# MỤC LỤC

[MỤC LỤC 1](#_Toc332702879)

[DANH MỤC HÌNH ẢNH 4](#_Toc332702880)

[DANH MỤC BẢNG BIỂU 4](#_Toc332702881)

[LỜI NÓI ĐẦU 5](#_Toc332702882)

[Phần 1: Tổng quan về thư viện đồ họa OpenGl 6](#_Toc332702883)

[Chương I: Giới thiệu thư viện OpenGl 6](#_Toc332702884)

[1.1. Lịch sử phát triển 6](#_Toc332702885)

[1.2. Khái niệm 6](#_Toc332702886)

[Chương II: Cơ chế hoạt động của OpenGL 7](#_Toc332702887)

[2.1. Cơ chế hoạt động 7](#_Toc332702888)

[2.2. Môi trường hoạt động 8](#_Toc332702889)

[Phần 2: Tìm hiểu OpenGL trong không gian 3D 8](#_Toc332702890)

[Chương I: Cơ bản về OpenGL 8](#_Toc332702891)

[1.3. Cấu trúc lệnh OpenGl và OpenGl Utility Toolkit 8](#_Toc332702892)

[1.3.1. Cấu trúc lệnh OpenGL 8](#_Toc332702893)

[1.3.2. Cấu trúc lệnh OpenGl Utility Toolkit 9](#_Toc332702894)

[1.4. Cấu trúc một chương trình OpenGl 9](#_Toc332702895)

[1.5. Một số hàm vẽ cơ bản 10](#_Toc332702896)

[1.5.1. Lệnh cơ sở 10](#_Toc332702897)

[1.5.2. Lệnh vẽ đối tượng phức tạp 12](#_Toc332702898)

[Chương II: Các phép biến đổi trong không gian 3D 13](#_Toc332702899)

[2.1. Tổng quan về các phép biến đổi trong không gian 3D của OpenGL 13](#_Toc332702900)

[2.2. Một số hàm hỗ trợ biến đổi hình học 13](#_Toc332702901)

[2.3. Thiết lập quan sát 14](#_Toc332702902)

[2.3.1. Thiết lập view 14](#_Toc332702903)

[2.3.2. Thiết lập phép chiếu 15](#_Toc332702904)

[a. Chiếu phối cảnh 15](#_Toc332702905)

[b. Phép chiếu trực giao 16](#_Toc332702906)

[Chương III: Tô màu trong OpenGL 17](#_Toc332702907)

[3.1. Chế độ màu trong OpenGL 17](#_Toc332702908)

[3.2. Thiết lập mô hình tô màu 18](#_Toc332702909)

[Chương IV: Chiếu sáng trong OpenGL 18](#_Toc332702910)

[4.1. Tổng quan về chiếu sáng 18](#_Toc332702911)

[4.2. Nguồn sáng 19](#_Toc332702912)

[4.2.1. Thiết lập nguồn sáng 19](#_Toc332702913)

[4.2.2. Chuyển mảng cho OpenGL 20](#_Toc332702914)

[4.2.3. Kích hoạt nguồn sáng 21](#_Toc332702915)

[4.3. Định nghĩa tính chất vật liệu 21](#_Toc332702916)

[4.3.1. Màu và ánh sáng 21](#_Toc332702917)

[4.3.2. Thiết lập các mảng giá trị vật liệu 22](#_Toc332702918)

[4.3.3. Chuyển mảng vật liệu cho OpenGL 22](#_Toc332702919)

[4.4. Định nghĩa các pháp tuyến 23](#_Toc332702920)

[4.4.1. Pháp tuyến và vector 24](#_Toc332702921)

[4.4.2. Tính toán pháp tuyến 24](#_Toc332702922)

[4.5. Xác định kiểu đổ bóng và kích hoạt việc kiểm tra chiều sâu 25](#_Toc332702923)

[4.6. Định nghĩa đèn chiếu 26](#_Toc332702924)

[Phần 3: Ứng dụng OpenGl cài đặt vẽ các hình khối cơ bản trong không gian 3D 27](#_Toc332702925)

[Chương I: Mô phỏng các hình khối 3D cơ bản 27](#_Toc332702926)

[1.1. Sử dụng hàm vertext cơ bản 27](#_Toc332702927)

[1.2. Sử dụng hàm hỗ trợ có sẵn trong thư viện GLUT 28](#_Toc332702928)

[1.1.1. Hàm vẽ tứ diện 28](#_Toc332702929)

[1.1.2. Hàm vẽ hình hộp 28](#_Toc332702930)

[1.1.3. Hàm vẽ khối hình nhẫn 29](#_Toc332702931)

[Chương II: Mô phỏng tô màu và chiếu sáng 29](#_Toc332702932)

[LỜI KẾT 31](#_Toc332702933)

[TÀI LIỆU THAM KHẢO 32](#_Toc332702934)

# DANH MỤC HÌNH ẢNH

[Hình ảnh 1: Cơ chế hoạt động của OpenGL 5](#_Toc331784169)

[Hình ảnh 2: Mô tả cấu trúc lệnh trong OpenGL 6](#_Toc331784170)

[Hình ảnh 3: Hình ảnh mô tả các kiểu vẽ 10](#_Toc331784171)

[Hình ảnh 4: Hình ảnh minh họa thiết lập view 13](#_Toc331784172)

[Hình ảnh 5: Mô tảhàm glFrustum() 14](#_Toc331784173)

[Hình ảnh 6: Mô tả hàm gluPerspective() 14](#_Toc331784174)

[Hình ảnh 7: Mô tả phép chiếu trực giao của hàm glOrtho() 15](#_Toc331784175)

[Hình ảnh 8: Mô phỏng tứ diện bằng hàm vertex cơ bản 25](#_Toc331784176)

[Hình ảnh 9: Mô phỏng 2 dạng tứ diện đều bằng hàm có sẵn 26](#_Toc331784177)

[Hình ảnh 10: Mô phỏng hai dạng hình hộp bằng thư viện có sẵn 27](#_Toc331784178)

[Hình ảnh 11: Mô phỏng hai dạng khối hình nhẫn bằng thư viện có sẵn 27](#_Toc331784179)

[Hình ảnh 12: Mô phỏng thiết lập chiếu sáng khối cầu màu xanh 28](#_Toc331784180)

# DANH MỤC BẢNG BIỂU

[Bảng 1: Một số lệnh hợp lệ trong cặp glBegin(); glEnd(); 8](#_Toc331784181)

[Bảng 2: Tên và ý nghĩa các kiểu vẽ cơ bản 9](#_Toc331784182)

[Bảng 3: Các hàm hỗ trợ sẵn để vẽ các đối tượng hình học phức tạp 10](#_Toc331784183)

[Bảng 4: Bảng giá trị tham số kiểu nguyên 15](#_Toc331784184)

[Bảng 5: Đối số thứ hai của hàm glLight() 18](#_Toc331784185)

# LỜI NÓI ĐẦU

Đồ họa máy tính là một lĩnh vực lý thú và có nhiều ứng dụng trong thực tế, nó góp phần làm cho giao tiếp giữa con người và máy tính trở nên thân thiện hơn. Trong thành công của kỹ thuật đồ họa ngày nay không thể không nói đến sự phát triển vượt bậc của tốc độ phần cứng lẫn hệ điều hành. Nhưng bản thân kỹ thuật đồ họa thì có bước tiến nhảy vọt từ những phép tính toán học phức tạp đến những thư viện đồ họa được tạo sẵn. Các thư viện này cho phép giảm nhẹ thời gian và công sức của người lập trình, bởi với chúng, để có được một “tác phẩm” đồ họa không đòi hỏi phải có kiến thức cao siêu về hình học, tạo bóng…, mà chỉ cần ứng dụng các hàm đã được tạo sẵn. Một trong những thư viện đó là thư viện OpenGL, được xem như là tiêu chuẩn thiết kế công nghiệp cho đồ họa ba chiều.

Để mở rộng thêm kiến thức về lập trình đồ họa vi tính, đặc biệt là đồ họa ba chiều, chúng em muốn nghiên cứu về thư viện OpenGL một cách có chiều sâu. Chính bởi thế, nhóm chúng em đã quyết định lựa chọn đề tài: “*Tìm hiểu về OpenGL về vẽ hình trong không gian 3D và cài đặt minh họa*” để có cơ hội tìm hiểu sâu hơn về thư viện đồ họa OpenGL. Qua thời gian tìm hiểu, chúng em đã tìm hiều được tổng quan về thư viện OpenGL và thấy rằng đây là một thư viện rất thú vị.

Chúng em xin gửi lời cảm ơn chân thành tới cô giáo: *“****Vũ Minh Yến****”* đã tận tâm chỉ bảo và giúp đỡ chúng em hoàn thành đề tài này. Trong quá trình làm đề tài sẽ không tránh khỏi những sai sót, khuyết điểm. Vì vậy, nhóm thực hiện chúng em hy vọng nhận được sự đánh giá và đóng góp nhiệt tình từ phía thầy cô và các bạn để bài của nhóm chúng em được hoàn thiện hơn.

***Chúng em xin chân thành cảm ơn!***

# Phần 1: Tổng quan về thư viện đồ họa OpenGl

## Chương I: Giới thiệu thư viện OpenGl

### Lịch sử phát triển

Nguyên thủy, OpenGL do Silicon Graphics Incorporated (SGI) thiết kế để dùng cho các trạm làm việc (workstation) đồ họa IRIS của họ. IRIS GL đối với các cấu hình phần cứng khác nhau thì phát sinh lỗi.

OpenGL là kết quả nỗ lực của SGI nhằm cải thiện tính tương thích của IRIS GL. Ngôn ngữ mới này mang tất cả các ưu điểm của GL, đồng thời “open” (mở) nghĩa là dễ dàng tương thích với các loại cấu hình phần cứng, cũng như các hệ điều hành khác nhau.

Version 1.0 của OPENGL được giới thiệu vào ngày 01/07/1992.

Để đảm bảo tính “mở”, mọi sự nâng cấp OPENGL đều phải thông qua Ủy ban xem xét kiến trúc OpenGL (OpenGL Architecture Review Board AEB) gồm các thành viên sáng lập là SGI, Digittal Equipment Corporation, IBM ,Intel và Microsoft. ARB họp mỗi năm hai lần. [(Cá](http://kilobooks.com/)c công ty khác cũng có thể tham gia thảo luận nhưng không có quyền bỏ phiếu).

[Ope](http://kilobooks.com/)n GL version1.1 được ARB thông qua vào tháng 12/1995.

### Khái niệm

OpenGL (Open Graphics Library) được định nghĩa là “giao diện phần mềm cho phần cứng đồ họa”. Thực chất, OpenGL là một thư viện các hàm đồ họa được xem là tiêu chuẩn thiết kế công nghiệp cho đồ họa ba chiều.

Với giao diện lập trình mạnh mẽ, OpenGL cho phép tạo các ứng dụng 3D phức tạp với độ tinh vi, chính xác cao mà người thiết kế không phải đánh vật với các núi công thức toán học và các mã nguồn phức tạp.

Có thể định nghĩa OpenGL như sau:

OpenGL là một thư viện mã nguồn mở về các tính năng đồ họa, do một tập đoàn công nghiệp quản lý, nhằm cung cấp các đặc tả đa hệ thống cho việc dựng hình và tăng tốc phần cứng đối với hình ảnh 3 chiều.

OpenGL là một thư viện mã nguồn mở về các thủ tục đã được chuẩn hóa đối với các tính năng ba chiều, mỗi tính năng lại bao gồm các phương trình toán học và những phép tính phức tạp.

Thư viện có những thủ tục phục vụ cho các tính năng đồ họa thông thường như: xây dựng các bề mặt ba chiều từ những tam giác cực nhỏ, tạo cho bề mặt một hình nền hay những cạnh trơn nhẵn thay vì phải viết những đoạn mã riêng hay tạo lại và thay đổi đặc tả của các phép tính quan trọng thì các nhà phát triển giờ đây chỉ cần gọi thủ tục có sẵn trong OpenGL.

## Chương II: Cơ chế hoạt động của OpenGL

### 2.1. Cơ chế hoạt động

Evaluator

 Rasterization

 Textrue Assembly

Vertex Data

 Pixel Operations

Per-fragment operation

Pixel Data

Per-vertex operations and primitive assembly

Display List

 Framebuffer

Hình ảnh : Cơ chế hoạt động của OpenGL

OpenGL có cơ chế hoạt động theo kiểu ống dẫn tức là đầu ra của giai đoạn trước là đầu vào của giai đoạn sau.Từ sơ đồ thì các thành phần của cơ chế được giải thích như sau:

* Display List: Là nơi lưu lại một số lệnh để xử lý sau.
* Evaluator: Xấp xỉ các đường cong và mặt phẳng hình học bằng cách đánh giá các đa thức của dữ liệu đưa vào.
* Per-vertex operations and primitive assembly: Xử lý các primitive (điểm,đoạn,đa giác) được mô tả bởi các vertex. Các vertex sẽ được xử lý và các primitive được cắt xén vào viewport để chuẩn bị cho khâu kế tiếp.
* Rasterization: sinh ra một loạt các địa chỉ framebuffer và các giá trị liên quan bằng cách sử dụng mô tả 2 chiều của điểm, đoạn,đa giác.Mỗi phần tử (fragment) được sinh ra sẽ đưa vào giai đoạn kế tiếp.
* Per-fragment operations: Các tác vụ sau cùng (cập nhập có điều kiện cho framebuffer dựa vào dữ liệu vào và dữ liệu được lưu trữ trước đó của giá trị z (đối với z buffering), thực hiện trộn màu cho các pixel và làm một số thao tác khác) sẽ được thực hiện trên dữ liệu trước khi nó được chuyển thành pixel và đưa vào framebuffer.

Trong trường hợp dữ liệu vào ở dạng pixel không phải vertex, nó sẽ đưa thẳng vào giai đoạn xử lý pixel. Sau giai đoạn này, dữ liệu ở dạng pixel sẽ được lưu trữ vào texture memory để đưa vào giai đoạn Per-fragment operation hoặc đưa vào Rasterization như dữ liệu dạng Vertex(tức là các điểm).

### 2.2. Môi trường hoạt động

OpenGL là thư viện độc lập với hệ điều hành bởi nó không thực hiện tác vụ thuộc về hệ điều hành cũng như không nhận dữ liệu nhập của người dùng (người dùng giao tiếp với OpenGL thông qua OpenGL API).

Để sử dụng được thư viện người dùng cần tải thư viện OpenGL về và cài đặt cùng với 1 ngôn ngữ lập trình khác như: C/C++, C#, Java,…

# Phần 2: Tìm hiểu OpenGL trong không gian 3D

## Chương I: Cơ bản về OpenGL

### Cấu trúc lệnh OpenGl và OpenGl Utility Toolkit

#### Cấu trúc lệnh OpenGL

Cú pháp lệnh của OpenGL:

* Lệnh: OpenGL sử dụng tiền tố **gl** và các từ tiếp theo được bắt đầu bằng một ký tự hoa (ví dụ: **glClearColor()**).
* Hằng: OpenGL sử dụng tiền tố **GL\_** các từ tiếp theo được viết hoa được ngăn cách bằng ký tự gạch dưới (ví dụ: **GL\_COLOR\_BUFFER\_BIT**)

**gl Color 3 f(…);**

Thư viện

Hàm cơ bản

Số đối số

Kiểu dữ liệu của đối số

Hình ảnh : Mô tả cấu trúc lệnh trong OpenGL

#### Cấu trúc lệnh OpenGl Utility Toolkit

OpenGl Utility Toolkit là một thư viện cung cấp sẵn một số hàm cấp cao được xây dựng nên từ OpenGL. Thư viện này được xây dựng lên nhằm khắc phục một số hạn chế của OpenGL. Thư viện còn được gọi tắt là GLUT

Cấu trúc lệnh của thư viện GLUT giống như cấu trúc của một lệnh OpenGL nhưng những lệnh thuộc thư viện này có tiền tố bắt đầu là *glut.*

### Cấu trúc một chương trình OpenGl

Qua tìm hiểu về thư viện OpenGL và thực hành, làm các bài tập cơ bản. Nhóm chúng em nhận thấy với một chương trình đồ họa sử dụng thư viện OpenGL dù là một chương trình đơn giản hay phức tạp hơn nữa thì cũng đều sử dụng một cấu trúc chương trình chung. Sau đây, chúng em sẽ trình bày khung cần thiết cho một chương trình sử dụng thư viện đồ họa OpenGL. (Ngôn ngữ mô tả là C++)

/\* Khai báo thư viện\*/

#include <>

/\* Hàm theo tác vẽ\*/

void display(){

// Tập lệnh vẽ hình

}

/\* Thân chương trình\*/

**int** main(int avg, char \*\* argc){

glutInit(&avg, argc);

glutInitDisplayMode(Glenum mode | Glenum mode …); // Thiết lập chế độ vẽ

glutInitWindowSize(**int** width, **int** height);// thiết lập độ rộng của sổ

glutInitWindowPosition(**int** x, **int** y); // vị trí đặt cửa sổ trên màn hình

glutCreateWindow(char s[] ); // thiết lập tên cho cửa sổ

//Khởi tạo chế độ đồ họa và một số chế độ khác

glutDisplayFunc(display); // Vẽ

// Thực thi một số hàm khác như chuyển hệ quan sát, vẽ lại,…

glutMainLoop();//Bắt đầu chế độ lặp (dừng cửa sổ đồ họa)

return 0;

}

### Một số hàm vẽ cơ bản

#### Lệnh cơ sở

Trong lập trình OpenGL, mọi thao tác vẽ hàm sẽ được thể hiện trong hàm vẽ với tên thường thấy là **display().** Trong OpenGL, các đối tượng hình học phần lớn được cấu thành từ các lệnh đỉnh hay còn gọi là vertex. Một hình được xác định bằng một số các vertex và được đặt trọng cặp lệnh:

glBegin(Glenum mode);

// các vertex và một số hàm thiết lập thông số...

glEnd();

Bảng : Một số lệnh hợp lệ trong cặp glBegin(); glEnd();

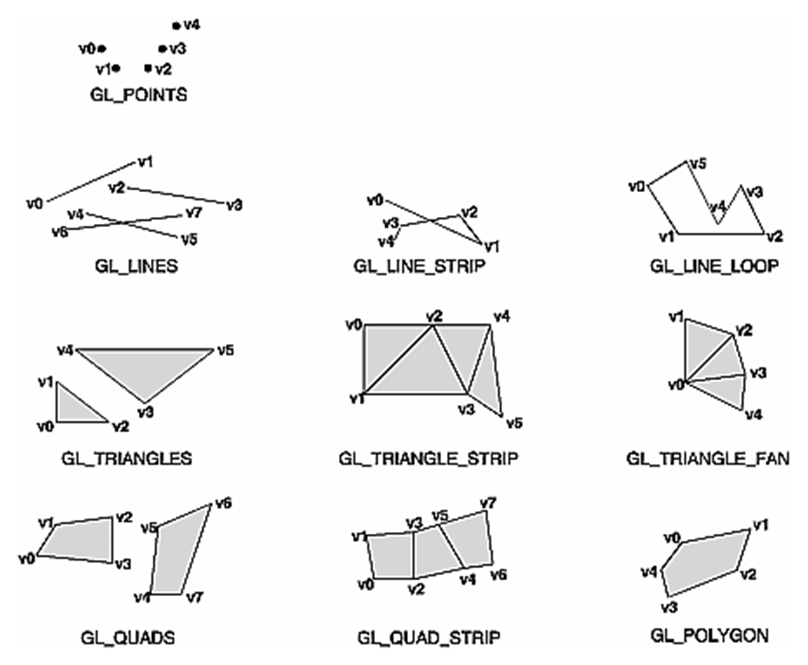
|  |  |
| --- | --- |
| **Lệnh** | **Ý nghĩa** |
| glVertex\*() | Khai báo vertex |
| glColor\*() | Sử dụng màu |
| glIndex\*() | Thiết lập chỉ mục màu |
| glNormal\*() | Thiết lập tọa độ vector chỉ phương |
| glEvalCoord\*() | Sinh tọa độ |
| glEvalPoint\*() | Sinh tọa độ |
| glCallList() | Thực thi Display list |
| glCallLists() | Thực thi Display list |
| glTexCoord\*() | Thiết lập tọa độ texture |
| glEdgeFlag\*() | Điểu khiển việc vẽ cạnh |
| glMaterial\*() | Thiết lập thuộc tính chất liệu |

Không có lệnh nào hợp lệ nằm giữa cặp lệnh *glBegin()* và *glEnd()* ngoài những lệnh nói trên, và nếu thực hiện lời gọi hàm khác sẽ sinh ra lỗi. Một số câu lệnh về mảng đỉnh như glEnableClientState() và glVertexPointer() khi được gọi giữa ***glBegin ()*** và ***glEnd ()***, có hành vi không xác định nhưng không nhất thiết tạo ra lỗi. Ngoài ra, các thủ tục liên quan đến OpenGL, như ***glX\*()*** thường có hành vi không xác định khi nằm giữa cặp lệnh ***glBegin()*** và ***glEnd()***, những trường hợp này nên tránh dùng, việc gỡ lỗi có thể là khó khăn hơn.

Bảng : Tên và ý nghĩa các kiểu vẽ cơ bản

|  |  |
| --- | --- |
| **Hằng số** | **Ý nghĩa** |
| GL\_POINT | Vẽ điểm |
| GL\_LINES | Vẽ đường thẳng nối 2 điểm |
| GL\_POLYGON | Vẽ hình tứ giác lồi |
| GL\_TRIANGLES | Vẽ tam giác |
| GL\_QUADS | Vẽ tứ giác |
| GL\_LINE\_STRIP | Tập hợp các đoạn thẳng được nối với nhau |
| GL\_LINE\_LOOP | Vẽ đường gấp khúc khép kín |
| GL\_TRIANGLE\_STRIP | Vẽ một tập hợp các tam giác liền nhau có chung một cạnh |
| GL\_QUAD\_STRIP | Vẽ một tập hợp các tứ giác liền nhau có chung một cạnh |
| GL\_TRIANGLE\_FAN | Vẽ hình quạt |

Hình ảnh mình họa:



Hình ảnh : Hình ảnh mô tả các kiểu vẽ

#### Lệnh vẽ đối tượng phức tạp

Bảng : Các hàm hỗ trợ sẵn để vẽ các đối tượng hình học phức tạp

|  |  |
| --- | --- |
| **Tên hàm** | **Hình mô tả** |
| void **glutWireSphere**(GLdouble radius, GLint slices, GLint stacks);  void **glutSolidSphere**(GLdouble radius, GLint slices, GLint stacks); | Khối cầu |
| void **glutWireCube**(GLdouble size);  void **glutSolidCube**(GLdouble size); | Khối lập phương |
| void **glutWireTorus**(GLdouble innerRadius, Gldouble outerRadius, GLint nsides, GLint rings);  void **glutSolidTorus**(GLdouble innerRadius, GLdouble outerRadius, GLint nsides, GLint rings); | Khối nhẫn |
| void **glutWireIcosahedron**(void);  void **glutSolidIcosahedron**(void); | Khối 20 mặt tam giác đều |
| void **glutWireOctahedron**(void);  void **glutSolidOctahedron**(void); | Khối 8 mặt tam giác đều |
| void **glutWireTetrahedron**(void);  void **glutSolidTetrahedron**(void); | Khối tứ diện đều |
| void **glutWireDodecahedron**(GLdouble radius);  void **glutSolidDodecahedron**(GLdouble radius); | Khối 12 mặt ngũ giác đều |
| void **glutWireCone**(GLdouble radius, GLdouble height, GLint slices, GLint stacks);  void **glutSolidCone**(GLdouble radius, GLdouble height, GLint slices, GLint stacks); | Khối nón |
| void **glutWireTeapot**(GLdouble size);  void **glutSolidTeapot**(GLdouble size); | Ấm trà |

## Chương II: Các phép biến đổi trong không gian 3D

### 2.1. Tổng quan về các phép biến đổi trong không gian 3D của OpenGL

Các phép biến đổi hình học trong không gian 3D của OpenGL được xây dựng dựa trên cơ sở đồ họa của không gian thực 3 chiều. Các phép biến đổi trên một điểm thực chất vẫn là nhân ma trận biểu diễn của điểm v với ma trận biến đổi M kích thước 4x4:

v’ = M.v;

Tọa độ của một điểm 3D trong OpenGL được thể hiện (x, y, z, w)T, thông thường w=1, vector điểm được biểu diễn dưới dạng cột.

### 2.2. Một số hàm hỗ trợ biến đổi hình học

Trong OpenGL, để thực hiện các bước chuyển đổi ta cần thông báo với chương trình rằng sẽ thực thi các bước biến đổi để khởi tạo ma trận ban đầu. Sau đó các hàm biến đổi được gọi lần lượt. Lưu ý thứ tự thực hiện biến đổi sẽ ngược lại với thứ tự viết lệnh biến đổi bởi tại mỗi thời điểm lệnh ma trận biến đổi sẽ được đẩy vào stack và quá trình lấy ra thực hiện sẽ bị ngược.

Lệnh thao tác trên stack:

* Đẩy vào stack: void **glFushMatrix**();
* Lấy ma trận ở đỉnh stack: void **glPopMatrix**();

Lệnh khai báo thực hiện phép chuyển đổi và khởi tạo ma trận ban đầu:

* **glMatrixMode**(GL\_MODELVIEW);
* **glLoadIdentity**();

Các hàm biến đổi được hỗ trợ:

* Tịnh tiến: void **glTranslate{fd}**(TYPEx, TYPE y, TYPEz);
* Quay: void **glRotate{fd}**(TYPE angle, TYPE x, TYPE y, TYPE z);
* Biến đổi tỉ lệ: void **glScale{fd}**(TYPEx, TYPE y, TYPEz);

Ngoài ra, ta có thể tự thiết lập phép biến đổi khác thông qua việc thiết lập ma trận biến đổi: void **glLoadMatrix{fd}**(const TYPE \*m); và để thực thi phép biến đổi bằng ma trận biến đổi ta dùng lệnh void **glMultMatrix{fd}**(const TYPE \*m);

### 2.3. Thiết lập quan sát

#### 2.3.1. Thiết lập view

Viewport là một khung hình chữ nhật trong cửa sổ chương trình vẽ. Nó là nơi mà vật thể sẽ được vẽ vào (có thể có kích thước nhỏ hơn, bằng hoặc lớn hơn kích thước cửa sổ chương trình). Hàm void **glViewport**(GLint x, GLint y, GLsizei width, Glsizei height) xác định một khung nhìn có tọa độ góc trái dưới là (x,y) so với góc trái dưới của cửa sổ và được tính bằng pixel; width và height xác định kích thước của khung nhìn. Mặc định, viewport có kích thước bằng kích thước cửa sổ. Tỷ lệ giữa chiều dài và chiều rộng của viewport nên bằng với tỉ lệ tương ứng trong không gian nhìn mà ta đã xác định thông qua các phép chiếu, nếu không hình ảnh sẽ bị biến dạng.

Ngoài ra, GLUT có một hàm giúp thiết lập view một cách nhanh chóng.

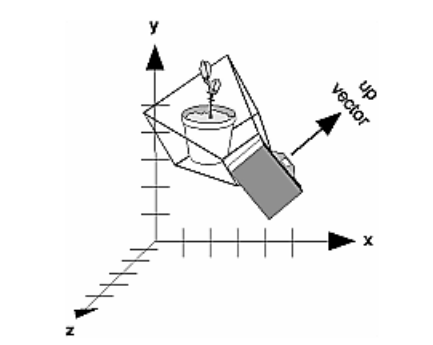
void **gluLookAt**(*GLdouble eyex, GLdouble eyey, GLdouble eyez, Gldouble centerx, GLdouble centery, GLdouble centerz, GLdouble upx, Gldouble upy, GLdouble upz*)

Trong đó:

* + (eyex, eyey, eyez) là vị trí đặt của view,
  + (centerx, centery, centerz) là điểm nằm trên đường thẳng xuất phát từ tâm view hướng ra ngoài,
  + (upx, upy, upz) là vector chỉ hướng lên trên của view

***Ví dụ:***

* (eyex, eyey, eyez) = (4, 2, 1)
* (centerx, centery, centerz) = (2, 4, -3)
* (upx, upy, upz) = (2, 2, -1)



Hình ảnh : Hình ảnh minh họa thiết lập view

#### 2.3.2. Thiết lập phép chiếu

Trước khi thực hiện các thao tác chiếu, chúng ta gọi 2 hàm

**glMatrixMode**(GL\_PROJECTION);

**glLoadIdentity**();

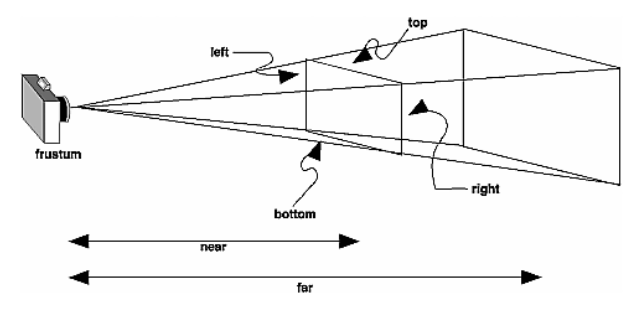
##### Chiếu phối cảnh

Đặc điểm: đối tượng càng lùi ra xa thì trông càng nhỏ.

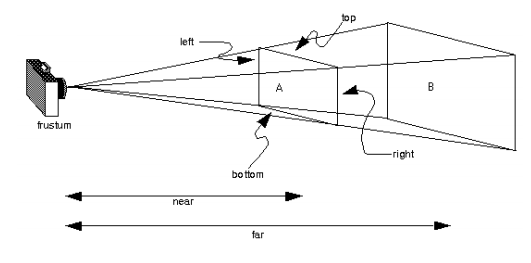
Hàm:

void **glFrustum**(GLdouble left, GLdouble right, GLdouble bottom,GLdouble top, GLdouble near, GLdouble far);

void **gluPerspective**(GLdouble fovy, GLdouble aspect, GLdouble near, GLdouble far);



Hình ảnh : Mô tảhàm glFrustum()



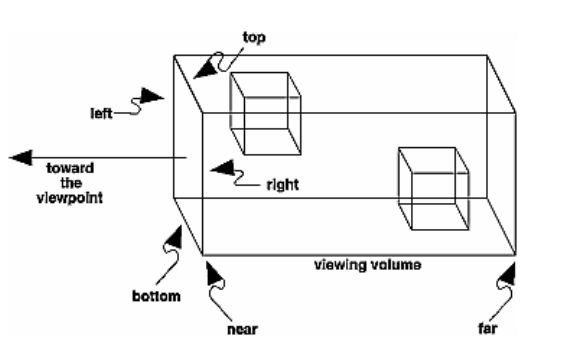
Hình ảnh : Mô tả hàm gluPerspective()

##### Phép chiếu trực giao

Đặc điểm: Kích thước của đối tượng không phụ thuộc khoảng cách

Hàm:

void **glOrtho**(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far);



Hình ảnh : Mô tả phép chiếu trực giao của hàm glOrtho()

## Chương III: Tô màu trong OpenGL

### 3.1. Chế độ màu trong OpenGL

OpenGL hỗ trợ hai chế độ màu: RGBA và Color – Index.

Trong khuôn khổ bài tập lớn này, chúng em xin phép chỉ tìm hiểu và trình bày và chế độ màu RGBA.

Chế độ màu RGBA là chế độ 3 màu cơ bản: Red – Green – Blue. Thông số A không tác động trực tiếp lên màu của Pixel. Thành phần A được dùng để xác định độ trong suốt hay một thông số nào đó cần được quan tâm.

Để thiết lập màu cho chế độ này ta có thể sử dụng các hàm:

void **glColor3**{b s i f d ub us ui} (TYPEr, TYPEg, TYPEb);

void **glColor4**{b s i f d ub us ui} (TYPEr, TYPEg, TYPEb, TYPEa);

void **glColor3**{b s i f d ub us ui}**v** (const TYPE\*v);

void **glColor4**{b s i f d ub us ui}**v** (const TYPE\*v);

Nếu giá trị tham số chuyền là kiểu thực thì giá trị sẽ thuộc đoạn [0, 1].

Bảng : Bảng giá trị tham số kiểu nguyên

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Suffix** | **Data Type** | **Minimum Value** | **Min Value Maps to** | **Maximum Value** | **Max Value Maps to** |
| b | 1-byte integer | -128 | -1.0 | 127 | 1.0 |
| s | 2-byte integer | -32,768 | -1.0 | 32,767 | 1.0 |
| i | 4-byte integer | -2,147,483,648 | -1.0 | 2,147,483,647 | 1.0 |
| ub | unsigned 1- byte integer | 0 | 0.0 | 255 | 1.0 |
| us | unsigned 2-byte integer | 0 | 0.0 | 65,535 | 1.0 |
| ui | unsigned 4-byte integer | 0 | 0.0 | 4,294,967,295 | 1.0 |

### 3.2. Thiết lập mô hình tô màu

Một đoạn thẳng có thể được tô bởi một màu đồng nhất (chế độ flat) hay bởi nhiều màu sắc khác nhau (chế độ smooth). Để thiết lập chế độ shading phù hợp, chúng ta có thể sử dụng hàm

void **glShadeModel** (GLenum mode);

Trong đó mode là chế độ mong muốn, nhận 1 trong 2 giá trị GL\_SMOOTH hoặc GL\_FLAT.

## Chương IV: Chiếu sáng trong OpenGL

### 4.1. Tổng quan về chiếu sáng

Ánh sáng là thiết lập quan trọng trong việc mô tả đối tượng trong không gian thực 3 chiều. Ánh sáng cung cấp màu sắc và tạo ánh sáng cho đối tượng 3- D, làm hình ảnh nhận được trở nên thật hơn.

OpenGL có bốn kiểu ánh sáng đó là: ánh sáng môi trường, ánh sáng khuếch tán, ánh sáng phản chiếu, và nguồn phát sáng. Ánh sáng môi trường là ánh sáng đến từ mọi hướng cùng lúc, ví dụ ánh sáng trong một phòng được chiếu sáng đầy đủ là ánh sáng môi trường. Ánh sáng khuếch tán là ánh sáng đến từ một hướng, chiếu vào bề mặt đối tượng, làm cho bề mặt này trở nên sáng chói hơn, sau đó ánh sáng bị khuếch tán đi mọi hướng. Ánh sáng phản chiếu là ánh sáng tạo đốm phản chói, thường là màu trắng, trên các bề mặt có tính phản chiếu cao. Cuối cùng, nguồn phát là nguồn ánh sáng phát ra từ đối tượng như bóng đèn chẳng hạn.

Tất nhiên, mọi nguồn ánh sáng, kể cả ánh sáng môi trường, đến từ một nơi nào đó. Có thể có 8 nguồn phát khác nhau trong một chương trình OpenGL. Thêm vào đó, có thể xác định tính phản xạ ánh sáng của bề mặt đối tượng trong một hoạt cảnh. Đối tượng sẽ phụ thuộc vào ánh sáng mà nó phản xạ. Ví dụ khi thiết lập bề mặt đối tượng phản xạ ánh sáng xanh dương, OpenGL sẽ thể hiện đối tượng với các bóng xanh dương khác nhau.

### 4.2. Nguồn sáng

#### 4.2.1. Thiết lập nguồn sáng

Khi thiết lập mảng giá trị có các loại ánh sáng, ta cần xác định giá trị của các thành phần RGBA tương ứng đó là các thành phần màu đỏ, màu xanh dương, màu xanh lục và alpha (thành phần này có thể bỏ qua). Các giá trị này có giá trị từ 0.0 đến 1.0. Ví dụ, các mảng sau chứa giá trị dương định nghĩa một nguồn sáng :

Glfloat ambientLight0[] = {0.3f, 0.3f, 0.3f, 1.0f};

Glfloat diffuseLight0[] = {0.5f, 0.5f, 0.5f, 1.0f};

Glfloat specularLight0[] = {0.0f, 0.0f, 0.0f, 1.0f};

Trong mảng ambientLigh0[], lượng màu được thiết lập cho ánh sáng môi trường gồm 30% đỏ và 30% xanh dương, 30% xanh lục. (Giá trị 1.0 là thành phần alpha). Mảng diffuselight0[], chứa lượng màu cho ánh sánh khuếch tán, gồm 50% đỏ và 50% xanh dương, và 50% xanh lục. Do các thành phần màu trong các mảng trên bằng nhau, chúng định nghĩa ánh sáng môi trường và ánh sáng khuếch tán là ánh sáng trắng với các độ sáng riêng. Thay đổi tỷ lệ phần trăm các thành phần màu sẽ mang lại ánh sáng màu; Ví dụ lượng màu đỏ lớn hơn lượng màu khác, thì ánh sáng sẽ mang màu đỏ.

Một yếu tố nữa của ánh sáng cũng cần được thiết lập mảng giá trị đó là vị trí nguồn sáng. Để thiết lập mảng giá trị của vị trí nguồn sáng ta cần xác định các giá trị X, Y, Z, W của nguồn sáng. Trong đó, X, Y, Z là tọa độ của nguồn sáng, W chỉ có hai giá trị là 0 và 1. Nếu W bằng 0 ta có nguồn sáng vô hướng, là nguồn sáng ở khoảng cách vô hạn so với vật, tức các tia sáng đến với vật là song song với nhau. Nếu W bằng 1 ta có nguồn sáng theo vị trí, là nguồn sáng có vị trí gần hoạt cảnh, các tia sáng chiếu vào đối tượng theo các góc khác nhau. Ví dụ mặt trời là nguồn sáng vô hướng, bóng đèn là nguồn sáng theo vị trí. Một ví dụ về mảng giá trị của nguồn sáng như sau:

glLightfv (GL\_LIGHT0,GL\_POSITION, positionLight0);

#### 4.2.2. Chuyển mảng cho OpenGL

Các giá trị định nghĩa nguồn sáng được chuyển cho OpenGL bằng cách gọi hàm glLight() như sau: ***glLightfv (GLenum light, GLenum pname, const GLfloat \*params);***

Ví dụ: glLightfv (GL\_LIGHT0,GL\_AMBIENT,ambientLight0);

glLightfv (GL\_LIGHT0,GL\_DIFFUSE, diffuseLight0);

glLightfv (GL\_LIGHT0,GL\_SPECULAR, specularLight0);

glLightfv (GL\_LIGHT0,GL\_POSITION, positionLight0);

Ba đối số của glLightfv() gồm: nguồn sáng để định nghĩa, hằng số thể hiện tính chất ánh sáng mà ta muốn thay đổi, và địa chỉ mảng chứa các giá trị dùng thay đổi tính chất ánh sáng. Đối với các thuộc tính ánh sáng môi trường, khuếch tán, và phản chiếu, mảng chứa các giá trị RGBA. Đối với địa chỉ nguồn sáng, mảng chứa các tọa độ X,Y,Z,W của nguồn sáng.

Do một chương trình OpenGL có thể có tới 8 nguồn sáng, đối số đầu tiên là hằng số **từ GL\_LIGHT0** đến **GL\_LIGHT7**

Đối số thứ hai là một trong mười hằng số ở bảng dưới. Các hằng số thường dùng với các hàm này là GL\_AMBIENT,GL\_DIFFUSE, GL\_SPECULAR,và GL\_POSITION:

Bảng : Đối số thứ hai của hàm glLight()

|  |  |
| --- | --- |
| **Hằng số** | **Ý nghĩa** |
| GL\_AMBIENT | Xác định ánh ánh sáng môi trường |
| GL-CONSTANT\_ATTENUATION | Xác định lượng giảm ánh sáng theo một hệ số |
| GL\_DIFFUSE | Xác định ánh sáng khuếch tán |
| GL-LINEAR\_ATTENUATION | Xác định lượng giảm ánh sáng trên cơ sở khoảng cách từ nguồn sáng đến đối tượng |
| GL\_POSITION | Xác định vị trí nguồn sáng |
| [GL](http://kilobooks.com/)\_QUADRATIC\_ATTENUATION | Xác định lượng giảm ánh sáng theo bình phương khoảng cách từ nguồn sáng đến đố tượng |
| GL\_SPECULAR | Xác định ánh sáng phản chiếu |
| GL\_SPOT\_CUTOFF | Xác định góc trải của đèn chiếu |
| GL\_SPOT\_DIRECTION | Xác định hướng đèn chiếu |
| GL\_SPOT\_EXPONENT | Xác định cường độ đèn chiếu, có tính đến góc của ánh sáng. |

#### 4.2.3. Kích hoạt nguồn sáng

Khi đã có định nghĩa nguồn sáng, ta phải kích hoạt chúng, tương tự như việc bật công tắt đèn, bằng hàm glEnable():

glEnable(GL\_LIGHTING);

glEnable(GL\_LIGHT0);

Lời gọi đầu kích hoạt chiếu sáng. Lời gọi thứ hai bật nguồn sáng 0.Vị trí một nguồn sáng thì bị ảnh hưởng bởi phép biến hình đối tượng như mọi đối tượng khác. Việc biến hình xảy ra khi hàm glLight() được gọi để xác định vị trí hay hướng của nguồn sáng. Do đó, phải gọi glMatrixMode(gl\_MODELVIEW) và glLoadldentity() trước khi gọi glLight(), như trước khi định nghĩa các vector của đa giác. Dĩ nhiên, nếu muốn biến đổi vị trí và hướng nguồn sáng theo các đối tượng khác trong hoạt cảnh, phải thực hiện các phép tịnh tiến, co giãn, và quay trước khi gọi glLight().

### 4.3. Định nghĩa tính chất vật liệu

Tính chất phản chiếu của vật liệu rất ảnh hưởng đến cảnh. Ví dụ, hình khối màu trắng trong ánh sáng đỏ sẽ có màu đỏ chứ không là màu trắng. Trong khi, một khối đỏ trong ánh sáng xanh lục sẽ có màu đen. Ngoài ra, các vật liệu có độ sáng khác nhau sẽ phản xạ ánh sáng khác nhau.

#### Màu và ánh sáng

Màu của đối tượng trong cảnh phụ thuộc vào màu của ánh sáng mà nó phản xạ. Ví dụ, nếu đối tượng phản xạ ánh sáng xanh lục, thì nó sẽ có màu xanh lục trong ánh sáng trắng. Do các thành phần màu đỏ và xanh dương của ánh sáng bị hấp thụ chỉ còn lại thành phần xanh lục đến được mắt người.

Thông thường cả ánh sáng môi trường và ánh sáng khuếch tán phản xạ từ đối tượng theo cùng một cách. Tức là nếu đối tượng phản xạ ánh sáng màu lục, thì nó phản xạ ánh sáng màu lục đối với ánh sáng môi trường lẫn ánh sáng khuếch tán. Trong khi ánh sáng phản chiếu luôn luôn cùng màu với ánh sáng chiếu vào đối tượng. Ví dụ, xét một khối vuông được thể hiện trong ánh sáng vàng (màu vàng là tổ hợp đồng lượng của màu đỏ và màu xanh dương). Nếu muốn khối vuông có màu đỏ, vật liệu đó phải phản xạ màu đỏ của ánh sáng môi trường và ánh sáng khuếch tán. Để làm khối vuông trở nên sáng hơn, thì phải thiết lập cho nó tính phản xạ màu vàng đối với ánh sáng phản chiếu.

OpenGL cho phép định nghĩa các tính chất của đối tượng, để xác định loại ánh sáng mà đối tượng phản xạ, và như vậy gián tiếp xác định màu cuối cùng của đối tượng.

#### Thiết lập các mảng giá trị vật liệu

Bước đầu tiên để định nghĩa các tính chất của vật liệu là thiết lập các mảng chứa các giá trị phản xạ ánh sáng môi trường khuếch tán và ánh sáng phản chiếu. Các thành phần của mảng này tương tự như mảng giá trị của ánh sáng. Ví dụ như:

Glfloat materialAmbient[] = {0.0f, 0.7f, 0.0f, 1.0f}

Glfloat materialSpecular[] = {1.0f, 1.0f, 1.0f, 1.0f}

Mỗi mảng này chứa các giá trị RGBA cho một kiểu phản xạ ánh sáng riêng biệt, đó là các giá trị RGBA tương ứng với màu sắc ánh sáng dội ra từ vật liệu đối tượng. Mảng materialAmbient[] định nghĩa vật liệu phản xạ 70 % màu xanh lục trong ánh sáng môi trường khuếch tán mà nó nhận được, nhưng không phản xạ màu đỏ và xanh dương. Mảng materialSpecular[] định nghĩa vật liệu phản xạ ánh sáng trắng, gồm 100% các thành phần đỏ, xanh dương, xanh lục.

#### Chuyển mảng vật liệu cho OpenGL

Để chuyển mảng vật liệu cho OpenGL, dùng các con trỏ trỏ đến mảng làm đối số trong lời gọi ***glMaterialv()***

Ví dụ:

GLMaterialfv(GL\_FRONT,GL\_AMBIENT\_AND\_DIFFUSE, materialAmbient);

GLMaterialfv (GL\_FRONT, GL\_SPECULAR, materialSpecular);

Các đối số hàm *glMaterialfv()* gồm một hằng xác định bề mặt đa giác mà ta muốn định nghĩa vật liệu, một hằng định nghĩa tính chất vật liệu, và con trỏ trỏ đến mảng chứa giá trị dùng thiết lập tính chất vật liệu:

Đối số đầu tiên là GL\_FRONT, GL\_BACK, hay GL\_FRONT\_AND\_BACK, tương ứng với sự lựa chọn mặt trước, mặt sau, hay cả mặt trước lẫn sau của đa giác tạo nên đối tượng. Đối với đối tượng đặc như hình khối, GL\_FRONT thường được chọn vì mặt sau sẽ không bao giờ thấy được. Nhưng nếu hình khối mất một mặt, thì GL\_FRONT\_AND\_BACK được chọn, vì ở một cảnh nào đó có thể nhìn thấy cả hai mặt sau và trước đa giác.

Đối số thứ hai có thể là một trong số các hằng ở bảng 3.2. Do đối tượng luôn phản xạ các ánh sáng môi trường và khuếch tán như nhau, nên hằng GL\_AMBIENT\_END\_DIFFUSE thường được dùng. Hàm glMaterialfv() có bốn phiên bản

Bảng 6: Đối số thứ hai của hàm glMaterialv()

|  |  |
| --- | --- |
| Hằng | Ý nghĩa |
| GL\_AMBIENT | Ánh sáng môi trường bị ánh xạ |
| GL\_AMBIENT\_AND\_DIFFUSE | Ánh sáng môi trường và khuếch tán bị ánh xạ |
| GL\_COLOR\_INDEXES | Xác định các chỉ số màu cho ánh sáng |
| GL\_DIFFUSE | Ánh sáng khuếch tán bị phản xạ |
| [GL](http://kilobooks.com/)\_EMISSION | Xác định nguồn sáng phát |
| [GL](http://kilobooks.com/)\_SHININESS | Xác định độ bóng vật liệu |
| [GL](http://kilobooks.com/)\_SPECULAR | Ánh sáng phản chiếu bị phản xạ |

Trong các hằng ở bảng trên, có hằng GL\_COLOR\_INDEXES xác định các chỉ số màu cho ánh sáng. Nhưng ở đây, ta chỉ sử dụng OpenGL ở chế độ màu RGBA. Nếu đối tượng phản xạ ánh sáng phản chiếu, ta phải xác định độ bóng vật liệu, là hằng số quyết định độ phản xạ ánh sáng phản chiếu. Lời gọi hàm glMaterialfv() lúc đó như sau:

glMaterialf (GL\_FRONT, GL\_SHININESS,60.0f);

Các đối số hàm glMaterialf() gồm một hằng xác định bề mặt đa giác cần định nghĩa vật liệu, hằng GL\_SHININESS (chỉ dùng trong trường hợp này), và một giá trị trong khoảng từ 0 đến128 định nghĩa độ bóng.

Cần phân biệt rõ sự khác nhau giữa nguồn sáng phát và ánh sáng do phản xạ. Khi định nghĩa các nguồn sáng, ta quyết định lượng các ánh sáng môi trường, khuếch tán, và phản chiếu có trong cảnh. Khi định nghĩa tính chất vật liệu, ta quyết định lượng ánh sáng do đối tượng phản xạ các ánh sáng môi trường, khuếch tán, và phản chiếu.

### Định nghĩa các pháp tuyến

Để việc chiếu sáng OpenGL được thực hiện tốt trên đối tượng, không chỉ định nghĩa các vector của đối tượng mà còn cần định nghĩa pháp tuyến cho mỗi vector.

#### Pháp tuyến và vector

Pháp tuyến là một vertor đơn vị biểu thị hướng đối diện đa giác, cho phép OpenGL tính toán góc ánh sáng chiếu vào đa giác. Pháp tuyến luôn trực giao với bề mặt mà nó biểu thị.

Pháp tuyến của các vector của một đa giác được định nghĩa bởi lời gọi hàm Normal3f() trước lời gọi glVertex() định nghĩa vector:

glBegin (GL\_POLYGON);

GlNormal3f((Glfloat) x, (GLfloat) y, (GLfloat) z);

glVertex3f(-5.0f, 0.0f, -0.5f);

glVertex3f( 5.0f, 0.0f, 0.5f);

glVertex3f( 0.5f, 0.0f, -0.5f);

glEnd();

Đối số hàm glNormal3f() là các giá trị Glfloat xác định các tọa độ X,Y, và Z của điểm cuối pháp tuyến. Có 9 phiên bản hàm glNormal(). Chú ý rằng pháp tuyến liên kết với mỗi vector được định nghĩa trong đoạn mã trên sẽ tồn tại cho đến khi glNormal() được gọi lại để định nghĩa pháp tuyến mới.

#### Tính toán pháp tuyến

Việc tính toán pháp tuyến cho một đa giác đòi hỏi một kiến thức toán học phức tạp. Ở đây chỉ trình bày một hàm tạo sẵn có tên CalcNormal () dùng tính toán pháp tuyến từ các tọa độ X,Y,Z của ba vector bất kì của một đa giác (không cùng nằm trên một đường thẳng):

double p1[] = {-0.5, 0.0, -0.5};

double p2[] = {-0.5, 0.0, 0.5};

double p3[] = { 0.5, 0.0, 0.5};

double n[3];

CalcNormal (p1, p2, p3, n) ;

Đối số của CalcNormal() là các con trỏ trỏ đến bốn mảng giá trị double. Ba mảng đầu chứa các tọa độ X,Y,Z của ba vertex. Mảng thứ tư, n[], chứa các tọa độ pháp tuyến. Sau lời gọi CalcNormal(), mảng n[] có thể dùng định nghĩa pháp tuyến cho các vertex của đa giác như sau:

glBegin (GL\_POLYGON);

GlNormal3f((Glfloat) n[0], (GLfloat) n[1], (GLfloat) n[2]);

glVertex3f(-5.0f, 0.0f, -0.5f);

glVertex3f( 5.0f, 0.0f, 0.5f);

glVertex3f( 0.5f, 0.0f, -0.5f);

glEnd();

Đoạn code dưới đây mô tả sơ lược các phép toán trong CalcNormal();

Void CalcNormal (double \* p1,duoble \* p2,double \* p3,duoble \* n) {

//Tạo hai vertor từ các con trỏ.

Double a[3], b[3];

a[0] = p2[0] – p1[0];

a[1] = p2[1] – p1[1];

a[2] = p2[2] – p1[2];

a[0] = p3[0] – p1[0];

a[1] = p3[1] – p1[1];

a[2] = p3[2] – p1[2];

//Tính toán chéo của haivertor.

n[0] = a[1] \* b[2] – a[2] \* b[1];

n[1] = a[2] \* b[0] – a[0] \* b[2];

n[2] = a[0] \* b[1] – a[1] \* b[0];

//Tiêu chuẩn hóa vertor mới.

Double length = sqrt (n[0]\*n[0]+n[1]+n[2]\*n[2]);

If (length !=0) {

n[0]= n[0]/ length;

n[1]= n[1]/ length;

n[2]= n[2]/ length;

}

}

### Xác định kiểu đổ bóng và kích hoạt việc kiểm tra chiều sâu

Có hai kiểu bóng: mượt (smooth) và phẳng (flat). Để chọn kiểu bóng, sử dụng [hàm](http://kilobooks.com/) glShadeModel() như sau:

***glShadeModel (GL\_FLAT);***

Đối số đơn của hàm là hằng thể hiện kiểu bóng được chọn là GL\_SMOOTH hoặc GL\_FLAT. Với đối số GL\_FLAT, OpenGL phải xác định màu bóng cho từng điểm ảnh trên đa giác, trong khi với đối số GL\_SMOOTH toàn bộ đa giác cùng chung một bóng. Xem ví dụ dưới đây để thấy được sự khác biệt giữa hai đối số này.

### Định nghĩa đèn chiếu

Đèn chiếu là chùm tia tập trung hẹp. Cũng như mọi nguồn sáng OpenGL khác, cùng với việc định nghĩa đèn chiếu, ta đồng thời phải tạo cho nó một nguồn sáng theo vị trí

Đoạn code dưới định nghĩa một đèn chiếu:

Glfloat ambientLight1[] = {0.2f, 0.2f, 0.2f, 1.0f};

Glfloat diffuseLight1[] = {1.0f, 1.0f, 1.0f, 1.0f};

Glfloat specularLight[] = {0.0f, 0.0f, 0.0f, 1.0f};

Glfloat positionLight1[] = {0.0f, 0.0f,-2.0f, 1.0f};

Glfloat directionLight1[] = {0.0f, 0.0f, -1.0f};

glLightfv (GL\_LIGHT1, GL\_AMBIENT, ambientLight1);

glLightfv (GL\_LIGHT1, GL\_DIFFUSE, diffuseLight1);

glLightfv (GL\_LIGHT1, GL\_SPECULAR, specularLight1);

glLightfv (GL\_LIGHT1, GL\_POSITION, positionLight1);

glLightf (GL\_LIGHT1, GL\_SPOT\_CUTOFF, 10.0f);

glLightfv (GL\_LIGHT1, GL\_SPOT\_DIRECTION, directionLight1);

Trong đoạn mã trên, lời gọi glLightf (GL\_LIGHT1, GL\_SPOT\_CUTOFF, 10.0f) giới hạn chùm tia đèn chiếu theo cung 20 độ. Đối số 10.0 là đường góc giữa đường tâm và đường sinh chùm tia. Với góc cắt lớn hơn, chùm tia đèn chiếu sẽ rộng hơn.

Sự khác nhau tiếp theo của đèn chiếu so với các nguồn sáng khác là lời gọi glLightfv(GL\_LIGHT1, GL\_SPOT\_DIRECTION, directionLight1), thiết lập hướng chiếu của đèn. Các tọa độ hướng chiếu được chứa trong mảng directionLight1[ ] là tọa độ của điểm mà ánh sáng đèn hướng vào.

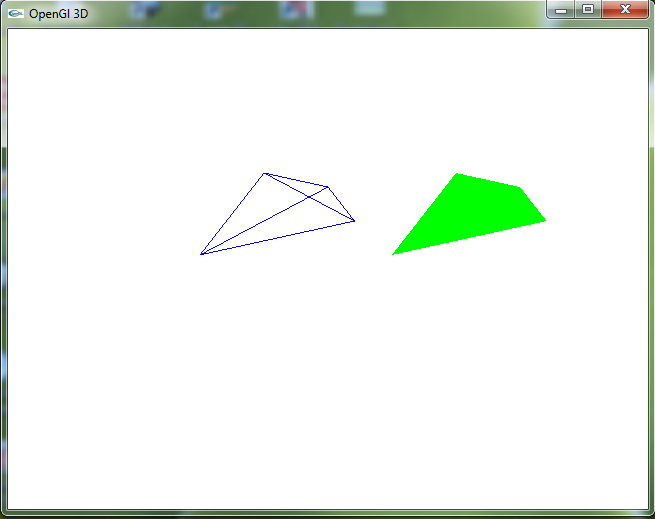
# Phần 3: Ứng dụng OpenGl cài đặt vẽ các hình khối cơ bản trong không gian 3D

Dựa vào những kiến thức mà chúng tìm hiểu được trình bày ở 2 phần trên. Sau đây, nhóm chúng em sẽ áp dụng các lý thuyết trên để minh họa các phần riêng biệt để giúp hiểu rõ lý thuyết các phần ở phía trên hơn và áp dụng các lý thuyết trên mô phỏng một số hình ảnh trong không gian thực 3 chiều.

Các cài đặt mô phỏng được nhóm chúng em thực hiện cùng với ngôn ngữ C++, viết trên phần mềm DevC phiên bản 4.9.9.2.

## Chương I: Mô phỏng các hình khối 3D cơ bản

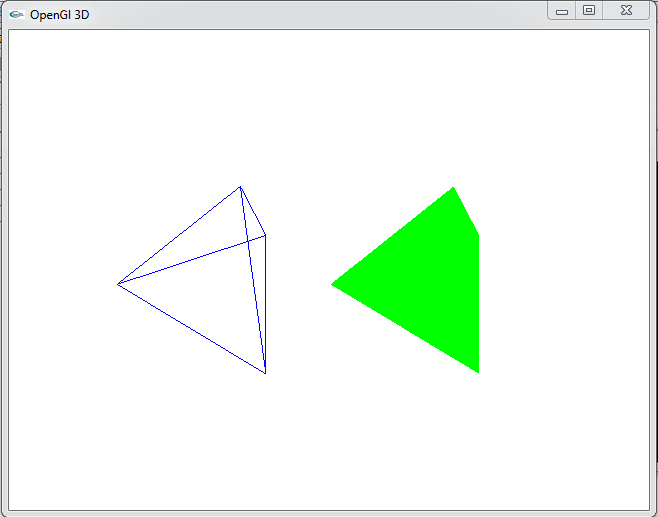
### Sử dụng hàm vertext cơ bản



Hình ảnh : Mô phỏng tứ diện bằng hàm vertex cơ bản

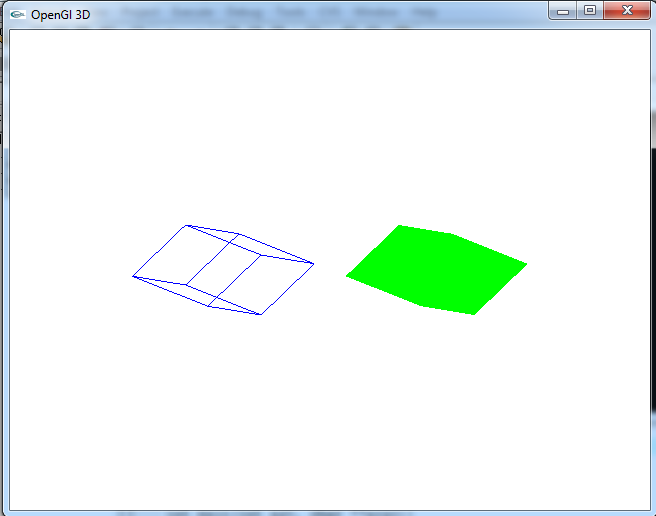
### Sử dụng hàm hỗ trợ có sẵn trong thư viện GLUT

#### Hàm vẽ tứ diện



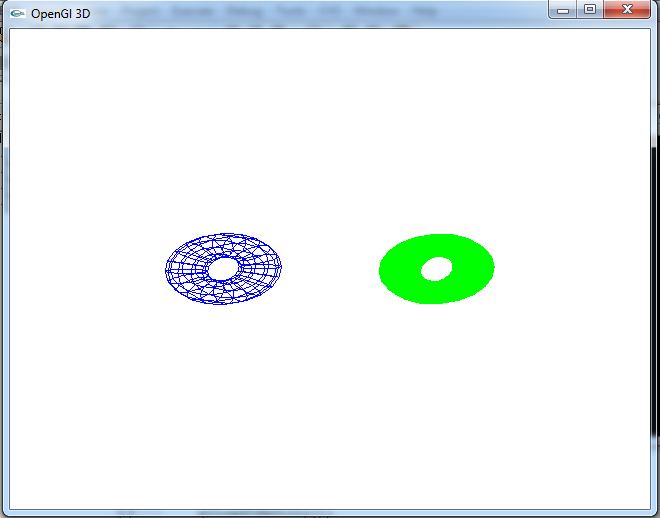
Hình ảnh : Mô phỏng 2 dạng tứ diện đều bằng hàm có sẵn

#### Hàm vẽ hình hộp



Hình ảnh : Mô phỏng hai dạng hình hộp bằng thư viện có sẵn

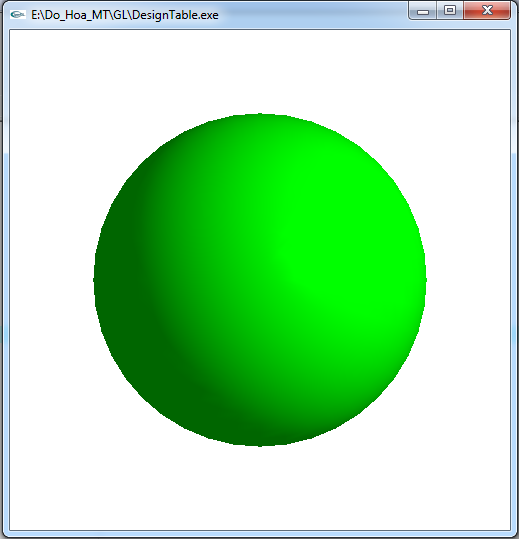
#### Hàm vẽ khối hình nhẫn



Hình ảnh : Mô phỏng hai dạng khối hình nhẫn bằng thư viện có sẵn

## Chương II: Mô phỏng tô màu và chiếu sáng

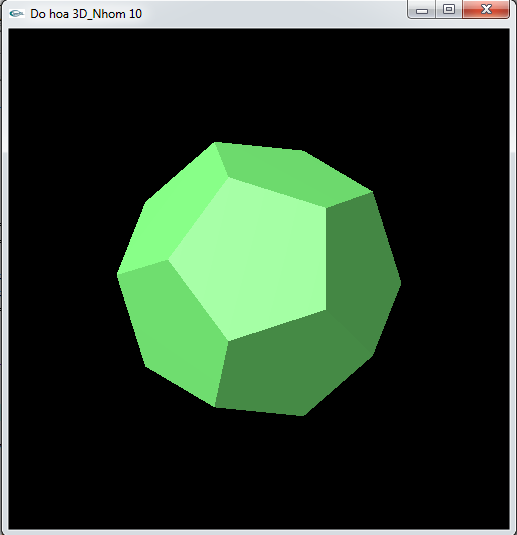
Thực bản chất, khi ta sử dụng thư viện đồ họa OpenGL để xây dựng các khối hình 3D thì việc tô màu đều được cài đặt chung với các hàm cơ bản xây dựng các khối hình. Vậy nên, trong phần minh họa về phần này nhóm chúng em sẽ viết chương trình minh họa chung cho các hàm tô màu và chiếu sáng.



Hình ảnh : Mô phỏng thiết lập chiếu sáng khối cầu màu xanh

## Chương III: Mô phỏng các hình khối và một số hình ảnh thực tế

### 3.1. Mô phỏng các hình khối cơ bản



Hình ảnh : Mô phỏng hình khối cơ bản

### 3.2. Mô phỏng một số đồ vật thực tế



# LỜI KẾT

Đây là lần đầu tiên nhóm chúng em thực kiện việc mô phỏng các hình khối 3 chiều bằng thư viện đồ họa OpenGL. Vận dụng kiến thức cơ sở về đồ họa 3 chiều đã được học ở trường, kết hợp với những tài liệu tham khảo trên mạng , sách báo, và đặc biệt là sự hướng dẫn của thầy, cô nhóm thực hiện chúng em đã hoàn thành bài tập lớn. Qua bài báo chúng em muốn gửi lời cảm ơn cô giáo hướng dẫn cô *”Vũ Minh Yến”*, rất cảm ơn cô đã nhiệt tình dạy bảo và hướng dẫn chúng em hoàn thành bài tập lớn này. Trong qua trình hoàn thành bài báo cáo không tránh khỏi thiếu sót .Vì thế , nhóm 10 chúng em rất mong nhận được chỉ dẫn, bổ sung, sửa chữa, những đóng góp và sự cảm thông sâu sắc của thầy, cô cũng như bạn bè để bài tập của chúng em được hoàn thiện hơn.

Xin chân thành cảm ơn!

# TÀI LIỆU THAM KHẢO

[1]. Jackie Neider, Tom Davis and Mason Woo, *OpenGL Progamming Guide,* Addison-Wesley Publishing Company, First Printing, 1993

[2]. Nguyễn Đức Nhiên, Đồ án môn học, Đại học Bách Khoa Hà Nội năm 2008.

[3]. Bùi Minh Trường, *Lập trình OpenGL với thư viện AUX.*

[4]. Lê Phong, *Hướng dẫn lập trình OpenGL căn bản*

[5]. ThS. Vũ Minh Yến, *Slide bài giảng Đồ họa máy tính*

[6]. http://www.youtube.com/

[7]. <http://www.swiftless.com/>

[8]. <http://www.wikepedia.org/>

[9]. <http://www.opengl.org/>