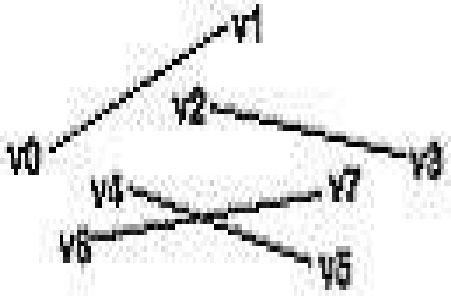
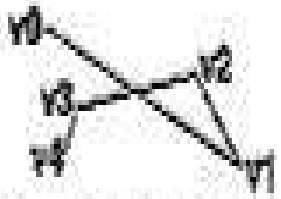
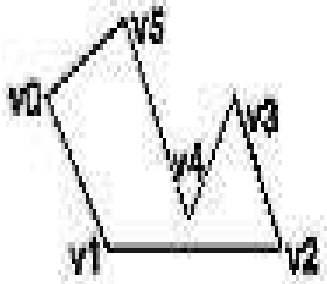
GL_POINTS



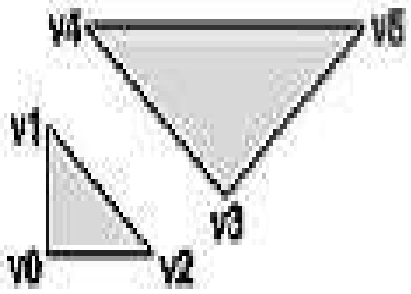
GL_LINES



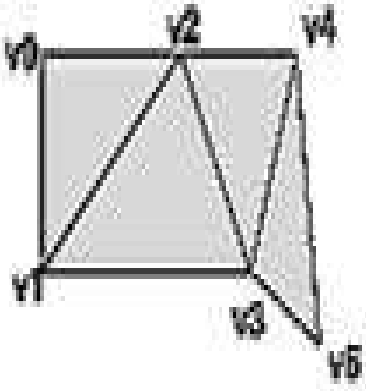
GL_LINE_STRIP



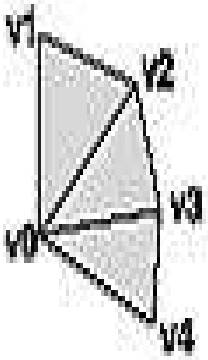
GL_LINE_LOOP



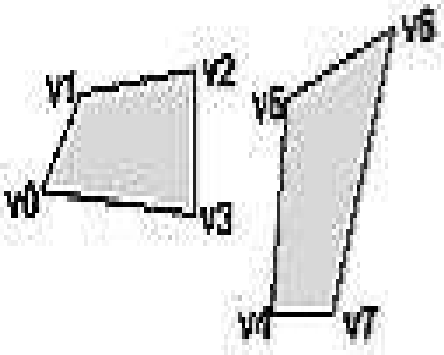
GL_TRIANGLES



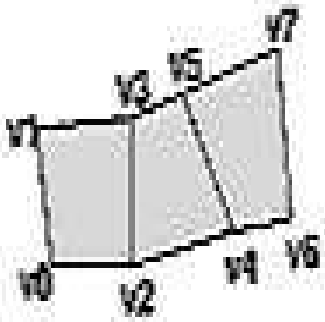
GL_TRIANGLE_STRIP



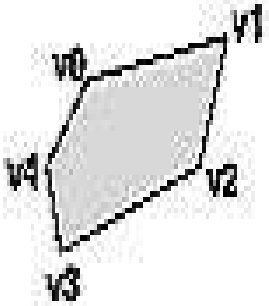
GL_TRIANGLE_FAN



GL_QUADS



GL_QUAD_STRIP



GL_POLYGON

Lập trình Opengl với thư viện AUX

Bùi Minh Trường

Chào mừng các bạn đón đọc đầu sách từ dự án sách cho thiết bị di động
Nguồn: http://vnthuquan.net
Phát hành: Nguyễn Kim Vỹ.

Mục lục

Bùi Minh Trường

Lập trình Opengl với thư viện AUX

(Tài liệu tham khảo của trường đại học wakayama Nhật)

1

I-Giới thiệu về Opengl:

Opengl là thư viện lập trình đồ hoạ 3D, các bạn muốn biết thêm thì xem tại trang chủ opengl. Org.Bài viết này dành cho những bạn đã biết opengl là gì, và nó cũng là bài đầu tiên cho việc học đồ hoạ với opengl sau này.

II-Opengl trong Windows:

Lập trình opengl trong Windows bằng Visual C, bạn phải sử dụng ba thư viện sau glaux.lib glu32.lib và opengl32.lib.Trong Visual C muốn link tới các thư viện này các bạn làm như sau: trên menu(trình đơn) chọn Project sau đó chọn setting rồi cuối cùng trong tab link bạn dánh tên 3 thư viện trên vào(nhớ là có dấu cách giữa các tên của thư viện).Nếu bạn thích sử dụng phím tắt thì chỉ việc bấm Alt+F7 thì cũng được kết quả như trên.Một điều cũng rất quan trọng là khi tạo một project mới bạn phải chọn Window32 console application.Từ bây giờ bạn đã có thể sẵn sàng viết mã lệnh của mình.(Nên nhớ là bạn không phải thêm bất cứ cái gì nữa vì trong VC đã có đầy đủ những cái tôi đề cập ở trên).

III-Tạo một cửa sổ trong opengl:

Dưới đây là mã nguồn cho chương trình đầu tiên của bạn để tạo một cửa sổ.Hãy lưu nó với tên gì tuỳ bạn(ví dụ hello.c như truyền thống)

1-Chương trình đầu tiên của bạn:

```
/*filename: hello.c*/
/*Chương trình đầu tiên tạo một cửa sổ trong opengl*/
#ifdef unix             /*Phần này dùng để xác định môi trường làm việc của bạn*/
#include <GL/gl.h>      /*Nó sẽ xác định bạn biên dịch chương trình này trên unix*/
#include “aux.h“       /*hay Windows, với lập trình viên trên windows bạn có   */
```

```
#define CALLBACK        /*thể bỏ phần bên trên đi và chỉ lấy phần in đậm*/
#else
#include<windows.h>
#include<GL/gl.h>
#include<GL/glaux.h>
#endif
int main(int argc, char *argv[])
{
 auxInitWindow(argv[0]);
 return 0;
}
```

Lệnh auxInitWindow(string); có tác dụng tạo một cửa sổ mới, string là tiêu đề của cửa sổ đó, bạn có thể viết tiêu đề như thế nào là tuỳ bạn.

Chương trình này sau khi biên dịch thì nó mới chỉ hiện ra một cửa sổ rồi đóng ngay, nếu windows của bạn chạy nhanh quá thì bạn sẽ không nhìn thấy chi hết

Sau đây chúng ta sẽ bắt Window dùng lại chừng 1 giây để chúng ta quan sát.Cũng với mã lệnh trên bạn chỉ cần thêm một dòng lệnh: sleep(số_giây_muốn_xem x 1000);(tức là lệnh này bắt window tạm dừng trong vòng 1 phần nghìn giây)

2-Theo dõi Window

```
/*file name: hello1s.cpp*/
#ifdef unix
#include <GL/gl.h>
#include "aux.h"
#define CALLBACK
#else
#include<windows.h>
#include<GL/gl.h>
#include<GL/glaux.h>
#endif
int main(int argc, char *argv[])
{
 auxInitWindow(argv[0]);
/*dòng lệnh mới , window sẽ hiện trong vòng 1 giây*/
 Sleep(1000);
/*dòng lệnh mới*/
 return 0;
}
```

Trong phần source code mã nguồn này nằm trong file hello1s.cpp.

3-Xoá màn hình trong opengl

Tiếp theo tôi sẽ giới thiệu với các bạn cách xoá màn hình trong opengl.Dưới đây là mã nguồn:

```
/*filename: clear.cpp*/
#ifdef unix
#include <GL/gl.h>
#include "aux.h"
#define CALLBACK
```

```
#else
#include<windows.h>
#include<GL/gl.h>
#include<GL/glaux.h>
#endif
int main(int argc, char *argv[])
{
 auxInitWindow(argv[0]);
/*Những dòng lệnh mới*/
 glClearColor(1.0,1.0,1.0,0.0);
 glClear(GL_COLOR_BUFFER_BIT);
 glFlush();
/*Những dòng lệnh mới*/
 Sleep(1000);
 return 0;
}
```

Các lệnh glClearColor(), glClear(),glFush() là những lệnh cơ bản của Opengl.glClearColor() có nhiệm vụ chọn màu để xoá window, bạn dễ dàng nhận ra là nó có 4 tham số, 4 tham số đó là RGBA(red green blue alpha).Không giống với hàm RGB() trong Win32 API , 4 tham số này có giá trị trong khoảng 0.0f đến 1.0f(kiểu float).Ba tham số đầu là màu đỏ xanh lá cây và xanh da trời, còn tham số thứ 4 là độ sáng tối của window.Bây giờ hãy thay đổi các giá trị của màu xem thử!Hàm glClear() mới thực sự xoá window, nó có những hằng số xác định.Có trường hợp có những hàm chưa được chạy đến khi kết thúc chương trình, để tránh trường hợp này hàm glFlush()được gọi, nó sẽ thực hiện tất cả các hàm chưa được chạy và kết thúc chương trình.

4-Vẽ hình trong opengl

Từ trước đến giờ chúng ta mới chỉ nói về cách tạo và xoá cửa sổ, bây giờ chúng ta sẽ thực hiện vẽ một số hình đơn giản:

```
/*filename line.cpp*/
#ifdef unix
#include <GL/gl.h>
#include "aux.h"
#define CALLBACK
#else
#include<windows.h>
#include<GL/gl.h>
#include<GL/glaux.h>
#endif
int main(int argc, char *argv[])
{
 auxInitWindow(argv[0]);
 glClearColor(1.0,1.0,1.0,0.0);
 glClear(GL_COLOR_BUFFER_BIT);
/*những dòng lệnh mới*/
 glBegin(GL_LINE_LOOP);
 glVertex2d(0.1,0.1);
 glVertex2d(0.9,0.1);
 glVertex2d(0.9,0.9);
 glVertex2d(0.1,0.9);
/*những dòng lệnh mới*/
 glEnd();
 glFlush();
```

```
 Sleep(1000);
 return 0;
}
```

Tất cả các hình khối được vẽ trong opengl đều được nằm giữa hai dòng lệnh glBegin() và glEnd() (Hơi giống với pascal- bạn nào học pascal thì dễ hiểu nhé!).Có thể có nhiều cặp dòng lệnh như vậy, tức là bạn có thể viết các hàm vẽ khác nhau và dùng cặp câu lệnh trên trong các hàm đó.Tham số của glBegin() là GL_LINE_LOOP có nghĩa là nó bảo window vẽ một đường khép kín điểm đầu trùng với điểm cuối.

Dưới đây là một số hằng số cơ bản:

Hằng số ý nghĩa

GL_POINT Vẽ điểm

GL_LINÉ Vẽ đường thẳng nối hai điểm

GL_LINE_STRIP Tập hợp của những đoạn được nối với nhau

GL_LINE_LOOP Đường gấp khúc khép kín

GL_TRIANGLES Vẽ hình tam giác

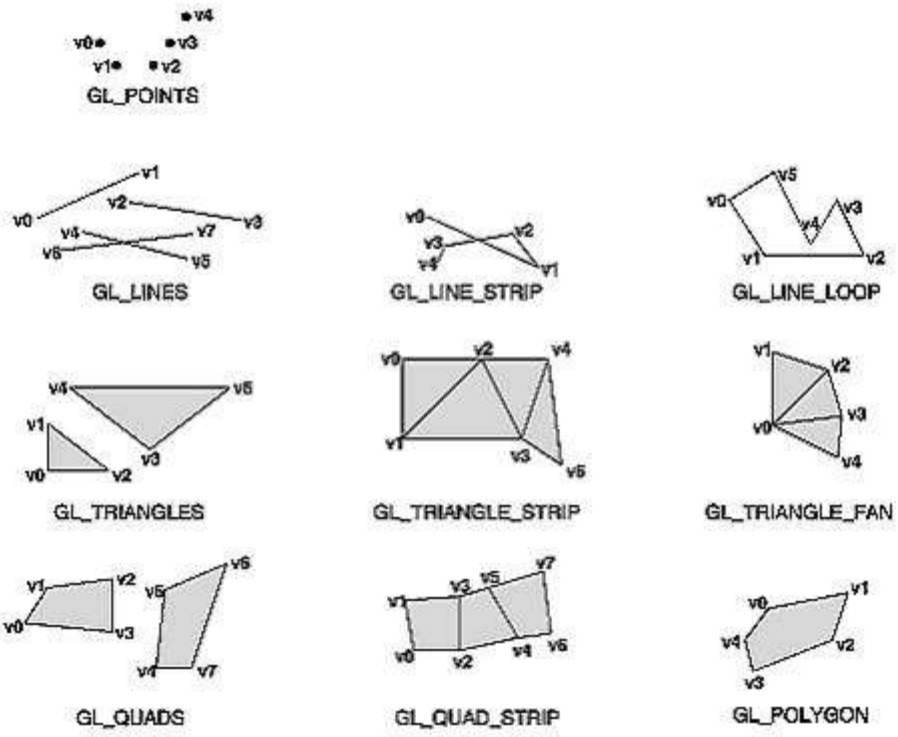
GL_QUADS Vẽ tứ giác

GL_TRIANGLES_STRIP Vẽ một tập hợp các tam giác liền nhau, chung một cạnh

GL_QUAD_STRIP Vẽ một tập hợp các tứ giác liền nhau, chung một cạnh

GL_TRIANGLE_FAN Vẽ hình quạt

Dưới đây là bức tranh toàn cảnh về các thông số này.



Hàm glVertex2d() xác định điểm hai chiều.Bạn nên biết một số tiền tố các hàm của opengl, các hàm dùng thư viện nào sẽ bắt đầu bằng tên của thư viện đó ví dụ dùng các hàm cơ bản của opengl thì thường là bắt đầu với gl, các hàm dùng thư viện glut thì bắt đầu với glu các hàm dùng thư viện aux thì bắt đầu với aux.....Các hàm cũng có hậu tố ví dụ glVertex2d() là vẽ điểm 2 chiều, glVertex3d() là vẽ điểm 3 chiều,....dần dần học các bạn sẽ phát hiện ra nhiều hơn.

5-Sử dụng màu vẽ:

Tiếp theo tôi sẽ hướng dẫn các bạn cách sử dụng màu để vẽ và cách thể hiện nó.

Dưới đây là mã nguồn:

```
/*filename: color1.cpp*/
#ifdef unix
#include <GL/gl.h>
#include "aux.h"
#define CALLBACK
#else
```

```
#include<windows.h>
#include<GL/gl.h>
#include<GL/glaux.h>
#endif
int main(int argc, char *argv[])
{
 auxInitDisplayMode(AUX_RGBA); /*hàm mới*/
 auxInitWindow(argv[0]);
 glClearColor(0.0,0.0,0.0,0.0);
 glClear(GL_COLOR_BUFFER_BIT);
 glColor3d(1.0,0.0,0.0);  /*hàm mới*/
 glClearColor(0.0,0.0,0.0,0.0);
 glClear(GL_COLOR_BUFFER_BIT);
 glBegin(GL_QUADS);          /*tham số mới*/
 glVertex2d(0.1,0.1);
 glVertex2d(0.9,0.1);
 glVertex2d(0.9,0.9);
 glVertex2d(0.1,0.9);
 glEnd();
 glFlush();
 Sleep(1000);
 return 0;
}
```

Hàm auxInitDisplayMode() báo với window rằng chúng ta chọn cách hiển thị những gì mà chúng ta sắp vẽ tới đây, tham số của nó là AUX_RGBA chính là mode RGBA mà tôi đề cập ở trên.Hàm glColor3d() cho phép chúng ta chọn màu vẽ, tham số của nó là red green và blue nhưng các giá trị này là kiểu double nếu bạn muốn dùng kiểu float thì có hàm glColor3f(), cả hai kiểu trên giá trị của màu vẫn nằm trong khoảng 0 đến 1.Chú ý là chương trình trên chúng ta đã đổi tham số mới cho hàm glBegin(), bây giờ nó sẽ vẽ một tứ giác, và trong chương trình này thì là một hình vuông.

Trong phần này tôi muốn trình bày với các bạn một kỹ thuật nữa, chương

trình trên chỉ cho chúng ta nhìn thấy một màu đỏ do chúng ta đặt một màu duy nhất trước khi vẽ. Để có thể tạo nhiều màu ấn tượng bạn có thể cài đặt đi cài đặt lại hàm glColor3d() mỗi khi chúng ta vẽ mới.
Dưới đây là mã nguồn:

```
/*filename: color2.cpp*/
#ifdef unix
#include <GL/gl.h>
#include "aux.h"
#define CALLBACK
#else
#include<windows.h>
#include<GL/gl.h>
#include<GL/glaux.h>
#endif
int main(int argc, char *argv[])
{
 auxInitDisplayMode(AUX_RGBA);
 auxInitWindow(argv[0]);
 glClearColor(0.0,0.0,0.0,0.0);
 glClear(GL_COLOR_BUFFER_BIT);
 glClearColor(0.0,0.0,0.0,0.0);
 glClear(GL_COLOR_BUFFER_BIT);
 glBegin(GL_QUADS);
 glColor3d(1.0,0.0,0.0);  /*hàm này đã được chuyển xuống đây*/
 glVertex2d(0.1,0.1);
 glColor3d(0.0,1.0,0.0);  /*tham số mới cho hàm*/
 glVertex2d(0.9,0.1);
 glColor3d(0.0,0.0,1.0);  /*tham số mới cho hàm*/
 glVertex2d(0.9,0.9);
 glColor3d(1.0,0.0,1.0);  /*tham số mới cho hàm*/
 glVertex2d(0.1,0.9);
 glEnd();
```

```
 glFlush();
 Sleep(1000);
 return 0;
}
```

Biên dịch và chạy thử bạn có một hình vuông trông khá đẹp mắt, nhưng hãy tiếp tục học, chúng ta còn có thể tạo nhiều hiệu ứng ấn tượng hơn nhiều.

Nói thêm chút nữa về cách sử dụng hàm, với các hậu tố: ví dụ với hàm glVertex*() và glColor*(), hay các hàm khác có dấu hoa thị * thì nó có thể có rất nhiều hậu tố.Và nó có cấu tạo như sau: lấy ví dụ hàm glVertex*()

Có hàm glVertex4dv(Gldouble x,Gldouble y,Gldouble z,Gldouble w)

số 4 thể hiện rằng hàm có 4 tham số, chữ d thể hiện rằng tham số có giá trị double(ngoài ra nó còn có thể là float,int,short, unsigned int, unsigned short, unsigned char,char) chữ v thể hiện rằng nó dùng pointer.Các bạn chỉ cần hiểu qua như vậy, sau này chúng ta sẽ nói rõ hơn.

6-Giao diện của cửa sổ và quản lý cửa sổ:
Với những chương trình chỉ cần vẽ đơn giản thì bạn có thể dùng các chương trình trên, nhưng với các chương trình phức tạp sau này chúng ta không thể viết như thể được nữa.Dưới đây tôi sẽ trình bày với các bạn cấu trúc của chương trình trong opengl.
Trước hết là từ khoá CALLBACK, đối với các bạn đã lập trinh WIN API thì có thể hiểu rõ được lệnh này, nhưng có thể nói đơn giản là khi sử dụng thư viện AUX thì ta phải dùng từ khoá này để chỉ định nó.Các chương trình bên trên chúng ta viết đều dùng lệnh Sleep(1000) để bắt window dừng lại cho chúng ta theo dõi, sắp tới đây chúng ta sẽ làm một cách chuyên nghiệp hơn là dùng hàm auxMailLoop() trong thân của hàm main() – hàm chính của chương trình.Tham số của hàm này là con trỏ trỏ đến hàm mà chúng ta vẽ , hiện thị những gì chúng ta muốn(trong chương trình này tham số chính là hàm draw()).Điều gì sẽ xảy ra nếu người dùng thay đổi kích cỡ của cửa sổ? Để thực hiên điều này chúng ta cũng dùng một hàm tương tự như hàm auxMainLoop(),đó là hàm auxReshapeFunc(), tham số của nó cũng là con trỏ chỉ đến hàm mà chúng ta có thể thay đôi thông số của cửa sổ, tham số của nó trong chương trình này là hàm resize().Nếu bạn đã học qua về đồ hoạ máy tính thì sẽ dễ dàng hiểu về toạ độ trong đồ hoạ, hàm glLoadIdentity() có nhiệm vụ thiết định ma trận của toạ độ là ma trận đơn vị.
Mã nguồn dưới đây sẽ cho chúng ta rõ hơn:

```
/*filename: interface.cpp*/
#ifdef unix
#include <GL/gl.h>
#include "aux.h"
#define CALLBACK
```

```
#else
#include<windows.h>
#include<GL/gl.h>
#include<GL/glaux.h>
#endif
GLvoid CALLBACK draw(void){  /*chú ý bạn có thể không cần chữ void
trong */
 glClearColor(0.0,0.0,0.0,0.0); /*khi lập trình với VC, Glvoid là kiểu */
 glClear(GL_COLOR_BUFFER_BIT);/*hàm trong opengl, nó tương tụ */
 glClearColor(0.0,0.0,0.0,0.0); /*như void trong C hay C++*/
 glClear(GL_COLOR_BUFFER_BIT);
 glBegin(GL_QUADS);
 glColor3d(1.0,0.0,0.0);
 glVertex2d(0.1,0.1);
 glColor3d(0.0,1.0,0.0);
 glVertex2d(0.9,0.1);
 glColor3d(0.0,0.0,1.0);
 glVertex2d(0.9,0.9);
 glColor3d(1.0,0.0,1.0);
 glVertex2d(0.1,0.9);
 glEnd();
 glFlush();
}
GLvoid CALLBACK resize(GLsizei w,GLsizei h)
{
 glLoadIdentity();
}

int main(int argc, char *argv[])
{
 auxInitDisplayMode(AUX_RGBA);
 auxInitWindow(argv[0]);
```

```
 auxReshapeFunc(resize);
 auxMainLoop(draw);
 return 0;
}
```

7-Quan sát – Khung nhìn:

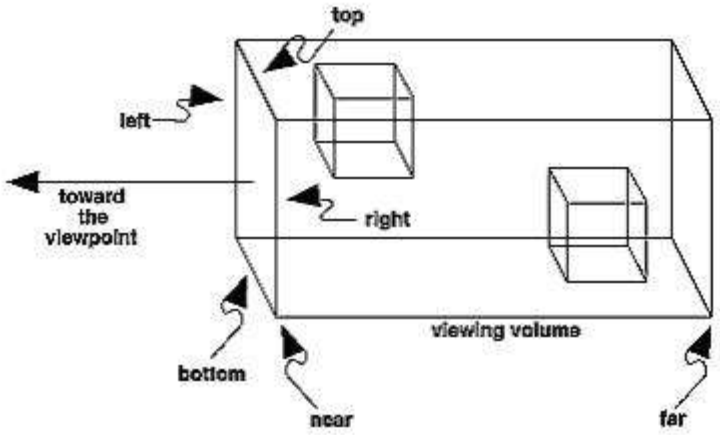
Chương trình trên, khi bạn thay đôi kích cỡ có lúc bạn không nhìn thấy hình vuông mà chúng ta đã vẽ nữa, tại sao lại như vậy?Câu trả lời nằm trong chương trình dưới đây:

```
/*filename: view.cpp*/
#ifdef unix
#include <GL/gl.h>
#include "aux.h"
#define CALLBACK
#else
#include<windows.h>
#include<GL/gl.h>
#include<GL/glaux.h>
#endif
GLvoid CALLBACK draw(void){
 glClearColor(0.0,0.0,0.0,0.0);
 glClear(GL_COLOR_BUFFER_BIT);
 glClearColor(0.0,0.0,0.0,0.0);
 glClear(GL_COLOR_BUFFER_BIT);
 glBegin(GL_QUADS);
 glColor3d(1.0,0.0,0.0);
 glVertex2d(0.1,0.1);
 glColor3d(0.0,1.0,0.0);
 glVertex2d(0.9,0.1);
 glColor3d(0.0,0.0,1.0);
 glVertex2d(0.9,0.9);
 glColor3d(1.0,0.0,1.0);
```

```
 glVertex2d(0.1,0.9);
 glEnd();
 glFlush();
}
GLvoid CALLBACK resize(GLsizei w,GLsizei h)
{
 glLoadIdentity();
 glViewport(0,0,w,h);   /*hàm mới*/
 glOrtho(-1.0,1.0,-1.0,1.0,0.0,1.0);    /*hàm mới*/
}

int main(int argc, char *argv[])
{
 auxInitDisplayMode(AUX_RGBA);
 auxInitWindow(argv[0]);
 auxReshapeFunc(resize);
 auxMainLoop(draw);
 return 0;
}
```

Tôi sẽ giới thiệu với các bạn thế nào là Viewport.Viewport xác định cổng nhìn cho chúng ta, tức là phần không gian trên cửa sổ window ma người quan sát được phép quan sát.Nó chính là một hình chữ nhật.Hai tham số đầu tiên của hàm này xác định toạ độ của đỉnh trên cùng phía tay trái của hình chữ nhật, hai toạ dộ sau xác định chiều rộng và chiều cao của hình chữ nhật ấy.Với các tham số trên ta có thể thấy , chương trình trên cho phép ta quan sát toàn bộ màn hình.

Tiếp theo là kiểu nhìn glOrtho().Quan sát hình vẽ dưới đây:



Như bạn đã thấy trên hình, hàm glOrtho(), xác lập một ma trận cho phép chúng ta nhìn theo kiểu như hình vẽ, đây là hàm tổng quát:

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

Tương ứng với chương trình trên của chúng ta left là –1.0, right là 1.0, bottom là –1.0, top là 1.0, near là 0.0 và far là 1.0.

Trong phần này tôi muốn trình bày thêm một hàm số nữa.Các chương trình trên đều tạo cửa sổ với chiều dài và rộng xác đinh, muốn tạo một cửa sổ có kích cỡ theo ý muốn bạn dùng hàm sau: auxInitPosition(), nó có 4 thông số là toạ độ x, y của đỉnh trên bên tay trái của cửa sổ, chiều rộng và chiều dài của cửa sổ.Dưới đây là mã nguồn:

```
/*filename : size.cpp*/
#ifdef unix
#include <GL/gl.h>
#include "aux.h"
#define CALLBACK
#else
#include<windows.h>
#include<GL/gl.h>
#include<GL/glaux.h>
#endif
GLvoid CALLBACK draw(void){
 glClearColor(0.0,0.0,0.0,0.0);
```

```
 glClear(GL_COLOR_BUFFER_BIT);
 glClearColor(0.0,0.0,0.0,0.0);
 glClear(GL_COLOR_BUFFER_BIT);
 glBegin(GL_QUADS);
 glColor3d(1.0,0.0,0.0);
 glVertex2d(0.1,0.1);
 glColor3d(0.0,1.0,0.0);
 glVertex2d(0.9,0.1);
 glColor3d(0.0,0.0,1.0);
 glVertex2d(0.9,0.9);
 glColor3d(1.0,0.0,1.0);
 glVertex2d(0.1,0.9);
 glEnd();
 glFlush();
}
GLvoid CALLBACK resize(GLsizei w,GLsizei h)
{
 glLoadIdentity();
 glViewport(0,0,w/2,h/2);
 glOrtho(-1.0,1.0,-1.0,1.0,0.0,1.0);
}

int main(int argc, char *argv[])
{
 auxInitPosition(200,100,640,480);   /*hàm mới*/
 auxInitDisplayMode(AUX_RGBA);
 auxInitWindow(argv[0]);
 auxReshapeFunc(resize);
 auxMainLoop(draw);
 return 0;
}
```

8-Chuột:

Trong các trò chơi ta đều thấy sự quan trọng của việc sử dụng chuột, trong phần này chúng ta sẽ xem xét làm thế nào để chương trình chúng ta nhận ra chúng ta đang bấm trái chuột, chúng ta đang di chuyển chuột. Để làm được điều này chúng ta sử dụng hàm auxMouseFunc().Dưới đây là mã nguồn của chương trình mouse1.cpp

```
/*filename mouse1.cpp*/
#ifdef unix
#include <GL/gl.h>
#include "aux.h"
#define CALLBACK
#else
#include<windows.h>
#include<GL/gl.h>
#include<GL/glaux.h>
#include"stdio.h"  /*nếu bạn không có dòng này thì hàm printf() không thực hiện*/
#endif
GLvoid CALLBACK draw(void){
 glClearColor(0.0,0.0,0.0,0.0);
 glClear(GL_COLOR_BUFFER_BIT);
 glClearColor(0.0,0.0,0.0,0.0);
 glClear(GL_COLOR_BUFFER_BIT);
 glBegin(GL_QUADS);
 glColor3d(1.0,0.0,0.0);
 glVertex2d(0.1,0.1);
 glColor3d(0.0,1.0,0.0);
```

```
 glVertex2d(0.9,0.1);
 glColor3d(0.0,0.0,1.0);
 glVertex2d(0.9,0.9);
 glColor3d(1.0,0.0,1.0);
 glVertex2d(0.1,0.9);
 glEnd();
 glFlush();
}
GLvoid CALLBACK left(AUX_EVENTREC *event)
{
                        printf("%d,%d\n",event->data[AUX_MOUSEX],event-
>data[AUX_MOUSEY]);
}
GLvoid CALLBACK resize(GLsizei w,GLsizei h)
{
 glLoadIdentity();
 glViewport(0,0,w/2,h/2);
 glOrtho(-1.0,1.0,-1.0,1.0,0.0,1.0);
}

int main(int argc, char *argv[])
{
 auxInitPosition(200,100,640,480);
 auxInitDisplayMode(AUX_RGBA);
 auxInitWindow(argv[0]);
 auxReshapeFunc(resize);
/*hàm mới*/
 auxMouseFunc(AUX_LEFTBUTTON,AUX_MOUSEDOWN,left);
/*hàm mới*/
 auxMainLoop(draw);
 return 0;
}
```

Trong chương trình trên , chúng ta thấy xuất hiện hàm left() và hàm auxMouseFunc().Hàm auxMouseFunc() có gọi đến hàm left(), nó có ý nghĩa rằng, khi chuột được bấm thì sẽ thực hiện hàm left().Trong tham số của hàm auxMouseFunc() có các tham sô sau: tham số đầu tiên nói đến phần nào của chuột được tác động, tham số thứ 2 nói đến nó được tác động như thế nào, và tham số cuối cùng muốn nói tác động rồi thì làm gì.Trong hàm left() tham số có dạng con trỏ và có kiểu là AUX_EVENTREC, nó lấy dữ liệu về toạ độ x và y của chuột.Trong một chương trình không phải là chỉ có một hàm auxMouseFunc() mà bạn có thể dùng bao nhiêu tuỳ thích, miễn là đừng va chạm nhau là được, trong phần mã nguồn tôi có cho thêm một chương trình ví dụ về cách dùng 2 lần hàm auxMouseFunc()(trong file mouse2.cpp)

Dưới đây tôi sẽ trình bày một chương trình khá thú vị , mã nguồn của nó như sau:

```
/*filename connectlines.cpp*/
#ifdef unix
#include <GL/gl.h>
#include "aux.h"
#define CALLBACK
#else
#include<windows.h>
#include<GL/gl.h>
#include<GL/glaux.h>
#include"stdio.h"
#endif
GLvoid CALLBACK draw(void){
}
GLvoid CALLBACK left(AUX_EVENTREC *event)
{
 static int flag=0;
 static GLint x,y;
 if(flag){
```

```
  glColor3d(0.0,0.0,0.0);
  glBegin(GL_LINE_STRIP);
  glVertex2i(x,y);
                                  glVertex2i(event->data[AUX_MOUSEX],event-
>data[AUX_MOUSEY]);
  glEnd();
  glFlush();
 }
 x=event->data[AUX_MOUSEX];
 y=event->data[AUX_MOUSEY];
 flag=1;
}
GLvoid CALLBACK resize(GLsizei w,GLsizei h)
{
 glLoadIdentity();
 glViewport(0,0,w,h);
 glOrtho(0.0,(GLdouble)w,(GLdouble)h,0.0,0.0,1.0);/* đổi thông số*/
 glClearColor(1.0,1.0,1.0,0.0);  /*chuyển vị trí 2 hàm này*/
 glClear(GL_COLOR_BUFFER_BIT);
}

int main(int argc, char *argv[])
{
 auxInitPosition(200,100,640,480);
 auxInitDisplayMode(AUX_RGBA);
 auxInitWindow(argv[0]);
 auxReshapeFunc(resize);
 auxMouseFunc(AUX_LEFTBUTTON,AUX_MOUSEDOWN,left);
 auxMainLoop(draw);
 return 0;
}
```

Thực ra chương trình này rất dễ hiểu, có lẽ không phải trình bày gì

nhiều.Nó lưu cá điểm lại và nối thành một đường gấp khúc.Nhược điểm của chương trình trên hẳn các bạn đã rõ khi biên dịch nó.Nó không vẽ lại cửa sổ của bạn khi cửa sổ của bạn bị che bởi một cửa sổ khác, hay bị minimize, tức là hình mà bạn muốn vẽ không được gửi tới hàm draw().Vì vậy bạn phải lưu những điểm đã chọn và vẽ lại chúng trong hàm draw().Dưới đây là mã nguồn:

```
/*filename connectlines1.cpp*/
#ifdef unix
#include <GL/gl.h>
#include "aux.h"
#define CALLBACK
#else
#include<windows.h>
#include<GL/gl.h>
#include<GL/glaux.h>
#endif
#define MAXPOINTS 100   /*số điểm tối đa có thể được chọn*/
GLint point[MAXPOINTS][2];  /*mảng lưu trữ các điểm đó*/
int num=0;     /*số điểm đã chọn đến thời điểm hiện tại*/
GLvoid CALLBACK draw(void)
{
 int i;
 if(num>=2){
  glClearColor(1.0,1.0,1.0,0.0);
  glClear(GL_COLOR_BUFFER_BIT);
  glColor3d(0.0,0.0,0.0);
  glBegin(GL_LINE_STRIP);  /*bạn hãy nhớ cấu trúc này*/
  for(i=0;i<num;i++)
  {
glVertex2iv(point[i]);
  }
  glEnd();
```

```
  glFlush();
 }
}
GLvoid CALLBACK left(AUX_EVENTREC *event)
{
 if(num>=MAXPOINTS) return;            /*giới hạn số điểm bạn vẽ  */
  point[num][0]=event->data[AUX_MOUSEX]; /*lưu trữ toạ độ x của chuột*/
  point[num][1]=event->data[AUX_MOUSEY]; /*lưu trữ toạ độ y của chuột*/
 num++;      /*tăng số điểm sau mỗi lần bấm*/
}
GLvoid CALLBACK resize(GLsizei w,GLsizei h)
{
 glLoadIdentity();
 glViewport(0,0,w,h);
 glOrtho(0.0,(GLdouble)w,(GLdouble)h,0.0,0.0,1.0);
}

int main(int argc, char *argv[])
{
 auxInitPosition(200,100,640,480);
 auxInitDisplayMode(AUX_RGBA);
 auxInitWindow(argv[0]);
 auxReshapeFunc(resize);
 auxMouseFunc(AUX_LEFTBUTTON,AUX_MOUSEDOWN,left);
 auxMainLoop(draw);
 return 0;
}
```

Bây giờ bạn không phải lo đến việc cửa sổ không chịu vẽ lại khi nó bị che mất.Một điều cũng đáng chú ý trong chương trình trên là chúng ta đã sử dụng hàm glVertex2iv() hàm này có tham số là thành viên của mảng và

thành viên của mảng có các giá trị x,y là số nguyên, chữ i trong phần hậu tố của hàm trên biểu hiện cho giá trị nguyên còn chữ v biểu hiện cho kiểu pointer.Dưới đây cung cấp cho bạn một chương trình có thể vẽ được cả những đường gấp khúc và các đa giác.Mã nguồn không có gì phức tạp và đáng bàn ở đây cả, nó chỉ là cách sắp xếp dữ liệu và có thêm một hàm right() mà thực ra tôi đã đề cập ở các phần trên.

```
/*filename connectlines2.cpp*/
#ifdef unix
#include <GL/gl.h>
#include "aux.h"
#define CALLBACK
#else
#include<windows.h>
#include<GL/gl.h>
#include<GL/glaux.h>
#endif
#define MAXPOINTS 100
GLint point[MAXPOINTS][2];
int num=0;
int flag=0;
GLvoid CALLBACK draw(void)
{
 int i;

 if(num>=2){
  if(flag){
   flag=0;
   i=num-2;
   glColor3d(0.0,0.0,0.0);
   glBegin(GL_LINE_STRIP);
  }
  else{
```

```
   i=0;
  glClearColor(1.0,1.0,1.0,0.0);
  glClear(GL_COLOR_BUFFER_BIT);
  glColor3d(0.0,0.0,0.0);
  glBegin(GL_POLYGON);
  }
  for(;i<num;i++)
  {
   glVertex2iv(point[i]);
  }
  glEnd();
  glFlush();
 }
}
GLvoid CALLBACK left(AUX_EVENTREC *event)
{
 if(num>=MAXPOINTS) return;
 point[num][0]=event->data[AUX_MOUSEX];
 point[num][1]=event->data[AUX_MOUSEY];
 num++;
 flag=1;
}
GLvoid CALLBACK right(AUX_EVENTREC *event)
{
 draw();
}
GLvoid CALLBACK resize(GLsizei w,GLsizei h)
{
 glLoadIdentity();
 glViewport(0,0,w,h);
 glOrtho(0.0,(GLdouble)w,(GLdouble)h,0.0,0.0,1.0);
 glClearColor(1.0,1.0,1.0,0.0);
```

```
 glClear(GL_COLOR_BUFFER_BIT);
}

int main(int argc, char *argv[])
{
 auxInitPosition(200,100,640,480);
 auxInitDisplayMode(AUX_RGBA);
 auxInitWindow(argv[0]);
 auxReshapeFunc(resize);
 auxMouseFunc(AUX_LEFTBUTTON,AUX_MOUSEDOWN,left);
 auxMouseFunc(AUX_RIGHTBUTTON,AUX_MOUSEDOWN,right);
 auxMainLoop(draw);
 return 0;
}
```

9-Thể hiển toạ độ 3 chiều:

Đến giờ các bạn mới biết đến toạ độ 2 chiều trong opengl, nếu chỉ có vậy thì chẳng khác gì trong lập trình Window cả.Vì vậy trong phần này chúng ta sẽ cùng xem opengl vẽ các hình 3 chiều như thế nào.

```
/*filename : rotated45.cpp*/
#ifdef unix
#include <GL/gl.h>
#include "aux.h"
#define CALLBACK
#else
#include<windows.h>
#include<GL/gl.h>
#include<GL/glaux.h>
#endif
GLvoid CALLBACK draw(void)
{
 glClearColor(0.0,0.0,0.0,0.0);
 glClear(GL_COLOR_BUFFER_BIT);
 glRotated(45,0.0,1.0,0.0);  /*quay quanh trục OY 45 độ*/
 glBegin(GL_QUADS);
 glColor3d(1.0,0.0,0.0);
 glVertex2d(0.1,0.1);
 glColor3d(0.0,1.0,0.0);
 glVertex2d(0.9,0.1);
 glColor3d(0.0,0.0,1.0);
 glVertex2d(0.9,0.9);
 glColor3d(1.0,0.0,1.0);
```

```
 glVertex2d(0.1,0.9);
 glEnd();
 glFlush();
}
GLvoid CALLBACK resize(GLsizei w,GLsizei h)
{
 glLoadIdentity();
 glViewport(0,0,w,h);
 glOrtho(-1.0,1.0,-1.0,1.0,0.0,1.0);
}

int main(int argc, char *argv[])
{
 auxInitPosition(200,100,640,480);
 auxInitDisplayMode(AUX_RGBA);
 auxInitWindow(argv[0]);
 auxReshapeFunc(resize);
 auxMainLoop(draw);
 return 0;
}
```

Chương trình này không có gì đặc biệt ngoài hàm glRotated() , hàm này cho phép chúng ta quay hình tứ giác của chúng ta quanh trục OY với góc quay 45 độ.Tham số đầu tiên của nó là góc sẽ được quay, 3 tham số sau là tham số của vector mà hình của chúng ta sẽ quay với góc quay trên.Bạn nhận thấy rằng các giá trị của vector chúng ta là : toạ độ x bằng 0, toạ độ y bằng 1, toạ độ z bằng 0. Tức là véctơ của chúng ta thẳng đứng theo trục OY, bạn có thể thay đổi các thông số của vector này để kiểm nghiệm hàm này xem !Các giá trị của các thông số này là kiểu double.(Chú ý nếu không thử các thông số khác thì bạn sẽ rất khó để quan sát hàm này hoạt động ra sao)

Tiếp theo tôi xin trình bày với các bạn cách vẽ một hình lập phương thật sự bằng opengl.

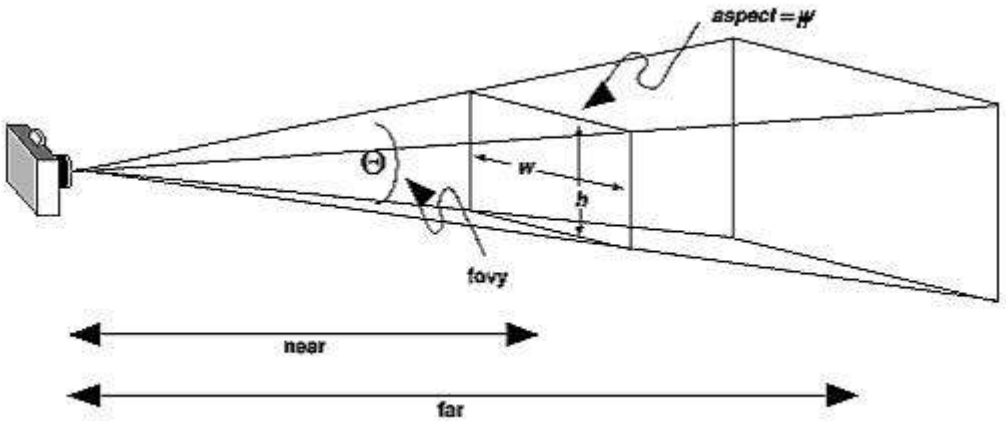
```
/*filename cube1.cpp*/
#ifdef unix
#include <GL/gl.h>
#include "aux.h"
#define CALLBACK
#else
#include<windows.h>
#include<GL/gl.h>
#include<GL/glaux.h>
#endif
GLdouble vertex[][3]={ /*Khai báo dữ liệu cho tám đỉnh của hình lập phương*/
 {0.0,0.0,0.0},
 {1.0,0.0,0.0},
 {1.0,1.0,0.0},
 {0.0,1.0,0.0},
 {0.0,0.0,1.0},
 {1.0,0.0,1.0},
 {1.0,1.0,1.0},
 {0.0,1.0,1.0}
};
int edge[][2]={ /*Khai báo các cạnh, mà chúng ta sẽ sư dụng dữ liệu*/
 {0,1},  /*của các đỉnh bên trên*/
 {1,2},
 {2,3},
 {0,3},
 {4,5},
 {5,6},
 {6,7},
 {7,4},
 {0,4},
 {1,5},
```

```
 {2,6},
 {3,7}
};
GLvoid CALLBACK draw(void)
{
 int i;
 glClearColor(0.0,0.0,0.0,0.0);
 glClear(GL_COLOR_BUFFER_BIT);

 glColor3d(1.0,1.0,1.0);
 glBegin(GL_LINES);
 for(i=0;i<12;i++){
  glVertex3dv(vertex[edge[i][0]]); /*hàm mới*/
  glVertex3dv(vertex[edge[i][1]]); /*hàm mới*/
 }
 glEnd();
 glFlush();
}
GLvoid CALLBACK resize(GLsizei w,GLsizei h)
{
 glLoadIdentity();
 glViewport(0,0,w,h);
 glOrtho(-2.0,2.0,-2.0,2.0,0.0,2.0);
 //gluPerspective(30.0,1.0,1.0,10.0);
 //gluLookAt(3.0,4.0,5.0,0.0,0.0,0.0,0.0,1.0,0.0);
}

int main(int argc, char *argv[])
{
 auxInitPosition(200,100,512,512);
 auxInitDisplayMode(AUX_RGBA);
 auxInitWindow(argv[0]);
```

```
 auxReshapeFunc(resize);
 auxMainLoop(draw);
 return 0;
}
```

Phần khai báo dữ liệu, đối với các bạn chắc thật dễ hiểu, điều đáng nói thứ nhất là chúng ta dùng hàm glVertex3dv() thay cho các hàm vẽ đỉnh 2 chiều trước đây, hàm này nhận tham số là thành viên của mảng, giá trị của các thành viên phải là double, và nó có toạ độ 3 chiều.

Trước hết biên dịch chương trình trên, bạn sẽ chưa thấy gì nếu không cho hai hàm mà tôi đã đánh dấu đỏ bên trên vào.Các bạn hãy nhớ lại cách quan sát bằng glOrtho() trước đây và dễ dàng nhận thấy chúng ta không thể nhìn thấy toàn bộ hình lập phương được mà chỉ là một hình vuông, vì cách nhìn bằng glOrtho() chỉ cho ta nhìn song song thôi.Chính vì vậy mà chúng ta phải chuyển qua cách quan sát bằng gluPerspective(), bốn tham số của nó được trình bày như sau:



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

Tham số đầu tiên là góc quan sát trong mặt phẳng XOZ, trong ví dụ trên góc đó là 30độ, tham số thứ hai là tỉ lệ giữa w và h, nó bằng w/h, tham số thứ 3 và thứ 4 thì hẳn các bạn đã quen khi quan sát bằng glOrtho().Nếu chỉ có hàm này không thôi thì chúng ta vẫn chưa quan sát được hình lập phương mà chúng ta vẽ. Để quan sát được chúng ta phải dùng thêm

hàmgluLookAt()

Hàm này có tới 9 tham số, nhưng thực ra nó nằm trong 3 tham số chính.Tham số đầu tiên là vị trí của mắt, cũng có thể coi đó là vị trí của camera(chú ý là trong toạ độ 3 chiều, nên vị trí của mắt chứa 3 toạ độ), tham số thứ 2 là điểm nhìn, và tham số thứ 3 gọi là upvector, từ này không biết dịch ra tiếng việt ra sao.Upvector , hãy tưởng tượng bạn đang theo dõi một vật, upvector chính là vector từ tim bạn lên đỉnh đầu, nếu thay đổi số liệu cũng tương tự như bạn nghiêng đầu sang phải sang trái.Vậy là 9 tham số đã rõ, bây giờ hãy bỏ lệnh glOrtho()đi và cho 2 lệnh đánh dấu đỏ vào, chúng ta sẽ quan sát được hình lập phương đó, mã nguồn nằm trong file cube2.cpp

10.Animation(Hoạt cảnh)

Phần này sẽ giới thiệu với các bạn về cách tạo hoạt cảnh trong opengl.Hoạt cảnh luôn luôn có sức thu hút người lập trình, nó là một phần quan trọng trong lập trình đồ hoạ.

Trước hết chúng ta sẽ xem xét hai hàm auxIdleFunc() và auxMainLoop().Hàm auxIdleFun() có nhiệm vụ gọi các hàm trong khi không nhận một sự kiện (event) của người dùng, trong chương trình dưới đây, cụ thể là nó sẽ vẽ lại window khi không có event nào.Còn hàm auxMainLoop() chỉ vẽ lại window khi có một sự kiện cụ thể như người dùng di chuyển cửa sổ, nút được bấm, bị cửa sổ khác đè lên.... Để quan sát được rõ ràng chúng ta cũng phải dùng đến hàm glMatrixMode().Khi thay đổi modeling và viewing thì phải thay đổi ma trận của nó, bằng cách dùng hai thông số GL_MODELVIEW và GL_PROJECTION, vì nếu chỉ thay đổi trong lúc khởi tạo window thì ta sẽ không thu được tác dụng của các hàm này khi cửa sổ bị thay đổi, chính vì thế mà chúng ta để nó trong hàm resize(), vì ma trận trên được lặp đi lặp lại nên chúng ta để hàm glMatrixMode(GL_MODELVIEW) sau cùng.Dưới đây là mã nguồn:

```
/*filename animation1.cpp*/
#ifdef unix
#include <GL/gl.h>
#include "aux.h"
#define CALLBACK
#else
#include<windows.h>
#include<GL/gl.h>
#include<GL/glaux.h>
#endif
```

```
GLdouble vertex[][3]={
 {0.0,0.0,0.0},
 {1.0,0.0,0.0},
 {1.0,1.0,0.0},
 {0.0,1.0,0.0},
 {0.0,0.0,1.0},
 {1.0,0.0,1.0},
 {1.0,1.0,1.0},
 {0.0,1.0,1.0}
};
int edge[][2]={
 {0,1},
 {1,2},
 {2,3},
 {0,3},
 {4,5},
 {5,6},
 {6,7},
 {7,4},
 {0,4},
 {1,5},
 {2,6},
 {3,7}
};
GLvoid CALLBACK none(void)
{
}
GLvoid CALLBACK draw(void)
{
 int i;
 static int r=0;
 glClearColor(0.0,0.0,0.0,0.0);
```

```
 glClear(GL_COLOR_BUFFER_BIT);
glLoadIdentity();
 gluLookAt(3.0,4.0,5.0,0.0,0.0,0.0,0.0,1.0,0.0);
 glRotated((double)r,0.0,1.0,0.0);
 glColor3d(1.0,1.0,1.0);
 glBegin(GL_LINES);
 for(i=0;i<12;i++){
  glVertex3dv(vertex[edge[i][0]]);
  glVertex3dv(vertex[edge[i][1]]);
 }
 glEnd();
 glFlush();
 if(++r>=360) r=0;
}
GLvoid CALLBACK resize(GLsizei w,GLsizei h)
{
 glMatrixMode(GL_PROJECTION);
 glLoadIdentity();
 glViewport(0,0,w,h);
 gluPerspective(30.0,1.0,1.0,10.0);
 glMatrixMode(GL_MODELVIEW);
}

int main(int argc, char *argv[])
{
 auxInitPosition(200,100,512,512);
 auxInitDisplayMode(AUX_RGBA);
 auxInitWindow(argv[0]);
 auxReshapeFunc(resize);
 auxIdleFunc(draw);
 auxMainLoop(none);
 return 0;
```

}
Trong chương trình trên, mỗi lần hàm draw()được gọi thì giá trị r được tăng lên một đơn vị nếu vượt quá 360 độ thì nó sẽ trở về 0.Chúng ta phải thành lập hàm none() mặc dù nó không thực hiện một chức năng gì, nhưng hàm auxMainLoop() cần một hàm để gọi đến nó nên ta đã tạo hàm none.Tuy vậy bạn cũng chỉ nhìn thấy nhấp nháy của hình lập phương, để có thể quan sát được hãy biên dịch mã nguồn của chương trình sau:

```
/*filename animation2.cpp*/
#ifdef unix
#include <GL/gl.h>
#include "aux.h"
#define CALLBACK
#else
#include<windows.h>
#include<GL/gl.h>
#include<GL/glaux.h>
#endif
GLdouble vertex[][3]={
 {0.0,0.0,0.0},
 {1.0,0.0,0.0},
 {1.0,1.0,0.0},
 {0.0,1.0,0.0},
 {0.0,0.0,1.0},
 {1.0,0.0,1.0},
 {1.0,1.0,1.0},
 {0.0,1.0,1.0}
};
int edge[][2]={
 {0,1},
 {1,2},
 {2,3},
 {0,3},
```

```
 {4,5},
 {5,6},
 {6,7},
 {7,4},
 {0,4},
 {1,5},
 {2,6},
 {3,7}
};
GLvoid CALLBACK none(void)
{
}
GLvoid CALLBACK draw(void)
{
 int i;
 static int r=0;
 glClearColor(0.0,0.0,0.0,0.0);
 glClear(GL_COLOR_BUFFER_BIT);
 glLoadIdentity();
 gluLookAt(3.0,4.0,5.0,0.0,0.0,0.0,0.0,1.0,0.0);
 glRotated((double)r,0.0,1.0,0.0);
 glColor3d(1.0,1.0,1.0);
 glBegin(GL_LINES);
 for(i=0;i<12;i++){
  glVertex3dv(vertex[edge[i][0]]);
  glVertex3dv(vertex[edge[i][1]]);
 }
 glEnd();
 auxSwapBuffers(); /*hàm mới*/
 if(++r>=360) r=0;
}
GLvoid CALLBACK resize(GLsizei w,GLsizei h)
```

```
{
 glMatrixMode(GL_PROJECTION);
 glLoadIdentity();
 glViewport(0,0,w,h);
 gluPerspective(30.0,1.0,1.0,10.0);
 glMatrixMode(GL_MODELVIEW);
}

int main(int argc, char *argv[])
{
 auxInitPosition(200,100,512,512);
 auxInitDisplayMode(AUX_RGBA|AUX_DOUBLE);/*thông số mới*/
 auxInitWindow(argv[0]);
 auxReshapeFunc(resize);
 auxIdleFunc(draw);
 auxMainLoop(none);
 return 0;
}
```

Trong hàm draw() tôi đã bỏ đi hàm glFlush() và thay bằng hàm auxSwapBuffers(), sử dụng double buffer là một kỹ thuật để tránh hiện tượng nhấp nháy màn hình mà các bạn mới lập trình đồ hoạ thường mắc phải.Kỹ thuật này được mô tả như sau: dùng một offbuffer, rồi vẽ lên đó sau đó mới đưa lên màn hình, tưởng tượng nếu bạn cắt từng chữ rồi dán lên để người xem thấy thì họ sẽ nhìn thấy bạn dán từng chữ một, nhưng nếu bạn dán lên đằng sau tờ giấy rồi lật ngược lại thì họ không biết là nó được dán từng chữ một(tượng trưng thôi, chứ người ta biết thừa)Để dùng được double buffers bạn phải thêm thông số AUX_DOUBLE trong hàm auxInitDisplayMode().Bây giờ bạn đã có một hình lập phương chuyển động mịn màng quanh trục OY.

Thật ra hình lập phương mà chúng ta đã vẽ chỉ là một khung của hình lập phương thôi, còn các mặt thì chúng ta chưa vẽ, vì thế mà các nét khuất chúng ta vẫn nhìn thấy, bây giờ chúng ta sẽ dùng tham số GL_QUADS để

vẽ hình lập phương với các mặt đầy đủ và phần bị khuất sẽ không nhìn thấy được.Mã nguồn:

```
/*filename : animation3.cpp*/
#ifdef unix
#include <GL/gl.h>
#include "aux.h"
#define CALLBACK
#else
#include<windows.h>
#include<GL/gl.h>
#include<GL/glaux.h>
#endif
GLdouble vertex[][3]={
 {0.0,0.0,0.0},
 {1.0,0.0,0.0},
 {1.0,1.0,0.0},
 {0.0,1.0,0.0},
 {0.0,0.0,1.0},
 {1.0,0.0,1.0},
 {1.0,1.0,1.0},
 {0.0,1.0,1.0}
};
int face[][4]={
 {0,1,2,3},
 {1,5,6,2},
 {5,4,7,6},
 {4,0,3,7},
 {4,5,1,0},
 {3,2,6,7}
};
GLvoid CALLBACK none(void)
{
```

```
}
GLvoid CALLBACK draw(void)
{
 int i,j;
 static int r=0;
 glClearColor(0.0,0.0,0.0,0.0);
 glClear(GL_COLOR_BUFFER_BIT);
 glLoadIdentity();
 gluLookAt(3.0,4.0,5.0,0.0,0.0,0.0,0.0,1.0,0.0);
 glRotated((double)r,0.0,1.0,0.0);
 glColor3d(1.0,1.0,1.0);
 glBegin(GL_QUADS);
 for(i=0;i<6;i++)
 for(j=0;j<4;j++){
  glVertex3dv(vertex[face[i][j]]);
 }
 glEnd();
 auxSwapBuffers();
 if(++r>=360) r=0;
}
GLvoid CALLBACK resize(GLsizei w,GLsizei h)
{
 glMatrixMode(GL_PROJECTION);
 glLoadIdentity();
 glViewport(0,0,w,h);
 gluPerspective(30.0,1.0,1.0,10.0);
 glMatrixMode(GL_MODELVIEW);
}

int main(int argc, char *argv[])
{
 auxInitPosition(200,100,512,512);
```

```
 auxInitDisplayMode(AUX_RGBA|AUX_DOUBLE);
 auxInitWindow(argv[0]);
 auxReshapeFunc(resize);
 auxIdleFunc(draw);
 auxMainLoop(none);
 return 0;
}
```

Tiếp theo chúng ta sẽ thêm màu vào để hình của chúng ta được sinh động, bạn chú ý cách sắp xếp dữ liệu của tôi, rất nhiều nhà lập trình thích cách sắp xếp này.Mã nguồn:

```
/*filename animation4.cpp*/
#ifdef unix
#include <GL/gl.h>
#include "aux.h"
#define CALLBACK
#else
#include<windows.h>
#include<GL/gl.h>
#include<GL/glaux.h>
#endif
GLdouble vertex[][3]={
 {0.0,0.0,0.0},
 {1.0,0.0,0.0},
 {1.0,1.0,0.0},
 {0.0,1.0,0.0},
 {0.0,0.0,1.0},
 {1.0,0.0,1.0},
 {1.0,1.0,1.0},
 {0.0,1.0,1.0}
};
int face[][4]={
 {0,1,2,3},
```

```
 {1,5,6,2},
 {5,4,7,6},
 {4,0,3,7},
 {4,5,1,0},
 {3,2,6,7}
};
GLdouble color[][3]={
 {1.0,0.0,0.0},
 {0.0,1.0,0.0},
 {0.0,0.0,1.0},
 {1.0,1.0,0.0},
 {1.0,0.0,1.0},
 {0.0,1.0,1.0}
};
GLvoid CALLBACK none(void)
{
}
GLvoid CALLBACK draw(void)
{
 int i,j;
 static int r=0;
 glClearColor(0.0,0.0,0.0,0.0);
 glClear(GL_COLOR_BUFFER_BIT);
 glLoadIdentity();
 gluLookAt(3.0,4.0,5.0,0.0,0.0,0.0,0.0,1.0,0.0);
 glRotated((double)r,0.0,1.0,0.0);
 glBegin(GL_QUADS);
 for(i=0;i<6;i++){
  glColor3dv(color[i]);
  for(j=0;j<4;j++){
   glVertex3dv(vertex[face[i][j]]);
  }
```

```
 }
 glEnd();
 auxSwapBuffers();
 if(++r>=360) r=0;
}
GLvoid CALLBACK resize(GLsizei w,GLsizei h)
{
 glMatrixMode(GL_PROJECTION);
 glLoadIdentity();
 glViewport(0,0,w,h);
 gluPerspective(30.0,1.0,1.0,10.0);
 glMatrixMode(GL_MODELVIEW);
}

int main(int argc, char *argv[])
{
 auxInitPosition(200,100,512,512);
 auxInitDisplayMode(AUX_RGBA|AUX_DOUBLE);
 auxInitWindow(argv[0]);
 auxReshapeFunc(resize);
 auxIdleFunc(draw);
 auxMainLoop(none);
 return 0;
}
```

Tạm thời dừng phần animation ở đây chúng ta chuyển qua phần khác.

10-Sử dụng Depth buffer test:

Mới nghe tên thôi các bạn cũng có thể hình dung được công việc chúng ta sắp làm đó là tạo chiều sâu và độ xa từ khi quan sát vật từ điểm quan sát.Trong phần này chúng ta sử dụng thêm thư viện nữa vì thế hãy thêm vào tiêu đề file GL/glu.h(cái này cũng có sẵn trong VC bạn không phải download ở đâu cả).Dưới đây là mã nguồn

```
/*filename depth1.cpp*/
```

```
#ifdef unix
#include <GL/gl.h>
#include "aux.h"
#define CALLBACK
#else
#include<windows.h>
#include<GL/gl.h>
#include<GL/glaux.h>
#endif
#include<GL/glu.h>
GLdouble vertex[][3]={
 {0.0,0.0,0.0},
 {1.0,0.0,0.0},
 {1.0,1.0,0.0},
 {0.0,1.0,0.0},
 {0.0,0.0,1.0},
 {1.0,0.0,1.0},
 {1.0,1.0,1.0},
 {0.0,1.0,1.0}
};
int face[][4]={
 {0,1,2,3},
 {1,5,6,2},
 {5,4,7,6},
 {4,0,3,7},
 {4,5,1,0},
 {3,2,6,7}
};
GLdouble color[][3]={
 {1.0,0.0,0.0},
 {0.0,1.0,0.0},
 {0.0,0.0,1.0},
```

```
 {1.0,1.0,0.0},
 {1.0,0.0,1.0},
 {0.0,1.0,1.0}
};
GLvoid CALLBACK none(void)
{
}
GLvoid CALLBACK draw(void)
{
 int i,j;
 static int r=0;
 glClearColor(0.0,0.0,0.0,0.0);
   glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);/*thôn
g sô mới*/
 glLoadIdentity();
 gluLookAt(3.0,4.0,5.0,0.0,0.0,0.0,0.0,1.0,0.0);
 glRotated((double)r,0.0,1.0,0.0);
 glEnable(GL_DEPTH_TEST);  /*hàm mới*/
 glBegin(GL_QUADS);
 for(i=0;i<6;i++){
  glColor3dv(color[i]);
  for(j=0;j<4;j++){
   glVertex3dv(vertex[face[i][j]]);
  }
 }
 glEnd();

 glDisable(GL_DEPTH_TEST); /*hàm mới*/

 auxSwapBuffers();
 if(++r>=360) r=0;
}
```

```
GLvoid CALLBACK resize(GLsizei w,GLsizei h)
{
 glMatrixMode(GL_PROJECTION);
 glLoadIdentity();
 glViewport(0,0,w,h);
 gluPerspective(30.0,1.0,1.0,10.0);
 glMatrixMode(GL_MODELVIEW);
}

int main(int argc, char *argv[])
{
 auxInitPosition(200,100,512,512);
   auxInitDisplayMode(AUX_RGBA|AUX_DOUBLE|AUX_DEPTH);/*thô
ng số mới*/
 auxInitWindow(argv[0]);
 auxReshapeFunc(resize);
 auxIdleFunc(draw);
 auxMainLoop(none);
 return 0;
}
```

Trong chương trình trên chúng ta đã sử dụng tham số GL_DEPTH_BUFFER_BIT, muốn xoá buffer z thì chúng ta phải dùng tham số này(mục đích là cài đặt z buffer), hai hàm glEnable() và glDisable() có tác dụng khởi động buffer z(chiều sâu) và tắt nó đi, được dùng với thông số GL_DEPTH_TEST.Trong hàm auxInitDisplayMode() chúng ta cũng phải thêm thông số AUX_DEPTH để có thể quan sát được chiều sâu của đồ vật.Tuy vậy chương trình trên có một số yếu điểm, đó là việc liên quan đến các mặt khuất.Thử biên dịch mã nguồn dưới đây và quan sát sự khác nhau:

```
/*filename depth2.cpp*/
#ifdef unix
#include <GL/gl.h>
```

```
#include "aux.h"
#define CALLBACK
#else
#include<windows.h>
#include<GL/gl.h>
#include<GL/glaux.h>
#endif
#include<GL/glu.h>
GLdouble vertex[][3]={
 {0.0,0.0,0.0},
 {1.0,0.0,0.0},
 {1.0,1.0,0.0},
 {0.0,1.0,0.0},
 {0.0,0.0,1.0},
 {1.0,0.0,1.0},
 {1.0,1.0,1.0},
 {0.0,1.0,1.0}
};
int face[][4]={
 {0,1,2,3},
 {1,5,6,2},
 {5,4,7,6},
 {4,0,3,7},
 {4,5,1,0},
 {3,2,6,7}
};
GLdouble color[][3]={
 {1.0,0.0,0.0},
 {0.0,1.0,0.0},
 {0.0,0.0,1.0},
 {1.0,1.0,0.0},
 {1.0,0.0,1.0},
```

```
 {0.0,1.0,1.0}
};
GLvoid CALLBACK none(void)
{
}
GLvoid CALLBACK draw(void)
{
 int i,j;
 static int r=0;
 glClearColor(0.0,0.0,0.0,0.0);
 glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
 glLoadIdentity();
 gluLookAt(3.0,4.0,5.0,0.0,0.0,0.0,0.0,1.0,0.0);
 glRotated((double)r,0.0,1.0,0.0);
 glEnable(GL_DEPTH_TEST);
 glEnable(GL_CULL_FACE);
 glBegin(GL_QUADS);
 for(i=0;i<6;i++){
  glColor3dv(color[i]);
  for(j=0;j<4;j++){
   glVertex3dv(vertex[face[i][j]]);
  }
 }
 glEnd();
 glDisable(GL_CULL_FACE);
 glDisable(GL_DEPTH_TEST);

 auxSwapBuffers();
 if(++r>=360) r=0;
}
GLvoid CALLBACK resize(GLsizei w,GLsizei h)
{
```

```
 glMatrixMode(GL_PROJECTION);
 glLoadIdentity();
 glViewport(0,0,w,h);
 gluPerspective(30.0,1.0,1.0,10.0);
 glMatrixMode(GL_MODELVIEW);
}

int main(int argc, char *argv[])
{
 auxInitPosition(200,100,512,512);
 auxInitDisplayMode(AUX_RGBA|AUX_DOUBLE|AUX_DEPTH);
 auxInitWindow(argv[0]);
 auxReshapeFunc(resize);
 auxIdleFunc(draw);
 glCullFace(GL_BACK);
 auxMainLoop(none);
 return 0;
}
```

các bạn thấy rõ sự khác nhau rồi chứ?Chương trình trước, các cạnh dường như song song, trông không giống một vật 3 chiều chút nào, vì các mặt được chọn để thể hiện chưa được làm chính xác.Muốn hiểu rõ hơn xin xem tại trang opengl.org.

11-Sử dụng ánh sáng trong opengl.
Để xác định mặt nào được chiếu sáng và với cường độ sáng bao nhiêu, người ta dùng véc tơ pháp tuyến (normal vector).Trong chương trình dưới đây tôi sẽ giới thiệu cách dùng véc tơ này.Mã nguồn:

```
/*filename: light1.cpp*/
#ifdef unix
#include <GL/gl.h>
#include "aux.h"
#define CALLBACK
#else
#include<windows.h>
#include<GL/gl.h>
#include<GL/glaux.h>
#endif
#include<GL/glu.h>
GLdouble vertex[][3]={
 {0.0,0.0,0.0},
 {1.0,0.0,0.0},
 {1.0,1.0,0.0},
 {0.0,1.0,0.0},
 {0.0,0.0,1.0},
 {1.0,0.0,1.0},
 {1.0,1.0,1.0},
 {0.0,1.0,1.0}
};
int face[][4]={
 {0,1,2,3},
```

```
 {1,5,6,2},
 {5,4,7,6},
 {4,0,3,7},
 {4,5,1,0},
 {3,2,6,7}
};
GLdouble normal[][3]={
 {0.0,0.0,-1.0},
 {1.0,0.0,0.0},
 {0.0,0.0,1.0},
 {-1.0,0.0,0.0},
 {0.0,-1.0,0.0},
 {0.0,1.0,0.0}
};
GLvoid CALLBACK none(void)
{
}
GLvoid CALLBACK draw(void)
{
 int i,j;
 static int r=0;
 glClearColor(0.0,0.0,0.0,0.0);
 glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
 glLoadIdentity();
 gluLookAt(3.0,4.0,5.0,0.0,0.0,0.0,0.0,1.0,0.0);
 glRotated((double)r,0.0,1.0,0.0);
 glEnable(GL_DEPTH_TEST);
 glEnable(GL_LIGHTING); /*tham số mới*/
 glEnable(GL_LIGHT0);  /*tham số mới*/
 glBegin(GL_QUADS);
 for(i=0;i<6;i++){
  glNormal3dv(normal[i]);  /*hàm mới*/
```

```
  for(j=0;j<4;j++){
   glVertex3dv(vertex[face[i][j]]);
  }
 }
 glEnd();
 glDisable(GL_LIGHT0);  /*tham số mới*/
 glDisable(GL_LIGHTING); /*tham số mới*/
 glDisable(GL_DEPTH_TEST);

 auxSwapBuffers();
 if(++r>=360) r=0;
}
GLvoid CALLBACK resize(GLsizei w,GLsizei h)
{
 glMatrixMode(GL_PROJECTION);
 glLoadIdentity();
 glViewport(0,0,w,h);
 gluPerspective(30.0,1.0,1.0,10.0);
 glMatrixMode(GL_MODELVIEW);
}

int main(int argc, char *argv[])
{
 auxInitPosition(200,100,512,512);
 auxInitDisplayMode(AUX_RGBA|AUX_DOUBLE|AUX_DEPTH);
 auxInitWindow(argv[0]);
 auxReshapeFunc(resize);
 auxIdleFunc(draw);
 glCullFace(GL_BACK);
 auxMainLoop(none);
 return 0;
}
```

Các bạn đã thấy thông số mới trong hàm glEnable() và glDisable(), các thông số này cho phép chúng ta sử dụng ánh sáng trong khi tạo đồ vật.Opengl cung cấp cho chúng ta 8 nguồn sáng mà chương trình trên mới chỉ sử dụng một nguồn sáng LIGHT0, hàm glNormal3dv() thiết lập véc tơ pháp tuyến cho mỗi mặt chúng ta vẽ, vì thế khi vật thể quay quanh trục thì ta sẽ thấy độ sáng tối thay đổi.

Trong chương trình light0.cpp trên chúng ta không sử dụng màu nên màu sẽ được xác định là màu mặc định như các bạn đã thấy.Tiếp theo đây tôi xin giới thiệu cách sử dụng vật liệu để tạo vật với các màu sắc ấn tượng khác nhau.Chúng ta khai báo dữ liệu sau:

```
GLfloat ambient[]={ 0.2,0.2,0.2,1.0};
GLfloat diffuse[]={ 0.8,0.0,0.0,1.0};
GLfloat specular[]={0.5,0.5,0.5,1.0};
GLfloat shininess=40.0;
```

Trong đó các thông số trên đều được đặt theo hệ RGBA(như đã đề cập ở các phần trên), chính vì thế mà chúng có 4 giá trị.Thông số ambient biểu hiện cho độ sáng của môi trường bên ngoài, diffuse là độ khuếch tán, specular là độ phản xạ và shininess là độ bóng(độ bóng sáng). Các thông số trên được hàm glMaterial*() sử dụng để tạo vật thể của chúng ta.Hãy quan sát cách khai báo hàm trên:

```
glMaterialfv(GL_FRONT_AND_BACK,GL_AMBIENT,ambient);
 glMaterialfv(GL_FRONT_AND_BACK,GL_DIFFUSE,diffuse);
 glMaterialfv(GL_FRONT_AND_BACK,GL_SPECULAR,specular);
 glMaterialf(GL_FRONT_AND_BACK,GL_SHININESS,shininess);
```

Chú ý là lệnh cuối cùng không phải là “fv” như bình thường, vì shininess chỉ là một số.

Đây là mã nguồn của chương trình.

```
/*filename light1.cpp*/
#ifdef unix
#include <GL/gl.h>
#include "aux.h"
#define CALLBACK
```

```
#else
#include<windows.h>
#include<GL/gl.h>
#include<GL/glaux.h>
#endif
#include<GL/glu.h>
GLdouble vertex[][3]={
 {0.0,0.0,0.0},
 {1.0,0.0,0.0},
 {1.0,1.0,0.0},
 {0.0,1.0,0.0},
 {0.0,0.0,1.0},
 {1.0,0.0,1.0},
 {1.0,1.0,1.0},
 {0.0,1.0,1.0}
};
int face[][4]={
 {0,1,2,3},
 {1,5,6,2},
 {5,4,7,6},
 {4,0,3,7},
 {4,5,1,0},
 {3,2,6,7}
};
GLdouble normal[][3]={
 {0.0,0.0,-1.0},
 {1.0,0.0,0.0},
 {0.0,0.0,1.0},
 {-1.0,0.0,0.0},
 {0.0,-1.0,0.0},
 {0.0,1.0,0.0}
};
```

```
GLfloat ambient[]={ 0.2,0.2,0.2,1.0};
GLfloat diffuse[]={ 0.8,0.0,0.0,1.0};
GLfloat specular[]={0.5,0.5,0.5,1.0};
GLfloat shininess=40.0;
GLvoid CALLBACK none(void)
{
}
GLvoid CALLBACK draw(void)
{
 int i,j;
 static int r=0;
 glClearColor(0.0,0.0,0.0,0.0);
 glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
 glLoadIdentity();
 gluLookAt(3.0,4.0,5.0,0.0,0.0,0.0,0.0,1.0,0.0);
 glRotated((double)r,0.0,1.0,0.0);
 glEnable(GL_DEPTH_TEST);
 glEnable(GL_LIGHTING);
 glEnable(GL_LIGHT0);
 glMaterialfv(GL_FRONT_AND_BACK,GL_AMBIENT,ambient);
 glMaterialfv(GL_FRONT_AND_BACK,GL_DIFFUSE,diffuse);
 glMaterialfv(GL_FRONT_AND_BACK,GL_SPECULAR,specular);
 glMaterialf(GL_FRONT_AND_BACK,GL_SHININESS,shininess);
 glBegin(GL_QUADS);
 for(i=0;i<6;i++){
  glNormal3dv(normal[i]);
  for(j=0;j<4;j++){
   glVertex3dv(vertex[face[i][j]]);
  }
 }
 glEnd();
 glDisable(GL_LIGHT0);
```

```
 glDisable(GL_LIGHTING);
 glDisable(GL_DEPTH_TEST);

 auxSwapBuffers();
 if(++r>=360) r=0;
}
GLvoid CALLBACK resize(GLsizei w,GLsizei h)
{
 glMatrixMode(GL_PROJECTION);
 glLoadIdentity();
 glViewport(0,0,w,h);
 gluPerspective(30.0,1.0,1.0,10.0);
 glMatrixMode(GL_MODELVIEW);
}

int main(int argc, char *argv[])
{
 auxInitPosition(200,100,512,512);
 auxInitDisplayMode(AUX_RGBA|AUX_DOUBLE|AUX_DEPTH);
 auxInitWindow(argv[0]);
 auxReshapeFunc(resize);
 auxIdleFunc(draw);
 glCullFace(GL_BACK);
 auxMainLoop(none);
 return 0;
}
```

Bùi Minh Trường

Lập trình Opengl với thư viện AUX

(Tài liệu tham khảo của trường đại học wakayama Nhật)

10

12-Vẽ nhiều vật-Dùng Ma trân.

Trong phần này tôi sẽ giới thiều với các bạn cách vẽ có vẻ chuyên nghiệp hơn một chút, có thể từ đây bạn sẽ tạo được những điều mà mình mong muốn.

Trước hết hãy tạo một hàm vẽ hình lập phương:

```
void cube()
{
 int i,j;
 glBegin(GL_QUADS);
 for(i=0;i<6;i++){
  glNormal3dv(normal[i]);
  for(j=0;j<4;j++){
   glVertex3dv(vertex[face[i][j]]);
  }
 }
 glEnd();
}
```

Vì điểm nhìn không thay đôi nên khi cỡ của window thay đổi ta cũng phải thay đổi theo. Dưới đây là mã nguồn để thay đổi khung nhìn của chúng ta:

```
GLvoid CALLBACK resize(GLsizei w,GLsizei h)
{
 glMatrixMode(GL_PROJECTION);
 glLoadIdentity();
 glViewport(0,0,w,h);
 gluPerspective(30.0,1.0,1.0,10.0);
 glMatrixMode(GL_MODELVIEW);
 glLoadIdentity();
```

```
 gluLookAt(3.0,4.0,5.0,0.0,0.0,0.0,0.0,1.0,0.0);
}
```

Chú ý là hai hàm cuối trong hàm trên đã được chuyển từ hàm draw() sang.Nếu thiết đặt ma trân như trên thì khi hàm glRotated() làm thay đổi vị trí của vật sẽ làm cho window của chúng ta trở nên không bình thường.Vì vậy trước khi dùng hàm glRotated() thì chúng ta phải lưu ma trận vào đã rồi khì thực hiện xong hàm này ta lại trả lại ma trận thì sẽ bình thương. Để làm việc này chúng ta sử dụng 2 hàm glPushMatrix() và glPopMatrix().Mã nguồn được trình bày dưới đây.

```
/*filename matrix1.cpp*/
#ifdef unix
#include <GL/gl.h>
#include "aux.h"
#define CALLBACK
#else
#include<windows.h>
#include<GL/gl.h>
#include<GL/glaux.h>
#endif
#include<GL/glu.h>
GLdouble vertex[][3]={
 {0.0,0.0,0.0},
 {1.0,0.0,0.0},
 {1.0,1.0,0.0},
 {0.0,1.0,0.0},
 {0.0,0.0,1.0},
 {1.0,0.0,1.0},
 {1.0,1.0,1.0},
 {0.0,1.0,1.0}
};
int face[][4]={
 {0,1,2,3},
```

```
 {1,5,6,2},
 {5,4,7,6},
 {4,0,3,7},
 {4,5,1,0},
 {3,2,6,7}
};
GLdouble normal[][3]={
 {0.0,0.0,-1.0},
 {1.0,0.0,0.0},
 {0.0,0.0,1.0},
 {-1.0,0.0,0.0},
 {0.0,-1.0,0.0},
 {0.0,1.0,0.0}
};
void cube()
{
 int i,j;
 glBegin(GL_QUADS);
 for(i=0;i<6;i++){
  glNormal3dv(normal[i]);
  for(j=0;j<4;j++){
   glVertex3dv(vertex[face[i][j]]);
  }
 }
 glEnd();
}
GLvoid CALLBACK none(void)
{
}
GLvoid CALLBACK draw(void)
{
 static int r=0;
```

```
 glClearColor(0.0,0.0,0.0,0.0);
 glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
 glEnable(GL_DEPTH_TEST);
 glEnable(GL_LIGHTING);
 glEnable(GL_LIGHT0);
 glPushMatrix();
 glRotated((double)r,0.0,1.0,0.0);
 cube();
 glPopMatrix();
 glDisable(GL_LIGHT0);
 glDisable(GL_LIGHTING);
 glDisable(GL_DEPTH_TEST);

 auxSwapBuffers();
 if(++r>=360) r=0;
}
GLvoid CALLBACK resize(GLsizei w,GLsizei h)
{
 glMatrixMode(GL_PROJECTION);
 glLoadIdentity();
 glViewport(0,0,w,h);
 gluPerspective(30.0,1.0,1.0,10.0);
 glMatrixMode(GL_MODELVIEW);
 glLoadIdentity();
 gluLookAt(3.0,4.0,5.0,0.0,0.0,0.0,0.0,1.0,0.0);
}

int main(int argc, char *argv[])
{
 auxInitPosition(200,100,512,512);
 auxInitDisplayMode(AUX_RGBA|AUX_DOUBLE|AUX_DEPTH);
 auxInitWindow(argv[0]);
```

```
 auxReshapeFunc(resize);
 auxIdleFunc(draw);
 glCullFace(GL_BACK);
 auxMainLoop(none);
 return 0;
}
```

Bây giờ ta sẽ tìm cách để vẽ hai hình lập phương mà không mất công vẽ lại từng mặt của hình lập phương mới nữa. Để làm được điều này bạn dùng hàm glTranslated(), hàm này có 3 thông số, nó có nhiệm vụ chuyển ta đến vị trí mới để vẽ(qua ba thông số của nó).Thực ra nó nhân ma trận hiện tại với một ma trận khác để chuyển toạ độ cho chúng ta, vì thế mà chúng ta sẽ phải dùng 2 lần các hàm glPushMatrix() và glPopMatrix().Hãy nhớ là bao nhiêu lần gọi hàm glPushMatrix() thì cũng phải từng đấy lần gọi hàm glPopmatrix().Dưới đây là mã nguồn:

```
/*filename matrix2.cpp*/
#ifdef unix
#include <GL/gl.h>
#include "aux.h"
#define CALLBACK
#else
#include<windows.h>
#include<GL/gl.h>
#include<GL/glaux.h>
#endif
#include<GL/glu.h>
GLdouble vertex[][3]={
 {0.0,0.0,0.0},
 {1.0,0.0,0.0},
 {1.0,1.0,0.0},
 {0.0,1.0,0.0},
 {0.0,0.0,1.0},
 {1.0,0.0,1.0},
```

```
 {1.0,1.0,1.0},
 {0.0,1.0,1.0}
};
int face[][4]={
 {0,1,2,3},
 {1,5,6,2},
 {5,4,7,6},
 {4,0,3,7},
 {4,5,1,0},
 {3,2,6,7}
};
GLdouble normal[][3]={
 {0.0,0.0,-1.0},
 {1.0,0.0,0.0},
 {0.0,0.0,1.0},
 {-1.0,0.0,0.0},
 {0.0,-1.0,0.0},
 {0.0,1.0,0.0}
};
void cube()
{
 int i,j;
 glBegin(GL_QUADS);
 for(i=0;i<6;i++){
  glNormal3dv(normal[i]);
  for(j=0;j<4;j++){
   glVertex3dv(vertex[face[i][j]]);
  }
 }
 glEnd();
}
GLvoid CALLBACK none(void)
```

```
{
}
GLvoid CALLBACK draw(void)
{
 static int r=0;
 glClearColor(0.0,0.0,0.0,0.0);
 glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
 glEnable(GL_DEPTH_TEST);
 glEnable(GL_LIGHTING);
 glEnable(GL_LIGHT0);
 glPushMatrix();
 glRotated((double)r,0.0,1.0,0.0);
 cube();
 glPushMatrix();
 glTranslated(1.0,1.0,1.0);
 cube();
 glPopMatrix();
 glPopMatrix();
 glDisable(GL_LIGHT0);
 glDisable(GL_LIGHTING);
 glDisable(GL_DEPTH_TEST);

 auxSwapBuffers();
 if(++r>=360) r=0;
}
GLvoid CALLBACK resize(GLsizei w,GLsizei h)
{
 glMatrixMode(GL_PROJECTION);
 glLoadIdentity();
 glViewport(0,0,w,h);
 gluPerspective(30.0,1.0,1.0,10.0);
 glMatrixMode(GL_MODELVIEW);
```

```
 glLoadIdentity();
 gluLookAt(3.0,4.0,5.0,0.0,0.0,0.0,0.0,1.0,0.0);
}

int main(int argc, char *argv[])
{
 auxInitPosition(200,100,512,512);
 auxInitDisplayMode(AUX_RGBA|AUX_DOUBLE|AUX_DEPTH);
 auxInitWindow(argv[0]);
 auxReshapeFunc(resize);
 auxIdleFunc(draw);
 glCullFace(GL_BACK);
 auxMainLoop(none);
 return 0;
}
```

Tiếp theo chúng ta sẽ cho hình lập phương thứ hai quay với vận tốc khác, vận tốc mới của nó là 2*r, để làm được điều này bạn chỉ cần thêm hàm glRotated((double)(2*r),0.0,1.0,0.0); vào sau hàm glTranslated(1.0,1.0,1.0); v à trước hàm cube() thứ hai.Mã nguồn nằm trong file matrix3.cpp.

Bây giờ hãy cùng tô màu cho chúng.

```
/*filename matrix4.cpp*/
#ifdef unix
#include <GL/gl.h>
#include "aux.h"
#define CALLBACK
#else
#include<windows.h>
#include<GL/gl.h>
#include<GL/glaux.h>
#endif
#include<GL/glu.h>
GLdouble vertex[][3]={
```

```
 {0.0,0.0,0.0},
 {1.0,0.0,0.0},
 {1.0,1.0,0.0},
 {0.0,1.0,0.0},
 {0.0,0.0,1.0},
 {1.0,0.0,1.0},
 {1.0,1.0,1.0},
 {0.0,1.0,1.0}
};
int face[][4]={
 {0,1,2,3},
 {1,5,6,2},
 {5,4,7,6},
 {4,0,3,7},
 {4,5,1,0},
 {3,2,6,7}
};
GLdouble normal[][3]={
 {0.0,0.0,-1.0},
 {1.0,0.0,0.0},
 {0.0,0.0,1.0},
 {-1.0,0.0,0.0},
 {0.0,-1.0,0.0},
 {0.0,1.0,0.0}
};
void cube()
{
 int i,j;
 glBegin(GL_QUADS);
 for(i=0;i<6;i++){
  glNormal3dv(normal[i]);
  for(j=0;j<4;j++){
```

```
   glVertex3dv(vertex[face[i][j]]);
  }
 }
 glEnd();
}
GLvoid CALLBACK none(void)
{
}
GLvoid CALLBACK draw(void)
{
 static int r=0;
 static GLfloat red[]={ 1.0,0.0,0.0,1.0};
 static GLfloat blue[]={ 0.0,0.0,1.0,1.0};
 glClearColor(0.0,0.0,0.0,0.0);
 glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
 glEnable(GL_DEPTH_TEST);
 glEnable(GL_LIGHTING);
 glEnable(GL_LIGHT0);
 glPushMatrix();
 glRotated((double)r,0.0,1.0,0.0);
 glMaterialfv(GL_FRONT_AND_BACK,GL_DIFFUSE,red);
 cube();
 glPushMatrix();
 glTranslated(1.0,1.0,1.0);
 glRotated((double)(2*r),0.0,1.0,0.0);
 glMaterialfv(GL_FRONT_AND_BACK,GL_DIFFUSE,blue);
 cube();
 glPopMatrix();
 glPopMatrix();
 glDisable(GL_LIGHT0);
 glDisable(GL_LIGHTING);
 glDisable(GL_DEPTH_TEST);
```

```
 auxSwapBuffers();
 if(++r>=360) r=0;
}
GLvoid CALLBACK resize(GLsizei w,GLsizei h)
{
 glMatrixMode(GL_PROJECTION);
 glLoadIdentity();
 glViewport(0,0,w,h);
 gluPerspective(30.0,1.0,1.0,10.0);
 glMatrixMode(GL_MODELVIEW);
 glLoadIdentity();
 gluLookAt(3.0,4.0,5.0,0.0,0.0,0.0,0.0,1.0,0.0);
}

int main(int argc, char *argv[])
{
 auxInitPosition(200,100,512,512);
 auxInitDisplayMode(AUX_RGBA|AUX_DOUBLE|AUX_DEPTH);
 auxInitWindow(argv[0]);
 auxReshapeFunc(resize);
 auxIdleFunc(draw);
 glCullFace(GL_BACK);
 auxMainLoop(none);
 return 0;
}
```

Đến đây thì không cần giải thích gì các bạn cũng đã hiểu chương trình này. Đây là vài gợi ý cho những bạn thích tìm tòi, các bạn có thể dùng các hàm vừa đề cập để vẽ một hệ mặt trời với các hành tinh, hay vẽ một cánh tay robốt cử động chẳng hạnNếu có điều kiện tôi sẽ còn tiếp tục viết tiếp những bài khác về opengl , có thể là thư viên glut vì nó rất mạnh.Hi vọng một ngày nào đó gặp lại. Nếu bài viết của tôi có gì sai thì vui lòng gửi cho tôi

lời nhắn, tôi sẽ sửa lại để nó thật đúng và thật hợp.Mời các bạn ghé qua trang web của tôi, trên đó có một số chương trình khá thú vị nếu bạn nào cần mã nguồn thì mail cho tôi, tôi sẽ sẵn lòng gửi cho.Thân ái.

Lời cuối: Cám ơn bạn đã theo dõi hết cuốn truyện.
Nguồn: http://vnthuquan.net
Phát hành: Nguyễn Kim Vỹ.

Nguồn: http://cct2.st.toba-cmt.ac.jp/~442/
Được bạn: mickey đưa lên
vào ngày: 18 tháng 8 năm 2004