



OpenGL là gì ?

Được phát triển đầu tiên bởi Silicon Graphic, Inc., là một giao diện phần mềm hỗ trợ theo chu trình công nghiệp hỗ trợ đồ họa 3 chiều. Cung cấp khoảng 120 tác vụ và các primitive trong nhiều mode khác nhau. Với OpenGL, bạn có thể tạo ra ảnh 3 chiều có tính và đáng với chi phí thấp.

Là một giao diện phần mềm độc lập với phần cứng (hardware – independent software interface) hỗ trợ cho lập trình đồ họa. Để làm được điều này, OpenGL không thực hiện các tác vụ thu thập dữ liệu hành cũng như không nhận dữ liệu nhập của người dùng (người dùng giao tiếp với OpenGL thông qua OpenGL API). Nó là lớp trung gian giữa người dùng và phần cứng. Nghĩa là nó giao tiếp trực tiếp với driver của thiết bị đồ họa.

GLUT (pronounced like the glut in gluttony) is the OpenGL Utility Toolkit, a window system independent toolkit for writing OpenGL programs. It implements a simple windowing application programming interface (API) for OpenGL. GLUT makes it considerably easier to learn about and explore OpenGL programming. GLUT provides a portable API so you can write a single OpenGL program that works across all PC and workstation OS platforms.

Download phiên bản mới nhất của glut ở [đây](#).

Giới thiệu và đặt các file vào đúng vị trí:

- glut32.dll vào C:WINDOWSsystem
- glut.lib vào C:Program FilesMicrosoft Visual Studio 9.0 VC lib
- glut.h vào C:Program FilesMicrosoft Visual Studio 9.0 VC Include GL

Tất nhiên GLUT là thư viện phụ thuộc OpenGL cho nên cần có gl.h, glu.h,glu32.dll, opengl32.dll, opengl32.lib, glu32.lib nữa. Tìm tại [đây](#) . Giải nén và đặt vào các vị trí ghi ng như trên.

Add dòng sau đây vào trước hàm main()

```
#pragma comment( linker, "/subsystem:""windows" /entry:""mainCRTStartup"" )
```

Xong, bây giờ tôi sẽ hướng dẫn bạn vẽ một hình tam giác với GLUT

```
#include <GL/glut.h>
#pragma comment( linker, "/subsystem:""windows" /entry:""mainCRTStartup"" )
void renderScene(void) {
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_TRIANGLES);
    glVertex3f(-0.5,-0.5,0.0);
    glVertex3f(0.5,0.0,0.0);
    glVertex3f(0.0,0.5,0.0);
    glEnd();
    glFlush();
} void main(int argc, char **argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DEPTH | GLUT_SINGLE | GLUT_RGBA);
    glutInitWindowPosition(100,100);
    glutInitWindowSize(320,320);
    glutCreateWindow("Vẽ hình tam giác!!!!");
    glutDisplayFunc(renderScene);
    glutMainLoop();
}
```

Một số quy ước (convention) về tên hàm OpenGL:

Tên hàm thư viện OpenGL có hình thức như sau:

Gl{tên hàm}({số tham số}{loại tham số})

Ví dụ : `glClearColor()`, `glColor3f()`

Phần tên hàm được viết hoa cho những cái đứng đầu 1 từ trong tên hàm và ít nhất 1 chữ cái đầu của mỗi chữ cái hàm. Phần số tham số và loại tham số xuất hiện tùy theo hàm. Số tham số cho ta biết số lượng tham số sẽ đưa vào khi gọi hàm.

Gợi ý thích các hàm:

`void glutInit(int *argc, char **argv);` → //Khi khởi động GLUT , argc, argv là 2 đối số dòng lệnh của hàm main

`void glutInitWindowPosition(int x, int y);` → //Khi khởi tạo vị trí bắt đầu của cửa sổ, x là left of the screen, y là top of the screen, nói chung đây là điểm bên trái, phía trên của cửa sổ, từ đây ta kéo xuống phía dưới, bên phải là được 1 cửa sổ. Đơn vị của x, y là pixel.

`void glutInitWindowSize(int width, int height);` → //Khi khởi tạo kích thước của cửa sổ. Với chiều dài và chiều rộng, chúng thêm 1 điểm bắt đầu của màn hình trên màn hình, bên đã tạo ra được 1 cái cửa sổ chưa

`void glutInitDisplayMode(unsigned int mode)` → //định nghĩa mode hiển thị, chọn ra màu của mode và số + kích thước của buffer

- + GLUT_RGBA or GLUT_RGB : cửa sổ màu RGBA, đây là mode mặc định
- + GLUT_SINGLE : cửa sổ buffer đơn
- + GLUT_DOUBLE : cửa sổ buffer đôi
- + GLUT_DEPTH : cửa sổ buffer sâu

int glutCreateWindow(char *title); → tạo cửa sổ có tiêu đề title

Bây giờ ta cần tìm hiểu hàm để trình diễn. Hàm này là do người dùng tạo ra.

```
void renderScene(void) {
    glClear(GL_COLOR_BUFFER_BIT); //xóa màn hình
    glBegin(GL_TRIANGLES); //vẽ tam giác
    glVertex3f(-0.5,-0.5,0.0);
    glVertex3f(0.5,0.0,0.0);
    glVertex3f(0.0,0.5,0.0);
    glEnd(); //kết thúc
    glFlush();
}
```

void glutDisplayFunc(void (*func)(void)); → Hàm này báo cho GLUT biết trình diễn theo hàm nào, để biết cửa sổ nó là một con trỏ hàm trả về kiểu void

void glutMainLoop(void) → cuối cùng ta phải gọi hàm main liên tục để “trình diễn hình tam giác”. Giống như người ta làm phim hoạt hình đó, các frame nối tiếp nhau trên màn hình.

Hic, mọi cái này dịch từ english sang, ko hiểu lắm, cần tìm hiểu kỹ đã.

THAM KHẢO THÊM

Các hàm `glClearColor()`, `glClear()`, `glFlush()` là những hàm cơ bản của `Opengl.glClearColor()` có nhiệm vụ xóa màu để xóa window, bản thân dòng lệnh này 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).

Các giá trị `R,G,B` trong OpenGL thì ≥ 0.0 (không có) và ≤ 1.0 (độ sáng của màu). Ba tham số đầu là màu xanh lá cây và xanh da trời, còn tham số thứ 4 là độ sáng của cửa sổ. Bây giờ hãy thay đổi các giá trị của màu xem thế nào! Hàm `glClear()` mà mình nhắc sẽ xóa window, nó có

những hàng số xác định. Có những hàng có những hàm chia để chia y định khi kết thúc chương trình, để tránh những hàng này hàm glFlush() để chia g, nó sẽ thực hiện tất cả các hàm chia để chia y và kết thúc chương trình.

Dùng glClear*() để định màu xóa cho các buffer.

Sau đó glClear(GLbitfield mask) để xóa buffer tương ứng với mask

Tham số param của hàm glClear() có thể nhận từ 1 đến 4 giá trị sau:

Buffer	Name
Color buffer	GL_COLOR_BUFFER_BIT
Depth buffer	GL_DEPTH_BUFFER_BIT
Accumulation buffer	GL_ACCUM_BUFFER_BIT
Stencil buffer	GL_STENCIL_BUFFER_BIT

Mỗi giá trị trên chia định một buffer tương ứng như trên bảng. Các giá trị có thể kết hợp thông qua toán tử '|' (bitwise-OR). Khi muốn xóa nhiều bộ định nghĩa thì, nên dùng bitwise-OR hơn là gọi tách riêng từng lệnh như glClear() một tham số vì hàm glClear() có thể thực sự xóa định nghĩa nhiều buffer (chức năng này tùy thuộc phiên bản).

Để định màu xóa cho mỗi buffer, ta dùng các hàm glClear*() như sau: glClearColor(), glClearDepth(), glClearAccum(), glClearStencil().

Buộc việc hoàn tất

Đi với các ứng dụng để chia qua mạng, trong đó client chia phần chương trình chính và hiện thì kết quả đến server, thì ứng dụng thì client sẽ gom nhiều lệnh vào một packet, sau đó gửi đi đến server. Nhưng làm sao để client biết được khi nào thì lệnh trên server đã xong và gửi tiếp packet khác? Do đó nó sẽ đợi đến khi nào packet đến mới gửi tiếp. Nhưng packet có thể không bao giờ đến vì việc bên client đã hoàn tất và nhận về server không thực sự nhận kết quả về OpenGL cung cấp hàm glFlush() để chúng ta gửi quy trình này. Lệnh này buộc client gửi packet ngay cả khi packet chia để chia y. Lệnh này không đợi cho việc hoàn tất, nó buộc việc phải bắt đầu thực hiện và do đó để bỏ tất cả các lệnh trước đó thực hiện trong một thời gian gửi hơn. Nếu máy chia local thì ta không cần dùng lệnh này.

Một lệnh khác cũng gửi gửi là lệnh glFinish(), nó thực hiện chức năng gửi gửi glFlush() nhưng

nó đi phiên hành trình phiên công đang về học công việc khác đi nh việc đã hoàn tất. Lành này học đúng khi ta muốn đi công bố hóa các phiên về. Ví dụ ta muốn một phần nào đó về hoàn tất xong thì mới tiếp tục hành các phiên khác. Nhưng làm công việc này sẽ làm chậm công việc vì nó phải đi phiên hành.

Cấu trúc vertex

Trong OpenGL, muốn đi trình hình học để mô tả các vertex. Cấu trúc này bao gồm bốn số thực chia để trong không gian. Để chia để nh một vertex, ta dùng:

```
glVertex{234}{sifd}[v](TYPEcoords);
```

Vì sao có số 4? Vì OpenGL dùng để chia để trình đi theo một số thực. Khi ta ghi để chia để trình (x,y,z,w) thì số thực là để chia để trình 3 chiều (x/w,y/w,z/w). Mặc để nh w=1.0, z=0.

Ví dụ:

```
glVertex2s(2, 3); // khai báo vertex có để chia để trình (2,3,0)
glVertex3d(0.0, 0.0, 3.1415926535898); // để chia để trình nh tham số
glVertex4f(2.3, 1.0, -2.2, 2.0); // để chia để trình (1.15,0.5,1.1)
GLdouble dvect[3] = {5.0, 9.0, 1992.0}; //khai báo 1 vectơ
glVertex3dv(dvect); //khai báo vertex thông qua vector nêu trên
```

Trong một số máy tính, việc truy cập tham số bằng vector sẽ hiệu quả hơn truy cập ba số riêng (tùy thuộc phiên công).

Để tạo một trình hình học các vertex, ta bao gồm khai báo vertex bằng hai hàm glBegin(param) và glEnd(). Tham số param để vào cho hàm glBegin() sẽ giúp OpenGL quyết định về gì các vertex khai báo bên trong.

Ví dụ:

```
glBegin(GL_POLYGON);  
glVertex2f(0.0, 0.0);  
glVertex2f(0.0, 1.0);  
glVertex2f(0.5, 1.0);  
glVertex2f(1.0, 0.5);  
glVertex2f(0.5, 0.0);  
glEnd();
```

Các tham số có thể đưa vào hàm glBegin(Glenum mode)

Giá trị

Ý nghĩa

- GL_POINTS** Tập hợp vertex được vẽ riêng
- GL_LINES** Mỗi cặp vertex được coi như 2 đầu mút đoạn thẳng
- GL_POLYGON** Các vertex được xem như biên của mặt đa giác (lồi)
- GL_TRIANGLES** Bộ 3 vertex được xem các đỉnh mặt tam giác (không lồi)
- GL_QUADS** Bộ 4 vertex được xem như 4 đỉnh 1 đa giác (không lồi).
- GL_LINE_STRIP** Một loạt các đoạn được nối với nhau (có lồi)
- GL_LINE_LOOP** Như trên nhưng vertex đầu và cuối được nối với nhau (lồi vòng)
- GL_TRIANGLE_STRIP** Bộ 3 vertex xem như các đỉnh tam giác (có lồi)
- GL_TRIANGLE_FAN** Vertex 0 là đỉnh chung kết hợp các đỉnh bên kia tạo tam giác

Sưu tập & chỉnh sửa