Chương 4
Một số công cụ hữu dụng của OpenGL

1. Các mảng đỉnh (Vertex Arrays)

Ta đã học trong chương 2 cho phép vẽ các đối tượng hình học cơ bản như điểm, đường và đa giác thông qua việc gọi các thủ tục có sẵn. Nếu như các đối tượng vẽ ít thì sẽ không có vấn đề gì nhưng giả sử ta cần vẽ một đa giác có 20 đỉnh khi đó ta cần tới lời gọi 22 thủ tục: Một lần gọi **glBegi**n(), 20 lần gọi thủ tục **glVertex***() cho 20 đỉnh và một lần gọi thủ tục **glEnd**(). Rõ ràng là chương trình trở nên cồng kềnh.

Một vấn đề nữa là vấn đề xử lí dư thừa toạ độ các đỉnh chung cúa các đa giác liền kề. Xét một ví dụ, giả sử ta phải vẽ một hình hộp chữ nhật khi biết toạ độ các đỉnh. Khi đó ta phải vẽ từng mặt của hình hộp thông qua các đỉnh của nó, vậy mỗi đỉnh của hình hộp được gọi ba lần và tổng số thủ tục **glVertex***() là 24 lần gọi trong khi đó ta chỉ cần 8 lần gọi là đủ.

Để khắc phục nhược điểm đó OpenGL cho phép sử dụng vertex array lưu trữ các giá trị (như toạ độ các điểm, toạ độ các véc tơ pháp, các thông số trong chế độ màu RGB,....) và một số các thủ tục truy cấp số liệu từ vertex array.

Việc sử dụng vertex array cho phép giảm bớt việc gọi các thủ tục cũng như giảm bớt sự dư thừa khi xử lí các điểm chung do đó cải thiện tốc độ thực hiện của chương trình ứng dụng.

Có ba bước chính để sử dụng vertex array biểu diễn các đối tượng hình ảnh:

- Kích hoạt các mảng, mảng lưu trữ các giá trị thuộc một trong số tám kiểu dữ liệu sau: Toạ độ các đỉnh (vertex coordinate), các véc tơ pháp của mặt (surface normal), màu RGBA (RGBA colors), màu thứ sinh (secondary colors), chỉ mục màu (colors indices), toạ độ tạo sương mù (fog coordinate), toạ độ texture (texture coordinate), cờ hiệu các cạnh của đa giác (polygon edge flags).

- Chỉ định dữ liệu của mảng

- Vẽ các đối tượng hình học với dữ liệu đã có trong mảng: Ta có thể truy cập đến từng phần tử, một phần cũng như lần lượt tất cả các phần tử của mảng.

Ta sẽ xét các thủ tục cho phép thực hiện từng bước kể trên

Bước 1: Kích hoạt mảng

glEnableClientState(GLenum array)

Tham số array xác định kiểu mảng cần kích hoạt có thể nhận một trong số các hằng sau: GL_VERTEX_ARRAY, GL_COLOR_ARRAY, GL_SECONDARY_COLOR_ARRAY, GL_INDEX_ARRAY, GL_NORMAL_ARRAY, GL_FOG_COORDINATE_ARRAY, GL_TEXTURE_COORD_ARRAY, và GL_EDGE_FLAG_ARRAY

Ví dụ ta muốn kích hoạt mảng các toạ độ và mảng các véc tơ pháp của một mặt cong ta dùng hai lệnh sau:

glEnableClientState(GL_VERTEX_ARRAY);
glEnableClientState(GL_NORMAL_ARRAY);

Bước 2: Chỉ định dữ liệu của mảng

Có tám thủ tục khác nhau dùng để chỉ định dữ liệu của mảng tuỳ thuộc vào kiểu dữ liệu.

glVertexPointer(*GLint size, GLenum type, GLsizei stride,const GLvoid *pointer*);

Chỉ định dữ liệu toạ độ các đỉnh trong đó tham số *pointer* là địa chỉ của điểm đầu tiên trong mảng, tham số *type* xác định kiểu dữ liệu (GL_SHORT, GL_INT, GL_FLOAT, hoặc GL_DOUBLE), *size* là kích thước toạ độ mỗi đỉnh (2,3 hoặc 4), tham số *stride* xác định cách truy cập dữ liệu trong mảng (nếu mảng chỉ chứa một loại dữ liệu thì *stride* bằng 0)

Các thủ tục dùng cho các kiểu dữ liệu khác là:

glColorPointer(*GLint size, GLenum type, GLsizei stride, const GLvoid *pointer);*

glSecondaryColorPointer(*GLint size, GLenum type, GLsizei stride, const GLvoid *pointer);*

glIndexPointer(*GLenum type, GLsizei stride, const GLvoid *pointer);*

glNormalPointer*(GLenum type, GLsizei stride, const GLvoid *pointer);*
glFogCoordPointer*(GLenum type, GLsizei stride, const GLvoid *pointer);*
glTexCoordPointer*(GLint size, GLenum type, GLsizei stride, const GLvoid *pointer);*
glEdgeFlagPointer*(GLsizei stride, const GLvoid *pointer);*

Tuỳ theo kiểu dữ liệu trong mảng mà các tham số có thể có hoặc không có, ví dụ đối với mảng các véc tơ pháp vì toạ độ véc tơ pháp của một mặt luôn có kích thước bằng 3 do đó trong thủ tục **glNormalPointer**() ta không cần dùng tham số *size*.

Bảng sau cung cấp giá trị của tham số *side* và *type* cho các thủ tục trên

Thủ tục	Side	Type
glVertexPointer	2, 3, 4	GL_SHORT, GL_INT, GL_FLOAT, GL_DOUBLE
glColorPointer	3, 4	GL_BYTE, GL_UNSIGNED_BYTE, GL_SHORT, GL_UNSIGNED_SHORT, GL_INT, GL_UNSIGNED_INT, GL_FLOAT, GL_DOUBLE
glSecondaryColorPointer	3, 4	GL_BYTE, GL_UNSIGNED_BYTE, GL_SHORT, GL_UNSIGNED_SHORT, GL_INT, GL_UNSIGNED_INT, GL_FLOAT, GL_DOUBLE
glIndexPointer	1	GL_UNSIGNED_BYTE, GL_SHORT, GL_INT, GL_FLOAT, GL_DOUBLE
glNormalPointer	3	GL_BYTE, GL_SHORT, GL_INT, GL_FLOAT, GL_DOUBLE
glFogCoordPointer	1	GL_FLOAT, GL_DOUBLE
glTexCoordPointer	1, 2, 3, 4	GL_SHORT, GL_INT, GL_FLOAT, GL_DOUBLE
glEdgeFlagPointer	1	Không có kiểu, dữ liệu ngầm định là GLboolean

Ví dụ: Xét một đoạn mã trong chương trình varray.c

```
static GLint vertices[] = {25, 25,
100, 325,
175, 25,
175, 325,
250, 25,
325, 325};
static GLfloat colors[] = {1.0, 0.2, 0.2,
0.2, 0.2, 1.0,
0.8, 1.0, 0.2,
0.75, 0.75, 0.75,
0.35, 0.35, 0.35,
0.5, 0.5, 0.5};
glEnableClientState (GL_COLOR_ARRAY);
glEnableClientState (GL_VERTEX_ARRAY);
glColorPointer (3, GL_FLOAT, 0, colors);
glVertexPointer (2, GL_INT, 0, vertices);
```

Ta đề cập thêm về cách dùng tham số *stride* trong trường hợp dùng một mảng lưu trữ hai loại dữ liệu khác nhau. Ví dụ ta lưu trữ màu RGB của từng điểm và toạ độ của điểm tương ứng trong cùng một mảng sau:

```
static GLfloat intertwined[] =
                {1.0, 0.2, 1.0, 100.0, 100.0, 0.0,
                1.0, 0.2, 0.2, 0.0, 200.0, 0.0,
                1.0, 1.0, 0.2, 100.0, 300.0, 0.0,
                0.2, 1.0, 0.2, 200.0, 300.0, 0.0,
                0.2, 1.0, 1.0, 300.0, 200.0, 0.0,
                0.2, 0.2, 1.0, 200.0, 100.0, 0.0};
```

Giá trị màu của điểm (100.0, 100.0, 0.0) là (1.0, 0.2, 1.0)..., ta thấy chỉ số màu cũng như toạ độ tương ứng của hai điểm liên tiếp trong mảng cách nhau 6*sizeof(GLfloat)và khoảng cách này chính là giá trị của tham số *stride* trong hai lệnh sau:

glColorPointer (3, GL_FLOAT,6*sizeof(GLfloat), intertwined);

glVertexPointer(3, GL_FLOAT,6*sizeof(GLfloat), &intertwined[3]);

Bước 3: Vẽ các đối tượng hình học

Có ba cách để truy cập phần tử của mảng đó là: Truy cập từng phần tử riêng lẻ, truy cập một danh sách các phần tử thông qua mảng chỉ số và truy cập một dãy con các phần tử.

- Truy cập từng phần tử riêng lẻ

glArrayElement*(GLint ith);*

Truy cập đến phần tử thứ ith trong tất cả các mảng đã được kích hoạt.Ví dụ về việc sử dụng lệnh **glArrayElement**() để định nghĩa màu và toạ độ các đỉnh trong chương trình varray.c

```
glEnableClientState (GL_COLOR_ARRAY);
glEnableClientState (GL_VERTEX_ARRAY);
glColorPointer (3, GL_FLOAT, 0, colors);
glVertexPointer (2, GL_INT, 0, vertices);

glBegin(GL_TRIANGLES);
glArrayElement (2);
glArrayElement (3);
glArrayElement (5);
glEnd();
```

Năm câu lệnh cuối trong đoạn mã trên tương đương với:

```
glBegin(GL_TRIANGLES);
glColor3fv(colors+(2*3*sizeof(GLfloat));
glVertex3fv(vertices+(2*2*sizeof(GLint));
glColor3fv(colors+(3*3*sizeof(GLfloat));
glVertex3fv(vertices+(3*2*sizeof(GLint));
glColor3fv(colors+(5*3*sizeof(GLfloat));
glVertex3fv(vertices+(5*2*sizeof(GLint));
glEnd();
```

- Truy cập một danh sách các phần tử thông qua mảng chỉ số

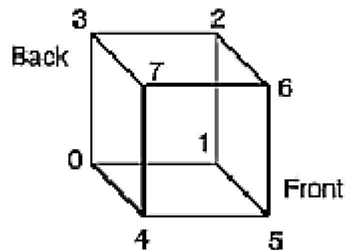
glDrawElements*(GLenum mode, GLsizei count, GLenum type, void *indices);*

Truy cấp đến *count* phần tử của mảng, chỉ số các phần tử được lưu trữ trong mảng indices. Tham số *mode* giống như tham số của thủ tục **glBegin**(), tham số type chỉ định kiểu dữ liệu của mảng *indices* và nhận một trong số các giá trị GL_UNSIGNED_BYTE, GL_UNSIGNED_SHORT, hoặc GL_UNSIGNED_INT.

Hiệu lực của lệnh **glDrawElements**() tương đương với đoạn mã sau:

```
int i;
glBegin (mode);
for (i = 0; i < count; i++)
glArrayElement(indices[i]);
glEnd();
```

Xét ví dụ giả sử ta muốn vẽ một hình lập phương mà các đỉnh được đánh dấu như hình vẽ dưới đây:



Khi đó đoạn mã sau cho phép vẽ hình hộp sau khi ta đã kích hoạt mảng chứa toạ độ 8 đỉnh:

```
static GLubyte frontIndices = {4, 5, 6, 7};
static GLubyte rightIndices = {1, 2, 6, 5};
static GLubyte bottomIndices = {0, 1, 5, 4};
static GLubyte backIndices = {0, 3, 2, 1};
static GLubyte leftIndices = {0, 4, 7, 3};
static GLubyte topIndices = {2, 3, 7, 6};
glDrawElements(GL_QUADS, 4, GL_UNSIGNED_BYTE, frontIndices);
glDrawElements(GL_QUADS, 4, GL_UNSIGNED_BYTE, rightIndices);
glDrawElements(GL_QUADS, 4, GL_UNSIGNED_BYTE, bottomIndices);
glDrawElements(GL_QUADS, 4, GL_UNSIGNED_BYTE, backIndices);
glDrawElements(GL_QUADS, 4, GL_UNSIGNED_BYTE, leftIndices);
glDrawElements(GL_QUADS, 4, GL_UNSIGNED_BYTE, topIndices);
```

Hoặc một cách tốt hơn khi ta gộp chỉ số các đỉnh vào một mảng duy nhất:

```
static GLubyte allIndices = {4, 5, 6, 7, 1, 2, 6, 5,
                             0, 1, 5, 4, 0, 3, 2, 1,
                             0, 4, 7, 3, 2, 3, 7, 6};
glDrawElements(GL_QUADS, 24, GL_UNSIGNED_BYTE, allIndices);
```

- Truy cập một dãy con các phần tử

glDrawArrays*(GLenum mode, GLint first, GLsizei count);*

Truy cập *count* phần tử của mảng kể từ phần tử thứ *first*, tham số *mode* giống như tham số của **glBegin**()

Hiệu lực của lệnh **glDrawArrays**() tương đương với đoạn mã sau:

```
int i;
glBegin (mode);
for (i = 0; i < count; i++)
glArrayElement(first + i);
glEnd();
```

2. Các danh sách hiển thị (Display Lists)

Một danh sách hiển thị là một nhóm các câu lệnh của OpenGL có thể được gọi nhiều lần để thực hiện một nhiệm vụ nào đó. Khái niệm danh sách hiển thị được hiểu giống như khái niệm hàm và thủ tục trong các ngôn ngữ lập trình. Ví dụ ta muốn vẽ một cái ôtô có bốn bánh, thay vì việc phải viết đoạn mã vẽ bánh ôtô bốn lần ta sẽ tạo một danh sách hiển thị làm nhiệm vụ vẽ bánh ôtô và gọi nó bốn lần.

Danh sách hiển thị là đặc biệt hiệu quả trong mô hình client-server khi ta muốn giảm lưu lượng thông tin truyền tải qua mạng, khi client thực hiện danh sách hiển thị nó sẽ lưu trữ danh sách hiển thị đó cho các lần thực hiện sau.

2.1 Tạo một danh sách hiển thị

Tạo một danh sách hiển thị theo cấu trúc như sau:

listName= **glGenLists**(*range);*
glNewList (*GLuint listName*, *GLenum mode*);
<Các lệnh của OpenGL>
glEndList (void);

Trong chương trình có thể ta phải sử dụng đồng thời nhiều danh sách hiển thị, khi đó OpenGL gán cho mỗi danh sách hiển thị một chỉ số duy nhất. Ta dùng hàm **glGenLists**() để tự động tạo ra một chỉ số chưa được sử dụng và gán cho danh sách hiển thị, cú pháp lệnh như sau:

Hàm **glGenLists**(*range*); trả về một giá trị duy nhất trong khoảng từ 1 đến *range* chưa được sử dụng trước đó là gán cho danh sách hiển thị. Hàm sẽ trả về giá trị 0 nếu như không còn giá trị nào trống trong miền được chỉ định hoặc khi *range*=0.

glNewList (*GLuint listName, GLenum mode*);

Chỉ định điểm bắt đầu của một danh sách hiển thị, tham số *mode* có thể nhận một trong số các giá trị GL_COMPILE hoặc GL_COMPILE_AND_EXECUTE. Sử dụng GL_COMPILE nếu muốn các lệnh trong danh sách hiển thị chỉ được thực hiện khi ta gọi nó và sử dụng GL_COMPILE_AND_EXECUTE nếu muốn các lệnh trong danh sách hiển thị thực hiện ngay tức khắc cũng như cho những lần gọi sau.

glEndList (void);

Đánh dấu điểm kết thúc của một danh sách hiển thị

2.2 Thi hành lời gọi một danh sách hiển thị

Để thi hành lời gọi một danh sachs hiển thị đã được định nghĩa trước đó ta dùng thủ tục sau:

glCallList (*listName*);

Thi hành danh sách hiển thị được đánh dấu bởi *listName*, nếu *listName* chưa được gán giá trị thì lệnh thi hành danh sách hiển thị không có hiệu lực.

Ví dụ: Thực hiện chương trình torus.c

Chương trình là vẽ một gờ tròn và nhìn nó từ nhiều góc độ khác nhau. Cách tốt nhất là dùng danh sách hiển thị lưu trữ lệnh vẽ gờ tròn, mỗi lần muốn thay đổi điểm nhìn ta thay đổi modelview matrix và gọi danh sách hiển thị vẽ gờ tròn.

Ví dụ: Thực hiện chương trình list.c

Chương trình vẽ 10 tam giác màu đỏ và một đoạn thẳng, hãy giải thích vị trí xuất hiện của đoạn thẳng.

2.3 Thi hành nhiều danh sách hiển thị

OpenGl cho phép thi hành nhiều danh sách hiển thị bằng cách đưa chỉ số của các danh sách hiển thị vào một mảng trước khi thực hiện lời gọi **glCallList** (). Ta khởi tạo giá trị chỉ mục bằng cách sử dụng hàm **glListBase** ()

glListBase(*GLuint base*);

Chí định rằng *base* là giá trị được cộng thêm vào (*offset*) các phần tử của mảng *list* trong lệnh **glCallList**() dưới đây, kết quả được dùng là chỉ số của các danh sách hiển thị được gọi trong lệnh **glCallList**(). Giá trị mặc định của *base* là 0.

glCallLists(*GLsizei n, GLenum type, const GLvoid *lists*);

Thực hiện n danh sách hiển thị, chỉ số của các danh sách được tính bằng cách cộng thêm giá trị *base* (được xác định bởi lệnh **glListBase**()) với giá trị số nguyên không âm trong mảng chỉ số được chỉ bởi con trỏ *list*. Tham số type xác định kiểu dữ liệu được lưu trữ trong *list*

Ví dụ: Thực hiện chương trình multilist.c

Chương trình minh hoạ cách tạo và thi hành lời gọi nhiều danh sách hiển thị

2.4 Các danh sách hiển thị phân cấp

Ta có thể tạo các danh sách hiển thị phân cấp , nghĩa là một danh sách hiển thị có thể gọi các danh sách hiển thị khác bằng thủ tục **glCallList**() giữa hai thủ tục **glNewList**() và **glEndList**().

Danh sách hiển thị phân cấp là đặc biệt hữu dụng khi đối tượng vẽ bao gồm nhiều thành phần và có một số thành phần được sử dụng nhiều hơn một lần.

Ví dụ đoạn mã sau minh hoạ vẽ một chiếc xe đạp bằng cách gọi các danh sách hiển thị vẽ các bộ phận của nó.

```
glNewList(listIndex,GL_COMPILE);
glCallList(handlebars);
glCallList(frame);
glTranslatef(1.0,0.0,0.0);
glCallList(wheel);
glTranslatef(3.0,0.0,0.0);
glCallList(wheel);
glEndList();
```

3. Hiển thị các kí tự (Character/ Text)

Đồ hoạ máy tính sử dụng hai cách để biểu diễn kí tự (ta còn gọi là font chữ). Cách thứ nhất là định nghĩa kí tự dưới dạng bitmap font (raster font), theo cách này mỗi kí tự được biểu diễn dưới dạng một miền hình chữ nhật các ô mà mỗi ô là một bit 0 hoặc 1. Cách thứ hai là định nghĩa kí tự dưới dạng stroke font (outline font), mỗi kí tự được tạo thành từ các đoạn thẳng ghép lại.

Trước khi cho hiển thị font ta phải xác định vị trí sẽ hiển thị font trên cổng nhìn bằng cách áp dụng lệnh sau:

glRasterPos*{234}{sifd}(TYPE x, TYPE y, TYPE z, TYPE w);*

Thư viện GLUT cung cấp cả hai kiểu kí tự trên bằng cách gọi các thủ tục sau:

Thủ tục vẽ bitmap font:

glutBitmapCharacter (*font, character*);

Trong đó tham số font xác định kiểu và kích thước của font chữ nhận một trong số các giá trị sau GLUT_BITMAP_8_BY_13, GLUT_BITMAP_9_BY_15, GLUT_BITMAP_TIMES_ROMAN_10, GLUT_BITMAP_HELVETICA_10. Tham số character có thể là mã ASCII của kí tự hoặc kí tự tương ứng.

Thủ tục vẽ stroke font:

glutStrokeCharacter (*font, character*);

Trong đó tham số font xác định kiểu và kích thước của font chữ nhận một trong số các giá trị sau GLUT_STROKE_ROMAN, GLUT_STROKE_MONO_ROMAN. Tham số character có thể là mã ASCII của kí tự hoặc kí tự tương ứng.

Ví dụ: Hai lệnh sau sẽ cho hiển thị kí tự bitmap 'A' trên màn hình tại vị trí (50,50) trong cổng nhìn

glRasterPos2i(50,50);

glutBitmapCharacter(GLUT_BITMAP_8_BY_13,65);

Chú ý:

- Để định vị được vị trí hiển thị font trên màn hình ta phải thiết lập modelview matrix và projection matrix về không gian hai chiều bằng các câu lệnh sau:

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0.0, (GLfloat) width, 0.0, (GLfloat) height);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
```

Trong đó width và height là hai kích thước của cửa sổ hiển thị

- OpenGL cung cấp lệnh **glBitmap**() cho phép hiển thị các kí tự bitmap (đọc thêm chương 8- Red Book)

Ví dụ: Thực hiện chương trình stroke.c

4. Cách tạo các trình đơn (menus)

GLUT cung cấp các câu lệnh cho phép tạo trình đơn bật lên (pop-up menu). Một pop-up menu được tạo bởi câu lệnh sau:

glutCreateMenu(*menu_function*);

Tham số **menu_function** là tên của một thủ tục được gọi khi menu được chọn, *menu_function* còn được gọi là hàm gọi lại (callback function) được đăng kí bởi glutCreateMenu(). Thủ tục *menu_function* có một tham số là một số nguyên tương ứng với vị trí được lựa chọn trong menu. Thủ tục *menu_function* có dạng:

void **menu_function**(*GLint menuID*);

Giá trị nguyên được truyền cho *menuID* dược sử dụng bởi *menu_function* để thực hiện mọt số công việc nào đó.

glutAddMenuEntry (*entry, menuID*);

Lệnh này tạo thêm một mục *entry* trong menu, nếu nó được chọn thì *menuID* sẽ được trả lại cho hàm gọi lại *menu_function*().

Cuối cùng ta phải chỉ định thao tác chuột sẽ dùng để lựa chọn trình đơn bằng lệnh:

glutAttachMenu (*button*);

Trong đó tham số *button* nhận một trong số các giá trị GLUT_LEFT_BOTTON, GLUT_MIDDLE_BOTTON, GLUT_RIGHT_BUTTON.

Ví dụ: Thực hiện chương trình menu.c minh hoạ cách xây dựng menu

Ví dụ: Thực hiện chương trình newpaint2.c

5. Tạo nhiều cửa sổ hiển thị (Multiple Windows)

Ví dụ: Thực hiện chương trình windows.c

Ta có thể tạo nhiều cửa sổ để hiển thị các hình ảnh độc lập với nhau.

WindowID=**glutCreateWindow**("Window_name");

Lệnh trên tạo ra một cửa sổ có tên là Window_name và trả về giá trị là một số nguyên WindowID. Giá trị WindowID sẽ được truyền cho thủ tục **glutSetWindow**(WindowID) trong thủ tục hiển thị hình ảnh.

6. Các mặt phẳng cắt (Clip Planes)

Ví dụ: Thực hiện chương trình clip.c

Ta sử dụng mặt phẳng cắt để cắt một phần của hình ảnh không cho phép hiển thị lên màn hình.

glClipPlane(GL_CLIP_PLANEi, **equation*);

Trong đó *equation* trỏ đến mảng {A, B, C, D} là các hệ số xác định phương trình của mặt phẳng cắt Ax+By+Cz+D=0. Lệnh trên sẽ cắt một nửa phần /không gian gồm các điểm (x,y,z) thoả mãn Ax+By+Cz+D<0.

Mặt phẳng cắt phải được kích hoạt bởi lệnh **glEnable**(GL_CLIP_PLANEi);