http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

ỤC LỤC

Lời mở đầu

PHẦN 1: TÌM HIỂU THƯ VIỆN ĐỒ HỌA OPENGL 1

Chương 1: Sơ lược về OPENGL 1

1.1. Lịch sử phát triển 1

1.2. Khái niệm.....1

1.3. Thành phần 5

Chương 2: Đồ họa hai chiều GDI 6

2.1. Tọa độ đề các và tọa độ màn hình.....6

2.2. Định nghĩa vertex và kiểu dữ liệu hình dạng.....8

2.3. Các phép biến hình.....10

2.4. Sử dụng ma trận cho các phép biến hình.....17

Chương 3: Đồ họa ba chiều GDI 25

3.1. Hệ tọa độ ba chiều 25

3.2. Định nghĩa đối tượng ba chiều 25

3.3. Các phương pháp thể hiện hình 3-D lên màn hình.....28

3.4. Biến hình đối tượng 3-D 31

Chương 4: Chương trình OpenGL tối thiểu36

4.1. Các kiểu dữ liệu OpenGL 36

4.2. Ngữ cảnh biểu diễn 36

4.3. Định dạng điểm vẽ 38

4.4. Tạo ngữ cảnh biểu diễn.....44

4.5. Tổng kết: 48

Chương 5: Vẽ hình và sử dụng màu: 48

5.1. Cú pháp lệnh OpenGL 48

5.2. Các trạng thái OpenGL 49

5.3. Xét một chương trình OpenGL tối thiểu.....50

5.4. Định nghĩa và vẽ điểm 53

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

5.5. Định nghĩa và vẽ đường 56
5.6. Định nghĩa và vẽ đa giác 61
5.7. Tổng kết 74

Chương 6: Các phép biếnhình OpenGL 75

6.1. Xây dựng đối tượng 3-D từ các đa giác 75
6.2. Phép chiếu 77
6.3. Phép biến hình đối tượng 79
6.4. Phép biến đổi viewport 85
6.5. Tổng kết 88

Chương 7: Chiếu sáng đối tượng 3-D 89

7.1. Các loại nguồn sáng 89
7.2. Định nghĩa một nguồn sáng 90
7.3. Định nghĩa tích chất vật liệu 92
7.4. Định nghĩa các pháp tuyến 95
7.5. Xác định kiểu bóng và kích hoạt việc kiểm tra chiều sâu 97
7.6. Định nghĩa đèn chiếu 98
7.7. Thể hiện đối tượng 3-D được chiếu sáng 99
7.8. Bảng màu logic 103
7.9. Tổng kết 107

Chương 8: Tạo cảnh 3-D 108

8.1. Sử dụng các phép biến hình OpenGL để tạo cảnh 3-D 108
8.2. Sử dụng các stack ma trận 113
8.3. Tạo ảo giác chuyển động với OpenGL 117
8.4. Tổng kết 119

Chương 9: Anh và gán cấu trúc 119

9.1. Bitmap và ảnh OpenGL 120
9.2. Bitmap phụ thuộc thiết bị và bitmap độc lập với thiết bị 125
9.3. Định dạng DIB 125
9.4. Giới thiệu lớp Cdib 129
9.5. Gán cấu trúc cho đa giác 139

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

9.6. Tổng kết.....147

ương 10: Pha trộn , giảm hiệu ưng răng cưa, và sương mù 148

10.1. Pha trộn 148

10.2. Giảm hiệu ứng răng cưa 154

10.3. Sương mù 157

Chương 11: Display List 160

11.1. Định nghĩa: 160

11.2. Tại sao phải dùng display list 160

11.3. Các tính chất của display list. 162

11.4. Các trường hợp có thể sử dụng display list. 162

11.5. Nhược điểm của display list. 162

11.6. Tạo và thực thi một display list. 163

11.7. Quản lý biến trạng thái trong display list 164

Chương 12: Quadric. 164

PHẦN 2: MÔ PHỎNG CÁC GIẢI THUẬT ĐỒ HỌA 3 D VƠI OPENGL: 166

Chương 1: Tổng quan: 166

1.1. Một số khái niệm liên quan: 166

1.2. Các phép biên đổi: 167

Chương 2: Xây dựng ứng dụng mô phỏng thuật giải: 169

2.1. Xây dựng ứng dụngOpenGL 169

2.2. Cách làm việc của ứng dụng 172

2.3. Bảng kê chương trình: 179

KILOBOOKS.COM

Lời Mở Đầu

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

ó câu rằng "một hình ảnh bằng cả nghìn lời nói ". Điều đó thật không thể phủ ận. Và rõ ràng là nếu hiển thị thông tin chỉ với các ký hiệu, chữ cái, chữ số ông thôi thì không thể hấp dẫn và dễ hiểu như khi có thêm biểu diễn đồ họa Kỹ thuật đồ hoạ cũng là công cụ không thể thiếu trong các ngành khoa học kỹ thuật, giáo dục, nghệ thuật, giải trí, quảng cáo…(để diễn đạt máy móc thiết bị, kiến trúc, cấu trúc cơ thể, thông tin thiên văn địa lý, hình ảnh minh hoạ..). Chính vì vậy, đồ họa là một vấn đề được quan tâm trong ngành công nghệ thông tin.

Cùng với sự phát triển của tin học, kỹ thuật đồ họa trên máy vi tính, ngày càng trở nên tinh xảo. Giao diện các phần mềm ngày nay trở nên thân thiện, đẹp mắt nhờ các thể hiện đồ họa. Sự hỗ trợ của tin học cho các ngành khác trở nên đắc lực hơn nhờ khả năng đồ họa vi tính. Và thế giới thực thì được biểu diễn một cách sinh động, linh hoạt, đầy màu sắc bằng không gian ba chiều.

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 một kiến thức hùng hậu về đường cong Bezier, B-spline, về hình học, tạo bóng…, mà chỉ ứng dụng các hàm tạo sẵn. Một trong những thư viện đó là OpenGL, được xem là tiêu chuẩn thiết kế công nghiệp cho đồ họa ba chiều.

Mục tiêu của luận văn này là tìm hiểu thư viện đồ họa của OpenGL trong đồ họa ba chiều, đồng thời cũng cố gắng đưa ra một ứng dụng của OpenGL trong việc minh họa các giải thuật đồ họa ba chiều.

Tuy nhiên, đề tài không thể không tránh khỏi những hạn chế và thiếu sót, nên rất mong được sự chỉ bảo, góp ý của quý Thầy Cô và bạn bè.

PHẦN 1:
TÌM HIỂU THƯ VIỆN ĐỒ HỌA OPENGL

Chương1:Sơ Lược về OPENGL

1.1.Lịch Sử Phát Triển :

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

OpenGL là kết quả nổ lực của SGI nhằm cải thiện tính tương thích của IRIS GL. Ngôn ngữ mới này có khả năng của GL, đồng thời "mở " nghĩa là dễ dàng tương thích với các loại cấu hình phần cứng, cũng như các hệ điều hành khác nhau.

Version1.0 của OPENGL được giới thiệu vào ngày 01/7/1992.

Để bảo đảm tính "mở ", mọi sự nâng cấp OpenGL phải thông qua Uy 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.

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

ic công ty khác cũng có thể tham gia thảo tuận nhưng không có quyền bỏ phiếu). en GL version1.1 được ARB thông qua vào tháng 12/1995.

1.2. Khái Niệm :

OpenGL 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 3-D 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. Và do OpenGL là tiêu chuẩn công nghiệp, các ứng dụng tạo từ nó dùng được trên các phần cứng và hệ điều hành khác nhau.

Các mục sau sẽ giới thiệu cách nhìn tổng quát về cách làm việc của OpenGL:

- Các phần tử đồ họa cơ bản và lệnh giới thiệu về các phần tử đồ họa cơ bản (primitive) và sự thực hiện lệnh
- Cách làm việc của OpenGL cho biết các loại thao tác đồ họa mà OpenGL kiểm soát
- Mô hình hoạt động nói về mô hình client/server cho việc thông dịch lệnh OpenGL
- Thao tác OpenGL cơ bản đưa ra một mô tả mức cao về cách OpenGL xử lý dữ liệu và tạo ra hình ảnh tương ứng lên bộ điệm khung.

Các phần tử đồ họa cơ bản và lệnh:

Primitive được xác định bởi nhóm của một hay nhiều vertex là điểm trong không gian. Mỗi vertex xác định một điểm, một đầu đoạn thẳng hay một đỉnh đa giác. Dữ liệu (bao gồm tọa độ vertex, màu sắc, normal, texture và cờ xác định loại cạnh) kết hợi với vertex. Khi xử lý primitive, mỗi cập vertex và dữ liệu liên kết với nó được sử lý độc lập với các cập khác, theo thứ tự và cùng một phương pháp. Ngoại lệ duy nhất là trong trường hợp khử phần khuất của primirite(clipping). Khi đó, dữ liệu vertex được sửa và các vertex khác được tạo ra. Loại clipping tuỳ thuộc loại primirite mà nhóm vertex biểu diễn.

Các lệnh luôn luôn được xử lý theo thứ tự mà nó tiếp nhận, mặt dù có sự trì hoãn không xác định trước khi lệnh có hiệu lực. Nghĩa là mỗi primirite được vẽ trọn vẹn trước khi lệnh tiếp theo có hiệu lực.

Cách làm việc của OpenGL:

OpenGL là ngôn ngữ đồ họa theo thủ tục chứ không phải ngôn ngữ mô tả.Thay vì tả các cảnh và cách chúng xuất hiện, OpenGL đưa ra các bước cần thiết để có được sự thể hiện hay hiệu quả nhất định. Các "bước"này là các lời gọi đến giao diện lập trình ứng dụng gồm xăp xỉ 120 lệnh và hàm. Chúng được dùng để vẽ các phần tử đồ họa cơ bản như điểm, đường và đa giác trong không gian ba chiều. Ngoài ra, OpenGL còn hổ trợ chiếu sáng, tô bóng, gán cấu trúc, tạo ảo giác chuyển động và các hiệu quả đặc biệt khác.

OpenGL không có các chức năng quản lý cửa sổ, tương tác với người dùng hay xuất nhập file. Môi trường chủ (tức hệ điều hành) có các chức năng này và chịu trách nhiệm thực hiện các biện pháp quản lý cho OpenGL.

Mô hình hoạt động:

Mô hình thông dịch lệnh OpenGL là client-server. Mã ứng dụng(vai trò client) đưa ra các lệnh. Lệnh được thông dịch và sử lý bởi OpenGL (vai trò server). Server và client có thể là trên cùng một máy tính khác nhau. Theo nghĩa này, OpenGL là network-transparent

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

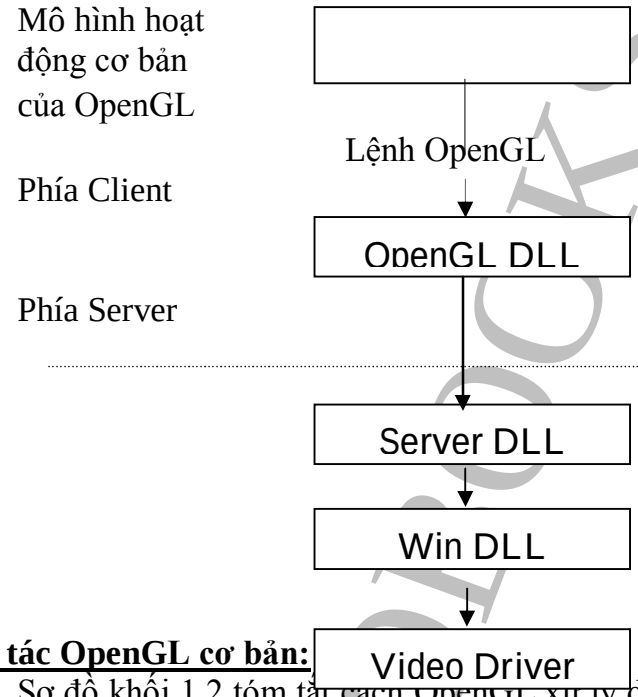
http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

n dịch là mạng trong suốt). Server duy trì nhiều ngữ cảnh OpenGL, mỗi ngữ cảnh là t trạng thái OpenGL. Client có thể nói với bất cứ ngữ cảnh nào. Giao thức mạng được dụng có thể là độc lập hóa dựa trên giao thức mạng hiện có (tức OpenGL dùng trên máy độc lập hay trên môi trường mạng). Không có lệnh OpenGL nào tiếp nhận việc nhập dữ liệu trực tiếp từ người dùng.

Cuối cùng, hệ thống cửa sổ kiểm soát tác dụng của các lệnh OpenGL trên bộ đệm khung qua các thao tác:

- Quyết định các phần của bộ đệm khung mà OpenGL có thể truy xuất tại thời điểm cho phép.
- Truyền đạt cho OpenGL thông tin về cấu trúc các phần đó.
- Như vậy, không có lệnh OpenGL nào định dạng bộ đệm khung hay khởi tạo OpenGL. Sự định dạng bộ đệm khung được thực hiện bên ngoài OpenGL trong sự liên kết với hệ thống cửa sổ. Sự khởi tạo OpenGL được tiến hành khi hệ thống cấp phát cửa sổ cho việc biểu diễn.

Hình 1.1
Mô hình hoạt động cơ bản của OpenGL



Thao tác OpenGL cơ bản:

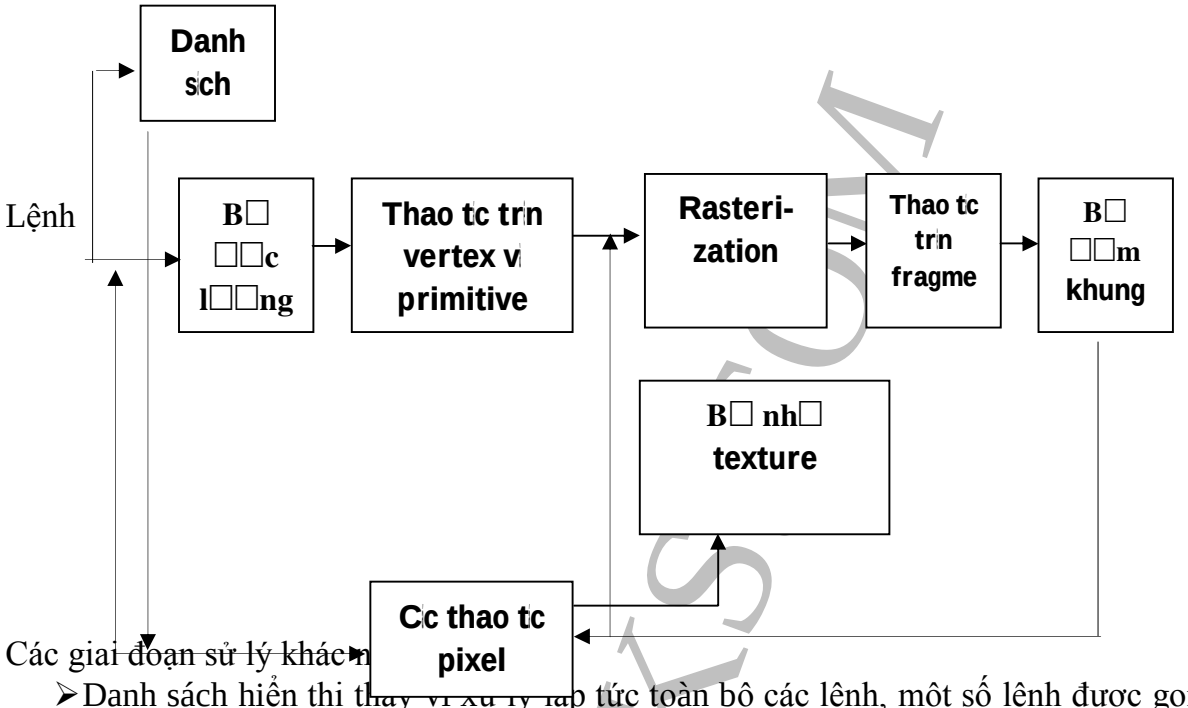
Sơ đồ khối 1.2 tóm tắt cách OpenGL xử lý dữ liệu. Các lệnh đi vào phía trái sơ đồ và qua "đường ống xử lý". Một số lệnh xác định đối tượng hình học được vẽ, và số khác kiểm soát cách quản lý đối tượng qua các giai đoạn sử lý khác nhau.

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

Hình 1.2 Sơ đồ xử lý dữ liệu của OpenGL:



➢Danh sách hiển thị thay vì xử lý lập tức toàn bộ các lệnh, một số lệnh được gom lại trong một danh sách để xử lý sau.

➢Bộ ước lượng: ước lượng là quá trình OpenGL sinh ra các vertex và thông số từ các phương trình Bézier xác định trước, cung cấp một phương cách hiệu quả để xắp xỉ hình học các bề mặt và đường cong bằng cách đánh giá các lệnh đa thức cửa giá trị đưa vào.

➢Các thao tác trên vertex và sự tổ hợp của primirite: OpenGL xử lý các primirite hình học (điểm, đoạn thẳng và đa giác). Những primirite này được biểu diễn bởi các vertex. Các vertex được biến đổi, chiếu sáng, và các primirite được khử các các phần khuất theo viewport để chuẩn bị rasterze.

➢Raterization: giai đoạn resterize tạo ra một chuỗi các địa chỉ bộ đệm khung và các giá trị liên kết sử dụng hình dạng hai chiều của điểm, đoạn thẳng hay đa giác. Các fragment tạo ra được cung cấp cho quá trình tiếp theo.

➢Các thao tác trên fragment: là các thao tác cuối cùng trên dữ liệu, trước khi lưu trữ dữ liệu dưới dạng các pixel trong bộ đệm khung.

➢Các thao tác này bao gồm việc cập nhật (có điều kiện) bộ đệm khung dựa trên giá trị lưu trữ và giá trị vừa có, việc pha trộn các màu vừa có và màu lưu trữ, cũng như thao tác mask và các thao tác logic khác trên các giá trị pixel.

➢Dữ liệu có thể được đưa vào dưới dạng cac pixel. Khi đó, sau giai đoạn thao pixel, dữ liệu pixel.

➢Hoặc được lưu trữ như là bộ nhớ texture, để dùng cho giai đoạn rasterizatrion.

➢Hay rasterize, với kết quả các fragment được kết hợp vào trong bộ đệm khung, nếu chúng phát sinh từ dữ liệu hình học.

1.3. Thành Phần:

OpenGL gồm 5 bộ hàm :

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

- Bộ hạt nhân có 115 hàm cơ bản. Tên các hàm này bắt đầu bằng GL. Windows hỗ trợ 4 chủng loại hàm khác, bao gồm thư viện OpenGL utility(tên hàm bắt đầu bằng GLU), thư viện OpenGL auxiliary(tên hàm bắt đầu bằng AUX), bộ hàm"WGL" (tên hàm bắt đầu bằng WGL), và các hàm WIN32 API (tên hàm không có tiền tố đặc biệt).
- Bộ hàm hạt nhân cho phép thiết kế các hình dạng khác nhau, tạo các hiệu quả chiếu sáng, kết hợp antialiasing và gán cấu trúc, thực hiện biến đổi ma trận…
- Do các hàm cơ bản được thể hiện ở nhiều dạng khác nhau tùy thuộc vào loại dữ liệu mà chúng tiếp nhận, nên trên thực tế có hơn 300 nguyên mẫu (prototype) các hàm cơ bản.
- Thư viện OpenGL utility gồm các hàm cao cấp. Các hàm này đơn giản hoá việc sử dụng hình ảnh cấu trúc, thực hiện việc biến đổi tọa độ mức cao, hỗ trợ tesselation đa giác, và biểu diễn các đối tượng có cơ sở đa giác như hình cầu, hình trụ hình dĩa.
- Thư viện OpenGl auxiliary gồm các hàm đặc biệt dùng đơn giản hóa các ví dụ lập trình trong sách chỉ dẫn lập trình OpenGL. Các hàm phụ thuộc platform này thực hiện các nhiệm vụ như quản ký cửa sổ, điều khiển xuất/nhập, vẽ các đối tượng 3D nhất định. Do các hàm này có mục đích thiết minh nên không được dùng trong các mã sản xuất.
- Các hàm "WGL"kết nối OpenGL với WINdows NT, cho phép người lập trình xây dựng và chọn lựa các ngữ cảnh biểu diễn, tạo các bitmap font, các hàm này chỉ dùng trên Windows NT.
- Cuối cùng, các hàm Win32 API được dùng giải quyết các định dạng điểm ảnh và tạo bộ đệm đôi.

Chương2: Đồ Họa Hai Chiều GDI

OpenGL cung cấp nhiều hàm mạnh mẽ, làm đơn giản các việc vẽ các hình ảnh phức tạp. Dù vậy, để hiểu OpenGL, cần có một chút kiến thức trong thực tiễn lập trình đồ họa tiêu chuẩn. Bước đầu tiên về mục tiêu đó là tìm hiểu cách thao tác hình ảnh hai chiều GDI

Các phần sau sẽ giới thiệu cơ sở lập trình đồ hoạ 2-D :

- Tọa độ đề các và tọa độ màn hình
- Sử dụng các Vertex để định nghĩa một hình dạng phẳng
- Tịnh tiến, co giãn và quay
- Sử dụng ma trận trong biến hình 2-D

2.1.Tọa Độ Đề Các Và Tọa Độ Màn Hình:

Để kẻ một đoạn thẳng trên màn hình, ta thường gọi các hàm định vị điểm đầu, sau đó vẽ đoạn thẳng. Ví dụ các hàm MFC được gọi là:

```
CclientDC dc(this)
dc.MoveTo(x,y)
dc.LineTo(x,y)
```

Mối quan tâm ở đây là việc biểu diễn các tọa độ x,y.
Trong tọa độ cửa sổ, cũng như tọa độ màn hình, gốc tọa độ ở vị trí góc trái trên, chiều tăng của tọa độ x về phía phải, chiều tăng tọa độ y đi về phía dưới (hình 2.1)

Tuy nhiên, đối tượng đồ họa được xác định bởi tọa độ các Vertex của nó trong hệ đề các (hình2.2). Các điểm khác biệt giữa hai tọa độ này là:
Hệ đề các xác định chiều tăng tọa độ y là chiều đi lên.

KILOBOOKS.COM
Nếu bạn cần file word? Xin hãy nhấp vào đây!

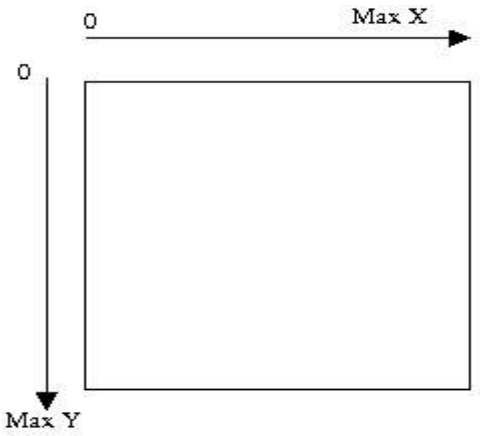
http://kilobooks.com

Hệ đề các cho phép có tọa độ âm.

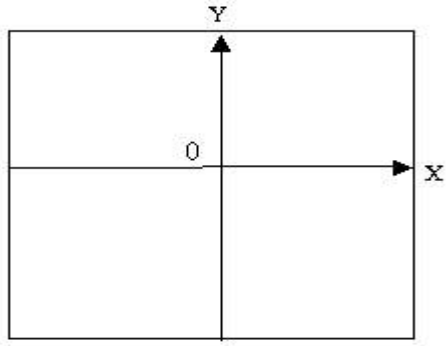
như vậy, vấn đề phát sinh khi biểu diễn một đối tượng được định nghĩa trong hệ đề :. Trong tọa độ màn hình, ta lại nhận được một tam giác như (hình 2.4), nếu với mã tạo hình như sau :

```
CCLientDC dc(this)
dc.MoveTo(2,5);
dc.LineTo(5,2);
dc.LineTo(2,2);
dc.LineTo(2,5);
```

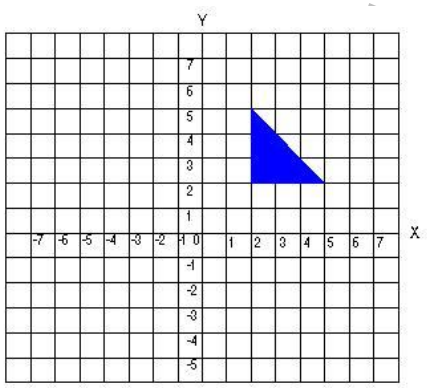
Do sự khác biệt giữa màn hình thể hiện và tọa độ đề các, cần có một phương pháp chuyển đổi giữa chúng. Thuật ngữ đồ họa gọi là mapping các điểm trong tọa độ đề các sang tọa độ màn hình để thể hiện đối tượng đúng vị trí.

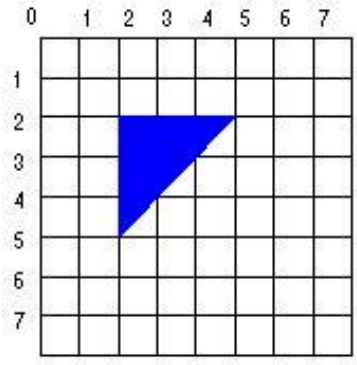


Hình 2.1 Tọa độ màn hình



Hình 2.2 Tọa độ Đề các





Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

Hình 2.3 Tam giác trong tọa độ đề các

Hình 2.4 vẽ tam giác không có sự chuyển đổi giữa tọa độ đề các và tọa độ màn hình

Nếu bỏ qua vấn đề gía trị âm, thì việc thể hiện điểm (x1,y1) trong tọa độ đề các sang điểm (x2,y2) trong tọa độ màn hình đòi hỏi công thức sau:

X2=X1;
Y2=maxY – Y1;

Dĩ nhiên, để thực hiện công thức này, trước hết cần biết kích thước hiện tại của cửa sổ, bằng cách gọi hàm GetClientRect(). Hàm này điền đầy cấu trúc RECT với kích thước cửa sổ. Sử dụng hàm MFC, đoạn mã sau vẽ tam giác đã cho lên cửa sổ, mapping giữa tọa độ đề các và tọa độ màn hình:

```
Int triangle[6]={2,5,5,2,2,2,};
CClientDC dc(this);
Int newX,newY,startX,startY;
RECT clientRect;
GetClientRect(&clientRect);
    For (int x=0;x<3;++x)
    {
       newX = triangle[x*2];
       newY = maxY – triangle[x*2+1];
    if(x = = 0)
    {
       dc. MoveTo(newX,newY);
       startX = newX;
       startY = newY;
    }
    else dc.LineTo(newX,newY);
    }
       dc.LineTO(startX,startY);

    }
```

Dòng đầu tiên cửa đoạn mã định nghĩa một mãng chức các tọa độ Đề các tam giác. Tiếp theo, đoạn mã tìm ngữ cảnh dụng cụ của cửa sổ và khai báo các biến cực bộ. Các biến newX, newY chức các tọa độ màn hình của một điểm, còn startX, startY chứa tọa độ màn hình điểm đầu tiên cửa tam giác. Cấu trúc RECT, clientRect, chứa kích thước cửa sổ. Sau đó hàm GetClientRect() được gọi để điền đầy cấu trúc clientRect. thành phần bottom chứa chiều cao cửa sổ. Giá trị này được gán cho biến maxY.

Vòng lặp for lặp đi lặp lại mảng tọa độ tam giác. Trong thân vòng lặp các tọa độ (x,y) được mapping giữa tọa độ đề các và tọa độ màn hình .

Điểm đầu tiên là điểm bắt đầu tam giác. Các điểm tiếp theo được dùng để vẽ đường thẳng tam giác. Lới gọi LienTo() bên ngoài vòng lặp nối điểm cuối cùng với điểm đầu tiên.

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

. *Định Nghĩa Vertex Và Kiểu Dữ Liệu Hình Dạng:*

ɔt hình phẳng thì được xác định bởi một tập hợp các vertex. Các vertex nối với nhau ıg các đoạn thẳng. Khi tất cả vertex được nối thì hoàn thành hình dạng.

Để quản lý các kiểu hình dạng khác nhau trong một chương trình được dễ dàng, cần định nghĩa cặp cấu trúc bao gồm kiểu dữ liệu vertex.

```
Typedef struct vertex
{
   int x, y;            //Cấu trúc này chỉ đơn giản là lưu trử
}VERTEX;       //các tọa độ đề các vertex
```

Và kiểu dữ liệu hình phẳng

```
Typedef struct shape
{
      int numVerts;//Gồm một số nguyên chỉ số lượng
      VERTEX*vertices;  //vertex trong hình và một con trỏ
}SHAPE;          //trỏ đến mảng cấu trúc
```

Với các kiểu dữ liệu mới này, ta có thể viết đoạn mã vẽ đoạn thẳng tổng quát hơn như sau :

```
VERTEX triangleVerts[3]= {2,5,5,2,2,2}
SHAPE shape1 = {3,triangleVerts};
DrawShape(shape1);

Void DrawShape(SHAPE& shape1)
{
      CclientDC dc (this);
      Int newX,newY,startX,startY;
      RECT clientRect;
      GetClientRect(&clientRect);
      Int maxY = clientRect.bottom;
      For (int x =0 ;x<shape1.numVerts;++x)
      {
            newX shape1.vertices[x].x;
            newY = maxY – shape1.vertices[x].y;
            if(x = = 0)
            {
                  dc.MoveTo(newX,newY);
                  startX = newX;
                  startY = newY;
                  }
            else dc.LineTo(newX,newY);

          }
          dc.LineTo(startX,startY);
```

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

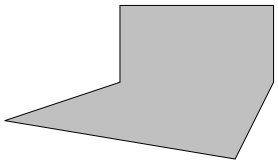
http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

}

Do DrawShape() được tổng quát hóa để làm việc với cấu trúc SHAPE, hàm này có thể vẽ mọi loại hình phẳng.Ví dụ để vẽ một hình chử nhật, thì shape1 được định nghĩa như sau:

```
VERTEX rectangleVerts[4]= {10,10,10010, 100 ,50,10,50};
SHAPE shape1 = {4,rectangleVerts};
```

Hay đối với một đa giác sáu cạnh như sau:



Hình 2.5

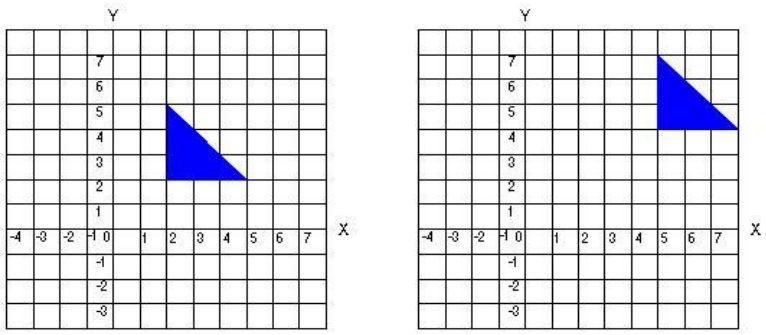
```
VERTEX shapeVerts[6]= {10,10,75,5,100,20,100,50,50,50,25};
SHAPE shape1 = {6,shapeVerts};
```

2.3.Các Phép Biến Hình:

Một hình phẳng được tạo bằng cách định nghĩa vertex cửa nó trong tọa độ đề các, mapping giữa tọa độ đề các và tọa độ màn hình, rồi vẽ các đoạn thẳng nói các vertex. Tuy nhiên, đây chỉ là sự khởi đầu. Để vẽ hình ở mọi nơi trên màn hình và theo mọi hướng, hình ảnh cần được thao tác theo nhiều cách khác nhau. Các thao tác như vậy gọi là các phép biến hình, bao gồm tịnh tuyến, co giãn và quay.

2.3.1.Phép Tịnh Tiến:

Tịnh tiến một hình đến vị trí mới chỉ đơn giản là cộng hoặc trừ tọa độ của mỗi vertex với một giá trị. Tam giác trong hình 2.6 được tịnh tiến 3 đơn vị theo trục x và 2 đơn vị theo trục Y.



Hình 2.6: Tịnh tiến một tam giác

Giả sử cần tịnh tiến 20 đơn vị theo trục X vá 30 đơn vị theo trục y, một tam giác có định nghĩa như sau:

```
VERTEX triangleVerts[3]= { 20,50,50,50,20,100};
SHAPE shape1 = {3,triangleVerts};
```

Công thức tịnh tiến mỗi vertex là :

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

```
X2 = X1 + xTranslation;
Y2 = Y1 + yTranslation;
```

Trong chương trình toàn bộ phép tịnh tiến như sau:

```
VERTEX triangleVerts[3]= { 20,50,50,50,20,100};
SHAPE shape1 = {3,triangleVerts};
DrawShape(shape1);
Translate(shape1,20,30);
Void Traslate(SHAPE& shape,intxTrans,int yTrans)
{
        for(int x =0;x< shape.numVerts;++x)
        {
                shape.vertices[x].x+ xtrans;
           shape.vertices[x].y+ ytrans;
        }
}

Void DrawShape(SHAPE& shape1)
{
        CclientDC dc (this);
        Int newX, newY, startX, startY;
        RECT clientRect;
                GetClientRect(&clientRect);
        Int maxY = clientRect.bottom;
        For (int x =0 ;x<shape1.numVerts;++x)
        {
                 newX shape1.vertices[x].x;
                 newY = maxY – shape1.vertices[x].y;
                 if(x = = 0)
                 {
                        dc.MoveTo(newX,newY);
                        startX = newX;
                        startY = newY;
                 }
                 else dc.LineTo(newX,newY);
                }
                dc.LineTo(startX,startY);
        }
```

Hàm Translate() có các đối số là tham chiếu đến cấu trúc SHAPE, lượng tịnh tiến x và lượng tịnh tiến y. Nó sử dụng vòng lặp for để gán lượng tịnh tiến trong tọa độ (x,y) của mỗi vertex.

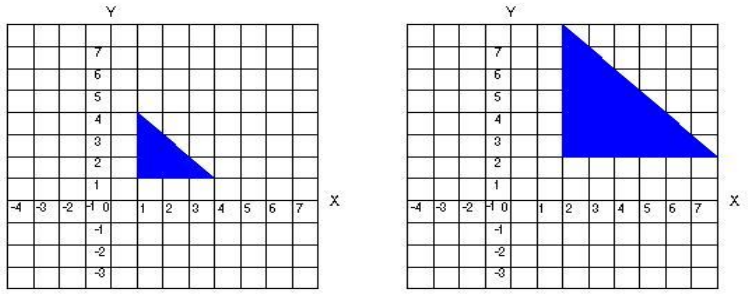
2.3.2 .Phép Co Giãn:

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

Thay vì cộng hoặc trừ các vertex với một giá trị như khi tịnh tiến, co giãn một hình là nhân các vertex của hình đó với một hệ số co giãn. Hình 2.7 biểu diễn một tam giác giãn hai lần (hệ số co giãn là 2).



Hình 2.7: Co giãn một tam giác

Chú ý rằng không chỉ riêng hình, mà toàn bộ hệ tọa độ co giãn. Tức là, một điểm có tọa độ x là 2 đơn vị, sẽ là 4 đơn vị khi co giãn.

Giả sử cần giãn 4 lần (hệ số co giãn là 4) một tam giác có định nghĩa như sau:

```
VERTEX triangleVerts[3]= { 20,50,50,50,20,100};
SHAPE shape1 = {3,triangleVerts};
```

Thì công thức co giãn mỗi vertex là :

```
X2 = X1*scaleFactor;
Y2 = Y1*scaleFactor;
```

Đoạn mã phép co giãn sẽ như sau:

```
VERTEX triangleVerts[3]= { 20,50,50,50,20,100};
SHAPE shape1 = {3,triangleVerts};
Scale(shape1,4);
DrawShape(shape1);
Void Scale(SHAPE& shape,float scaleFactor)
{
    for(int x =0;x< shape.numVerts;++x)
    {
        shape.vertices[x].x+=(int)(shape.xerticse[x]x*scaleFactor);
        shape.vertices[x].y+=(int)(shape.xerticse[x]y*scaleFactor);
    }
}
Void DrawShape(SHAPE& shape1)
{
    CClientDC dc (this);
    Int newX,newY,startX,startY;
    RECT clientRect;
    GetClientRect(&clientRect);
    Int maxY = clientRect.bottom;
    For (int x =0 ;x<shape1.numVerts;++x)
    {
        newX shape1.vertices[x].x;
```

KILOBOOKS.COM
Nếu bạn cần file word? Xin hãy nhấp vào đây!

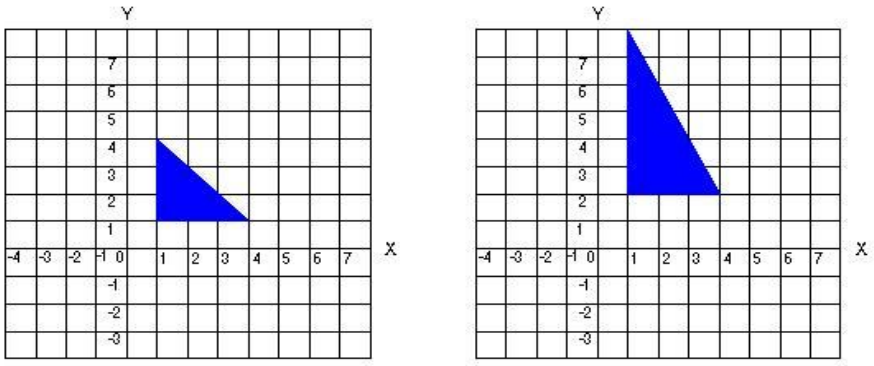
http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

```
            newY = maxY – shape1.vertices[x].y;
            if(x = = 0)
            {
                  dc.MoveTo(newX,newY);
                  startX = newX;
                  startY = newY;
            }
            else dc.LineTo(newX,newY);
          }
          dc.LineTo(startX,startY);
    }
```

Hàm Scale() có các đối số là tham chiếu đến cấu trúc SHAPE và hệ số co giãn, thì hệ số co giãn sẽ nhỏ hơn 1. Chú ý rằng có thể dùng hệ số co giãn khác nhau đối với 2 tọa độ:

```
Void Scale(SHAPE& shape, float xScale, float yScale)
{
      for(int x =0;x< shape.numVerts;++x)
      {
            shape.vertices[x].x+=(int)(shape.xerticse[x]x*xScale);
            shape.vertices[x].y+=(int)(shape.xerticse[x]y*yScale);
          }
      }
```

Trong trường hợp này, sau khi co giãn, không nhận được hình đồng dạng. Hình 2.8 biểu diễn việc co giãn một tam giác, với hệ số co giản trục x bằng 1 , và hệ số co giãn trục y bằng 2.



Hình 2.8: Co giãn một tam giác với hai hệ số co giãn khác nhau

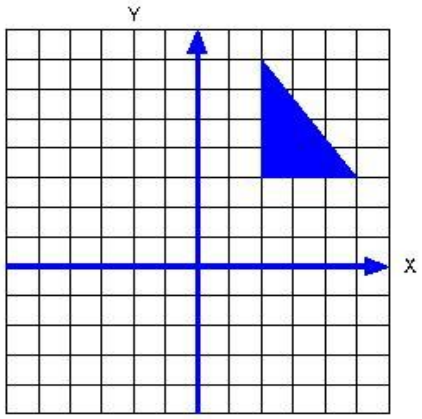
2.3.3.Phép Quay:

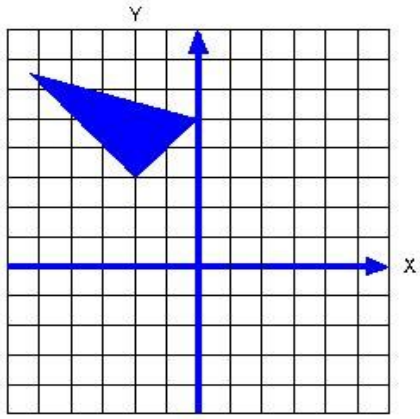
Quay một hình thì rắc rối hơn tịnh tiến hay co giãn, vì phải dùng đến phép toán phức tạp hơn, phải tính toán sin, cosin. Tuy nhiên ở đây ta chỉ áp dụng công thức quay, mà không tìm hiểu tại sau, làm gì..

Hình2.9 biểu diễn việc quay 45 độ một tam giác quanh gốc tọa độ.

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN





Giả sử cần quay một tam giác 45^0 như sau :

```
VERTEX triangleVerts[3]= { 20,50,50,50,20,100};
SHAPE shape1 = {3,triangleVerts};
```

Các công thức sau được áp dụng cho các vertex trong tam giác:

```
RotatedX = x*cos(angle) – y*sine(angle);
RotatedY = y*cos(angle) – x*sine(angle);
```

Để từ đó nhận được vertex sau:

```
VERTEX triangleVerts[3]= { -21,49,0,70,-56,84};
```

Chú ý rằng trong đó có hai tọa độ âm, do tam giác quay sang phần âm của trục x. Giá trị âm được chấp nhận trong tọa độ đề các, nhưng không thể biểu diễn lên màn hình. Để hiển thị tam giác cần tịnh tiến nó sang phần dương của trục x . Toàn bộ phép quay và tịnh tiến sẽ như sau:

```
VERTEX triangleVerts[3]= { 20,50,50,50,20,100};
SHAPE shape1 = {3,triangleVerts};
Rotate(shape1,45);
Translate(shape1,100,0);
DrawShape(shape1);
Void Rotate(SHAPE& shape,int degrees)
{
   int rotatedX,rotatedY;
   double radians = 6.283185308/(360.0/degrees);
   double c = cos(radians);
   double c = sin(radians);
      for(int x =0;x< shape.numVerts;++x)
      {
         rotatedX = (int) (shape.verticse[x]x*c –
                                 shape.vertices[x].y*s);
         rotatedY = (int) (shape.verticse[x]y*c –
                                 shape.vertices[x].x*s);
         shape.vertices[x].x = rotatedX;
```

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

```
            shape.vertices[x].y = rotatedY;
        }
    }

    Void Translate(SHAPE& shape1,int xtrans,int yTrans)
    {
        for(int x =0;x< shape.numVerts;++x)
        {
            shape.vertices[x].x += xTrans;
            shape.vertices[x].y += yTrans;
        }
    }

    Void DrawShape(SHAPE& shape1)
    {
        CClientDC dc (this);
        Int newX,newY,startX,startY;
        RECT clientRect;
            GetClientRect(&clientRect);
        Int maxY = clientRect.bottom;
        For (int x =0 ;x<shape1.numVerts;++x)
        {
            newX shape1.vertices[x].x;
            newY = maxY – shape1.vertices[x].y;
            if(x = = 0)
            {
                dc.MoveTo(newX,newY);
                startX = newX;
                startY = newY;
            }
            else dc.LineTo(newX,newY);
        }
        dc.LineTo(startX,startY);
}
```

Hàm Rotate( ) nhận tham số là tham chiếu đến cấu trúc SHAPE và góc quay(độ). Công việc đầu tiên của nó là đổi độ sang radian, do radian là đơn vị đo mà các hàm sin( ), và hàm cosin( ) của visual c++ yêu cầu. Với góc quay dương thì hình sẽ quay ngược chiều kim đồng hồ, và ngược lại

Cũng giống như các hàm Translate( ) và Scale( ), hàm Rotate( ) sử dụng vòng lặp for để tính tọa độ (x,y) cho từng vertex.

2.4. Sử Dụng Ma Trận Trong Các Phép Biến Hình:

Một chương trình đồ họa thường thực hiện tất cả các phép tính toán trên vertex của đối tượng trước khi thể hiện đối tượng ra màn hình. Có thể gọi các hàm Translate( ), Scale( ), và Rotate( ) cho các phép tính này. Tuy nhiên, việc thực hiện nhiều phép tính

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

như vậy cho nhiều vertex sẽ tiêu phí thời gian. Đó cũng là lý do mà ma trận thường được dùng cho các phép biến hình.

Một ma trận đơn giãn là một bảng số sắp xếp trên các hàng và cột. Giống như mảng trong lập trình, kích thước ma trận được định nghĩa trên số hàng và cột mà nó có. ví dụ ma trận 3*4 là ma trận có 3 hàng và 4 cột:

4 7 2 4
4 6 7 3
4 5 2 2

Trong chương trình ma trận được trình bài như sau:

```
Int matrix[3][4]=
   {
        4,7,2,4,
        4.6.7.3.
        4,5,2,2,
   };
```

Thuận lợi của ma trận trong lặp trình đồ họa là có thể trình bày nhiều phép biến hình với một ma trận đơn. Nghĩa là mọi ma trận đơn chứa mọi giá trị cần thiết để đồng thời dùng trong tịnh tiến, co giãn và quay một hình. Để thực hiện điều đó thì cần biết đặt giá trị nào vào ma trận và cần biết phép nhân ma trận.

2.4.1.Các Kiểu Dữ Liệu Dùng Trong Đồ Họa 2-D:

Đồ họa 2-D dùng trong ma trận : 1*3 và 3*3.

Ma trận 1*3 là ma trận đặc biệt , gọi là vector. Vector chứa các giá trị x,y và w để thể hiện một vertex. Vậy kiểu dữ liệu vertex như sau :

```
Typedef struct vector
{
      int x,y,w;
        }VECTOR;
```

Trong đó ,w là giá trị thường dùng để đơn giãn hóa thao tác ma trận, mặt dù OpenGL đôi khi sử dụng đặt biệt giá trị này.Trong hầu hết các trường hợp, w bằng 1. Nghĩa là vertex được biểu diễn bởi vector có dạng: x, y, z Ma trận 3*3 chứa các giá trị cần thiết cho các phép biến hình một vertex (được biểu diễn bằng kiểu dữ liệu vector, tức cũng là một ma trận) Kiểu dữ liệu ma trận 3*3 như sau:

```
Typedef double MATRIX3*3[3][3];
```

2.4.2.Các Ma Trận Biến Hình :

Bước đầu tiên là cung cấp các giá trị thích hợp cho ma trận. Giá trị được dùng và vị trí của nó trong ma trận phụ thuộc kiểu biến hình.

Ma trận dùng cho phép tịnh tiến có dạng:

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

1	0	0
0	1	0
xTrrans	yTrans	1

Với các biến xTrans và yTrans tương ứng là số đơn vị theo chiều ngang và dọc dùng cho phép tịnh tiến, (tương tự như khi dùng công thức tịnh tiến). Trong chương trình ma trận này được khởi tạo như sau:

```
MATRIX3*3 m;
    m[0][0] = 1.0;          m[0][1] = 0.0;          m[0][2] = 0.0;
    m[1][0] = 0.0;          m[1][1] = 1.0;          m[1][2] = 0.0;
    m[2][0] = xTrans;       m[2][1] = yTrans;       m[2][2] = 1.0;
```

- Ma trận dùng cho phép co giãn có dạng:

XScaleFactor	0	0
0	yScaleFactor	0
0	0	1

Với các biến xScaleFactor và yScaleFactor tương ứng là độ co giãn theo chiều ngang và dọc. Trong chương trình, ma trận này được khởi tạo như sau :

```
MATRIX3*3 m;

m[0][0] = xScaleFactor;   m[0][1] = 0.0;              m[0][2] = 0.0;
m[1][0] = 0.0;            m[1][1] = xScaleFactor;     m[1][2] = 0.0;
m[2][0] = 0.0;            m[2][1] = 0.0;              m[2][2] = 1.0;
```

- Cuối cùng ma trận dùng cho phép quay có dạng :

Cos(radians)	sin(radians)	0
-sin(radians)	cos(radians)	0
0	0	1

Với biến radian là góc quay (đơn vị radian). Trong chương trình, ma trận này được khởi tạo như sau :

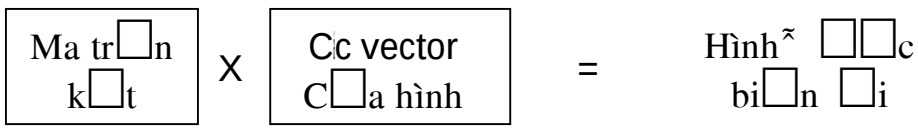
```
MATRIX3*3 m;
m[0][0] = cos(radians);     m[0][1] =sin(radians);      m[0][2] = 0.2;
m[1][0] = -sin(radians);    m[1][1] = cos(radians);     m[1][2] = 0.0 ;
m[2][0] = 0.0;              m[2][1] = 0.0;              m[2][2] = 1.0;
```

2.4.3.Kết Hợp Các Phép Biến Hình :

Ở mục trước, từng phép biến hình được biểu diễn bằng các các ma trận riêng biệt. Tuy nhiên, như đã nói, có thể kết hợp nhiều phép biến hình trong một ma trận.

Để kết hợp hai phép biến hình, ta nhân hai ma trận của chúng với nhau Kết hợp phép biến hình tiếp theo bằng cách nhân ma trận của nó với ma trận nhận được ở phép kết hợp trước. Hình 2.10 biểu diễn các phép biến hình

Một cách nhìn khác được biểu diễn ở hình 2.12, nhưng kết quả của từng phép kết hợp các ma trận không được thể hiện .



vord? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

Để nhân hai ma trận với nhau, thì số cột trước phải bằng số hàng của ma trận sau. Như vậy, ta có thể nhân hai ma trận 1X3 vớ ma trận 3X3, hay nhân hai ma trận 3X3 với nhau .

Hàm sau nhân hai ma trận 3X3:

```
Void MultMatrix(MATRIX3X3& product,
                        MATRIX3X3& matrix1, MATRIX3X3& matrix2)
{
        for (int x = 0; x < 3; ++y)
        {
                double sum = 0;
                        for (int z = 0; z < 3; ++z)
                           sum + = matrix1[x][z]*matrix2[z][y];
                           product[x][y] = sum;
        }
 }
```

Các tham số của hàm trên bao gồm một ma trận chứa kết quả nhân, và hai ma trận tham gia phép nhân .Ví dụ về cách sử dụng hàm như sau:

```
MATRIX3X m1,m2,m3;

   m[0][0] = 1.0;      m[0][1] = 0.0;      m[0][2] = 0.0;
   m[1][0] = 0.0;      m[1][1] = 1.0;      m[1][2] = 0.0;
   m[2][0] = 0.0;      m[2][1] = 0.0;      m[2][2] = 1.0;

   m[0][0] = 9.0;      m[0][1] = 8.0;      m[0][2] = 7.0;
   m[1][0] = 6.0;      m[1][1] = 5.0;      m[1][2] = 4.0;
   m[2][0] = 3.0;      m[2][1] = 2.0;      m[2][2] = 3.0;
        MultiMatrix(m3,m1,m2);
```

Đoạn mã khai báo ba ma trận 3X3 là m1, m2, m3. Sau đó khởi tạo m1, m2, rồi gọi hàm MulMatrix() để nhân m1 với m2, và lưu trữ kết quả trong m3. Do m1 là ma trận đơn vị, kết quả chứa trong m3 sẽ có cùng giá trị như trong m2.

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

Ma trận đơn vị i cấp n được định nghĩa là loại ma trận đường chéo nXm chiều, với các ìn tử đường chéo hình đơn vị:

$$I \equiv [\delta_{ik}], \quad \text{với } \delta_{ik} = 0 \text{ nếu } i \neq k;$$

$$\delta_{ik} = 1 \text{ nếu } i = k;$$

Trong lập trình đồ họa, ma trận đơn vị thường được dùng để khởi tạo ma trận chính là ma trận dùng kết hợp các phép biến hình. Việc khởi tạo như vậy sẽ chắc chắn không tồn tại giá trị lạ trong ma trận .

2.4.4.Thực Hiện Biến Hình:

Sau khi kết hợp các phép biến hình, ta nhận được một ma trận chính chứa các giá trị chính xác cần thiết để đồng thời tịnh tiến, co giãn và quay hình. Công việc biến hình bây giờ chỉ đơn giản là nhân ma trận chính với các vector của hình (xem hình 2.10,2.11). Và như vậy cần một hàm không chỉ nhân vector 1X3 với ma trận 3X3, mà nhân ma trận với toàn bộ danh sách vector:

```
Void Transform(SHAPE& shape, MATRIX3X3& m)
{
      int transformedX,transformY;
      for (int x= 0; x <shape.numverts; ++x)
      {
         transformX = (int) (shape.vertices[x].x*m[0][0] +
                              shape.vertices[x].y*m[1][0] + m[2][0]);

      transformY = (int) (shape.vertices[x].y*m[0][0] +
                              shape.vertices[x].x*m[1][0] + m[2][0]);

      }
}
```

Hàm trên có các đối số là tham chiếu đến các cấu trúc SHAPE và tham chiếu đến mãng MATRIX3X3.

2.4.5.Một Số Hàm Ma Trận Tiện Ích:

Có hai hàm ma trận tiện ích thường được dùng để làm dễ dàng hơn việc sử dụng ma trận trong lập trình. Chúng gồm:

- Hàm khởi tạo ma trận theo ma trận đơn vị:

```
MATRIX3X m1,m2,m3;
  m[0][0] = 1;    m[0][1] = 0;      m[0][2] = 0;
  m[1][0] = 0;    m[1][1] = 1;      m[1][2] = 0;
  m[2][0] = 0;    m[2][1] = 0;      m[2][2] = 1;
```

- Hàm sao chép ma trận:

```
Void intMatrix(MATRIX3X3& m)
{
for (int i=0; i <3; ++i)
   for (int j=0; j <3; ++j)
```

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

```
            dst[i][j] = src [i][j];
        }
```

2.4.6.Các Hàm Biến Hình Dùng Ma Trận:

- Hàm tịnh tiến :

```
Void Translate(MATRIX3X3& m,int xTrans,int yTrans)
{
MATRIX3X m1,m2;

  m[0][0] = 1;          m[0][1] = 0;          m[0][2] = 0;
  m[1][0] = 0;          m[1][1] = 1;          m[1][2] = 0;
  m[2][0] = xTrans;     m[2][1] =yTrans0;     m[2][2] = 1;

        MultMatrix(m2,m1,m);
        CopyMatrix(m,m2);
}
```

Hàm có đối số là tham chiếu đến ma trận chính (chứa trạng thái hiện tại của phép biến hình) và các giá trị tịnh tiến x, y. Đầu tiên, nó tạo ma trận tịnh tiến ; Rồi nhân ma trận tịnh tiến với ma trận chính, lưu kết quả trong ma trận cục bộ m2; Cuối cùng sao chép m2 trở lại ma trận chính.

- Hàm co giãn:

```
Void Scale(MATRIX3X3& m,double xScale,double yScale)
{
MATRIX3X m1,m2;

  m[0][0] = xScale;   m[0][1] = 0;          m[0][2] = 0;
  m[1][0] = 0;        m[1][1] = yScale;     m[1][2] = 0;
  m[2][0] = 0;        m[2][1] =0;           m[2][2] = 1;

        MultMatrix(m2,m1,m);
        CopyMatrix(m,m2);

        }
```

- Hàm quay:

```
        Void Rotate(MATRIX3X3& m,int degrees)
        {
                MATRIX3X m1,m2;
                If (degrees = = 0) return;
                Double radians = 6.283185308/(360.0/ degrees);
                Double c = cos(radians);
                Double s = sin(radians);
                m[0][0] = c;      m[0][1] = s;        m[0][2] = 0;
                m[1][0] = -s;     m[1][1] = c;        m[1][2] = 0;
                m[2][0] = 0;      m[2][1] =0;         m[2][2] = 1;
```

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

```
MultMatrix(m2,m1,m);
CopyMatrix(m,m2);

}
```

Hàm quay có các đối số là tham chiếu đến ma trận chính và góc quay(độ). Đầu tiên nó kiểm tra góc quay có bằng không hay không. Nếu góc quay bằng không, hàm kết thúc để loại trừ, lỗi chia cho 0. Nếu khác không, hàm đổi độ ra radians và tính sin, cosin. Tiếp theo, Rotate khởi tạo ma trận quay, nhân nó với ma trận chính, lưu kết quả vào ma trận cục bộ m2. Cuối cùng, sao chép m2 trở lại ma trận chính.

Bây giờ ta đã có một bộ hàm dùng ma trận. Hãy xét cách dùng chúng trong biến hình ở đoạn mã sau:

```
MATRIX3X3 m;
IntMatrix(m);
Translate(m,10,15);
Scale(m,0.5,0.5);
Rotate(m,45);
Transform(shape1,m);
DrawShape(shape1);
```

Trước tiên đoạn mã khai báo một ma trận biến hình 3X3 là m. Sau đó gọi hàm IntMatrix() để khởi tạo m theo ma trận đơn vị. M sẽ là:

```
1.0000000000000    0.0000000000000    0.0000000000000
0.0000000000000    1.0000000000000    0.0000000000000
0.0000000000000    0.0000000000000    1.0000000000000
```

Lời gọi hàm Translate( ) kết hợp m với ma trận tịnh tiến chứa các giá trị 10 và

15. Matrận biến hình m sẽ là:

```
1.0000000000000    0.0000000000000    0.0000000000000
0.0000000000000    1.0000000000000    0.0000000000000
10.0000000000000   15.0000000000000    1.0000000000000
```

Sau khi gọi hàm Scale( ), m chứa các giá trị của phép tịnh tiến và co giãn:

```
1.0000000000000    0.0000000000000    0.0000000000000
0.0000000000000    0.5000000000000    0.0000000000000
10.0000000000000   15.0000000000000   1.0000000000000
```

Cuối cùng sau lời gọi hàm Rotate( ), m chứa đầy đủ các phép tịnh tiến và quay:

```
0.35355339055702    0.3535533906295    0.0000000000000
-o.35355339062953   0.35355339055702    0.0000000000000
10.000000000000     15.000000000000     1.0000000000000
```

Lời gọi hàm Translate() áp dụng ma trận cho một vertex của shape1. Sau đó Drawshape() vẽ hình đã biến hình lên màn hình.

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

Chương 3:Đồ Họa Ba Chiều GDI

ong chương này, chúng ta sẽ tìm hiểu:
Hệ tọa độ đề các ba chiều.
- Định nghĩa vertex cho đối tượng 3-D.
- Phép chiếu song song và phép chiếu phối cảnh.
- Chuyển động đối tượng 3-D.

3.1.Hệ Tọa Độ Ba Chiều:

Một đối tượng ba chiều thì không thể biểu diễn trong hệ tọa độ hai chiều chỉ gồm hai trục x và y. Để tạo hệ tọa độ mới, ta chỉ đơn giản thêm một trục z vào mặt phẳng đề các, để biến đổi thành hình khối. Mặt dù các trục x, y, z có thể hướng theo mọi phương, ta qui định trục x theo chiều ngang, trục y theo chiều đứng, trục z theo phương đi vào và ra màn hình. Đó là hướng trục logic cho nhiều chương trình, xuất phát từ việc trục x và y chạy theo các phương tương ứng với tọa độ màn hình.

3.2 .Định Nghĩa Đối Tượng Ba Chiều:

Việc tạo một hình hai chiều chỉ đơn giản là định nghĩa tập vertex của nó, rồi nói các vertex với nhau. Nhưng đối với đối tượng 3-D, vấn đề có phứp tạp hơn, bởi vì số lượng vertex là nhiều hơn, đòi hỏi việc xác định việc nói các vertex để hình thành đúng đối tượng yêu cầu. Do đó, đối với chúng, không chỉ định nghĩa các vertex, mà còn phải định nghĩa các cạnh. Một đối tượng 3-D được xây dựng trên các vertex và cạnh được gọi là một mô hình khung lưới (wireframe model).

Để định nghĩa một mô hình khung lưới cần danh sách các vertex và cạnh. Do đó, hiển nhiên là cần một số kiểu dữ liệu mới.

-Trước tiên là kiểu dữ liệu định nghĩa vertex của đối tượng 3-D với ba tọa độ x, y, z như sau:

```
Typedef struct vertex
{
 int x,y,z,w;
}VERTEX;
```

-Để vẽ một cạnh, ta cần biết vertex khởi đầu và vertex kết thúc, sau đó nối chúng lại với nhau. Kiểu dữ liệu định nghĩa cạnh như sau:

```
Typedef struct edge
{
   UINT vertex1,vertex2;
      }EDGE;
```

Với vertex1 là điểm xuất phát cạnh, và vertex2 cuối.

- Kiểu dữ liệu của mô hình khung lưới như sau:

```
Typedef struct model
      {
      UINE numverts;
      VERTEX* vertices;
      UINE numEdges;
      EDGE*edges;
```

KILOBOOKS.COM

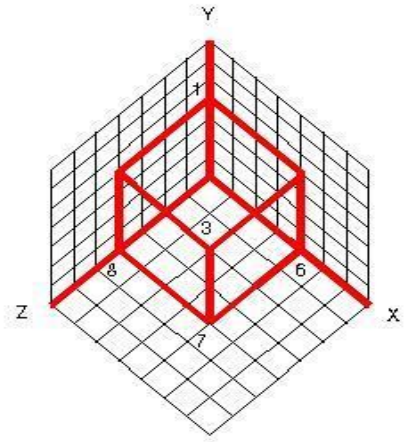
Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

}MODEL;

Như vậy, cấu trúc MODEL bao gồm hai số nguyên không dấu chứa vertex và cạnh tượng, một con trỏ chỉ đến kiểu dữ liệu VERTEX chứa địa chỉ của danh sách vertex, và mộ con trỏ chỉ đến kiểu dữ liệu EDGE chứa địa chỉ của danh sách cạnh.

Hình 3.1Hình khối:



Sử dụng các kiểu dữ liệu trên cho khối vuông, với số thứ tự các đỉnh như hình 3.1. Tọa độ các đỉnh lần lược là (0,4,0), (4,4,0), (4,4,-4), (0,4,-4), (0,0,0,), (4,0,0), (4,0,-4),(0,0,-4).

Danh sách vertex sẽ như sau :

```
VERTEX cubeVerts[8] =
{
    0,4,0,
    4,4,0,
    4,4,-4,
    0,4,-4,
    0,0,0,
    4,0,0,
    4,0,-4,
    0,0,-4,
}
```

Chú ý rằng tọa độ z cửa đối tượng đã cho bằng 0 hoặc âm, do chiều tăng tọa độ z hướng ra khỏi màn hình .

Các tọa độ dùng định nghĩa một mô hình khung lưới, như trong cấu trúc cubeverts ở trên, thì được xem là các tọa độ cục bộ. Tọa độ cục bộ trở thành tọa độ thế giới khi chương trình sử dụng các phép tịnh tiến co giãn hoặc quay về biến mô hình. Cuối cùng, chương trình đồ họa biến đổi các tọa độ thế giới thành tọa độ màn hình để có thể thể hiện mô hình lên màn hình.

Danh sách cạnh của khối vuông trên như sau :

```
EDGE cubeedges[12]=
{
        1,2,
        2,3,
        3,4,
        4,1,
        5,6,
        6,7,
        7,8,
        8,5,
        5,1,
        6,2,
```

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

```
        7,3,
        8,4,
    };
```

Bốn cặp tọa độ đầu tạo thành đỉnh hình khối. Bốn cặp tọa độ tiếp theo hình thành đáy hình khối. Bốn cặp tạo độ cuối cùng nối đỉnh và đáy hình khối với nhau.

Với danh sách vertex và danh sách cạnh đã có, khối vuông trên hình 3.1 được định nghĩa như sau:

```
MODEL cube = {8,cubeVerts,12,cubeEdges};
```

3.3.Các Phương Pháp Thể Hiện 3-D Lên Màn Hình:

Một đối tượng 3-D được xác định bởi các tọa độ x,y,z,nhưng màn hình thì được biểu diễn chỉ với các tọa độ x, và y. Vậy bằng cách nào để có thể biểu diễn một đối tượng 3-D lên màn hình ?

Có hai phương pháp để thực hiện vấn đề trên là phép chiếu song song và phép chiếu phối cảnh.

3.3.1.Phép Chiếu Song Song:

Với phép chiếu song song, một đối tượng 3-D được thể hiện lên màn hình bằng cách bỏ qua các tọa độ z. Kết quả là một hình 2-D đơn giãn. Như vậy, với trường hợp khối vuông, hình ảnh thể hiện trên màn hình chỉ là một hình vuông. Hàm vẽ khối vuông sử dụng phép chiếu song song như sau :

```
Void DrawModel (CDC* pDC, MODEL& mode)
{
    int newX,int newY;
    RECT clientRect;
    GetClientRect (&clientRect);
    Int maxY = clientRect.bottom;
    For (UINT I=0; I<model.numEdges; ++i)
    {
        UINT vertNum = model.edges[i].vertex1;
        NewX = model.vertices[vertNum -1].x;
        NewY = maxY – model.vertices[vertNum -1].y –1;
            pDC ->MoveTo(newX,newY);

        vertNum = model.edges[i].vertex2;
        newX = model.vertices[vertNum –1].x;
    newY = maxY – model.vertices[vertNum -1].y –1;
        pDC -> LineTo(newX,newY);
      }
}
```

Hàm trên gọi GetClientRect() để xác định tọa độ Y lớn nhất cửa cửa sổ. Sau đó dùng vòng lập for để vẽ các cạnh của đối tượng. Bên trong vòng lặp, hàm tính số hiệu của vertex dùng làm điểm đầu của cạnh, rồi dùng số đó để tìm các tọa độ X, Y của vertex. Các tọa độ này được dùng trong lời gọi đến MoveTo() để xác định vị trí điểm đầu. Các tọa độ X, Y, của điểm thứ hai được tìm bằng cách tương tự, sau đó dùng trong hàm gọi đến LineTo() để vẽ cạnh:

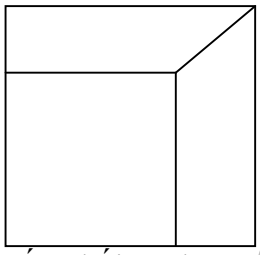
KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

3.3.2 .Phép Chiếu Phối Cảnh:

Phép chiếu song song không thể sử dụng, nhưng nó không mang lại một kết quả ı ý, trừ khi dùng cho một chương trình phác thảo. Một yếu tố then chốt của hình ảnh 3-D là chiều sâu ảnh. Ao giác mà được tạo bằng cách dùng phép chiếu phối cảnh, làm cho đối tượng ở xa có vẽ nhỏ hơn đối tượng ở gần . Ví dụ hình 3.2 biểu diễn hình chiếu phối cảnh của khối vuông, khi quan sát từ mặt trước, góc dưới trái.

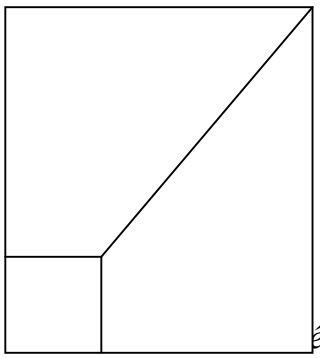


Hình 3.2 vẽ khối vuông với phép chiếu phối cảnh

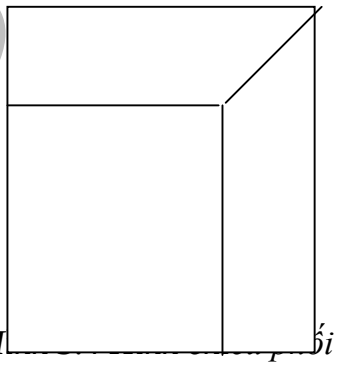
Các công thức dùng cho phép chiếu phối cảnh :

```
Double t  1.0/(1.0 – zCoord / eye);
PerspX  = (int) ( xCoord * t);
PerspY  = (int) ( yCoord * t);
```

Biến t chứa giá trị co giãn các tọa độ X và Y trên cơ sở tọa độ Z . Tọa độ Z càng nhỏ (càng xa gốc tọa độ), thì tác dụng co giãn càng lớn . Các biến xCoord, yCoord và zCoord chứa các tọa độ vertex. Các biến perspX, prspY chứa tọa độ màn hình của vertex sau phép chiếu phối cảnh. Biến eye xác định mức độ tác động của phép chiếu trên mô hình. Khi quan sát đối tượng càng gần (eye càng nhỏ) thì hiệu quả phép chiếu càng lớn .



ếu phối cảnh khi quan sát gần(eye nhỏ)



Hi ối cảnh khi quan sát xa (eye lớn)

Hàm gán phép chiếu phối cảnh cho các vertex của một mô hình khung lưới như sau :

```
Void PerspProject (MODEL& model, double eye)
  {
        for (UINT I =0; I<model.numVerts; ++i)
      {
          int xCoor = model.vertices[i].x;
              int yCoor = model.vertices[i].y;
              int zCoor = model.vertices[i].z;
              double t = 1.0/(1.0 – zCoor/ eye);
              model.vertices[i].x = (int) (xCoor *t);
              model.vertices[i].y = (int) (yCoor *t);
```

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

```
        }
    }
```

Hàm trên có các đối số là tham chiếu đến các cấu trúc MODEL và giá trị eye dùng tính toán t. PerspProject()dùng vòng lặp for để gán phép chiếu phối cảnh cho từng vertex của mô hình.

Đoạn mã ví dụ sau cho thấy các định nghĩa mô hình, gán phép chiếu và vẽ đối tượng:

```
VERTEX cubeVerts[8] =
    {0,     100,    0,      1,
     100,   100,    0,      1,
     100,   100,    -100,   1,
     0,     100,    -100,   1,
     0,     0,      0,      1,
     100,   0,      0,      1,
     100,   0,      -100,   1,
     0,     0,      -100,   1];
EDGE cubeEdges[12] =
  {  1,2,
     2,3,
     3,4,
     4,1,
     5,6,
     6,7,
     7,8,
     8,5,
     5,1,
     6,2,
     7,3,
     8,4};
    MODEL cube;
    Cube.numVerts = 8;
    Cube.vertices = cubeVerts;
    Cube. numEdges   = 12;
    PerspProject (cube, 200);
    DrawModel (pDC, cube);
```

3.4.Biến Hình Đối Tượng 3-D:

Như vậy vẽ một đối tượng 3-D thì không khó. Nhưng với góc nhìn ở phía trước, góc dưới trái như hình 3.2 thì cũng không đẹp đẽ gì. Để có một góc nhìn khác, ta cần biến hình đối tượng .

Việc biến hình đối tượng 3-D không khác mấy so với biến hình đối tượng 2-D, chỉ đơn giản là chỉ xét thêm tọa độ z . Ma trận biến hình dùng trong trường hợp này có kích thước 4x4, với kiểu dữ liệu như sau:

```
Typedef double MATRIX4x4[4][4];
```

Các hàm InitMtrix(), CopyMatrix(), MultMatrix() bây giờ như sau:

Khởi tạo ma trận:

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

```
Void InitMatrix(MATRIX4x4& m)
{
        m[0][0] = 1;    m[0][1] = 0;    m[0][2] = 0;    m[0][3] = 0;
        m[1][0] = 0;    m[1][1] = 1;    m[1][2] = 0;    m[1][3] = 0;
        m[2][0] = 0;    m[2][1] =0;     m[2][2] = 1;    m[2][3] = 0;
        m[3][0] = 0;    m[3][1] = 0;    m[3][2] = 0;    m[3]3[] = 1;
}
```

Hàm sao chép ma trận :

```
Void CopyMatrix(MATRIX4x4& src)
{
      for (int =0; I<4; ++i)
          for(int j =0; j<4; ++j)
          dst[i][j];
      }
```

Hàm nhân ma trận:

```
Void    MultMatrix(MATRIX4x4&product,MATRIX4x4&
matrix1,MATRIX4x4& matrix2)
{
   for (int x=0; x<4; ++x)
      for(int y=0; y<4; ++y)
      {
   double sum =0;
   for (intz 0; z<4; ++z)
      sum +=matrix[x][z]* matrix2[z][y];
   product[x][y] = sum;
}
}
```

3.4.1.Phép Tịnh Tiến 3-D:

Ma trận tịnh tiến:

```
1          0          0          0
0          1          0          0
0          0          1          0
xTrans     yTrans     zTrans     1
```

Hàm tịnh tiến:

```
Void Translate(MATRIX4x4& m,int xTrans,yTrans,zTrans)
{
   m[0][0] = 1;          m[0][1] = 0;          m[0][2] = 0;          m[0][3] = 0;
   m[1][0] = 0;          m[1][1] = 1;          m[1][2] = 0;          m[1][3] = 0;
   m[2][0] = 0;          m[2][1] =0;           m[2][2] = 1;          m[2][3] = 0;
   m[3][0] = xTrans;     m[3][1] = yTrans;     m[3][2] = zTrans;     m[3]3[] = 1;
   MultMatrix(m2,m1,m);
   CopyMatrix(m,m2);

}
```

3.4.2 .Phép Co Giãn:

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

Ma trận co giãn:

XFactor	0	0	0
0	YFactor	0	0
0	0	ZFactor	0
0	yTrans	zTrans	1

Hàm co giãn:

```
Void Scale(MATRIX4x4& m,double xScale,yScale,zScale)
{  MATRIX4x4 m1,m2;
   m[0][0] = xScale;   m[0][1] = 0;        m[0][2] = 0;        m[0][3] = 0;
   m[1][0] = 0;        m[1][1] = yScale;   m[1][2] = 0;        m[1][3] = 0;
   m[2][0] = 0;        m[2][1] =0;         m[2][2] = zScale;   m[2][3] = 0;
   m[3][0] = 0;        m[3][1] = 0;        m[3][2] = 0;        m[3]3[] = 1;
   MultMatrix(m2,m1,m);
   CopyMatrix(m,m2);

   }
```

3.4.3. Phép Quay 3-D:

Khi quay một hình phẳng, thực chất là ta quay nó quanh trục z. Đối với đối tượng 3-D, ta có thể quay nó quanh 3 trục x, y, z. Thế nhưng không có một ma trận đơn giãn có thể thực hiện 3 phép quay cùng một lúc. Thay vào đó, mỗi phép quay coi như một phép biến hình riêng biệt.

a) Quay Quanh Trục Z:

Ma trận quay quanh trục z:

Cos(radians)	sin(radians)	0	0
-sin(radians)	cos(radians)	0	0
0	0	1	0
0	0	0	1

Hàm quay quanh trục z:

```
Void Rotate(MATRIX4x4& m,int zAngle)
{
      if (zAngle = =0) return;
        double radians = 6.383185308 / (360.0 / zAngle);
       double c = cos(radians);
       double s = sin(radians);

      m[0][0] = c;    m[0][1] = s;     m[0][2] = 0;     m[0][3] = 0;
      m[1][0] = -s;m[1][1] = c;     m[1][2] = 0;     m[1][3] = 0;
      m[2][0] = 0;    m[2][1] = 0;     m[2][2] = 1;     m[2][3] = 0;
       m[3][0] = 0;    m[3][1] = 0;     m[3][2] = 0;     m[3]3[] = 1;

      MultMatrix(m2,m1,m);
      CopyMatrix(m,m2);
 }
```

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

Quay Quanh Trục X:

trận quay quanh trục x:

1	0	0	0
0	cos(radians)	sin(radians)	0
0	-sin(radians)	cos(radians)	0
0	0	0	1

Hàm quay quanh trục X:

```
Void Rotate(MATRIX4x4& m,int zAngle)
   {
      if (zAngle = =0) return;
         double radians = 6.383185308 / (360.0 / zAngle);
         double c = cos(radians);
         double s = sin(radians);
              m[0][0] = 1;    m[0][1] = 0;     m[0][2] = 0;     m[0][3] = 0;
              m[1][0] = 0;    m[1][1] = c;     m[1][2] = s;     m[1][3] = 0;
              m[2][0] = 0;    m[2][1] =-s;     m[2][2] = c;     m[2][3] = 0;
              m[3][0] = 0;    m[3][1] = 0;     m[3][2] = 0;     m[3]3[] = 1;

         MultMatrix(m2,m1,m);
         CopyMatrix(m,m2);
       }
```

c) Quay Quanh Trục Y:

Ma trận quay quanh trục y:

cos(radians)	0	-sin(radians)	0
0	1	0	0
sin(radians)	0	cos(radians)	0
0	0	0	1

Hàm quay quanh trục Y:

```
   Void Rotate(MATRIX4x4& m,int zAngle)
   {
      if (zAngle = =0) return;
         double radians = 6.383185308 / (360.0 / zAngle);
         double c = cos(radians);
         double s = sin(radians);
          m[0][0] = c;     m[0][1] = 0;     m[0][2] = -s;     m[0][3] = 0;
          m[1][0] = 0;     m[1][1] = 1;     m[1][2] = 0;      m[1][3] = 0;
          m[2][0] = s;     m[2][1] = 0;     m[2][2] = c;      m[2][3] = 0;
          m[3][0] = 0;     m[3][1] = 0;     m[3][2] = 0;      m[3]3[] = 1;

      MultMatrix(m2,m1,m);
      CopyMatrix(m,m2);
```

KILOBOOKS.COM

Chương 4: Chương Trình OpenGL Tối Thiểu

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

Chương này trình bày cách khởi tạo ứng dụng OpenGL và cách kết hợp OpenGL) một chương trình MFC :

Các yêu cầu của một chương trình OpenGL tối thiểu .

_ Các kiểu dữ liệu và cú pháp của lệnh OpenGL.

_ Cách tạo ngữ cảnh biểu diễn và định dạng điểm vẽ.

_ Cách thiết lập một chương trình MFC cho OpenGL .

4.1.Các Kiểu Dữ Liệu OpenGL :

OpenGL định nghĩa kiểu dữ liệu để dễ dàng thao tác trong chương trình . Các kiểu dữ liệu này được trình bày trong bảng 4.1.

Bảng 4.1 : Các kiểu dữ liệu OpenGL & hậu tố tương ứng của lệnh

Kiểu dữ liệu OpenGL	Kiểu dữ liệu tương ứng trong C	Ý nghĩa
Glbyte	Signed char	Số nguyên 8 bit
Glshort	Short	Số nguyên 16 bit
GLint,GLsizei	Long	Số nguyên 16 bit
Glfloat,GLclampf	Float	Dấu chấm động 32 bit
Gldouble,Glclampd	Double	Dấu chấm động 64 bit
Glubyte,Glboolean	Unsigned char	Char không dấu 8 bit
Glushort	Unsigned short	Số nguyên không dấu 16 bit
Gluint, GLenum, Glbitfield	Unsigned long	Số nguyên không dấu 32 bit
Glvoid	Void	Hàm không đối số hoặc không trả về một giá trị
HGLRC	HGDIOBJ	Handle của đối tượng GDI

4.2. Ngữ Cảnh Biểu Diễn :

Mọi chương trình trên Windows đều phải sử dụng ngữ cảnh dụng cụ (DC,Device Context), là cấu trúc dữ liệu chứa đựng thông tin về cách mà dữ liệu đồ họa được thể hiện trên của sổ. Ngữ cảnh dụng cụ xác định màu sắc bút vẽ và cọ, kiểu vẽ, nội dung bảng màu, kiểu mapping (kiểu thể hiện các phần tử của không gian tọa độ nguồn lên không gian tọa độ đích), và các thuộc tính khác mà Windows cần biết để thể hiện thông tin đồ họa.

OpenGL cũng sử dụng ngữ cảnh dụng cụ như mọi chương trình Windows khác. Nhưng đồng thời phải sử dụng ngữ cảnh biểu diễn (RC,Rendering Context). Toàn bộ lệnh OpenGL đều phải qua ngữ cảnh biểu diễn. Mọi thread thực hiện lời gọi OpenGL phải có một ngữ cảnh biểu diễn hiện hành. Ngữ cảnh biểu diễn kết nối OpenGL vào hệ thống cửa sổ của Windows NT va Windows95. Ứng dụng sẽ xác định ngữ cảnh dụng cụ khi nó tạo ngữ cảnh biểu diễn. Ngữ cảnh biểu diễn này phù hợp với thuật vẽ trên các ứng dụng đã chỉ định bởi ngữ cảnh dụng cụ. Đặc biệt là ngữ cảnh biểu diễn có cùng định dạng điểm vẽ (pixel format) như ngữ cảnh dụng cụ . Mặc dù vậy, Ngữ cảnh biểu diễn không phải là ngữ cảnh dụng cụ . Ngữ cảnh dụng cụ chứa các thông tin thích hợp đối với GDI của Windows

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com

' và Windows. Còn ngữ cảnh biểu diễn chứa các thông tin thích hợp đối với OpenGL . ɔt lời gọi GDI phải xác định rỏ ràng như một ngữ cảnh dụng cụ. Và một lời gọi enGL phải xác định rỏ ràng một ngữ cảnh biểu diễn .

Một thread thực hiện lời gọi OpenGL phải có một ngữ cảnh biểu diễn hiện hành. Nếu ứng dụng tiến hành lời gọi OpenGL từ một thread không có ngữ cảnh biểu diễn, thì lời gọi không hiệu lực. Thông thường, ứng dụng tạo ngữ cảnh biểu diễn, gán ngữ cảnh đó là ngữ cảnh biểu diễn hiện hành của thread, rồi gọi hàm OpenGL. Khi kết thúc việc gọi hàm OpenGL, ứng dụng tháo bỏ ngữ cảnh biểu diễn khỏi thread, và xóa nó. Một của sổ cùng một lúc có thể có nhiều ngữ cảnh biểu diễn, nhưng chỉ một trong số đó hiện hành và có tác dụng.

Một ngữ cảnh biểu diễn hiện hành thì có một ngữ cảnh dụng cụ kết hợp. Ngữ cảnh dụng cụ đó không cần phải là ngữ cảnh dụng cụ dùng khi ngữ cảnh biểu diễn được tạo ra, nhưng phải tham chiếu đến cùng một dụng cụ và có cùng định dạng điểm vẽ.

Một thread có duy nhất một ngữ cảnh biểu diễn hiện hành.Và một ngữ cảnh biểu diễn là hiện hành cho một thread duy nhất .

Mọi lời gọi hàm OpenGL tiềm ẩn một ngữ cảnh biểu diễn. Trong khi đối với ngữ cảnh dụng cụ, mỗi hàm GDI cần một handle, thì đối với ngữ cảnh biểu diễn, chỉ cần handle khi tạo ngữ cảnh biểu diễn hiện hành.

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

Bảng 4.2 : Các hàm WGL quản lý ngữ cảnh biểu diễn

ı hàm	Chức năng
WglCreateContext ()	Tạo một ngữ cảnh biểu diễn mới
WglDeleteContext ()	Xóa ngữ cảnh biểu diễn
WglGetCurrentContext ()	Trả handle về ngữ cảnh biểu diễn hiện hành
WglGetCurrentDC ()	Lấy handle của ngữ cảnh dụng cụ liên kết với ngữ cảnh biểu diễn hiện hành
WglMakeCurrent ()	Thiết lập ngữ cảnh biểu diễn hiện hành

4.3.Định dạng điểm vẽ :

Trước khi tạo ngữ cảnh biểu diễn, chương trình phải thiết lập định dạng điểm vẽ chứa các thuộc tính của bề mặt vẽ. Các thuộc tính này bao gồm chế độ màu là RGBA hay chỉ mục, bộ đệm điểm vẽ là đơn hay đôi, số lượng các bit dùng trong bộ đệm chiều sâu và bộ đệm stencil, cùng các thông tin đồ họa OpenGL khác .

Bảng 4.3 : Các hàm Win32 quản lý các định dạng điểm vẽ

Tên hàm	Chức năng
ChoosePixelFormat ()	Trả về một định dạng điểm vẽ phù hợp nhất với định dạng điểm vẻ yêu cầu
DescribePixelFormat ()	Lấy thông tin về định dạng điểm vẽ đã cho
GetPixelFormat ()	Lấy định dạng điểm vẽ của ngữ cảnh dụng cụ đã cho
SetPixelFormat ()	Thiết lập định dạng điểm vẽ của ngữ cảnh dụng cụ đã cho

4.3.1.Cấu Trúc PIXELFORMATDESCRIPTOR:

Mỗi thiết bị hiển thị OpenGL hỗ trợ một số định dạng điểm vẽ riêng. Dĩ nhiên, các định dạng điểm vẽ này dựa trên khả năng thiết bị. Các thuộc tính của một định dạng điểm vẽ riêng biệt thì được miêu tả bởi cấu trúc PIXELFORMATDESCRIPTOR, gồm 26 trường thông tin. Win32 định nghĩa cấu trúc này như sau :

```
Typedef struct tag PIXELFORMATDESCRIPTOR
{    WORD nSize;
       WORD nVersion;
       DWORD dwFlags;
       BYTE iPixelType;
       BYTE cColorBits;
       BYTE cRedBits;
       BYTE cRedShift;
       BYTE cGreenBits;
       BYTE cGreenShift;
       BYTE cBlueBits;
       BYTE cBlueShift;
```

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

```
    BYTE cAlphaBits;
    BYTE cAlphaShift;
    BYTE cAccumBits;
    BYTE cAccumRedBits;
    BYTE cAccumGreenBits;
    BYTE cAccumBlueBits;
    BYTE cAccumAlphaBits;
    BYTE cDepthBits;
    BYTE cStencilBits;
    BYTE cAuxBuffer;
    BYTE iLayerType;
    BYTE bServered;
    DWORD dwLayerMask;
    DWORD dwVisibleMask;
    DWORD dwDamageMask;
} PIXELFORMATDESCRIPTOR,* PPIXELFORMATDESCRIPTOR,
FAR * LPPIXELFORMATDESCRIPTOR
```

Khi thiết lập định dạng điểm vẽ của ngữ cảnh dụng cụ, thì cấu trúc PIXELFORMATDESCRIPTOR được điền đầy, và địa chỉ của nó được dùng làm đối số cho hàm SetPixelFormat () .

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

Bảng 4.4 :Các thành phần của cấu trúc PIXELFORMATDESCRIPTOR

ành phần	Mô tả
Nsize	Kích thước tính bằng byte của cấu trúc . có thể thiết lập bằng sizeof(PIXELFORMATDESCRIPTOR).
Nversion	Số phiên bản (version) của cấu trúc , hiện bằng 1.
DwFlags	Các cờ đặc tính cho định dạng điểm ảnh (bảng 4.6)
IpixelType	Mô tả dữ liệu màu của điểm vẽ : - PFD_TYPE_RGBA :đối với chế độ màu RGBA - PFD_TYPE_INDEX đối với chế độ màu chỉ mục
CColorBits	Số bit dùng thể hiện một màu . Nó quyết định số màu thể hiện . Ví dụ 8 bit thể hiện được 256 màu . Đối với chế độ màu RGBA, thì không tính đến alpha bitplane.Đối với chế độ màu chỉ mục , cColorBits xác định kích thước bộ đệm màu .
cRedBits	Số bit đỏ trong bộ đệm màu RGBA
cRedShift	Tổng độ dời các bitplane đỏ trong bộ đệm màu RGBA
cGreenBits	Số bit xanh lá trong bộ đệm màu RGBA
cGreenShift	Tổng độ dời các bitplane xanh lá trong bộ đệm màu RGBA
cBlueBits	Số bit xanh dương trong bộ đệm màu RGBA
cBlueShift	Tổng độ dời các bitplane xanh dương trong bộ đệm màu RGBA
cAlphaBits	Số bit alpha trong bộ đệm màu RGBA . Không dùng trên microsoft windows
cAlphaShift	Tổng độ dời các bitplane alpha trong bộ đệm màu RGBA. Không dùng trên microsoft windows
cAccumBits	Số bitplane trong bộ đệm tích lũy
cAccumRedBits	Số bit đỏ trên một pixel trong bộ đệm tích lũy
cAccumGreenBits	Số bit xanh lá trên một pixel trong bộ đệm tích lũy
cAccumBlueBits	Số bit xanh dương trên một pixel trong bộ đệm tích lũy
cAccumAlphaBits	Số bit alpha trên một pixel trong bộ đệm tích lũy
cDepthBits	Số bitplane trong bộ đệm chiều sâu
cStencilBits	Số bitplane trong bộ đệm stencil
cAuxBuffer	Số bộ đệm phụ .không dùng trên microsoft windows
iLayerType	Xác định kiểu lớp .trên Microsoft Windows là PFD_MAIN_PLANE . Ngoài ra các kiểu lớp khác là FPD_OVERLAY_PLANE và PFD_UNDERLAY_PLANE
bServered	Dùng dự trử .thường bằng 0
dwLayerMask	Hai trường hợp này đi đôi với nhau để xác định một lớp che một lớp .Trên Microsoft Windows không dùng lớp
dwVisibleMask	
dwDamageMask	Dùng khi nhiều hơn một định dạng điểm vẽ sử dụng chung bộ đệm khung. Nếu bitwise AND của các thành

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

	viên dwDamageMask của hai khung định dạng điểm vẽ khác 0 , thì chúng sử dụng chung một bộ đệm khung.

Bảng 4.5 :Các cờ đặc tính cho định dạng điểm ảnh

Cờ	Mô tả
PFD_DRAW_TO_BITMAP	Bộ đệm được dùng vẽ vào bitmap bộ nhớ
PFD_DRAW_TO_WINDOW	Bộ đệm được dùng vẽ vào cửa sổ trên màn hình hay thiết bị khác như máy in
PFD_DOUBLEBUFFER	Thiết lập khi dùng bộ đệm đôi.Không dùng cùng với cờ PFD_SUPPORT_GDI.
PFD_GENERIC_FORMAT	Thiết lập khi chọn định dạng điểm vẽ tổng quát ,là định dạng điểm ảnh được hổ trợ bởi phần cứng hay trình điều khiển thiết bị
PFD_NEED_PALETTE	Thiết lập khi dùng bảng màu logic
PFD_NEED_SYSTEM_PALETTE	Thiết lập khi hệ thống sử dụng phần cứng OpenGL chỉ được hổ trợ bởi một bảng màu phần cứng duy nhất . (Bảng màu phần cứng mapping một_một với bảng màu logic)
PFD_STEREO	Thiết lập khi chọn bộ đệm lập thể.không dùng trên Microsoft Windows
PFD_SUPPORT_GDI	Bộ đệm hổ trợ các hàm đồ họa GDI .Không dùng cùng với cờ PFD_DOUBLEBUFFER
PFD_SUPPORT_OPENGL	Bộ đệm hổ trợ các hàm đồ họa OpenGL
PFD_DOUBLE_BUFFER_DONTCARE	Định dạng có thể dùng bộ đệm đơn hay đôi ,không ưu tien cho bất cho riêng loai nào
PFD_STEREO_DONTCARE	Cảnh lập thể hoặc không lập thể, không ưu tiên cho riêng loại nào

4.3.2 .Khởi Tạo PIXELFORMATDESCRIPTOR:

Cấu trúc PIXELFORMATDESCRIPTOR là phức tạp và để có được một kết quả mong muốn, khi điền vào cấu trúc cần có một sự hiểu biết thực sự về OpenGL và hoat động của nó.Tuy nhiên, có thể dùng giá tri mặc định cho hầu hết các thành phần của cấu trúc. Các giá trị mặc định này hoạt động tốt trên nhiều hệ thống.

```
PIXELFORMATDESCRIPTOR pfd=
{
      sizeof (PIXELFORMATDESCRIPTOR)   //Kích thước cấu trúc.
     1,                  //Số version của cấu trúc
     PFD_DRAW_TO_WINDOW |      //Vẽ vào cửa sổ
      PFD_SUPPORT_OPENGL,        //Hỗ trợ lời gọi OpenGL
     PFD_TYPE_RGBA,             //Chế độ màu RGBA
     24,                  //24 bit màu
```

KILOBOOKS.COM
Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

```
    0,0,0,0,0,0,        //không lưu chọn
    0,0,0,0,0,0,0,0,    //các chế độ
    32,                 //bộ đệm chiều sâu 32 bit
    0,0,                //Không dùng bộ đệm stencil hay
                        bộ đệm phụ trợ
    PFD_MAIN_PLANE,     //Kiểu lớp chủ yếu
    0,                  //Không chọn
    0,0,0,              //Không chọn
};
```

với khởi tạo như trên, các cờ đặc tính trong dwFlags cho phép ứng dụng sử dụng các hàm OpenGL vẽ vào cửa sổ, trong khi cờ PFD_TYPE_DATA trong iPixelType chọn chế độ màu RGBA. Trường cColorBits chọn 24 màu để thể hiện 16.7 triệu màu (màu sẽ không ổn định trên hệ thống 256 màu). Các thành phần từ cRedBits đến cBlueShift không được chọn. Ứng dụng sẽ không dùng các bộ đệm alpha hay bộ đệm tích lũy, do thiết lập các trường từ cAccumBits đến cAccumAlphaBits bằng 0. Trường cDepthBits chọn bộ đệm chiều sâu 32 bit. Hai số 0 tiếp theo cho biết ứng dụng không dùng các bộ đệm phụ trợ hay bộ đệm stencil. Cờ PFD_MAIN_PLANE trong iLayerType là giá trị cờ duy nhất dùng cho kiểu lớp trong Microsoft Windows. Cuối cùng, các thành phần dự trữ và không hỗ trợ được thiết lập bằng 0.

4.3.3.Thiết Lập Định Dạng Điểm Vẽ :

Khi đã khởi tạo cấu trúc PIXELFORMATDESCRIPTOR, ta có thể thiết lập định dạng điểm vẽ, như đoạn sau :

```
CclientDC clientDC(this);
Int pixelFormat=ChoosePixelFormat(clientDC.m_hDC,&pfd);
BOOL result=SetPixelFormat(clientDC.m_hDC,pixelFormat,&pfd);
```

Ở dòng đầu tiên, chương trình lấy DC cho vùng client của cửa sổ ứng dụng. Sau đó, ChoosePixelFormat() ftrong dòng thứ hai tìm kiếm một chỉ số cho định dạng vẻ phù hợp nhất vơí yêu cầu định dạng.

Hai đối số của handle dối với DC (để chọn lựa định dạng điểm vẽ),
và địa chỉ của cấu trúc PIXELFORMATDESCRIPTOR
(giữ thuộc tính của định dạng điểm vẽ yêu cầu). Nếu việc gọi hàm không thành công, ChoosePixelFormat() trả về giá trị 0. Nếu thành công, nó trả về chỉ số định dạng điểm vẽ.

Dòng thứ 3 gọi SetPixelFormat() để thiết lập định dạng điểm vẽ. Ba đối số của hàm này là handle đối vơí DC, chỉ số định dạng điểm vẽ, và địa chỉ cấu trúc PIXELFORMATDESCRIPTOR . Nếu thành công , hàm trả về giá trị TRUE, ngược lại, nó trả về giá trị FALSE

4.3.4.Liệt Kê Định Dạng Điểm Vẽ:

Chú ý rằng ChoosePixelFormat() trả về một chỉ số định dạng điểm vẽ phù hợp với yêu cầu định dạng .Thông thường, không có vấn đề ở đây. Tuy nhiên, luôn luôn phải kiểm tra định dạng này trước khi chắc chắn nó phù hợp với ứng dụng. Một cách thực hiện là xem xét mọi định dạng có hiệu lực, rồi chọn ra định dạng nào được coi là tốt nhất cho ứng dụng. Đoạn mã thực hiện như sau :

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

```
PIXELFORMATDESCRIPTOR pfd;
CClientDC clientDC (this);
Int numFormat = DescribePixelFormat(clientDC.m_hDC,1,sizeof(pfd),&pfd);
For(int i= 0;I<=numFormats;++i)
{
    DescribePixelFormat(clientDC.m_hDC,i,sizeof(pfd),&pfd);
}
```

Dòng đầu tiên của đoạn mã khai báo cấu trúc PIXELFORMATDESCRIPTOR với tên là pfd. Dòng thứ 2 lấy DC cho vùng client của cửa sổ ứng dụng .

Dòng thứ 3 gọi DescribePixelFormat() để lấy số của pixel format được hổ trợ trên platform hiện hành, Bốn đối số của hàm này là handle đối với DC chỉ số điểm vẽ để kiểm tra, kích thước và địa chỉ cấu trúc PIXELFORMATDESCRIPTOR. Khi gọi DescribePixelFormat(), để lấy số của định dạng có sẳn, thì đối số thứ hai bằng 1

Sau khi có tổng số định dạng điểm vẽ, vòng for được lập từ 1 đến số đó. Trong vòng lập , DescribePixelFormat() điền cấu trúc PIXELFORMATDESCRIPTOR cho mỗi định dạng .Sau mỗi lời gọi đến DescribePixelFormat(), chương trình có thể kiểm tra cấu trúc PIXELFORMATDESCRIPTOR về các thuộc tính mà ứng dụng yêu cầu.

Đoạn mã trên không sử dụng GetPixelFormat(). Hàm này chỉ đơn giản trả chỉ số định dạng điểm vẽ hiện hành của DC. Nó được gọi như sau:

```
Int pixelFormat= GetPixelFormat(hDC);
```

Đối số duy nhất của GetPixelFormat() là handle DC mà ta cần định dạng điểm vẽ của nó. Khi không thành công nó trả về giá tri 0.

4.4. Tạo Ngữ Cảnh Biểu Diễn:

Ngữ cảnh biểu diễn được tạo ra và trở nên hiện hành trước khi các hàm OpenGL được sử dụng để vẽ vào cửa sổ. Tùy theo nhu cầu của chương trình, mà các kỹ thuật sau được sử dụng để quản lý ngữ cảnh biểu diễn:

Phương pháp 1:

Tạo ngữ cảnh biểu diễn và làm cho nó trở nên hiện hành khi đáp ứng thông báo WM_CREATE. Xóa ngữ cảnh biểu diễn, bao gồm cả việc làm cho nó trở nên không hiện hành, khi đáp ứng thôngWM_DESTROY.

Phương pháp 2 :

Tạo ngữ cảnh biểu diễn khi đáp ứng thông báo WM_CREATE, nhưng chỉ làm cho nó trở nên hiện hành khi sẳn sàng vẽ với OpenGL và ngay khi hoàn thành việc vẽ thì làm cho nó trở nên không hiện hành. Xóa ngữ cảnh biểu diễn khi đáp ứng thông báo WM_DESTROY

Cần chú ý các điểm đã nêu trong muc 4.2, đó là chỉ có thread với ngữ cảnh biểu diễn hiện hành là có thể gọi các hàm OpenGL. Ngoài ra, một thread chỉ có một ngữ cảnh hiện diện duy nhất. Điều này đúng ngay cả khi ứng dụng chỉ có một thread. Cuối cùng , một ứng dụng luôn luôn có ít nhất 1 thread.

4.4.1.Phương Pháp 1:

KILOBOOKS.COM
Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

Thuận lợi của phương pháp này là chương trình chỉ cần một lần làm cho ngữ cảnh u diễn trở nên hiện hành. Vì việc làm cho ngữ cảnh biểu diễn trở nên hiện hành chiếm ều thời gian xử lý, nên phương pháp 1 cho phép ứng dụng đáp ứng thông báo WM_PAINT nhanh hơn.

Điểm bất lợi là ở chổ phải duy trì một ngữ cảnh dụng cụ cho toàn bộ thời gian chạy chương trình. Ngoài ra, với các chương trình tạo ra bởi AppWizard của Visual C++, thì phương pháp này không thích hợp.

4.4.2.Phương Pháp 2 :

Với phương pháp này, chương trình tạo và giải phóng ngữ cảnh dụng cụ của cửa sổ cho mỗi lần vẽ, như vậy không cần duy trì ngữ cảnh dụng cụ cho toàn bộ thời gian chạy chương trình.Tuy nhiên, cứ mỗi lần tạo ngữ cảnh dụng cụ, chương trình đồng thời phải làm cho ngữ cảnh biểu diễn trở nên hiện hành .

Đoạn mả sau quản lý ngữ cảnh biểu diễn theo phương pháp 2:

```
IntOpenglView::OnCreate(LPCSREATESTRUCT  lpCreateStruct)
{
if(Cview :: ONCreate(lpCreateStruct) = = -1)
  return -1;
PIXELFORMATDESCRIPTOR pfd=
{
     sizeof (PIXELFORMATDESCRIPTOR) //Kích thước cấu trúc.
     1,                    //Số version của cấu trúc
     PFD_DRAW_TO_WINDOW |        //Vẽ vào cửa sổ
     PFD_SUPPORT_OPENGL,          //Hỗ trợ lời gọi OpenGL
     PFD_TYPE_RGBA,          //Chế độ màu RGBA
     24,                  //24 bit màu
     0,0,0,0,0,0,                //không lưu chọn
     0,0,0,0,0,0,0,0,            //các chế độ
     32,                  //bộ đệm chiều sâu 32 bit
     0,0,      //Không dùng bộ đệm stencil hay bộ đệm phụ trợ
     PFD_MAIN_PLANE,              //Kiểu lớp chủ yếu
     0,                   //Không chọn
     0,0,0,                //Không chọn
};
     CClientDC clientDC(this);
     int pixelFormat=ChoosePixelFormat(clientDC.m_hDC,&pfd);
   BOOL success=SetPixelFormat(clientDC.m_hDC,pixelFormat,&pfd);
   m_hDC = wglCreateContext(client.m_hDC)
   return 0;
}
void OpenglView::OnDraw(CDC* pDC)
{
     Copengl1Doc* pDoc=GetDocument();
```

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

```
    ASSERT_VALID(pDoc);

    wglMakeCurrent(pDC->m_hDC,m_hRC);
    DrawWithOpenGL();
    wglMakeCurrent(pDC->m_hDC,NULL);

}
void CopenglView ::ONDestroy()
{
    wglDeleteContext(m_hRC);
}
```

Hàm OnCreate() đáp ứng thông báo WM_CREATE bằng cách trước tiên lấy ngữ cảnh dụng cụ cho vùng client của cửa sổ. Sau đo, gọi ChoosePixelFormat () để tìm chỉ số định dạng điểm vẽ yêu cầu cho DC và gọi hàm SetPixelFormat () để thiết lập định dạng điểm vẽ.

Tiếp theo, OnCreate() gọi wglCreateContext() tạo RC thích hợp với ngữ cảnh dụng cụ đã cho. Đối số của hàm này là handle của DC. Nếu thành công, wglCreateContext() trả về một handle RC. Nếu không thành công, hàm trả về giá trị 0.

Trong đoạn trương trình trên, OnCreate() chỉ tạo một DC tạm thời, chỉ có phạm vi cục bộ, nên tự động hủy khi trả về, Do DC bị hủy, nó không thể làm cho RC trở nên hiện hành tại điểm này trong chương trình.

Đến khi cập nhật nội dung của cửa sổ ứng dụng, MFC gọi hàm OnDraw() của lớp view. Lúc này hàm OnDraw() đủ sức làm cho RC trở nên hiện hành, bằng cách gọi wglMakeCurrent() sử dụng handle của đối tượng DC (DC bút vẽ) đã được truyền cho hàm. Hai đối số của OnDraw() là handle DC và handle của RC. Nếu thành công, wglMakeCurrent() trả về giá trị TRUE, ngược lại nó trả về giá trị FALSE.

Sau khi làm cho RC trở nên hiện hành, OnDraw() gọi cục bộ DrawWithOpenGL() để vẽ lên cửa sổ. (Phải viết code cho hàm này)

Lời gọi lần 2 đến wglMakeCurrent(), với thông số thứ hai là NULL làm cho ngữ cãnh biểu diễn trở nên không hiện hành. (Với mục đích này, cũng có thể dùng giá trị NULL cho cả hai đối số)

Cuối cùng, khi kết thúc ứng dụng, MFC gọi hàm OnDestroy() của lớp view. Trong OnDestroy(), chương trình chỉ gọi wglDeleteContext() để xóa ngữ cảnh biểu diễn . Thông số duy nhất của hàm là handle của ngữ cảnh biểu diễn. Nếu thành công , thì wglDeleteContext() ftrả về giá trị là TRUE , trái lại nó trả về giá tri FALSE. Do ngữ cảnh dụng cụ được tạo và hủy trong đáp ứng cho thông báo WM_PAINT, nên không có ngữ cảnh dụng cụ để hủy ở cuối chương trình.

Có hai điều biết thêm là:

- Có thể cùng lúc làm cho ngữ cảnh biểu diễn hiện tại trở nên không hiện hành, và một ngữ cảnh biểu diễn mới trở nên hiện hành, bằng cách sử dụng handle của ngữ cảnh biểu diễn mới làm thông số thứ hai của hàm wglMakeCurrent ().
- Một ngữ cảnh dụng cụ không thể bị xóa khi còn ràng buột với ngữ cảnh biểu diễn. Trước khi xóa ngữ cảnh dụng cụ, phải gọi wglMakeCurrent() với giá trị của đối số thứ hai là NULL để tách ngữ cảnh dụng cụ ra khỏi ngữ cảnh biểu diễn.

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

Đoạn mã OnCreate() trên cho thấy cách dùng các hàm wglCreateContext(), lDeleteContext(), wglMakeCurrent(). Còn các hàm wglGetCurrentContext(), lGetCurrentDC(), dùng tìm kiếm thông tin về các ngữ cảnh biểu diễn và dụng cụ, thì sử dụng như sau :

Lời gọi wglCurrentContext();

HGLRC hRC = wglGetCurrentContext();

Nếu thành công, hàm trả về một handle theo RC hiện hành, trái lại nó trả về gía trị NULL .

Lời gọi wglCurrentDC()

HDC Hdc = wglGetCurrentDC();

Nếu thành công, hàm trả về một handle theo ngữ cảnh dụng cụ ràng buột với ngữ cảnh dụng cụ hiện hành.Trái lại nó trả về NULL

4.5.TổngKết:

Để gọi thành công các hàm OpenGL, chương trình phải tạo ngữ cảnh biểu diễn là phiên bảng OpenGL của ngữ cảnh dụng cụ- cho cửa sổ mà OpenGL sẽ vẽ vào. Để tạo ngữ cảnh biểu diễn, trước tiên phải thiết lập định dạng điểm vẽ của cửa sổ.

Khi đã sẵn sàng dùng các hàm OpenGL để vẽ, chương trình phải làm cho ngữ cảnh biểu diễn trở nên hiện hành, nghĩa là ràng buột ngữ cảnh biểu diễn với ngữ cảnh dụng cụ của cửa sổ. Trước khi kết thúc chương trình, ngữ cảnh biểu diễn phải được làm trở nên không hiện hành và bị xóa

Chương 5:Vẽ Hình Và Sử Dụng Màu

Chương này trình bày :

- Cách gọi cách hàm OpenGL
- Các trạng thái OpenGL
- Cách chọn màu
- Cách định nghĩa hình dạng
- Các hình dạng cơ bản được hổ trợ bởi OpenGL

5.1.Cú Pháp Lệnh OpenGL:

Qui ước đặt tên hàm OpenGL như sau :

g l C o l o r 3 f (…)

Thư viện hàm cơ bản số lượng đối số kiểu dữ liệu của đối số

Các bảng 5.1, 5.2 tổng kết các tiền tố, hậu tố và ý nghĩa của chúng.

Do thể hiện ở nhiều dạng khác nhau, với các hậu tố khác nhau, tùy thuộc vào số lượng và loại thông số mà chúng tiếp nhận, nên trên thực tế số lệnh mẫu (prototype hơn số lệnh cơ bản) nhiều hơn số lệnh cơ bản. Như bao gồm hạt nhân OpenGL có hơn 300 nguyên mẫu, trong khi đó chỉ gồm 115 hàm cơ bản:

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

Ví dụ lệnh glVertex2f() có hậu tố là 2f , cho biết lệnh có đối số là 2 dấu chấm động . glVertex có đến 24 dạng:

glVertex2d()	glVertex2f()	glVertex3d()	glVertex3f()	glVertex4d()	glVertex4f()
glVertex2i()	glVertex2s()	glVertex3i()	glVertex3s()	glVertex4i()	glVertex4s()
glVertex2dv()	glVertex2fv()	glVertex3dv()	glVertex3fv()	glVertex4dv()	glVertex4fv()
glVertex2iv()	glVertex2sv()	glVertex3iv()	glVertex3sv()	glVertex4iv()	glVertex4sv()

Chú ý rằng có 12 lệnh sau kết thúc bằng chữ v, cho biết các đối số được cho dưới dạng vector.

Bảng 5.1 : Các tiền tố lệnh OpenGL

Thư viện	Bộ hạt nhân	Utility	Auxiliary	WGL	Win32 API
Tiền tố	gl	glu	Aux	wgl	Không có tiền tố đặc biệt

5.2.Các Trạng Thái OpenGL:

Môi trường OpenGL được thiết lập trong mỗi chương trình duy trì một trạng thái riêng biệt.Ví dụ, khi thiết lập màu vẽ với lời gọi glColor(), thì màu này sẽ có tác dụng cho đến khi glColor() được gọi một lần nữa để thay đổi nó. Tư tưởng này rất quen thuộc trong lập trình trong Windows ,vì ngữ cảnh dụng cụ làm việc tương tự. Khi lựa chọn 1 bút vẽ, thì mọi đường thẳng sẽ có bề rộng và màu của bút vẽ được chọn. Tác dụng của bút sẽ được duy trì cho đến khi nó được thay thế.

Bảng 5.2 : Các hậu tố lệnh OpenGL

Hậu tố	Kiểu dữ liệu của đối số
B	Glbyte
B	GLdouble hay Glclampd
F	GLfloat hay Glclampf
I	GLint hay Glsizei
S	Glshort
Ub	GLubyte hay Glboonlean
Ui	GLuint, GLenum hay Glbitfield
Us	Glushort
Bv	GLbyte dạng vector
Dv	GLdouble hay GLclampd dạng vector
Fv	GLfloat hay GLclampf dạng vector
Iv	GLint hay GLsizei dạng vector
Sv	GLshort dạng vector
ubv	GLubyte hay GLboolean dạng vector
uiv	GLuint, GLenum hay Glbitfield dạng vector
usv	GLushort dạng vector

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

Các thuộc tính khác nhau của trạng thái OpenGL được chứa trong biến trạng thái. dể dàng có tác dụng 1 cách nhanh chóng, mỗi biến trạng thái có một giá trị mặc định. : giá trị này tồn tại cho đến khi các hàm tương ứng được gọi để thay thế thuộc tính. Ví dụ màu xóa (biến trạng thái tham chiếu bởi hằng GL_COLOR_CLEAR_VALUE) có giá trị mặc định là màu đen, hay màu vẽ (biến trạng thái tham chiếu bởi hằng GL_CURRENT_COLOR) có giá trị mặc định là màu trắng , có gần 200 biến trạng thái trong OpenGL.

5.3.Xét Một Chương Trình OpenGL Tối Thiểu:

Xét đoạn mã sử dụng các hàm OpenGL để thể hiện 1 đoạn thẳng đen trắng trên nền đen

```
void DrawWithOpenGL()
{
    glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
    glClear(GL_COLOR_BUFFER_BIT);
    glColor(1.0f, 1.0f, 1.0f);
  glBegin(GL_LINE)
    glVertex2f(0.25f, 0.25f);
    glVertex2f(0.75f, 0.25f);
  glEnd();
  glFlush();
}
```

Bây giờ ta xem xét kỹ hơn từng dòng của DrawWithOpenGL()

5.3.1.Đặt màu xóa:

Ở dòng đầu tiên, DrawWithOpenGL() gọi hàm glClearColor(), để thiết lập màu nền ở chế độ RGBA. Bốn đối số của hàm này là các giá trị có kiểu dữ liệu GLclampf, biểu diễn các thành phần màu: đỏ, xanh lá, xanh dương
và alpha
RGBA(red, green, blue, alpha) là chế độ màu kết hợp hệ thống RGB với alpha (là thành phần dùng điều khiển việc trộn màu)
RGB(red, green, blue) là chế độ pha trộn hay phương pháp thể hiện màu trên phương tiện dùng nguồn sáng như màng hình màu. RGB pha trộn theo tỷ lệ phần trăm các màu cơ bản là đỏ, xanh lá và xanh dương để tạo nên các màu khác. Hệ điều hành windows định nghĩa các tỷ lệ này bằng giá trị 8 bit, gọi là giá trị RGB. Giá trị RGB(0,0,0), tức không phần trăm (cường độ thấp nhất) của cả 3 màu cơ bản, cho ra màu đen. Giá trị RGB(255,255,255), tức 100 phần trăm (cường độ cao nhất) của cả 3 màu cơ bản, cho ra màu trắng.
Giá trị alpha =1 tương ứng độ mờ đục hoàn toàn, giá trị alpha = 0 tương ứng tính trong suốt hoàn toàn
OpenGL còn chế độ màu chỉ mục, là chế độ màu mà mỗi màu được chỉ định bởi 1 chỉ số trong bảng màu.

5.3.2.Xóa bộ đệm màu:

Dòng thứ hai trong DrawWithOpenGL() xóa bộ đệm màu với màu xóa hiện hành. Hàm glClear() hiểu đối số của nó như một chuỗi các cờ bit cho biết bộ đệm nào cần xóa

KILOBOOKS.COM
Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

m không trả về giá trị). Bộ đệm màu (được biểu thị bởi cờ _COLOR_BUFFER_BIT) là vùng bộ nhớ chứa ảnh thực được thể hiện trên màn hình. enGL sử dụng các loại bộ nhớ khác, bao gồm bộ đệm chiều sâu và bộ đệm stencil, để giúp nó xử lý ảnh.

5.3.3.Đặt Màu Vẽ Hiện Hành:

Lời gọi glColor3f() thiết lập màu vẽ hiện hành. Ba đối số của nó (được cho biết bởi hậu tố 3f) là các giá tri GLfloat xác định các thành phần màu : đỏ, xanh lá, xanh dương. Có 32 phiên bản của hàm glColor().

Sau khi đặt màu vẽ, mọi đối tượng được tạo bởi các lời gọi hàm OpenGL sẽ đươc vẽ bằng màu vẽ đã chọn. Như đã biết, màu này sẽ duy trì tác dụng cho đến khi nào bị thay đổi bởi lời gọi glColor() khác.

Có thể tác động một cách sâu sắc đến màu của các đối tượng trên màn hình nhờ vào các thao tác đồ họa như chiếu sáng, tạo bóng, làm mờ và pha trộn.

Tên hệ thống 256 màu, màu được chọn có thể không thể hiện trên màn hình như mong muốn. Đó là vì bảng màu tiêu chuẩn của Windows chỉ gồm 20 màu, gây khó khăn cho OpenGL trong việc thể hiện. Kỹ thuật chiếu sáng (lighting) sẽ cải thiện việc kiểm soát màu của OpenGL trên máy 256 màu, bằng cách tạo bảng màu logic.

5.3.4.Định Nghĩa Một Hình :

Trước khi ra lệnh cho OpenGL vẽ một hình, ta phải cho OpenGL biết hình đó ra sao. Điều này được thực hiện bằng cách định nghĩa các vertex đại diện cho đối tượng, tương tự như ở các chương đồ họa 2 chiều và đồ họa 3 chiều. Điểm khác biệt là không dùng các kiểu dử liệu tự định nghĩa như Vertex và Model. Thay vào đó, ta sử dụng các lệnh OpenGL glBegin(), glVertex(), glEnd().

Trong DrawWithOpenGL(), glBegine(GL_LINES) bắt đầu việ định nghĩa một hình. Trong trường hợp này, hình là đường thẳng, được biểu thị bằng cờ GL_LINES. Các cờ khác có thể sử dụng vơi glBegin là GL_POINTS, GL_LINE_STRIP, GL_LOOP, GL_TRIANGLES, GL_TRIANGLES_STRIP, GL_TRIANGLES_FAN, GL_QUADS, GL_QUADS_STRIP, và GL_POLYGONS

Do một đường thẳng có 2 vertex, sau khi gọi glBegin(), DrawWithOpenGL() gọi hàm glVertex2f() hai lần. Hàm glVertex2f(), định nghĩa một vertex, dùng các giá trị đối số kiểu GLfloat để mô tả tọa độ X và Y của vertex (Hàm glVertex() có 24 phiên bản)

Tại sao 1 thư viện 3-D như OpenGL lại định nghĩa 1 đối tượng chỉ với tọa độ X và Y. Còn các giá trị Z và W thì sao. Thực ra, mặc dù glVertex2f() chỉ xác định 2 giá trị X,Y của vertex, nhưng vẫn có giá trị z và x liên kết với vertex. Chúng mang các giá trị mặc định 0.0 cho z, và1.0 cho w.

Hình càng phức tạp càng dùng nhiều lời glVertex() để định nghĩa, có khi là một danh sách dài giữa cặp lời gọi glBegin(), glEnd(). Ví dụ, để định nghĩa 4 đường thẳng tạo nên hình chử nhật, các lời gọi như sau :

```
glBegin(GL_LINES)
```

KILOBOOKS.COM

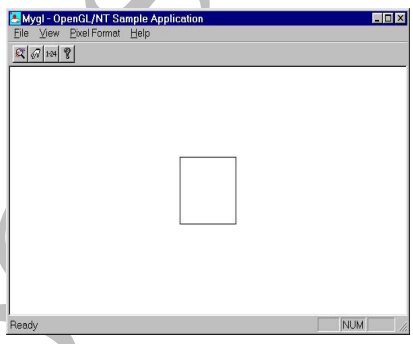
Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

```
glVertex2f(-0.5f,-0.5f);
glVertex2f(0.5f,-0.5f);

glVertex2f(0.5f,-0.5f);
glVertex2f(0.5f,0.5f);
glVertex2f(0.5f,0.5f);
glVertex2f(-0.5f,0.5f);

glVertex2f(-0.5f,0.5f);
glVertex2f(-0.5f,-0.5f);
glEnd();
```

Giá trị của các vertex trên dựa trên các tọa độ Decart mặc định thiết lập trong OpenGL. Các tọa độ mặc định này tạo nên một khối vuông 2x2x2, với gốc tọa độ (0,0) tại tâm khối vuông. Vì vậy, đoạn mã trên vẽ một hình chữ nhật ở tâm cửa sổ, như hình 5.1 .

Hình 5.1: Hình chữ nhật vẽ bằng OpenGL



Hàm glVertex() không đối số, luôn luôn kết thúc danh sách glVertex().
Hàm glVertex() chỉ là 1 trong 12 hàm có thể dùng giữa cặp glBegin() và glEnd(). Các hàm đó là :
glVertex(), glCallist(), glCallists(), glColor(), glEdgeFlag(),
glEvalCoord(), glIndex(), glMaterial(), glNormal(), glTextCoord(),

5.3.5.Bảo Đảm Các Thao Tác Vẽ Hoàn Tất:

Cho đến đây, chương trình đã thiết lập màu xóa , xóa nền, thiết lập màu vẽ, và định nghĩa đường thẳng. Mặc dù lúc này, ảnh cuối cùng có lẻ đã xuất hiện trên màng hình, thông thường thao tác vẽ được kết thúc với lời gọi glFlush(). Hàm này bảo đảm rằng mọi lệnh OpenGL ghi trong bộ đệm được thực hiện (glFlush() không có đối số)
Một hàm tương tự là glFinish(),thực hiện cùng một nhiệm vụ như glFlush(), nhưng trả về chỉ khi các thao tác vẽ hoàn tất.

5.4.Định Nghĩa Và Vẽ Điểm:

Hàm glBegin() khởi đầu định nghĩa một hình, đối số đơn của nó là kiểu hình sắp tạo. Các hình được đại diện bởi các hằng GL_POINTS, GL_LINE_STRIP, GL_LOOP, GL_TRIANGLES, GL_TRIANGLES_STRIP, GL_TRIANGLES_FAN, GL_QUADS, GL_QUADS_STRIP, và GL_POLYGONS.

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

iều hình được vẽ như tên gọi của chúng. Ví dụ, hình tạo bởi glBegin(GL_POINTS) là t điểm trên màn hình. Một chuỗi các điểm được định nghĩa như sau:

```
glBegin(GL_POINTS);
glVertex2f(0.0f,0.0f);
glVertex2f(0.75f,0.75f);
glVertex2f(-0.75f,-0.75f);
glVertex2f(-0.75f,0.75f);
glVertex2f(0.75f,-0.75f);
glEnd();
```

Ở đây, mỗi năm glVertex() định nghĩa một điểm đơn. Chú ý rằng đoạn mã trên định nghĩa các điểm trên mặt phẳng Decart 2D, chỉ với hai tọa độ X,Y. Nhưng như đã biết , OpenGL tự động thiết lập giá trị mặt định 0 và 1 cho các biến tương ứng z và w. Nếu định nghĩa các điểm trong tọa độ 3D, đoạn mã như sau:

```
glBegin(GL_POINTS);
        glVertex3f(0.0f,0.0f,0.0f);
        glVertex3f(0.75f,0.75f,1.0f);
        glVertex3f(-0.75f,-0.75f,1.0f);
        glVertex3f(-0.75f,0.75f,-1.0f);
        glVertex3f(0.75f,-0.75f,-1.0f);
glEnd();
```

Trong đoạn mã này, phiên bản 3f của glVertex() được dùng để định nghĩa các tọa độ 3-D X,Y,Z

Với màn hình có độ phân giải cao hiện nay, thì sẽ khó nhận ra một điểm đơn trên màn hình. Thật may, OpenGL có lệnh glPointSize(), cho phép vẽ điểm với bất kỳ kích thước nào. Ví dụ vẽ 1 điểm với kích thước là 4 pixel

```
glPointSize(4f);
```

Đối số đơn của hàm là giá trị GLfloat cho biết đường kính yêu cầu của điểm. Để xác định một dãy kích thước điểm (được hỗ trợ bởi OpenGL), glGetFloatv() được dùng như sau

```
glFloat ps[2];
glGetFloatv(GL_POINT_SIZE_RANGE,ps);
```

Với ps là 2 mảng thành phần có giá trị GLFloat chứa kích thước điểm nhỏ nhất và điểm lớn nhất. Lời gọi glGetFloatv() đòi hỏi các đối số gồm một hằng số cho biết giá trị cần đạt được và địa chỉ mảng glFloat chứa các giá trị đó. Có gần 150 hằng số có thể dùng với các dạng khác nhau của hàm glGet().

Cũng có thể đạt được kích thước điểm hiện hành với cách tương tự.

```
glFloat ps;
glGetFloatv(GL_POINT_SIZE_RANGE,&ps);
```

Trong trường hợp này, ps là biến kiểu glFloat chúa kích thước điểm hiện hành. Địa chỉ biến này được cho bởi đối số thứ hai trong hàm glGetFloat().

Đoạn mã sau vẽ các điểm như trên hình 5.2:

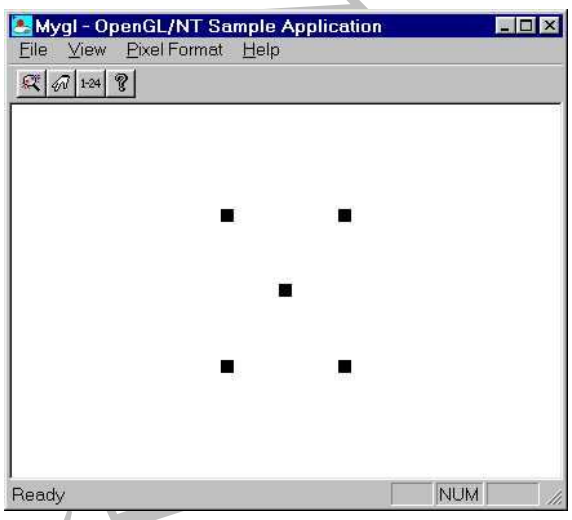
```
glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
```

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

```
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(0.0f, 0.0f, 0.0f);
glFloat ps[2];
glGetFloatv(GL_POINT_SIZE_RANGE,ps);
glPointSize(ps[1]);
glBegin(GL_POINTS);
glVertex2f(0.0f,0.0f);
glVertex2f(0.75f,0.75f);
glVertex2f(-0.75f,-0.75f);
glVertex2f(-0.75f,0.75f);
glVertex2f(0.75f,-0.75f);
glEnd();
glFlush()
```



Hình 5.2: Vẽ các điểm với OpenGL.

Chú ý rằng mặc dù có thể thiết lập đường kích của điểm, trên thực tế kích thước điểm ảnh bị ảnh hưởng bởi antialiasing, là thao tác đồ họa làm giảm hoặc loại trừ hiện tượng bậc thang của đường thẳng trên màn hình. Điều này cũng đúng cho các hình khác.

5.5.Định Nghĩa Và Vẽ Đường:

Một đường được xác định bởi hai vertex xác định điểm đầu và điểm cuối đường

```
glBegin(GL_LINES)
glVertex2f(0.0f,0.0f);
glVertex2f(1.0f,0.0f);
glEnd();
```

Và có thể định nghĩa cùng lúc nhiều đường

```
glBegin(GL_LINES)
glVertex2f(0.0f,0.0f);
glVertex2f(1.0f,0.0f);
glVertex2f(0.0f,0.0f);
glVertex2f(0.0f,1.0f);
glEnd();
```

Để xác định kích thước đường,ta dùng hàm glLineWidth() :

```
glLineWidth(4.0f);
```

KILOBOOKS
Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

Đối số đơn của hàm này là giá trị GLfloat cho biết bề rộng của đường.
xác định dãi bề rộng đường (được hỗ trợ bởi OpenGL), ta dùng hàm glGetFloatv()

```
GLfloat lw[2];
glGetFloatv(GL_POINT_SIZE_RANGE,lw);
```

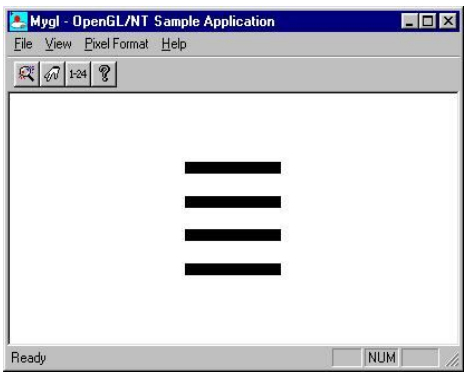
Với lw là mảng hai thành phần các giá trị GLfloat chứa bề rộng đường nhỏ nhất và lớn nhất.
Cũng có thể dùng glGetFloat() để lấy bề rộng hiện hành:

```
GLfloat lw;
glGetFloatv(GL_LIINE_WIDTH,&lw);
```

Với lw là biến kiểu GLfloat chứa bề rộng hiện hành của đường. Địa chỉ biến này được cho như là đối số thứ 2 trong hàm glGetFloatv()
Đoạn mã sau thể hiện hình hình 5.3

```
glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(0.0f, 0.0f, 0.0f);
glFloat lw[2];
glGetFloatv(GL_POINT_SIZE_RANGE,lw);
glLineWidth(lw[1]);
glBegin(GL_LINES);
glVertex2f(-0.75f,-0.75f);
glVertex2f(0.75f,0.75f);
glVertex2f(-0.75f,0.25f);
glVertex2f(0.75f,0.25f);
glVertex2f(-0.75f,-0.25f);
glVertex2f(0.75f,-0.25f);
glVertex2f(-0.75f,-0.75f);
glVertex2f(0.75f,-0.75f);
glEnd();
glFlush();
```

Hình5.3: Vẽ các đường với OpenGL



KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

5.5.1.Vẽ Đường Chấm:

OpenGL có thể vẽ đường chấm, là đường tạo nên bởi các điểm hay nét gạch. Bước ı tiên vẽ đường chấm là gọi glLineStipple() để xác định kiểu vẽ bằng chấm. Và để làm điều đó, cần định nghĩa màu chấm,bằng cách tạo một giá trị nhị phân, với 1 tương ứng với chấm, 0 tương ứng với không chấm

Ví dụ mẫu chấm 0000000011111111 cho kết quả là một đường tạo nên bởi khoảng trống và nét gạch xem kẻ. Hay mẫu chấm 0000111100001111 tạo một đường của các nét gạch ngắn.

Để sử dụng mẫu chấm làm đối số trong lời gọi glLineStipple(), phải chuyển nó sang giá trị thập lục phân. Ở ví dụ trên, các giá trị tương ứng là 0x00FF, và 0x0F0F.

Lời gọi glLineStipple() như sau .

```
glLineStipple(1,0x0F0F)
```

Hai đối số của hàm gồm 1 giá trị GLint biểu thị hệ số lặp mẫu (số lần lặp các điểm trong mẫu chấm), và một giá trị GLushort chứa mẫu chấm. Ví dụ, với hệ số lặp bằng 2, mẫu chấm là 01010101, thì đường chấm được vẽ là0011001100110011. Có thể hiểu hệ số lặp như tỷ lệ co giãn theo phương ngang của đường.

Sau khi thiếp lặp mẫu chấm, glEnable() được gọi để cho phép việc vẽ bằng chấm:

```
glEnable(GL_LINE_STIPPLE);
```

Hàm glEnable() có đối số đơn của là khả năng của OpenGL mà ta muốn sử dụng. Ví dụ, hằng GL_LINE_STIPPLE bắt đầu khả năng vẽ bằng chấm, GL_LIGHTING bắt đầu khả năng chiếu sáng. Có gần 50 hằng để sử dụng dvới các hàm glEnable() và glDisable().

Đoạn mã sau thể hiện như hình 5.4:

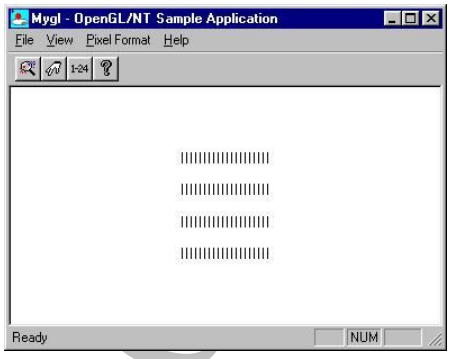
```
glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(0.0f, 0.0f, 0.0f);
glFloat lw[2];
glGetFloatv(GL_POINT_SIZE_RANGE,lw);
glLineWidth(lw[1]);
glLineStipple(1,0x4444)
glEnable(GL_LINE_STIPPLE);
glBegin(GL_LINES);
        glVertex2f(-0.75f,-0.75f);
        glVertex2f(0.75f,0.75f);
        glVertex2f(-0.75f,0.25f);
        glVertex2f(0.75f,0.25f);
        glVertex2f(-0.75f,-0.25f);
        glVertex2f(0.75f,-0.25f);
        glVertex2f(-0.75f,-0.75f);
        glVertex2f(0.75f,-0.75f);
glEnd();
glFlush();
```

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

Hình5.4: Vẽ các đường chấm với OpenGL



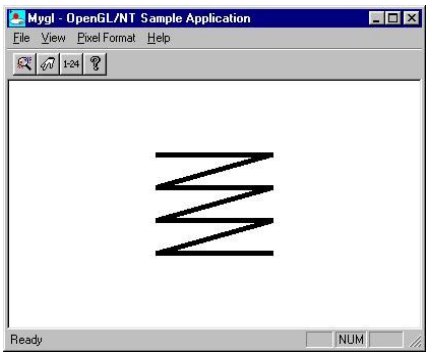
5.5.2.Vẽ Dải Đường:

Dải đường là một loạt đường nối với nhau. Khi định nghĩa dải đường, cặp vertex đầu tiên định nghĩa đường đầu tiên, mỗi vertex sau đó định nghĩa điểm kế tiếp mà OpenGL sẽ vẻ đến. Tương tự lệnh MoveTo() với 1 đoạn lệnh LineTo() theo sau. Trong trường hợp này, hằng GL_LINE_STRIP làm đối số cho hàm glBegin().

Đoạn mã sau thể hiện như hình 5.5:

```
glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(0.0f, 0.0f, 0.0f);
glFloat lw[2];
glGetFloatv(GL_POINT_SIZE_RANGE,lw);

glLineWidth(lw[1]/2);
glLineStipple(1,0x4444)
glBegin(GL_LINE_STRIPPLE);

    glVertex2f(-1.00f,0.75f);
    glVertex2f(1.00f,0.75f);

    glVertex2f(-1.00f,0.25f);
    glVertex2f(1.00f,0.25f);
    glVertex2f(-1.00f,-0.25f);
    glVertex2f(1,00f,-0.25f);
    glVertex2f(-1.00f,-0.75f);
    glVertex2f(1.00f,-0.75f);

glEnd();
glFlush();
```

Hình5.5: Vẽ dải đường với OpenGL



KILOBOOKS.COM
Nếu bạn cần file word? Xin hãy nhấp vào đây!

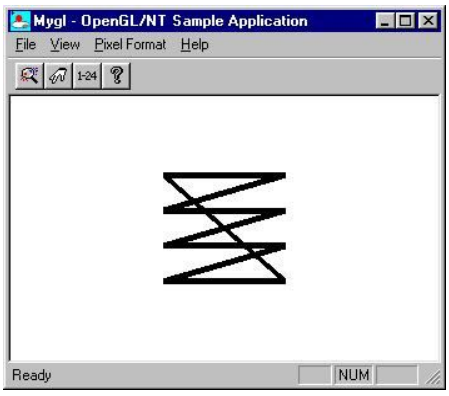
http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

5.5.3.Vẽ Đường Khép Kín:

Tương tự như dải đường, đường khép kín là 1 loạt các đường nối với nhau, nhưng vertex cuối cùng nối với đường đầu tiên. Khi định nghĩa đường khép kín, hằng GL_LINES_LOOP làm đối số cho hàm glBegin().

Đoạn mã sau thể thể hiện như hình 5.6 :

```
glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(0.0f, 0.0f, 0.0f);
glFloat lw[2];
glGetFloatv(GL_POINT_SIZE_RANGE,lw);

glLineWidth(lw[1]/2);
glBegin(GL_LINE_LOOP);
        glVertex2f(-1.00f,0.75f);
        glVertex2f(1.00f,0.75f);
        glVertex2f(-1.00f,0.25f);
        glVertex2f(1.00f,0.25f);
        glVertex2f(-1.00f,-0.25f);
        glVertex2f(1,00f,-0.25f);
        glVertex2f(-1.00f,-0.75f);
        glVertex2f(1.00f,-0.75f);

glEnd();
glFlush();
```



Hình5.6: Vẽ đường kép kín Với OpenGL.

5.6.*Định Nghĩa Và Vẽ Các Đa Giác:*

Đa giác là hình được tạo bởi các đường nối (cạnh) giữa một tập hợp các vertex. Một đa giác có ít nhất 3 cạnh. Như vậy, đa giác đơn giản nhất là tam giác.

OpenGL có thể vẽ đa giác như các điểm, các đường ngoài, hay đối tượng đặc biệt. OpenGL còn có thể điền đầy các đa giác với mẫu đã được định nghĩa. Thêm vào đó, một đa giác OpenGL có hai mặt trước và sau có thể vẽ theo hai cách riêng, và có thể quay 1 đa giác để xem mặt bên kia.

Hàm glPolygonMode() được dùng để vẽ đa giác:

```
glPolygonMode(GL_FRONT_AND_BACK,GL_LINE);
```

Hai đối số của hàm này là là các giá tri GLenum. Đối số đầu tiên có thể là hằng GL_FRONT, GL_BACK, hay GL_FRONT_AND_BACK, cho biết mặt đaa giác mà ta muốn thiết lập chế độ vẽ là mặt trước hay mặt sau, hay cả hai mặt. Đối số thứ 2 cho biết chế độ vẽ cho mặt đa giác đã chọn. Các chế độ vẽ có thể là GL_POINT (vẽ điểm tại các vertex của đa giác), GL_LINE (vẽ đa giác với các cạnh đa giác khung lưới), GL_FILL (đa giác đặc)

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

Để cho OpenGL biết mặt nào của đa giác là trước , và mặt nào là sau, ta dùng thứ khi định nghĩa các vertex tạo nên đa giác. Một đa giác được xem là đối-diện-mặt-trước, các vertex của nó được định nghĩa ngược chiều kim đồng hồ. Tuy nhiên có thể thay đổi quan niệm của OpenGL bằng cách gọi glFontFace():

```
glFontFace(GL_CW);
```

Lời gọi này cho OpenGL biết đa giác đối-diện-mặt-trước chọn định nghĩa theo chiều nào. Đối số đơn của hằng là GL_CW khi chọn chiều kim đồng hồ, và GL_CCW khi chọn chiều ngược chiều kim đồng hồ(mặc định)

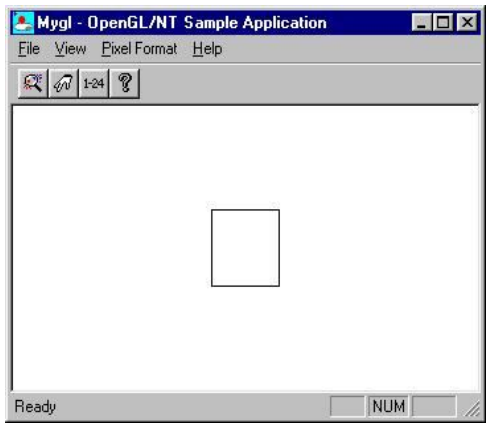
Định nghĩa một đa giác cũng dể như định nghĩa 1 đường . Chỉ cần ít nhất 3 vertex, và phải định nghĩa các vertex đúng chiều,(theo giá trị hiện hành củ biến trạng thái GL_FRONT_FACE). Đoạn mã sau định nghĩa một hình vuông đối-diện-mặt-trước:

```
glBegin(GL_POLYGON);
glVertex2f(-0.5f,0.5f);
glVertex2f(-0.5f,-0.5f);
glVertex2f(0.5f,-0.5f);
glVertex2f(0.5f,0.5f);
glEND();
```

Trong đoạn mã trên, hằng số GL_POLYGON được dùng với glBegin() để định nghĩa đa giác. Sau đó, định nghĩa các vertex của đa giác theo chiều ngược chiều kim đồng hồ (theo mặc định). Cuối cùng, như thường lệ, kết thúc việc định nghĩa bằng lời gọi glEND().

Đoạn mã sau thể hiện như hình 5.7:

```
glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(0.0f, 0.0f, 0.0f);
glPolygonMode(GL_FRONT,GL_LINE);
glPolygonMode(GL_BACK,GL_FILL);
glFontFace(GL_CCW);
glBegin(GL_POLYGON);
glVertex2f(-0.5f,0.5f);
glVertex2f(-0.5f,-0.5f);
glVertex2f(0.5f,-0.5f);
glVertex2f(0.5f,0.5f);
glEnd();
glFlush();
```



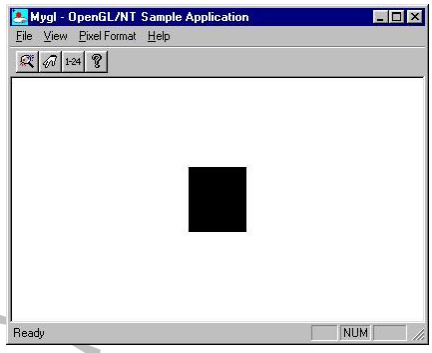
Hình 5.7: Vẽ đa giác với OpenGL.

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

Trong đoạn mã trên, hai lời gọi glPolygonMode() yêu cầu mặt trước đa giác vẽ o chế độ khung lưới, mặt sau theo chế độ đa giác đặc.



Hình 5.8: Vẽ đa giác đối diện mặt sau.

Vì đa giác là đối-diện-mặt-trước, nên chỉ mặt trước được thể hiện. Nếu thay đổi glFontFace(GL_CCW) bằng glFontFace(GL_CW), để báo cho OpenGL rằng đa giác là đối-diện-mặt-sau, chương trình sẽ thể hiện như hình 5.8

5.6.1.Vẽ Đa Giác Bằng Chấm:

Giống như khi vẽ đường với đường mẫu chấm do người dùng định nghĩa, ta có thể vẽ đa giác được điền đầy với 1 mẫu nào đó, bước đầu tiên để thực hiện là định dạng mẫu. Mẫu là bitmap 32x32, trong đó mỗi giá trị 1 thể hiện ra 1 điểm trên màn hình,

Ví dụ ở hình 5.9 là 1 dấu chấm dùng đa giác OpenGL . Mẫu được xây dựng trong hệ thống đường kẻ ô 32x32. Mỗi ô vuông được điền đầy trong mẫu thể hiện o một bit có giá trị 1.

Mẫu cần được chuyển đổi sang các giá trị thập lục phân (mảng giá trị GLubyte) để có thể dùng trong chương trình. Việc chuyển đổi này bắt đầu với byte ở góc trái hệ thống đường kẻ ô, rồi dần theo hướng phải và đi lên. Dữ liệu của mẫu hình 5.9 như sau:

```
GLubyte pattern[]=
{
        0xff,0x00,0xff,0x00,
        0x0f,0xf0,0x0f,0xf0,
        0x00,0xff,0x00,0xff,
        0x0f,0xf0,0x0f,0xf0,
        0xff,0x00,0xff,0x00,
        0x0f,0xf0,0x0f,0xf0,
        0x00,0xff,0x00,0xff,
        0x0f,0xf0,0x0f,0xf0,
        0xff,0x00,0xff,0x00,
        0x0f,0xf0,0x0f,0xf0,
        0x00,0xff,0x00,0xff,
        0x0f,0xf0,0x0f,0xf0,
        0xff,0x00,0xff,0x00,
        0x0f,0xf0,0x0f,0xf0,
        0x00,0xff,0x00,0xff,
        0x0f,0xf0,0x0f,0xf0,
        0xff,0x00,0xff,0x00,
```

KILOBOOKS.COM
Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

```
        0x0f,0xf0,0x0f,0xf0,
        0x00,0xff,0x00,0xff,
        0x0f,0xf0,0x0f,0xf0,
        0xff,0x00,0xff,0x00,
        0x0f,0xf0,0x0f,0xf0,
        0x00,0xff,0x00,0xff,
        0x0f,0xf0,0x0f,0xf0,
        0xff,0x00,0xff,0x00,
        0x0f,0xf0,0x0f,0xf0,
        0x00,0xff,0x00,0xff,
        0x0f,0xf0,0x0f,0xf0,
        0xff,0x00,0xff,0x00,
        0x0f,0xf0,0x0f,0xf0,
        0x00,0xff,0x00,0xff,
        0x0f,0xf0,0x0f,0xf0,
};
```

Chú ý rằng có 32 dòng dử liệu trong mảng trên. Mỗi dòng dử liệu là một dòng trên mẫu ở hình 5.9. Dòng đầu tiên trên trong mảng là dòng cuối cùng của mẫu chấm, và dòng dử liệu cuối cùng là dòng dử liệu đầu tiên của mẫu chấm.

Hàm glPolygonStipple() được gọi để chuyển mẫu chấm cho OpenGL:

```
glPolygonStipple(pattern);
```

Đối số của hàm là địa chỉ dữ liệu định nghĩa mẫu chấm.
Việc tiếp theo là gọi glEnable() để cho phép việc vẽ bằng chấm:

```
glEnable(GL_POLYGON_STIPPLE);
```

Sau lời gọi này mọi đa giác đặc trưng được vẽ điền đầy bởi mẫu đã chọn. Để trở lại việc vẽ đa giác thông thường, ta dùng hàm glDisable() :

```
glDisable(GL_POLYGON_STIPPLE);
```

Đoạn mã sau thể hiện hình 5.10

```
GLubyte pattern[]=
{
        0xff,0x00,0xff,0x00,  0x0f,0xf0,0x0f,0xf0,
                0x00,0xff,0x00,0xff,0x0f,0xf0,0x0f,0xf0,
                0xff,0x00,0xff,0x00,0x0f,0xf0,0x0f,0xf0,
                0x00,0xff,0x00,0xff,0x0f,0xf0,0x0f,0xf0,
                0xff,0x00,0xff,0x00,0x0f,0xf0,0x0f,0xf0,
                0x00,0xff,0x00,0xff,0x0f,0xf0,0x0f,0xf0,
                0xff,0x00,0xff,0x00,0x0f,0xf0,0x0f,0xf0,
                0x00,0xff,0x00,0xff,0x0f,0xf0,0x0f,0xf0,
                0xff,0x00,0xff,0x00,0x0f,0xf0,0x0f,0xf0,
                0x00,0xff,0x00,0xff,0x0f,0xf0,0x0f,0xf0,
                0xff,0x00,0xff,0x00,0x0f,0xf0,0x0f,0xf0,
                0x00,0xff,0x00,0xff,0x0f,0xf0,0x0f,0xf0,
                0xff,0x00,0xff,0x00,0x0f,0xf0,0x0f,0xf0,
```

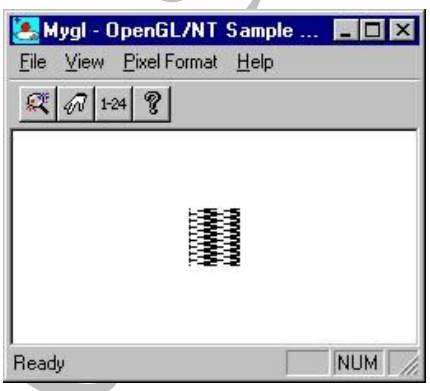
KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

```
                0x00,0xff,0x00,0xff,0x0f,0xf0,0x0f,0xf0,
                0xff,0x00,0xff,0x00,0x0f,0xf0,0x0f,0xf0,
                0x00,0xff,0x00,0xff,0x0f,0xf0,0x0f,0xf0,
};
        glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
        glClear(GL_COLOR_BUFFER_BIT);
        glColor3f(0.0f, 0.0f, 0.0f);
        glPolygonMode(GL_FRONT_AND_BACK,GL_FILL);
        glPolygonStipple(pattern);
        glEnable(GL_POLYGON_STIPPLE);
        glBegin(GL_POLYGON);
        glVertex2f(-0.5f,0.5f);
        glVertex2f(-0.5f,-0.5f);
        glVertex2f(0.5f,-0.5f);
        glVertex2f(0.5f,0.5f);
        glEND();
    glDisable(GL_POLYGON_STIPPLE);
```

Hình5.10: Đa giác chấm



Để vẽ hình chữ nhật một cách hiệu quả, OpenGL dùng hàm glRect(). Hàm này có tám phiên bản.

5.6.2.Vẽ Đa Giác Khung Lưới Lồi:

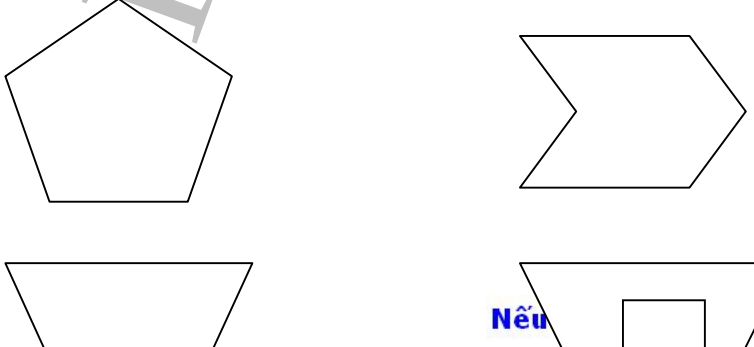
OpenGL có 2 sự hạn chế trong việc vẽ đa giác: (hình 5.11)

- Đa giác không được có các cạnh chéo nhau.
- Đa giác lồi, tức không chứa các vùng lõm hay lỗ bên trong

Tuy vậy, các hạn chế này không gây trở ngại nào, bởi vì có thể tạo hình dạng bất kỳ từ 2 hay nhiều đa giác lồi. Vấn đề là ở chổ khi nào tạo 1 đa giác không lồi dạng khung lưới bằng cách như vậy thì hình sẽ chứa những đường không mong muốn bên trong.

Để khử chúng, ta dùng hàm glEdgeFlag(). Khi gọi glEdgeFlag(FALSE) trước khi định nghĩa vertex, OpenGL sẽ không vẽ cạnh nối vertex đó với vertex được định nghĩa tiếp theo. Lời gọi glEdgeFlag(TRUE) sẽ trở lại việc vẽ cạnh bình thường.

Hình 5.11 Đa giác lồi và không lồi



Nếu ord? Xin hãy nhấp vào đây!

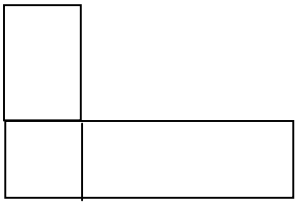
http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

Đa giác lồi Đa giác không lồi

Hình 5.12 :Tạo 1 đa giác lồi từ 2 đa giác không lồi

Hình 5.13 là đa giác không lồi chữ L tạo bởi ba hình chử nhật. Đoạn mã sau tạo hình tương tự nhưng không chứa các hình bên trong (Hình 5.14).

```
glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(0.0f, 0.0f, 0.0f);
glPolygonMode(GL_FRONT_AND_BACK,GL_LINE);
glBegin(GL_POLYGON);
        glEdgeFlag(TRUE);
        glVertex2f(-0.2f,0.3f);
        glEdgeFlag(FALSE);
        glVertex2f(-0.2f,-0.1f);
        glEdgeFlag(TRUE);
        glVertex2f(0.0f,-0.1f);
        glVertex2f(0.0f,0.3f);
glEnd();
glBegin(GL_POLYGON);
        glEdgeFlag(TRUE);
        glVertex2f(-0.2f,-0.1f);
        glVertex2f(-0.2f,-0.4f);
        glEdgeFlag(FALSE);
        glVertex2f(0.0f,-0.4f);
        glVertex2f(0.0f,-0.1f);
glEnd();
glBegin(GL_POLYGON);
        glEdgeFlag(FALSE);
        glVertex2f(0.0f,-0.1f);
        glEdgeFlag(TRUE);
 glVertex2f(0.0f,-0.4f);
        glVertex2f(0.3f,-0.4f);
```



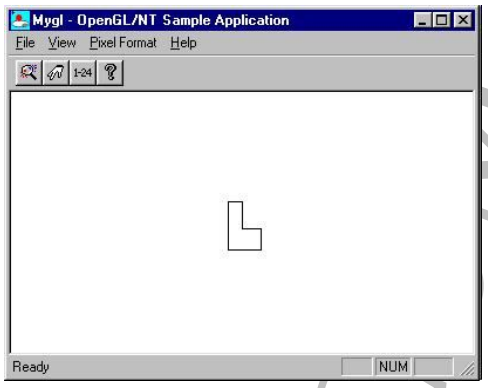
Hình 5.13: Tạo hình chữ L với 3 hình chữ nhật khung lưới

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

```
        glVertex2f(0.3f,-0.1f);
glEnd();
  glFlush();
```

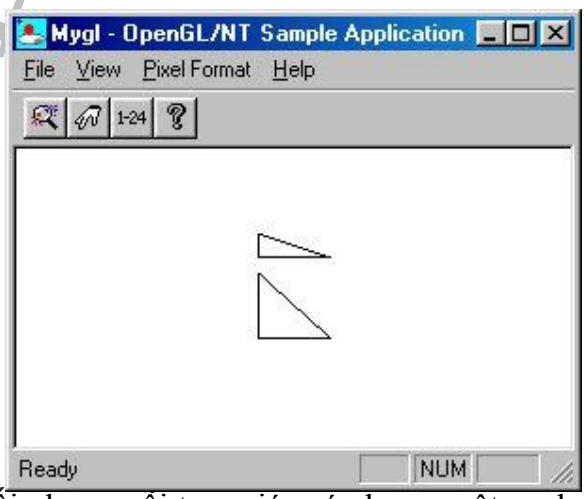


Hình 5.14 : Vẽ hình chữ L không chứa các đường bên trong

5.6.3.Vẽ tam giác:

OpenGL hỗ trợ việc vẽ nhiều đa giác đặt biệt, tam giác là 1 trong số đó, Việc định nghĩa tam giác không khác các đa giác khác, ngoại trừ việc sử dụng đối số GL_TRIANGLES cho hàm glBegin(). Trong đoạn mã sau thể hiện tam giác như hình 5.15 :

```
        glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
        glClear(GL_COLOR_BUFFER_BIT);
        glColor3f(0.0f, 0.0f, 0.0f);

        glPolygonMode(GL_FRONT_AND_BACK,GL_LINE);

        glBegin(GL_TRIANGLES);
        glVertex2f(-0.5f,0.3f);
        glVertex2f(-0.5f,-0.5f);
        glVertex2f(0.5f,-0.5f);

        glVertex2f(-0.5f,0.8f);
        glVertex2f(-0.5f,0.5f);
        glVertex2f(0.5f,0.5f);
        glEnd();
  glFlush();
```



Hình 5.15: Tam Giác OpenGL

5.6.4.Vẽ Dải Tam Giác:

Dải tam giác là chuỗi tam giác nối nhau, mỗi tam giác có chung một cạnh với tam giác vẽ trước nó. Nghĩa là sau khi định nghĩa 3 vertex cho tam giác đầu tiên, thì chỉ cần định nghĩa một vertex cho mỗi tam giác tiếp theo trong dãy tam giác.

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

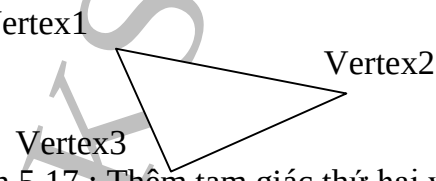
http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

ứ tự để định nghĩa các vertex của 1 dải tam giác là quan trọng. Tam giác đầu tiên của có các vertex định nghĩa như hình 5.16. Cạnh dùng chung với tam giác thứ hai trong được xác định bởi các vertex 3 và 2 theo thứ tự. Nghĩa là tam giác thứ 2 trong dải là được xác định bởi các vertex 3,2 và 4 (hình 5.17). Tiếp theo, vertex 3 và 4 xác định cạnh dùng chung của tam giác thứ 2 và 3…Ở trường hợp minh họa này, các tam giác là đối-diện-mặt-trước, nên vertex của chúng được định nghĩa ngược chiều kim đồng hồ.

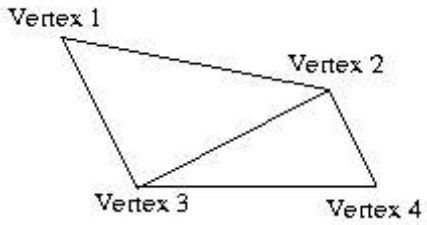
Bảng tổng kết thứ tự các vertex trong dải tam giác:

Số thứ tự tam giác	Cạnh 1	Cạnh 2	Cạnh 3
1	Vertex1	Vertex2	Vertex3
2	Vertex3	Vertex2	Vertex4
3	Vertex2	Vertex4	Vertex5
4	Vertex5	Vertex4	Vertex6
5	Vertex5	Vertex6	Vertex7
6	Vertex7	Vertex6	Vertex8
7	Vertex7	Vertex8	Vertex9

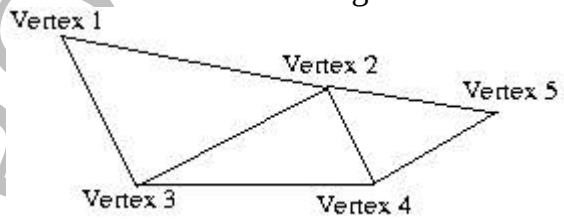
Hình 5.16 : Các vertex của tam giác đầu tiên trong dải tam giác



Hình 5.17 : Thêm tam giác thứ hai vào dải tam giác



Hình 5.18: Thêm tam giác thứ 3 vào dải tam giác

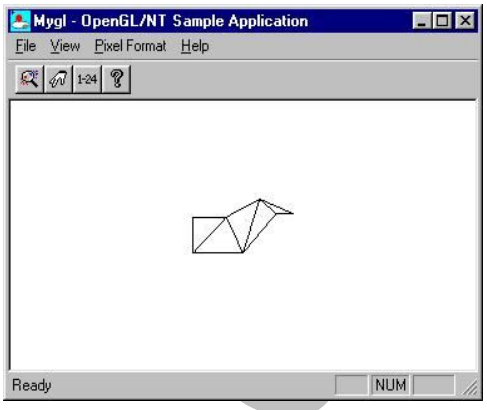



```
glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(0.0f, 0.0f, 0.0f);
glPolygonMode(GL_FRONT_AND_BACK,GL_LINE);
glBegin(GL_TRIANGLE_TRIP);
glVertex2f(-0.75f,0.25f);
glVertex2f(-0.75f,-0.25f);
glVertex2f(-0.25f,0.25f);
glVertex2f(0.0f,-0.25f);
glVertex2f(0.25f,0.5f);
```

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

```
glVertex2f(0.5f,0.3f);
glVertex2f(0.75f,0.3f);
glEnd();
glFlush();
```



Hình 5.19 : Dải tam giác

5.6.5.Vẽ quạt tam giác:

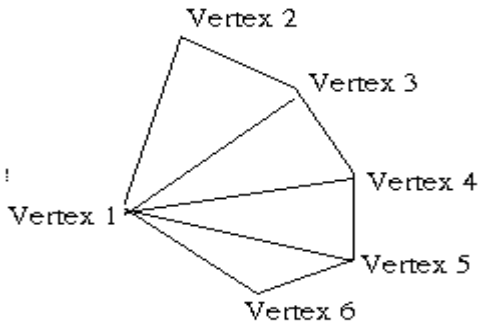
Quạt tam giác, được thể hiện bởi hằng GL_TRIANGLE_FAN, là chuỗi các tam giác có chung 1 vertex. Việc định nghĩa quạt tam giác tương tự như việc định nghĩa dải tam giác. Đầu tiên, định nghĩa 3 vertex cho tam giác thứ 1 trong quạt. Vì các tam giác kế tiếp có 1 cạnh dùng chung với tam giác trước, nên chỉ cần định nghĩa thêm 1 vertex. Hình 5.20 cho thấy thứ tự định nghĩa vertex của 1 quạt.

Bảng tổng kết thứ tự các vertex trong quạt tam giác:

Số thứ tự tam giác	Cạnh 1	Cạnh 2	Cạnh 3
1	Vertex1	Vertex2	Vertex3
2	Vertex1	Vertex3	Vertex4
3	Vertex1	Vertex4	Vertex5
4	Vertex1	Vertex5	Vertex6
5	Vertex1	Vertex6	Vertex7
6	Vertex1	Vertex7	Vertex8
7	Vertex1	Vertex8	Vertex9

Đoạn mã sau thể hiện như hình 5.21.

```
glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(0.0f, 0.0f, 0.0f);
glPolygonMode(GL_FRONT_AND_BACK,GL_LINE);

glBegin(GL_TRIANGLE_FAN)
glVertex2f(-0.25f,0.25f);

glVertex2f(-0.25f,-0.25f);
glVertex2f(0.25f,0.25f);

glVertex2f(0.25f,0.5f);
glVertex2f(-0.05f,0.7f);
glVertex2f(-0.45f,0.5f);
```



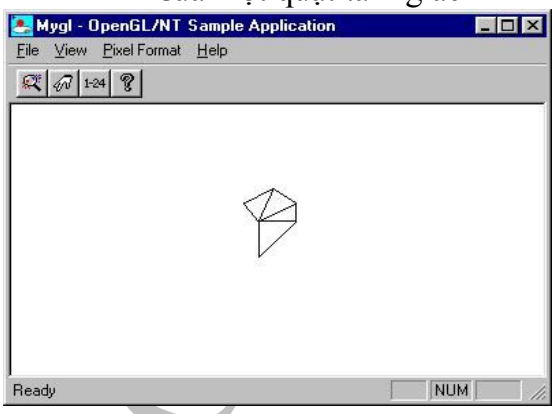
Hình 5.20 :Thứ tự định nghĩa các

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

của một quạt tam giác

```
glEnd();
glFlush();
```

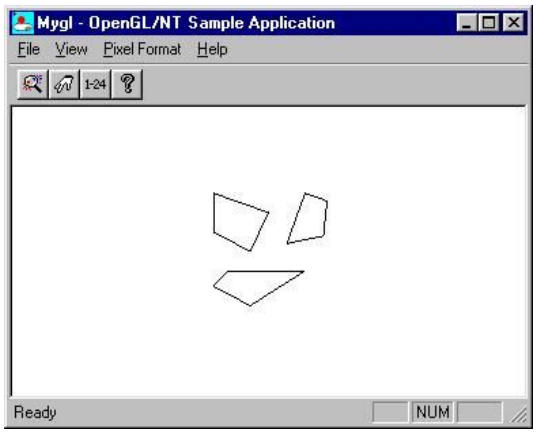


Hình 5.21 : Quạt tam giác

5.6.6.Vẽ Tứ Giác:

Để định nghĩa tứ giác, hằng GL_QUADS được sử dụng làm đối số cho hàm glBegin(). Mỗi tứ giác được định nghĩa bằng 4 vertex. Và có thể định nghĩa nhiều tứ giác giữa 1 cặp lệnh glBegin() và glEnd(). Đoạn mã sau thể hiện hình 5.22

```
glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(0.0f, 0.0f, 0.0f);

glPolygonMode(GL_FRONT_AND_BACK,GL_LINE);

glBegin(GL_QUADS) ;
      glVertex2f(-0.75f,0.75f);
      glVertex2f(-0.75f,0.25f);
      glVertex2f(-0.25f,0.0f);
      glVertex2f(0.0f,0.5f);
      glVertex2f(-0.55f,-0.25f);

      glVertex2f(-0.75f,-0.45f);
      glVertex2f(-0.25f,-0.7f);
      glVertex2f(0.5f,-0.25f);

      glVertex2f(0.5f,0.75f);
      glVertex2f(0.25f,0.1f);
      glVertex2f(0.75f,0.2f);
      glVertex2f(0.8f,0.65f);
glEnd();
glFlush();
```



Hình 5.22 : vẽ tứ giác

KILOBOOKS.COM
Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

5.6.7.Vẽ Dải Tứ Giác:

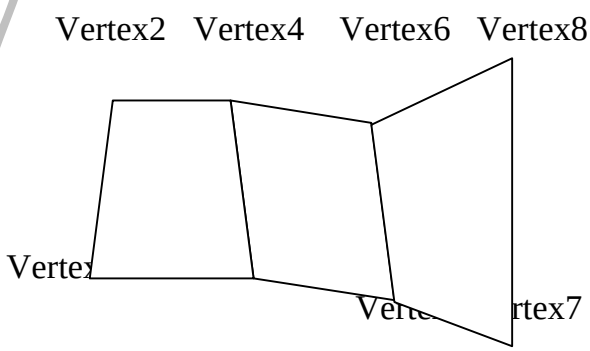
Dải tứ giác là chuỗi các tứ giác, trong đó mỗi cặp tứ giác có chung với nhau 2 rtex. Để định nghĩa dải tứ giác, ta dùng GL_QUAD-STRIP với hàm glBegin(). Sau đó định nghĩa 4 vertex cho tứ giác đầu tiên của dải, với mỗi tứ giác tiếp theo, ta định nghĩa thêm 2 vertex. Thứ tự định nghĩa vertex là rất quan trọng. Hình 5.23 cho thấy thứ tự định nghĩa vertex của 1 dải tứ giác.

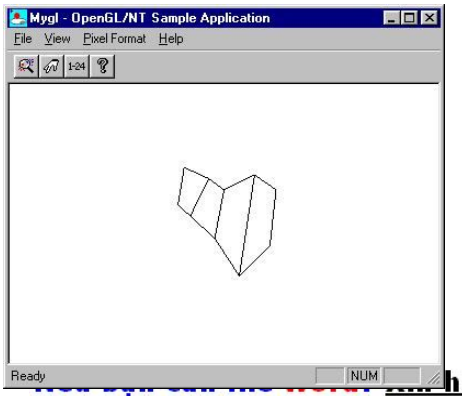
Bảng tổng kết thứ tự các vertex trong dải tam giác:

Số thứ tự tam giác	Cạnh 1	Cạnh 2	Cạnh 3	Cạnh 3
1	Vertex1	Vertex2	Vertex3	Vertex4
2	Vertex3	Vertex4	Vertex5	Vertex6
3	Vertex5	Vertex6	Vertex7	Vertex8
4	Vertex7	Vertex8	Vertex9	Vertex10
5	Vertex9	Vertex10	Vertex11	Vertex12
6	Vertex11	Vertex12	Vertex13	Vertex16
7	Vertex13	Vertex14	Vertex15	Vertex18

Đoạn mã sau thể hiện như hình 5.24

```
glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(0.0f, 0.0f, 0.0f);

glPolygonMode(GL_FRONT_AND_BACK,GL_LINE);
glBegin(GL_QUAD_STRIP) ;
      glVertex2f(-0.75f,0.25f);
      glVertex2f(-0.65f,0.75f);
      glVertex2f(-0.55f,0.1f);
      glVertex2f(-0.25f,0.6f);

glVertex2f(-0.15f,-0.2f);
      glVertex2f(0.0f,-0.45f);

      glVertex2f(0.25f,-0.7f);
      glVertex2f(0.5f,0.65f);


      glVertex2f(0.75f,-0.3f);
      glVertex2f(0.85f,0.45f);

glEnd();
glFlush();
```





KILOBOOKS.COM

hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

Hình 5.24 Dải tứ giác

5.7. Tổng Kết :

Một số hàm OpenGL có nhiều phiên bản, phụ thuộc vào số lượng và kiểu dữ liệu của đối số. Các phiên bản này được phân biệt bởi hậu tố theo sau tên hàm cơ bản.

Các hàm OpenGL thường thay đổi giá trị của biến trạng thái OpenGl. Các biến trạng thái này (ví dụ màu vẽ hiện hành) có hiệu lực cho đến khi bị thay thế. Điều này tương tự cách Windows quản lý DC.

Một chương trình OpenGL đơn giản thường bắt đầu bằng việc thiết lặp màu xóa và màu vẽ, rồi xóa bộ đệm màu để có "vùng trống" dùng để vẽ. Sau đó, định nghĩa các vertex của hình giữa 1 cặp hàm glBegin() và glEnd(). Đối số đơn của glBegin() quyết định kiểu hình đang được định nghĩa các vertex.

OpenGL có thể vẽ các hình như điểm, đường, đa giác, và tam giác. Thêm vào đó, có thể phối hợp các hình này vào dải đa giác, quạt tam giác, đườngg khép kín.và các kiểu hình khác phức tạp hơn. Mặc dù OpenGL không thể quản lý các đa giác không lồi, vẫn có thể tạo các đa giác này bằng cách ráp nhiều đa giác lồi lại với nhau.

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

Chương 6: Các Phép Biến Hình OpenGL

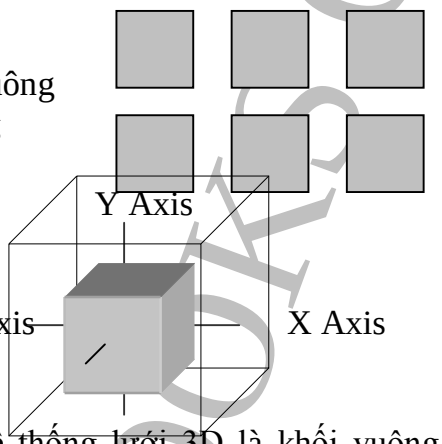
.Xây Dựng Đối Tượng 3D Từ Các Đa Giác:

Việc xây dựng đối tượng 3D từ các đa giác tương tự như gắn các mẫu giấy có hình dạng khác nhau(đa giác) thành các vật thể 3D.

Ví dụ ta có 6 mẫu giấy hình vuông 1" (hình 6.1), và ta muốn gắn chúng lại với nhau để có một khối vuông 1". Cho rằng ta đã có một khối vuông 2" được tạo thành từ các hình vuông 2" thuộc hệ thống lưới Decart 3D. Hình vuông lớn này thể hiện thế giới mà ta sẽ tổ hợp các mẫu giấy và tạo nên đối tượng. Khối vuông bằng giấy được tạo nên bằng cách đặt các mẫu giấy vào vị trí sao cho gốc hệ tọa độ Decart(0,0,0) sẽ nằm tại tâm khối vuông giấy. Tức tạo độ là tâm của hai khối vuông(hình 6.2).

Đó chính là cách xây dựng đối tượng 3D trong OpenGL. Ta sẽ cung cấp cho OpenGL một danh sách các vertex, thể hiện các tọa độ của đối tượng . Danh sách này không chỉ báo cho OpenGL cách vẽ đa giác, mà còn cho biết vị trí đa giác trong hệ thống lưới Decart 3D.

Hình 6.1 : Sáu hình vuông dùng tạo 1 khối vuông



Hình 6.2: Tổ hợp khối vuông

OpenGL mặc định hệ thống lưới 3D là khối vuông 2", với gốc tọa độ tại tâm. Nghĩa là các tọa độ X,Y,và Z là dãy từ –1.0 đến 1.0. Các giá trị trên trục X tăng từ trái sang phải, Các giá trị trên trục Y tăng từ dưới lên trên, Các giá trị trên trục Z tăng từ sau ra trước.

Như vậy, để hình thành khối vuông như hình 6.2, đoạn mã sau được sử dụng để xác định vertex cho các đa giác.

```
glBegin(GL_POLYGON) ;
        glVertex3f(-0.5f, 0.5f, 0.5f);
        glVertex3f(-0.5f, -0.5f, 0.5f);
        glVertex3f(0.5f, -0.5f, 0.5f);
        glVertex3f(0.5f, 0.5f, 0.5f);
glEnd()
glBegin(GL_POLYGON) ;
        glVertex3f(0.5f, 0.5f, 0.5f);
        glVertex3f(0.5f, -0.5f, 0.5f);
        glVertex3f(0.5f, -0.5f, -0.5f);
        glVertex3f(0.5f, 0.5f, -0.5f);
glEnd()
glBegin(GL_POLYGON) ;
```

KILOBOOKS.COM
Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

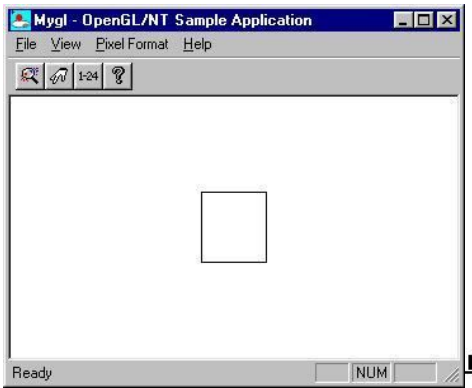
```
        glVertex3f(0.5f, 0.5f, -0.5f);
        glVertex3f(0.5f, -0.5f, -0.5f);
        glVertex3f(-0.5f, -0.5f, -0.5f);
        glVertex3f(-0.5f, 0.5f, -0.5f);
glEnd()
glBegin(GL_POLYGON) ;
        glVertex3f(-0.5f, 0.5f, -0.5f);
        glVertex3f(-0.5f, -0.5f, -0.5f);
        glVertex3f(-0.5f, -0.5f, 0.5f);
        glVertex3f(-0.5f, 0.5f, 0.5f);
glEnd()
glBegin(GL_POLYGON) ;
        glVertex3f(-0.5f, -0.5f, 0.5f);
        glVertex3f(-0.5f, -0.5f, -0.5f);
        glVertex3f(0.5f, -0.5f, -0.5f);
        glVertex3f(0.5f, -0.5f, 0.5f);
glEnd()
glBegin(GL_POLYGON) ;
        glVertex3f(-0.5f, 0.5f, -0.5f);
        glVertex3f(-0.5f, 0.5f, 0.5f);
        glVertex3f(0.5f, 0.5f, 0.5f);
        glVertex3f(0.5f, 0.5f, -0.5f);
glEnd()
```

Đoạn mã trên trước hết định nghĩa mặt trước khối vuông, kế tiếp là cạnh phải, phía sau, cạnh trái, đáy, và cuối cùng là đỉnh. Chú ý rằng các vertex được định nghĩa theo cách mặt trước các đa giác tạo nên bên ngoài khối vuông, trong khi mặt sau các đa giác ở phía trong khối vuông. Nghĩa là, vertex của các đa giác đối-diện-mặt-trước (như mặt trước khối vuông) được định nghĩa theo chiều ngược chiều kim đồng hồ, còn vertex của đa giác đối-diện-mặt-sau (như mặt sau khối vuông) được định nghĩa theo chiều kim đồng hồ.

Việc định nghĩa vertex các đa giác tạo nên đối tượng 3D đòi hỏi phải theo chiều thích hợp. Ở trạng thái mặc định, các vertex của đa giác đối-diện-mặt-trước được định nghĩa chiều ngược chiều kim đồng hồ. Việc không tuân theo quy tắc này có thể mang lại các kết quả không mong muốn.

6.2.Phép Chiếu:

Nếu đưa đoạn mã trên vào một chương trình OpenGl, kết quả nhận được sẽ như hình 6.3. Do OpenGL sử dụng phép chiếu trực giao (phép chiếu song song) đrể thể hiện hình khối đã cho.Với phép chiếu này, mặt sau khối vuông sẽ không nhỏ hơn, mà là cùng kích thước với mặt trước khối vuông, do đó không thể thấy được.



Để thể hiện khối vuông 1 cách thực hơn, ta phải sử dụng phép chiếu phối cảnh.

KILOBOOKS.COM

n hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

Hình 6.3: Hình khối thể hiện bằng phép chiếu trực giao

6.2.1. Chọn Ma Trận Chiếu:

Bước đầu tiên để sử dụng phép chiếu phối cảnh là chọn ma trận chiếu:

```
glMatrixMode(GL_PROTECTION);
```

OpenGL sử dụng 3 loại stack ma trận khác nhau để biến đổi các vertex của đối tượng cần thể hiện. Stack ma trận giống như stack mà máy tính vẫn dùng, nhưng lưu trữ giá trị cho các ma trận liên quan, chứ không chứa các kiểu dữ kiểu trạng thái như trong stack thông thường. Có thể nghĩ rằng một stack ma trận là một ma trận 4x4

Hàm glMatrixMode() cho phép lựa chọn ma trận. Đối số đơn của hàm có thể là 1 trong các hằng GL_MODEVIEW, GL_PROJECTION VÀ GL_TEXTURE. Các hằng này biểu diễn stack ma trận cần thao tác.

6.2.2.Khởi Tạo Ma Trận:

Để sử dụng ma trận vào việc biến hình, trước tiên cần khởi tạo nó bằng ma trận đơn vị. OpenGL có 1 hàm riêng biệt để thực hiện nhiệm vụ này, đó là :

```
glLoadIdentity();
```

Hàm glLoadIdentity() khởi tạo ma trận được chọn hiện hành bằng ma trận đơn vị, bảo đảm ma trận được khởi tạo không chứa các giá trị có thể ảnh hưởng đến các phép tính

6.2.3.Xác Định Không Gian Quan Sát:

Bước tiếp theo là gọi hàm glFrustum() để xác định giới hạn của không gian quan sát, là vùng 3D mà đối tượng tồn tại.

```
glFrustum(GLdouble,GLdouble,GLdouble,GLdouble,GLdouble,GLdouble);
```

Hàm này tạo ra một ma trận phối cảnh và nhân nó với ma trận chiếu được chọn hiện hành. Sáu đối số của hàm xác định các tọa độ trái, phải, đáy, đỉnh, gần, xa của không gian quan sát.

Các đối số gần và xa thể hiện khoảng cách từ "mắt". Như vậy mang các giá trị dương. Mắt là điểm từ đó ta nhìn thấy cảnh. Nếu không thay đổi thì "mắt" ở vị trí gốc tọa độ (0,0,0).

Mặt phẳng gần bắt đầu một không gian quan sát. Mặt phẳng xa kết thúc 1 không gian quan sát. Mọi đối tượng hay bộ phận đối tượng nằm giữa mắt và mặt phẳng gần, hay

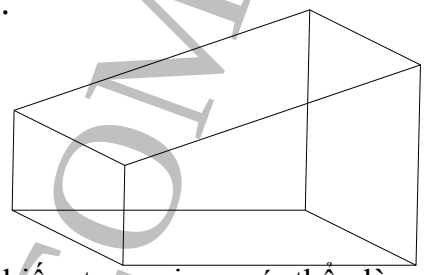
KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

.a hơn mặt phẳng xa, hoặc ở bên ngoài giới hạn trái, phải, đáy và đỉnh sẽ không xuất n trong màn hình. Nói cách khác mọi đối tượng hay bộ phận đối tượng ở bên ngoài ›ng gian quan sát đều bị cắt bỏ, không được thể hiện. Các mặt phẳng giới hạn không gian quan sát được gọi là các mặt phẳng cắt.

Hình 6.5 thể hiện thế giới được xác định bởi lời gọi glFrustum(). Để được thể hiện trên hình, dối tượng phải nằm trên thế giới này.



Hình 6.5:Thế giới được xác định bởi lời gọi gglFrushtum()

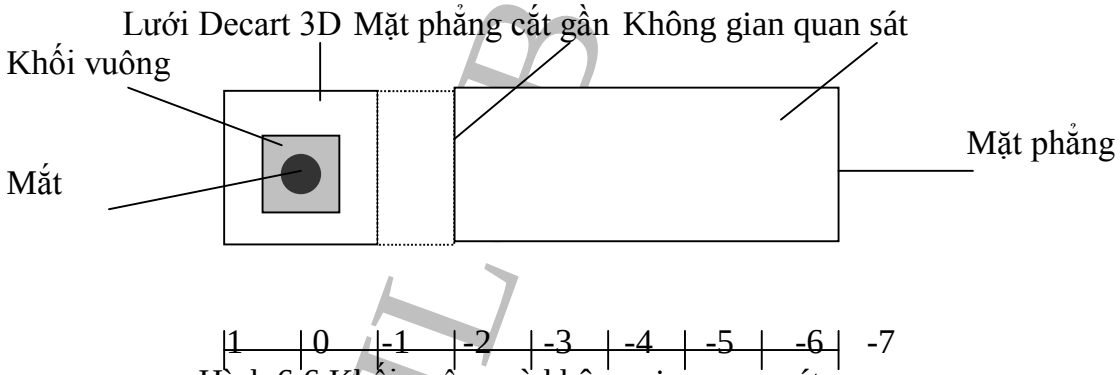
Trong trường hợp cần sử dụng phép chiếu trực giao, có thể dùng các hàm glOrtho() và glOrtho2D() để thiết lập phép chiếu.

6.3.Phép Biến Hình Đối Tượng (Modelview Transformation):

Để biểu diễn được đối tượng lên màn hình, cần phải di chuyển đối tượng vào không gian quan sát đã được xác định. Vì sau lời gọi glFrustum(), đối tượng vẫn nằm bên ngoài không gian quan sát.

Hình 6.6 mô tả trường hợp lời gọi glFrustum(-1.0, 1.0, -1.0, 1.0, 2.0, 7.0) thiết lập mặt phẳng cắt gần cách mắt 2 đơn vị, và mặt phẳng cắt xa cách mắt 7 đơn vị. Nhưng khối vuông vẫn ở vị trí góc tọa độ, hoàn toàn nằm ngoài không gian quan sát. Mặt khác khoảng cách từ mắt đến mặt phẳng cắt gần luôn là số dương, nên trong mọi trường hợp, không thể đặt đối tượng vào hẳn trong không gian quan sát đã được xác định. Giải pháp cho vấn đề này là tịnh tiến đối tượng.

(Thực ra cũng có thể sử dụng hàm glClipPlane() để thể hiện đối tượng ở ngoài không gian quan sát đã được xác định bởi glFrustum())



Hình 6.6 Khối vuông và không gian quan sát.

6.3.1.Phép Tịnh Tiến :

Đoạn mã sau thực hiện việc tịnh tiến khối vuông theo trục Z, để đặt nó vào không gian quan sát:

```
glMatrixMode(GL_MODEVIEW);
glLoadIdentity();
glTranslatef(0.0f, 0.0f, -3.5f);
```

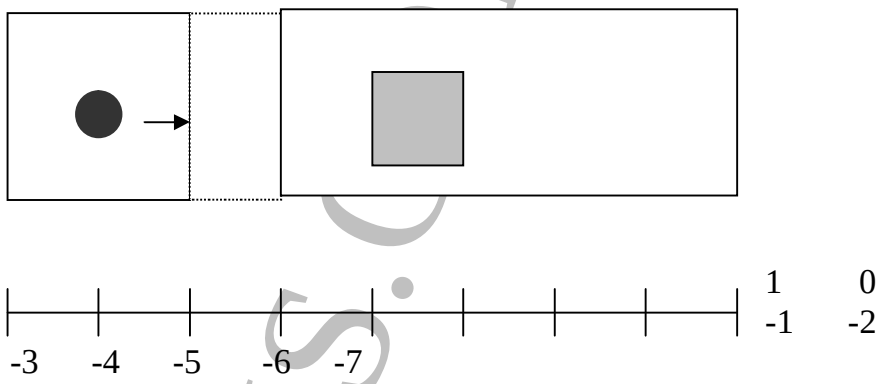
KILOBOOKS.COM
Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

ng thứ nhất báo cho OpenGL biết kiểu stack ma trận cần thao tác là modelview. Dòng hai khởi tạo ma trận modelview với ma trận đơn vị. Dòng thứ 3 tịnh tiến khối vuông đơn vị theo chiềm âm trục Z. Hàm glTranslatef() tạo một ma trận tịnh tiến, rồi nhân ma trận này với ma trận được chọn hiện hành. Ba đối số của nó là các giá trị GLfloat cho biết các lượng tịnh tiến trên trục X,Y và Z. Một phiên bản khác của hàm này là glTranslated(), có đối số kiểu GLdouble.

Sau khi các dòng trên được thực hiện, đối tượng di chuyển vào không gian quan sát như hình 6.7. Tất nhiên, do khối vuông ở xa mắt hơn trước, nên hình sẽ nhỏ hơn.

Hình 6.7: Khối vuông tiến vào không gian quan sát



Đoạn mã sau thể hiện hình 6.8

```
glMatrixMode(GL_PROTECTION);
glLoadIdentity();
glFrustum(-1.0, 1.0, -1.0, 1.0, 2.0, 7.0);

glMatrixMode(GL_MODEVIEW);
glLoadIdentity();
glTranslatef(0.0f, 0.0f, -3.5f);

glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(0.0f, 0.0f, 0.0f);

glPolygonMode(GL_FRONT_AND_BACK,GL_LINE);
glBegin(GL_POLYGON) ;
        glVertex3f(-0.5f,  0.5f, 0.5f);
        glVertex3f(-0.5f, -0.5f, 0.5f);
glVertex3f(0.5f, -0.5f,  0.5f);
glVertex3f(0.5f, 0.5f, 0.5f);
glEnd()
glBegin(GL_POLYGON) ;
      glVertex3f(0.5f, 0.5f, 0.5f);
      glVertex3f(0.5f, -0.5f, 0.5f);
      glVertex3f(0.5f, -0.5f, -0.5f);
      glVertex3f(0.5f, 0.5f, -0.5f);
```

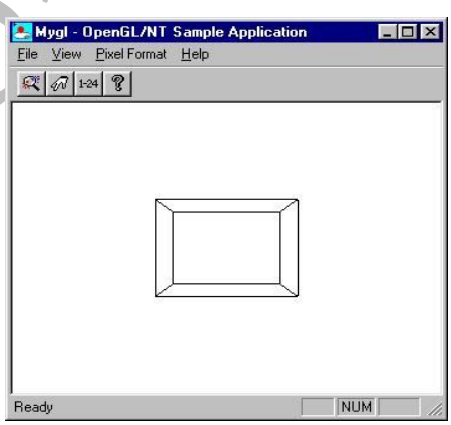
KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

```
glEnd()
glBegin(GL_POLYGON) ;
        glVertex3f(0.5f, 0.5f,-0.5f);
        glVertex3f(0.5f, -0.5f, -0.5f);
        glVertex3f(-0.5f, -0.5f, -0.5f);
        glVertex3f(-0.5f,0.5f, -0.5f);
glEnd()
glBegin(GL_POLYGON) ;
        glVertex3f(-0.5f, 0.5f, -0.5f);
        glVertex3f(-0.5f, -0.5f, -0.5f);
        glVertex3f(-0.5f, -0.5f, 0.5f);
        glVertex3f(-0.5f, 0.5f, 0.5f);
glEnd()
glBegin(GL_POLYGON) ;
        glVertex3f(-0.5f, -0.5f, 0.5f);
        glVertex3f(-0.5f, -0.5f, -0.5f);
        glVertex3f(0.5f, -0.5f, -0.5f);
        glVertex3f(0.5f, -0.5f, 0.5f);
glEnd()
glBegin(GL_POLYGON) ;
        glVertex3f(-0.5f, 0.5f, -0.5f);
        glVertex3f(-0.5f, 0.5f, 0.5f);
        glVertex3f(0.5f, 0.5f, 0.5f);
        glVertex3f(0.5f, 0.5f, -0.5f);
glEnd()
glFlush();
```

Đoạn mã thiết lập phép chiếu và phép biến đổi đối tượng quan sát, rồi vẽ khối vuông lên màn hình. Để vẽ khối vuông, OpenGL cung cấp đến từng vertex của khối vuông, trước tiên là phép biến đổi đối tượng quan sát, kế tiếp là phép chiếu phối cảnh, cuối cùng là phép biến đổi viewport.



Hình 6.8 : Hình phối cảnh khối vuông

Tất nhiên, cũng có thể tịnh tiến đối tượng theo các trục khác. Ví dụ thay các dòng sau vào đoạn mã ví dụ trên không chỉ tịnh tiến khối vuông theo chiều âm trục Z, mà còn đưa khối vuông đi lên và sang phải.(hình 6.9)

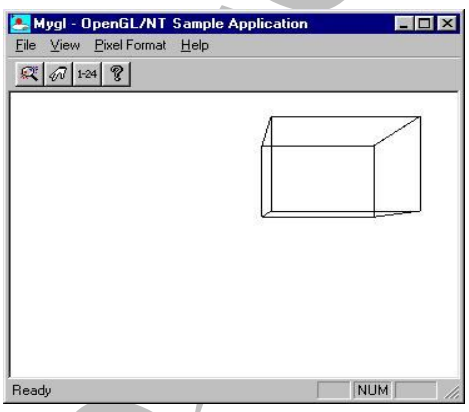
KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

```
glMatrixMode(GL_MODEVIEW);
glLoadIdentity();
glTranslatef(0.75f, 0.75f, -3.5f);
```

Một cách khác sử dụng phép biến hình là dùng hàm gluLookAt(). Hàm này cho phép xem cảnh mà không cần các phép tịnh tiến và quay.



Hình 6.9:Tịnh tiến Khối vuông theo ba trục

6.3.2.Phép co giãn:

Hàm Scalef() tạo một ma trận co giãn, và nhân nó với ma trận được chọn hiện hành. Ba đối số của hàm là các giá trị GLfloat xác định các hệ số co giãn theo các trục X,Y và Z. Một phiên bản khác của hàm là glScaled() có các đối số kiểu Gldouble.

Đoạn mã sau thể hiện như hình 6.10, trong đó DrawCube() là hàm định nghĩa các vertex của hình khối:

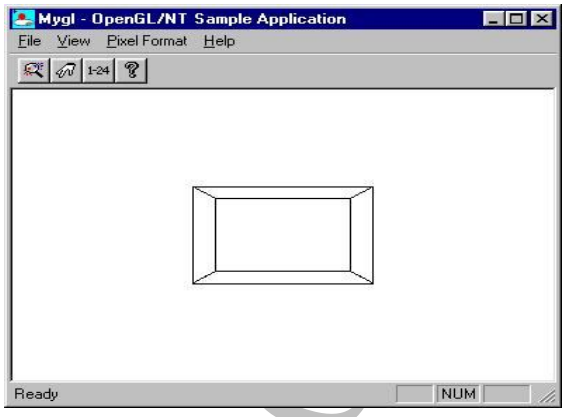
```
glMatrixMode(GL_PROTECTION);
glLoadIdentity();
glFrustum(-1.0, 1.0, -1.0, 1.0, 2.0, 7.0);

glMatrixMode(GL_MODEVIEW);
glLoadIdentity();
glTranslatef(0.0f, 0.0f, -3.5f);

glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(0.0f, 0.0f, 0.0f);

glPolygonMode(GL_FRONT_AND_BACK,GL_LINE);
DrawCube();
glFlush();
```

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

Hình 6.10: Khối vuông giãn hai lần theo trục X



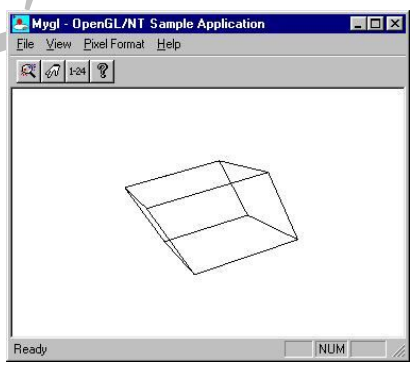
6.3.3.Phép quay:

Hàm glRotatef() tạo một ma trận quay, và nhân nó với ma trận được chọn hiện hành. Bốn đối số của hàm là các giá trị GLfloat xác định gốc quay ngược chiều kim đồng hồ và các tọa độ X,Y và Z của verter quay (để quay theo 1 trục, đặt giá trị 1.0 cho trục đó và0.0 cho trục còn lại). Một phiên bản khác của hàm glRotated() có các đối số kiểu Gldouble.

Đoạn mã sau thể hiện như hình 6.11, chú ý rằng ba lời gọi glRotatef (20.0f, 1.0f, 1.0f, 1.0f), vì lời gọi này sẽ quay hình quanh vector vẽ từ gốc tọa độ đến điểm(1.0,1.0,1.0).

```
glMatrixMode(GL_PROTECTION);
glLoadIdentity();
glFrustum(-1.0, 1.0, -1.0, 1.0, 2.0, 7.0);

glMatrixMode(GL_MODEVIEW);
glLoadIdentity();
glTranslatef(0.0f, 0.0f, -3.5f);
glRotatef ( 20.0f, 1.0f, 0.0f, 0.0f);
glRotatef ( 20.0f, 0.0f, 1.0f, 0.0f);
glRotatef ( 20.0f, 0.0f, 0.0f, 1.0f);
glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(0.0f, 0.0f, 0.0f);

glPolygonMode(GL_FRONT_AND_BACK,GL_LINE);
DrawCube();
glFlush();
```

Hình 6.11: Quay khối vuông theo 3 trục tọa độ



KILOBOOKS.COM
Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

Ở cả 3 phép biến hình vừa trình bày trên, nếu muốn thực hiện 1 cách trực tiếp như với ma trận như ở chương 2 "Đồ họa hai chiều", ta có thể sử dụng các hàm OpenGL glLoadMatrix(), và glMulMatrix(). glLoadMatrix() thiết lập tập giá trị cho ma trận hiện hành, còn glMulMatrix() nhân ma trận hiện hành với ma trận đã cho.

6.4.Phép Biến Đổi Viewport:

Như đã nói, để vẽ 1 hình OpenGL thực hiện phép biến hình đối tượng, để đặt vào vị trí mong muốn, (việc này liên quan đến điểm nhìn). Kế đến, OpenGL cung cấp phép chiếu phối cảnh(hoặc song song) để xén hình và chiếu phối cảnh(hoặc song song). Cuối cùng, OpenGL thực hiện phép biến đổi viewport, để quyết định cách mà các vertex được thể hiện trên thiết bị(màn hình hay máy in).Viewoport chỉ đơn giản là ô chử nhật mà hình được thể hiện trên đó.

6.4.1.Thiết Lập Viewport:

Khi không cần xác định một viewport riêng biệt, ta có thể sử dụng một viewport mặc định để thể hiện hình. Tuy nhiên, có lúc chỉ cần sử dụng viewport là 1 phần cửa sổ, hay cần viewport thay đổi khi thay đổi kích thước của cửa sổ… Khi đó, hàm glViewport() được sử dụng để thiết lập kích thước và vị trí viewport() như sau:

```
glViewport(left,bottom,width,height);
```

Hai đối số đầu của hàm là các giá trị Glint xác định góc trái dưới. Hai đối số tiếp theo là các giá trị Glsizesi xác định chiều rộng và chiều cao viewport.

6.4.2. Giải Quyết Tỉ Lệ Thể Hiện:

Viewport có thể cùng kích thước với cửa sổ, bất chấp kích thước không gian quan sát. Và khi cửa sổ thay đổi, viewport cũng được thiết lập lại theo kích thước của cửa sổ mới. Điều này làm cho hình co giản trên cở sở tỉ lệ thể hiện của cửa sổ, là tỉ lệ chiều rộng trên chiều cao cửa sổ.

Ví dụ, khi cửa sổ là vuông, thì hình vuông vẽ trên cửa sổ sẽ là vuông (hình 6.12). Nhưng khi cửa sổ bị kéo dài ra, thì hình vuông cũng bị kéo dài theo(hình 6.13). Nếu tỉ lệ thể hiện của viewport và tỉ lệ thể hiện của không gian quan sát là một. Thì hình OpenGL sẽ không biến dạng.

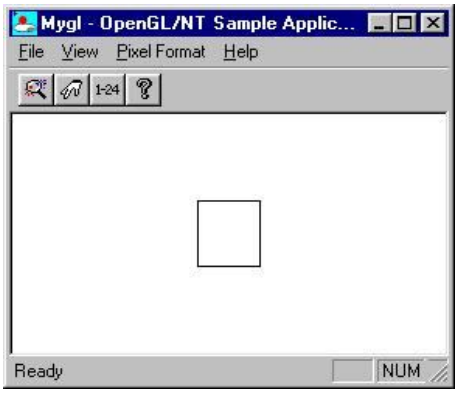
Để thay đổi kích thước viewport mỗi khi kích thước cửa sổ bị thay đổi, thông báo WM_SIZE của Windows phải được đáp ứng. Ứng dụng nhận thông báo này khi cửa sổ thể hiện lần đầu, cũng như mỗi khi kích thước cửa sổ bị thay đổi. Có thể xử lý thông báo WM_SIZE như sau:

```
Void Copengl3dView::OnSize(UINT nType, int cx, int cy)
{
Cview::OnSize(nType,cx,cy);
        //TODO:Add your message handler code here
        CClientDC clientDC(this);
```

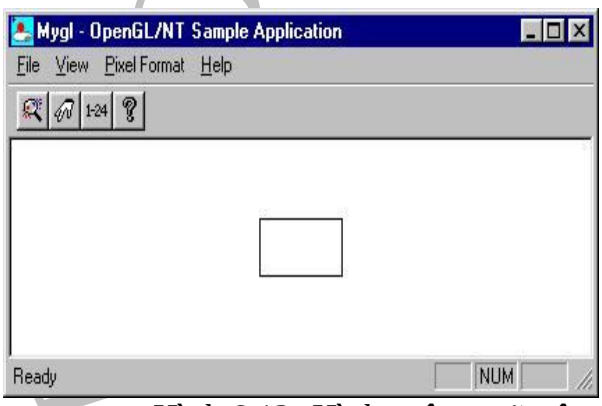
KILOBOOKS.COM
Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

```
        wglMakeCurrent(clientDC.m_hDC,m_hRC);

        glMatrixMode( GL_PROJECTION);
        glLoadIdentity();

        glFrustum(-1.0,1.0,-1.0,1.0,2.0,7.0);
        glViewport(0,0,cx,cy);
        wglMakeCurrent(NULL,NULL);
}
```



Hình 6.12: Hình vuông vẽ trên viewport vuông



Hình 6.13: Hình vuông vẽ trên viewport hình chữ nhật

Trước tiên đoạn mã tạo một ngữ cảnh dụng cụ cho cửa client, rồi dùng handle theo ngữ cảnh dụng cụ này làm cho đối số thứ nhất trong lời gọi wglMakeCurrent. Lời gọi này liên kết ngữ cảnh biểu diễn OpenGL với ngữ cảnh dụng cụ. Sau đó, OnSize() gọi GlMatrixMode() với đối số GL_PROJECTION để báo cho OpenGL biết chương trình thao tác ma trận chiếu. Lời gọi glLoadIdentity() khởi tạo ma trận chiếu theo ma trận đơn vị. Cuối cùng, glFrustum() được gọi để định nghĩa không gian quan sát, glViewport() thiết lập viewport theo kích thước của sổ client, và wglMakeCurrent() với đối số NULL làm cho RC hết hiện hành.

Quyết định đúng các đối số cho glFrustum() không phải là việc dễ dàng. Do đó, thư viện OpenGL utility cung cấp hàm gluPerpective(), cho phép định nghĩa không gian quan sát theo 1 cách trực giác hơn:

```
gluPerpective(viewAngle, aspectRatio, nearPlane,farPlane);
```

Bốn đối số của hàm là các giá trị Gldouble. Đối số đầu tiên là cho biết góc nhìn, là góc tạo thành giữa các đường thẳng từ mắt đến đỉnh và đáy mặt phẳng cắt gần(hình 6.14). Góc này càng lớn, các mặt phẳng cắt gần và xa càng cao. Đối số thứ hai cho biết tỉ lệ thể hiện của không gian quan sát. Khi giá trị này bằng 1, thì không gian quan sát có chiều rộng bằng chiều cao. Hai đối số cuối cùng là các giá trị dương, cho biết khoảng cách từ điểm nhìn đến các mặt phẳng cắt gần và xa.

Để giữ cho viewport và không

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

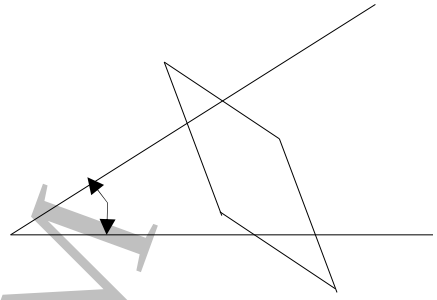
gian quan sát có cùng tỉ lệ thể hiện,
dù người dùng thay đổi kích thước cửa sổ,

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

đoạn mã sau được dùng đáp ứng thông báo



Hình 6.14 Góc nhìn:

```
WM_SIZE :
        CClientDC clientDC(this);
        wglMakeCurrent(clientDC.m_hDC,m_hRC);

        glMatrixMode( GL_PROJECTION);
        glLoadIdentity();
        GLdouble aspectRatio=( GLdouble) cx /( GLdouble) cy;
        gluPerspective(40.0,aspectRatio,3.5,10.0);
        glViewport(0,0,cx,cy);
        wglMakeCurrent(NULL,NULL);
```

Khi dùng đoạn mã trên thiết lập không gian quan sát và viewport, hình vuông sẽ luôn luôn vuông cho dù người dùng thay đổi kích thước cửa sổ. Tuy nhiên, hình sẽ nhỏ đi khi chiều cao cửa sổ bị giảm.

6.5.*Tổng kết:*

Với OpenGL, khi xác định vị trí đa giác, điều quan trọng là định nghĩa các vertex của đa giác đối-diện-mặt-trước theo chiều ngược chiều kim đồng hồ, trừ khi sự thay đổi chiều được thực hiện một cách rõ ràng với hàm glFrontFace().

OpenGL thực hiện biến hình trên các vertex của đối tượng(trước khi thể hiện đối tượng) theo thứ tự :biến hình đối tượng, biến đổi phép chiếu, và biến đổi viewport. Phép biến hình đối tượng thì được xác định trên ma trận đối tượng, còn phép chiếu và biến đổi viewport thì được xác định trên ma trận chiếu. Khi gọi các hàm tác động đến ma trận, trước hết phải gọi glMatrixMode() để chọn đúng ma trận.

OpenGL có các hàm tạo không gian quan sát, tổ hợp biến hình. Chúng ta không phải thao tác trực tiếp hoán ma trận.

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

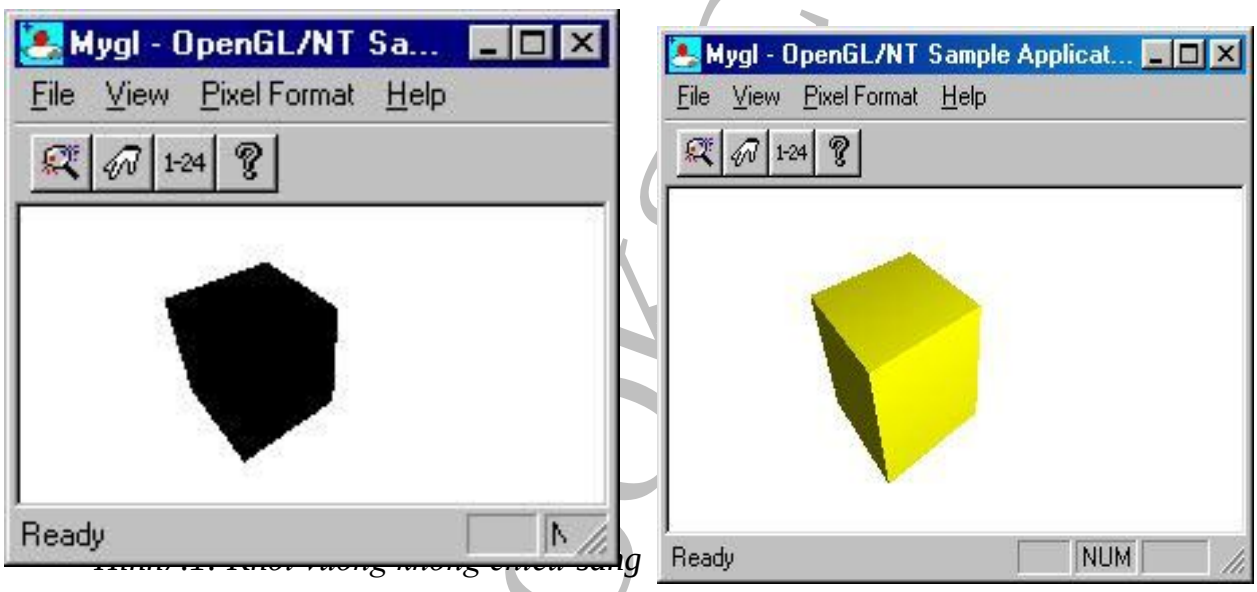
Chương 7:Chiếu Sáng Đối Tượng

ɔ mục trong chương này gồm:

- Định nghĩa nguồn sáng.
- Tạo các loại ánh sáng môi trường, khuếch tán, phản chiếu, và ánh sáng phát.
- Định nghĩa tính chất vật liệu.
- Định nghĩa vector trực giao cho vertex đa giác.
- Tạo bảng màu logic.

7.1. Các Loại Nguồn Sáng:

Ánh sáng cung cấp màu sắc và tạo ánh sáng cho đối tượng 3-D, làm hình ảnh nhận được trở nên thực hơn (hình7.1,7.2).



Hình 7.1: Khối vuông không chiếu sáng

OpenGL có bốn loại nguồn sáng: môi trường, khuyếch tán, phản chiếu, vá nguồn phát. Anh 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 chối hơn, sau đó ánh sáng bị khuếch tán đi mọi hướng. Anh 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 cảnh. Đối tượng sẽ phụ thuộc vào ánh sáng mà nó phản xạ. Ví du khi thiết lập bề mặt đố tượng phản xạ ánh sáng xanh dương, OpenGL sẽ thể hiện đối tượng với các bóng xanh dương khác nhau (giả sử ánh sáng chiếu vào đối tượng là ánh sáng trắng).

7.2.Định Nghĩa Một Nguồn Sáng:

OpenGL Cho phép có tám nguồn sáng khác nhau trong một chương trình. mỗi nguồn sáng có nhiều thuộc tính để kiểm soát các ánh sáng tác động đến một cảnh. Các ánh sáng

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

bao gồm ánh sáng môi trường, ánh sáng khuếch tán, ánh sáng phản chiếu và ánh sáng trí. Cần xát định các thành phần phần trăm màu đỏ, xanh dương, xanh lục trong các thuộc tính ánh sáng môi trường, khuếch tán,và phản chiếu; Đồng thời phải xác định các tọa độ X,Y,Z và W của nguồn sáng. Công việc này được quản lý bằng các mảng chứa các giá trị yêu cầu.

7.2.1.Thiết Lập Các Mảng Giá Trị Anh Sáng:

Các mảng sau chứa các 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};
Glfloat positionLight0[] =  {0.0f, 0.0f, 1.0f, 0.0f};
```

Trong mảng ambientLightO[], lượng màu được thiết lập cho ánh sáng môi trường gồm 30% đỏ và 30% xanh dương, 30% xanh lục. (Giá trị 1.0 lá thành phần alpha, có thể bỏ qua). Mảng diffuselightO[], 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. (Các giá trị RBGA từ 0.0 đên1.0).

Mảng specularar Light0[] chứa các giá trị của ánh sáng trắng tối nhất (hay không có tí ánh sáng nào)

Cuối cùng, mảng positionLight0[] chứa vị trí nguồn sáng, xác định bởi các tọa độ X,Y,Z, và W. Trong đó, W chỉ có hai giá trị 1và 0. Nếu W bằng 0,ta có nguồn sáng theo hướng, là nguồn sáng ở khoảng cách vô hạn so với cảnh, tức các tia sáng đến đối tượng là song song với nhau. Mặt trời là ví dụ cho nguồn sáng vô hướng.Nếu W bằng 1, ta có nguồn sáng theo vị trí , là nguồn sáng có vị trí gần cảnh, nên các tia sáng chiếu vào đối tượng theo các góc khác nhau. Đèn là nguồn sáng theo vị trí.

7.2.2 .Chuyển Mảng Cho OpenGL:

Các giá trị định nghĩa nguồn sáng được chuyển cho OpenGL bằng cách gọi hàm glLight() như sau:

```
glLightfv (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, và W cửa nguồn sáng.

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

Do một số chương trình OpenGL có thể có tám nguồn sáng, đối số đầu tiên là hằng từ GL_LIGHT0 đến GL_LIGHT7

Đối số thứ hai là một trong mười hằng số ở bảng7.1. Các hằng số thường dùng với các hàm này là GL_AMBIENT,GL_DIFFUSE, GL_SPECULAR,và GL_POSITION.

Bảng 7.1: Hằng số tính chất cho hàm glight()

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

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

_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 khoảng 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.

Các phiên bản khác của hàm glLight() gồm glLight(), glLighti(), và glLightiv() được dùng để thiết lập các tính chất khác của nguồn sáng. Ví dụ, có thể dùng một trong các hàm này (hay nhiều hàm cùng lúc) để định nghĩa đèn chiếu (spotling) trong cảnh. Sau này ta sẽ thấy đèn chiếu cho phép tạo một nguồn sáng trực tiếp với chùm tia hẹp.

7.2.3.Kích Hoạt Nguồn Sáng:

Khi đã có định nghĩa nguồn sáng, ta phải kích hoạt chúng, tương tự như việc bậc 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 vertex cứ đa giác. Dĩ nhiên, nếu muốn biến đổi vị trí và hướng nguồn sáng theo các đối tượng khác trong cảnh, phải thực hiện các phép tịnh tiến, co giãn, và quay trước khi gọi glLight().

7.3.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:

7.3.1.Màu Và Anh 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 xanh lục, thì nó phản xạ màu xanh lục đối với ánh sáng môi trường lẫn ánh sáng khuếch tán. Trong khi ánh sáng phản chiếu hầu như 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 xanh dương). Nếu muốn khối vuông có màu đỏ, vật liệu của nó phải phản xạ màu đỏ của

KILOBOOKS.COM
Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

ı sáng mội trường và ánh sáng khuếch tán. Để làm khối vuông trở nên sáng hơn, thì ıi 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.

7.3.2.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.

```
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.

7.3.3. 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():

```
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 7.2 Các hằng xác định tính chất vật liệu. 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 7.2: Các hằng tính chất dùng cho hàm glMaterialfv()

Hằng số	Y nghĩa
GL_AMBIENT	Anh sáng môi trường bi ánh xạ
GL_AMBIENT_AND_DIFFUSE	Anh sáng môi trường và khuếch tán bị ánh xạ
GL_COLOR_INDEXES	Xác định các chỉ số màu cho ánh sáng
GL_DIFFUSE	Anh sáng khuếch tán bị phản xạ

KILOBOOKS.COM
Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

_EMISSION	Xác định nguồn sánh phát.
_SHININESS	Xác định độ bóng vật liệu.
GL_SPECULAR	Anh 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 đên128 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.

7.4.Định Nghĩa Các Pháp Tuyến:

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

7.4.1.Pháp Tuyến Và Vertex:

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 tiến của các vertex một đa giác được định nghĩa bởi lời gọi hàm Normal3f() trước lời gọi glVertex() định nghĩa vertex:

```
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 vertex được định nghĩa trong đoạn mã trên sẽ tồn tại cho đến khi glNormal() được gọi lại để định nghĩa pháp tuyến mới.

7.4.2. Tính Toán Pháp Tuyến :

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

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

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

```
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 doumle. 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 mã sau 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;
    }

}
```

7.5. Xác Định Kiểu Bóng Và Kích Hoạt Việc Kiểm Tra Chiều Sâu:

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

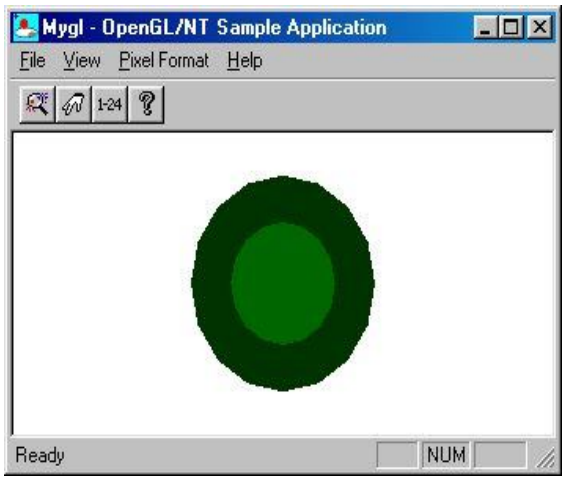
http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

Một phương diện quan trọng của OpenGL là kiểu bóng dùng thể hiện đối tượng ng cảnh. Có hai kiểu bóng: mượt (smooth) và phẳng (flat). Để chọn kiểu bóng, sử dụng ...n glShadeModel() như sau:

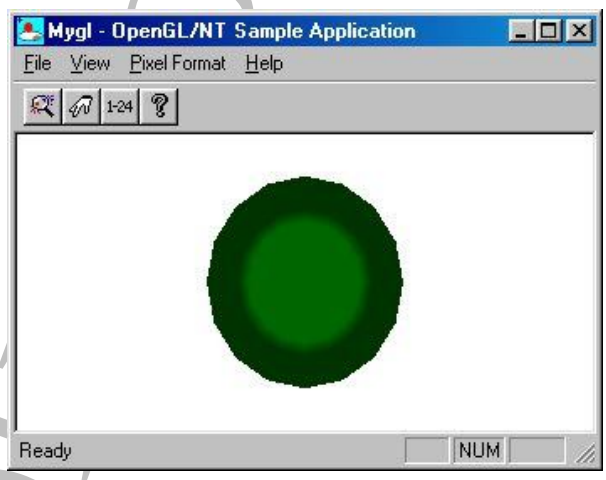
```
glShadeModel (GL_FLAT);
```

Đối số đơn của hàm là hằng thể hiện kiểu bóng được chọn, lá GL_SMOOTH, hoặc GL_FLAT. Với bóng mượt, OpenGL phải xác định màu bóng cho từng pixel trên đa giác. Trong khi đối với bóng phẳng, toàn bộ đa giác dùng chung một màu bóng.

Hình 7.4 Hình cầu với bóng phẳng



Hình 7.5 Hình cầu với bóng mượt



Để vẽ và tô bóng cho các đa giác được chính xác, OpenGL cần biết vị trí tương đối của mỗi đa giác trong cảnh. Hiển nhiên là các đa giác nằm dưới các đa giác khác sẽ không được thể hiện. Việc xác định vị trí trước, sau của đa giác được gọi là kiểm tra chiều sâu. Để thực hiện, phải kích hoạt việc kiểm tra bằng hàm glEnable():

```
glEnable (GL_DEPTH_TEST);
```

Nếu việc kiểm tra không được thực hiện, OpenGL sẽ gặp khó khăn trong việc xác định vẽ đa giác nào và bỏ qua đa giác nào, dẫn đến kết quả không như ý.

Tuy vậy, chỉ kích hoạt thôi thì cũng chưa đủ. Do OpenGL sử dụng một bộ đệm riêng cho việc kiểm tra chiều sâu, ta phải xóa nó trước khi thể hiện cảnh:

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

Đối số đơn của hàm là giá trị có được từ phéo OR giữa các hằng GL_COLOR_BUFFER_BIT và GL_DEPTH_BUFFER_BIT. (Phép toán OR so sánh từng bit tương ứng giữa hai toán hạng; Nếu cả hai bít bằng 1, bít kết quả bằng 1; Trái lại, bít kết quả bằng 0). Như vậy, hàm sẽ xóa cùng lúc cả hai bộ đệm màu và chiều sâu.

7.6. Định Nghĩa Đèn Chiếu:

Đèn chiếu cho 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(W bằnh một trong lời gọi positionLight(), xác định một góc cắt cho chùm tia, định nghĩa vị trí và hướng của nguồn. Đoạn mã sau định nghĩa một đèn chiếu:

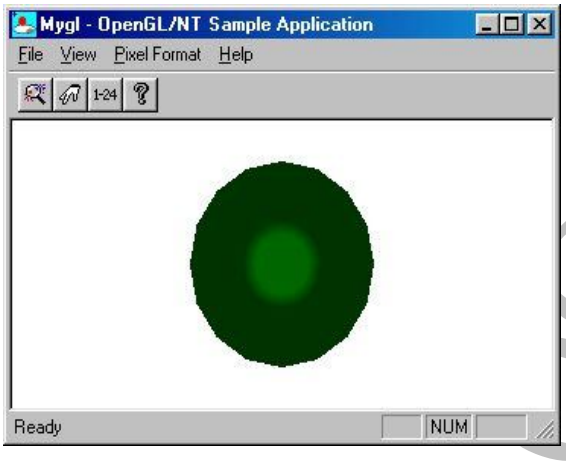
```
Glfloat ambientLight1[] = {0.2f, 0.2f, 0.2f, 1.0f};
Glfloat diffuseLight1[]  = {1.0f, 1.0f, 1.0f, 1.0f};
```

Nếu bạn cần file word? Xin hãy nhấp vào đây!

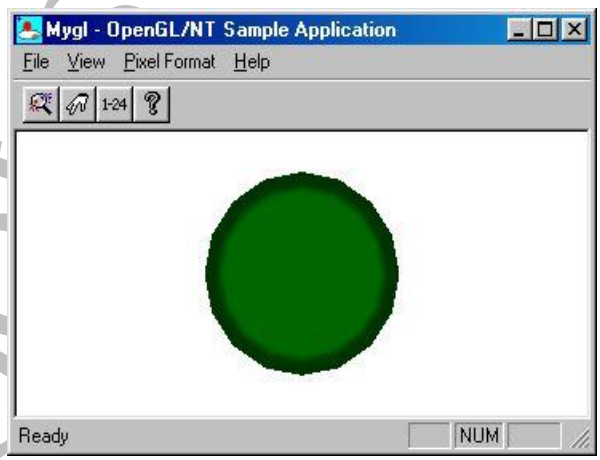
http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

```
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);
glLightfv (GL_LIGHT1, GL_SPOT_CUTOFF, 10.0f);
glLightfv (GL_LIGHT1, GL_SPOT_DERECTION, derectionLight1);
```

Trong đoạn mã trên, lời gọi glLight (GL_LIGHT!, 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.



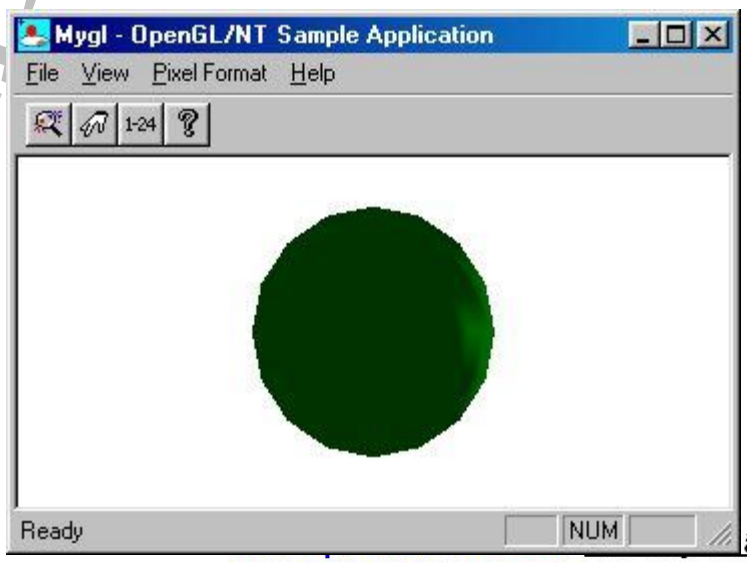
Hình7.6: chùm tia hẹp trên hình cầu



Hình 7.7: chùm tia rộng hơn trên hình cầu

Sự khác nhau tiếp theo của đèn chiếu so với các nguồn sáng khác là lời gọi glLightfv(GL_LIGHT!, 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.

Hình7.8 Thay đổi tọa độ hướng chiếu trong mảng directionLight[].



ấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

7.7.Thể Hiện Đối Tượng 3-D Được Chiếu Sáng:

Việc thể hiện một cảnh OpenGL được chiếu sánh không đơn giản. Để nguồn sáng tác động đến đối tượng trong cảnh đòi hỏi phải hiểu rỏ cách làm việc của kiểu chiếu sáng OpenGL.

7.7.1. Các Hiệu Quả Của Nguồn Sáng Môi Trường:

Như đã biết, một cảnh có thể có ba loại nguồn sáng: môi trường, khuếch tán, và phản chiếu. Ngoài ra, các tính chất vật liệu đối tượng quyết định kiểu ánh sáng phản xạ từ đối tượng, cùng màu sắc và các hiệu quả bóng trên đối tượng. Ta xem xét đoạn mã sau:

```
glShadeModel (GL_SMOOTH);
glEnable (GL_DEPTH_TEST);
glClearColor (1.0f, 1.0f, 1.0f, 1.0f);
glClearColor (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

glMatrixMode (GL_MODELVIEW);
glLoadldentity();

Glfloat materialAmbient[] = {0.0f, 1.0f, 0.1f, 1.0f};
Glfloat materialSpecular[] = {0.7f, 0.7f, 0.7f, 1.0f};

glMaterialfv (GL_FRONT, GL_AMBIENT_AND_DIFFUSE, aterialAmbient);
glMaterialfv (GL_FRONT, GL_SPECULAR, materialAmbient);
glMaterialfv (GL_FRONT, GL_SHININESS,100.0f);

Glfloat ambientLight0[] = {0.7f, 0.7f, 0.7f, 1.0f};
Glfloat diffuseLight0[] = {0.0f, 0.0f, 0.0f, 1.0f};
Glfloat specularLight0[] = {0.0f, 0.0f, 0.0f, 1.0f};
Glfloat positionLight0[] = {0.0f, 0.0f, 1.0f, 0.0f};
glLightfv (GL_LIGHT0, GL_AMBIENT, ambientLightO);
glLightfv (GL_LIGHT0, GL_DIFFUSE, diffuseLightO);
glLightfv (GL_LIGHT0, GL_SPECULAR, specularLightO);
glLightfv (GL_LIGHT0, GL_POSITION, positionLightO);
glEnable (GL_LIGHTING);
glEnable (GL_LIGHT0);
glTranslatef (0.0f, 0.0f, -3.5f);
auxSolidSphere (1.0);
glFlush();
```

Đoạn mã trên xác định vật liệu phản xạ ánh sáng môi trường , khuếch tán xanh lục và ánh sáng phản chiếu trắng. Nguồn sáng trong cảnh thì không được thiết lập, ngoại trừ ánh

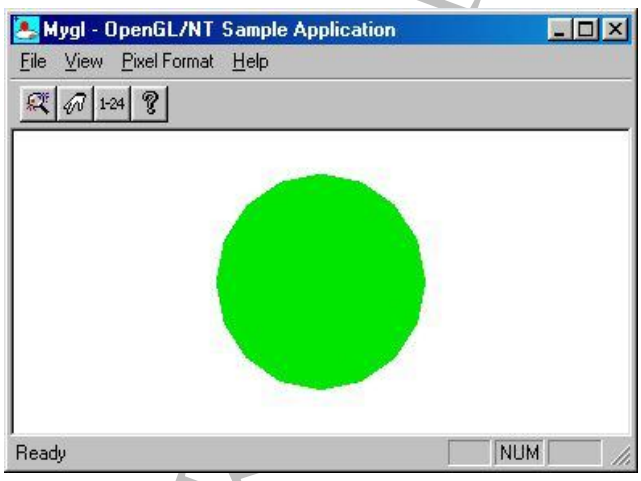
KILOBOOKS.COM
Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

g môi trường trắng. Cuối cùng, đối tượng là hình cầu đặc , được vẽ tại gốc tọa độ bởi n auxSolidSphere(). Hàm này có đối số là bán kính hình vầu, và nó tự tính toán pháp ến.

Hình 7.9 thể hiện đoạn mã trên. Chú ý rằng hìng cầu chỉ giống một hình tròn đặc. Đó là do nó chỉ phản xạ ánh sáng môi trường, là loại ánh sáng đến từ mọi hướng. Trong khi các đa giác tạo nên hình cầu phản xạ cùng một lượng ánh sáng. Hay nói cách khác, chỉ riêng ánh sáng môi trường không cung cấp bóng để tạo cảm giác ba chiều.

Hình 7.9 Hình cầu chỉ với ánh sáng môi trường:



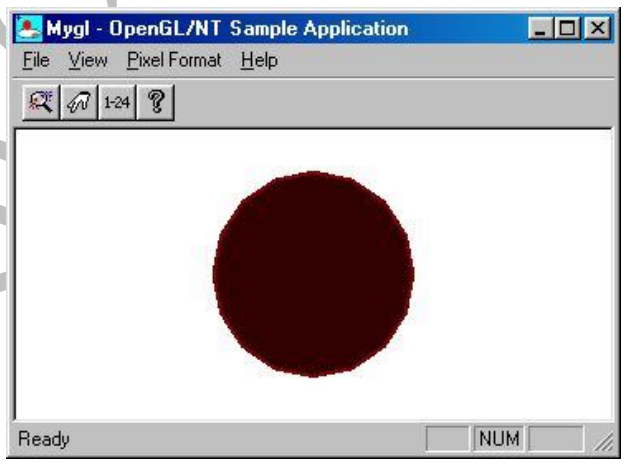
7.7.2.Các Hiệu Quả Của Nguồn Sáng Khuếch Tán:

Thay đổi đoạn mã trên để cảnh chỉ có đoạn mã khuếch tán:

```
Glfloat diffuseLight0[]  = {0.0f, 0.0f, 0.0f, 1.0f};
Glfloat ambientLight0[] = {0.7f, 0.7f, 0.7f, 1.0f};
Glfloat specularLight0[] = {0.0f, 0.0f, 0.0f, 1.0f};
Glfloat positionLight0[] = {0.0f, 0.0f, 1.0f, 0.0f};
```

Do ánh sáng khuếch tán đến từ một hướng xác định, các tia sáng chiếu vào đối tượng theo các góc khác nhau, nên đường đi của các tia sáng đến các đa giác (tạo nên hình cầu) khác nhau, tạo hiệu quả bóng cho hình 7.10:

Hình 7.10 Hình cầu chỉ với ánh sáng khuếch tán:



7.7.3.Các Hiệu Quả Của Nguồn Sáng Phản Chiếu:

KILOBOOKS.COM
Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

Xét cảnh chỉ có nguồn sáng phản chiếu:

```
Glfloat diffuseLight0[]  = {0.0f, 0.0f, 0.0f, 1.0f};
Glfloat ambientLight0[] = {0.7f, 0.7f, 0.7f, 1.0f};
Glfloat specularLight0[] = {0.0f, 0.0f, 0.0f, 1.0f};
Glfloat positionLight0[] = {0.0f, 0.0f, 1.0f, 0.0f};
```

Do cảnh không có ánh sáng môi trường hay khuếch tán, hình cầu tối màu hầu hết diện tích (hình 7.11). Điểm sáng trên màn hình là ánh sáng phản chiếu, tạo nên một "điểm nóng" trên bề mặt hình cầu.

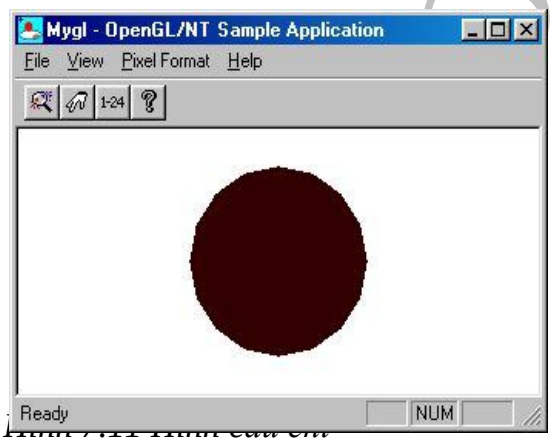
7.7.4.Hiệu Quả Anh Sáng Tổng Hợp:

Để tạo một cảnh sinh động, các loại ánh sáng của nguồn sáng, cũng như các loại ánh sáng phản xạ từ bề mặt đố tượng, cùng vị trí nguồn sáng cần được cân nhắc kỹ. Đoạn mã sau thể hiện hình 7.12:

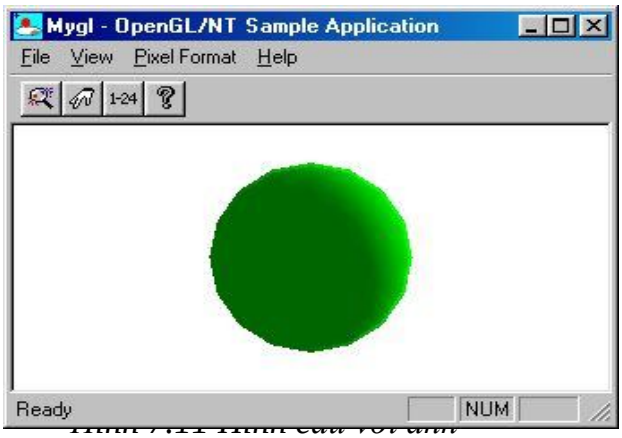
```
Glfloat ambientLight0[] = {0.2f, 0.2f, 0.2f, 1.0f};
Glfloat diffuseLight0[]  = {0.7f, 0.7f, 0.7f, 1.0f};
Glfloat specularLight0[] = {1f, 1f,1f, 1.0f};
Glfloat positionLight0[] = {1.0f, 0.5f, 1.0f, 0.0f};
Glfloat materialAmbient[] = {0.0f, 1.0f,0.0f, 1.0f};
Glfloat materialSpecular[] = {1.0f, 1.0f,1.0f, 1.0f};
```

7.8.*Bảng Màu Logic:*

OpenGL thực sự được thiết kế cho hệ thống thể hiện 64000 màu hoặc hơn. Trên hệ thống 256 màu, OpenGL gặp khó khăn trong việc thể hiện chính xát màu cần có. Lý do là trên hệ thống 256 màu, Windows dành riêng mảng màu chỉ gồm 20 màu để vẽ desktop, trong khi OpenGL không thể vẽ chính xác các đối tượng 3-D có bóng chỉ với 20 màu.



với ánh sáng phản chiếu



sáng môi trường, khuếch tán vá phản chiếu

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

Như vậy, để có kết quả ngoạn mục, giải pháp tốt nhất là dùng Windows trên các hệ thống hỗ trợ nhiều người dùng hơn. Tuy nhiên, cũng còn một cách để cung cấp cho OpenGL số màu vừa đủ sử dụng trên hệ thống 256 màu. Đó là thiết lập bảng màu logic.

7.8.1.Tạo Bảng Màu Logic:

Mọi ứng dụng phải định nghĩa màu để có bảng màu logic của riêng nó. Khi người dùng đổi ứng dụng, ứng dụng mớ sẽ chuyển mảng màu logic của nó cho Windows để Windows thiết lập màu thích hợp. Như vậy, bảng màu logic là tập hợp màu liên kết với một ứng dụng riêng biệt. Trong khi bảng màu hệ thống chứa các màu đang được thể hiện trên màn hình. Khi chuyển đổi ứng dụng, bảng màu logic của ứng dụng mới được thể hiện trên bảng màu hệ thống. Nghĩa là có thể có nhiều mảng màu logic, nhưng chỉ có một mảng màu hệ thống.

Xét đoạn mã sau định nghĩa một cấu trúc chứa thông tin Windows cần để tạo bảng màu logic:

```
Struct
{
   WORD Version;
   WORD NumberOfEntries;
   PALETTEENTRY aEntries [256];

}logicalPalette = {0x300, 256};
```

Các dòng trên thể hiện cấu trúc LOGPALETTE, một kiểu dữ liệu được định nghĩa bởi Winsdows và được sử dụng khi quản lý các bảng màu logic. Thành phần đầu tiên của cấu trúc là số phiên bản, ở đây là giá trị thập lục phân 0x300. Thành phần thư hay là số màu trong bảng màu. Ở đây, là 256 màu. Thành phần cuối cùng là mảng màu cấu trúc PALETTEENTRY.Windows định nghĩa cấu trúc

PALETTEENTRY như sau:

```
Typedef struct
{
   BYTE peRed;
   BYTE peGreen;
   BYTE peBlue;
   BYTE peFlgs;
}PALETTETRY;
```

Các thành phần peRed, peGreen, peBlue của cấu trúc chứa cường độ các màu đỏ, xanh lục, và xanh dương. Thành phần peFlags xác định cách Windows quản lý entry (mục ghi trong mảng màu), có thể có các giá trị PC_EXPLICT, PC_NOCOLLAPSE, PC_RESERVED, hay 0.

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

Cờ PC_EXPLICT cho biết entry chứa một chỉ số về bảng màu hệ thống, thay vì chứa giá trị màu thực sự. Chỉ số này được lưu trữ từ thấp của entry. Rất hiếm khi dùng cờ PC_EXPLICT.

Cờ PC_NOCOLLAPSE cho biết Windows sẽ đưa màu vào entry rỗng của bảng màu hệ thống, thay vì map nó và entry hiện có. Nếu như không có các entry rỗng trong bảng màu hệ thống, Windows bỏ qua cờ PC_NOCOLLAPSE.

Cờ PC_RESERVED ngăn Windows sử dụng màu của các ứng dụng khác. Cờ này thường dùng với bảng màu thường xuyên thay đổi màu.

Khi để Windows tư quản lý entry theo cách mà nó thấy phù hợp, thì giá trị 0 được thiết lập cho thành phần peFlags.

Với cấu trúc LOGPALETTE đã định nghĩa, có thể tạo bảng màu chứa một khoảng màu rộng như ví dụ sau:

```
BYTE Reds[] = {0,36,72,109,145,182,218,255};
BYTE Greens[] = {0,36,72,109,145,182,218,255};
BYTE Blues[] = {0,85,170,255};

For (int colorNum[]=0; colorNum<256; ++colorNum)
   {
      logicalPalette.aEntries[coloeNum].peRed =
            reds[coloeNum& 0x07];
      logicalPalette.aEntries[coloeNum].peGreen =
            greens[(coloeNum >> 0x03) & 0x07];

      logicalPalette.aEntries[coloeNum].peBlue =
            blues[(coloeNum >> 0x06) & 0x03];

      logicalPalette.aEntries[coloeNum].peFlags =0;
   }
```

Trong vòng lập for, chương trình thực hiện một số thao tác bit để tính toán các chỉ số (theo các mảng red[], green[], và blue[]) chứa các cường độ màu để điền vào mảng màu.

Cuối cùng, bảng màu được tạo bằng lời gọi hàm Windows API CreatePalette().

7.8.2.Lựa Chọn ,Thực Hiện, Và Xóa Bỏ Bảng Màu Logic:

Trước khi vẽ lên màn hình, ứng dụng phải lựa chọn bảng màu vào ngữ cảnh dụng cụ rồi thực hiện bảng màu. Thực hiện bảng màu tức là báo cho Windows lấy bảng màu logic để thể hiện bảng màu hệ thống.

Trước tiên, phải gọi hàm Windows API SelectPalette() ví dụ:

```
SelectPalette (pDC->m_hDC, m_hPalette, FALSE);
```

Hàm SelectPalette(0 chọn bảng màu vào ngữ cảnh dụng cụ.Ba đối số của nó là handle theo DC, handle theo bảng màu, và một giá trị Boolean là true biểu thi bảng màu luôn

KILOBOOKS.COM
Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

n là bảng màu nền (backgroundpalette), hay FALSE biểu thị bảng màu có thể là bảng u cận cảnh (foreground palette). Thông thường, giá trị FALSE được dùng.

Tiếp theo, hàm WindowsAPI Realizepalette() được gọi để thực hiện bảng màu:

```
RealizePalette (pDC ->m_hDC);
```

Hàm này có đối số đơn là handle của ngữ cảnh dụng cụ mà bảng màu của nó được thực hiện.

Sau đó, có thể tạo thể hiện OpenGL, ví dụ:

```
WglMakeCurrent (pDC ->m_hDC, m_hRC);
DrawWithOpenGL();
WglMakeCurrent (pDC ->m_hDC, NULL);
```

Cuối cùng, trước khi kết thúc, phải xóa bảng màu logic đã tạo, ví dụ:

```
If (m_hPalette)
DeleteOject (m_hPalette);
```

Đối số đơn của deleteOject() là handle theo đối tượng cần xóa.

7.8.3.Kiểm Tra Khi Nào Cần Tạo Bảng Màu Logic:

Một ứng dụng có thể có nhiều ứng dụng khác nhau. Nhưng bảng màu chỉ cần tạo ra khi hệ thống đòi hỏi. Như vậy, cần kiểm tra định dạng điểm vẽ được lựa chọn theo yêu cầu bảng màu của hệ thống.

Sau khi thiết lập định dạng điểm vẽ, cấu trúc PIXELFORMATDESCRIPTOR được điền đầy bằng các giá trị định dạng diểm vẽ hiện hành. Lời gọi hàm như sau:

```
DescribePixelformat (clientDC.m_hDC, pixelformat, sizeof (pfd), &pfd);
```

Nếu thành phần dwFlags của cấu trúc PIXELFORMATDESCRIPTOR có chứa cờ PFD_NEED_PALETTE, bảng màu logic sẽ được tạo cho ứng dụng. Đoạn mã kiểm tra như sau:

```
If(pfd.dwFlags & PFD_NEED_PALETTE)
    SetupLogicalPalette();
```

7.8.4.Đáp Ứng Các Thông Báo Bảng Màu:

Khi một ứng dụng được kích hoạt, nó có thể thực hiện bảng màu logic của riêng nó. Điều này có thể dẫn đến sự thay đổi lớn màu sắc trong bảng hệ thống, trở nên xa lạ với các ứng dụng nền. Giải pháp cho vấn đề này là phải cho các ứng dụng nền biết về sự thay đổi của bãng màu hệ thống, bằng thông báo WM_PALETTECHANGED.

Khi ứng dụng nền nhận được thông báo này, nó thực hiện bảng màu logic của nó, tức báo cho Windows thể hiện lại bảng màu logic của nó theo bảng màu hệ thống. Do ứng dụng đang có tác dụng có quyền ưu tiên trước hết về màu sắc trên bảng màu hệ thống, nên dù có` sự đáp ứng đối với WM_PALETTECHANGED, sự thể hiện của các ứng dụng nền

KILOBOOKS.COM
Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

thể bị mất tính toàn vẹn. Tuy nhiên Windows cố gắng cho sự hài hòa giữa các bảng u logic của ứng dụng nền với bảng màu hệ thống mới.

Khi một nền được kích hoạt lại, nó có thể đòi quyền ưu tiên trên bảng màu hệ thống. Windows báo cho ứng dụng biết cơ hội này bằng thông báo WM_QUERYNEWPALETTE. Ứng dụng trả lời thông báo bằng cách thực hiện bảng màu logic của nó, thể hiện toàn bộ màu sắc của bảng màu logic
vào bảng màu hệ thống. Tại thời điểm này, các ứng dụng khác, nay là ứng dụng nền, nhận thông báo WM_PALETTECHANGED, rồi thể hiện lại bảng màu logic của chúng theo bảng màu hệ thống mới.

7.9. Tổng kết:

OpenGL cung cấp bốn loại nguồn sáng: môi trường khuếch tán, phản chiếu, và nguồn phát. Thể hiện của đối tượng trên màn hình phụ thuộc vào loại nguồn sáng có trong cảnh và cách phản xạ ánh sáng của đối tượng. Như vậy để chiếu sáng một cảnh, cần định nghĩa một hay nhiều nguồn sáng, và định nghĩa tính chất vật liệu của đối tượng.

Để OpenGL có thể xác định góc ánh sáng chiếu vào đối tượng cần định nghĩa pháp tuyến của các vertex trên tất cả đa giác tạo nên đối tượng. Một số hàm trong thư viện OpenGL auxiliary, cung cấp pháp tuyến của đa giác mà nó tạo ra. Để tính toán pháp tuyến của đa giác bất kỳ, ta dùng hàm CalcNormal() đã nói ở trên.

Một thể hiện OpenGL là tốt nhất khi được dùng trên hệ thống 64.000 màu hoặc hơn. Khi chạy trên hệ thống 256 màu, do bảng màu 20 màu của Windows cung cấp quá ít màu cho thể hiện OpenGL, nên ứng dụng cần tạo, chọn lựa và thực hiện bảng màu logic. Ngoài ra để thể hiện của nó được cập nhật chính xác ứng dụng phải đáp ứng các thông báo WM_PALETTECHANGED và WM_QUERYNEWPALETTE.

Chương 8:Tạo Cảnh 3D

Các chương trước đã cung cấp hầu hết các dung cụ để tạo bất cứ cảnh 3-D nào, nhưng việc tổ hợp một cảnh từ hàng chục, hàng trăm, hay hàng ngàn đa giác không phải là việc đơn giản. Ngay cả khi sử dụng các hình 3-D tạo sẵn như khối vuông, hình chóp và hình cầu của thư viện OpenGL auxiliary, cũng cần phải thực hiện các phép biến hình để đặt chúng vào đối tượng thể hiện.

Chương này trình bày:

- Gốc tọa độ cục bộ và gốc tọa độ thế giới.
- Tổ hợp các cảnh từ các đối tượng 3-D.
- Sử dụng stack ma trận.
- Sử dụng bộ đệm đôi để sinh động hóa cảnh 3-D.

8.1.Sử Dụng Các Phép Biến Hình OpenGL Để Tạo Cảnh 3-D:

Chúng ta đã biết cách cung cấp các phép biến hình bao- gồm tịnh tiến, co giãn, và quay cho đối tượng. Nhưng chỉ mới là các đối tượng đơn lẻ. Chứ chưa giải quyết việc sắp đặt nhiều đối tượng trên màn hình và giữ cho chúng một tổ chức logic khi thực hiện biến hình. Đây là một trong những vấn đề khó trong đồ họa.

8.1.1. Gốc Tọa Độ Cục Bộ Và Gốc Tọa Độ Thế Giới.

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

Có hai phương pháp để tưởng tượng về cách mà phép biến hình đối tượng (modelview transformation) OpenGL tác động lên cảnh 3-D như thế nào: theo gốc tọa độ cục bộ và gốc tọa độ thế giới, hay theo hệ tọa độ cố định.

Ở chương "Đồ họa hai chiều GDI" và "Đồ họa ba chiều GDI", ta đã lập trình trên một hệ tọa độ cố định. Tức là khi cung cấp phép quay cho một mô hình, ta cho rằng toàn bộ hệ như là một tấm nhựa quay quanh một trục trung tâm, và mọi đối tượng vẻ trên tấm nhựa đó thì quay theo nó.

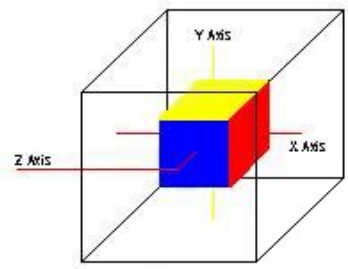
Khi tổ hợp các đối tượng nhỏ vào đối tượng lớn, thì cách suy nghĩ về các phép biến hình theo các gốc tọa độ cục bộ và thế giới sẽ đơn giản hơn. Gốc tọa độ thế giới (world origin) luôn luôn là gốc tọa độ đề các ba chiều. Đó là điều hoàn toàn hợp lý vì hệ đề các 3-D là thế giới chứa các cảnh được vẽ. Như vậy khi định nghĩa các vị trí trong hệ đề các 3-D, các tọa độ được dùng là các tọa độ thế giới (world coordinate), có liên quan đến gốc tọa độ thế giới.

Trong khi đó gốc tọa độ cục bộ (locol origin) được đặt tâm đối tượng và di chuyển cùng đối tượng. Với hệ thống này có thể tượng việc xây dựng các đối tượng 3-D là thay vì di chuyển toàn bộ thế giới trước khi đặt các phần của đối tượng vào vị trí, thì ta chỉ đặt chúng vào vị trí trong một thế giới tĩnh.

Giả sử khởi tạo ma trận modelview theo ma trận đơn vị như sau :

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
```

Tại đây, nếu gọi auxSolidCube(),OpenGL sẽ vẽ một khối vuông 3-D có tâm trùng với tâm đề các 3-D (hình 8.1). (Khi vẽ khối vuông, OpenGL nhân các vertex khối vuông với ma trận moldelview, hiện là ma trận đơn vị).



Hình 8.1
Khối vuông vẽ ở khối Tọa độ thế giới.

Hiển nhiên là không có việc vẽ một tá đối tượng tại gốc tọa độ thế giới, mà các đối tượng phải được đặt ở các vị trí khác nhau để tạo nên các hình có ý nghĩa. Một cách để thực hiện điều đó là tịnh tiến, tức di chuyển đối tượng với gốc tọa độ cục bộ của nó đến vị trí mới.

8.1.2.Phép Tịnh Tiến:

Giả sử glTranslatef() được gọi để thêm phép tịnh tiến vào ma trận modelview như sau:

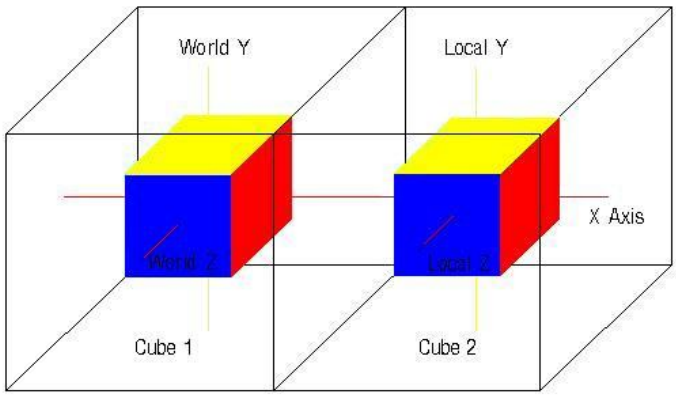
```
glTranslatef(1.0f, 0.0f, 0.0f);
```

KILOBOOKS.COM
Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

Nếu gọi auxSolidCube() để vẽ khối vuông thứ hai, thì khối vuông đó sẽ xuất hiện ở vị cách khối vuông thứ nhất một đơn vị phía bên phải, do ma trận modelview chứa phép ...h tiến một đơn vị trên trục X. Điều này giống như di chuyển gốc tọa độ cục bộ của khối vuông một đơn vị về phía bên phải (hình 8.2).

Hình 8.2
Khối vuông được vẽ sau khi tịnh tiến một đơn vị theo chiều dương

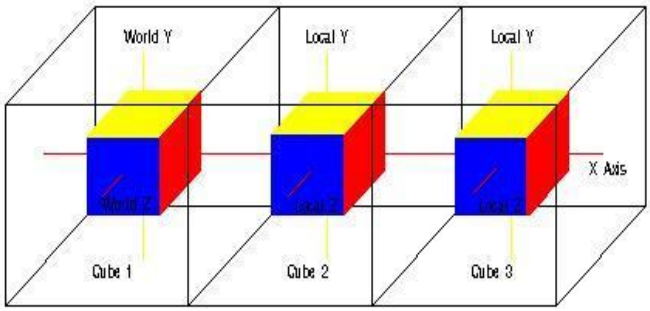


Nếu tiếp tục thực hiện phép tịnh tiến:

```
glTranslatef(1.0f, 0.0f, 0.0f);
```

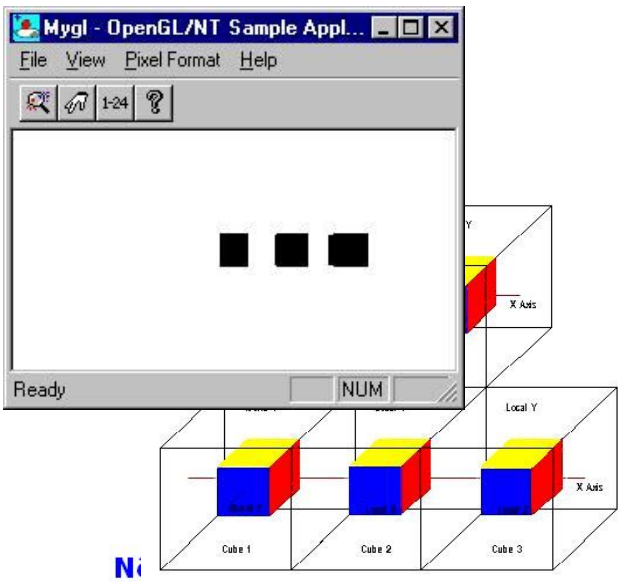
rồi vẽ thêm một khối vuông bằng lời gọi auxSolidCube(), Kết quả nhận được sẽ như hình 8.3. Tại sao phép tịnh tiến cuối cùng chỉ là một đơn vị theo chiều dương trục X, mà OpenGL lại vẽ khối vuông thứ hai cách gốc tọa độ thế giới hai đơn vị ? Lý do là mỗi phép biến hình được gọi đều là bộ phận của ma trận modelview hiện hành kiểu như các lệnh MoveTo().

Hình 8.3
Khối vuông được vẽ sau lời gọi phép tịnh tiến thứ hai.



Đoạn mã sau thể hiện như hình 8. ...ca để thấy được các khối vuông):

```
auxSolidCube(0.5f);
glTranslatef(1.0f, 0.0f, 0.0f);
auxSolidCube(0.5f);
glTranslatef(1.0f, 0.0f, 0.0f);
auxSolidCube(0.5f);
```



Hình 8.4: Thể hiện OpenGL của ba khối vuông

Tất nhiên là có thể tịnh tiến theo ...

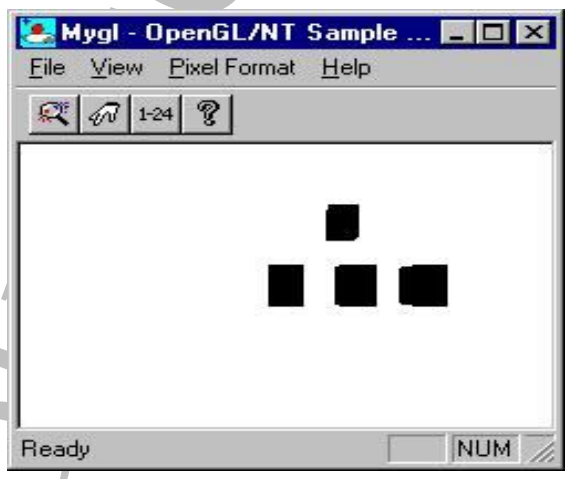
Ni... ...p vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

ɔi hướng, trên mọi trục. Ví dụ đoạn mã sau cho hình 8.5, có thể hiện OpenGL như hình
ɔ:

```
glTranslatef(-1.0f, 1.0f,-1.0f);
auxSolidCube(0.5f);
```

Hình 8.5:Thêm phép tịnh tiến và khối vuông thứ tư

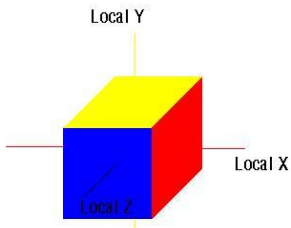


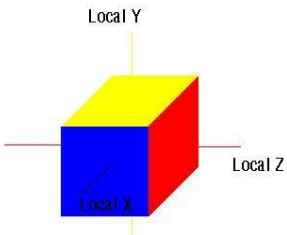
Hình 8.6:OpenGL của bốn khối vuông

8.1.3.Phép Quay:

Tác dụng của phép tịnh tiến thì dễ hiểu. Nhưng đối với phép quay thì không như vậy, vì phép quay thay đổi hướng các trục cục bộ của đối tượng. Ví dụ xét khối vuông trong hình 8.7 . Sau khi quay 90^0 trên trục Y, hướng của khối vuông như hình 8.8, nghĩa là các trục X và Z đã đổi hướng cho nhau. Nghĩa là nếu muốn tịnh tiến khôi vuông sang bên phải màn hình, ta phải tịnh tiến nó theo hướng dương trục Z, chứ không phải theo hướng dương trục X.

Hình 8.7:Khối vuông trước khi quay





rd? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

Hình 8.8: Khối vuông sau khi quay 90^0 trên trục Y

Điều cần nhớ là sau khi được thực hiện, phép quay tiếp tục tồn tại trong ma trận modelview cho đến khi ma trận này được khởi tạo lại. Xét đoạn mã sau:

```
glTranslatef(1.0f, 0.0f, 0.0f);
glRotatef(90.0f, 0.0f, 1.0f, 0.0f);
auxSolidCube(0.5f);
glTranslatef(0.0f, 0.0f, 1.0f);
auxSolidCube(0.5f);
```

Dòng đầu tiên tịnh tiến gốc tọa độ cục bộ một đơn vị theo chiều dương trục X (tức sang phải, giả sử điểm nhìn lúc đầu nhìn theo hướng âm trục Z). Sau đó dòng thứ hai quay gốc tọa độ cục bộ quanh trục Y, rồi dòng thứ ba vẽ khối vuông. Lúc này, để vẽ khối vuông thứ hai bên phải khối thứ nhất, ta phải tịnh tiến theo trục Z chứ không phải trục X như ở dòng thứ tư. Do khối vuông thư hai quay tương tự khối vuông thứ nhất, nên trục Z của nó sẽ chạy từ trái sang phải.

Trong trường hợp hệ tọa độ cố định, việc sử dụng các phép biến hình phải được hiểu theo cách ngược lại với trường hợp hệ tọa độ cục bộ. Ví dụ, khi sử dụng hệ tọa độ cục bộ, nếu gọi glTranslatef() để tịnh tiến theo trục dương trục X, rồi gọi glRotatef() để quay quanh trục Z, thì hình đã vẽ vẫn nằm trên trục X sau khi quay. Nhưng nếu glRotatef() được gọi trước, thì hình sẽ nằm giữa các trục X và Y. Còn đối với hệ tọa độ cố định thì sao ?

Hãy tưởng tượng một tấm nhựa trong với khối vuông bằng gỗ tại gốc tọa độ. Trên mặt bàn phía dưới tấm nhựa, ta vẽ các trục X và Y.(hay nói cách khác, khi tấm nhựa quay quanh trục Z tại gốc tọa độ, các trục X,Y vẫn cố định). Trước tiên, tấm nhựa được quay quanh gốc tọa độ cố định. Sau khi quay khối vuông vẫn ở góc tọa độ. Bây giờ phép tịnh tiến được thực hiện bằng cách trượt tấm nhựa sang phải. Khối vuông di chuyển cùng với tấm nhựa nhưng vẫn trên trục X. Nếu gọi glTranslatef() trước glRotatef(), tấm nhựa di chuyển sang phải, rồi mới quay quanh gốc tọa độ cố định trên bàn. Như vậy, khối vuông quay cùng với tấm nhựa sẽ nằm đâu đó giữa các trục X và Y.

(Trong trường hợp gốc tọa độ cục bộ, ta không di chuyển tấm nhựa, mà chỉ di chuyển khối vuông.)

8.2.Sử Dụng Các Stack Ma Trận:

Sẽ thật là hữu ích nếu có khả năng thiết lập một ma trân modelview làm ma trận nền, dùng nó để thực hiện các phép biến hình, rồi trả lại ma trận nền, theo trạng thái ban đầu của nó. Ví dụ ta muốn thiết lập điểm nhìn lúc bắt đầu chương trình, sau đó không phải quan tâm đến nó nữa. Nhưng mọi phép biến hình được thực hiện sau khi thiết lập điểm nhìn sẽ thay đổi ma trận modelview, mà ma trận này chứa phép biến đổi điểm nhìn. Có nghĩa là phải khởi tạo lại điểm nhìn ở lần thay đổi cảnh kế tiếp.

KILOBOOKS.COM
Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

Nhưng OpenGL có cách để lưu trữ và hòan trả nôi dung một ma trận, đó là sử dụng các ck ma trận. Stack ma trận thì tương tự như stack mà máy tính sử dụng, nhưng chứa các g.i trị cho các ma trận liên quan, thay vì các kiểu dữ liệu dùng cho stack máy tính. Mỗi ma trận biến hình có stack ma trận riêng. Để lưu nội dung ma trận hiện hành, chỉ đơn giản thực hiện lời gọi hàm sau:

```
glPushMatrix();
```

Sau khi đẩy ma trận hiện hành vào phía trên stack ma trận, ta có thể thực hiện mọi phép biến hình mà không lo ma trận gốc bị biến đổi. Khi muốn trở lại ma trận gốc, lờ gọi sau được thực hiện:

```
glPopMatrix();
```

Cả hai hàm trên đều không có đối số. Chúng luôn luôn tác động đến ma trận hiện được chọn. Ví dụ các dòng sau lưu nội dung ma trân modelview:

```
glMatrixMode(GL_MODELVIEW);
glPushMatrix();
```

Còn các dòng sau lưu nội dung ma trận chiếu:

```
glMatrixMode(GL_PROJECTION);
glPushMatrix();
```

Sử dụng các thao tác stack ma trận trên, ta dễ dàng thiết lập một ma trận modelview cơ sở để định nghĩa một điểm nhìn, và trả nó về khi cần thiết. Khi bắt đầu chương trình (trong chương trình MFC là hàm OnCreate()), các lời gọi hàm như sau:

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glFrustum(-1.0, 1.0, -1.0, 2.0, 9.0);
glViewport(0, 0, cx, cy);

glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslatef(0.0f, 0.0f ,-5.0f);
```

Bốn dòng đầu tiên thiết lập không gian quan sát và viewport. Trong khi ba dòng sau định nghĩa điểm nhìn bằng cách di chuyển “mắt “ 5 đơn vị theo chiều âm trục Z. (Thực ra là di chuyển ra xa mắt gốc tọa độ cục bộ của bất kỳ đối tượng nào được vẽ).

Để vẽ cảnh mà không làm thay đổi nôi dung ma trận modelview (là ma trận đang chứa phép biến hình thiết lập điểm nhìn), thì phải lưu nó trước khi thực hiện bất kỳ phép biến đổi nào:

```
glMatrixMode(GL_MODELVIEW);
glPushMatrix();
glTranslatef(1.0f, 0.0f ,0.0f);
glRotatef(90.0f, 0.0f, 1.0f, 0.0f);
auxSolidCube(0.5f);
glTranslatef(0.0f, 0.0f ,1.0f);
auxSolidCube(0.5f);
glPopMatrix();
```

Đọan mã trên lưu ma trận modelview, thực hiện các phép biến hình cần thiết để vẽ đối tượng, sau đó trả lại ma trận modelview, xóa mọi phép biến hình đã thực hiện sau lời gọi

KILOBOOKS.COM
Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

PushMatrix(). Ở lần thực hiện biến hình tiếp theo, phép biến hình sẽ không ảnh hưởng i các phép biến hình trước. Như vậy ta có thể thấy stack ma trận có giá trị lớn lao khi ạo ảo giác chuyển động OpenGL, vì khi đó các phép biến hình tương tự nhau sẽ được gọi lập đi lập lại với các giá trị khác nhau.

Mỗi loại ma trận có stack ma trận riêng. Chẳng hạn ma trận modelview có stack riêng, và ma trận chiếu có stack riêng. Khi gọi glPushMatrix(), thực chất là bản sao của ma trận hiện tại được đẩy vào đỉnh stack ma trận của riêng nó, và ta có hai ma trận đơn vị trên đỉnh stack. Khi gọi glPopMatrix(), OpenGL loại bỏ bản sao, lúc này có thể đã bị thay đổi, và để lại ma trận gốc trên đỉnh stack.

Đoạn mã sau đây thể hiện một người máy như hình 8.9:

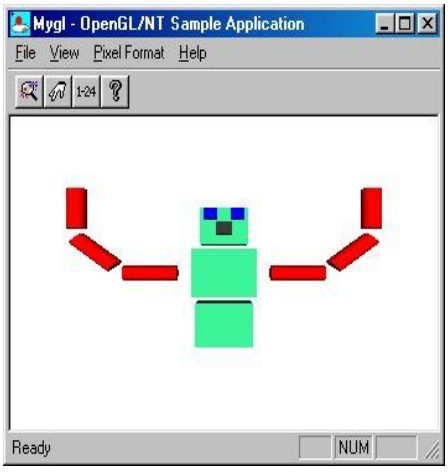
```
GLfoat materialAmbDiffGreen[]={0.2f, 0.8f, 0.5f, 1.0f};
GLfoat materialAmbDiffRed[]={1.0f, 0.0f, 0.0f, 1.0f};
GLfoat materialAmbDiffGray[]={0.2f , 0.2f, 0.2f, 1.0f};
GLfoat materialAmbDiffBlue[]={0.0f, 0.0f, 1.0f, 1.0f};

//Save the base tranformatin.
    glMatrixMode(GL_MODELVIEW);
    glPushMatrix();

  //Draw the robot's body
    glMaterialfv(GL_FRONT,GL_AMBIENT_AND_DIFFUSE,
  materialAmbDiffGreen);
    auxSolidCube(1.0f);
    glTranslatef(0.0f, -1.1f ,0.0f);
    auxSolidCube(0.9f);

  //Draw the robot's head
    glTranslatef(0.0f, 2.1f ,0.0f);
    auxSolidCube(0.75f);
    glMaterialfv(GL_FRONT,GL_AMBIENT_AND_DIFFUSE,
  materialAmbDiffGray);

    glTranslatef(0.0f, -0.1f ,0.6f);
    auxSolidCube(0.25f);
    glTranslatef(-0.2f, 0.3f ,0.0f);
    glMaterialfv(GL_FRONT,GL_AMBIENT_AND_DIFFUSE,
  materialAmbDiffBlue);

    auxSolidCube(0.2f);
    glTranslatef(0.4f, 0.0f ,0.0f);
    auxSolidCube(0.2f);

        //save the robot's transformation.
    glPushMatrix();
```

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

```
    //Draw the robot's right arm.
        glMaterialfv(GL_FRONT,GL_AMBIENT_AND_DIFFUSE,
    materialAmbDiffRed);
        glTranslatef(-0.7f, -1.2f ,0.0f);
        glRotatef(90.0f, 0.0f ,0.0f, 1.0f);
        auxSolidCylinder(0.15f, 0.8f);

        glTranslatef(0.0f, 1.0f ,0.0f);
        glRotatef(-45.0f, 0.0f ,0.0f, 1.0f);
        auxSolidCylinder(0.15f, 0.8f);

        glTranslatef(0.0f, 1.0f ,0.0f);
        glRotatef(-45.0f, 0.0f ,0.0f, 1.0f);
        auxSolidCylinder(0.15f, 0.8f);
//restore the body transformation.
glPopmatrix();
    //Draw the robot's left arm.
        glTranslatef(0.3f, -1.2f ,0.0f);
        glRotatef(-90.0f, 0.0f ,0.0f, 1.0f);
        auxSolidCylinder(0.15f, 0.8f);

        glTranslatef(0.0f, 1.0f ,0.0f);
        glRotatef(45.0f, 0.0f ,0.0f, 1.0f);
        auxSolidCylinder(0.15f, 0.8f);

        glTranslatef(0.0f, 1.0f ,0.0f);
        glRotatef(45.0f, 0.0f ,0.0f, 1.0f);
        auxSolidCylinder(0.15f, 0.8f);
//restore the base transformation.
glPopMatrix();
```



Hình 8.9: Người máy OpenGL

Đoạn mã bắt đầu bằng việc định nghĩa các mảng dùng cho vật liệu đối tượng trong cảnh. Ma trận modelview được lựa chọn. Sau đó bản sao của nó được đẩy vào phía trên stack ma trận modelview. Như vậy lúc này trên stack có hai bản sao ma trận biến đổi hiện hành (chứa phép biến đổi điểm nhìn để dùng ở mọi chỗ trong chương trình).

Kế tiếp đoạn mã gọi glmaterialfv() để thiết lập các tính chất vật liệu, và vẽ hai khối vuông thể hiện thân người máy. Khối vuông thứ hai được đặt dưới khối vuông thứ nhất bằng phép tịnh tiến.

KILOBOOKS.COM
Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

Để vẽ đầu robot, trước tiên đoạn mã tịnh tiến theo phương Y, rồi vẽ khối vuông n khối vuông thứ nhất của phần thân mắt và mũi được vẽ sau khi thiết lập các tính chất ..t liệu và thực hiện phép tịnh tiến.

Đến đây, stack ma trận modelview chứa ma trận gốc (là ma trận chỉ có phép biến đổi điểm hình), và trên đỉnh của nó là bản sao ma trận gốc đã được thay đổi (chứa phép biến đổi điểm nhìn và phép biến hình khác). Do các tay người máy được vẽ tương quan với thân, nên đoạn mã gọi glPushMatrix( ) để lưu trạng thái hiện tại của phép biến đổi. Như vậy, stack ma trận modeliview bây giờ chứa một ma trận gốc, và hai bản sao phép tịnh tiến hình người máy.

Tiếp theo việc vẽ cánh tay trái bắt đầu bằng hàm glMaterialfv( ) để xác định tính chất vật liệu một phép tịnh tiến và một phép quay thiết lập ma trận modelview để vẽ đoạn thứ nhất tại một gốc chính xác với thân và lời gọi auxSolidCylinder( ) thực sự vẽ đoạn tay này. Các đoạn còn lại của tay trái được vẽ với phép tịnh tiến và quay tương tự.

Sau khi hoàn tất việc vẽ cánh tay trái, lời gọi glPopMatrix ( ) loại bỏ ma trận đang chứa các phép biến hình dùng cho cánh tay trái ra khỏi stack ma trận, khôi phục lại trạng thái ma trận sau khi vừa vẽ song thân và đầu người máy. Rồi đoạn mã vẽ tiếp cánh tay phải. Cuối cùng glPopMatrix ( ) được gọi để stack ma trận chỉ còn chứa ma trận gốc.

8.3. Tạo Ao Giác Chuyển Động Với OpenGL:

Các cảnh 3- D sẽ trở nên sống động khi bộ đệm đôi được dùng. Và trong trường hợp này, stack ma trận đặt biệt hữu dụng.

8.3.1. Giới Thiệu Bộ Đệm Đôi:

Việc thiết kế ảo giác chuyển động nhịp nhàng luôn là sự thách thức đối với người lập trình. Trước tiên, chương trình phải có khả năng vẽ 30 hình (hay "khung") trong một giây, để tạo sự chuyển động uyển chuyển. Điều này đòi hỏi phần cứng có tốc độ cao và chương trình được thiết kế tốt. Kế tiếp, mọi việc phải vẽ khuất, và mỗi khung hình phải tạo được sự chuyển động đối với mắt người dùng. May thay, vấn đề tốc độ ngày càng trở nên ít quan trọng khi máy tính ngày càng nhanh hơn. Còn giải pháp cho vấn đề thứ hai là sử dụng bộ đệm đôi .

Với bộ đệm đôi, hai vùng nhớ được sử dụng để tạo ảo giác chuyển động. Một vùng chứa ảnh trên màn hình. Trong thời gian vùng bộ đệm này đang được thể hiện, không lệnh vẽ nào được dùng để sửa đổi nó. Vùng thứ hai được sử dụng để chương trình vẽ vào, nhưng không được thể hiện trên màn hình. Nên người dùng không thấy các thao tác vẽ. Sau khi việc vẽ hoàn tất, chương trình chỉ đơn giản hoán đổi các bộ đệm.

8.3.2. Bộ Đệm Đôi Trong OpenGL:

Trong OpenGL, các bộ đệm trong bộ đệm đôi được gọi là bộ đệm trước (bộ đệm không thấy được). Khi thiết lập bộ đệm đôi OpenGL, mọi lệnh vẽ sẽ luôn luôn đến bộ đệm sau hiện hành, như vậy người dùng sẽ không thấy thao tác vẽ. Khi chương trình vẽ xong ảnh, nó hoán đổi bộ đệm trước và sau để làm xuất hiện ảnh mới trên màn hình. Sau đó chương trình tạo khung hình kế tiếp trong bộ đệm sau mới, rồi lại hoán đổi các bộ đệm. Qúa trình này tiếp diễn trong suốt toàn bộ thời gian ảo giác chuyển động được thực hiện, và việc hoán đổi giữa các bộ đệm được thực hiện nhiều lần trong giây.

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

8.3.3. Sử Dụng Bộ Đệm Đôi:

Mặc dù việc quản lý bộ đệm đôi là kỹ thuật khó trong lập trình đồ họa thông thường OpenGL thực hiện điều đó chỉ với hai thay đổi nhỏ trong chương trình. Trước tiên OpenGL phải được thông báo thiết lập môi trường bộ đệm đôi. Việc này được thực hiện bằng cách thêm cờ PFD_DOUBLEBUFFER vào thành phần dwFlags của cấu trúc PIELFORMATDESCRIPTOR như sau:

```
PIXELFORMATDESCRIPTOR pfd =
{
        sizeof(PIXELFORMATDESCRIPTOR),  // Kích thước cấu trúc.
        1,                      //Số version của cấu trúc.
        PFD_DRAW_TO_WINDOW  |                   // các cờ thuộc tính
      PFD_SUPPORT_OPENGL |
      PFD_DOUBLEBUFFER,
        PFD_TYPE_RGBA,
        24,                     // 24 bit màu
        0, 0, 0, 0, 0, 0,       // không chọn lựa các chế độ.
         0,0,0, 0, 0, 0,0,0
        32                      // bộ đệm chiều sâu 32 bit
        0,0,                    // không dùng bộ đệm stencil hay bộ đệm phụ trợ
      PFD_MAIN_PLANE,       //kiểu lớp chủ yếu
        0,
        0, 0, 0
};
```

Như vậy, ta chỉ đơn giản sử dụng phép toán OR cờ PFD_DOUBLEBUFFER với các cờ khác trong thành phần cấu trúc dwFlags.

Một thay đổi khác cần có lời gọi hàm SwapBuffers() sau khi chương trình vẽ xong hình mới. Lời gọi này đổi bộ đệm sau (là bộ đệm vừa được vẽ vào) thành bộ đệm trước, và như vậy người dùng thấy được hình mới. Còn bộ đệm trước trở thành bộ đệm sau, và mọi lệnh vẽ trước lời gọi SwapBuffers() kế tiếp sẽ có tác dụng với nó.

8.4.Tổng Kết:

Việc sắp xếp các hình dạng vào một cảnh 3-D chỉ là vấn đề sử dụng các phép tịnh tiến và quay để định vị từng đối tượng.

Để dễ dàng tưởng tượng về phép biến hình yêu cầu, thì ta dùng các gốc tọa độ thế giới và cục bộ. Điều quan trọng phải ghi nhớ mỗi phép biến hình được thực hiện sẽ trở thành một phần của ma trận modelview hiện hành. Các phép biến hình này tiếp tục có tác dụng cho đến khi ma trận bị xóa, bằng lời gọi glLoadIdentity() hay bằng cách khôi phục ma trận được lưu giữ trong stack ma trận.

Để tạo một cảnh chuyển động, OpenGL dùng bộ đệm đôi. Trong đó, bộ đệm trước chứa cảnh đang được thể hiện trên màn hình, còn bộ đệm sau chứa cảnh đang được vẽ. Việc hoán đổi giữa các bộ đệm sẽ chuyển cảnh mới ra màn hình.

Chương 9: Anh Và Gán Cấu Trúc:

KILOBOOKS.COM
Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

Trong lập trình đồ họa thường phải giải quyết các ảnh có nhiều chi tiết hơn là ta có thể với OpenGL. Ví dụ như mặc dù có thể một hình chữ nhật thể hiện mặt bàn, nhưng đa giác được điền đầy không có các chi tiết như là thớ gỗ, để có thể thể hiện đó là mặt bàn. Để cung cấp các chi tiết như vậy, ta sử dụng các ảnh OpenGL để thực hiện gán cấu trúc (texture mapping) cho các đối tượng.

Nhưng trước khi tìm hiểu texture mapping, ta cần biết về các ảnh OpenGL và cách sử dụng các bitmap độc - lập - với - thiết- bị (DIBs, device – independent bitmaps). Hai kiểu dữ liệu đồ họa này cho phép tạo các hình ảnh bằng cách sao chép các hình chữ nhật từ bộ nhớ, hay vẽ chúng bằng một chương trình vẽ. Một khi có các ảnh trong bộ nhớ, ta có thể sử dụng chúng để nâng cấp các cảnh 3-D.

Chương trình này trình bày:

- Bitmap và ảnh OpenGL.
- Làm thế nào để nạp bitmap độc - lập - với - thiết bị.
- Bảng danh sách màu và các phương thức chuyển dời pixel.
- Cách thiết lập texture mapping OpenGL.
- Cách thực hiện texture mapping một đa giác

9.1.Bitmap và ảnh OpenGL.

OpenGL hỗ trợ hai loại đối tượng đồ họa. Loại đầu tiên là bitmap, là một mask đơn sắc dùng tạo cá thể đơn giản như các ký tự trong một font. Loại thứ hai là ảnh, là bitmap đầy màu sắc, và có thể chứa đựng một hình tượng hay một bức tranh đầy đủ chi tiết.

Không may là các loại máy khác nhau lưu trữ ảnh theo các cách khác nhau. Do là thư viện chuẩn, OpenGL phải có cách hỗ trợ bất cứ kiểu lưu trữ hình ảnh nào, nên nó định nghĩa một số hằng kiểu pixel như ở bảng 9.1.

Bảng 9.1: Các hằng số kiểu pixel OpenGL

Hằng số	Ý nghĩa
GL_ALPHA	Mỗi pixel là một phần tử màu alpha
GL_BLUE	Mỗi pixel là một phần tử màu xanh dương
GL_COLOR_INDEX	Mỗi pixel là một chỉ số màu
GL_DEPTH_COMPONENT	Mỗi pixel là một phần tử chiều sâu
GL_GREEN	Mỗi pixel là một phần tử màu xanh lá
GL_LUMINANCE	Mỗi pixel là một phần tử luminance
GL_LUMINANCE_ALPHA	Mỗi pixel là một phần tử màu alpha và luminance
GL_RED	Mỗi pixel là một phần tử màu đỏ
GL_RGB	Mỗi pixel là một phần tử màu đỏ, xanh lá, xanh lục
GL_RGBA	Mỗi pixel là một phần tử màu đỏ, xanh lá, xanh lục,và alpha
GL_STENCIL_INDEX	Mỗi pixel là một chỉ số stencil

Ngoài ra, kích thước dữ liệu thể hiện một phần tử ảnh đơn thì thay đổi giữa các ảnh và giữa các máy, nên OpenGL định nghĩa một số hằng kiểu dữ liệu như ở bảng 9.2.

Bảng 9.2: Các hằng số kiểu dữ liệu OpenGL

KILOBOOKS.COM
Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

ing số	Ý nghĩa
_BITMAP	Số nguyên 8-bit không dấu chứa dữ liệu bit đơn
_BYTE	Số nguyên 8-bit có dấu
GL_FOAT	Dấu chấm động độ chính xác đơn
GL_INT	Số nguyên 32-bit có dấu
GL_SHORT	Số nguyên 16-bit có dấu
GL_UNSIGNED_BYTE	Số nguyên 8-bit không dấu
GL_UNSIGNED_INT	Số nguyên 32-bit không dấu
GL_UNSIGNED_SHORT	Số nguyên 16-bit không dấu

9.1.1.Đọc Anh OpenGL:

OpenGL cung cấp ba hàm cơ bản để thao tác dữ liệu ảnh: glReadPixel(), glDrawPixel(), và glCopyPixels(). Hàm glReadPixel() sao chép hình chữ nhật từ bộ đệm màu và lưu trữ dữ liệu trong bộ nhớ. Ví dụ các dòng sau định nghĩa kho chứa ảnh và đọc dữ liệu ảnh từ bộ đệm màu hiện hành vào kho:

```
Glubyte image[64][64][3];
    glReadPixels(0, 0, 64, 64, GL_RGB, GL_UNSIGNED_BYTE, image)
```

Đoạn mã trên định nghĩa kho như mảng ba chiều lưu trữ ảnh 64x64 gồm ba thành phần pixel, trong trường hợp này là các giá trị RGB. Hàm glReadPixels() đọc dữ liệu ảnh từ bộ đệm màu và lưu trữ trong mảng image[][][]. Bảy đối số của hàm là các tọa độ X, Y của góc trái dưới hình chữ nhật, chiều rộng và chiều cao của hình chữ nhật, kiểu pixel và kiểu dữ liệu của pixel được đọc, cuối cùng là địa chỉ lưu trữ dữ liệu ảnh. Lời gọi trên đọc một ảnh 64x64 từ góc trái dưới của bộ đệm màu. Anh này được thể hiện bởi các phần tử màu đỏ, xanh lá, và xanh lục lưu trữ như các byte không dấu.

9.1.2.Vẽ Anh OpenGL:

Một khi đã có dữ liệu ảnh trong bộ nhớ, có thể dùng hàm glDrawPixel() để thể hiện dữ liệu lên màn hình. Anh đã lưu bằng đoạn mã trước được thể hiện với lời gọi như sau:

```
glDrawPixels(64, 64, GL_RGB, GL_USIGNED_BYTE, image );
```

Hàm glDrawPixels() thể hiện ảnh tại vị trí raster hiện tại. Năm đối số của hàm là chiều rộng và chiều cao ảnh, kiểu pixel và kiểu dữ liệu của dữ liệu ảnh và địa chỉ ảnh.

Để kiểm soát vị trí raster mà tại đó glDrawPixel() thể hiện dữ liệu ảnh, ta dùng hàm glRasterPos3f() như sau:

```
glRasterPos3f(1.0f, 1.0f, 0.0f);
```

Hàm này thiết lập vị trí raster hiện hành bằng ba đối số X, Y, Z. Các giá trị này không phải là các tọa độ màn hình như trong hàm glReadPixels(), mà la các tọa độ Đề các được biến đổi sang tọa độ màn hình bởi các phép biến hình đối tượng và phối cảnh. glRasterPos() có 24 phiên bản.

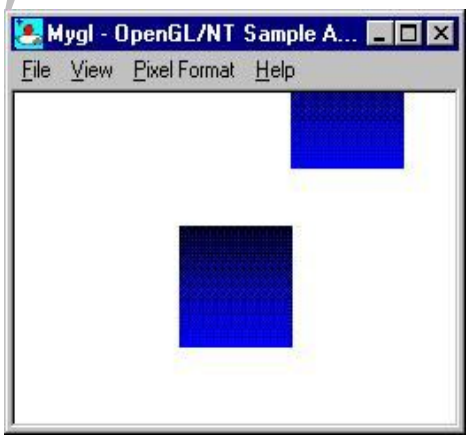
9.1.3.Tìm Hiểu Thao Tác Anh:

KILOBOOKS.COM
Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

Đoạn mã sau tạo một ảnh RGB trong bộ nhớ, thể hiện ảnh ở góc dưới trái cửa sổ, sao ép ảnh vào một vùng nhớ khác, và thể hiện bản sao ở góc trên phải cửa sổ như hình 9.1:

```
Glubyte image1[64][64][3];
Glubyte image2[64][64][3];

for (int i = 0; i < 64; i++)
   for(int j = 0; j < 64; j++)
   {
      image1[i][j][0]=(GLubyte) 0;
      image1[i][j][1]=(GLubyte) 0;
      image1[i][j][2]=(GLubyte) (255-i*3.5);
   }
glRasterPos3f(-1.0f,- 1.0f, 0.0f);
        glDrawPixels(64, 64, GL_RGB, GL_UNSIGNED_BYTE, image1);
Glint rasPos[4];
glGetIntegerv(GL_CURRENT_RASTER_POSITION, rasPos);
        glReadPixels(rasPos[0],rasPos[1],64,64,GL_RGB,
                     GL_UNSIGNED_BYTE, image2);
        glRasterPos3f(1.0f, 1.0f, 0.0f);
        glDrawPixels(64, 64, GL_RGB, GL_UNSIGNED_BYTE, image2);

        glFlush();
```



Hình 9.1: đọc và vẽ các ảnh OpenGL

Vòng lập for trong đoạn mã trên tạo một ảnh từ các đường thẳng xanh dương xếp từ sáng tại đáy đến đậm ở đỉnh. (Anh được lưu trữ đảo ngược trong bộ nhớ, nên các byte đầu tiên trong mảng sẽ ở đáy ảnh). Sau khi tạo ảnh, đoạn mã thiết lập vị trí raster và vẽ ảnh tại vị trí mới.

Để sao chép ảnh, chương trình cần biết các tọa độ cửa sổ của ảnh. Nhưng glRastePos3f() sư dụng các tọa độ đề các. Vậy làm thế nào để biết các tọa độ đề các được biến đổi sang các tọa độ màn hình ra sao? Điều đó được thực hiện nhờ hàm glGetInteger() với đối số là hằng GL_CURRENT_RASTER_POSITION và địa chỉ của mảng bốn phần tử, mà ở đó hàm lưu trữ vị trí raster hiện hành trong tọa độ cửa sổ.

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

Sau khi các tọa độ cửa sổ của ảnh, đoạn mã đọc ảnh vào mảng thứ hai, image[][][]. Sau thiết lập lại vị trí raster và thể hiện ảnh mới, chứng tỏ bản gốc thực sự được sao chép. ...t nhiên giả thuyết rằng không gian quan sát và điểm nhìn được thiết lập đúng đắn. Nếu không ta sẽ thấy gì trên màn hình.

9.1.4.Sao Chép Anh OpenGL:

Hàm glCopyPixels() cho phép sao chép ảnh từ nơi này sang nơi khác trên màn hình mà không cần trước đó phải đọc ảnh vào bộ đệm. Có thể nói lời gọi glCopyPixels() tương tự như lời gọi cả hai hàm glReadPixels() và glDrawPixels(), mà không đòi hỏi kho chứa phụ trợ. Nếu sử dụng glCopyPixels(), đoạn mã trên được viết lại như sau:

```
        Glubyte image1[64][64][3];

for (int i = 0; i < 64; i++)
   for(int j = 0; j < 64; j++)
   {
      image1[i][j][0]=(GLubyte) 0;
      image1[i][j][1]=(GLubyte) 0;
      image1[i][j][2]=(GLubyte) (255-i*3.5);
      image1[i][j][3]=(GLubyte) 1;
   }
glRasterPos3f(-1.0f,- 1.0f, 0.0f);
         glDrawPixels(64, 64, GL_RGB, GL_UNSIGNED_BYTE, image1);
   Glint rasPos[4];
glGetIntegerv(GL_CURRENT_RASTER_POSITION, rasPos);
         glRasterPos3f(1.0f, 1.0f, 0.0f);
         glCopyPixels(rasPos[0], rasPos[1], 64, 64, GL_COLOR);

         glFlush();
```

Phiên bản này ngắn hơi và yêu cầu bộ nhớ ít hơn. Hàm glCopyPixels() có các đối số là các tọa độ cử sổ X, Y của hình chữ nhật được sao chép, chiều rộng và chiều cao của hình chữ nhật, và kiểu dữ liệu sao chép. Kiểu dữ liệu sao chép có thể là GL_COLOR, GL_DEPTH, hoặc GL_STENCIL. Trong trường hợp GL_COLOR, OpenGL sao chép hình chữ nhật như là dữ liệu GL_RGBA hay GL_COLOR_INDEX, tùy thuộc chế độ màu hiện hành của hệ thống. Trong các trường hợp khác, OpenGL sao chép hình chữ nhật như là dữ liệu GL_DEPTH_COMPONENT hay GL_STENCIL_INDEX.

Như vậy xử lý ảnh OpenGL thì khá dễ. Nhưng việc thể hiện bitmap Windows thường xảy ra hơn việc thể hiện ảnh OpenGL. Do đó, phần tiếp theo là trình bày cách tải bitmap Windows vào bộ nhớ, nơi OpenGL sẽ được dùng để thao tác chúng.

9.2.Bitmap Phục Thuộc Thiết Bị Và Bitmap Độc Lập Với Thiết Bị:

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

Khi lập trình trong Windows, bitmap ở khắp nơi, do định dạng BMP là định dạng đồ a duy nhất mà Window hỗ trợ trực tiếp. Các hàm như CreateBitmap(), LoadBitmap(), etchDIBits() và BitBIt() trong lập trình Windows cho phép tạo, tải hay thể hiện bitmap lên màn hình. Trong khi không có hàm tương tự cho cá kiểu dạng đồ họa khác.

Có hai loại windows bitmap:

- Bitmap phụ thuộc thiết bị (Device dependent bitmap) là các ảnh đồ họa chỉ có thể hiện trên một loại thiết bị vật lý duy nhất. Ví dụ các hàm Windows như CreateBitmap() và LoadBitmap() tạo trong bộ nhớ một ảnh bitmap tương thích với thiết bị nhất định, thường là màn hình. Các loại bitmap như vậy đôi khi được gọi là GDI bitmap, do giao diện thiết kế đồ họa Windows có thể thao tác chúng một cách trực tiếp. Bitmap phụ thuộc thiết bị được lưu trữ mà không có bảng danh sách màu (color table), do chúng sử dụng màu của thiết bị mà chúng kết hợp. Ngoài ra, bitmap phụ thuộc thiết bị chỉ trú ngụ trong bộ nhớ, chứ không thể tọa file trên đĩa.
- Bitmap độc lập với thiết bị (Device independent bitmap, DIB) là các ảnh đồ họa có thể thể hiện trên nhiều loại thiết bị vật lý khác nhau. Các loại bitmap này mang theo bảng danh sách màu để thiết bị hiện hành sử dụng khi thể hiện chúng. Như vậy các thể hiện của bitmap độc lập với thiết bị trên các thiết bị khác nhau là như nhau. Ví dụ thể hiện của một bitmap độc lập với thiết bị trong Windows hay DOS hay OS/2 sẽ như nhau. Do bitmap độc lập với thiết bị có tính cơ động giữa các hệ thống, chúng thường có dạng file trên đĩa. Ví dụ có thể tìm thấy trong thư mục Windows, nhiều file có phần mở rộng BMP, là các bitmap độc lập với thiết bị. Có thể tọa các bitmap độc lập với thiết bị bằng cách sử dụng các loại chương trình vẽ khác nhau, kể cả Windows PaintBrush, hay sử dụng phần biên soạn bitmap của Visual C++.

9.3.Định Dạng DIB:

Dù lưu trữ trên đĩa hay trên bộ nhớ thì cấu trúc của DIB không thay đổi. Một DIB được tạo thành từ nhiều kiểu cấu trúc khác nhau nối tiếp nhau. Các kiểu cấu trúc này bao gồm BITMAPFILEHEADER, BITMAPINFO, BITMAPINFOHEADER, và RGBQUAD.

9.3.1.Cấu Trúc BITMAPFILEHEADER:

Khởi đầu một DIB file là cấu trúc BITMAPFILEHEADER, được định nghĩa bởi Windows như sau:

```
Typedef struct tagBITMAPFILEHEADER
        {
                WORD    bfType;
                DWORD   bfSize;
                WORD    bfReserved1;
                WORD    bfReserved2;
                DWORD bfOffBits;
} BITMAPFILEHEADER;
```

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

Cấu trúc này bắt đầu một file DIB trên đĩa, nhưng không cần thiết trong trường hợp :map trong bộ nhớ. Thành phần đầu tiên của cấu trúc,bfType là mã ASCII của các chữ ..i BM, cho biết file là bitmap. Trong hệ thập lục phân, dfType là 4D42. Với các giá trị khác, file không phải là bitmap. Thành phần thứ hai, bfSize là kích thước file bitmap tính theo byte. Tuy nhiên, do sự không rõ ràng trong tài liệu Windows gốc, bfSize thì không đáng tin cậy và có thể bỏ qua. Ngoài ra, thành phần bfOffBits chứa số byte từ đầu bitmap file đến dữ liệu bitmap.

Bảng 9.3 Cấu trúc BITMAPFILEHEADER

Thành phần	Kiểu dữ liệu	Mô tả
BfType	WORD	Chứa giá trị ASCII của các chữ cái BM
BfSize	DWORD	Kích thước file
BfReserved1	WORD	Bằng 0
BfReserved2	WORD	Bằng 0
BfOffBits	DWORD	Số byte từ đầu file đến dữ liệu bitmap

9.3.2.Cấu Trúc BITMAPINFO:

Theo sau cấu trúc BITMAPFILEHEADER là cấu trúc BITMAPINFO, được Windows định nghĩa như sau:

```
Typedef struct tagBITMAPINFO
        {
   BITMAPFINFOHEADER  bmiHeader;
              RGBQUAD    bmiColors[1];
        }BITMAPINFO;
```

Cấu trúc này gồm một header, thể hiện bởi cấu trúc BITMAPFINFOHEADER, và một bảng danh sách màu (color table), thể hiện bởi một mảng của cấu trúc RGBQUAD.

9.3.3.Cấu Trúc BITMAPFINFOHEADER:

Cấu trúc BITMAPFINFOHEADER được Windows định nghĩa như sau:

```
        Typedef struct tagBITMAPFINFOHEADER
        {
              DWORD    biSize;
              DWORD    biWidth;
              DWORD    biHeight;
              DWORD    biPlanes;
              DWORD    biBitCount;
              DWORD    biCompression;
              DWORD    biSizeImage;
              DWORD    biXPelsPerMeter;
              DWORD    biYPelsPerMeter;
```

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

```
    DWORD   biClrUsed;
    DWORD   biClrImportant;
} BITMAPFINFOHEADER;
```

biSize chứa kích thước của cấu trúc BITMAPFINFOHEADER, thường là 40 byte. biWidth và biHeight là chiều rộng và chiều cao bitmap theo pixel. biPlanes luôn luôn bằng 1. biBitCount biểu thị số bit trên một pixel, có thể là 1, 4, 8, hay 24, tương ứng với ảnh đơn sắc, 16 màu, 256 màu hay 16.7 triệu màu.

biCompression cho biết kiểu nén được dùng với ảnh bitmap, với 0 là không nén, 1 tương ứng kiểu nén RLE-8, và 2 tương ứng kiểu nén RLE-4. Rất hiếm khi nén DIB, mà thường là biCompression mang giá trị 0.

biSizeImage là kích thước bitmap theo byte và chỉ dùng bitmap nén. Giá trị này lưu tâm đến số byte trong mỗi dòng của bitmap phải luôn là bội số của 4. Nếu cần các byte trống sẽ được đệm vào dòng để đảm bảo điều đó. Tuy nhiên trừ khi viết chương trình tạo file DIB, ta không phải xử lý việc thêm vài dòng và sự phức tạp của mã sinh ra từ nó.

Các thành phần biXPelsPerMeter và biXPelsPerMeter chứa số pixel ngang và dọc trên mét, nhưng thường thiết lập bằng 0. biClrUsed và biClrImportant chứa tổng số màu dùng trong bitmap và số màu quan trọng trong bitmap, cũng thường thiết lập bằng 0.

Chú ý rằng các thành phần của cấu trúc BITMAPFINFOHEADER sau biBitCount thường bằng 0, nên khi đọc cấu trúc từ file, có thể bỏ qua các giá trị này. Điều cần biết là cách tính các giá trị cần thiết như số màu dùng trong ảnh và cách lưu trữ chúng trong các thành phần thích hợp để truy xuất về sau.

Bảng 9.4 Cấu trúc BITMAPFINFOHEADER

Thành phần	Kiểu dữ liệu	Mô tả
BiSize	DWORD	Kích thước cấu trúc theo byte (40 byte)
BiWidth	DWORD	Chiều rộng bitmap theo pixel
BiHeight	DWORD	Chiều cao bitmap theo pixel
BiPlanes	DWORD	Luôn luôn bằng 1
BiBitCount	DWORD	Số bit của một pixel. Là 1, 4, 8, hay 24
BiCompression	DWORD	Kiểu nén:0 =không nén, 1=RLE-8, 2=RLE-4
BiSizeImage	DWORD	Kích thước bitmap theo byte
BiXPelsPerMeter	DWORD	Số pixel ngang trên mét
BiYPelsPerMeter	DWORD	Số pixel dọc trên mét
BiClrUsed	DWORD	Số màu được dùng
BiClrImportant	DWORD	Số màu quan trọng

9.3.4.Cấu Trúc RGBQUAD:

Cấu trúc RGBQUAD được Windows định nghĩa như sau:

```
Typedef struct tagRBGQUAD
{
   BYTE    rgbBlue;
```

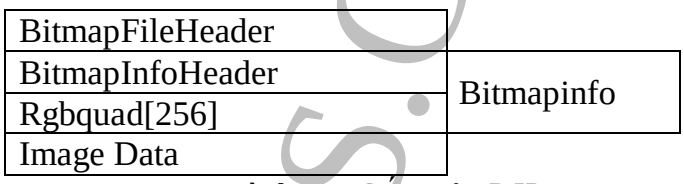
Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

```
    BYTE    rgbGreen;
    BYTE    rgbRed;
    BYTE    rgbReserve;
}RGBQUAD;
```

Cấu trúc này chứa cường độ các thành phần màu đỏ, xanh dương, xanh lục. Mỗi màu trong DIB được thể hiện bởi một cấu trúc RGBQUAD. Tức là bitmap 16 màu (4 bit) có một bảng danh sách màu gồm 16 cấu trúc RGBQUAD. Trong khi bitmap 256 màu (8 bit) có một bảng danh sách màu gồm 256 cấu trúc RGBQUAD. Ngoại trừ ảnh 24 bit màu thì không có bảng danh sách màu.

Sau cấu trúc BITMAPINFO của DIB là dữ liệu ảnh. Kích thước dữ liệu tất nhiên phụ thuộc kích thước ảnh. Hình 9.2 thể hiện toàn bộ bố trí của một DIB file.



Hình 9.2 Cấu trúc DIB

9.4.Giới Thiệu Lớp CDib:

Mặc dù Visual C++ mô tả lớp cho nhiều đối tượng đồ họa, nhưng không có lớp nào cho DIB. Với một lớp để quản lý DIB sẽ giúp tổ chức mã vào các module sử dụng nhiều lần. Do đó ta sẽ xây dựng một lớp đơn giản, với tên gọi Cdib, để đọc các CDib từ đĩa vào bộ nhớ, và trả về thông tin quan trọng về DIB.

9.4.1.Giao diện lớp CDib:

Giao diện lớp CDib được thể hiện bởi header file trong bảng kê 9.1 như sau:

Bảng kê 9.1: **CDib.h**

```
#ifndef _ _CDIB_H
    #define _ _CDIB_H

    class CDib : public CObject
    {
    protected:
  LPBITMAPINFOHEADER m_pBmInfoHeader;
  LPBITMAPINFO m_pBmInfo;
  LPBITMAPFILEHEADER m_pBmFileHeader;
  BYTE* m_numColors;
  RGBQUAD* m_pRGBTable;
  BYTE* m_pDibBits;

    public:
  CDib(const char* fileName);
  ~CDib();
```

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

```
DWORD GetDibSizeImage();
UINT GetDibNumColors();
UINT GetDibHeight();
UINT GetDibWidth();
LPRGBQUAD GetDibRGBTablePtr();
LPBITMAPINFO GetDibInfoPtr();
LPBITMAPINFOHEADER GetDibInfoHeaderPtr();
BYTE* GetDibBitsPtr();

  protected:
void LoadBitmapFile(const char*  fileName);
  };

  #endif
```

Các thành phần dữ liệu lớp CDib hầu hết là các con trỏ trỏ đến các phần khác nhau DIB. Các con trỏ này chứa địa chỉ các cấu trúc BITMAPINFO, BITMAPFILEHEADER, và BITMAPINFOHEADER, cũng như địa chỉ bảng danh sách màu và dữ liệu ảnh. Thành phần m_numColors chứa số màu trong DIB.

Như hầu hết các lớp, CDib có hàm tạo và hàm hủy. Theo khai báo hàm tạo, một đối tượng CDib được tạo bằng cách truyền tên file của bitmap cần tải cho hàm.

Lớp CDib mô tả tám hàm thành viên chung, cho phép ta được thông tin quan trọng về Dib sau khi tải nó. Bảng kê 9.5 liệt kê các hàm trên và nhiệm vụ của chúng.

Cuối cùng hàm thành viên protected LoadBitmapFile() của lớp CDib được dùng gọi nội bộ để tải file DIB. Hàm này không thể gọi trực tiếp.

Bảng kê 9.5: Các hàm thành viên chung của lớp CDib

Tên hàm	Mô tả
GetDibSizeImage()	Trả về kích thước ảnh theo byte
GetDibWidth()	Trả về chiều rộng DIB theo pixel
GetDibHeight()	Trả về chiều cao DIB theo pixel
GetDibNumColors()	Trả về số màu trong DIB
GetDibInfoHeaderPtr()	Trả về con trỏ trỏ đến cấu trúc BITMAPINFO HEADER
GetDibInfoPtr()	Trả về con trỏ trỏ đến cấu trúc BITMAPINFO
GetDibRGBTablePtr()	Trả về con trỏ trỏ đến bảng danh sách màu
GetDibBitsPtr()	Trả về con trỏ trỏ đến dữ liệu ảnh

9.4.2.Lập Trình Lớp CDib:

Đoạn mã trong CDib.cpp ở bảng kê 9.2 định nghĩa lớp CDib.

Bảng kê 9.2: **CDib.cpp**

```
//////////////////////////////////////////////////////
```

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

```
// CDib.cpp : implementation file
//////////////////////////////////////////////////
#include "stdafx.h"
#include "DibView.h"
#include "CDib.h"
#include "windowsx.h"
//////////////////////////////////////////////////
// CDib ::CDib()
//////////////////////////////////////////////////
CDib::CDib(const char *fileName)
{
  //load the bitmat and initialize the class's data members.
  LoadBitmapFile(fileName);
}
//////////////////////////////////////////////////
// CDib ::~CDib
//////////////////////////////////////////////////
CDib::~CDib()
{
  //Free the memory assigned to the bitmap.
  GlobalFreePtr(m_pBmInfo);
}
//////////////////////////////////////////////////
// CDib ::LoadBitmapFile()
//////////////////////////////////////////////////
void CDib::LoadBitmapFile(const char *fileName)
{
  //Construct and open a file object
  CFile file(fileName, CFile::modeRead);
  //Read a bitmap's file header into memory
  BITMAPFILEHEADER bmFileHeader;
  file.Read((void*)&bmFileHeader,sizeof(bmFileHeader));

  //Check whether the file is really a bitmap
  if(bmFileHeader.bfType!=0x4d42)
  {
     AfxMessageBox(" Not a bitmap file");
  m_pBmFileHeader=0;
  m_pBmInfo=0;
  m_pBmInfoHeader=0;
  m_pRGBTable=0;
  m_pDibBits=0;
  m_numColors=0;
  }
```

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

```
  //
  else
  {  //Calculate the size of the DIB,which is the
     // file size minus the size of the file hear
  DWORD fileLength=file.GetLength();
  DWORD dibSize=fileLength-sizeof(bmFileHeader);
  // Allocate enough memory to fit the bitmap.
        BYTE* pDob=(BYTE*)GlobalAllocPtr(GMEM_MOVEABLE,dibSize);

  //Read the bitmap into memory and close the file.
  file.Read((void*)pDib,dibSize);
  file.Close();

  //Initialize pointer to the bitmap's
  //and BITMAPINFOHEADER structures.
  m_pBmInfo=(LPBITMAPINFO) pDib;
  m_pBmInfoHeader=(LPBITMAPINFOHEADER) pDib;

  //Calculate a pointer to the bitmap's color table.
        m_pRGBTable=(RGBQUAD*)(pDib + m_pBmInfoHeader->biSize);

  //Get the number of colors in the bitmap.
  int m_numColor = GetDibNumColors();

  //Calculate the bitmap image's size.
  m_pBmInfoHeader->biSizeImage = GetDibSizeImage();
  //Make sure the biClrUsed field is initialized property.
  if(m_pBmInfoHeader->biClrUsed ==0)
     m_pBmInfoHeader->biClrUsed=m_numColors;
  //Calculate a pointer to the bitmap's actual data.
        DWORD clrTableSize = m_numColors * sizeof(RGBQUAD);
         m_pDibBits = pDib + m_pBmInfoHeader->biSize + clrTableSize;

  }

}
////////////////////////////////////////////////////
// CDib ::GetDibSizeImage()
////////////////////////////////////////////////////
DWORD CDib::GetDibSizeImage()
{
  //if the bitmap's biSizeImage field contains
  if(m_pBmInfoHeader->biSizeImage ==0)
  {  //Get the width in bytes of a single row.
```

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

```
    DWORD byteWidth=(DWORD) GetDibWidth();

    //Fet the height of the bitmap.
    DWORD height=(DWORD) GetDibHeight();

    //Multiply the byte width by the number of rows.
    DWORD imageSize = byteWidth * height;

    return imageSize;

  }
  //Otherwise, just returnthe size stored in the BITMAPINFOHEADER structure.
  else
    return m_pBmInfoHeader->biSizeImage;
}
////////////////////////////////////////////////////
// CDib ::GetDibWidth()
////////////////////////////////////////////////////
UINT CDib::GetDibWidth()
{
  return(UINT)m_pBmInfoHeader->biWidth;
}

UINT CDib::GetDibHeight()
{
  return(UINT)m_pBmInfoHeader->biHeight;
}
////////////////////////////////////////////////////
// CDib ::GetDibNumColors()
////////////////////////////////////////////////////

UINT CDib::GetDibNumColors()
{
  if((m_pBmInfoHeader->biClrUsed        ==0)        &&(m_pBmInfoHeader-
>biBitCount<9))
    return(1<< m_pBmInfoHeader->biBitCount);
  else
    return(int)m_pBmInfoHeader->biClrUsed ;
}
////////////////////////////////////////////////////
// CDib ::GetDibInfoHeaderPtr()
////////////////////////////////////////////////////

LPBITMAPINFOHEADER CDib::GetDibInfoHeaderPtr()
```

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

```
{
  return m_pBmInfoHeader;
}
// CDib ::GetDibInfoPtr()

LPBITMAPINFO CDib::GetDibInfoPtr()
{
  return m_pBmInfo;
}
////////////////////////////////////////////////////
// CDib ::GetDibRGBTablePtr()
////////////////////////////////////////////////////

LPRGBQUAD CDib::GetDibRGBTablePtr()
{
  return m_pRGBTable;
}
////////////////////////////////////////////////////
// CDib ::GetDibBitsPtr()
////////////////////////////////////////////////////

BYTE* CDib::GetDibBitsPtr()
{
  return m_pDibBits;
}
```

9.4.3.Tải DIB Vào Bộ Nhớ:

Một đối tượng CDib được tạo bằng lời gọi hàm tạo với tên file DIB muốn tải. Hàm tạo truyền tên file này đến hàm thành viên protected LoadBitmapFile() là hàm thực sự tải bitmap. LoadBitmapFile() tạo mã cho lớp CDib. Các hàm thành viên CDib khác trả về các giá trị được tính tóan bởi LoadBitmapFile().

Trước tiên LoadBitmapFile() xây dựng một đối tượng Cfile:

```
CFile file(fileName, CFile::modeRead);
```

Dòng mã này không chỉ xây dựng đối tượng CFile tên là file, mà còn mở file (tên file được truyền trong đối số fileName) trong chế độ chỉ đọc (read-only).

Tiếp theo hàm khai báo một cấu trúc BIMAPFILEHEADER và đọc header file của DIB vào cấu trúc:

```
BITMAPFILEHEADER bmFileHeader;
file.Read((void*)&bmFileHeader,sizeof(bmFileHeader));
```

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

Hàm thành viên Read() của đối tượng CFile yêu cầu hai đối số gồm con trỏ trỏ đến bộ m lưu trữ dữ liệu và kích thước bộ đệm. Trong trường hợp này bộ đệm là cấu trúc TMAPFILEHEADER.

Sau đó header file của DIB được lưu trữ trong cấu trúc bmFileHeader. Nhiệm vụ đầu tiên là kiểm tra file mở có phải là bitmap không, bằng cách kiểm tra thàng phần bfType có mang giá trị thập lục phân 0x4D42 (mã ASCII của các chữ cái BM) không:

```
if(bmFileHeader.bfType!=0x4d42)
{
        AfxMessageBox(" Not a bitmap file");
        m_pBmFileHeader=0;
        m_pBmInfo=0;
        m_pBmInfoHeader=0;
        m_pRGBTable=0;
        m_pDibBits=0;
        m_numColors=0;
}
```

Nếu thành phần cấu trúc bfType không chứa giá trị đúng, hàm thiết lập mọi thành phần dữ liệu của lớp bằng 0 và thoát. Ngược lại LoadBitmapFile() sẽ gọi hàm thành viên GetLength() để lấy kích thước file mở:

```
DWORD  fileLength = file.GetLength();
```

Sau đó kích thước của DIB được tính toán bằng cách lấy kích thước file trừ đi kích thước file header:

```
DWORD  dibSize = fileLength- sizeof(bmFileHeader);
```

Chương trình sử dụng kết quả dibSize vừa có để cấp phát bộ nhớ lưu trữ DIB:

```
BYTE* pDib = (BYTE*) GlobalAllocPtr(GMEM_MOVEABLE, dibSize)
;
```

GlobalAllocPtr() là hàm Windows cấp phát bộ nhớ và trả về một con trỏ trỏ đến bộ nhớ đó. Hai đối số của hàm là một cờ cho biết bộ nhớ được cấp phát thế nào và kích thước bộ nhớ được cấp phát.

Sau khi cấp phát bộ nhớ, hàm thành viên Read() đọc DIB vào bộ nhớ rồi hàm Close() đóng file:

```
file.Read((void*) pDib, dibSize);
file.Close();
```

Tới đây, DIB đã được lưu trữ trong bộ nhớ, và LoadBitmapFile() bây giờ có thể tính toán các giá trị mà lớp CDib cần. Địa chỉ các cấu trúc BITMAPINFO và BITMAPINFOHEADER là đồng nhất, cùng một địa chỉ bộ đệm mà DIB được lưu trữ:

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

```
m_pBmInfo = (LPBITMAPINFO) pDib;
m_pBmInfoHeader = (LPBITMAPINFOHEADER) pDib;
```

LoadBitmapFile() tính toán con trỏ trỏ đến bảng danh sách màu bằng cách thêm kích thước cấu trúc BITMAPHEADER vào địa chỉ của DIB trong bộ nhớ:

```
m_pRGBTable = (RGBQUAD*) (pDib + m_pBmInfoHeader -> bisize);
```

Khi thực hiện kiểu toán con trỏ này, cần cẩn thận với kiểu dữ liệu sử dụng. Ví dụ do pDib là con trỏ theo BYTE, số byte lưu trữ trong biSize được thêm vào con trỏ này. Nhưng nếu pDib được định nghĩa như con trỏ theo cấu trúc BITMAPINFO, thì giá trị biSize* sizeof(BITMAPINFO) sẽ được thêm vào pDib.

Tiếp theo thành phần dữ liệu m_numColors được khởi tạo bằng lời gọi GetDibnumColors():

```
int m_numColors = GetDibnumColors();
```

Cách làm việc của GetDibnumColors() sẽ nói đến sau.

Thành phần dữ liệu biSizeImage của cấu trúc BITMAPINFOHEADER được điền đầy bằng lời gọi GetDibSizeImage():

```
m_pBmInfoHeader -> biSizeImage = GetDibSizeImage();
```

sau đó nếu thành phần biClrUsed của BITMAPINFOHEADER bằng 0, LoadBitmapFile() sẽ khởi tạo nó theo giá trị đúng, đang được lưu trữ trong thành phần dữ liệu m_numColors:

```
if(m_pBmInfoHeader -> biSizeImage = =0)
m_pBmInfoHeader -> biSizeImage = m_numColors;
```

Cuối cùng LoadBitmapFile() tính toán địa chỉ ảnh DIB bằng cách cộng kích thước cấu trúc BITMAPINFOHEADER và kích thước bảng danh sách màu vào con trỏ pDib (chứa địa chỉ của DIB trong bộ nhớ):

```
DWORD clrTableSize = m_numColors * sizeof(RGBQUAD);
    m_bDibBit = pDib + m_pBmInfoHeader -> biSize + clrTableSize;
```

9.4.4. Các Hàm Thành Viên Khác Của Lớp CDib:

Ở trên, một số thành phần dữ liệu CDib được khởi tạo bằng lời gọi các hàm thành viên CDib. Một trong số các hàm thành viên đó là GetDibImage(), dùng tính toán kích thước DIB theo byte. Hàm này trước tiên kiểm tra thành phần biSizeImage cấu trúc BITMAPINFOHEADER có chứa giá trị 0 hay không:

```
if(m_pBmInfoHeader -> biSizeImage = = 0)
```

Nếu có hàm trả về giá trị lưu trữ trong biSizeImage:

```
return m_pBmInfoHeader -> biSizeImage;
```

KILOBOOKS.COM
Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

Ngược lại, hàm sử dụng thông tin đã có để tính toán kích thước ảnh. Trước tiên hàm lấy chiều rộng và chiều cao DIB theo pixel:

```
DWORD  byteWidth = (DWORD) GetWidth();
DWORD height = (DWORD) GetDibHeight();
```

Sau đó, nhân chiều rộng với chiều cao để có kích thước toàn bộ ảnh bitmap:

```
DWORD imageSize = byteWidth * height;
```

Chú ý rằng các tính toán này được thực hiện sau từ WORD để loại trừ sự cắt bỏ thường xảy ra với số nguyên. DIB có thể lớn.

Một hàm thành viên khác có thể phải thực hiện một số tính toán là GetDibNumColors(). Trước tiên hàm kiểm tra giá trị thành phần biClrUsed của cấu trúc BITMAPINFOHEADER:

```
if((m_pBmInfoHeader -> biClrUsed = = 0) &&
   (m_pBmInfoHeader -> biBitCount < 9))
```

Nếu giá trị này khác 0, hàm trả về giá trị đó:

```
return (int) m_pBmInfoHeader -> biClrUsed;
```

Nếu giá trị này bằng 0, hàm tính số màu bằng cách dời giá trị 1 sang trái, dùng thành phần biBitCount làm độ dời:

```
return(1<< m_pBmInfoHeader -> biBitCount);
```

Kết quả là giá trị 2 cho 1 bit (đơn sắc) DIB, 16 cho 4 - bit DIB, và 256 cho 8 – bit DIB.

Các hàm thành viên còn lại của CDib trả về giá trị các thành phần dữ liệu CDib. Ví dụ hàm GetDibRGBTablePtr() trả về một con trỏ trỏ đến bảng danh sách màu của DIB, một giá trị được lưu trữ trong thành phần dữ liệu m_pRGBTable:

```
LPRGBQUAD CDib::GetDibRGBTablePtr()
   {
   return m_pRGBTable;
}
```

9.5.Gán Cấu Trúc Cho Đa Giác:

Như đã biết, các đa giác được điền đầy thiếu các chi tiết đồ họa làm cho đa giác giống đối tượng cần tạo. Ví dụ cần xây dựng khối chữ cái từ sáu đa giác tạo thành khối vuông. Nhưng khối vuông đơn giản thì không giống hình muốn tạo, mà phải cần sơn các chi tiết lên bề mặt đa giác. Việc này được thực hiện với texture mapping (gán cấu trúc).

Mặc dù texture mapping là quá trình đòi hỏi tính toán nhiều, OpenGL xử lý mọi phép toán cho bạn. Để gán cấu trúc cho một đa giác, ta chỉ cần thiết lập một số thông số và

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

o OpenGL biết vị trí ảnh để gán vào đa giác. Tuy nhiên cũng có sự phức tạp khi gán u trúc.

9.5.1.Anh và DIB:

Vấn đề lập trình là phải báo cho OpenGL cách quản lý các ảnh đã được tạo bởi các chương trình đồ họa Windows. Loại ảnh thông dụng nhất sử dụng với Windows là bitmap độc lập với thiết bị. Nhưng OpenGL không hỗ trợ trực tiếp việc sử dụng loại bitmap này. Để OpenGL sử dụng DIB trong quá trình gán cấu trúc, ta phải thông dịch dữ liệu ảnh DIB.

OpenGL quản lý một ảnh đồ họa dựa vào loại ảnh. Ví dụ một DIB 256 màu gồm các chỉ số màu biểu thị màu sắc trong bảng danh sách màu để vẽ các pixel trong ảnh. Hay một DIB 24-bit chứa các giá trị RGB thực sự, cho biết cường độ các thành phần màu đỏ, xanh lá, xanh dương cho mỗi pixel trong ảnh. Đó là loại thông tin mà OpenGL phải biết về DIB trước khi tìm ra cách thể hiện nó. Ở đây ta nghiên cứu cách dùng DIB 256 màu với OpenGL. Các loại DIB khác sẽ tương tự.

Do DIB 256 màu chứa các chỉ số màu, cần biến đổi sang dạng RGB để OpenGL có thể thể hiện nó trong chế độ màu RGBA. Điều này đòi hỏi các bước sau:

1. Tải DIB vào bộ nhơ.
2. Tạo các bảng danh sách màu (color table) từ bảng màu (palette) của DIB
3. Gọi glPixelMapfv() trao các bảng danh sách màu cho OpenGL.
4. Gọi plPixelTransferi thiết lập OpenGL theo chế độ color-mapping.

9.5.1.1.Tạo Các Bảng Danh Sách Màu:

Nhờ lớp CDib, DIB được tải vào bộ nhớ một cách đơn giản bằng lời gọi hàm tạo với tên file. Con trỏ trỏ đến bảng màu DIB cũng dễ dàng có được bằng lời gọi hàm GetDibRGBTablePtr(). Nhưng để sử dụng DIB với OpenGL thì phải tạo ba bảng danh sách cho các màu đỏ, xanh lá, xanh dương, OpenGL sẽ dùng các giá trị trong các bảng danh sách màu này để thể hiện các pixel của ảnh.

Dữ liệu ảnh của DIB gồm các chỉ số màu. Còn bảng màu của DIB chứa các giá trị RGB cho mỗi màu trong 256 màu dùng trong ảnh. Sẽ thật tuyệt nếu có thể sử dụng trực tiếp các giá trị RGB của bảng màu khi xây dựng các bảng danh sách màu OpenGL. Nhưng bảng màu DIB dùng các số nguyên từ 0 đến 256 để biểu diễn màu, trong khi các bảng danh sách màu dùng cho openGL sử dụng các giá trị dấu chấm động trong dải từ 0.0 đến 1.0. Tuy nhiên việc biến đổi các gía trị RGB của DIB sang các giá trị màuOpenGL thì dễ dàng. Đoạn mã sau thực hiện công việc đó:

```
Void CreateColorTable(CDib* pDib)
{
        LPRGBQUAD pColorTable = pDib                    -
        >GetDibRGBTablePtr(     )
        for(UINT i=0; i<256; i++)
        {
          red[i]= (GLfoat) pColorTable[i].rgbRed /255;
          green[i]= (GLfoat) pColorTable[i].rgbGreen /255;
          blue[i]= (GLfoat) pColorTable[i].rgbBlue /255;
```

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

```
        }
    }
```

Đối số đơn giản của hàn là con trỏ trỏ đến đối tượng CDib. Dòng đầu tiên trong thân hàm dùng con trỏ CDib để lấy con trỏ trỏ đến bảng màu DIB. Vòng lập for được thực hiện cho toàn bộ 256 màu trong bảng màu, biến đổi các giá trị số nguyên RGB của DIB thành các giá trị màu OpenGL dấu chấm động. Việc biến đổi chỉ là chia các giá trị đỏ, xanh lá, xanh dương cho 255. Kết quả tính toán được lưu trữ trong các bảng danh sách màu red[], green[], blue[], là các mảng 256 phần tử kiểu GLfoat.

Khi hàm CreateColorTable() kết thúc, các bảng danh sách màu red[], green[], blue[] chứa các giá trị từ 0.0 đến 1.0 thể hiện các phần tử màu đỏ, xanh lá và xanh dương cho mỗi chỉ số màu. Ví dụ để tìm các giá trị RGB cho pixel có chỉ số màu 57, OpenGL lấy giá trị tại chỉ số 57 trong bảng danh sách red[], giá trị tại chỉ số 57 trong bảng danh sách green[], giá trị tại chỉ số 57 trong bảng danh sách blue[]. Sau đó tìm màu trong bảng màu logic (Dược tạo cho ứng dụng) giống với màu kết quả nhất.

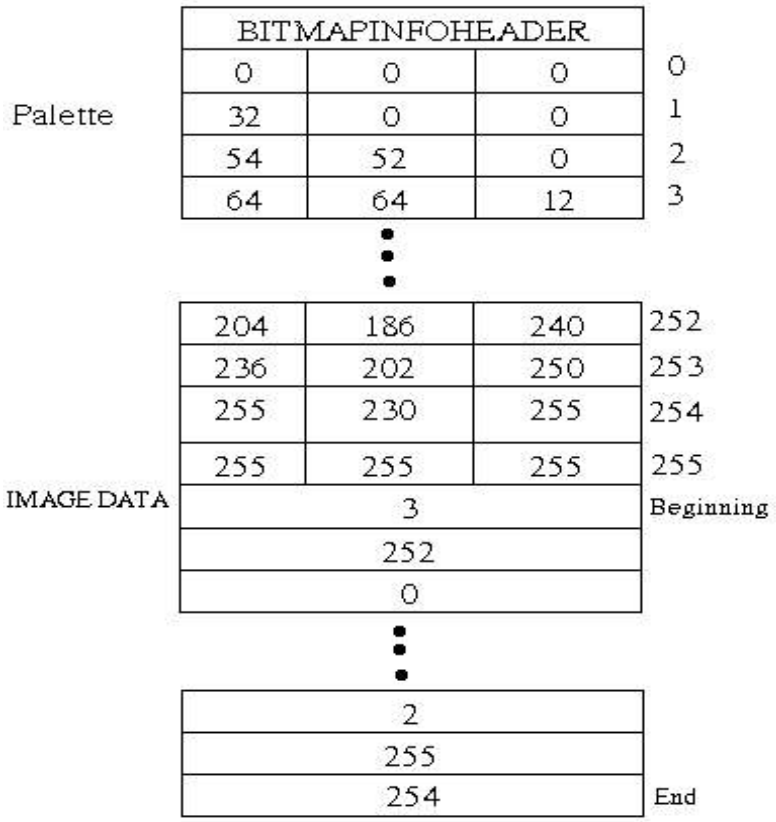
Hình 9.3 đến hình 9.5 tổng kết quá trình color-mapping. Trong hình 9.3, chương trình đọc DIB vào bộ nhớ. Như đã biết, dữ liệu đầy đủ của DIB boa gồm BITMAPINFOHEADER, bảng màu được thiết lập 256 đầu vào (entry), mỗi đầu vào chứa một giá trị đỏ, xanh lá, xanh dương. (Dữ liệu không thích hợp với quá trình color-mapping không thể hiện trên hình). Trong hình 9.4 hàm CreateColorTable() chia mỗi giá trị R, G, B cho 255 và đặt kết quả vào bảng danh sách màu tương ứng. Cuối cùng, trong hình 9.5, OpenGL xác định các màu cho ảnh bằng cách mapping các chỉ số màu DIB vào bảng danh sách màu hiện chứa các giá trị RGB có hiệu lực.

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

Hình 9.3 Tải DIBVào bộ nhớ

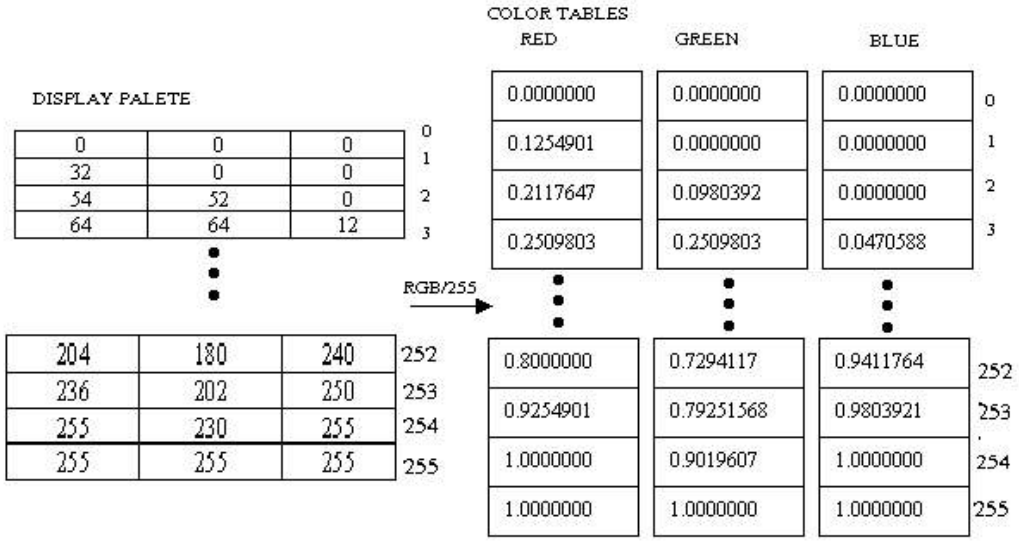


KILOBOOKS

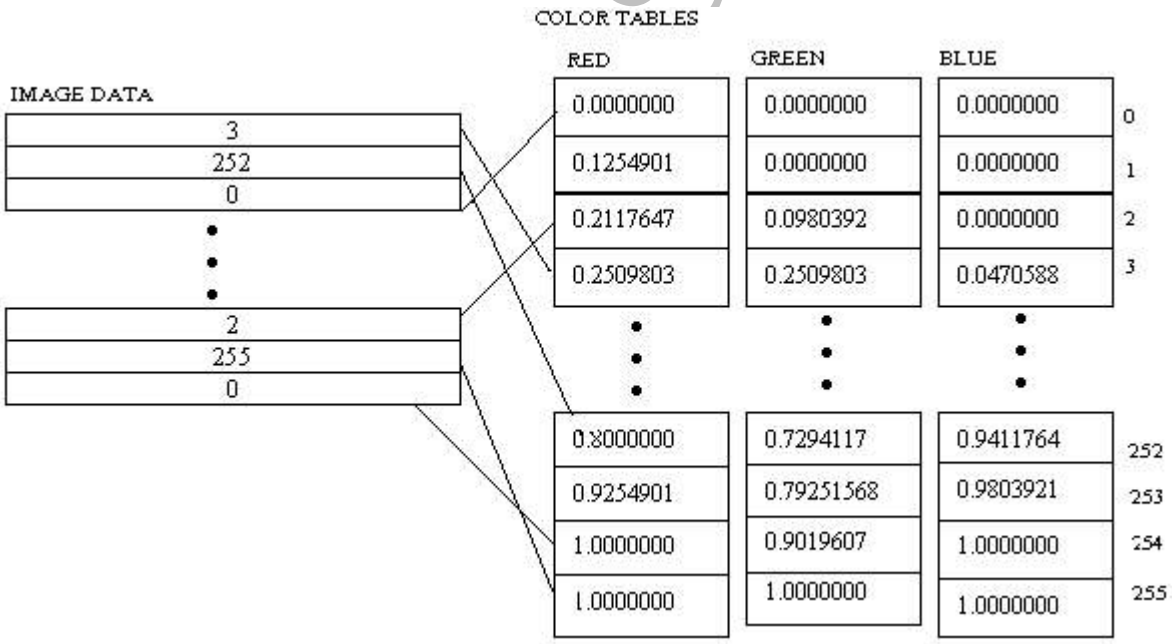
Hình 9.4 tọa các bảng danh sách màu OpenGL:

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN



Hình 9.5: Mapping dữ liệu ảnh DIB với các bảng danh s ách màu:



9.5.1.2 .Chuyển Các Bảng Danh Sách Màu Cho OpenGL:

Sau khi tạo các bảng danh sách màu, OpenGL phải được báo để thiết lập các bảng mapping bằng lời gọi hàm glPixelMapfv() như sau:

```
glPixelMapfv(GL_PIXEL_MAP_I_TO_R, 256, m_red);
glPixelMapfv(GL_PIXEL_MAP_I_TO_G, 256, m_green);
glPixelMapfv(GL_PIXEL_MAP_I_TO_B, 256, m_blue);
```

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

Ba đối số của hàm glPixelMapfv() gồm một hằng cho biết kiểu mapping, kích thước n đồ, và địa chỉ bản đồ. Kiểu mapping là một trong các hằng liệt kê ở bảng 9,6. Hàm glPixelMapfv() có hai phiên bản khác là glPixelMapuiv() và glPixelMapusv(). Ở đoạn mã trên, glPixelMapfv() được gọi ba lần, lần lượt cho một bảng danh sách màu đỏ, xanh lá, xanh dương.

Bảng 9.6: Các kiểu mapping cho hàm glPixelMap()

Tên kiểu	Mô tả
GL_PIXEL_MAP_A_TO_A	Map các thành phần alpha theo các thành phần alpha
GL_PIXEL_MAP_B_TO_B	Map các thành phần xanh dương theo các thành phần xanh dương
GL_PIXEL_MAP_G_TO_G	Map các thành phần xanh lá theo các thành phần xanh lá
GL_PIXEL_MAP_I_TO_A	Map các chỉ số màu theo các thành phần alpha
GL_PIXEL_MAP_I_TO_B	Map các chỉ số màu theo các thành phần xanh dương
GL_PIXEL_MAP_I_TO_G	Map các chỉ số màu theo các thành phần xanh lá
GL_PIXEL_MAP_I_TO_I	Map các chỉ số màu theo các chỉ số màu
GL_PIXEL_MAP_I_TO_R	Map các chỉ số màu theo các thành phần đỏ
GL_PIXEL_MAP_R_TO_R	Map các thành phần đỏ theo các thành phần đỏ
GL_PIXEL_MAP_S_TO_S	Map các thành phần stencil theo các chỉ số stencil

9.5.1.3.Mapping Màu:

Sau khi có các bảng danh sách màu và chuyển chúng cho OpenGL, hàm glTransferi() được gọi để mapping màu như sau:

```
glPixelTransferi(GL_MAP_COLOR, TRUE);
```

Hai đối số của hàm là hằng xác định kiểu bản đồ hóa pixel và giá trị thiết lập kiểu. Hàm glTransferi() có nhiều cách dùng. Nhưng phần lớn không phải để mapping dữ liệu ảnh DIB theo các bảng danh sách màu. Chỉ với hai đối số GL_MAP_COLOR và TRUE thì hàm khởi động việc mapping màu OpenGL.

Khi mapping màu, OpenGL để thực hiện DIB với 256 màu, tính toàn vẹn của thể hiện phụ thuộc độ phù hợp giữa bảng màu logic và các màu dùng để tọa DIB. Do OpenGL được thiết kế cho các hệ thống 64.000 màu hoặc hơn, nên ở các hệ thống đó OpenGL thể hiện DIB thực sự hoàn hảo. Còn hệ thống 256 màu, sự thể hiện sẽ có vài suy biến.

9.5.2.Thiết Lập Việc Gán Cấu Trúc:

Việc gán cấu trúc có thể bắt đầu sau khi đã tải DIB và tạo các bảng danh sách màu của nó. Bước đầu tiên là cho OpenGL biết cách thực hiện texture mapping, bằng cách gọi hàm glTexParameteri() để thiết lập các thông số khác nhau như sau:

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,
GL_CLAMP);
```

KILOBOOKS.COM
Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

```
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,
_CLAMP);
        glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,
GL_NEAREST);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_NEAREST);
```

Hàm glTexParameteri() có bốn phiên bản. Các đối số của hàm gồm hằng xác định kiểu cấu trúc, thông số cần thiết lập, và giá trị thiết lập cho thông số. Đối số đầu tiên là GL_TEXTURE_1D cho cấu trúc một chiều, hay GL_TEXTURE_2D cho cấu trúc hai chiều. GL_TEXTURE_2D, là ảnh hình chữ nhật, thường được sử dụng.

Đối số thứ hai cho OpenGL biết cách quản lý texture mapping dưới các điều kiện nào đó. Ví dụ với GL_TEXTURE_WRAP_S và GL_TEXTURE_WRAP_T, OpenGL dùng ảnh đơn hoặc đa cấu trúc, tương ứng với cấu trúc theo phương ngang hay đứng. Các thông số này phải thiết lập với GL_CLAMP (cho ảnh đơn) hay với GL_REPEAT (cho mẫu lập lại). Khi dùng ảnh đơn cho đa giác, phải sử dụng GL_CLAMP, nếu không kết quả nhận được sẽ kỳ dị. GL_TEXTURE_MAG_FILTER, GL_TEXTURE_MIN_FILTER xác định cách phóng to hay thu nhỏ cấu trúc khi diện tích nơi đến to hơn hoặc nhỏ hơn cấu trúc. Việc sữ dụng GL_NEAREST với các thông số này cho phép texture mapping nhanh, va nhận được kết quả sắc nét nhất. GL_NEAR thì chậm hơn do phải tính toán màu sắc cho một pixel trên cơ sở mức trung bình của bốn pixel.

Sau khi thiết lập các thông số cấu trúc, phải thiết lập mỗi trường cấu trúc, tức là xác định hàm cấu trúc nào (hàm toán học, không phải hàm C++) mà OpenGL dùng tính toán các giá trị màu cho bề mặt cấu trúc. Việc này được thực hiện với lời gọi hàm glTexEnvi():

```
glTexEnvi(GL_TEXTURE_ENVI, GL_TEXTURE_ENVI_MODE, GLDECAL);
```

Hai đối số đầu của hàm lần lượt phải là GL_TEXTURE_ENVI và GL_TEXTURE_ENVI_MODE, hay GL_MODULATE. Thông thường GL_DECAL được sữ dụng. Khi đó hàm thay thế các pixel nơi đến bằng cấu trúc. Hai hằng còn lại kết hợp nơi đến và các giá trị cấu trúc theo các cách khác nhau.

Bước cuối cùng là khởi động texture mapmping:

```
glEnable( GL_TEXTURE_2D);
```

Nếu cấu trúc một chiều được sử dụng thì đối số sẽ là GL_TETURE_1D.

9.5.3.Định Nghĩa Các Vertex Và Cấu Trúc Của Chúng:

Để thiết lập texture mapping có rất nhiều việc phải làm, nhưng hầu hết đã được hoàn tất ở trên. Các việc còn lại là trao cấu trúc cho OpenGL, định nghĩa các vertex đa giác và nêu quan hệ giữa chúng với các tọa độ cấu trúc. Đoạn mã sau thực hiện điều đó:

```
glTexImage2D(GL_TEXTURE_2D, 0, 3 , width, height,
0, GL_COLOR_INDEX, GL_UNIGNED_BYTE, pTextureBits);

 glBegin(GL_POLYGON);
glTexCoord2f(0.0f, 1.0f);
glVertex3f(-1,0f, 1.0f, 0.0f);
```

KILOBOOKS.COM
Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

```
        glTexCoord2f(0.0f, 0.0f);
        glVertex3f(-1,0f, -1.0f, 0.0f);
        glTexCoord2f(1.0f, 0.0f);
        glVertex3f(1,0f,-1.0f, 0.0f);
        glTexCoord2f(1.0f, 1.0f);
        glVertex3f(1,0f, 1.0f, 0.0f);
    glEnd();
    glFlush();
```

Trong đoạn mã trên, lời gọi glTexImage2D() cho OpenGL biết nơi chứa cấu trúc cần sử dụng. Đối số đầu tiên phải là GL_TEXTURE_2D. Đối số thứ hai là cấu của chi tiết, trừ khi dùng với ảnh đa cấu trúc, còn lại giá trị 0 được dùng cho đối số này. Đối số thứ ba cho biết thành phần màu trong cấu trúc. Trong trường hợp dùng ảnh RGB, giá trị này là 3. Đối số thứ tư và năm là chiều rộng và chiều cao cấu trúc. Đối số thứ sáu xác định chiều rộng đường viền cấu trúc, là 0 hay 1, thường là 0. Đối số thứ bảy cho biết định dạng của dữ liệu ảnh cấu trúc. Do dữ liệu ảnh DIB thể hiện chỉ số màu, đối số này là GL_COLORINDEX trong đoạn mã trên. Các giá trị khác có thể có là GL_ALPHA, GL_BLUE, GL_GREEN, GL_LUMINANCE, GL_LUMINANCE_ALPHA, GL_RED, GL_RGB, GL_RGBA. Hai đối số cuối cùng là kiểu dữ liệu và địa chỉ cấu trúc.

Sau khi trao cấu trúc cho OpenGL, ta phải định nghĩa các vertex của đa giác, cùng lúc cho biết quan hệ giữa chúng với các tọa độ cấu trúc bằng lời gọi hàm glTexCoord2f(). Hai đối số hàm này là các tọa độ S và T của cấu trúc. (khi nói về cấu trúc, S và T tương tự như X và Y trong khi R và Q tương tự như Z và W).

Hình 9.6 thể hiện các cách tọa độ cấu trúc phù hợp với ảnh cấu trúc.Nếu chỉ định góc trái trên của cấu trúc theo góc trái trên đa giác, góc trái dưới theo góc trái dưới, và tương tự, cấu trúc sẽ xuất hiện đúng theo hướng nhìn đa giác.Việc chỉ định các góc trên của cấu trúc theo các góc dưới của đa giác cho ta cấu trúc phản chiếu theo phương đứng. Tương tự, việc chỉ định các góc trái của cấu trúc theo các góc phải của đa giác cho ta một cấu trúc phản chiếu theo phương ngang. Các kiểu chỉ định khác sẽ dẫn đến kết quả kì quặc.

9.6.Tổng Kết:

OpenGL hổ trợ trực tiếp hai loại đối tượng đồ họa là bitmap và ảnh. Bitmap là ảnh đơn sắc thường dùng như các mask. Anh là bức tranh đầy màu sắc. Nhiều hàm OpenGL cho phép di chuyển ảnh trong bộ nhớ và thể hiện chúng lên màn hình. Ta có thể sử dụng ảnh để gán cấu trúc.

Gán cấu trúc là qúa trình phức tạp, đòi hỏi sự hiểu biết về cách lưu trữ ảnh trong bộ nhớ, và cách OpenGL thao tác chúng. Ví dụ, để sử dụng các bitmap độc lập với thiết bị (DIB) làm cấu trúc, phải biết tải DIB vào bộ nhớ và thiết lập cơ chế truyền pixel để map các màu sắc của DIB vào các bảng danh sách màu được tạo từ bảng màu DIB.

Để gán cấu trúc, trước hết phải khởi tạo các thông số gán cấu trúc, thiết lập môi trường gán cấu trúc, và kích hoạt việc gán cấu trúc. Ngoài ra, phải truyền cấu trúc cần gán cùng các thông tin thể hiện cấu trúc đó cho OpenGL. Cũng cần phải liên kết các tọa độ cấu trúc với các vertex của đa giác được gán cấu trúc.

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

Tuy vậy, chương này chỉ mới đề cập một cách tổng quát khả năng texture mapping a OpenGL.

Chương 10: Pha Trộn, Giảm Hiệu Ứng Răng Cưa Và Sương Mù

Chương trước đã trình bày các kỹ thuật cải thiện ảnh OpenGL 3-D gồm cách sử dụng ảnh bitmap độc lập với thiết bị và gán cấu trúc cho đa giác. Còn một số kỹ thuật tin vi khác như pha trộn, giảm hiệu ứng răng cưa và sương mù sẽ được trình bày trong chương này.

Với các thao tác pha trộn, antialiasing, và sương mù, ta có thể thêm nhiều hiệu ứng đặc biệt cho cảnh 3-D. Thông thường, pha trộn được dùng để tạo đối tượng trong mờ. Antialiasing giúp khử hiện tượng răng cưa, còn sương mù tạo cảm giác xa gần của đối tượng so với điểm nhìn.

10.1.Pha Trộn:

pha trộn (blending) là phương pháp kết hợp màu của các pixel nguồn và đích theo các cách khác nhau để tạo những hiệu quả đặt biệt. Pha trộn thường được sử dụng để tạo các đối tượng trong mờ. khi một đối tượng được pha trộn chồng lên đối tượng khác thì có thể nhìn xuyên qua đối tượng được bao phủ do màu của đối tượng nguồn kết hợp với màu của pixel được bao phủ tạo ra màu mới.

Cũng có thể dùng pha trộn để đạt được các hiệu quả đặt biệt khác. Ví dụ môt chương trình vẽ OpenGL chẳng hạn có thể sử dụng pha trộn để cho phép người dùng vẽ các lớp màu lên các bức tranh. OpenGL cũng dùng pha trộn để làm phẳng các mép răng cưa của đường thẳng (gọi là antialiasing sẽ nói đến sau)

10.1.1.Kích Hoạt Việc Pha Trộn:

Gống như hầu hết các hàm OpenGL đặt biệt (như chiếu sáng, kiểm tra chiều sâu, hay vẽ đường chấm), trước hết phải kích hoạt việc pha trộn như sau:

```
glEnable(GL_BLEND);
```

Hằng GL_BLEND báo OpenGL cho phép pha trộn. Để tắt việc pha trộn, cũng dùng hằng trên cho hàm glDisable():

```
glDisable(GL_BLEND);
```

10.1.2.Chọn Lựa Các Hàm Pha Trộn:

Để OpenGL biết cách kết hợp các màu ta dùng hàm glBlendFunc() lựa chọn các hàm pha trộn nguồn và đích như sau:

```
glBlendFunc(GL_SRC_ALPHA,GL_ONE_MINUS_SRC_ALPHA)
```

Các hằng làm đối số cho glBlendFunc() lần lượt xác định các hàm pha trộn nguồn và đích, được liệt kê trong các bảng 10.1 và 10.2.

Ý nghĩa các bảng này như sau: Để quyết định màu sắc cho một pixel được pha trộn, OpenGL phải nhân các thành phần R,G,B,A của màu nguồn và đích với một hệ số. Hệ số này được quyết định bởi các hàm pha trộn nguồn và đích . Mọi tính toán được thể hiện

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

ng các bảng trên dẫn đến bốn giá trị mà OpenGL dùng để nhân với các thành phần màu G, B, A.

Ví dụ, nếu chỉ đỊnh GL_ONE là hàm pha trộn nguồn, OpenGL sẽ nhân các thành phần màu R,G,B,A với 1 (dĩ nhiên là nhận được cùng màu xuất phát). Bốn số 1 trong ngoặc đơn tương ứng cho bốn thành phần màu. Trong trường hợp chọn GL_ZERO làm hàm nguồn, kết thúc sẽ là không có màu nguồn

Kết quả tính toán quyết định mức độ ảnh hưởng của các màu nguồn và đích lên màu cuối cùng nhận được. Ví dụ với chọn lựa glBlendFunc (GL_ONE , GL_ZERO), kết quả là không có sự pha trộn do màu nguồn hoàn toàn lau sạch màu đích

Chọn lựa thông dụng nhất như sau :

```
glBlendFunc (GL_SRC_ALPHA , GL_ONE_MINUS_SRC_ALPHA );
```

GL_SRC_ALPHA có nghĩa là OpenGL nhân các thành phần màu nguồn với gia trị alpha của màu nguồn.

Với GL_ONE_MINUS_SRC_ALPHA thì trước hết giá trị alpha của màu nguồn phải được trừ đi 1, sau đó nhân kết quả phép trừ với các giá trị RGBA đích.

Giả sử pixel nguồn (là pixel mà OpenGL sắp sửa vẽ) có các gía trị RGBA (0.7 , 0.5 , 0.8. 0.6) và pixel đích (là pixel mà OpenGL sẽ vẽ pixel nguồn lên) có các giá trị RGBA (0.6, 0.3 , 0.6 . 1.0). Trước tiên OpenGL cung cấp hàm GL_SRC_ALPHA cho các giá trị RGBA nguồn, tức là nhân 0.6 cho 0.7, 0.5 , 0.8, và 0.6. Kết quả nhận được các giá trị pixel nguồn là (0.42, 0.3, 0.48 , 0.36). Sau đó OpenGL cung cấp hàm
GL_ONE_MINUS_SRC_ALPHA cho pixel đích, tức là lấy 1 trừ đi giá trị alpha nguồn, được 0.4 rồi nhận kết quả này cho 0.6, 0.3, 0.6 và 1.0
Kết quả nhận được các giá trị pixel đích là (0.24, 0.12 , 0.24, 0.4)

Như vậy với chọn lựa GL_SRC_ALPHA là hàm pha trộn nguồn và GL_ONE_MINUS_SRC_ALPHA là hàm pha trộn đích, giá trị alpha nguồn sẽ quyết định phần trăm mỗi màu được OpenGL kết hợp để tạo nên màu cuối cùng. Với giá trị alpha 0.6, 60% màu đến từ nguồn và 40% màu đến từ đích

Giá trị alpha càng lớn thì kết quả càng trở nên mờ đục. Do đó có thể coi giá trị alpha như đô mờ đục muốn vẽ.

Bảng 10.1 :các hằng pha trộn nguồn

Hằng số	Hệ số kết quả
GL_DST_ALPHA	(A_D , A_D , A_D , A_D)
GL_DST_COLOR	(R_D , G_D , B_D , A_D)
GL_ONE	(1,1,1,1)
GL_ONE_MINUS_ DST_ALPHA	(1,1,1,1) - (A_D , A_D , A_D , A_D)
GL_ONE_MINUS_ DST_COLOR	(1,1,1,1) - (R_D , G_D , B_D , A_D)
GL_ONE_MINUS_SRC_ALPHA	(1,1,1,1) - (A_S,A_S,A_S,A_S)
GL_ SRC_ALPHA	(A_S,A_S,A_S,A_S)
GL_ SRC_ALPHA_SATURATE	(F,F,F,1) với F= min(A_S,(1-A_0))
GL_ZERO	(0,0,0,0)

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

Bảng 10.2 : Các hằng pha trộn đích

Hằng số	Hệ số kết quả
GL_DST_ALPHA	(A_D , A_D , A_D , A_D)
GL_ONE	(1,1,1,1)
GL_ONE_MINUS_ DST_ALPHA	$(1,1,1,1) - (A_D , A_D , A_D , A_D)$
GL_ONE_MINUS_SRC_ALPHA	$(1,1,1,1) - (A_S,A_S,A_S,A_S)$
GL_ONE_MINUS_SRC_COLOR	$(1,1,1,1) – (R_S , G_S , B_S , A_S)$
GL_ SRC_ALPHA	(A_S,A_S,A_S,A_S)
GL_ SRC_COLOR	(R_S , G_S , B_S , A_S)
GL_ZERO	(0,0,0,0)

10.1.3.Pha Trộn Đa Giác:

Để pha trộn đa giác, tất cả việc phải làm là chọn lựa màu vẽ và vẽ các hình. OpenGL sẽ cung cấp các hàm pha trộn và vẽ kết quả lên màn hình. Đoạn mã sau có thể hiện như hình 10.1 :

```
glClearColor (1.0f , 1.0f , 1.0f , 1.0f );
glClear  (GL_COLOR_BUFFER_BIT);
glEnable (GL_BLEND);
glBlendFunc(GL_SRC_ALPHA_GL_ONE_MINUS_SRC_ALPHA);
glColor4f(0.0f , 1.0f , 1.0f , 1.0f);
glBegin(GL_POLYGON);
        glVertex3f ( -1.0f , 0.25f , 0.0f );
        glVertex3f ( -1.0f , -1.0f , 0.0f  );
        glVertex3f ( 0.25f , -1.0f , 0.0f );
        glVertex3f ( 0.25f , 0.25f , 0.0f );
glEnd ();
glColor4f ( 1.0f , 0.0f , 0.0f , 0.5f  );
glBegin (GL_POLYGON);
        glVertex3f (-0.25f , 1.0f , 0.0f );
        glVertex3f ( -0.25f , -0.25f , 0.0f );
        glVertex3f ( 1.0f , -0.25f , 0.0f );
        glVertex3f ( 1.0f , 1.0f , 0.0f );
glEnd ();
glFlush;
glDisable( GL_BLEND ):
```

Trong đoạn mã trên, giá trị alpha dùng cho pha trộn là 0.5 (nằm ở lời gọi glColor4f() thứ hai). Có nghĩa là trong vùng mà hai hình vuông đè lên nhau, mỗi màu

KILOBOOKS.COM
Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

ıồn và đích ảnh hưởng 50% trên màu pha trộn cuối cùng. Giá trị alpha càng lớn hình ồng thứ hai càng trở nên mờ đục.

Ngoài ra, thứ tự vẽ các hình cũng ảnh hưởng đến việc pha trộn.

Ví dụ, nếu vẽ hình vuông đỏ trước có thể sẽ nhận được kết qủa khác với việc vẽ hình vuông xanh trước.

10.1.4. Pha Trộn Đối Tượng 3-D :

Pha trộn đối tượng 3-D thì phức tạp hơn pha trộn đối tượng 2-D, do việc kiểm tra chiều sâu OpenGL sẽ làm biến mất các đối tượng bị che. Ví dụ, khi vẽ một khối vuông trong mờ bên trong một khối vuông đặc, do việc khi kiểm tra chiều sâu OpenGL ta sẽ không thấy khối vuông đặc. Để kết hợp pha trộn trong cảnh 3-D phải theo các bước sau:

1. Kích hoạt việc kiểm tra chiều sâu
2. Vẽ tất cả đối tượng mờ đục
3. Thiết lập thuộc tính chỉ đọc(read-only cho bộ đệm chiều sâu)
4. Vẽ tất cả đối tượng trong mờ.
5. Trả lại thuộc tính đọc viết (read/write) cho bộ đệm chiều sâu

Dĩ nhiên các bước này không chỉ ra sự phức tạp vốn có khi thể hiện cảnh 3-D bao gồm việc khởi tạo việc chiếu sáng và thực hiện các phép biến hình modelview. Nhiệm vụ quan trọng trong các bước trên là thiết lập thuộc tính chỉ đọc cho bộ đệm chiều sâu trước khi vẽ các đối tượng trong mờ. OpenGl sử dụng bộ đệm chiều sâu để xác định hình nào trước hình nào sau. Việc thiết lập thuộc tính chỉ đọc cho bộ đệm chiều sâu ngăn trở OpenGL thay đổi nội dung bộ đệm này khi vẽ các đốt tượng trong mờ; Và như vậy OpenGL không thể xoá bỏ các đối tượng bị khuất mà ta muốn thấy xuyên qua các đối tượng trong mờ.

Để thiết lập thuôc tính chỉ đọc cho bộ đệm chiều sâu ta sử dụng hàm

```
glDepthMask() :
glDepthMask ( GL_FALSE );
```

Đối số Glboolean của hàm cho phép hay xóa bỏ khả năng ghi vào bộ đệm chiều sâu của OpenGL. Giá trị GL_FALSE thiết lập thuộc tính chỉ đọc cho bộ đệm chiều sâu trong khi GL_TRUE thiết lập thuộc tính đọc/viết .

Đoạn mã sau được thể hiện như hình 10.2 :

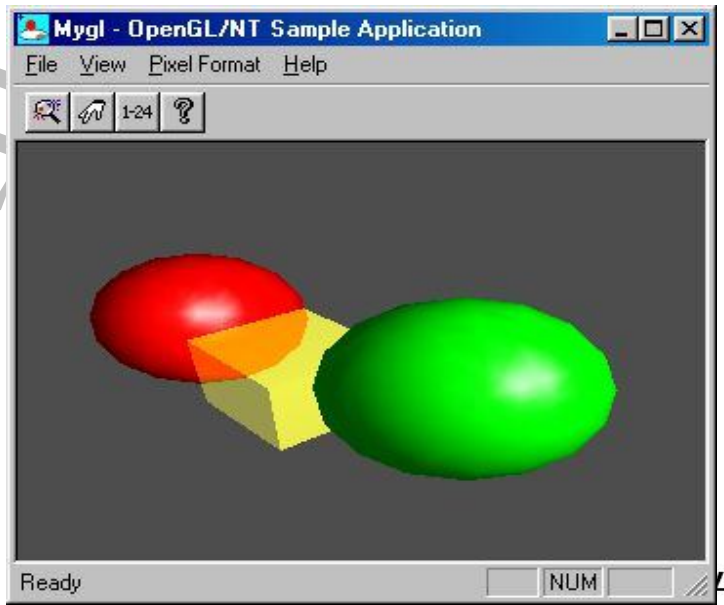
```
glClearColor ( 0.3f , 0.3f , 0.f , 0.0f );
glClear(GL_COLOR_BUFFER_BIT | GL_GL_DEPTH_BUFFER_BIT );
glEnable ( GL_BLEND );
glEnable (GL_LIGHTING );
glEnable (LIGHT0);
glEnable (GL_DEPTH_TEST);
glBlendFunc (GL_ONE , GL_ZERO);
GLfloat lightAmbient []={0.1f , 0.1f , 01f , 0.0f};
GLfloat lightDiffuse[]={1.0f , 1.0f , 1.0f , 0.0f};
```

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

```
GLfloat lightSpecular[]={1.0f , 1.0f , 1.0f , 0.0f};
GLfloat lightPosition[]= {2.0f , 0.5f , 3.0f , 1.0f };
GLfloat materialSpecular[]={1.0f , 1.0f , 1.0f , 0.0f};
GLfloat materialSphere1[]={1.0f , 0.0f , 0.0f , 0.0f};
GLfloat materialSphere2[]= {0.0f , 1.0f , 0.0f , 0.0f};
 GLfloat materialCube[]= {1.0f , 1.0f , 0.0f , 0.0f};
 glLightfv (GL_LIGHT0 , GL_ AMBIENT, lightAmbient);
 glLightfv (GL_LIGHT0 , GL_DIFFUSE, lightDiffuse);
 glLightfv (GL_LIGHT0, GL_SPECULAR, lightSpecular);
 glLightfv (GL_LIGHT0, GL_POSITION , lightPosition);
glMaterialfv(GL_FRONT , GL_SPECULAR , materialSpecular);
glMaterialf ( GL_FRONT , GL_SHININESS , 50.0f );
glTranslatef(-1.5f , 0.5f , -3.0f);
glMaterialfv(GL_FRONT,GL_AMBIENT_AND_DIFFUSE,
materialSphere1);
    auxSolidSphere(1.0);
    glTranslatef(2.0f ,- 0.8f , 3.0f);
    glMaterialfv(GL_FRONT,GL_AMBIENT_AND_DIFFUSE,
    materialSphere2);
    auxSolidSphere(0.75);
    glDepthMask(GL_FALSE);
    glBlendFunc(GL_SRC_ALPHA , GL_ONE);
    glMaterialfv(GL_FRONT,GL_AMBIENT_AND_DIFFUSE,
 materialCube);
    glTranslatef(-1.0f , 0.0f , 0.0f );
    glRotatef(40.0f , 1.0f , 0.0f , 0.0f );
    glRotatef(40.0f , 0.0f , 1.0f , 0.0f );
    auxSolidCube(1.0);
    glFlush();
    glDepthMask(GL_TRUE);
    glDisable(GL_LIGHTING);
    glDisable(GL_DEPTH_TEST);
    glDisable(GL_BLEND);
```



Hình 10.2
Pha trộn đối tượng 3-D

KILOBOOKS.COM
y nhấp vào đây!

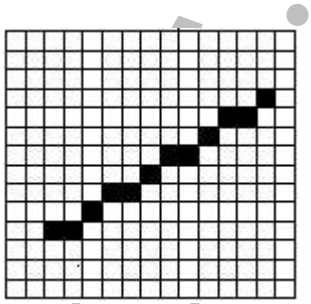
http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

Khi pha trộn trong cảnh 3-D có chiếu sáng , các giá trị màu dùng trong các hàm pha trộn nằm trong màu ánh sáng khuếch tán của vật liệu.Ví dụ trong đoạn mã trên giá trị alpha 0.5 cho hàm pha trộn của khối vuông là phần tử alpha của mảng materialCube[]

10.2.Giảm Hiệu Ứng Răng Cưa:

Trên màn hình một đường thẳng là tập hợp của các pixel được chiếu sáng trong hệ thống kẽ ô vuông (Hình 10.3). Do đó chỉ có đường nằm ngang hay thẳng đứng là được vẽ một cách suông sẽ còn các đường nghiêng sẽ có hiện tượng răng cưa (alias). Độ phân giải càng cao thì hiện tượng răng cưa càng giảm nhưng không thể không có. Giải pháp kĩ thuật để giảm hiệu ứng răng cưa (antialiasing) là sử dụng các sắt thái màu khác nhau để che dấu cạnh răng cưa của đường thẳng trên màn hình. Thực tế antialiasing chỉ là một kiểu hiệu quả pha trộn OpenGL.

Hình 10.3:Đườngrăng cưa trên màn hình



10.2.1.Kích Hoạt Antialiasing:

Giống như việc pha trộn, thao tác đầu tiên để thực hiện antialiasing là kích hoạt antialiasing như sau:

```
glEnable (GL_LINE_SMOOTH );
```

Hằng GL_LINE_SMOOTH báo cho OpenGL thực hiên antialiasing cho đường thẳng. Đối với điểm hay đa giác, antialiasing được kích hoạt bởi các hằng GL_POINT_SMOOTH hay GL_POLYGON_SMOOTH.

10.2.2.*Kích Hoạt Pha Trộn Cho Antialasing:*

Do antialiasing là một hiệu quả pha trộn nên cần phải kích hoạt pha trộn và chọn hàm pha trộn :

```
glEnable (GL_BLEND);
glBlendFunc(GL_SRC_ALPHA,GL_ONE_MINUS_SRC_ALPHA);
```

Với các hàm pha trộn khác nhau sẽ nhận được các kết quả antialiasing khác nhau. Nhưng thông dụng nhất là sử dụng hàm nguồn GL_SRC_ALPHA và hàm đích GL_ONE_MINUS_SRC_ALPHA.Cần nhớ rằng với chọn lựa như vậy, thành phần alpha của đối tượng sẽ ảnh hưởng tới kết quả antialiasing. Ngoài ra các màu trong bảng màu logic sẽ quyết định độ chính xác của antialiasing.

10.2.3.Gợi Ý Làm Phẳng Cho OpenGL:

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

OpenGL có thể thực hiện antialiasing theo nhiều cách tương ứng với sự thiết lập các ng số. Để cho OpenGL biết ý muốn thể hiện nhanh hay chậm với độ chính xác cao hay ...ỏng ưu tiên gì cả, ta dùng hàm glHint() như sau :

```
glHint (GL_LINE_SMOOTH_HINT, GL_DONT_CARE);
```

Đối số thứ nhất của hàm xác định thao tác đề nghị, có thể là GL_FOG_HINT , GL_LINE_SMOOTH, GL_PERSPECTIVE_CORRECTION_HINT, GL_POINT_SMOOTH_HINT hay GL_POLYGON_SMOOTH_HINT.

Đối số thứ hai xác định lời gợi ý , có thể là GL_DONT CARE (cho phép OpenGL thực hiện tùy ý) hay GLFASTEST (OpenGL thự hiện phương thức nhanh nhất) hay GL_NICEST (OpenGL thực hiện phương thức chính xác nhất).

10.2.4.Các Đường Thẳng Antialiasing:

Bước cuối cùng để vẽ các đường thẳng không có răng cưa là thiết lập bề rộng đường và màu. Phải nhớ rằng OpenGL sử dụng các giá trị màu được chọn trong các hàm pha trộn. Đoạn mã sau là một ví dụ chuẩn bị vẽ đường thẳng :

```
glLineWidth (2.0f);
glColor4f (0.0f , 0.0f , 0.0f , 1.0f );
```

Trong trường hợp các hàm pha trộn nguồn, đích là GL_SRC_ALPHA và GL_ONE_MINUS_SRC_ALPHA , chọn lựa màu trên cho giá trị alpha bằng 1.0 , đường được vẽ đậm nhất.Với alpha bằng 0.0 sẽ không có đường nào được vẽ cả.

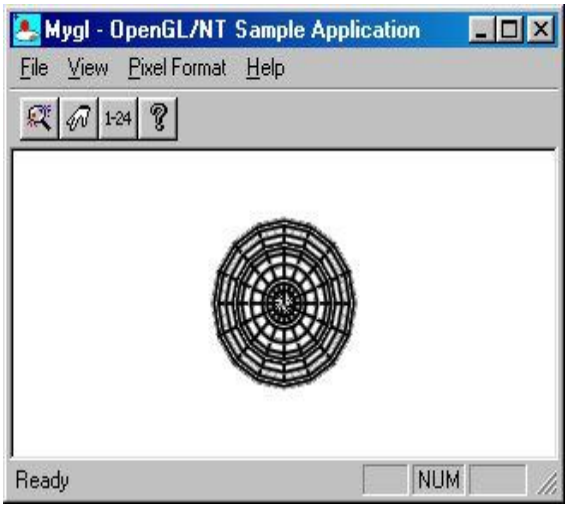
Đoạn mã sau được thực hiện như hình 10.4. Hình 10.5 cho cùng một hình mạng nhện nhưng không có antialiasing :

```
glClearColor (1.0f , 1.0f , 1.0f , 0.0f );
glClear (GL_COLOR_BUFFER_BIT);
glShadeModel (GL_FLAT);

glEnable (GL_LINE_SMOOTH);
glEnable (GL_BLEND);
glBlendFunc(GL_SRC_ALPHA,GL_ONE_MINUS_SRC_ALPHA);
glHint (GL_LINE_SMOOTH_HINT,GL_DONT_CARE);

glLineWidth (2.0);
glColor4f (0.0f , 0.0f , 0.0f , 1.0f );
auxWireSphere (1.0);

glDisable (GL_LINE_SMOOTH);
glDisable (GL_BLEND);
```

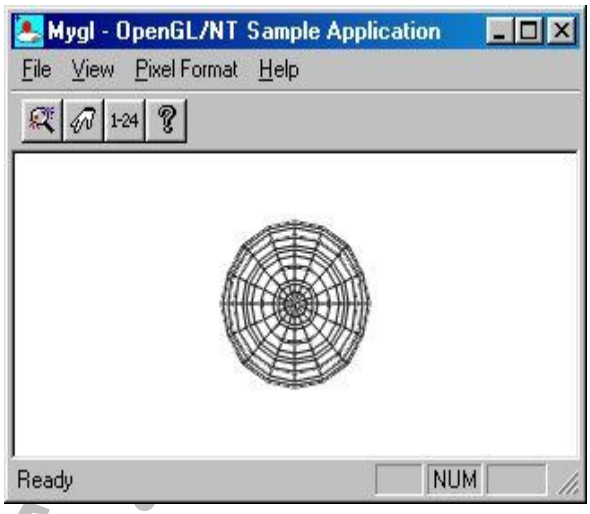
KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN



Hình 10.4: Hình mạng nhện với antialiasing



Hình 10.5: Hình mạng nhện không dùng antialiasing

10.3.Sương Mù:

Không khí cũng như mọi vật chất khác không hoàn toàn trong suốt nên ánh sáng đi qua nó sẽ bị hấp thụ một phần hay toàn bộ. Nói một cách khác, đối tượng càng ở xa thì thể hiện của nó càng mờ nhạt. OpenGL có thể xem sương mù như một loại kính lọc có hiệu quả càng lớn trên đối tượng càng xa. Bằng cách chọn lựa các hàm sương mù, màu sắc, mật độ v.v... ta có thể quyết định mức độ hiệu quả.

10.3.1.Kích Hoạt Fog :

Fog được kích hoạt như sau :

```
glEnable (GL_FOG);
```

Hằng GL_FOG báo cho OpenGL kích hoạt hiệu ứng sương mù

10.3.2.Chọn Hàm Sương Mù :

Giống như pha trộn sương mù được sử dụng theo nhiều cách. Để báo cho OpenGL biết loại hàm sương mù nào được sử dụng khi xác định màu thể hiện các pixel, hàm glFog() được gọi như sau:

```
glFogi (GL_FOG_MODE , GL_LINE);
```

Đối số đầu tiên của hàm xác định thông số sương mù muốn thiết lập, có thể là GL_FOG_DENSITY, GL_FOG_END, GL_FOG_INDEX , GL_FOG_MODE hay GL_FOG_START. Đối số thứ hai là giá trị thiết lập cho thông số đã xác định bởi đối số thứ nhất. Với GL_FOG_MODE, các giá trị có thể dùng ở đối số thứ hai là GL_EXP, GL_EXP2, GL_LINEAR (glFog() có bốn phiên bản).

GL_EXP và GL_EXP2 sử dụng các hàm số mũ để xác định màu của pixel sương mù. Tác dụng của GL_EXP2 lớn hơn tác dụng của GL_EXP. Do OpenGL quản lý hầu hết các chi

KILOBOOKS
Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

, các hàm GL_EXP và GL_EXP2 dễ sử dụng trong chương trình. Nhưng GL_LINEAR n cung cấp hiệu ứng sương mù trên cơ sở kích thước vùng sương mù cho phép người dùng kiểm soát nhiều hơn. Với GL_LINEAR có thể báo cho OpenGL chính xác nơi bắt đầu và kết thúc sương mù trong cảnh bằng lời gọi glFogf() :

```
glFogf (GL_FOG_START , 0.0f);
glFogF (GL_FOG_END, 10.0f );
```

Hằng GL_FOG_START báo cho OpenGL thiết lập điểm bắt đầu sương mù. Đối số thứ hai là khoảng cách kể từ điểm nhìn mà hiệu ứng fog bắt đầu. Hằng GL_FOG_END báo cho OpenGL thết lập điểm kết thúc sương mù, với đối số thứ hai cũng là khoảng cách kể từ điểm nhìn. Chỉ có thể thiết lập điểm bắt đầu và kết thúc sương mù khi dùng GL_LINEAR. Việc thiết lập là không hiệu lực khi dùng GL_EXP và GL_EXP2 . Tuy nhiên khi dùng GL_EXP và GL_EXP2 ta có thể thiết lập mật độ sương mù bằng lời gọi glFog() với GL_FOG_DENSITY làm đối số thứ nhất và giá tr ị mật độ làm đối số thứ hai.

Khoảng cách giữa điểm bắt đầu và kết thúc sương mù càng lớn thì hiệu quả sương mù càng kém. Ví dụ với điểm bắt đầu là 0.0 nếu điểm kết thúc là 5.0 thì các đối tượng trong vùng này sẽ chịu tác dụng sương mù nhiều, nhưng nếu điểm kết thúc là 20.0 thì hiệu quả sương mù bị giảm rõ rệt do tác dụng bị trải ra trên khoảng cách lớn hơn.

10.3.3.Thiết Lập Màu Sương Mù:

Với việc thiết lập màu sương mù có thể đạt được hiệu quả lớn hơn màu sắt sử dụng trong cảnh. Trong đa số trương hợp thường dùng sương mù màu trắng với độ sáng vừa phải:

```
GLfloat fogColor[]={0.6f , 0.6f , 0.6f , 1.0f};
GlFogfv (GL_FOG_COLOR , fogColor);
```

Hai đối số của hàm là thông số sương mù muốn thiết lập và địa chỉ mảng chứa giá trị thiết lập cho thông số. Có thể có được các kết quả thú vị bằng cách sử dụng các màu khác màu trắng. Sương mù màu đen cho phép tạo cảnh ban đêm hay trời tối.

10.3.4.Nêu Gọi Ý Sương Mù :

Tong antialiasing, ta đã làm quen với glHint(). Hàm này cũng có thể sử dụng trong hiệu ứng sương mù vớiGL_FOG_HINT làm đối số thứ nhất:

```
glHint (GL_FOG_HINT , GL_DONT_CARE);
```

Cũng như trước đây GL_DONT_CARE cho phép OpenGL sử dụng tùy ý các phương pháp sương mù.

10.3.5.Thực Hiện Sương Mù Cho Cảnh 3_D :

Trước khi thực hiện sương mù cho cảnh 3-D phải kích hoạt chiếu sáng và kiểm tra chiều sâu cũng như thiết lập các thông số chiếu sáng và vật liệu;

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

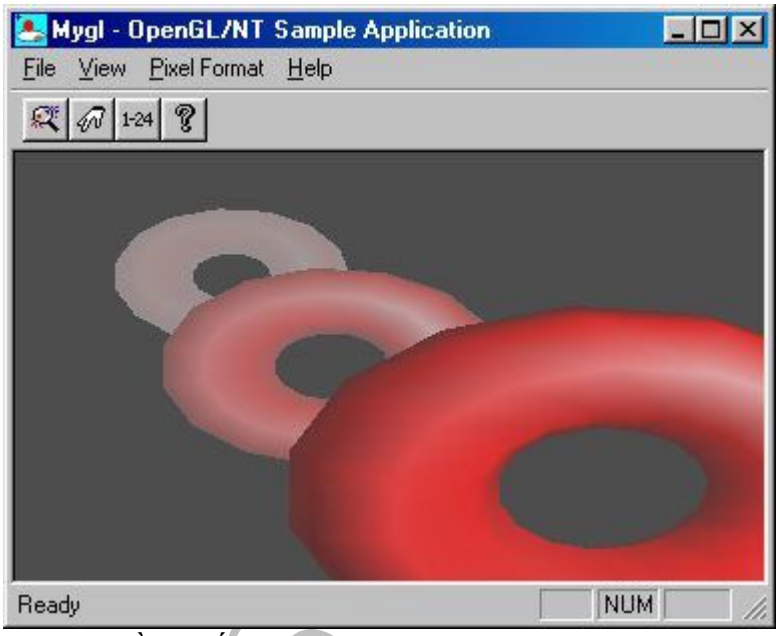
ı đó thực hiện các phép biến hình modelview và vẽ đối tượng. OpenGL sử dụng sương khi được chỉ định trên cơ sở khoảng cách đối tượng từ điểm nhìn. Đoạn mã sau thể ...ện như hình 10.6 :

```
glClearColor (0.3f , 0.3f , 0.3f , 1.0f );
glClear (GL_COLOR_BUFFER_BIT / GLDEPTH_BUFFER_BIT);
glShadeModel (GL_SMOOTH);
glEnable (GL_FOG):
Fogi (GL_FOG_MODE , GL_LINEAR);
Gloat fogColor[]={0.6f , 0.6f , 0.6f , 1.0f};
glFogfv (GL_FOG_COLOR , fogColor);
glFogfv (GL_FOG_START , 0.0f);
glFogfv (GL_FOG_END , 10.0f);
glHint (GL_FOG_HINT , GL_DONT_CARE);
glEnable (GL_LIGHTING);
glEnable (GL_LIGHT0);
glEnable (GL_DEPTH_TEST);
GLfloat lightAmbient []={0.1f , 0.1f , 01f , 0.0f};
GLfloat lightDiffuse[]={1.0f , 1.0f , 1.0f , 0.0f};
 GLfloat lightSpecular[]={1.0f , 1.0f , 1.0f , 0.0f};
GLfloat lightPosition[]= {2.0f , 0.5f , 3.0f , 1.0f };
GLfloat materialSpecular[]={1.0f , 1.0f , 1.0f , 0.0f};
GLfloat materialAmbDiff[]={1.0f , 0.0f , 0.0f , 0.0f);
glLightfv (GL_LIGHTO,GL_ AMBIENT, lightAmbient);
glLightfv (GL_LIGHTO , GL_DIFFUSE, lightDiffuse);
 glLightfv (GL_LIGHTO, GL_SPECULAR, lightSpecular);
glLightfv (GL_LIGHTO, GL_POSITION , lightPosition);
 glMaterialfv (GL_FRONT , GL_AMBIENT_AND_DIFFUSE ,
materialAmbDiff);
 glMaterialfv ( GL_FRONT , GL_SPECULAR , materialSpecular );
 glMaterialf ( GL_FRONT , GL_SHININESS , 30.0f );
 glTranslatef(-2.0f , 2.0f , -6.0f);
 auxSolidTorus (0.5 , 1.0);
 glTranslatef(1.5f ,-2.0f , 3.0f);
 auxSolidTorus (0.5 , 1.0);
 glTranslatef(1.5f ,-1.0f , 3.0f);
 auxSolidTorus (0.5 , 1.0);
 glDisable(GL_LIGHTING);
 glDisable(GL_DEPTH_TEST);
glDisable(GL_FOG);
```

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN



Hình 10.6
Thực hiện sương mù cho cảnh 3-D.

Hình thể hiện ba đường gờ tròn, xếp xa dần điểm nhìn. Hình gần nhất giữ được hình màu đỏ, trong khi hình xa nhất hầu như trắng mờ hoàn toàn do đối tượng càng ở xa thì tác dụng sương mù lên nó càng lớn.

Chương11:Display List

11.1.Định Nghĩa:

Là một nhóm lệnh OpenGL được lưu lại sau khi thực thi. Có thể sử dụng immediate mode và display list trong cùng một chương trình, các lệnh sẽ được thực thi theo đúng trật tự mà nó xuất hiện.

11.2.Tại Sao Phải Sử Dụng Display List ?:

Khi ta có ý định vẽ lại hoặc thay đổi trạng thái nhiều lần một đối tượng, sử dụng display list để làm tăng khả năng thực thi chương trình.

Trong mô hình client – server : display list sẽ làm giảm sự hao phí thời gian truyền dữ liệu.

Ví dụ:

```
Gluint thetorus;
Static void torus (int numc, int numt )
   {
            int i, k, j;
            double s, t, x, y, z, twopi;
            twpoi =2* (double) 3.14;
            for (i =0; i<numc ; j++) {
               glBegin (GL_QUAD_STRIP);
               for (j =0; j<numt; j++) {
                  for (k =0; k>=0; k--) {
                     s = (i+k)%numc +0.5;
```

KILOBOOKS
Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

```
                t = j%numt;

                x = (1 +0.1*cos (s*twopi/numc))
                *cos (t*twpoi / numt);
                y = (1 +0.1*cos (s*twopi/numc))
                *sin (t*twpoi / numt);
                z =0.1*sin (s*twopi / numc;
            glVertex3f (x, y, z);
          }
          }
        glEnd();
      }
    static void init (void)
      {
          theTorus = glGenList (1);
          glNewList (theTorus, GL_COMPILE);
          torus (8, 25);
              glEndList ();
          }
void Drawsine (void)
  {
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f (1.0, 1.0, 0.0);
    glCallList (theTorus);
    glFush ();
  }
```

11.3.Các Tính Chất Của Display List:

Nếu một display list tạo ra thì không thể nào sửa đổi. Display list cũng làm việc tốt với các lệnh của thư viện gl. Có nhiều cách để thực hiện display list.

Sự thực thi của display list không chậm hơn thực thi các lệnh được chứa bên trong nó một cách độc lập.

11.4.Các Trường Hợp Có Thể Sử Dụng Display List:

- Các tác vụ trên ma trận.
- Raster các bitmaps và các ảnh.
- Đặc tính nguồn sáng, chất liệu và mô hình chiếu sáng.
- Textures.
- Polygon stiple pattern
-

11.5.Nhược Điểm Của Dislay List:

- Tính không biến đổi của dislay list.
- Tốn vùng nhớ nếu cần lưu trữ dữ liệu được phân chia từ dislay list.
- Nếu danh sách là nhỏ thì không hiệu quả.

11.6.Tạo Và Thực Thi Một Dislay List:

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

11.6.1.Đặt Tên Vào Một Dislay List:

Hàm glNewList (Gluint list, Glenum mode) và glEndList (void) dùng để bắt đầu và kết thuc một dislay list. List là tên của một dislay list, mỗi dislay list được đặt tên bằng một giá trị integer. Giá trị của mode có thể là:GL_COMPLE and GL_COMPLE_AND_EXCUTE. Chỉ có một dislay list tại một thời điểm.

Nên để tránh sự trùng tên ta dùng hàm glGenList (Glsizei range).

Khi không sử dụng dislay list ta dùng hàm:

GlDeleteList (Glint list, Glsizei range) để xóa dislay list đã định nghĩa.

Hàm glIsList (Gluint list) dùng để kiểm tra dislay list có định nghĩa chưa.

11.6.2. Những Lệnh Không Được Chứa Trong Dislay List:

glColorPointer ()
glFlush ()
glNormalPointer ()
glDeleteList ()
glGenList ()
glPixelStore ()
glDisableClienState ()
glGet* ()
glReadPixel ()
glEdgeflagPointer (),…

11.6.3.Thực Thi Một Dislay List:

Dislay list được thực thi bằng glCallList (Gluin list), có thể gọi glCallList (Gluin list) ở bất cứ nơi nào trong chương trình miễn là OpenGL context truy cập dislay list tích cực.

11.6.4.Cấp Bậc Dislay List:

Là dislay list mà nó được định nghĩa trong một dislay list khác. Để tránh sự đệ qui vô hạn chế số phần tử trong dislay list.

OpenGL cho phép tạo một dislay list mà nó gọi tới một cái khác nhưng cái này chưa được nghĩa, kết quả là không có gì xảy ra.

11.7. Quản Lý Biến Trạng Thái Trong Dislay List:

Các biến trạng thái của OpenGL sau khi ra khỏi dislay list vẫn còn ảnh hưởng đến các lệnh tiếp theo trong chương trình.

Không thể dùng glGet*() trong dislay list, do đó để lưu trữ và phục hồi biến trạng thái phải dùng glPushAttr() và glPopAttrib().

Ví dụ:

```
glNewList  (listIndex, GL_COMPLE) ;
glPushMatrix  () ;
glPushAttrib  ( GL_ CURRENT_BIT) ;
glColor3f( 1.0,  0.0,  0.0) ;
glBegin ( BOLYGON );
```

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

```
        glvertex2f  ( 0.0,  0.0);
        glvertex2f  ( 1.0,  0.0);
        glvertex2f  ( 1.0,  1.0);
glEnd () ;
glTranslatef (1.5,  0.0,  0.0 );
glPopAttrib () ;
glPopMatrix  () ;
glEndList ();
```

Chương 12:<u>QUADRIC:</u>

Trong GLU có các hàm hổ trợ cho việc biểu diễn một số dạng 3D cơ bản như các khối cầu, cylinder, các đĩa… được gọi chung là Quadric.

Để sử dụng đối tượng Quadric, thực hiện theo các bước sau:

+ Tạo một đối tượng quadric:

GLUquadricObj* gluNewQuadric(void): tạo một đối tượng quadric và trả về một pointer.

+ Chỉ định những thuộc tính cho đối tượng quadric bằng: gluQuadricOrientation (), gluQuadricDrawStyle (), gluQuadricNormal ().

+ Quản lý sự xuất hiện lỗi trong quá trình biểu diễn.

void glQuadricCallback(GLUquadric quad, Glenum which, Glvoid (CallbackFunc) ()).

+ Gọi những routines biểu diễn những đối tượng mong muốn.

Void GluSphere(GLUquadricObj *qobj, Glduoble radius, Glint slices, Glint stacks) vẽ hình cầu bán kính radius tâm tại gốc.

Void gluCylinder(GLUquadricObj *qobj, Gldouble basicRadius, Gldouble topRadius, Gldouble height, Glint slices, Glint stacks) vẽ hình trụ có trục là trục z một mặt tiếp xúc với mặt z=0 và có chiều cao là height

Void gluDisk(GLUquadricObj *qobj, Gldouble innerRadius, Gldouble outerRadius, Glint slices, Glint stacks) vẽ hình đĩa có bán kính đường tròn trong interRadius và bán kính đường tròn ngoài outerRadius.

Void gluPartialDisk(GLUquadric *qobj, Gldouble innerRadius, Gldouble outerRadius, Glint slices, Glint rings, Gldouble starAngle, Gldouble sweepAngle);

+ Loại bỏ vùng void gluDeleteQuadric(GLUquadric*quad);

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

PHẦN 2: ỨNG DỤNG MÔ PHỎNG CÁC GIẢI THUẬT ĐỒ HỌA 3 –D VỚI OPENGL

Chương 1: Tổng Quan

Một trong những ưu điểm quan trọng của đồ họa là cho phép dễ dàng thao tác lên các đối tượng đã được tạo ra. Một nhà quản lý có nhu cầu phóng to hay thu nhỏ một đối tượng trong một báo cáo, hay những nhà thiết kế muốn xem sản phẩm của mình từ các góc độ khác nhau,… Tất cả các thao tác này có thể được hỗ trợ một cách dễ dàng nhờ vào các phép biến đổi hình học. Các phép biến đổi hình học sẽ làm thay đổi mô tả về tọa độ của các đối tượng, từ đó làm cho tọa độ thay đổi về hướng, kích thước và hình dạng. Các phép biến đổi hình học cỏ sở gồm: tịnh tiến (tranlation), quay (rotation) và biến đổi tỉ lệ. Ngoài ra một số biến đổi khác cũng thường được áp dụng đó là phép đối xứng (reflection) và biến dạng (shearing).

Có hai quan điểm về phép biến đổi hình học đó là: biến đổi đối tượng (object transformation) và biến đổi hệ tọa độ (coordinate transformation). Biến đổi đối tượng là thay đổi các điểm mô tả nó theo một quy tắc nào đó, còn biến đổi hệ tọa độ là tạo ra một hệ tọa độ mới và tất cả các điểm mô tả đối tượng sẽ được chuyển về hệ tọa độ mới. Hai cách này có mối liên hệ chặt chẻ với nhau. Ở đây chúng ta chỉ nói đến phép biến đổi hệ toạ độ.

1.1. Một Số Khái Niệm Liên Quan

1.1.1 Phép Biến Đổi Affine

Phép biến đổi affine là phép biến đổi tuyến tính, khả nghịch. Phép biến đổi này bảo toàn tính song song của các đường thẳng cũng như bảo toàn tính tỉ lệ về khoảng cách giữa các đoạn thẳng. Tuy nhiên , phép biến đổi này không bảo toàn góc nghiêng và chiều dài các đoạn thẳng. Các phép biến đổi này cũng bảo toàn về tỉ lệ về khoảng cách. Các hệ trục toạ độ quy ước bàn tay phải và bàn tay trái

Hình 1.1- các hệ tọa độ quy ước bàn tay trái (a) và quy ước bàn tay phải (b)

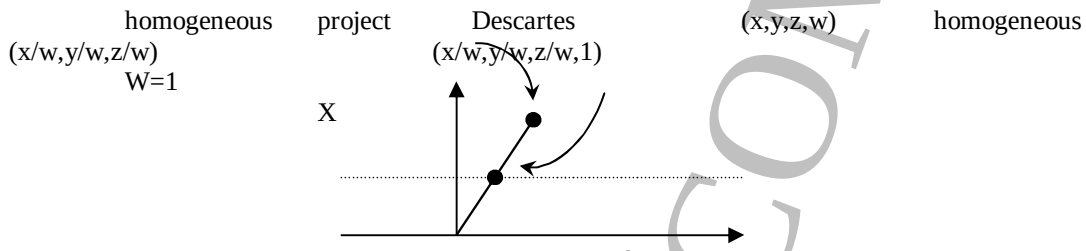
KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

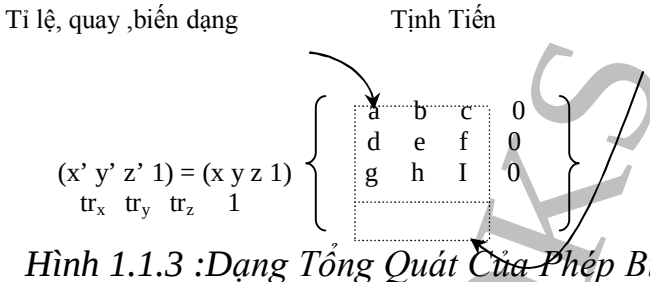
http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

.2. Hệ Tọa Độ Thuần Nhất:

Trong một hệ tọa độ thuần nhất , mỗi điểm (x,y,z) trong không gian Dercarters được biểu diễn bởi 4 tọa độ trong không gian 4 chiều thu gọn (hx,hy,hz,h). Để tiện lợi, người ta thường chọn h=1. Như vậy, một điểm (x,y,z) trong hệ tọa độ dercarts sẽ biến thành điểm (x,y,z,1) trong hệ tọa độ thuần nhất (với w <> 0) sẽ tương ứng với điểm (x/w,y/w,z/w) trong hệ tọa độ descartees



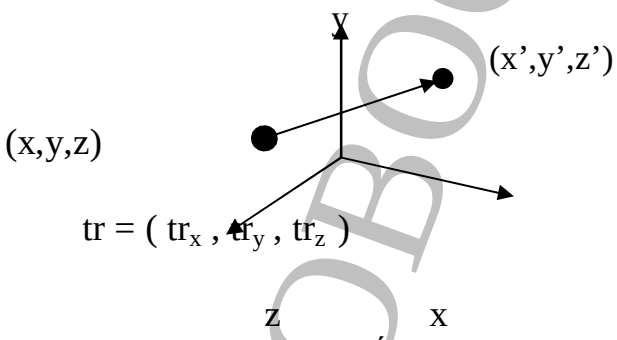
1.1.3. Dạng Ma Trận Cho Phép Biến Đổi Affine Trong Hệ Tọa Độ Thuần Nhất



Hình 1.1.3 :Dạng Tổng Quát Của Phép Biến Đổi Affine Ba Chiều

1.2. Các Phép Biến Đổi:

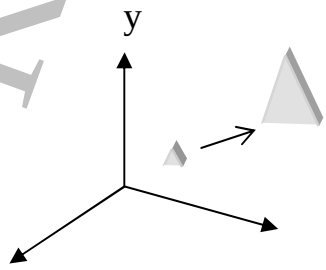
1.2.1. Phép Tịnh Tiến:



Hình1.2.1: Phép Tịnh Tiến Với vecto tịnh tiến tr = (tr_x , tr_y , tr_z)

Vector tịnh tiến tr trong phép biến đổi ba chiều có tác động rất trực quan: mỗi điểm được dịch di một khoảng là tr_x , tr_y , tr_z , theo ba trục.

1.2.2. Phép Biến Đổi Tỉ Lệ



KILOBOOKS.COM
Nếu bạn cần file word? Xin hãy nhấp vào đây!

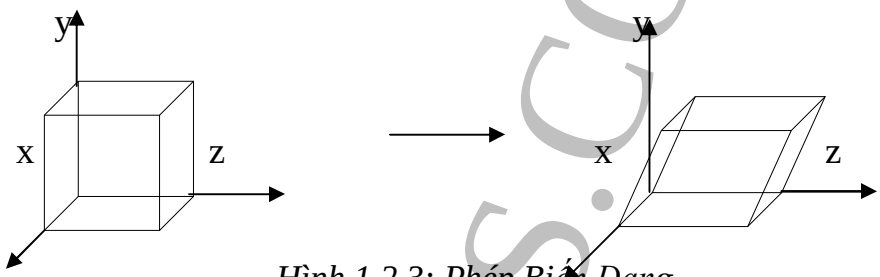
x z

Hình 1.2.2 :Phép biến đổi tỉ lệ

Như hình 1.2.2: đối tượng phóng to gấp đôi, đồng thời với tác động của phép biến đổi làm cho đối tượng bị đẩy ra xa góc tọa độ hơn

1.2.3. Phép Biến Dạng:

Biến dạng theo bất kỳ trục tọa độ nào cũng bị ảnh hưởng bởi tọa độ ứng với hai trục còn lại.



Hình 1.2.3: Phép Biến Dạng

1.2.4. Phép Quay:

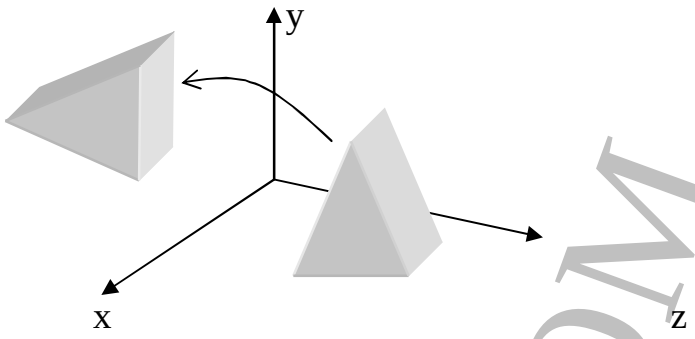
1.2.4.1. Quay quanh một trục tạo độ:

Khác với phép quay trong hai chiều quanh 1 điểm bất kỳ, trong ba chiều ta có phép quay quanh một trục tọa độ. Ở đây ta sử dụng hệ tọa độ quy ước bàn tay trái và quy định chiều quay dương là ngược chiều kim đồng hồ.

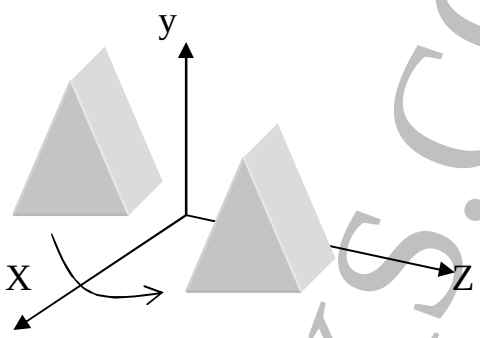
KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN



Hình 1.2.4.1: Phép Quay Quanh Trục Z



Chương 2: Xây dựng ứng dụng mô phỏng thuật giải đồ họa 3-D

Phần tiếp theo của luận văn sử dụng OpenGL để xây dựng một ứng dụng thuật giải đồ họa 3-D.

2.1.Xây dựng ứng dụng OpenGL:

Các bước sau thệ hiện quá trình xây dựng một ứng dụng có tên OpenGL, sử dụng thư viện OpenGL để minh họa.

2.1.1. Tạo ứng dụng cơ bản và sửa đổi giao diện:

2.1.1.1. Dùng AppWizard tạo ứng dụng OpenGL với các chọn lựa sau trong các hộp thoại:

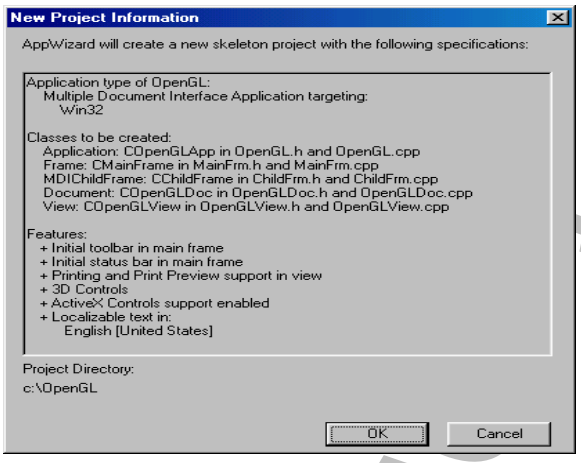
Bước 1: Multil document.
Bước 2: Mặc định.
Bước 3: Mặc định.
Bước 4: Use 3D Controls.
Bước 5: As a statically linked library.
Bước 6: Mặc định.

Bảng tổng kết chọn lựa như sau:

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN



Hình 2.1.1.1: Bảng tổng kết chọn lựa ứng dụng OpenGL

2.1.1.2. Tạo hộp thoại có ID là IDD_FORM_COMMAND và các Picture, Check box, Radio button, Slider có ID là IDC_FRAME_COLOR_BACK, IDC_FRAME_COLOR_LIGHT_AMBIENT, IDC_CHECK_ANTIALIAS, IDC_CHECK_SMOOTH, IDC_CHECK_LIGHTING, IDC_RADIO_MODEL_0, IDC_RADIO_MODEL_1, IDC_RADIO_MODEL_2, IDC_CHECK_VROTATION, IDC_CHECK_LINK_SCALE, IDC_SLIDER_X, IDC_SLIDER_Y, IDC_SLIDER_Z (hình 2.2)

2.1.1.3. Dùng ClassWizard tạo lớp CformCommandView cho hộp thoại IDD_FORM_COMMAND, với các biến kiểu BOOL như m_Antialias, m_Lighting, m_LinkScale, m_Smooth, m_Vrotate, và kiểu CStatic như m_ControlBackColor, m_ControlColorLightAmbient, kiểu CsliderCtrl như m_SliderScaleX, m_SliderScaleY, m_SliderScaleZ (hình 2.1.1.3)

Hình 2.1.1.3: Hộp thoại FormCommandView

2.1.2. Thêm các biến thành viên, các hàm đáp ứng thông báo, hàm COMMAND, các hàm thanh viên, hoàn chỉnh lớp COpenGLView

2.1.2.1. Thêm các biến thành viên public sau cho lớp COpenGLView:

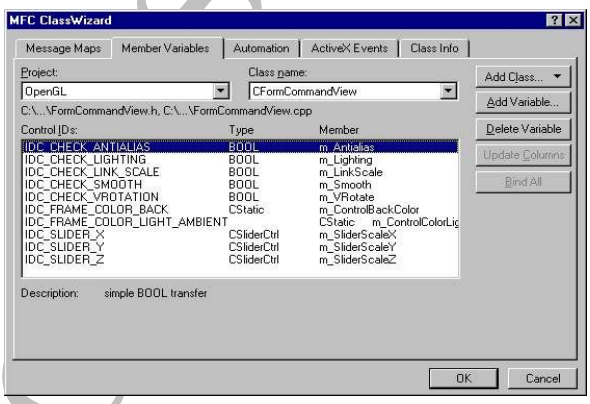
```
float    m_zTranslation;
float    m_yTranslation;
float    m_xTranslation;
float    m_zScaling;
float    m_yScaling;
```

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

```
float    m_xScaling;
float    m_zRotation;
float    m_yRotation;
float    m_xRotation;
CPoint    m_RightDownPos;
BOOL    m_RightButtonDown;
CPoint    m_LeftDownPos;
BOOL    m_LeftButtonDown;
HGLRC    m_hGLContext;
int    m_GLPixelIndex;
HCURSOR    m_CursorRotation;
float    m_ClearColorGreen;
float    m_ClearColorRed;
float    m_ClearColorBlue;
```

2.1.2.2. Dùng MFC ClassWizard thêm các hàm đáp ứng thông báo WM_CREATE, WM_DETROY, WM_LBUTTONDOWN, WM_LBUTTONUP, WM_MOUSEMOVE, WM_PAINT, WM_RBUTTONDOWN, WM_RBUTTONUP, WM_SIZE, và WM_TIMER.



Hình 2.1.2.2: Tạo lớp CFormCommandView

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

ềm mã cho các hàm OnCreate(), OnDetroy(), OnSize(),
LButtonDown(),OnLButtonUp(),OnRButtonDown(),OnRButtonUp(),
...MouseMove(), OnPaint() của lớp COpenGLView như bảng kê 2.3.2

2.1.2.3. Thêm các hàm thành viên public sau vào lớp COpenGLView:

```
void   InitGeometry(void);
BOOL   CreateViewGLContext(HDC hDC);
BOOL   SetWindowPixelFormat(HDC hDC);
```

Mã các hàm trên được cài đặt trong bảng kê 2.3.

2.1.2.5. Thêm vào sau chỉ thị #endif của OpenGLView.cpp các dòng sau:

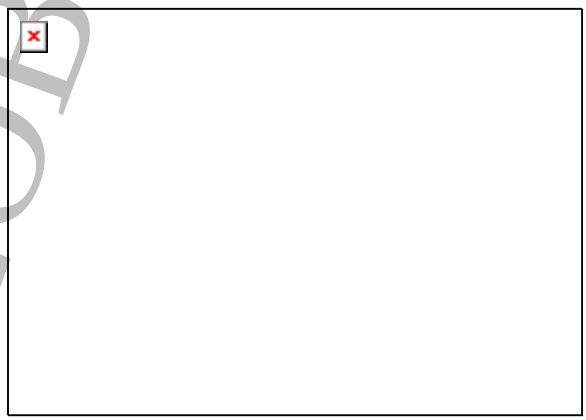
```
#include "gl\glu.h"
#include "gl\gl.h"
```

2.1.2.6. Thêm vào Project các file thư viện Glaux.lib, OpenGL32.lib và glu32.lib (đường dẫn ..\Vc98\lib)

Hình 2.1.2.5: Thêm các file thư viện

2.2 Cách làm việc của ứng dụng:

Hình 2.2 là cửa sổ khi bắt đầu ứng dụng thể hiện một hình cầu 3-D .Thông qua hình cầu này ta có thể minh họa được một số giải thuật đồ họa 3-D.



Hình 2.2: Trước khi thực hiện các phép biến hình

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

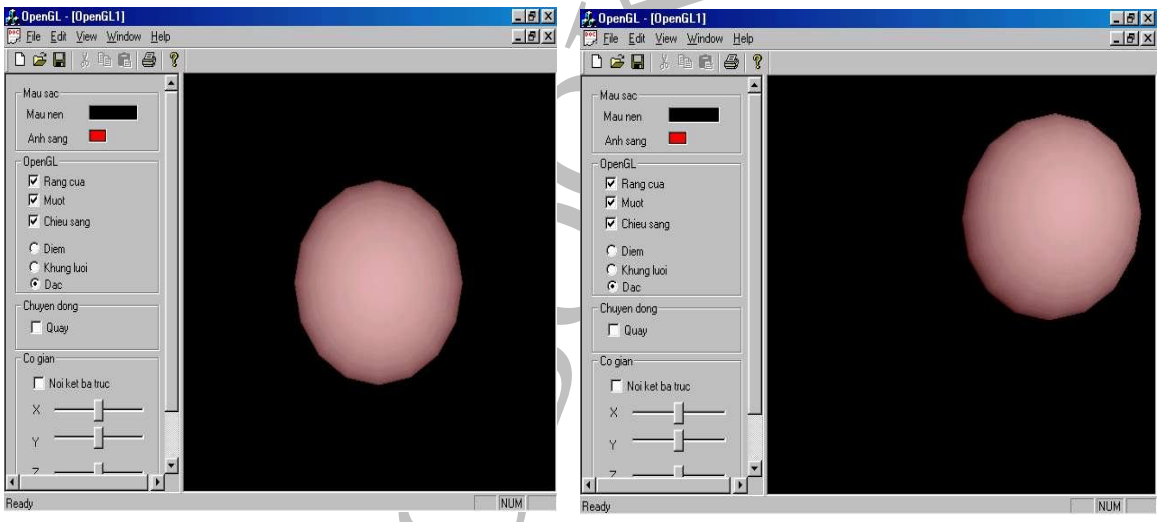
2.1. Các phép biến hình OpenGL.

Trước khi thực hiện các phép biến hình thì đối tượng hình cầu được khởi động bằng các giá trị:

```
m_xRotation = 0.0f;
m_yRotation = 0.0f;
m_zRotation = 0.0f;

m_xTranslation = 0.0f;
m_yTranslation = 0.0f;
m_zTranslation = -5.0f;

m_xScaling = 1.0f;
m_yScaling = 1.0f;
m_zScaling = 1.0f;
```

2.2.1.1.Phép tịnh tiến:

Để thực hiện phép tịnh tiến ta dùng phím phải của con chuột để di chuyển hình cầu, các biến m_xTranslation, m_yTranslation, m_zTranslation nhận giá trị mới thì InvalidateRect được gọi để vẽ lại hình.



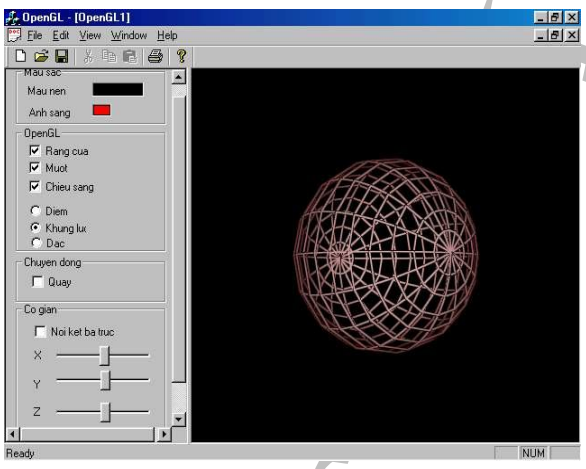
KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

2.2.1.2. Phép Quay:

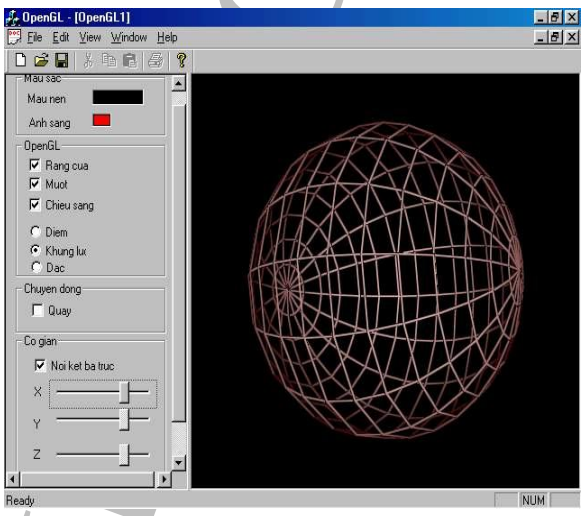
Ngược lại với phép tịnh tiến để thực hiện phép quay ta dùng phím trái của con chuột để điều khiển hình cầu quay theo các trục tuỳ ý, lúc này các biến m_xRotation, m_yRotation, m_zRotation nhận giá trị mới thì InvalidateRect được gọi để vẽ lại hình.



Hình 2.2.1.2: Sau khi thực hiện phép quay

2.2.1.3. Phép Co giãn:

Để thực hiện phép co giãn hình cầu ta thông qua ba thanh trượt tương ứng với ba trục X, Y, Z. Thanh trượt có giá trị từ 1 – 100, khi ta di chuyển các thanh trượt các biến m_xScaling, m_yScaling, m_zScaling nhận giá trị mới thì InvalidateRect được gọi để vẽ lại hình.



Hình 2.2.1.3: Sau khi thực hiện phép co giản theo ba trục

2.2.2. Kiểu bóng , chiếu sáng, hiệu ứng răng cưa và các thao tác vẽ đối tượng 3-D:

2.2.2.1. Kiểu bóng:

Một phương diện quan trọng của OpenGL là kiểu bóng dùng thể hiện đối tượng trong cảnh. Co hai kiểu bóng: mượt (smooth) và phẳng (flat). Để chọn kiểu bóng, sử dụng hàm glShadeMode() như sau:

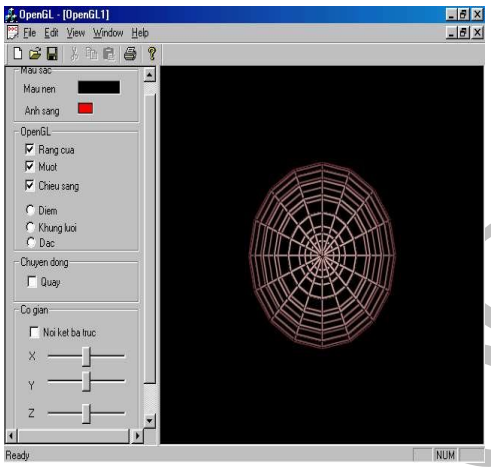
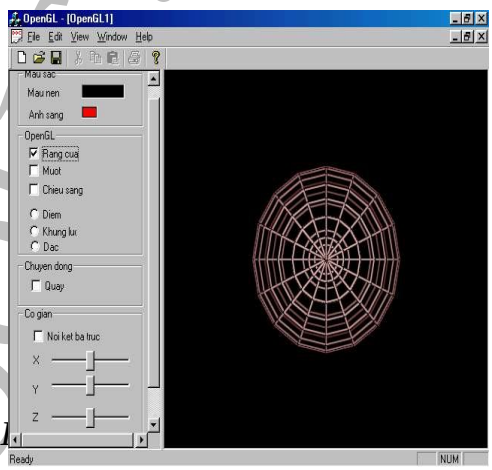
```
glShadeModel(GL_FAT);
```

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

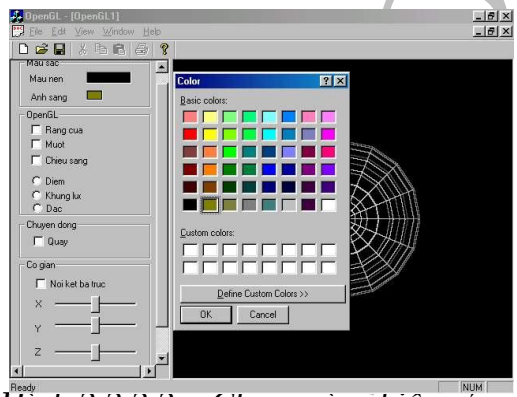
Đối số đơn của hàm là hằng thể hiện kiểu bóng được chọn, là GL_SMOOTH, hoặc GL_FAT. Với bóng mượt, OpenGL phải xác định màu bóng cho từng pixel trên đa giác. Trong đối số bóng phẳng, toàn bộ đa giác dùng chung một màu bóng. Hàm OnCheckSmooth() kiểm tra kiểu bóng.

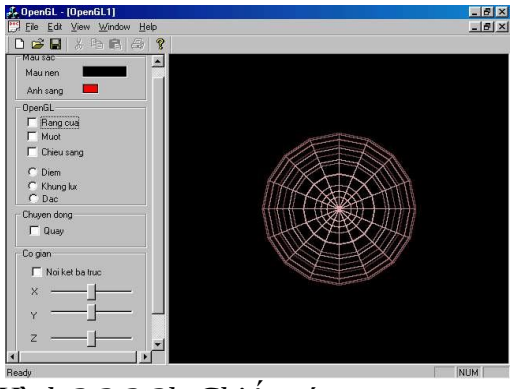
```
void CFormCommandView::OnCheckSmooth()
{
// TODO: Add your control notification handler code here
m_Smooth = !m_Smooth;
If(m_Smooth)
   glShadeModel(GL_SMOOTH);
else
   glShadeModel(GL_FLAT);
   this->GetRenderView()->InvalidateRect(NULL,FALSE);
```

2.2.2.2. Chiếu sáng đối tượng:

Đầu tiên đối tượng được vẽ có màu trong suốt để đối tượng có ánh sáng chiếu với màu sắc tùy ý chương trình sử dụng CColorDialog cho người dùng tùy ý chọn.


Hình 2.2.2.2a: Chọn màu chiếu sáng

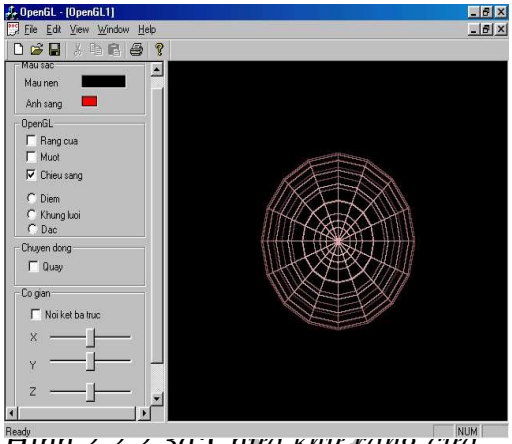

Hình 2.2.2.2b:Chiếu sáng

2.2.2.3. Hiệu ứng răng cưa:

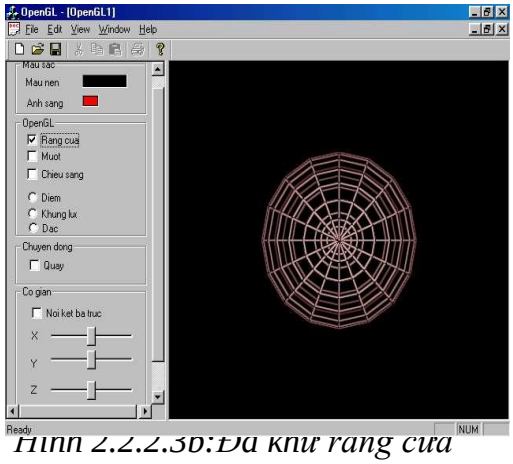
Nếu bạn cần file word? Xin hãy nhấp vào đây!

Trên màn hình, một đường thẳng là tập hợp của các pixel được chiếu sáng trong hệ ng kẽ ô vuông. Do đó, chỉ có đường nằm ngang hoặc thẳng đứng là được vẽ một cách đúng sẽ. Còn các đường nghiêng sẽ có hiện tượng răng cưa. Hàm OnCheckAntialias() sẽ làm giảm hiệu ứng răng cưa:

```
void CFormCommandView::OnCheckAntialias()
    {
      m_Antialias = !m_Antialias;
      if(m_Antialias)
    {
      glEnable(GL_LINE_SMOOTH);
      glEnable(GL_BLEND);
            glBlendFunc(GL_SRC_ALPHA,GL_ONE_MINUS_SRC_ALPHA);
      glHint(GL_LINE_SMOOTH_HINT,GL_NICEST);
      glLineWidth(1.5);
    }
    else
    {
      glDisable(GL_LINE_SMOOTH);
      glDisable(GL_BLEND);
    glBlendFunc(GL_SRC_ALPHA,GL_ONE_MINUS_SRC_ALPHA);
        glHint(GL_LINE_SMOOTH_HINT,GL_NICEST);
      glLineWidth(1.0);
    }
    GetRenderView()->InvalidateRect(NULL,FALSE);
      }
```



Hình 2.2.2.3a:Chưa khử răng cưa



Hình 2.2.2.3b:Đã khử răng cưa

2.2.2.4. Các thao tác vẽ đối tượng 3-D:

Đa giác là hình tạo bởi các đường nối (cạnh) giữa một tập hợp các vertex. Một đa giác có ít nhất ba cạnh. Như vậy, đa giác đơn giản nhất là tam giác.

OpenGL có thể vẽ đa giác như các điểm, các đường ngoài, hay đối tượng đặc. OpenGL còn có thể điền đầy đa giác với mẫu đã được định nghĩa. Thêm vào đó, một đa

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

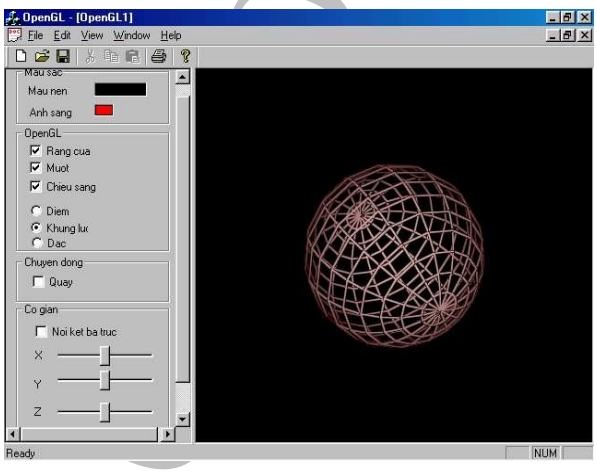
http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

c OpenGL có hai mặt trước và sau có thể vẽ theo cách riêng, và có thể quay một đa c để xem mặt bên kia.

Hàm glPogonMode() được dùng để vẽ đa giác:

Chương trình cho phép vẽ hình cầu với các chế độ điểm (GL_POINT), khung lưới (GL_LINE), đa giác đặc (GL_FILL).

```
void CFormCommandView::OnRadioModel0()
    {
   glPolygonMode(GL_FRONT_AND_BACK,GL_POINT);
   this->GetRenderView()->InvalidateRect(NULL,FALSE);
    }
void CFormCommandView::OnRadioModel0()
    {
      glPolygonMode(GL_FRONT_AND_BACK,GL_LINE);
   this->GetRenderView()->InvalidateRect(NULL,FALSE);
    }

    void CFormCommandView::OnRadioModel0()
    {
      glPolygonMode(GL_FRONT_AND_BACK,GL_FILL);
      this->GetRenderView()->InvalidateRect(NULL,FALSE);

    }
```

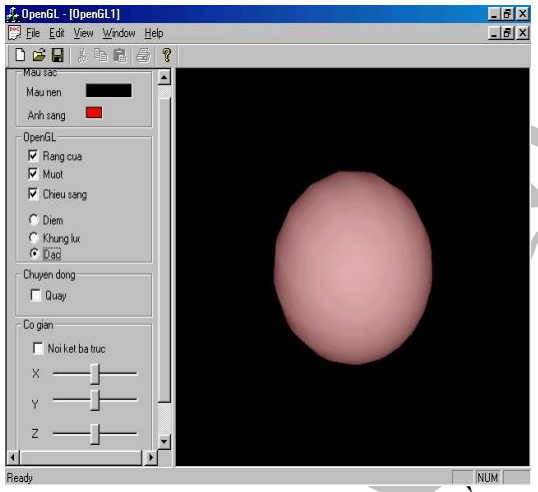


KILOBOOKS.COM

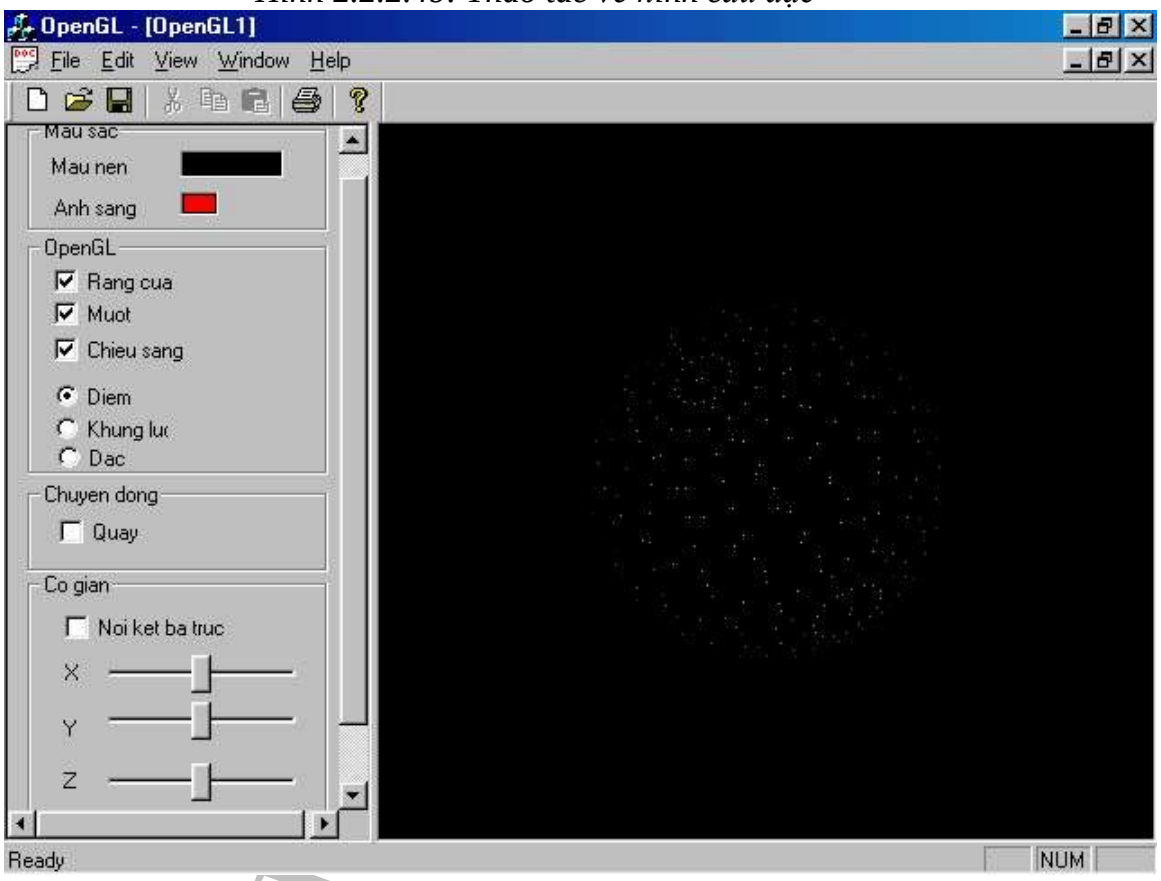
Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

Hình 2.2.2.4a: Hình cầu vẽ bằng khung lưới



Hình 2.2.2.4b: Thao tác vẽ hình cầu đặc



Hình 2.2.2.4c: Hình cầu vẽ bằng điểm

2.3. Bảng kê chương trình:

Bảng kê 2.3.1 : **OpenGLView.h**

```
// OpenGLView.h : interface of the COpenGLView class
//
```

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

```
#if
!defined(AFX_OPENGLVIEW_H__B5E77933_7225_11D5_9ABA_BAADD0C6
8D13__INCLUDED_)
#define
AFX_OPENGLVIEW_H__B5E77933_7225_11D5_9ABA_BAADD0C68D13__I
NCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
class COpenGLDoc ;
class COpenGLView : public CView
{
protected: // create from serialization only
  COpenGLView();
  DECLARE_DYNCREATE(COpenGLView)

// Attributes
public:
  COpenGLDoc* GetDocument();
// Operations
public:
// Overrides
  // ClassWizard generated virtual function overrides
  //{{AFX_VIRTUAL(COpenGLView)
  public:
  virtual void OnDraw(CDC* pDC);  // overridden to draw this view
  virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
  protected:
  virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
  virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
  virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);
  //}}AFX_VIRTUAL
// Implementation
public:
  float m_zTranslation;
  float m_yTranslation;
  float m_xTranslation;

  float m_zScaling;
  float m_yScaling;
  float m_xScaling;

  float m_zRotation;
```

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

```
  float m_yRotation;
  float m_xRotation;

  CPoint m_RightDownPos;
  BOOL m_RightButtonDown;
  CPoint m_LeftDownPos;
  BOOL m_LeftButtonDown;

  HGLRC m_hGLContext;
  int m_GLPixelIndex;

  HCURSOR m_CursorRotation;
  float m_ClearColorGreen;
  float m_ClearColorRed;
  float m_ClearColorBlue;

  void InitGeometry(void);
  BOOL CreateViewGLContext(HDC hDC);
  BOOL SetWindowPixelFormat(HDC hDC);
  virtual ~COpenGLView();
#ifdef _DEBUG
  virtual void AssertValid() const;
  virtual void Dump(CDumpContext& dc) const;
#endif

protected:
// Generated message map functions
protected:
  //{{AFX_MSG(COpenGLView)
  afx_msg void OnPaint();
  afx_msg void OnMouseMove(UINT nFlags, CPoint point);
  afx_msg void OnTimer(UINT nIDEvent);
  afx_msg void OnSize(UINT nType, int cx, int cy);
  afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
  afx_msg void OnLButtonUp(UINT nFlags, CPoint point);
  afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
  afx_msg void OnDestroy();
  //}}AFX_MSG
  DECLARE_MESSAGE_MAP()
};

#ifndef _DEBUG  // debug version in OpenGLView.cpp
inline COpenGLDoc* COpenGLView::GetDocument()
   { return (COpenGLDoc*)m_pDocument; }
```

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

```
#endif

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the
previous line.

#endif //
!defined(AFX_OPENGLVIEW_H__B5E77933_7225_11D5_9ABA_BAADD0C6
8D13__INCLUDED_)
```

Bảng kê 2.3.2: **OpenGLView.cpp**

```
// OpenGLView.cpp : implementation of the COpenGLView class
#include "stdafx.h"
#include "OpenGL.h"
#include "OpenGLDoc.h"
#include "OpenGLView.h"
#include "gl\glu.h"
#include "gl\gl.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
/////////////////////////////////////////////////////////////////////////////
// COpenGLView
IMPLEMENT_DYNCREATE(COpenGLView, CView)

BEGIN_MESSAGE_MAP(COpenGLView, CView)
  //{{AFX_MSG_MAP(COpenGLView)
  ON_WM_PAINT()
  ON_WM_MOUSEMOVE()
  ON_WM_TIMER()
  ON_WM_SIZE()
  ON_WM_LBUTTONDOWN()
  ON_WM_LBUTTONUP()
  ON_WM_CREATE()
  ON_WM_DESTROY()
  //}}AFX_MSG_MAP
  // Standard printing commands
        ON_COMMAND(ID_FILE_PRINT, CView::OnFilePrint)
        ON_COMMAND(ID_FILE_PRINT_DIRECT, CView::OnFilePrint)
```

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

```
        ON_COMMAND(ID_FILE_PRINT_PREVIEW,
        CView::OnFilePrintPreview)
END_MESSAGE_MAP()

////////////////////////////////////////////////////////////////////////
// COpenGLView construction/destruction
COpenGLView::COpenGLView()
{
  // TODO: add construction code here
  m_hGLContext = NULL;
  m_GLPixelIndex = 0;
  // Mouse
  m_LeftButtonDown = FALSE;
  m_RightButtonDown = FALSE;
        //      m_CursorRotation = AfxGetApp()-
                >LoadCursor(IDC_CURSOR_ROTATION);

  // Colors
        COpenGLApp *pApp = (COpenGLApp *)AfxGetApp();
        m_ClearColorRed   = GetRValue(pApp->m_OptionColorGlBack);
        m_ClearColorGreen = GetGValue(pApp->m_OptionColorGlBack);
        m_ClearColorBlue  = GetBValue(pApp->m_OptionColorGlBack);

  InitGeometry();

}

COpenGLView::~COpenGLView()
{
}

        BOOL COpenGLView::PreCreateWindow(CREATESTRUCT& cs)
{
  return CView::PreCreateWindow(cs);
}

////////////////////////////////////////////////////////////////////////
// COpenGLView drawing

void COpenGLView::OnDraw(CDC* pDC)
{
  COpenGLDoc* pDoc = GetDocument();
  ASSERT_VALID(pDoc);
}
```

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

```
//////////////////////////////////////////////////////////////////////////
// COpenGLView printing

BOOL COpenGLView::OnPreparePrinting(CPrintInfo* pInfo)
{
  return DoPreparePrinting(pInfo);
}

        void COpenGLView::OnBeginPrinting(CDC* /*pDC*/, CPrintInfo*
        /*pInfo*/)
{
}

        void COpenGLView::OnEndPrinting(CDC* /*pDC*/, CPrintInfo*
        /*pInfo*/)
{
}
//////////////////////////////////////////////////////////////////////////
// COpenGLView diagnostics

#ifdef _DEBUG
void COpenGLView::AssertValid() const
{
  CView::AssertValid();
}

void COpenGLView::Dump(CDumpContext& dc) const
{
  CView::Dump(dc);
}

        COpenGLDoc* COpenGLView::GetDocument() // non-debug version is
        inline
{
                ASSERT(m_pDocument-
                >IsKindOf(RUNTIME_CLASS(COpenGLDoc)));
  return (COpenGLDoc*)m_pDocument;
}
#endif //_DEBUG
//////////////////////////////////////////////////////////////////////////
// COpenGLView message handlers

void COpenGLView::OnPaint()
```

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

```
{
  CPaintDC dc(this);

                COpenGLDoc *pDoc = (COpenGLDoc*)GetDocument();
  ASSERT_VALID(pDoc);
  // Useful in multidoc templates
  HWND hWnd = GetSafeHwnd();
  HDC hDC = ::GetDC(hWnd);
  wglMakeCurrent(hDC,m_hGLContext);

                glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

                glClearColor(m_ClearColorRed,m_ClearColorGreen,m_ClearColor
                Blue,1.0f);

  glPushMatrix();

  // Position / translation / scale
  glTranslated(m_xTranslation,m_yTranslation,m_zTranslation);
  glRotatef(m_xRotation, 1.0, 0.0, 0.0);
  glRotatef(m_yRotation, 0.0, 1.0, 0.0);
  glScalef(m_xScaling,m_yScaling,m_zScaling);

  // Start rendering...
  pDoc->RenderScene();

  glPopMatrix();

  // Double buffer
  SwapBuffers(dc.m_ps.hdc);
}

void COpenGLView::OnMouseMove(UINT nFlags, CPoint point)
{
  if(m_LeftButtonDown)
  {
     m_yRotation -= (float)(m_LeftDownPos.x - point.x)/3.0f;
     m_xRotation -= (float)(m_LeftDownPos.y - point.y)/3.0f;
     m_LeftDownPos = point;
     InvalidateRect(NULL,FALSE);
  }

  CView::OnMouseMove(nFlags, point);
}
```

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

```
void COpenGLView::OnTimer(UINT nIDEvent)
{
  switch(nIDEvent)
      {
      case 0:
          break;
      // Rotation
      case 1:
          m_yRotation += 5.0f;
          m_xRotation += 5.0f;
          m_zRotation += 5.0f;
          InvalidateRect(NULL,FALSE);
          break;
      default:
          {}
      }

}

void COpenGLView::OnSize(UINT nType, int cx, int cy)
{
  CView::OnSize(nType, cx, cy);

  // Set OpenGL perspective, viewport and mode
  CSize size(cx,cy);
  double aspect;
  aspect = (cy == 0) ? (double)size.cx : (double)size.cx/(double)size.cy;
  glViewport(0,0,size.cx,size.cy);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluPerspective(45,aspect,1,15.0);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  glDrawBuffer(GL_BACK);
  glEnable(GL_DEPTH_TEST);
  // TODO: Add your message handler code here
}

        void COpenGLView::OnLButtonDown(UINT nFlags, CPoint point)
{
  m_LeftButtonDown = TRUE;
  m_LeftDownPos = point;
  CView::OnLButtonDown(nFlags, point);
```

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

```
}

void COpenGLView::OnLButtonUp(UINT nFlags, CPoint point)
{
  m_LeftButtonDown = FALSE;
  CView::OnLButtonUp(nFlags, point);
}

BOOL COpenGLView::SetWindowPixelFormat(HDC hDC)
{
      PIXELFORMATDESCRIPTOR pixelDesc;

      pixelDesc.nSize = sizeof(PIXELFORMATDESCRIPTOR);
      pixelDesc.nVersion = 1;
      pixelDesc.dwFlags = PFD_DRAW_TO_WINDOW |
      PFD_SUPPORT_OPENGL | PFD_DOUBLEBUFFER |
      PFD_STEREO_DONTCARE;

      pixelDesc.iPixelType = PFD_TYPE_RGBA;
      pixelDesc.cColorBits = 32;
      pixelDesc.cRedBits = 8;
      ixelDesc.cRedShift = 16;
      pixelDesc.cGreenBits = 8;
      ixelDesc.cGreenShift = 8;
      pixelDesc.cBlueBits = 8;
      ixelDesc.cBlueShift = 0;
      pixelDesc.cAlphaBits = 0;
      ixelDesc.cAlphaShift = 0;
      pixelDesc.cAccumBits = 64;
      pixelDesc.cAccumRedBits = 16;
      ixelDesc.cAccumGreenBits = 16;
      pixelDesc.cAccumBlueBits = 16;
      ixelDesc.cAccumAlphaBits = 0;
      pixelDesc.cDepthBits = 32;
      ixelDesc.cStencilBits = 8;
      pixelDesc.cAuxBuffers = 0;
      ixelDesc.iLayerType = PFD_MAIN_PLANE;
      pixelDesc.bReserved = 0;
      ixelDesc.dwLayerMask = 0;
      pixelDesc.dwVisibleMask = 0;
      ixelDesc.dwDamageMask = 0;

      m_GLPixelIndex = ChoosePixelFormat(hDC,&pixelDesc);
   if(m_GLPixelIndex == 0) // Choose default
```

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

```
  {
     m_GLPixelIndex = 1;
     if(DescribePixelFormat(hDC,m_GLPixelIndex,
        sizeof(PIXELFORMATDESCRIPTOR),&pixelDesc)==0)
        return FALSE;
  }

        if(!SetPixelFormat(hDC,m_GLPixelIndex,&pixelDesc))
     return FALSE;
     return TRUE;
}

BOOL COpenGLView::CreateViewGLContext(HDC hDC)
{
  m_hGLContext = wglCreateContext(hDC);

  if(m_hGLContext==NULL)
     return FALSE;

  if(wglMakeCurrent(hDC,m_hGLContext)==FALSE)
     return FALSE;

  return TRUE;
}

void COpenGLView::InitGeometry()
{
  m_xRotation = 0.0f;
  m_yRotation = 0.0f;
  m_zRotation = 0.0f;

  m_xTranslation = 0.0f;
  m_yTranslation = 0.0f;
  m_zTranslation = -5.0f;

  m_xScaling = 1.0f;
  m_yScaling = 1.0f;
  m_zScaling = 1.0f;

}

        int COpenGLView::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
  if (CView::OnCreate(lpCreateStruct) == -1)
```

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

```
    return -1;

  HWND hWnd = GetSafeHwnd();
  HDC hDC = ::GetDC(hWnd);

  if(SetWindowPixelFormat(hDC)==FALSE)
     return 0;

  if(CreateViewGLContext(hDC)==FALSE)
     return 0;

  // Default mode
  glPolygonMode(GL_FRONT,GL_LINE);
  glPolygonMode(GL_BACK,GL_LINE);
          glShadeModel(GL_FLAT);
  glEnable(GL_NORMALIZE);


  // Lights, material properties
  GLfloat ambientProperties[]  = {0.7f, 0.7f, 0.7f, 1.0f};
  GLfloat diffuseProperties[]  = {0.8f, 0.8f, 0.8f, 1.0f};
  GLfloat specularProperties[] = {1.0f, 1.0f, 1.0f, 1.0f};

  glClearDepth( 1.0 );

  glLightfv( GL_LIGHT0, GL_AMBIENT, ambientProperties);
  glLightfv( GL_LIGHT0, GL_DIFFUSE, diffuseProperties);
  glLightfv(GL_LIGHT0,GL_SPECULAR,specularProperties);
  glLightModelf(GL_LIGHT_MODEL_TWO_SIDE, 1.0);

  // Default : lighting
  glEnable(GL_LIGHT0);
  glEnable(GL_LIGHTING);
  return 0;
}

void COpenGLView::OnDestroy()
{
  if(wglGetCurrentContext() != NULL)
     wglMakeCurrent(NULL,NULL);

  if(m_hGLContext != NULL)
  {
     wglDeleteContext(m_hGLContext);
```

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

```
    m_hGLContext = NULL;
  }
  CView::OnDestroy();
}
```

Bảng kê 2.3.3: **FormCommandView.h**

```
#if
!defined(AFX_FORMCOMMANDVIEW_H__B5E7793B_7225_11D5_9ABA_B
AADD0C68D13__INCLUDED_)
#define
AFX_FORMCOMMANDVIEW_H__B5E7793B_7225_11D5_9ABA_BAADD0
C68D13__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// FormCommandView.h : header file
// CFormCommandView form view

#ifndef __AFXEXT_H__
#include <afxext.h>
#endif
class COpenGLDoc;
class CFormCommandView : public CFormView
{
protected:
  CFormCommandView();
  DECLARE_DYNCREATE(CFormCommandView)
// Form Data
public:
  //{{AFX_DATA(CFormCommandView)
  enum { IDD = IDD_FORM_COMAMAND };
  CStatic  m_ControlColorLightAmbient;
  CSliderCtrl    m_SliderScaleZ;
  CSliderCtrl    m_SliderScaleY;
  CSliderCtrl    m_SliderScaleX;
  CStatic  m_ControlBackColor;
  BOOL   m_Antialias;
  BOOL   m_Lighting;
  BOOL   m_LinkScale;
  BOOL   m_Smooth;
  BOOL   m_VRotate;
  //}}AFX_DATA
```

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

```
// Attributes
public:
// Operations
public:
  BOOL UpdateScale();
  COpenGLDoc* GetDocument();
  CView *GetRenderView();
// Overrides
  // ClassWizard generated virtual function overrides
  //{{AFX_VIRTUAL(CFormCommandView)
  public:
  virtual void OnInitialUpdate();
  protected:
  virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
  //}}AFX_VIRTUAL

// Implementation
protected:
  virtual ~CFormCommandView();
#ifdef _DEBUG
  virtual void AssertValid() const;
  virtual void Dump(CDumpContext& dc) const;
#endif

  // Generated message map functions
  //{{AFX_MSG(CFormCommandView)
  afx_msg void OnPaint();
  afx_msg void OnHScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar);
  afx_msg void OnLButtonUp(UINT nFlags, CPoint point);
  afx_msg void OnCheckAntialias();
  afx_msg void OnCheckLighting();
  afx_msg void OnCheckLinkScale();
  afx_msg void OnCheckSmooth();
  afx_msg void OnCheckVrotation();
  afx_msg void OnRadioModel0();
  afx_msg void OnRadioModel1();
  afx_msg void OnRadioModel2();
  //}}AFX_MSG
  DECLARE_MESSAGE_MAP()
};

#ifndef _DEBUG  // debug version in RenderView.cpp
inline COpenGLDoc* CFormCommandView::GetDocument()
```

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

```
  { return (COpenGLDoc*)m_pDocument; }

#endif
/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the
previous line.
#endif //
!defined(AFX_FORMCOMMANDVIEW_H__B5E7793B_7225_11D5_9ABA_B
AADD0C68D13__INCLUDED_)
```

Bảng kê 2.3.4: **FormCommandView.cpp**

```
// FormCommandView.cpp : implementation file
//
#include "stdafx.h"
#include "OpenGL.h"
#include "FormCommandView.h"
#include "ChildFrm.h"
#include "MainFrm.h"
#include "OpenGLView.h"
#include "OpenGLDoc.h"
//#include "
#include "gl\glu.h"
#include "gl\gl.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
/////////////////////////////////////////////////////////////////////////////
// CFormCommandView

IMPLEMENT_DYNCREATE(CFormCommandView, CFormView)

CFormCommandView::CFormCommandView()
  : CFormView(CFormCommandView::IDD)
{
  //{{AFX_DATA_INIT(CFormCommandView)
  m_Antialias = FALSE;
  m_Lighting = FALSE;
  m_LinkScale = FALSE;
  m_Smooth = FALSE;
```

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

```
  m_VRotate = FALSE;
  //}}AFX_DATA_INIT
}

CFormCommandView::~CFormCommandView()
{
}

void CFormCommandView::DoDataExchange(CDataExchange* pDX)
{
  CFormView::DoDataExchange(pDX);
  //{{AFX_DATA_MAP(CFormCommandView)
  DDX_Control(pDX, IDC_FRAME_COLOR_LIGHT_AMBIENT,
m_ControlColorLightAmbient);
  DDX_Control(pDX, IDC_SLIDER_Z, m_SliderScaleZ);
  DDX_Control(pDX, IDC_SLIDER_Y, m_SliderScaleY);
  DDX_Control(pDX, IDC_SLIDER_X, m_SliderScaleX);
  DDX_Control(pDX, IDC_FRAME_COLOR_BACK, m_ControlBackColor);
  DDX_Check(pDX, IDC_CHECK_ANTIALIAS, m_Antialias);
  DDX_Check(pDX, IDC_CHECK_LIGHTING, m_Lighting);
  DDX_Check(pDX, IDC_CHECK_LINK_SCALE, m_LinkScale);
  DDX_Check(pDX, IDC_CHECK_SMOOTH, m_Smooth);
  DDX_Check(pDX, IDC_CHECK_VROTATION, m_VRotate);
  //}}AFX_DATA_MAP
}
BEGIN_MESSAGE_MAP(CFormCommandView, CFormView)
  //{{AFX_MSG_MAP(CFormCommandView)
  ON_WM_PAINT()
  ON_WM_HSCROLL()
  ON_WM_LBUTTONUP()
  ON_BN_CLICKED(IDC_CHECK_ANTIALIAS, OnCheckAntialias)
  ON_BN_CLICKED(IDC_CHECK_LIGHTING, OnCheckLighting)
  ON_BN_CLICKED(IDC_CHECK_LINK_SCALE, OnCheckLinkScale)
  ON_BN_CLICKED(IDC_CHECK_SMOOTH, OnCheckSmooth)
  ON_BN_CLICKED(IDC_CHECK_VROTATION, OnCheckVrotation)
  ON_BN_CLICKED(IDC_RADIO_MODEL_0, OnRadioModel0)
  ON_BN_CLICKED(IDC_RADIO_MODEL_1, OnRadioModel1)
  ON_BN_CLICKED(IDC_RADIO_MODEL_2, OnRadioModel2)
  //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CFormCommandView diagnostics
```

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

```
#ifdef _DEBUG
void CFormCommandView::AssertValid() const
{
  CFormView::AssertValid();
}

void CFormCommandView::Dump(CDumpContext& dc) const
{
  CFormView::Dump(dc);
}

#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CFormCommandView message handlers

void CFormCommandView::OnInitialUpdate()
{
  CFormView::OnInitialUpdate();

  // Slider
  TRACE("Sliders : updating...\n");
  m_SliderScaleX.SetRange(1,100,TRUE);
  m_SliderScaleY.SetRange(1,100,TRUE);
  m_SliderScaleZ.SetRange(1,100,TRUE);

  m_SliderScaleX.SetPos(50);
  m_SliderScaleY.SetPos(50);
  m_SliderScaleZ.SetPos(50);
}

void CFormCommandView::OnPaint()
{

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

  // Options are stored in Application
  COpenGLApp *pApp = (COpenGLApp *)AfxGetApp();
  CRect rect;

  // Color back
  m_ControlBackColor.GetWindowRect(&rect);
  ScreenToClient(&rect);
```

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

```
  CBrush BrushBack(pApp->m_OptionColorGlBack);
  dc.FillRect(&rect,&BrushBack);

  // Color light ambient
  m_ControlColorLightAmbient.GetWindowRect(&rect);
  ScreenToClient(&rect);
  CBrush BrushLightAmbient(pApp->m_OptionColorGlLightAmbient);
  dc.FillRect(&rect,&BrushLightAmbient);     // TODO: Add your message handler
code here
  // Do not call CFormView::OnPaint() for painting messages
}

void CFormCommandView::OnHScroll(UINT nSBCode, UINT nPos,
CScrollBar* pScrollBar)
{
  // TODO: Add your message handler code here and/or call default
  switch( nSBCode )
  {
  case TB_BOTTOM:
     UpdateScale();
     break;
  case TB_ENDTRACK:
     UpdateScale();
     break;
  case TB_LINEDOWN:
     UpdateScale();
     break;
  case TB_LINEUP:
     UpdateScale();
     break;
  case TB_PAGEDOWN:
     UpdateScale();
     break;
  case TB_PAGEUP:
     UpdateScale();
     break;
  case TB_THUMBPOSITION:
     UpdateScale();
     break;
  case TB_THUMBTRACK:
     UpdateScale();
     break;
  case TB_TOP:
     UpdateScale();
```

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

```
        break;
    default:
        {}
    }
    // Update scaling
    GetRenderView()->InvalidateRect(NULL,FALSE);

    CFormView::OnHScroll(nSBCode, nPos, pScrollBar);
}

void CFormCommandView::OnLButtonUp(UINT nFlags, CPoint point)
{
    // TODO: Add your message handler code here and/or call default
    CRect rect;
    COpenGLApp *pApp = (COpenGLApp *)AfxGetApp();

    // Option back color
    m_ControlBackColor.GetWindowRect(&rect);
    ScreenToClient(&rect);
    if(rect.PtInRect(point))
    {
        CColorDialog dlg(pApp->m_OptionColorGlBack);
        if(dlg.DoModal()==IDOK)
        {
            pApp->m_OptionColorGlBack = dlg.GetColor();
            COpenGLView *pView = (COpenGLView *)GetRenderView();
            pView->m_ClearColorRed   = (float)GetRValue(pApp-
>m_OptionColorGlBack) / 255.0f;
            pView->m_ClearColorGreen = (float)GetGValue(pApp-
>m_OptionColorGlBack) / 255.0f;
            pView->m_ClearColorBlue  = (float)GetBValue(pApp-
>m_OptionColorGlBack) / 255.0f;
            this->InvalidateRect(&rect,FALSE);
            pView->InvalidateRect(NULL,FALSE);
        }
    }

    // Option ambient light color
    m_ControlColorLightAmbient.GetWindowRect(&rect);
    ScreenToClient(&rect);
    if(rect.PtInRect(point))
    {
        CColorDialog dlg(pApp->m_OptionColorGlLightAmbient);
        if(dlg.DoModal()==IDOK)
```

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

```
    {
        pApp->m_OptionColorGlLightAmbient = dlg.GetColor();
        COpenGLView *pView = (COpenGLView *)GetRenderView();

        // Refresh Light0
        float r = (float)GetRValue(pApp->m_OptionColorGlLightAmbient) /
255.0f;
        float g = (float)GetGValue(pApp->m_OptionColorGlLightAmbient) /
255.0f;
        float b = (float)GetBValue(pApp->m_OptionColorGlLightAmbient) /
255.0f;
        float  ambientProperties[]  = {r,g,b,1.0f};
        glLightfv( GL_LIGHT0, GL_AMBIENT, ambientProperties);

        // Refresh views
        this->InvalidateRect(&rect,FALSE);
        pView->InvalidateRect(NULL,FALSE);

    }
  }

  CFormView::OnLButtonUp(nFlags, point);
}

CView *CFormCommandView::GetRenderView()
{
  COpenGLApp *pApp = (COpenGLApp *)AfxGetApp();
  CMainFrame *pMainFrame = (CMainFrame *)pApp->m_pMainWnd;
  CChildFrame *pFrame = (CChildFrame *)pMainFrame->GetActiveFrame();
  CView *pView = (CView *)pFrame->m_wndSplitter.GetPane(0,1);
  return pView;

}

COpenGLDoc* CFormCommandView::GetDocument()
{
  ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(COpenGLDoc)));
  return (COpenGLDoc*)m_pDocument;
}

BOOL CFormCommandView::UpdateScale()
{
  COpenGLView *pView = (COpenGLView *)GetRenderView();
  pView->m_xScaling = (float)m_SliderScaleX.GetPos()/50.0f;
```

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

```
  pView->m_yScaling = (float)m_SliderScaleY.GetPos()/50.0f; // TODO : link
  pView->m_zScaling = (float)m_SliderScaleZ.GetPos()/50.0f;

  if(m_LinkScale)
  {
     m_SliderScaleY.SetPos(m_SliderScaleX.GetPos());
     m_SliderScaleZ.SetPos(m_SliderScaleX.GetPos());
     pView->m_yScaling = pView->m_zScaling = pView->m_xScaling;
  }

  return TRUE;

}

void CFormCommandView::OnCheckAntialias()
{
  // TODO: Add your control notification handler code here
  m_Antialias = !m_Antialias;

  if(m_Antialias)
  {
     glEnable(GL_LINE_SMOOTH);
     glEnable(GL_BLEND);
     glBlendFunc(GL_SRC_ALPHA,GL_ONE_MINUS_SRC_ALPHA);
     glHint(GL_LINE_SMOOTH_HINT,GL_NICEST);
     glLineWidth(1.5);
  }
  else
  {
     glDisable(GL_LINE_SMOOTH);
     glDisable(GL_BLEND);
     glBlendFunc(GL_SRC_ALPHA,GL_ONE_MINUS_SRC_ALPHA);
     glHint(GL_LINE_SMOOTH_HINT,GL_NICEST);
     glLineWidth(1.0);
  }
  GetRenderView()->InvalidateRect(NULL,FALSE);

}

void CFormCommandView::OnCheckLighting()
{
  // TODO: Add your control notification handler code here
  m_Lighting = !m_Lighting;
  if(m_Lighting)
```

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

```
    glEnable(GL_LIGHTING);
  else
    glDisable(GL_LIGHTING);
  this->GetRenderView()->InvalidateRect(NULL,FALSE);

}

void CFormCommandView::OnCheckLinkScale()
{
  m_LinkScale = !m_LinkScale;

  if(m_LinkScale)
  {
    COpenGLView *pView = (COpenGLView *)GetRenderView();
    m_SliderScaleY.SetPos(m_SliderScaleX.GetPos());
    m_SliderScaleZ.SetPos(m_SliderScaleX.GetPos());
    pView->m_yScaling = pView->m_zScaling = pView->m_xScaling;
  }

  m_SliderScaleY.EnableWindow(!m_LinkScale);
  m_SliderScaleZ.EnableWindow(!m_LinkScale);
  GetRenderView()->InvalidateRect(NULL,FALSE);
}

void CFormCommandView::OnCheckSmooth()
{
  m_Smooth = !m_Smooth;
  if(m_Smooth)
    glShadeModel(GL_SMOOTH);
  else
    glShadeModel(GL_FLAT);
  this->GetRenderView()->InvalidateRect(NULL,FALSE);

}

void CFormCommandView::OnCheckVrotation()
{
  // TODO: Add your control notification handler code here
    m_VRotate = !m_VRotate;
  COpenGLView *pView = (COpenGLView *)GetRenderView();
  if(m_VRotate)
    pView->SetTimer(1,10,NULL);
  else
    pView->KillTimer(1);
```

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!

http://kilobooks.com
THÖ VIEN ÑIEN TÖ TRÖC TUYEN

```
}

void CFormCommandView::OnRadioModel0()
{
  // TODO: Add your control notification handler code here
  glPolygonMode(GL_FRONT_AND_BACK,GL_POINT);
  this->GetRenderView()->InvalidateRect(NULL,FALSE);

}

void CFormCommandView::OnRadioModel1()
{
  glPolygonMode(GL_FRONT_AND_BACK,GL_LINE);
  this->GetRenderView()->InvalidateRect(NULL,FALSE);
}
void CFormCommandView::OnRadioModel2()
{
  // TODO: Add your control notification handler code here
  glPolygonMode(GL_FRONT_AND_BACK,GL_FILL);
  this->GetRenderView()->InvalidateRect(NULL,FALSE);

}
```

KILOBOOKS.COM

Nếu bạn cần file word? Xin hãy nhấp vào đây!