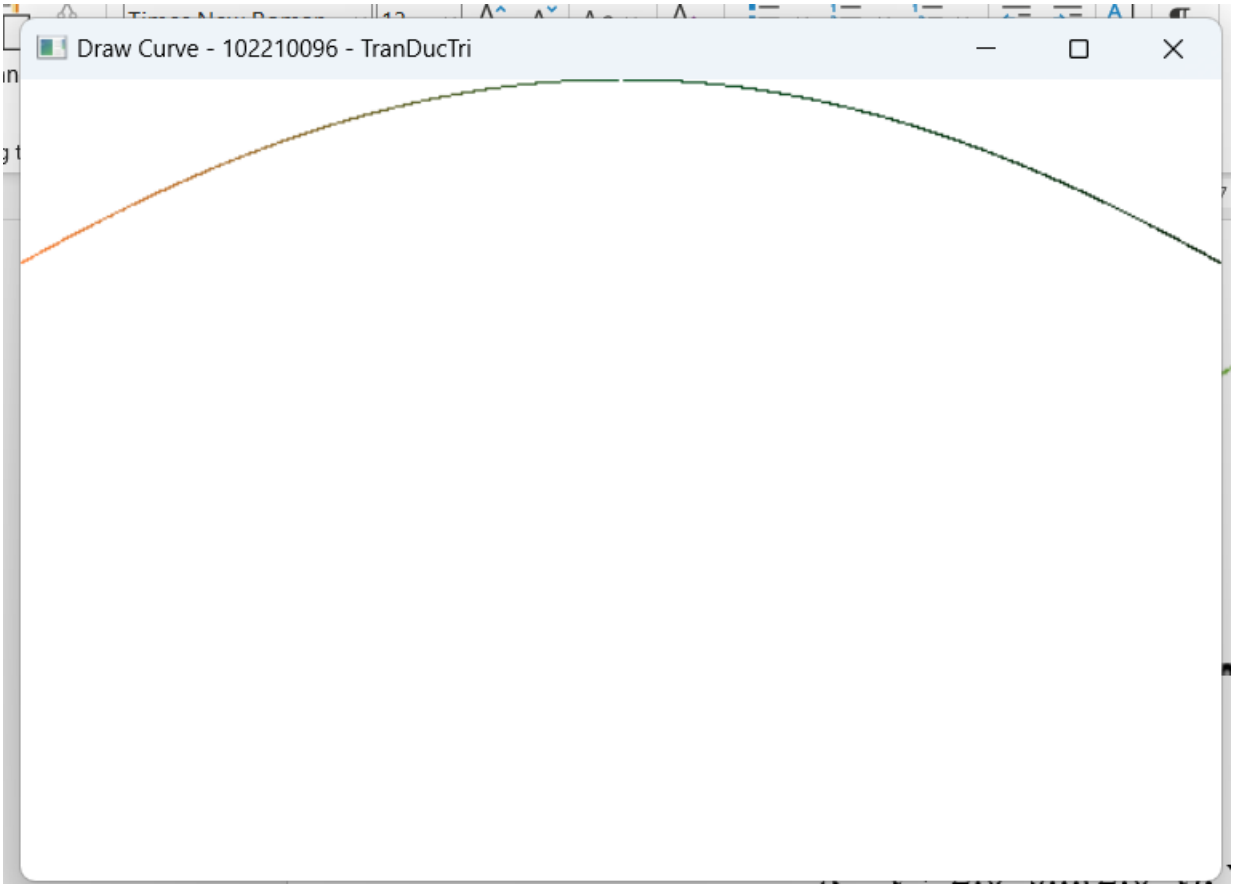


4. Lập trình OpenGL 2D vẽ các đường cong biểu diễn dưới dạng tham số như dưới đây. Lưu ý thiết lập cửa sổ, khung nhìn và khoảng biến thiên thích hợp cho tham số *t*, để được kết quả như hình vẽ 16.

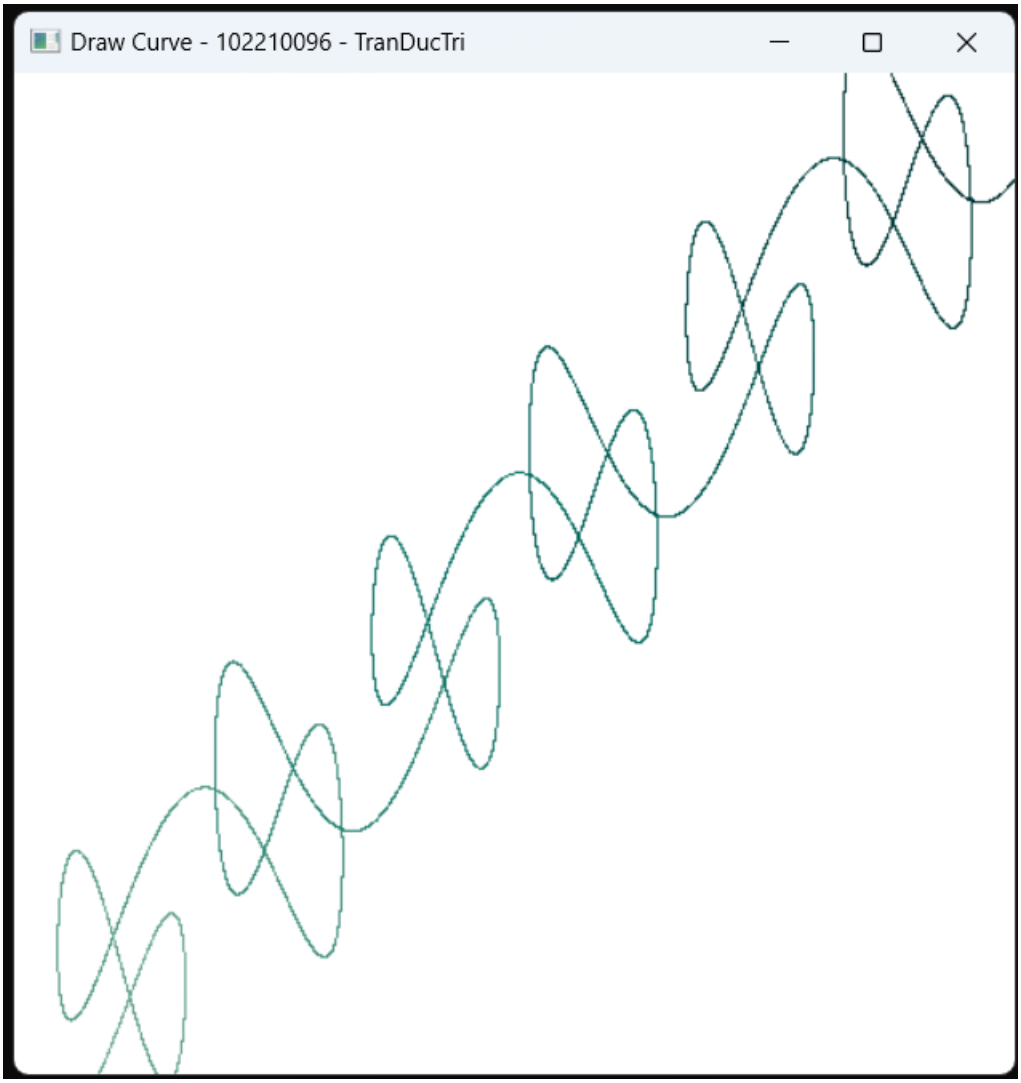
    - Đồ thị hàm y=sin(x)



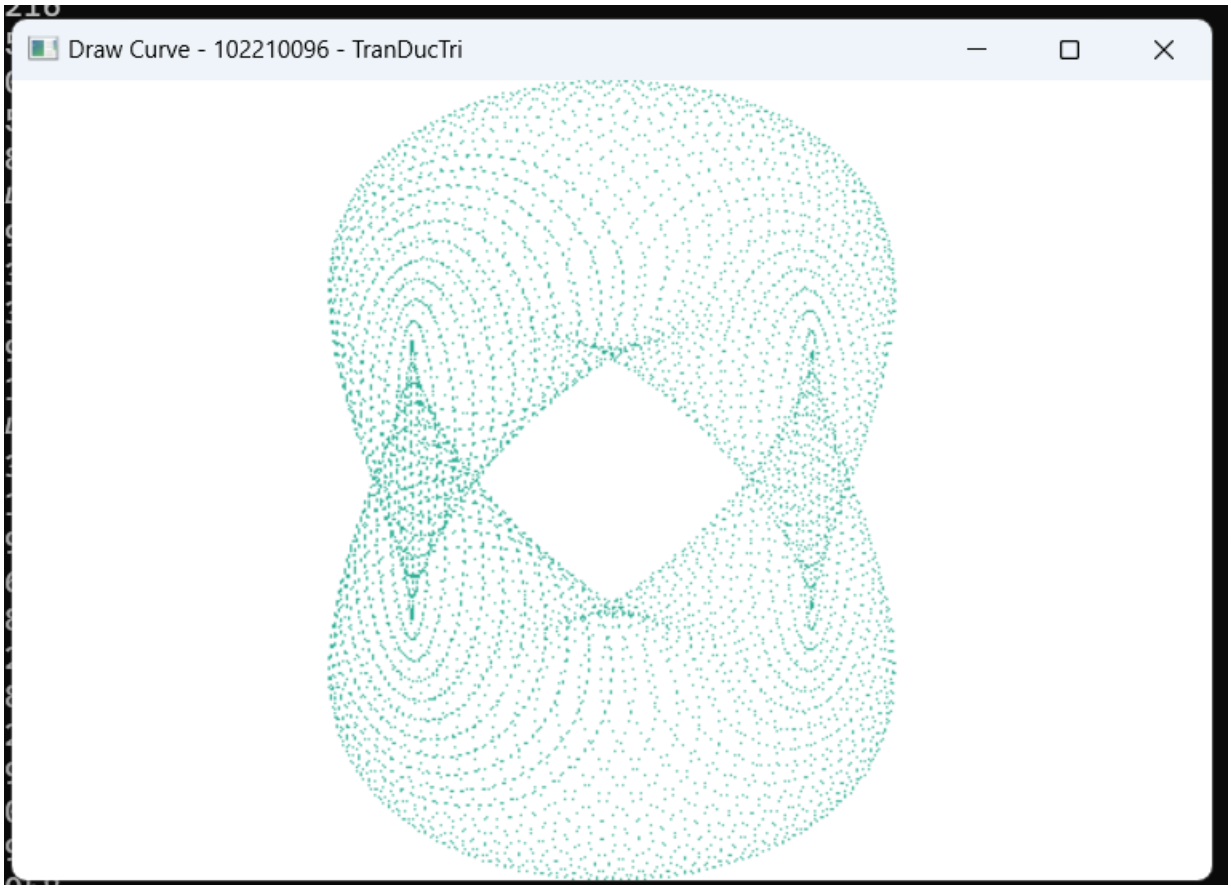
▪ Đồ thị hàm y=cos (x)



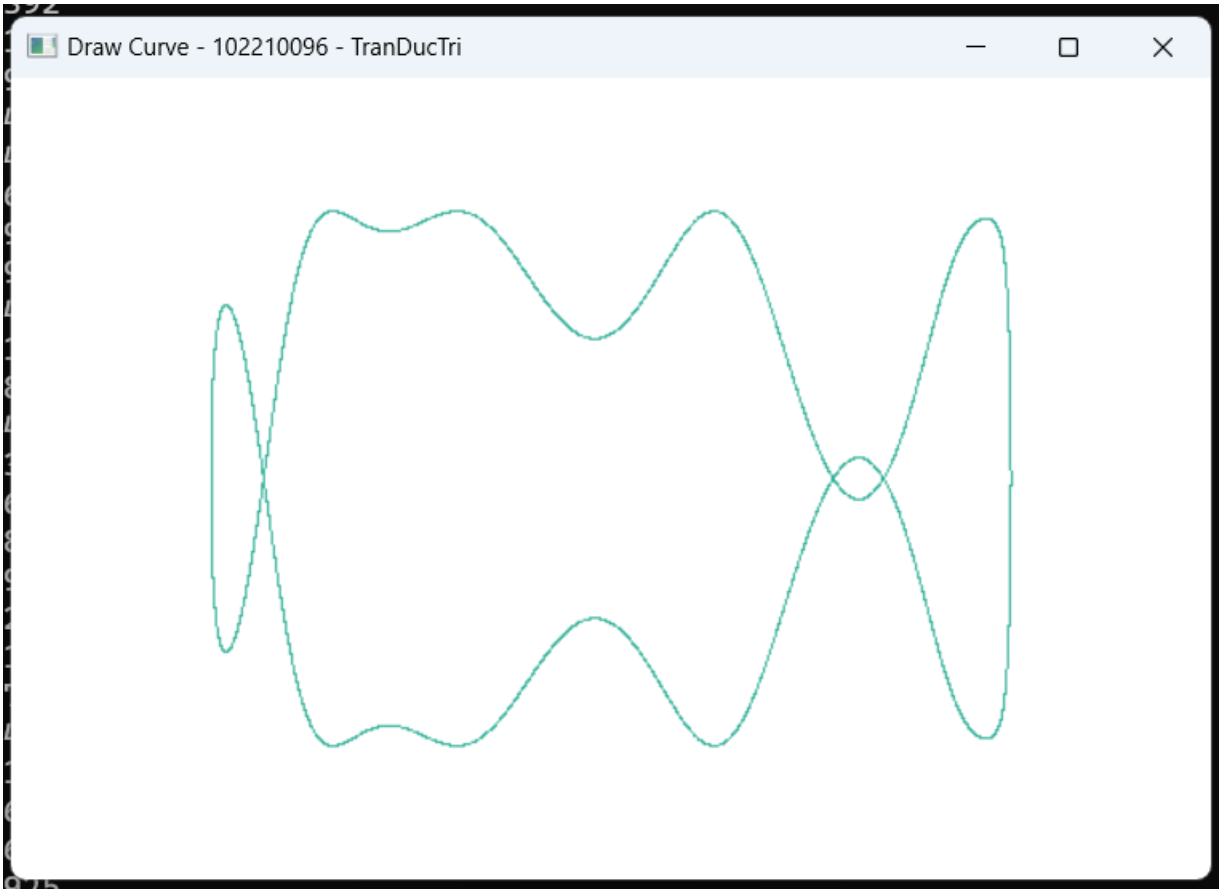
- x = t + 2.0*sin(2.0*t); y = t + 2.0*cos(5.0*t);



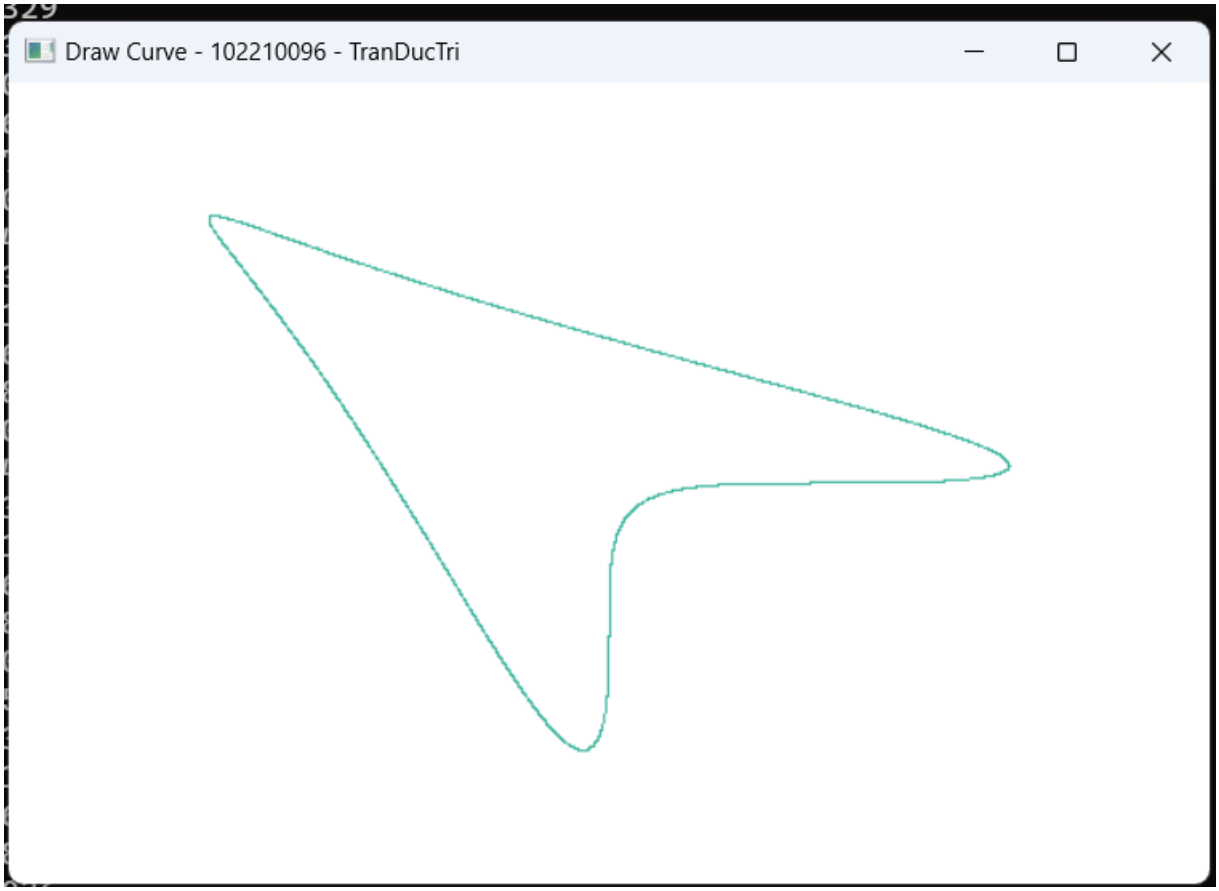
- x = cos(t) - cos(80*t)*sin(t); y = 2.0*sin(t) - sin(80*t);



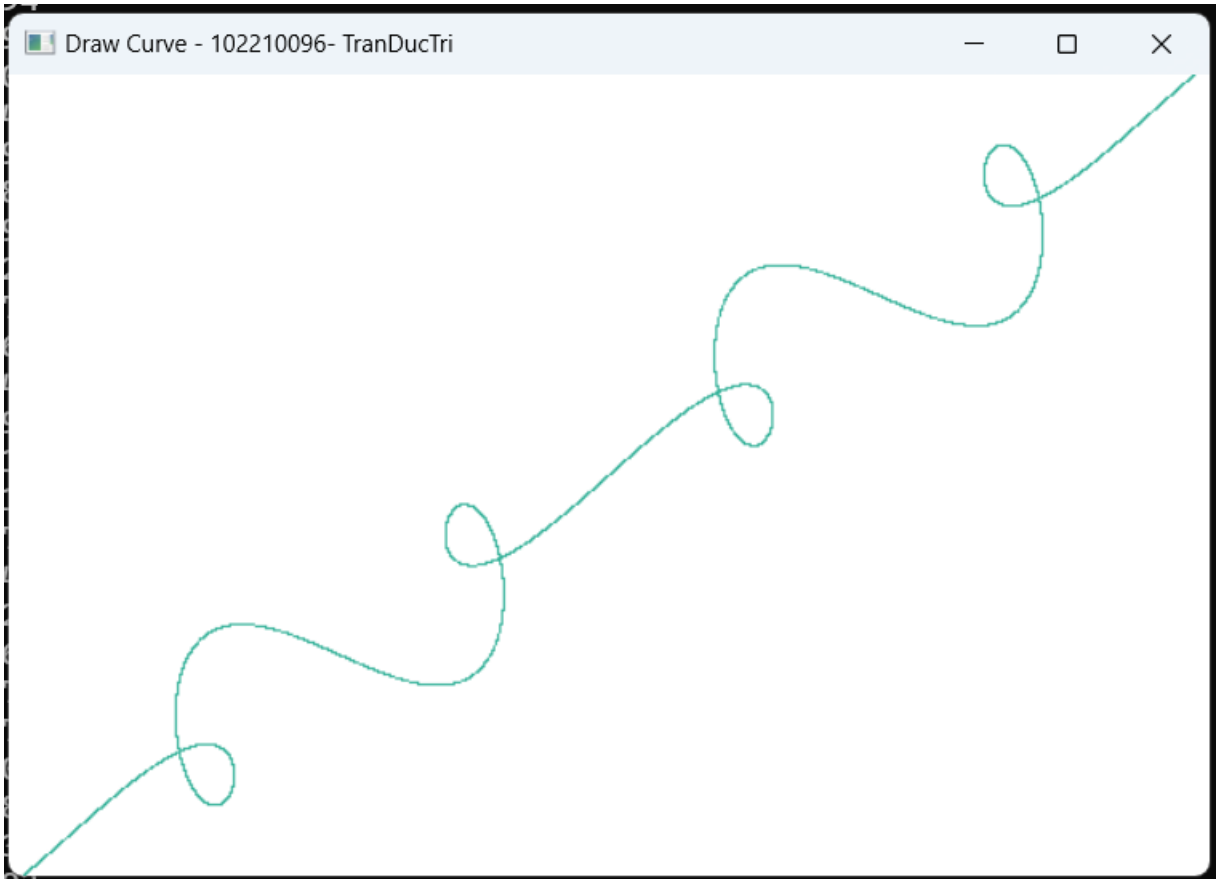
- x = cos(t); y = sin(t + sin(5.0*t));



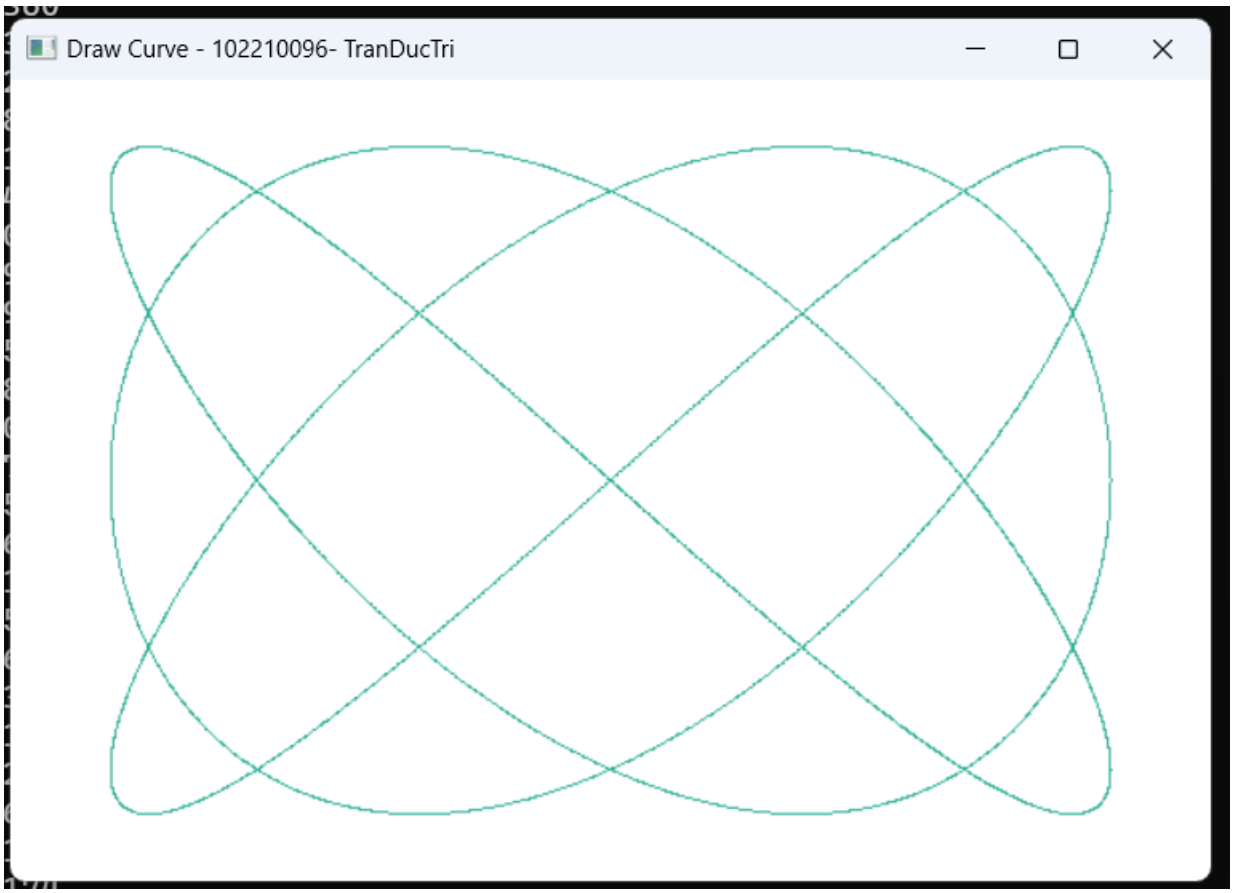
- x = sin(t+sin(t)); y = cos(t + cos(t));



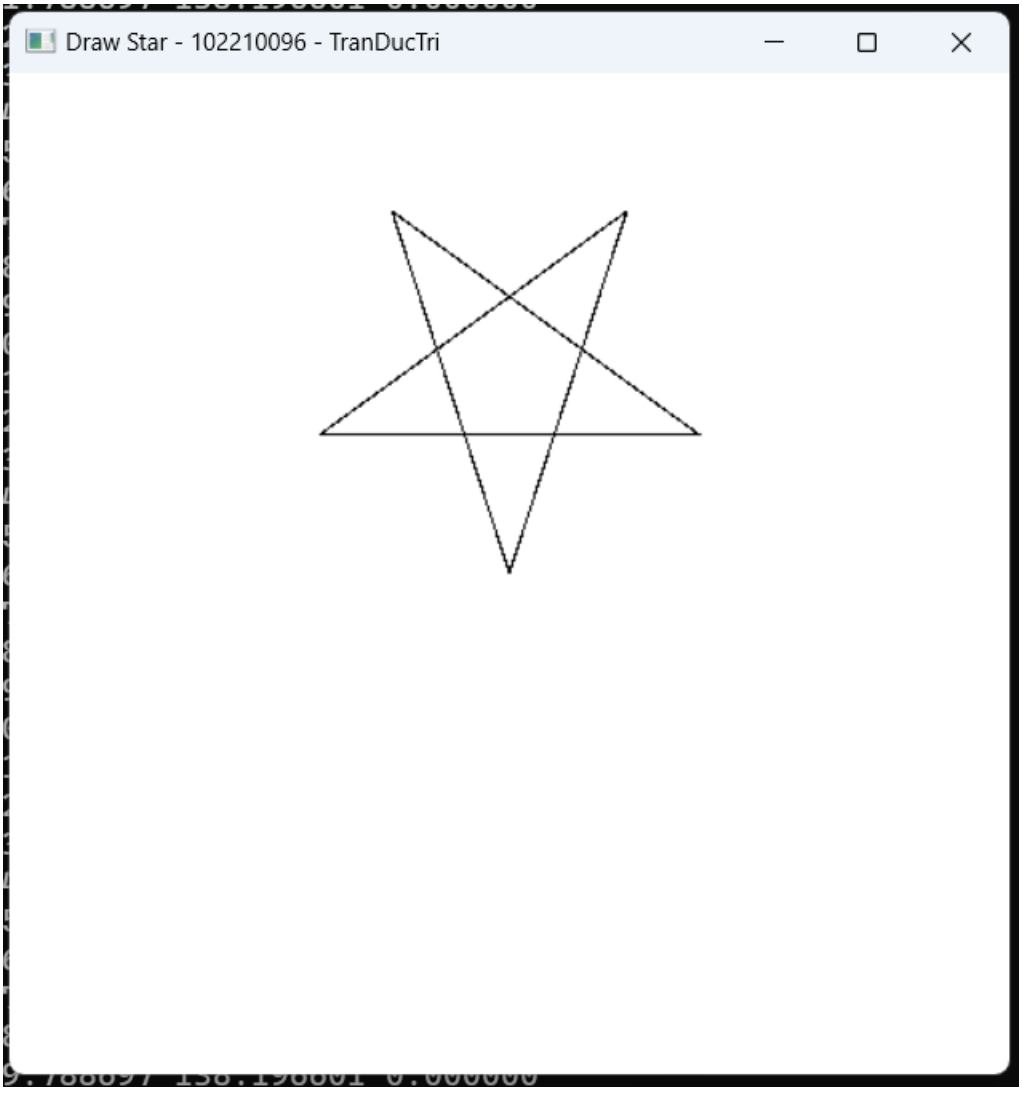
- x = t + sin(2.0*t); y = t + sin(3.0*t);



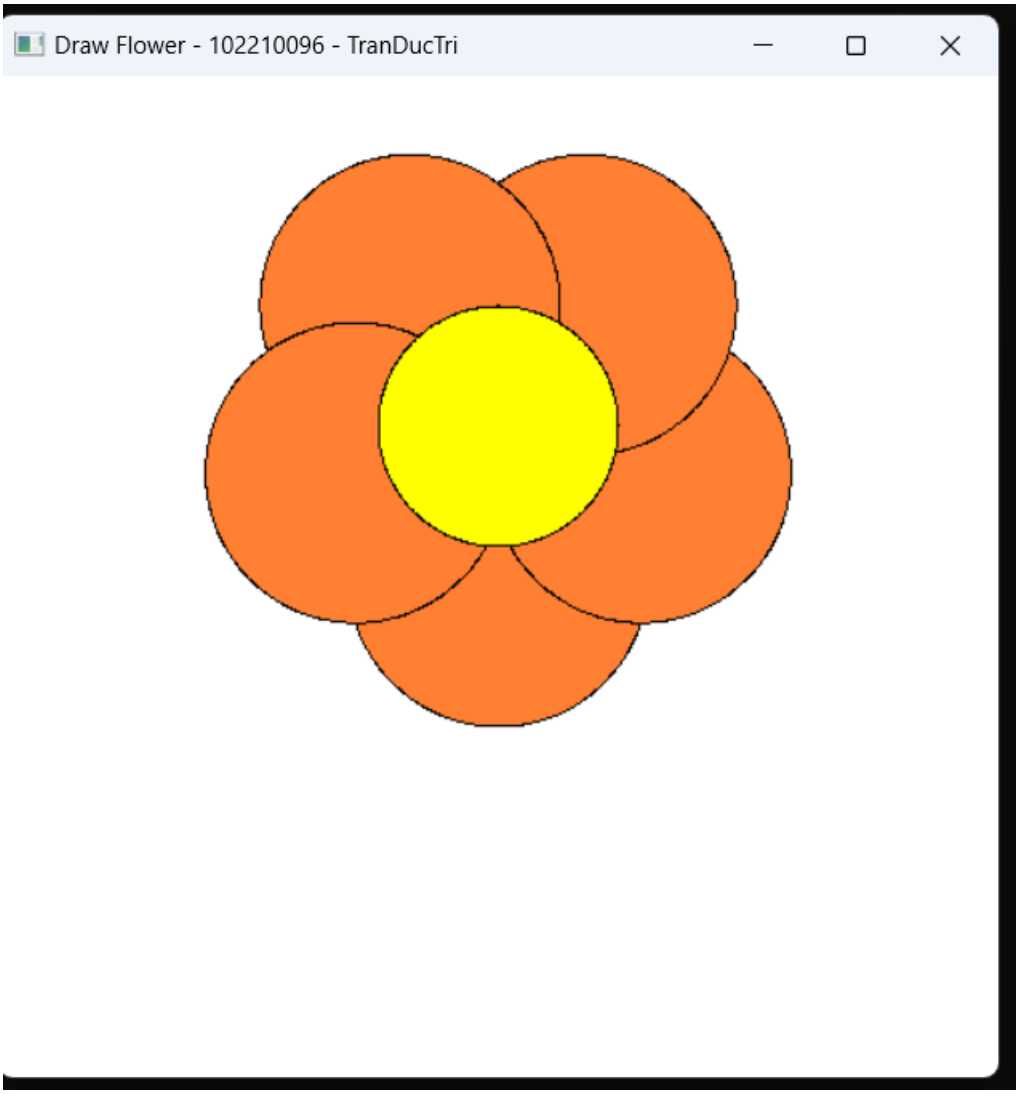
- x = sin(3.0*t); y = sin(4.0*t);



5. Lập trình OpenGL 2D vẽ ngôi sao 5 cánh

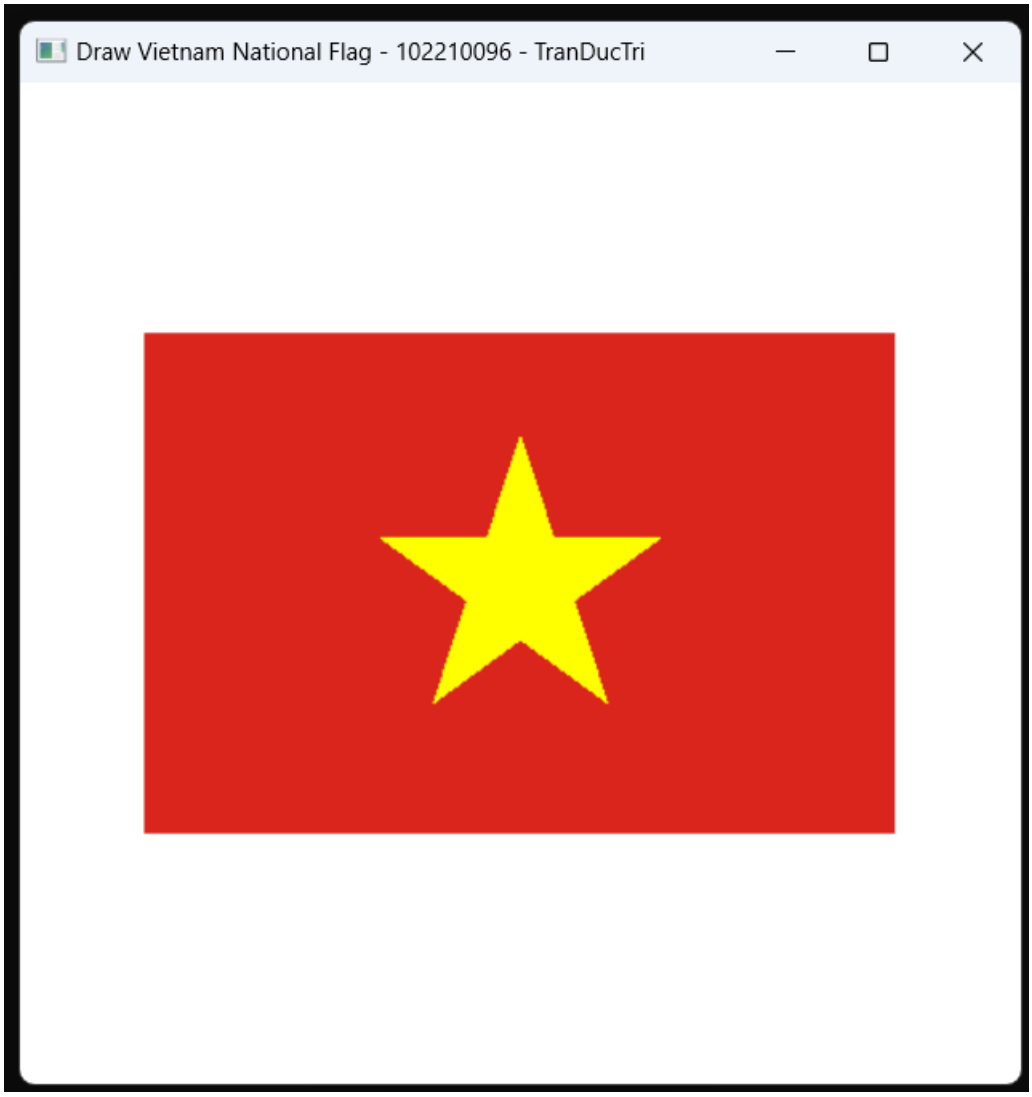


6. Lập trình OpenGL 2D vẽ bông hoa 5 cánh



7. Vẽ quốc kỳ Việt nam

https://giaphiep.com/blog/ve-quoc-ky-viet-nam-bang-flutter-28004


Draw Vietnam National Flag - 102210096 - TranDucTri
