

# Bài Thực Hành Số 1:

## Làm quen với lập trình đồ họa

(Lê Thành Sách)

### Nội dung:

- Cài đặt và cấu hình OpenGL, Visual Studio trên Windows.
- Biên dịch, thực thi, và tìm hiểu cấu trúc một chương trình đồ họa dùng OpenGL với OpenGL Utility Toolkit (GLUT) và Microsoft Foundation Class (MFC).
- Viết chương trình nhỏ vẽ các đối tượng hình học 2D (điểm, đường thẳng, đa giác, hình tròn, v.v) và 3D (hình hộp, đa diện, v.v) đơn giản.

*Để thực hiện tốt phần thực hành cho môn Computer Graphics, sinh viên cần có trong tay tài liệu, thư viện lập trình đồ họa, và các chương trình mẫu cho mỗi buổi thực hành.*

### Tài liệu:

1. Tài liệu hướng dẫn này.
2. OpenGL reference book (blue book).
3. GLUT reference book.

### Thư viện lập trình đồ họa:

1. OpenGL core library(GL):
  - a. Header file: GL/gl.h
  - b. Static link library: Opengl32.lib
  - c. Dynamic link library: Opengl32.dll
2. OpenGL utility library (GLU):
  - a. Header file: GL/glu.h
  - b. Static link library: glu32.lib
  - c. Dynamic link library: glu32.dll
3. OpenGL utility toolkit(GLUT):
  - a. Header file: GL/glut.h
  - b. Static link library: glut32.lib
  - c. Dynamic link library: glut32.dll
- 4.

### Chương trình mẫu:

#### For GLUT:

1. Dự án mẫu:
2. Dự án 1: vẽ hình vuông tĩnh
3. Dự án 2: vẽ hình vuông động
4. Dự án 3: vẽ tứ diện tĩnh
5. Dự án 4: vẽ tứ diện động

#### For MFC:

1. Dự án mẫu:
2. Dự án 1: vẽ hình vuông tĩnh
3. Dự án 2: vẽ hình vuông động
4. Dự án 3: vẽ tứ diện tĩnh
5. Dự án 4: vẽ tứ diện động

### Mục lục:

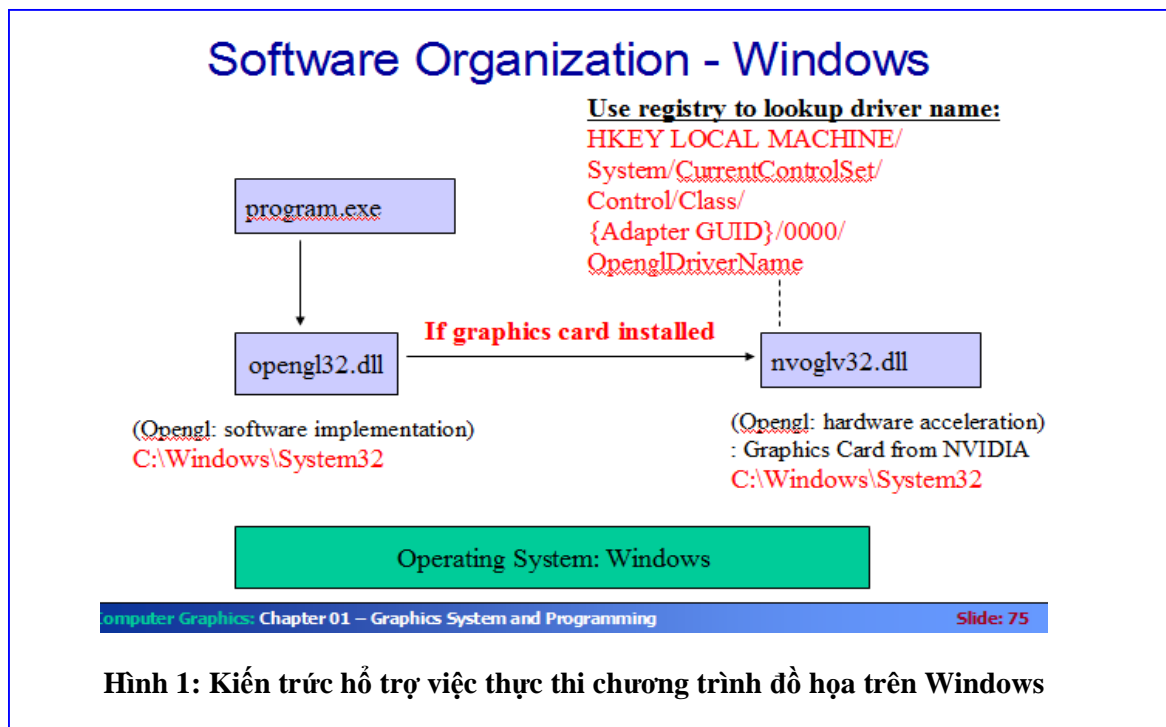
I. Kiến trúc OpenGL trên Windows: .....	3
II. Cấu hình môi trường phát triển tích hợp (IDE): .....	4
II.1 Cấu hình cho OpenGL.....	4

II.2 Cấu hình cho GLUT .....	5
II.3 Dự án mẫu dùng GLUT:.....	5
II.4 Dự án mẫu dùng MFC: .....	6
<b>III. Tìm hiểu cấu trúc chương trình đồ họa: .....</b>	<b>7</b>
III.1 Dùng GLUT:.....	7
1. Dự án 1: Vẽ hình vuông tĩnh .....	7
2. Dự án 2: Vẽ hình vuông động .....	8
3. Dự án 3: Vẽ đa diện .....	10
4. Dự án 4: Vẽ đa diện, thay đổi vị trí nhìn.....	12
III.2 Dùng MFC:.....	14
1. Dự án 1: Vẽ hình vuông tĩnh .....	14
2. Dự án 2: Vẽ hình vuông động .....	16
3. Dự án 3: Vẽ tứ diện tĩnh.....	18
<b>IV. Viết chương đồ họa đơn giản:.....</b>	<b>20</b>

## I. Kiến trúc OpenGL trên Windows:

Sau khi cài đặt Windows xong, thư mục **C:\Windows\System32** chứa tập tin “**opengl32.dll**” nhằm giúp Windows hỗ trợ thực thi các ứng dụng đồ họa dùng OpenGL. Cụ thể, khi một ứng dụng X (**program.exe**) được cho thực thi trên hệ điều hành Windows, mỗi khi chương trình “**program.exe**” thực hiện một hàm trong OpenGL thì đoạn mã tương ứng của hàm đó trong “**opengl32.dll**” sẽ được thực hiện (thông qua liên kết động).

Tuy nhiên, nếu như hệ thống phần cứng có thêm card đồ họa thì có thể đoạn mã trong một tập tin thư viện khác được thực thi. Lý do, “**opengl32.dll**” chỉ là phần hiện thực bằng phần mềm của Microsoft cho luồng đồ họa (graphic pipeline). Mỗi khi card đồ họa được cài đặt thì đoạn mã được tăng tốc bởi card nên được thay thế cho đoạn mã tương ứng trong “**opengl32.dll**”. Việc phát hiện và chuyển lời gọi hàm này trong Windows cho OpenGL là một sáng kiến của Microsoft, gọi là “**Installable Client Driver**”. Thông tin về card đồ họa nào được cài đặt và thư viện tăng tốc nào chứa đoạn mã cần thực thi thay thế cho “**opengl32.dll**” nằm trong *Registry* của hệ thống. Hình 1 sau đây minh họa việc thực thi chương trình đồ họa dùng Opengl và sự hỗ trợ của card *GeForce 8400M GT* của *Nvidia*.

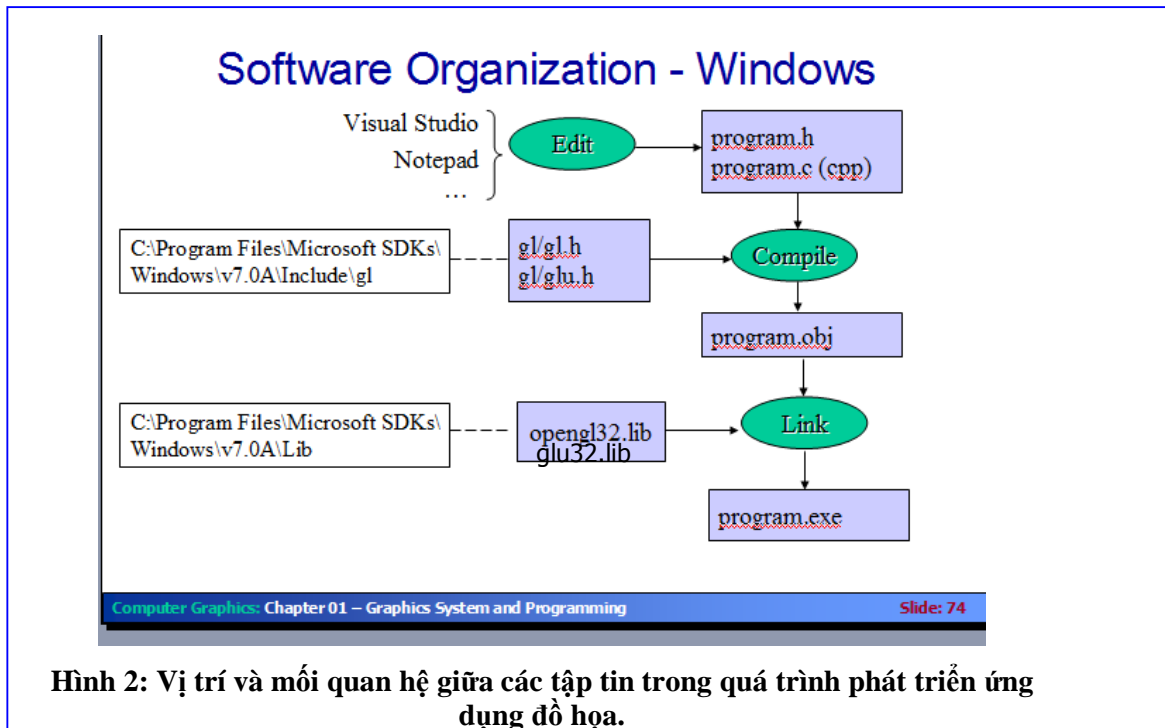


Đứng về phía người lập trình, để tạo ra chương trình đồ họa nói trên – **program.exe**, họ phải trải qua soạn thảo (edit), biên dịch (compile), liên kết (link), sau đó là chạy thử và sửa lỗi. Nếu dùng ngôn ngữ C/C++ thì phải cần đến các “*header file*” và thư viện liên kết “*library*”. Cụ thể là các tập tin như sau:

Header files: “**gl.h**”, “**glu.h**”

Library: “**opengl32.lib**”, “**glu32.lib**”

Nếu cài đặt môi trường phát triển tích hợp **Visual Studio (VS)** thì các tập tin nói trên nằm trong các thư mục chỉ ra trong hình 2 sau đây. Các tập tin trên cũng có thể được download miễn phí từ Internet.



Khi lập trình (dùng C/C++), người lập trình phải thực hiện các bước sau:

- Include tập tin “**gl.h**” và “**glu.h**” phía trước bất kỳ tham chiếu nào đến các hàm, hằng, etc của OpenGL, ngược lại sẽ có lỗi biên dịch.
- Chỉ rõ vị trí của các *header file* và *library*, ngược lại sẽ có lỗi biên dịch.
- Chỉ rõ khi liên kết thì dùng “**opengl32.lib**” và “**glu32.lib**”, ngược lại sẽ có lỗi liên kết.

## II. Cấu hình môi trường phát triển tích hợp (IDE):

### II.1 Cấu hình cho OpenGL

1. Chỉ rõ vị trí của các *header file* và *library*.

Giả sử đã tạo một project “**program**” trên Visual Studio.

- Mở hộp thoại thuộc tính cho dự án:** Nhấn nút chuột phải trên tên project ở cửa sổ navigator trong IDE, rồi chọn menu “*Properties*”, hay truy cập menu “**Project\Properties**” để mở hộp thoại thuộc tính cho dự án.
- Trên hệ thống cây thuộc tính trên hộp thoại, tìm đến folder “**VC++ Directories**”.
  - Cấu hình vị trí include file: thêm tên thư mục chứa tập tin “GL\gl.h” và “GL\glu.h” vào danh sách thư mục trong biến: “**Include Directories**” hiện trên hộp thoại. Chú ý là chỉ đến thư mục có chứa thư mục “**GL**” – không phải thư mục chứa *header files*.
  - Cấu hình vị trí của tập tin thư viện “**opengl32.lib**” và “**glu32.lib**”: Tương tự, thay đổi biến “**Library Directories**” hiện trên hộp thoại.

**Lưu ý:** Nếu là Visual Studio 2008 trở đi, thì các bước trên không cần phải thực hiện vì mặc nhiên visual studio đã nhìn thấy các vị trí nói trên.

2. Chỉ rõ khi liên kết thì dùng “**opengl32.lib**” và “**glu32.lib**”, ngược lại sẽ có lỗi liên kết. Đây là bước bắt buộc, nếu không thực hiện sẽ bị lỗi liên kết.

- Mở hộp thoại thuộc tính cho dự án**

- b. Trên hệ thống cây thuộc tính, tìm đến “Linker\Input\Additional Dependencies”. Bổ sung “**opengl32.lib; glu32.lib**” vào danh mục các tập tin thư viện dùng đến của dự án.

## II.2 Cấu hình cho GLUT

Thư viện OpenGL chỉ quan tâm hiện thực Graphic Pipeline mà không có hiện thực giao diện người dùng cũng tương tác giữa người dùng với ứng dụng qua Mouse, Keyboard, etc. Những khoảng trống này được dành cho các thư viện khác như GLUT, MFC, etc.

GLUT là tiện ích nhỏ và chỉ thích hợp cho việc học, nó gồm:

- Header file: **glut.h**
- Static Library file: **glut32.lib**
- Dynamic Library file: **glut32.dll**

Để dùng GLUT cần các bước sau:

1. Chỉ rõ vị trí của **header file**
2. Chỉ rõ vị trí của **library file** (\*.lib)
3. Thay đổi biến môi trường **PATH** để lúc thực thi hệ thống nhìn thấy tập tin liên kết động “**glut32.dll**”. Nhớ khởi động lại máy nếu thay đổi **PATH**. Cũng có thể copy “glut32.dll” vào cùng thư mục chứa tập tin thực thi, hay copy nó vào **C:\Windows\System32\**.

## II.3 Dự án mẫu dùng GLUT:

1. Mở Visual Studio
2. Tạo dự án: Win32 Console
  - a. Dùng menu: File\New\Project
  - b. Đặt tên dự án: **GlutTemplate**
3. Thực hiện các bước cấu hình nói trên cho OpenGL, GLU và GLUT
4. Thêm tập tin “**template.cpp**”
5. Soạn thảo đoạn chương trình mẫu như hình 3
6. Compile và chạy chương trình

```
#include <GL/glut.h>
/*
glut.h:
- Please open "glut.h" and find "gl.h"
- You will see that "glut.h" include "gl.h" and "glu.h" inside
- So, you do not include "gl.h" and "glu.h" manually whenever and wherever
you use GLUT
*/

#include <stdio.h>

int main(int argc, char** argv){
    printf("Add your own code to this program");
    printf("\n");
    getchar();
}
```

Hình 3: Chương trình mẫu với GLUT

## II.4 Dự án mẫu dùng MFC:

### Tạo dự án mẫu:

1. Mở Visual Studio
2. Tạo dự án: MFC Application
  - a. Dùng menu: File\New\Project
  - b. Đặt tên dự án: **MFCTemplate**
  - c. Các thông số:
    - Chọn: *Single Document Interface*
    - Chọn: *MFC Dynamic Link*
3. Thực hiện các bước cấu hình nói trên cho OpenGL và GLU
5. Compile và chạy chương trình

### Cấu trúc dự án mẫu:

Dự án mẫu MFCTemplate có chứa 4 lớp chính, chúng được sinh ra tự động bởi Visual Studio. Các lớp chính đó là:

#### 1) Lớp MFCTemplateApp:

- Khi ứng dụng đang được thi, một đối tượng của lớp này nằm trong bộ nhớ.
- Lớp này dùng để quản lý các hàm xử lý chung ở mức ứng dụng. Ví dụ, việc tạo ra các đối tượng cửa sổ chính, đối tượng quản lý tài liệu, đối tượng quản lý vùng nhìn (view), và quan hệ giữa những đối tượng này.

#### 2) Lớp CMainFrame:

- Khi ứng dụng đang được thi, một đối tượng của lớp này nằm trong bộ nhớ.
- Lớp này chứa các hàm quản lý khung cửa sổ chính của ứng dụng. Việc tạo ra menu, toolbar cũng làm nhiệm vụ của lớp này.

#### 3) Lớp MFCTemplateDoc:

- Khi ứng dụng đang được thi, một đối tượng của lớp này nằm trong bộ nhớ, vì MFCTemplate là ứng dụng dạng SDI – Single Document Interface. Trong trường hợp là MDI – Multiple Document Interface – thì có thể có nhiều hơn một đối tượng tài liệu.
- Lớp này chứa các hàm quản lý tài liệu của ứng dụng, như đọc/lưu từ tập tin.

#### 4) Lớp MFCTemplateView:

- Khi ứng dụng đang được thi, thì **ít nhất** một đối tượng của lớp này nằm trong bộ nhớ.
- Lớp này chứa các hàm quản lý vùng nhìn, nghĩa là vùng mà người dùng nhìn thấy. Cụ thể, với lập trình OpenGL thì hầu hết tác vụ vẽ đều nên tập trung ở lớp này. Khi lập trình OpenGL, người lập trình thường phải bổ sung một số hàm xử lý sự kiện vào lớp này. Các sự kiện phổ biến nhất là:

##### a) WM\_CREATE → OnCreate(param):

**Lưu ý:** Cách viết trên nghĩa là, khi dùng Visual Studio để bắt sự kiện WM\_CREATE thì một hàm OnCreate được bổ sung vào lớp MFCTemplateView.

- Hàm OnCreate chỉ được gọi một lần lúc tạo ra một cửa sổ làm vùng nhìn.
- Vì vậy, OnCreate là nơi thích hợp nhất để đặt những khởi động một lần duy nhất cho OpenGL – ví dụ, thiết lập định dạng Pixel (số bits, màu sắc, ...).

##### b) WM\_DESTROY → OnDestroy():

- Hàm OnDestroy chỉ được gọi một lần lúc hủy một cửa sổ vùng nhìn.
- Vì vậy, OnDestroy là nơi thích hợp nhất để gọi những hàm giải phóng tài nguyên trước khi đóng cửa sổ vùng nhìn.
- Với lập trình OpenGL: Việc xóa bỏ Rendering Context hay Timer nên đặt trong hàm này.

##### c) WM\_SIZE → OnSize(UINT nType, int cx, int cy):

- Trước khi cửa sổ vùng nhìn hiển thị, hàm `OnSize` được gọi một lần.
  - Sau đó, cứ mỗi khi người dùng thay đổi kích thước cửa sổ chính thì `OnSize` được gọi.
  - Với lập trình OpenGL: `OnSize` là nơi thích hợp nhất để thiết lập Viewport. Việc thiết lập phép chiếu (projection) hay thông số camera cũng có thể đặt ở đây.
- d) **WM\_TIMER → OnTimer(UINT\_PTR nIDEvent):**
- `OnTimer` chỉ được nếu như trước đó hàm `SetTimer` đã được gọi. Hàm `SetTimer` có thể được gọi ở `OnCreate` hay trong bất kỳ hàm xử lý sự kiện khác, như khi người dùng nhấn chuột hay phím nào đó thì gọi `SetTimer`.
  - Xem Code mẫu trong các dự án sau để biết cách gọi `SetTimer`.
  - Hàm `SetTimer` tạo ra một Timer mà nó sẽ được kích gọi sau mỗi T milliseconds. Như vậy, khi thời gian T đã trôi qua thì `OnTimer` sẽ được gọi.
  - Vì có thể tạo ra nhiều Timer cùng lúc, nên khi `OnTimer` được gọi, mã số của Timer cũng có thể đọc bởi người lập trình.
  - Sau khi dùng Timer, nên gọi hàm `KillTimer` để xóa bỏ Timer.
- e) **WM\_PAINT → OnPaint(UINT\_PTR nIDEvent):**
- `OnPaint` là **hàm quan trọng** với lập trình OpenGL. Vì tất cả các tác vụ vẽ của OpenGL đều được đặt trong hàm này. Về chức năng, hàm này tương tự hàm Callback được truyền vào `glutDisplayFunc` khi dùng GLUT.
  - `OnPaint` sẽ được gọi mỗi khi vùng nhìn cần vẽ lại, có thể là: i) khi vùng nhìn vừa mới hiển thị lần đầu, 2i) vùng nhìn được nhìn thấy trở lại sau khi bị che khuất, 3i) khi người lập trình yêu cầu vẽ lại bởi những hàm như **`InvalidateRect`**.

### III. Tìm hiểu cấu trúc chương trình đồ họa:

#### III.1 Dùng GLUT:

##### 1. Dự án 1: Vẽ hình vuông tĩnh

1. Trong dự án mẫu **GlutTemplate**, thay “**template.cpp**” bởi tập tin “**demo1.cpp**”
2. Soạn **demo1.cpp** như Hình 4
3. Biên dịch và thực thi

```
#include <GL/glut.h>

void mydisplay(){
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0f, 0.0f, 0.0f);
    glBegin(GL_POLYGON);
        glVertex2f(-0.5, -0.5);
        glVertex2f(-0.5, 0.5);
        glVertex2f(0.5, 0.5);
        glVertex2f(0.5, -0.5);
    glEnd();
    glFlush();
}

int main(int argc, char** argv){
    glutCreateWindow("simple");
    glutDisplayFunc(mydisplay);
    glutMainLoop();
}
```

Hình 4: Chương trình mẫu với GLUT

#### Cấu trúc chương trình:

- **main(,,):** điểm bắt đầu chương trình
- **mydisplay():** hàm vẽ đơn giản, được đăng ký với hàm *glutDisplayFunc* của GLUT. Tên hàm này có thể được thay đổi bất kỳ, nhưng phải theo quy tắc đặt tên một danh hiệu trong C.
- **glutCreateWindow:** của GLUT, cho phép tạo một cửa sổ với tiêu đề là thông số.
- **glutDisplayFunc:** dùng hàm này để đăng ký các hàm vẽ (dùng OpenGL) với GLUT.
- **glutMainLoop:** khởi động một vòng lặp xử lý sự kiện của GLUT.

Bên trong **mydisplay**:

1. Xóa color frame buffer: **glClear**
2. Xác lập màu vẽ hiện tại: **glColor3f**
3. Vẽ một polygon: **glBegin(GL\_POLYGON)**
4. Xuất ảnh ra màn hình: **glFlush**

Các cấu hình mặc nhiên (default):

- Viewer (camera): dùng phép chiếu trực giao, với volume là hình hộp vuông trùng tâm với Polygon nói trên, mỗi chiều có kích thước là 2 đơn vị (từ -1 đến 1).
- Chiếu sáng: không dùng
- Số color buffer: 1 (Single Buffer)

## 2. Dự án 2: Vẽ hình vuông động

1. Trong dự án mẫu GlutTemplate, thay “**template.cpp**” bởi tập tin “**demo2.cpp**”
2. Soạn **demo2.cpp** như Hình 5
3. Biên dịch và thực thi



```

#include <GL/glut.h>
#include <math.h>
#include <stdio.h>
GLfloat angle;
#define DEG2RAD (3.14159f/180.0f)

void mydisplay(){
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0f, 0.0f, 0.0f);

    GLfloat cx, cy;
    cx = 0.5*cos(DEG2RAD*angle); cy = 0.5*sin(DEG2RAD*angle);

    glBegin(GL_POLYGON);
        glVertex2f(cx - 0.5, cy - 0.5);
        glVertex2f(cx - 0.5, cy + 0.5);
        glVertex2f(cx + 0.5, cy + 0.5);
        glVertex2f(cx + 0.5, cy - 0.5);
    glEnd();
    glFlush();
    glutSwapBuffers();
}

void processTimer(int value){
    angle += (GLfloat)value;
    if(angle > 360) angle = angle - 360.0f;

    glutTimerFunc(100, processTimer, 10);
    glutPostRedisplay();
}

int main(int argc, char** argv){
    angle = 0.0f;
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutCreateWindow("simple");
    glutDisplayFunc(mydisplay);
    glutTimerFunc(100, processTimer, 10);
    glutMainLoop();
}

```

**Hình 5: Chương trình mẫu với GLUT**

#### Cấu trúc chương trình:

- **main(,,):** ...
- **mydisplay():** ...
- **glutCreateWindow:** ...
- **glutDisplayFunc:** ...
- **glutMainLoop:**....
- **glutInitDisplayMode:** Cài đặt chế độ hiển thị, cụ thể:
  - GLUT\_DOUBLE: Sử dụng hai buffer. Một gọi là front buffer, dùng để hiển thị ảnh ra màn hình, nhưng khi vẽ thì không vẽ trên buffer này. Buffer còn lại được gọi là Back Buffer, dùng để vẽ mà không hiển thị, khi vẽ xong thì hoán vị Back thành Front và ngược lại.
  - GLUT\_RGB: Ảnh màu, định dạng RGB
- **processTimer:** Được gọi bởi GLUT khi thời gian đăng ký với GLUT (bởi hàm *glutTimerFunc*) tới hạn.

- **glutTimerFunc:** Dùng hàm này để đăng ký hàm xử Timer với GLUT. Timer hay dùng để tạo animation. Hàm xử lý Timer trong GLUT có dạng:  
*void processFunc(int value)*, ở đó tên hàm có thể đặt khác, *value* là giá trị của tham số thứ 3 được truyền vào hàm glutTimerFunc.

Bên trong **mydisplay**:

1. Vẽ hình vuông mà tâm của nó được cập nhật theo thời gian, biến angle. Biến angle được thay đổi trong hàm processTimer.
2. Hoán vị hai buffer (font buffer và back buffer)

Bên trong **processTimer**:

1. Thay đổi biến angle (tầm vực global, để nó tác động đến các hàm khác nữa).
2. Cài đặt Timer cho lần đến.
3. Yêu cầu vẽ lại màn hình: **glutPostRedisplay**.

### 3. Dự án 3: Vẽ đa diện

1. Trong dự án mẫu GlutTemplate, thay “**template.cpp**” bởi tập tin “**demo3.cpp**”
2. Soạn **demo3.cpp** như sau:

---

```
#include <GL/glut.h>
#define min(a,b) ((a)<(b)?(a):(b))

float tetra_vertices[][3] = {
    {0.0, 0.0, 1.0},
    {0.0, 0.942809, -0.33333},
    {-0.816497, -0.471405, -0.33333},
    {0.816497, -0.471405, -0.33333}
};

void mydisplay(){
    //setup viewwer - camera parameter:
    //i.e., location (eye),
    //direction (from eye to the reference point), and orientation (up vector)
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt(
        1.5,    //eyeX
        1.5,    //eyeY
        1.5,    //eyeZ
        0.0,    //reference point X
        0.0,    //reference point Y
        0.0,    //reference point Z
        0.0,    //up vector X
        1.0,    //up vector Y
        0.0     //up vector Z
    );

    //clear screen
    glClear(GL_COLOR_BUFFER_BIT);

    //the order of the vertices in a triangle is important!
    glBegin(GL_TRIANGLES);
        //Face 1: defined by vertices: 0, 2, 1; colored: red
        glColor3f(1.0f, 0.0f, 0.0f);
        glVertex3fv(tetra_vertices[0]);
        glVertex3fv(tetra_vertices[2]);
        glVertex3fv(tetra_vertices[1]);

        //Face 2: defined by vertices: 0, 1, 3; colored: red
        glColor3f(0.0f, 1.0f, 0.0f);
```

```

        glVertex3fv(tetra_vertices[0]);
        glVertex3fv(tetra_vertices[1]);
        glVertex3fv(tetra_vertices[3]);

        //Face 3: defined by vertices: 0, 3, 2; colored: red
        glColor3f(0.0f, 0.0f, 1.0f);
        glVertex3fv(tetra_vertices[0]);
        glVertex3fv(tetra_vertices[3]);
        glVertex3fv(tetra_vertices[2]);

        //Face 4: defined by vertices: 3, 1, 2; colored: red
        glColor3f(1.0f, 0.0f, 0.0f);
        glVertex3fv(tetra_vertices[3]);
        glVertex3fv(tetra_vertices[1]);
        glVertex3fv(tetra_vertices[2]);

    glEnd();
    glFlush();
    glutSwapBuffers();
}
void reshape(int width, int height){
    //setup viewport
    int size = min(width, height);
    glViewport(0, 0, size, size);
}
void initOpenGL(){
    //setup projection type
    //glFrustrum: define viewing volume
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glFrustum(
        -1.0, //left
        1.0,  //right
        -1.0, //bottom
        1.0,  //top
        2.0,  //near
        10.0  //far
    );
    //Default MatrixMode is MODELVIEW
    glMatrixMode(GL_MODELVIEW);

    //setup background color, or clear color
    glClearColor(0.1f, 0.7f, 0.7f, 1.0f);
}

int main(int argc, char** argv){
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutCreateWindow("Drawing a Tetrahedron");
    initOpenGL();
    glutDisplayFunc(mydisplay);
    glutReshapeFunc(reshape);
    glutMainLoop();
}

```

---

### 3. Biên dịch và thực thi

#### **Cấu trúc chương trình:**

1. **main()**: điểm bắt đầu chương trình
2. **initOpenGL()**: xác lập các thông số của Opengl để dùng về sau.  
- Tên hàm này do programmer đặt.

- Là nơi thích hợp để xác định để:

- a) Xác định phép chiếu: bằng cách gọi `glFurstrum`, `glOrtho`, `gluOrtho2D`, `gluPerspective`
- b) Xác định màu mặc định cho foreground và background: bằng cách gọi `glColor3f` (cho foreground) và `glClearColor` (cho background).

2. **mydisplay()**: hàm vẽ chính, các tác vụ vẽ của OpenGL được đặt trong hàm này.

- Đặc tả Viewer, nhớ lại trong Computer Graphics cần đặc tả 3 đối tượng quan trọng: Object, Viewer (Camera), và Light Source. Bằng cách gọi hàm `gluLookAt()`, người lập trình có thể đặt tả: vị trí, phương nhìn (nối từ camera đến một điểm tham chiếu – reference point, và hướng lên của camera)

- Vẽ các mặt của tứ diện bởi các tam giác, dùng `GL_TRIANGLES`, tô màu các mặt là Red, Green, Blue, và Black. Chú ý, thứ tự các đỉnh khi đặt tả tam giác, vì nó tạo ra mặt âm và dương.

3. **reshape(int width, int height)**:

- Hàm này được đăng ký là làm callback với `glutReshapeFunc()`. Do đó, nó sẽ được gọi lần đầu khi xác lập kích thước cửa sổ. Sau đó, khi cửa sổ đã hiển thì hàm này cũng sẽ được gọi mỗi khi người dùng thay đổi kích thước cửa sổ, ví dụ dùng chuột kéo biên của cửa sổ.

- Hàm reshape là nơi thích hợp nhất để xác định viewport.

4. Biến toàn cục “**tetra\_vertices**”, định nghĩa tọa độ 4 đỉnh của hình tứ diện.

#### 4. Dự án 4: Vẽ đa diện, thay đổi vị trí nhìn

1. Trong dự án mẫu **GlutTemplate**, thay “**template.cpp**” bởi tập tin “**demo4.cpp**”

2. Soạn **demo4.cpp** như sau:

---

```
#include <GL/glut.h>
#include <math.h>
#define min(a,b) ((a)<(b)?(a):(b))
#define DEG2RAD (3.14159f/180.0f)

float tetra_vertices[][3] = {
    {0.0, 0.0, 1.0},
    {0.0, 0.942809, -0.333333},
    {-0.816497, -0.471405, -0.333333},
    {0.816497, -0.471405, -0.333333}
};

GLfloat angle;

void mydisplay(){
    //setup viewer - camera parameter:
    //i.e., location (eye),
    //direction (from eye to the reference point), and orientation (up vector)
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt(
        1.5*cos(DEG2RAD*angle),    //eyeX
        1.5*sin(DEG2RAD*angle),    //eyeY
        3.5,                       //eyeZ
        0.0,                       //reference point X
        0.0,                       //reference point Y
        0.0,                       //reference point Z
        0.0,                       //up vector X
        1.0,                       //up vector Y
        0.0,                       //up vector Z
    );
}
```

```

        //clear screen
        glClearColor(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

        //the order of the vertices in a triangle is important!
        glBegin(GL_TRIANGLES);
            //Face 1: defined by vertices: 0, 2, 1; colored: red
            glColor3f(1.0f, 0.0f, 0.0f);
            glVertex3fv(tetra_vertices[0]);
            glVertex3fv(tetra_vertices[2]);
            glVertex3fv(tetra_vertices[1]);

            //Face 2: defined by vertices: 0, 1, 3; colored: red
            glColor3f(0.0f, 1.0f, 0.0f);
            glVertex3fv(tetra_vertices[0]);
            glVertex3fv(tetra_vertices[1]);
            glVertex3fv(tetra_vertices[3]);

            //Face 3: defined by vertices: 0, 3, 2; colored: red
            glColor3f(0.0f, 0.0f, 1.0f);
            glVertex3fv(tetra_vertices[0]);
            glVertex3fv(tetra_vertices[3]);
            glVertex3fv(tetra_vertices[2]);

            //Face 4: defined by vertices: 3, 1, 2; colored: red
            glColor3f(1.0f, 0.0f, 0.0f);
            glVertex3fv(tetra_vertices[3]);
            glVertex3fv(tetra_vertices[1]);
            glVertex3fv(tetra_vertices[2]);

        glEnd();
        glFlush();
        glutSwapBuffers();
    }

    void reshape(int width, int height){
        //setup viewport
        int size = min(width, height);
        glViewport(0, 0, size, size);
    }

    void processTimer(int value){
        angle += (GLfloat)value;
        if(angle > 360.0f) angle -= 360.0f;
        glutTimerFunc(100, processTimer, value);
        glutPostRedisplay();
    }

    void initOpenGL (){
        //setup projection type
        //glFrustrum: define viewing volume
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        glFrustum(
            -2.0, //left
            2.0,  //right
            -2.0, //bottom
            2.0,  //top
            2.0,  //near
            10.0  //far
        );
        //Default MatrixMode is MODELVIEW
        glMatrixMode(GL_MODELVIEW);
    }

```

```

        glEnable(GL_DEPTH_TEST);
    }

    int main(int argc, char** argv){
        angle = 0.0f;
        glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
        glutCreateWindow("Drawing a Tetrahedron");
        initOpenGL();
        glutDisplayFunc(mydisplay);
        glutReshapeFunc(reshape);
        glutTimerFunc(100, processTimer, 10);
        glutMainLoop();
    }

```

---

3. Biên dịch và thực thi.

#### **Cấu trúc chương trình:**

Như dự án 3, chỉ thay đổi:

- a. Tham số hóa vị trí camera theo biến angle.
- b. Sử dụng DEPTH\_BUFFER
 

```
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
glEnable(GL_DEPTH_TEST);
```
- c. Trước khi vẽ xóa cả color buffer và depth buffer:
 

```
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```


### **III.2 Dừng MFC:**

#### **1. Dự án 1: Vẽ hình vuông tĩnh**

Nội dung công việc:

- Bổ sung Header files cho OpenGL.
- Cài đặt OpenGL.
- Thiết lập viewport.
- Thực hiện vẽ một hình đa giác đơn giản.
- Giải phóng tài nguyên.
- Biên dịch và thực thi.

Các bước thực hiện:

 Sử dụng lại dự án MFCTemplate, i.e., copy sang một folder khác rồi thay đổi mã như sau

- Bổ sung Header files cho OpenGL:
  1. Bổ sung hai dòng lệnh sau vào MFCTemplateView.cpp
 

```
#include <GL/gl.h>
#include <GL/glu.h>
```
- Cài đặt OpenGL:
  2. Bổ sung hàm **SetupOpenGL**.
  3. Bổ sung biến thành viên m\_hRC kiểu HGLRC vào lớp MFCTemplateView
  4. Soạn hàm SetupOpenGL như sau:

---

```

//Declare Pixel Format
static PIXELFORMATDESCRIPTOR pfd =
{
    sizeof(PIXELFORMATDESCRIPTOR),
    1,
    PFD_DRAW_TO_WINDOW | PFD_SUPPORT_OPENGL | PFD_DOUBLEBUFFER,
    PFD_TYPE_RGBA,

```

```

        32, // bit depth
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        16, // z-buffer depth
        0, 0, 0, 0, 0, 0, 0,
    };

    // Get device context only once.
    HDC hdc = GetDC()->m_hDC;

    // Set Pixel format.
    int nPixelFormat = ChoosePixelFormat(hdc, &pfd);
    SetPixelFormat(hdc, nPixelFormat, &pfd);

    // Create the OpenGL Rendering Context.
    m_hRC = wglCreateContext(hdc);
    wglMakeCurrent(hdc, m_hRC);

```

---

5. Bổ sung hàm xử lý cho sự kiện WM\_CREATE, Visual Studio sinh ra OnCreate(param).
6. Soạn hàm OnCreate để gọi SetupOpenGL như sau:

```

if (CView::OnCreate(lpCreateStruct) == -1)
    return -1;

// Add only the following code.
// The previous code is generated automatically by IDE
SetupOpenGL();
return 0;

```

---

- Thiết lập viewport:
  7. Bổ sung hàm xử lý cho sự kiện WM\_SIZE, Visual Studio sinh ra OnSize(.,.).
  8. Soạn hàm OnSize như sau:

```

CView::OnSize(nType, cx, cy);

// Add only the following codes.
// The previous codes is generated automatically by IDE

//Setting up viewport
CRect rect;
GetClientRect(rect);
int size = min(rect.Width(), rect.Height());
glViewport(0, 0, size, size);

```

---

- Thực hiện vẽ một hình đa giác đơn giản:
  9. Bổ sung hàm xử lý cho thông điệp WM\_PAINT, Visual Studio sinh ra OnPaint()
  10. Soạn hàm OnPaint như sau:

```

CPaintDC dc(this); // device context for painting

// Add only the following codes.
// The previous codes is generated automatically by IDE
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(1.0f, 0.0f, 0.0f);
glBegin(GL_POLYGON);
    glVertex2f(-0.5, -0.5);
    glVertex2f(-0.5, 0.5);
    glVertex2f(0.5, 0.5);

```

```

        glVertex2f(0.5, -0.5);
    glEnd();
    glFlush();

    SwapBuffers(dc.m_ps.hdc); // Swap the front and the back buffer to show the
    created graphics

```

- Giải phóng tài nguyên:
  11. Bổ sung hàm RemoveOpenGL(), kiểu void, vào lớp MFCTemplateView
  12. Soạn thảo hàm RemoveOpenGL như sau:

```

//Delete Rendering Context
if(wglGetCurrentContext() != NULL)
    wglMakeCurrent(NULL, NULL);

if(m_hRC != NULL)
{
    wglDeleteContext(m_hRC);
    m_hRC = NULL;
}

```

13. Bổ sung hàm xử lý sự kiện WM\_DESTROY, Visual Studio sinh ra OnDestroy().
14. Soạn hàm OnDestroy như sau:

```

CView::OnDestroy();
//Remove OpenGL
RemoveOpenGL();

```

- Biên dịch và thực thi:
  15. Biên dịch và thực thi.

#### Bài tập mở rộng từ Dự án 1:

- 1) Thay đoạn mã ở bước 9. để dùng glVertex2fv thay vì glVertex2f.
- 2) Ở bước 9., dùng glVertex3f hay glVertex3fv để vẽ các polygon ở mặt phẳng XY, XZ, và YZ. Những polygon nên có tọa độ x, y, và z đều nằm trong đoạn [-1,1]. Những polygon nào trong số các polygon trên nhìn thấy được, vì sao?
- 3) Ở bước 9., thay vì dùng GL\_POLYGON, hãy dùng các hằng khác (xem BlueBook) để vẽ điểm, đoạn thẳng, tam giác, chuỗi tam giác, ...

## 2. Dự án 2: Vẽ hình vuông động

#### Nội dung công việc:

- Bổ sung Header files cho OpenGL.
- Cài đặt OpenGL.
- Thiết lập viewport.
- Tham số hóa đa giác ở Dự án 1 để tâm của nó thay đổi theo thời gian.
- Cài đặt và sử dụng Timer.
- Giải phóng tài nguyên.
- Biên dịch và thực thi.

#### Các bước thực hiện:

 Sử dụng lại dự án MFCTemplate, i.e., copy sang một folder khác rồi thay đổi mã như sau

- Bổ sung Header files cho OpenGL.
- Cài đặt OpenGL.



- Thiết lập viewport.
  1. Thực hiện các bước 1.→9. trong Dự án 1.
- Tham số hóa đa giác ở Dự án 1 để tâm của nó thay đổi theo thời gian.
  2. Bổ sung biến thành viên m\_angle vào lớp MFCTemplateView.

```
float m_angle;
```

3. Bổ định nghĩa hằng DEG2RAD để đổi từ Degree sang Radian, bổ sung vào tập tin CPP của lớp MFCTemplateView.

```
#define DEG2RAD (3.14159f/180.0f)
```

4. Bổ sung biến thành viên m\_angle vào lớp MFCTemplateView.
5. Bổ sung hàm xử lý cho thông điệp WM\_PAINT, Visual Studio sinh ra OnPaint()
6. Soạn hàm OnPaint như sau:

---

```
CPaintDC dc(this); // device context for painting

// Add only the following codes.
// The previous codes is generated automatically by IDE
glColor3f(1.0f, 0.0f, 0.0f);

GLfloat cx, cy;
cx = 0.5*cos(DEG2RAD*m_angle); cy = 0.5*sin(DEG2RAD*m_angle);

glBegin(GL_POLYGON);
    glVertex2f(cx - 0.5, cy - 0.5);
    glVertex2f(cx - 0.5, cy + 0.5);
    glVertex2f(cx + 0.5, cy + 0.5);
    glVertex2f(cx + 0.5, cy - 0.5);
glEnd();
glFlush();

SwapBuffers(dc.m_ps.hdc); // Swap the front and the back buffer to show the
                           created graphics
```

---

- Cài đặt và sử dụng Timer.
  7. Bổ sung biến m\_hTimer như sau vào lớp MFCTemplateView

```
int m_hTimer; // keep the timer's handle
```

8. Bổ sung dòng lệnh sau đây vào phần hiện thực của hàm OnCreate:

```
//Create Timer
//handle of the timer = 1, foreach 100 milisec OnTimer will be called
m_hTimer = SetTimer(1, 100, NULL);
```

9. Bổ sung hàm xử lý sự kiện WM\_TIMER. Visual Studio sinh ra hàm OnTimer(param).
10. Soạn hàm OnTimer: Các dòng lệnh sau đây sẽ thay đổi tham số m\_angle và yêu cầu vẽ lại màn hình.

---

```
CView::OnTimer(nIDEvent);

//RePaint the window
m_angle += 10.0f;
if(m_angle > 360) m_angle = m_angle - 360.0f;

InvalidateRect(NULL, false);
```

---

- Giải phóng tài nguyên:
  11. Bổ sung hàm RemoveOpenGL(), kiểu void, vào lớp MFCTemplateView
  12. Soạn thảo hàm RemoveOpenGL như sau:

---

```
//Delete Rendering Context
if(wglGetCurrentContext() != NULL)
    wglMakeCurrent(NULL, NULL);

if(m_hRC != NULL)
{
    wglDeleteContext(m_hRC);
    m_hRC = NULL;
}
}
```

---

13. Bổ sung hàm xử lý sự kiện WM\_DESTROY, Visual Studio sinh ra OnDestroy().
14. Soạn hàm OnDestroy như sau:

---

```
CView::OnDestroy();

//Remove OpenGL
RemoveOpenGL();

//Remove timer
if(m_hTimer > 0){
    KillTimer(m_hTimer);
    m_hTimer = -1;
}
}
```

---

- Biên dịch và thực thi:
  15. Biên dịch và thực thi.

#### Bài tập mở rộng từ Dự án 2:

- 1) Thay đổi nội dung hàm OnPaint để hình vuông xoay quanh một tâm của nó. Hướng dẫn:
  - Tọa độ 1 đỉnh của nó tính theo tham số m\_angle.
  - 3 Tọa độ còn lại tính tương đối từ tọa độ đầu: theo phương trình  $Q = P + v \cdot t$ . Ở đó Q là tọa độ cần tính, P là tọa độ biết trước, v là vector chỉ phương PQ – v là vector đơn vị, và t là chiều dài cạnh hình vuông.

### **3. Dự án 3: Vẽ tứ diện tĩnh**

#### Nội dung công việc:

- Bổ sung Header files cho OpenGL.
- Cài đặt OpenGL.
- Thiết lập viewport.
- Khởi động OpenGL.
- Vẽ tứ diện (tetrahedron) với các tọa độ được biết trước.
- Giải phóng tài nguyên.
- Biên dịch và thực thi.

#### Các bước thực hiện:

-  Sử dụng lại dự án MFCTemplate, i.e., copy sang một folder khác rồi thay đổi mã như sau

- Bổ sung Header files cho OpenGL.

- Cài đặt OpenGL.
- Thiết lập viewport.
  1. Thực hiện các bước 1.→9. trong Dự án 1.
- Khởi động OpenGL:
  2. Bổ sung hàm thành viên InitOpenGL() vào lớp MFCTemplateView  
`void InitOpenGL(void);`
  3. Soạn InitOpenGL():

---

```
//setup projection type
//glOrtho: define viewing volume
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(
    -1.0, //left
    1.0,  //right
    -1.0, //bottom
    1.0,  //top
    -2.0, //near
    2.0   //far
);
//Default MatrixMode is MODELVIEW
glMatrixMode(GL_MODELVIEW);

//setup background color, or clear color
glClearColor(0.1f, 0.7f, 0.7f, 1.0f);

//Enable using depth info. to remove hidden surface
glEnable(GL_DEPTH_TEST);
```

---

4. Gọi hàm InitOpenGL() từ hàm OnCreate, tạo ở bước 1.

```
//Initialize OpenGL
InitOpenGL();
```

- Vẽ tứ diện (tetrahedron) với các tọa độ được biết trước.
  5. Bổ sung biến thành viên m\_tetra\_vertices của lớp MFCTemplateView.
    - Trong file \*.H của MFCTemplateView:  
`static const float m_tetra_vertices[][3];`
    - Trong file \*.CPP của MFCTemplateView:  
`const float CMFCTemplateView::m_tetra_vertices[][3] = {`  
`{0.0, 0.0, 1.0},`  
`{0.0, 0.942809, -0.33333},`  
`{-0.816497, -0.471405, -0.33333},`  
`{0.816497, -0.471405, -0.33333}`  
`};`
  6. Bổ sung hàm xử lý cho thông điệp WM\_PAINT, Visual Studio sinh ra OnPaint()
  7. Soạn hàm OnPaint như sau:

---

```
CPaintDC dc(this); // device context for painting

// Add only the following codes.
// The previous codes is generated automatically by IDE

//clear the frame buffer
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

//the order of the vertices in a triangle is important!
glBegin(GL_TRIANGLES);
    //Face 1: defined by vertices: 0, 2, 1; colored: red
```

---

```

glColor3f(1.0f, 0.0f, 0.0f);
glVertex3fv(m_tetra_vertices[0]);
glVertex3fv(m_tetra_vertices[2]);
glVertex3fv(m_tetra_vertices[1]);

//Face 2: defined by vertices: 0, 1, 3; colored: green
glColor3f(0.0f, 1.0f, 0.0f);
glVertex3fv(m_tetra_vertices[0]);
glVertex3fv(m_tetra_vertices[1]);
glVertex3fv(m_tetra_vertices[3]);

//Face 3: defined by vertices: 0, 3, 2; colored: blue
glColor3f(0.0f, 0.0f, 1.0f);
glVertex3fv(m_tetra_vertices[0]);
glVertex3fv(m_tetra_vertices[3]);
glVertex3fv(m_tetra_vertices[2]);

//Face 4: defined by vertices: 3, 1, 2; colored: x
glColor3f(0.0f, 0.5f, 0.9f);
glVertex3fv(m_tetra_vertices[3]);
glVertex3fv(m_tetra_vertices[1]);
glVertex3fv(m_tetra_vertices[2]);

glEnd();
glFlush();

SwapBuffers(dc.m_ps.hdc); // Swap the front and the back buffer to show the
created graphics

```

- Giải phóng tài nguyên:
  8. Thực hiện các bước 11. → 14. từ Dự án 1:
- Biên dịch và thực thi:
  9. Biên dịch và thực thi.

Bài tập mở rộng từ Dự án 3:

- 1) Ở bước 3., thay đổi giá trị “near” của hàm glOrtho từ -2, -1.5, -1, -0.5, 0, 0.5, 1.0  
Biên dịch và chạy lại để nhìn thấy sự khác nhau.
- 2) Thay đổi màu của các mặt trong tứ diện, màu bất kỳ.
- 3) Thử xóa dòng glEnable(GL\_DEPTH\_TEST) ở InitOpenGL. Sau đó, biên dịch và chạy lại.
- 4) Thử xóa giá trị GL\_DEPTH\_BUFFER\_BIT ở OnPaint. Sau đó, biên dịch và chạy lại.

## IV. Viết chương đồ họa đơn giản:

### Sử dụng GLUT + OpenGL:

1. Vẽ hình tròn bằng cách xuất ra tập các điểm trên biên của nó. Khoảng cách giữa các điểm tính theo góc là  $10^\circ$ .
2. Vẽ hình tròn bằng cách nối các đỉnh ở câu 1.
3. Chia hình tròn ở câu 2. thành các tam giác bằng cách vẽ đường nối với giữa tâm đến các đỉnh nằm trên biên của nó.
4. Chọn 3 tam giác từ tập tam giác ở câu 3, sao cho chúng nằm cân đối trên hình tròn. Tô màu 3 tam giác này. Không vẽ các tam giác khác, để cho 3 tam giác tạo thành cái quạt 3 cánh. Cho cái quạt xoay quanh tâm, cứ sau 100milisec thì quay.

### Sử dụng MFC + OpenGL:

5. Hiện thực các bài tập 1. → 4. nói trên bằng MFC + OpenGL.
6. Dùng MFC + OpenGL: Hiện thực vẽ hình tứ diện và điều chỉnh vị trí camera theo thời gian. Nghĩa là chuyển Dự án 4 trong phần GLUT vào MFC.