ĐẠI HỌC ĐÀ NẴNG

TRƯỜNG ĐẠI HỌC SƯ PHẠM

KHOA TIN HỌC



BÁO CÁO KẾT THÚC MÔN HỌC

ĐỒ HỌA MÁY TÍNH

NGÀNH CÔNG NGHỆ THÔNG TIN

ĐỀ TÀI:

**SNAKE – GAME RẮN SĂN MỒI**

GVHD : PHẠM ANH PHƯƠNG

SVTH : TRẦN LIÊM CHÍ

NGUYỄN HIỆP NHẤT

NGUYỄN THỊ THẢO LY

LỚP : 18CNTT2

ĐÀ NẴNG, 12/2019

**NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN**

………………………………………………………………………………………

………………………………………………………………………………………

………………………………………………………………………………………

………………………………………………………………………………………

………………………………………………………………………………………

………………………………………………………………………………………

………………………………………………………………………………………

………………………………………………………………………………………

………………………………………………………………………………………

………………………………………………………………………………………

………………………………………………………………………………………

………………………………………………………………………………………

………………………………………………………………………………………

………………………………………………………………………………………………………………………………………………………………………………

………………………………………………………………………………………………………………………………………………………………………………

*Đà nẵng, ngày tháng năm 2019*

*Giảng viên hướng dẫn*

*TS.Phạm Anh Phương*

MỤC LỤC

MỞ ĐẦU : GIỚI THIỆU VỀ OPENGL…………………………………………………1

CHƯƠNG 1: CÁCH SỬ DỤNG THƯ VIỆN OPENGL………………………..4

* 1. DOWLOAD…………………………………………………………….....4
  2. CÀI ĐẶT & CẤU HÌNH………………………………………………...7

CHƯƠNG 2: CHƯƠNG TRÌNH SỬ DỤNG ĐỒ HỌA OPENGL…………..14

2.1 CÁC LỆNH TRONG OPENGL………………………………………..14

2.2 TÔ MÀU TRONG OPENGL……………………………………..........17

2.3 LỆNH ĐIỀU KHIỂN……………………………………………………17

2.4 HÀM………………………………………………………………………19

2.5 SƠ ĐỒ CHÍNH CỦA CHƯƠNG TRÌNH………………………………21

2.6 MÔ TẢ CHƯƠNG TRÌNH…………………………...............................22

CHƯƠNG 3: MỘT SỐ THƯ VIỆN 2D,3D……………………………............23

3.1 THƯ VIỆN HỖ TRỢ OPENGL……………………………………….23

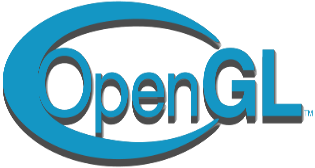
3.2 CÁC GIAO DIỆN LẬP TRÌNH ĐỒ HỌA KHÁC…………………….23

CHƯƠNG 4: PROJECT DEMO (SNAKE - GAME RẮN SĂN MỒI)………24

4. BÁO CÁO GAME……………………………………………………….24

MỞ ĐẦU

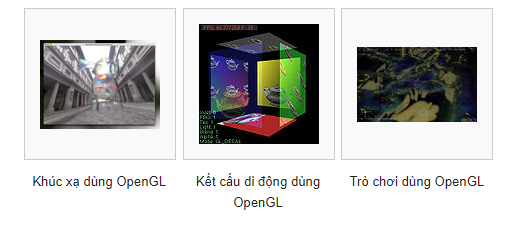
GIỚI THIỆU VỀ OPENGL

* 1. OpenGL là gì ?
* OpenGL (tiếng Anh Open Graphics Library) là một tiêu chuẩn kỹ thuật [đồ họa](https://vi.wikipedia.org/wiki/%C4%90%E1%BB%93_h%E1%BB%8Da) có mục đích định ra một [giao diện lập trình ứng dụng](https://vi.wikipedia.org/wiki/Giao_di%E1%BB%87n_l%E1%BA%ADp_tr%C3%ACnh_%E1%BB%A9ng_d%E1%BB%A5ng) ([tiếng Anh](https://vi.wikipedia.org/wiki/Ti%E1%BA%BFng_Anh): API) [đồ họa 3 chiều](https://vi.wikipedia.org/wiki/%C4%90%E1%BB%93_h%E1%BB%8Da_3_chi%E1%BB%81u). OpenGL cũng có thể được dùng trong các ứng dụng [đồ họa 2 chiều](https://vi.wikipedia.org/w/index.php?title=%C4%90%E1%BB%93_h%E1%BB%8Da_2_chi%E1%BB%81u&action=edit&redlink=1). Giao diện lập trình này chứa khoảng 250 [hàm](https://vi.wikipedia.org/wiki/H%C3%A0m) để vẽ các cảnh phức tạp từ những hàm đơn giản. Nó được dùng rộng rãi trong các [trò chơi điện tử](https://vi.wikipedia.org/wiki/Tr%C3%B2_ch%C6%A1i_%C4%91i%E1%BB%87n_t%E1%BB%AD). Ngoài ra nó còn dùng trong các ứng dụng [CAD](https://vi.wikipedia.org/wiki/CAD), [thực tế ảo](https://vi.wikipedia.org/wiki/Th%E1%BB%B1c_t%E1%BA%BF_%E1%BA%A3o), mô phỏng khoa học, mô phỏng thông tin, [phát triển trò chơi](https://vi.wikipedia.org/w/index.php?title=Ph%C3%A1t_tri%E1%BB%83n_tr%C3%B2_ch%C6%A1i&action=edit&redlink=1). OpenGL còn có một đối thủ cạnh tranh là [DirectX](https://vi.wikipedia.org/wiki/DirectX) của [Microsoft](https://vi.wikipedia.org/wiki/Microsoft).
* Không gian trong OpenGL được miêu tả qua [hình học xạ ảnh](https://vi.wikipedia.org/w/index.php?title=H%C3%ACnh_h%E1%BB%8Dc_x%E1%BA%A1_%E1%BA%A3nh&action=edit&redlink=1). Một điểm trong không gian này có tất cả bốn [tọa độ](https://vi.wikipedia.org/wiki/H%E1%BB%87_t%E1%BB%8Da_%C4%91%E1%BB%99). Cách thể hiện các điểm trong không gian bằng 4 tọa độ cho phép xử lý các điểm [vô tận](https://vi.wikipedia.org/wiki/V%C3%B4_t%E1%BA%ADn) một cách tổng quát. Vì vậy mã nguồn các ứng dụng đã được đơn giản hóa đi nhiều.

**0.2 Tác dụng của OpenGL**.

* OpenGL được thiết kế nhằm thỏa mãn mục đích chính sau:
* Che giấu sự tương tác phức tạp với các bộ máy xúc tiến 3 chiều bằng cách đưa ra một giao diện lập trình thống nhất.
* Che giấu các sự khác biệt giữa các phần cứng 3 chiều bằng cách bắt buộc các phần cứng tương thích OpenGL phải hỗ trợ tất cả các chức năng của giao diện OpenGL. Nếu cần, các chức năng chưa được hỗ trợ đầy đủ bởi phần cứng có thể được hỗ trợ bằng phần mềm.
* Các thao tác OpenGL cơ bản là nhận các nguyên hàm hình học như điểm, đường thẳng và đa giác rồi chuyển thành các điểm đồ họa (pixel) trên màn hình. Điều này được thực hiện bởi luồng ống dẫn đồ họa (graphics pipeline). Nó còn được gọi là bộ máy trạng thái OpenGL. Đa số các lệnh OpenGL được dùng để tạo ra các hình học cơ bản đã gặp ở trên hoặc là quy định cách chuyển đổi hình học trong bộ máy trạng thái OpenGL.
* Trước khi OpenGL 2.0 ra đời, mỗi giai đoạn trong luồng ống dẫn đồ họa thi hành một nhiệm vụ nhất định, khó có thể thay đổi được. Từ phiên bản OpenGL 2.0, một số giai đoạn đó có thể sửa đổi bằng cách dùng [ngôn ngữ chuyển màu GLSL](https://vi.wikipedia.org/w/index.php?title=GLSL&action=edit&redlink=1).

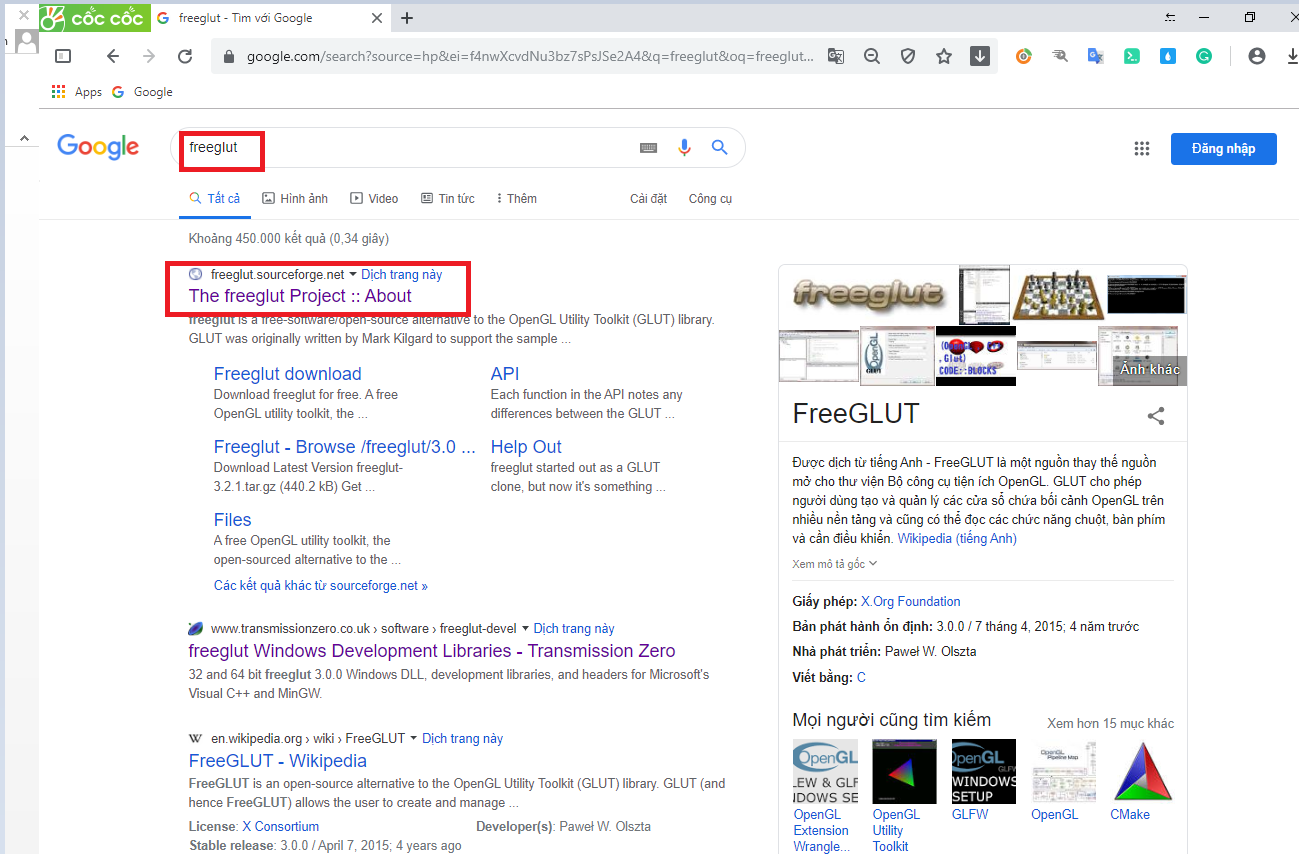
Hình ảnh minh họa :

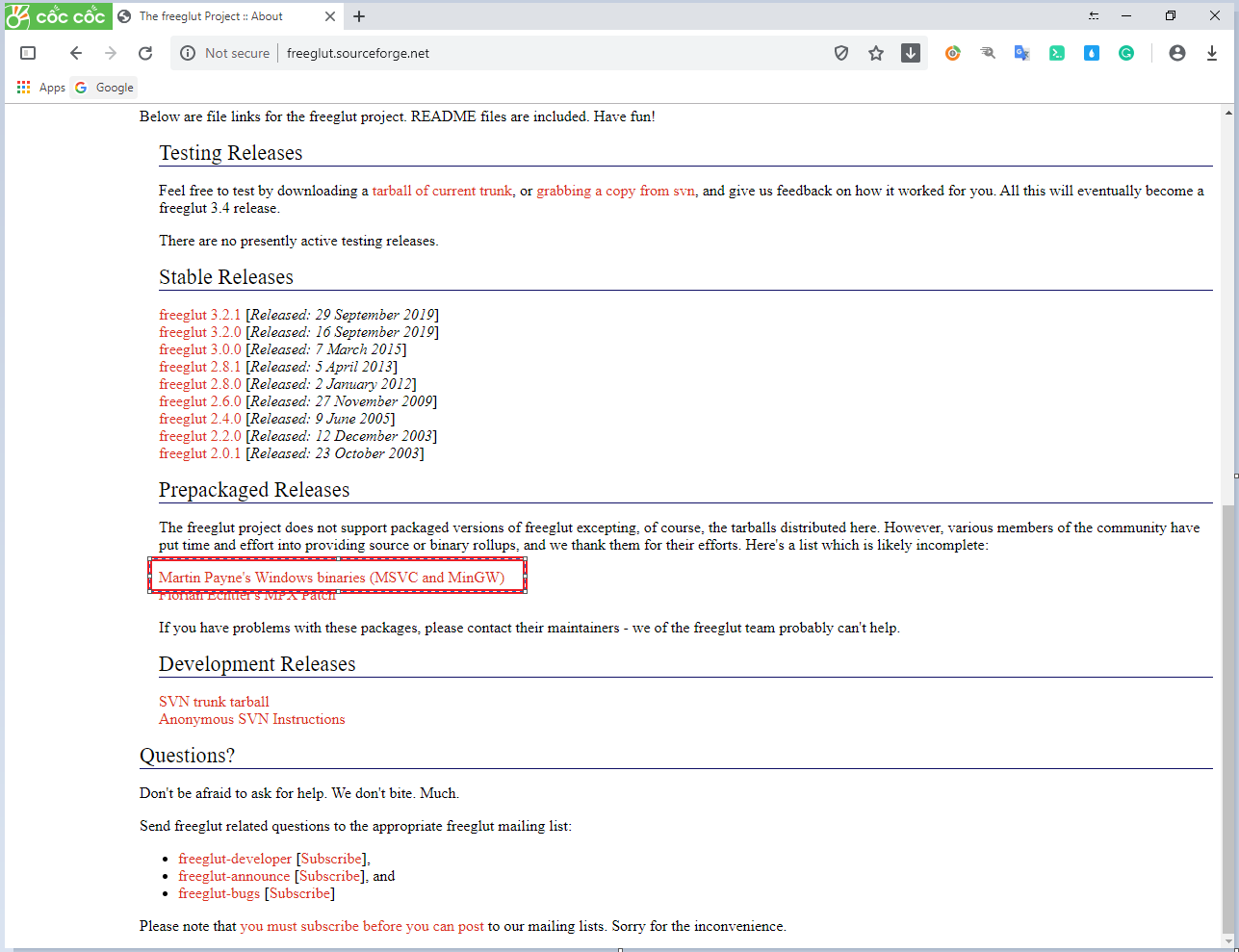


HƯỚNG DẪN DOWNLOAD, CÀI ĐẶT & CẤU HÌNH

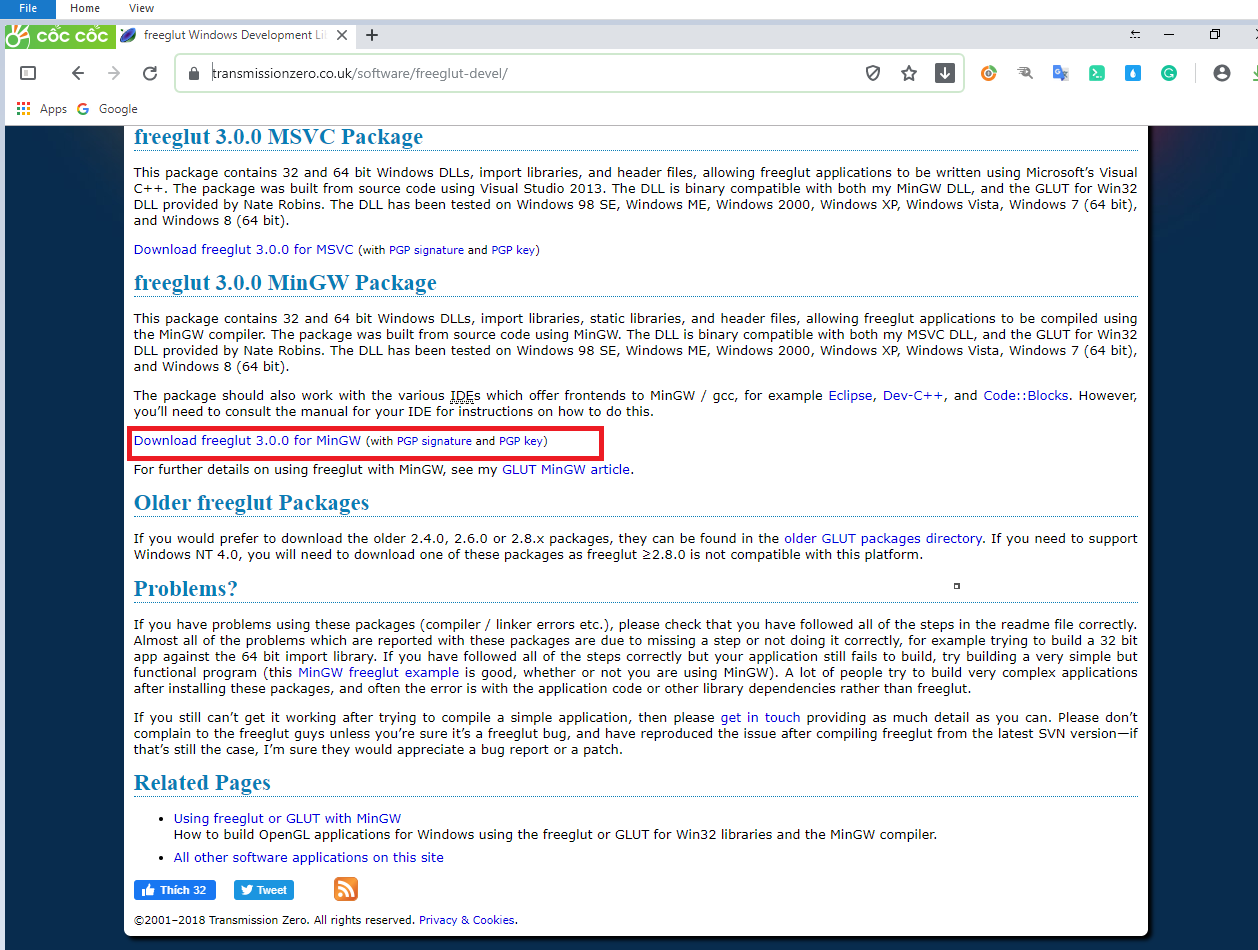
* 1. Download.

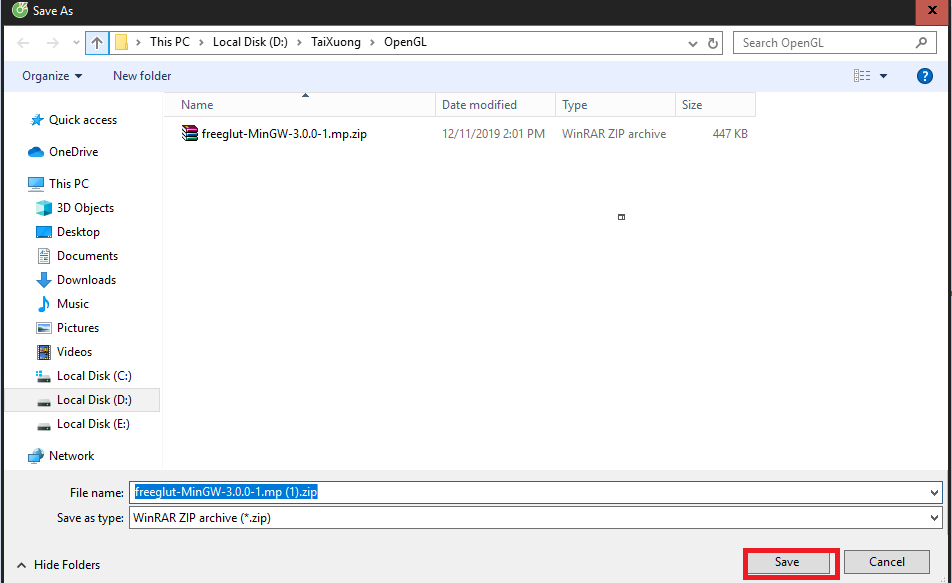
Cách 1 :

* Đầu tiên bạn mở trình duyệt, sau đó bạn search từ khóa “freeglut”. Trình duyệt sẽ dẫn bạn đến trang chủ. Dưới đây là hình ảnh minh họa

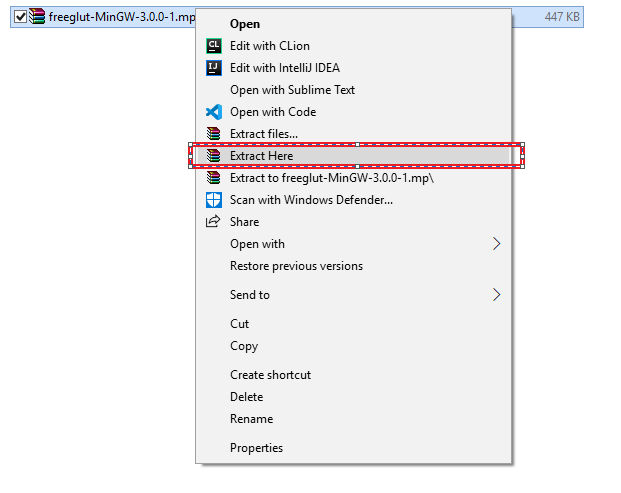


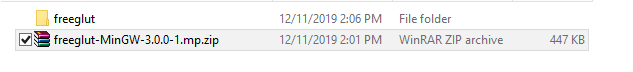
* Tiếp theo bạn cần chú ý để tránh nhầm lẫn giữa freeglut 3.0.0 MSVC Package với freeglut 3.0.0 MinGW Package. Màu đỏ là màu bạn chọn và tải về.





* Cuối cùng bạn đến thư mục đã lưu và giải nén. Giả nén bằng các kích chuột phải vào file nén. Dưới đây là hình ảnh minh họa.



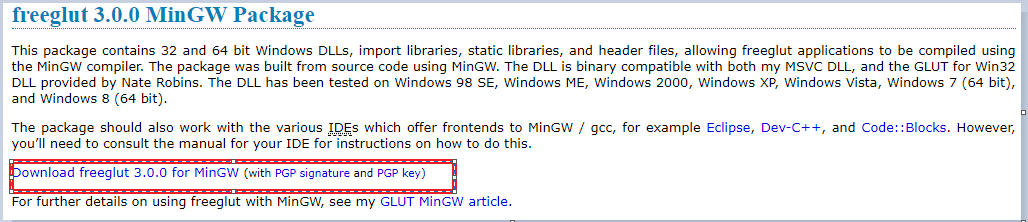


Như vậy bạn đã download xong.

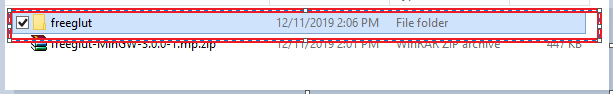
Cách 2 :

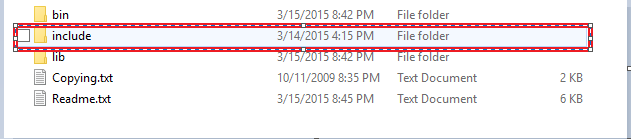
* Bạn copy đường dẫn (link) này dán vào trình duyệt sau đó bấm nút enter. Trình duyệt sẽ đưa bạn đến trang trủ để tải về.

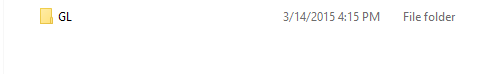
<https://www.transmissionzero.co.uk/software/freeglut-devel/>

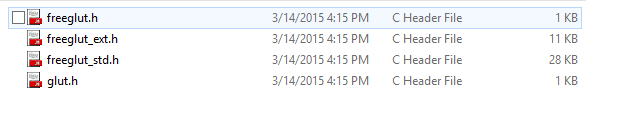


* Các bước tiếp theo tương tự như trên bạn có thể làm theo.
  1. Cài đặt & cấu hình.
  2. Cài đặt.
     1. Copy và dán vào thư mục GL.
* Vào thư mục tải và giải nén ở trên → include → GL → kích đúp để vào bên trong thư mục GL. Dưới đây là hình ảnh minh họa.

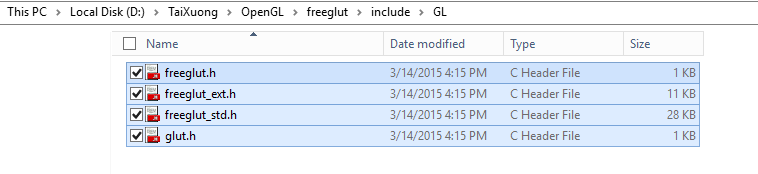






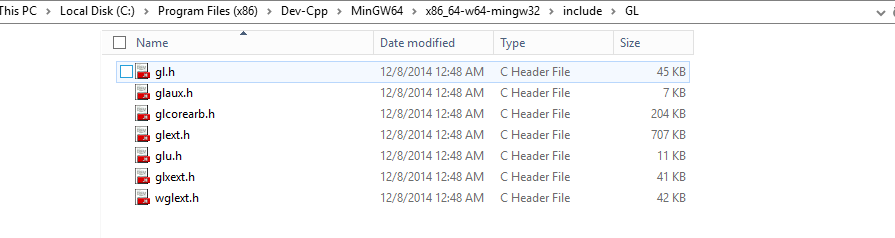


* Tiếp theo bạn copy tất cả các file trong thư mục **GL** ở file bạn bừa tải xuống và đã giải nén ở trên, dán vào thư mục **GL** được lưu ở C.

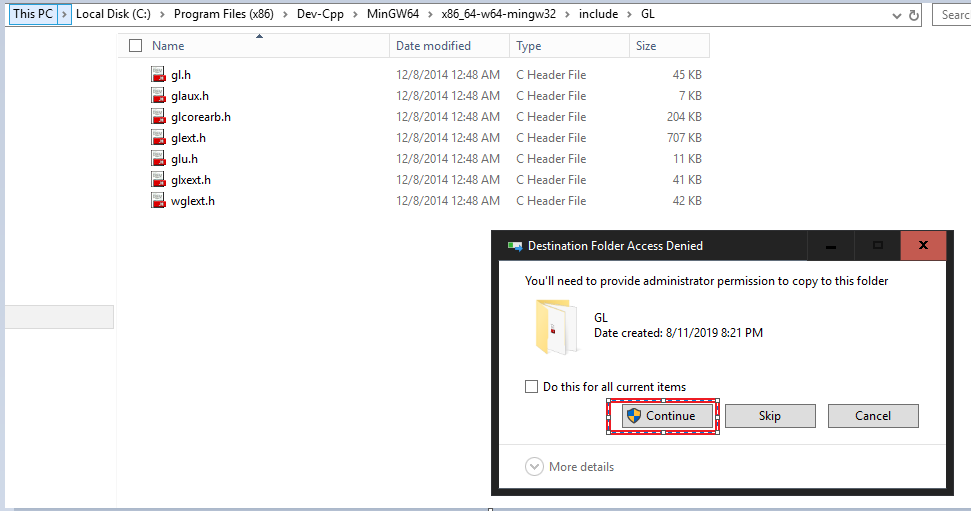


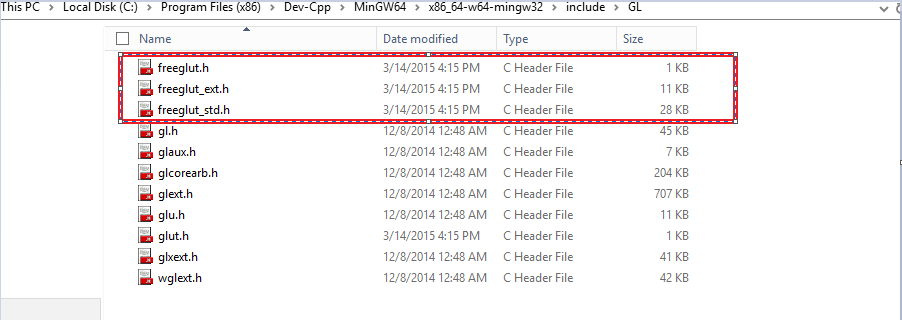
Đây là hình ảnh đã copy từ file GL freeglut 3.0.0 MinGW Package đã được tải xuống và giải nén.

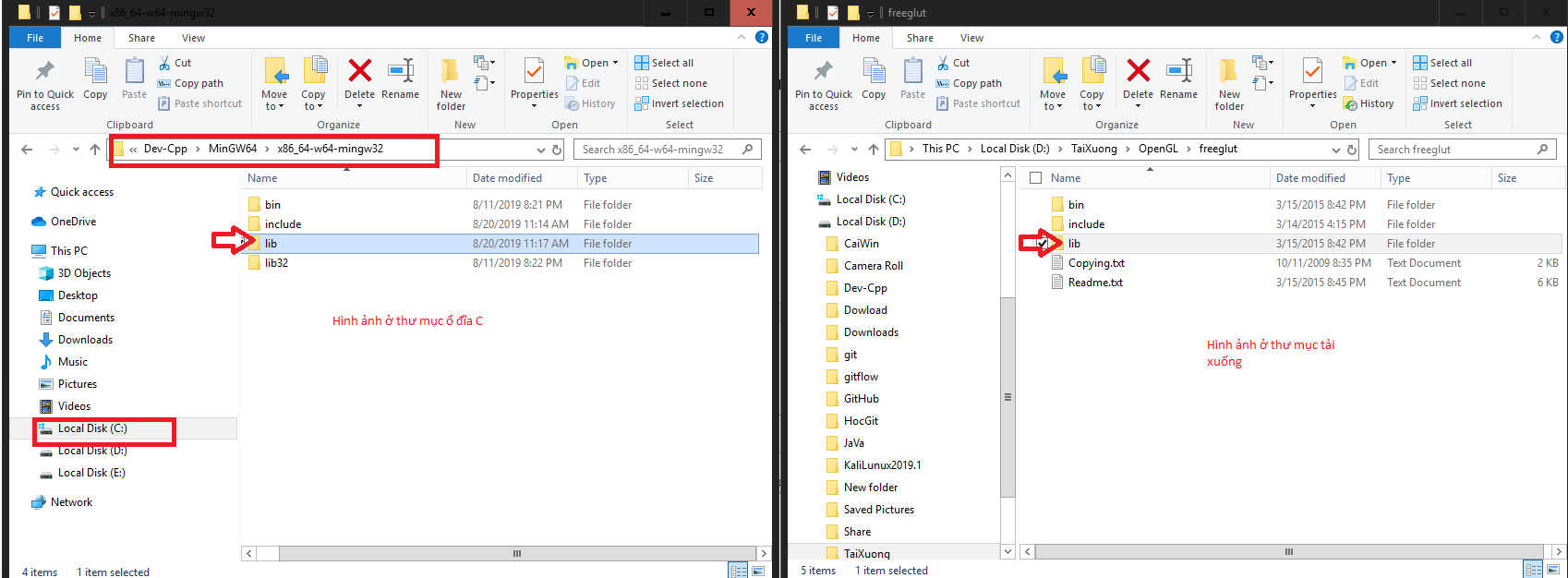
* Ổ C → Program Files(x86) → Dev-Cpp → MinGW64 → x86\_64-w64-mingw32 → include → GL (dán tất cả các file vào ).

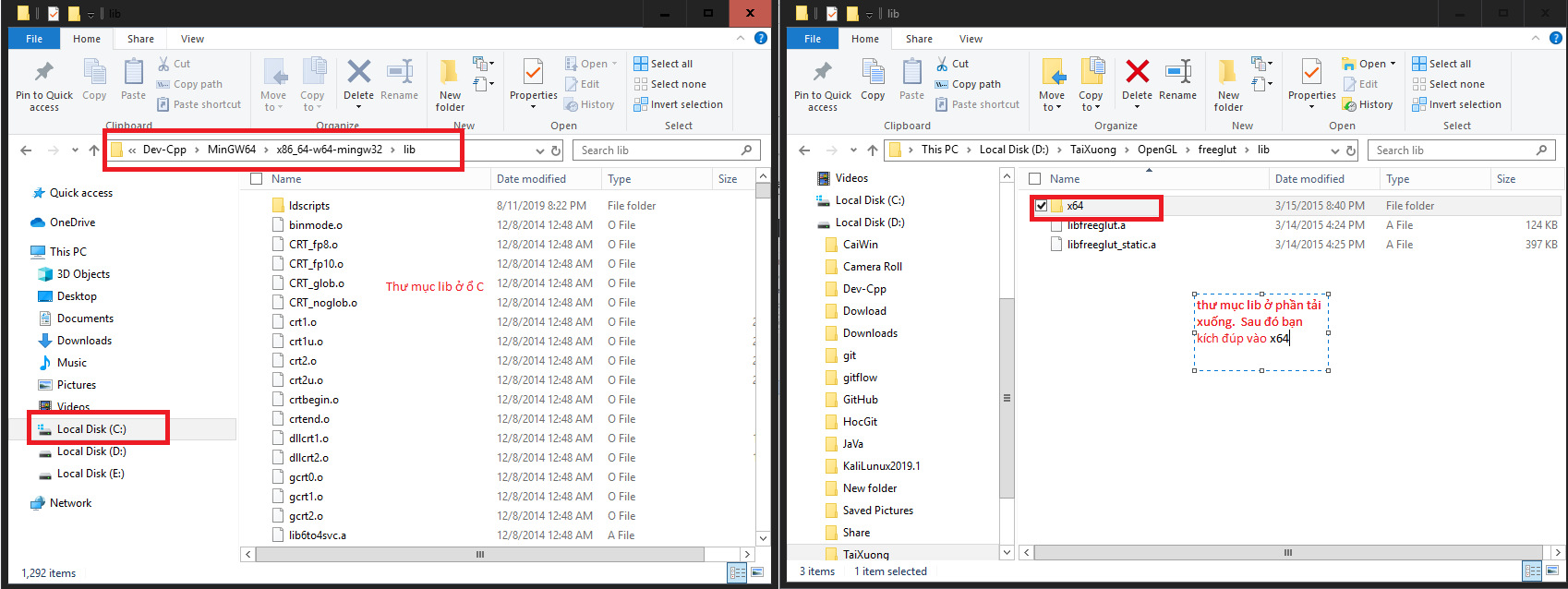


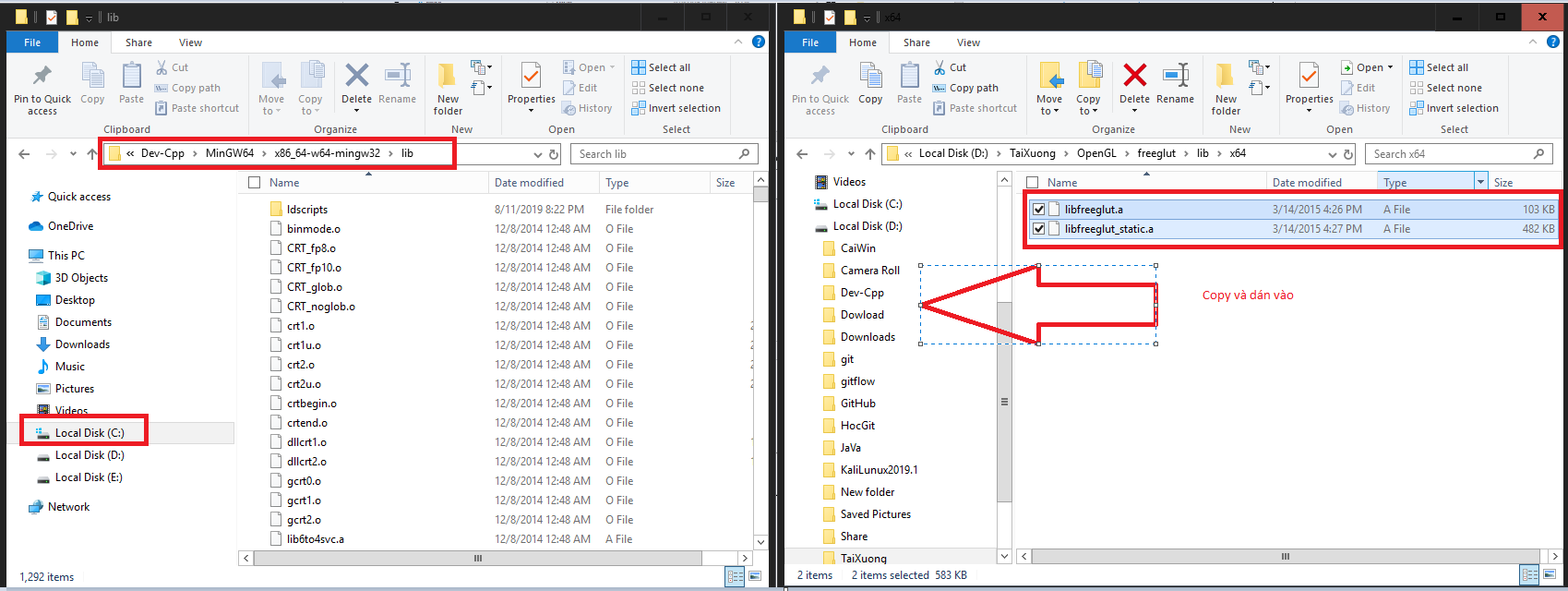
Hình ảnh file chưa được dán.

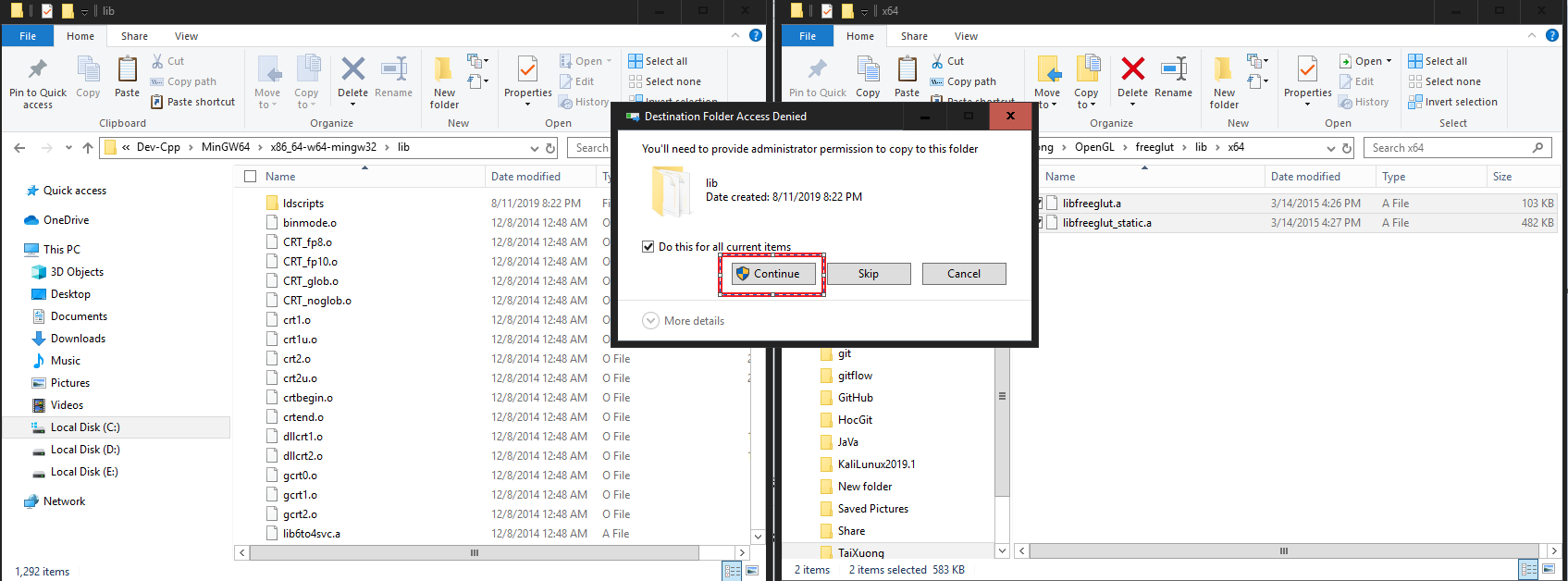




* + 1. Copy và dán vào thư mục lib.
* Tiếp đến các file tiếp theo bạn làm tương tự. Dưới đây là hình ảnh minh họa song song bạn hãy chú ý đường dẫn.
* 

* Bạn kích đúp vào lib của 2 thư mục
* Sau khi kích đúp vào ở thư mục tải xuống bạn copy và dán vào thư mục lib ở ổ đĩa C.

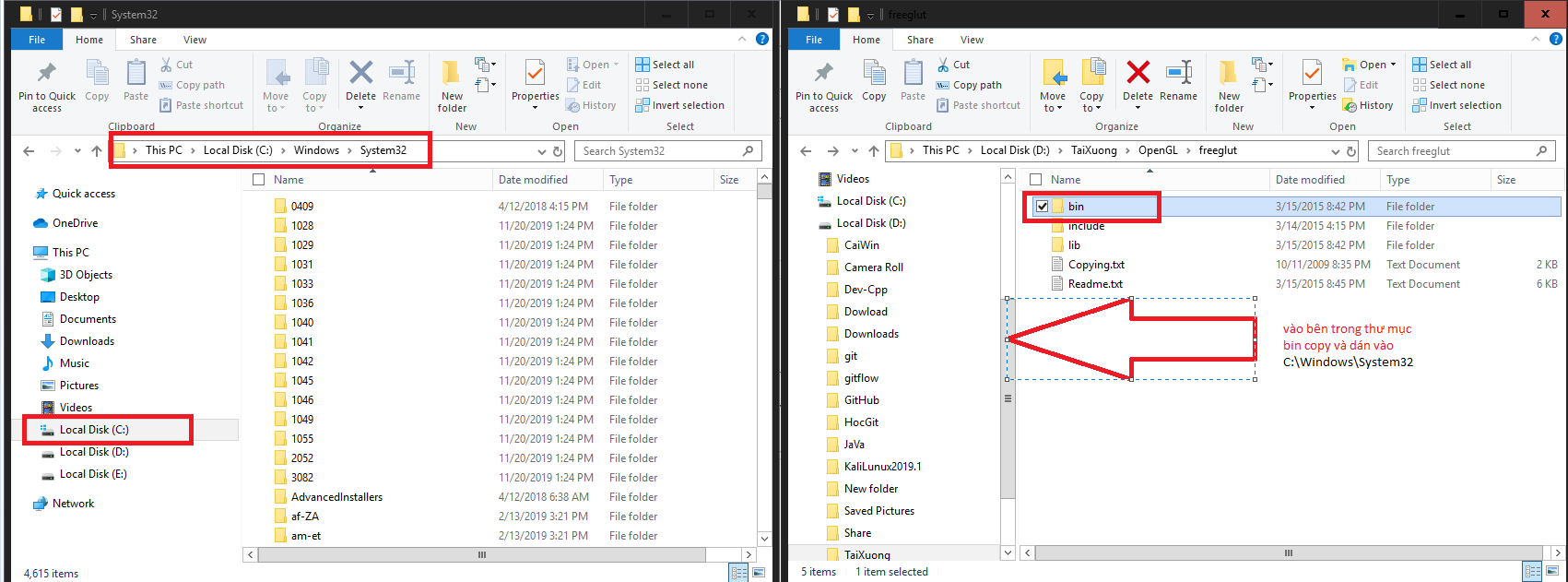


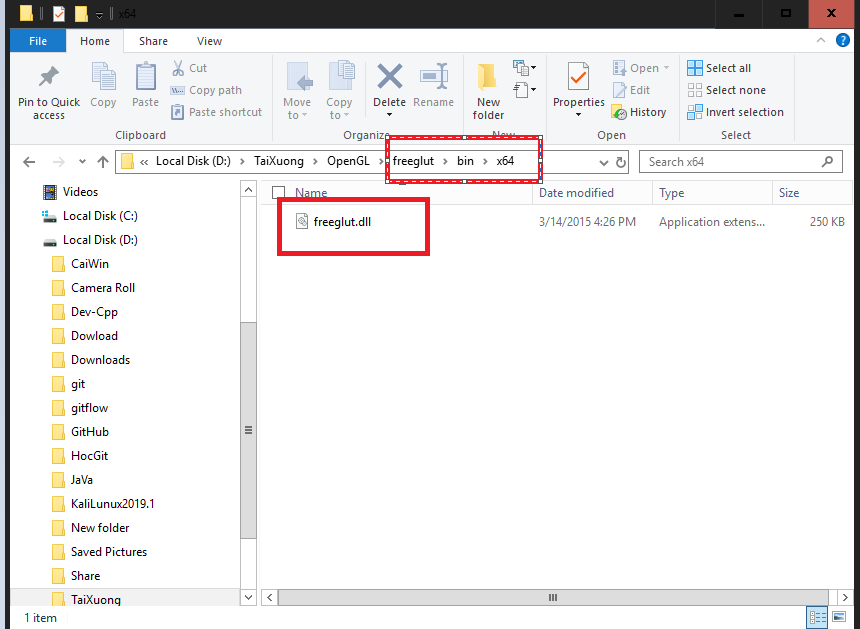


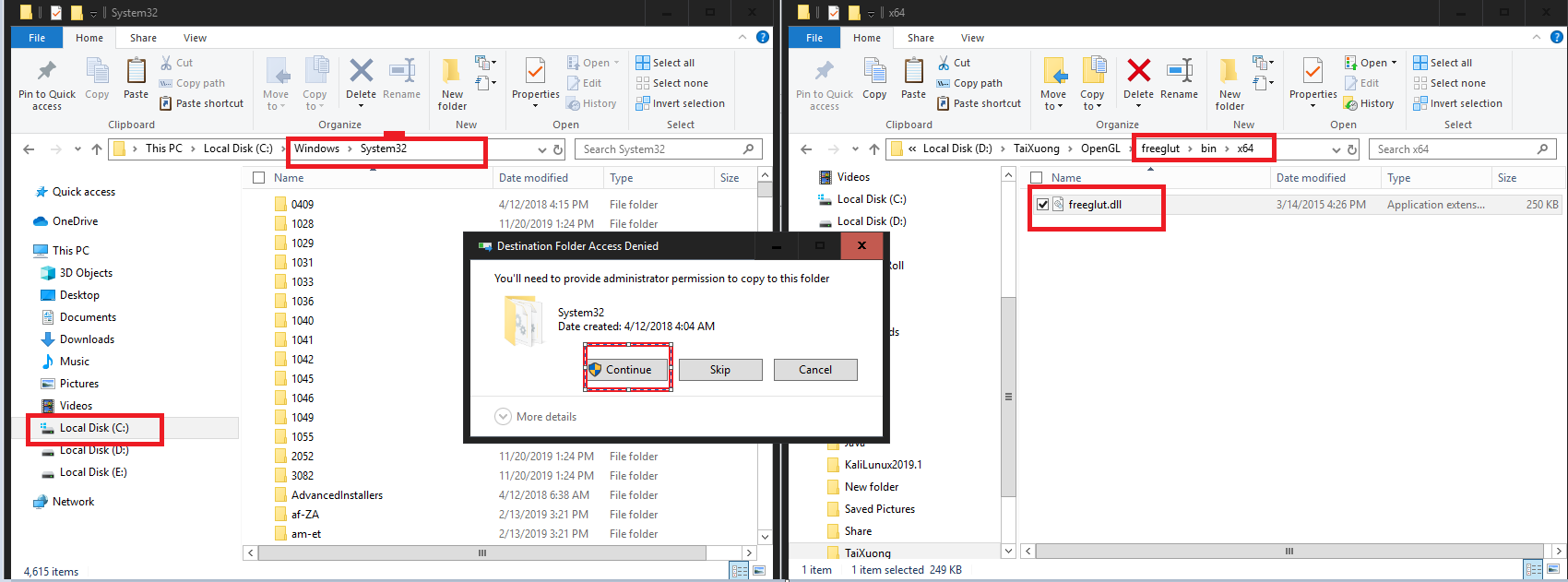
Như vậy bạn đã hoàn thành ở thư mục lib.

1.1.3 Copy và dán thư mục bin vào thư mục System32.

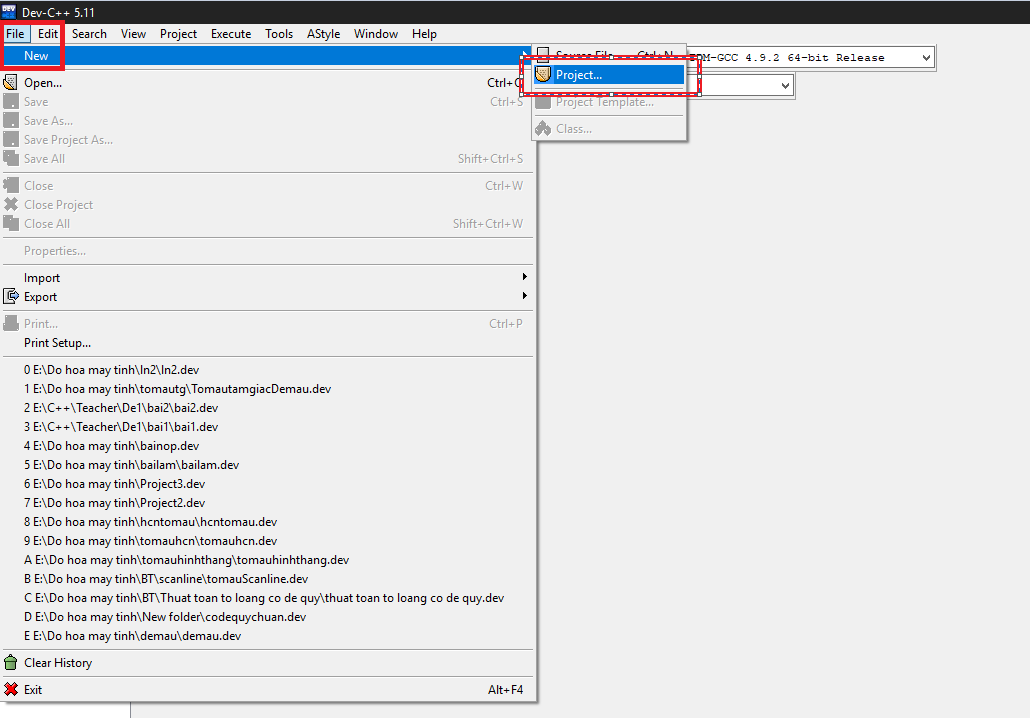
* Bạn copy ở thư mục bin và dán vào **C:\Windows\System32.**



* Vào bên trong **bin** kích đúp **x64** → copy và dán vào **C:\Windows\System32.**



* 1. Cấu hình.
* Bạn mở phần mềm DEV C++ → File → New → Project → Console Application → OK.



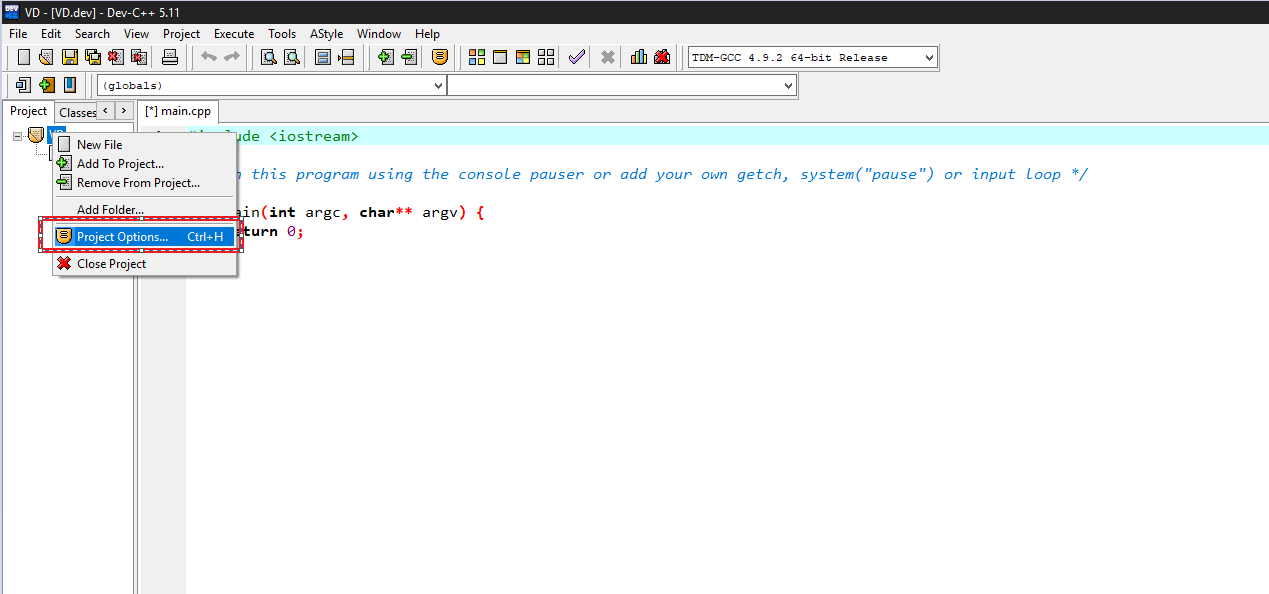
* Tiếp theo vào **Project Option** bằng cách nhấn tổ hợp phím **Ctrl + H**. Chọn tab **Parameters** và trong mục **Linker** điền đoạn lệnh sau:

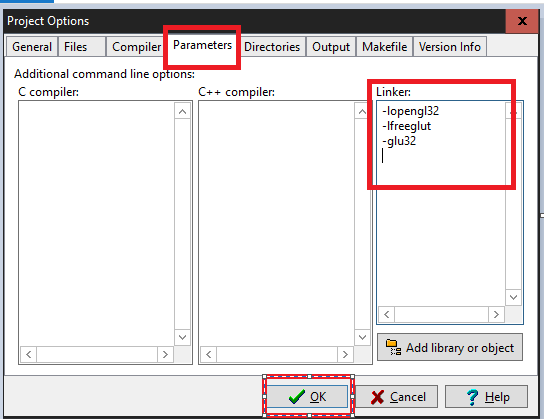
**-lopengl32**

**-lfreeglut**

**-glu32**

* nhấn **OK**





Như vậy bạn đã cấu hình thành công.

Chương 2 : CHƯƠNG TRÌNH SỬ DỤNG ĐỒ HỌA OPENGL

2.1. Các lệnh vẽ trong OpenGL.

* Một số ví dụ đơn giản

#include <gl\glut.h>

/\* hàm thực hiện các thao tác vẽ theo yêu cầu của chương trình \*/

void display(void) {

/\* xóa mọi pixel \*/

glClear (GL\_COLOR\_BUFFER\_BIT);

/\* vẽ hình chữ nhật có điểm trái-trên và phải-dưới \*/

/\* (0.25, 0.25, 0.0) and (0.75, 0.75, 0.0) \*/

glColor3f (1.0, 1.0, 1.0); /\* thiết lập màu vẽ: màu trắng \*/

glBegin(GL\_POLYGON); /\* bắt đầu vẽ đa giác \*/

glVertex3f (0.25, 0.25, 0.0); /\* xác định các đỉnh của đa giác \*/

glVertex3f (0.75, 0.25, 0.0);

glVertex3f (0.75, 0.75, 0.0);

glVertex3f (0.25, 0.75, 0.0);

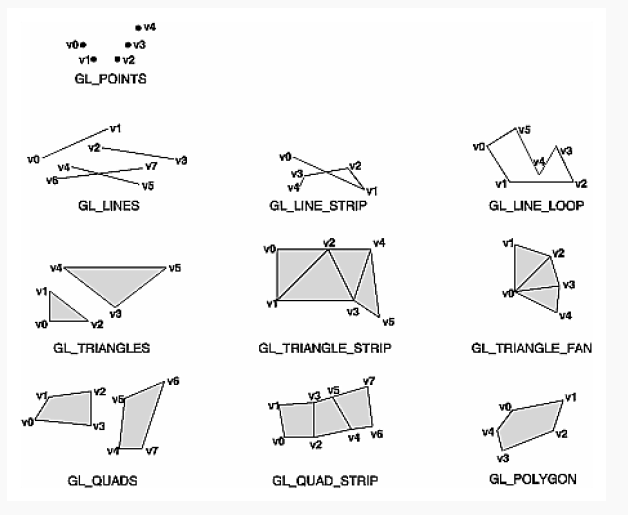
glEnd(); /\* kết thúc vẽ đa giác \*/

/\* thực hiện quá trình đẩy ra buffer \*/

glFlush ();

}

* Tất cả các hình khối trong openGL đều được nằm giữa hai dòng lệnh glBegin() và glEnd().
* Có thể có nhiều cặp dòng lệnh như vậy, tức là ta có thể viết các hàm vẽ khác nhau và dùng cặp 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, tức là điểm đầu trùng với điểm cuối.
* Một số hàm vẽ như sau:
* *GL\_POINT    Vẽ điểm.*
* *GL\_LINE    Vẽ đường thẳng.*
* *GL\_LINE\_STRIP    Tập hợp của những đoạn thẳng được nối với nhau.*
* *GL\_LINE\_LOOP    Đường gấp khúp khép kín.*
* *GL\_TRIANGLES    Vẽ hình tam giác.*
* *GL\_QUADS    Vẽ hình tứ giác.*
* *GL\_TRIANGLES\_STRIP    Vẽ tập hợp các tam giác đều liền nhau,chung cạnh.*
* *GL\_QUAD\_STRIP    Vẽ tập hợp các tứ giác liền nhau,chung một cạnh.*
* *GL\_TRIANGLE\_FAN    Vẽ hình quạt.*



* Ảnh minh họa cho các câu lệnh vẽ được sử dụng trong thư viện openGL  
  Hàm glVertex2d xác định điểm hai chiều, chúng ta 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 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.

***2.2.* Màu trong OpenGL.**

* Hàm thiết lập màu cho hình vẽ : glColor3f(x,x,x)
* Một số màu cơ bản:
* glColor3f(0.0,0.0,0.0); //màu đen
* glColor3f(1.0,1.0,1.0); //màu trắng
* glColor3f(1.0,0.0,0.0); //màu đỏ
* glColor3f(1.0,0.0,1.0); //màu tím
* glColor3f(1.0,1.0,0.0); //màu vàng
* glColor3f(0.0,1.0,0.0); //màu lục
* glColor3f(0.0,1.0,1.0); //màu tím xanh
* glColor3f(0.0,0.0,1.0); //màu xanh lơ

***2.3.* Câu lệnh điều khiển.**

* Điều khiển chuột :

void mouse(int button, int state, int x, int y)

{

switch (button)

{

case GLUT\_LEFT\_BUTTON:  /\* khi nhấn chuột trái \*/

if (state == GLUT\_DOWN)

glutIdleFunc(spinDisplay);

/\* khi chương trình đang trong trạng thái idle (không phải xử lý gì cả) thì sẽ thực hiện hàm spinDisplay \*/

break;

case GLUT\_MIDDLE\_BUTTON:  /\* khi nhấn nút giữa \*/

if (state == GLUT\_DOWN)

glutIdleFunc(NULL);

break;

default:

break;

}

}

* Hàm xử lý từ bàn phím với các kí tự nhận biết được dưới mã ASCII:  
  void glutKeyboardFunc(void (\*func) (unsigned char key, int x, int y))  
  Tham số (\*func)(unsigned char key, int x, int y) – con trỏ tới hàm xử lý
* *x – tọa độ x của chuột (hoành độ).*
* *y – tọa độ y của chuột (tung độ).*
* Hàm xử lý các kí tự trên bàn phím nhận biết được dưới mã ASCII :  
  void glutSpecialFunc (void (\*func) (int key, int x, int y)).
* Tham số tương tự glutKeyboardFunc().
* Những key được định nghĩa sẵn là một trong số mã sau:
* *GLUT\_KEY\_F1 F1*
* *GLUT\_KEY\_F2 F*

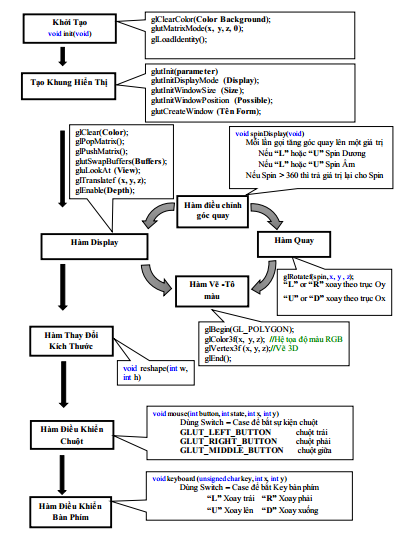
*……………………*

* *GLUT\_KEY\_F12 F12*
* *GLUT\_KEY\_LEFT //phím chức năng trái.*
* *GLUT\_KEY\_RIGHT //phím chức năng phải.*
* *GLUT\_KEY\_UP //phím chức năng trên.*
* *GLUT\_KEY\_DOWN //phím chức năng dưới.*
* *GLUT\_KEY\_PAGE\_UP //phím chức năng Page Up.*
* *GLUT\_KEY\_PAGE\_DOWN //phím chức năng Page Down.*
* *GLUT\_KEY\_HOME //phím chức năng Home.*
* *GLUT\_KEY\_END //phím chức năng End.*
* *GLUT\_KEY\_INSERT //phím chức năng Insert.*

***2.4.* Hàm trong OpenGL.**

* **glutInit**(int\* pargc,char \*\*argv): Khởi tạo thông số cho cửa sổ của glut. Các tham số pargc và argv được lấy từ hàm main().
* Danh sách các thiết lập:
* GLUT\_RGBA: thiết lâp cho chế độ màn hình RGBA.
* GLUT\_RGB: tương tự GLUT\_RGBA.
* GLUT\_INDEX: thiết lập chế độ màu mặc định.
* GLUT\_SINGLE: thiết lập chế độ màn hình buffer đơn.
* GLUT\_DOUBLE: thiết lập chế độ màn hình buffer đôi.
* GLUT\_ACCUM: thiết lập chế độ accumulation buffer.
* GLUT\_ALPHA: thiết lập chế độ đối tương anpha cho buffer màu.
* GLUT\_DEPTH: thiết lập cho depth buffer.
* GLUT\_STENCIL: thiết lập cho stencil buffer.
* GLUT\_MULTISAMPLE: thiết lập cho multisampling
* GLUT\_STEREO: thiết lập chế độ stereo.
* GLUT\_LUMINANCE: thiết lập cho chế độ màu "luminance".
* Sử dụng toán tử OR để kết hợp các chế độ màu, buffer với nhau.
* **glutInitWindowPosition**(int x, int y): tọa độ cửa sổ màn hình.
* **glutInitWindowSize**(int x, int y): thiết lập kích thước cửa sổ màn hình.
* **glutCreateWindow**(char\* title): tạo cửa sổ màn hình với tiêu đề là title.
* **glClearColor(**Glclampf red, Glclampf green, Glclampf blue, Glclampf alpha**):**thiết lập chế độ màu mới cho toàn bộ ứng dụng.
* **glClear**(int mode): GL\_COLOR\_BUFFER\_BIT: thông số để thiết lập cho màu sắc. Còn một số thiết lập nữa sẽ được đề cập sau.
* **glOrtho(...):**thiết lập tầm nhìn trực giao, sẽ được nói chi tiết trong bài về phép biến trongOpenGL.
* **glutDisplayFunc(void (\*f)(void)):** gọi hàm để thực hiện việc vẽ khi cửa sổ màn hình hiển thị. Trong ví dụ này là gọi hàm display.
* **glutMainLoop():** Lặp đi lặp lại các hàm callback - sẽ đề cập sau - đến khi nao cửa sổ đóng lại.
* Trong hàm display :
* **glColor3f**( red, green, blue ): thiết lập màu sắc cho đối tượng sắp vẽ.
* **glPointSize**(float size): kích thước điểm ảnh.
* **glBegin(int mode):**gồm các thông.

***2.5.* Sơ đồ hính của chương trình.**

[](https://4.bp.blogspot.com/-LqA9dHmu7mw/VRzPqTBToSI/AAAAAAAAC0I/zvF-XMTxnVg/s1600/2.PNG)

***2.6.* Mô tả chương trình.**

* Bước 1: Khởi tạo
* Tạo khung nhìn Form gồm : Chế độ nhập, Chế độ nhìn, Chế độ màu, Form size, Vị trí Form , Tiêu đề Form.
* Khởi tạo Chế độ nhập Init().
* Bước 2: Chức năng xử lý
* Dùng hàm Display() để xử lý vẽ hình, tô màu, quay hình.
* Khởi tạo màu nền và chế độ buffers.
* Xử lý xoay hình theo bàn phím key = { “L”, “R”, “U”, “D”}
* Gọi hàm SpinDisplay() thực hiện quay. Tăng góc quay sau mỗi lần thực hiện cho đến khi góc quay Spin lớn hơn 360 độ thì trả lại giá trị ban đầu cho Spin.
* Xử lý góc nhìn và độ sau góc nhìn.
* Xử lý vẽ Đa giác theo từng cạnh: Tô màu và vẽ theo trục không gian 3D.
* Thực hiện hoán đổi Bufers.
* Dùng hàm Reshape() để xử lý thay đổi kích thước khung Form.
* Dùng hàm Keyboard() để xử lý bàn phím máy tính.
* Dùng hàm Mouse() để xử lý chuột máy tính.
* Trong trường hợp nhấn chuột trái thực hiện hàm quay trái.
* Trong trường hợp nhấn chuột phải thực hiện hàm quay phải.
* Trong trường hợp nhấn giữa không thực hiện.
* Dùng hàm MainLoop() để dừng màn hình hiển thị.

Chương 3 : MỘT SỐ THƯ VIỆN 2D, 3D.

***3.1.* Thư viện hỗ trợ OpenGL.**

3.1.1. GLUT - Thư viện công cụ cho OpenGL.

* Glut là một thư viện các tiện ích cho các chương trình OpenGL, chủ yếu thực hiện I / O cấp hệ thống với hệ điều hành máy chủ. Các chức năng được thực hiện bao gồm định nghĩa cửa sổ, điều khiển cửa sổ và giám sát đầu vào bàn phím và chuột. ... GLUT được viết bởi MarJ.

3.1.2. GLU - Thư viện hàm cho các ứng dụng OpenGL.

* GLU là Thư viện tiện ích OpenGL. Đây là một tập hợp các hàm để tạo ra các mipmap kết cấu từ một hình ảnh cơ sở, tọa độ bản đồ giữa không gian màn hình và đối tượng và vẽ các bề mặt tứ giác và NURBS. GLU 1.2 là phiên bản của GLU đi kèm với OpenGL 1.1.  
  GLU 1.3 khả dụng và bao gồm các khả năng mới tương ứng với các tính năng OpenGL 1.2 mới.

3.1.3. ASSIMP - Thư viện nhập dữ liệu từ các tập tin đồ họa 3D.

* Open Asset Import Library (viết tắt là : Assimp)  là thư viện mã nguồn mở di động để nhập các định dạng mô hình 3D nổi tiếng khác nhau một cách thống nhất. Phiên bản gần đây nhất cũng biết cách xuất tệp 3d và do đó phù hợp làm công cụ chuyển đổi mô hình 3D đa năng.

***3.2.* Các giao diện lập trình đồ họa khác.**

3.2.1. Mesa 3D - Một thư viện thực thi OpenGL mã nguồn mở.

* Mesa , còn được gọi là Mesa3D và Thư viện đồ họa 3D Mesa , là một triển khai phần mềm nguồn mở của OpenGL, Vulkan và các đặc tả API đồ họa khác. ... Trình điều khiển đồ họa độc quyền (ví dụ: trình điều khiển Nvidia GeForce và Catalyst) thay thế tất cả Mesa , cung cấp việc triển khai API đồ họa của riêng họ.

3.2.2. Direct3D - Thư viện đồ họa của Microsoft.

* Direct3D là giao diện lập trình ứng dụng đồ họa (API) cho Microsoft Windows. Một phần của DirectX , Direct3D được sử dụng để hiển thị đồ họa ba chiều trong các ứng dụng có hiệu suất quan trọng, chẳng hạn như các trò chơi.

3.2.2. Light Weight Java Game Library

* Lightweight Java trò chơi Thư viện (LWJGL) là một mã nguồn mở Java phần mềm thư viện dành cho video trò chơi phát triển. Nó cho thấy các thư viện đa nền tảng hiệu suất cao thường được sử dụng trong việc phát triển các trò chơi video và các tựa game đa phương tiện, như Vulkan, OpenGL, OpenAL và OpenCL.

3.2.3. VirtualGL.

* VirtualGL là một chương trình mã nguồn mở chuyển hướng các lệnh kết xuất 3D từ các ứng dụng OpenGL của Unix và Linux sang phần cứng máy gia tốc 3D trong một máy chủ chuyên dụng và hiển thị đầu ra được kết xuất tương tác với một máy khách mỏng nằm ở nơi khác trên mạng.

**Chương 4 : PROJECT DEMO (SNAKE – GAME RẮN**

**SĂN MỒI).**

***4.* Báo cáo game.**

**#include <ctime>**

**#include <deque>**

**#include <GL/glut.h>**

**#include <unistd.h>**

**// mot macro cho cac bien khong duoc su dung (de bo qua cac canh bao G++)**

**#define UNUSED(param) (void)(param)**

**// macro huong ran**

**#define UP 1**

**#define DOWN 2**

**#define LEFT 3**

**#define RIGHT 4**

**char title[] = "SNAKE - GAME RAN SAN MOI";**

**float map\_half\_length = 30.0f;**

**int direction = DOWN;**

**int move\_speed = 100;**

**bool moved = false;**

**std::deque< std::deque<float> > part\_coords;**

**bool food\_available = false;**

**int food\_coords[2];**

**int growth\_stage = 0;**

**int growth = 2;**

**void spawnFood(){**

**if(!food\_available){**

**while(true){**

**bool collides = false;**

**// toa do ngau nhien tam thoi**

**int temp\_food\_coords[2] = { food\_coords[0] = 2 \* (rand() % ((int) map\_half\_length + 1)) - (int) map\_half\_length,**

**food\_coords[1] = 2 \* (rand() % ((int) map\_half\_length + 1)) - (int) map\_half\_length };**

**// co va cham voi ran khong?**

**for(unsigned int a = 0; a < part\_coords.size(); a++){**

**if(temp\_food\_coords[0] == part\_coords[a][0] &&**

**temp\_food\_coords[1] == part\_coords[a][1]){**

**collides = true;**

**}**

**}**

**// neu no khong va cham voi con ran, thi hay bien no thanh thanh toa do thuc an**

**if(collides == false){**

**food\_coords[0] = temp\_food\_coords[0];**

**food\_coords[1] = temp\_food\_coords[1];**

**food\_available = true;**

**break;**

**}**

**}**

**}**

**glLoadIdentity();**

**glTranslatef(food\_coords[0], food\_coords[1], -40.0f);**

**glColor3f(1.0f, 0.0f, 0.0f);**

**glBegin(GL\_POLYGON);**

**glVertex2d( 1.0f, 1.0f);**

**glVertex2d( 1.0f, -1.0f);**

**glVertex2d(-1.0f, -1.0f);**

**glVertex2d(-1.0f, 1.0f);**

**glEnd();**

**}**

**void display(){**

**glClear(GL\_COLOR\_BUFFER\_BIT);**

**glMatrixMode(GL\_MODELVIEW);**

**// thu tu dinh la chieu kim dong ho**

**// thu tu phu la truoc, sau, trai, phai, tren, duoi (neu co)**

**// lap lai kich thuoc con ran va ve tung phan tai toa do tuong ung cua no**

**for(unsigned int a = 0; a < part\_coords.size(); a++){**

**glLoadIdentity();**

**glTranslatef(part\_coords[a][0], part\_coords[a][1], -40.0f);**

**glColor3f(0.0f, 1.0f, 0.0f);**

**glBegin(GL\_POLYGON);**

**glVertex2d( 1.0f, 1.0f);**

**glVertex2d( 1.0f, -1.0f);**

**glVertex2d(-1.0f, -1.0f);**

**glVertex2d(-1.0f, 1.0f);**

**glEnd();**

**}**

**spawnFood();**

**glutSwapBuffers();**

**}**

**void moveSnake(int new\_direction){**

**direction = new\_direction;**

**int last\_part = part\_coords.size() - 1;**

**std::deque<float> new\_head = part\_coords[last\_part];**

**if(direction == UP){**

**// co phai ran da dung trung minh**

**for(unsigned int a = 0; a < part\_coords.size(); a++){**

**if(part\_coords[0][0] == part\_coords[a][0] &&**

**part\_coords[0][1] + 2.0f == part\_coords[a][1]){**

**exit(0);**

**}**

**}**

**// co phai ran da va vao tuong**

**if(part\_coords[0][1] == map\_half\_length){**

**exit(0);**

**}**

**// chung ta da nhan duoc thuc an**

**if(part\_coords[0][0] == food\_coords[0] &&**

**part\_coords[0][1] + 2.0f == food\_coords[1]){**

**growth\_stage++;**

**food\_available = false;**

**}**

**new\_head[1] = part\_coords[0][1] + 2.0f;**

**} else if(direction == DOWN){**

**for(unsigned int a = 0; a < part\_coords.size(); a++){**

**if(part\_coords[0][0] == part\_coords[a][0] &&**

**part\_coords[0][1] - 2.0f == part\_coords[a][1]){**

**exit(0);**

**}**

**}**

**if(part\_coords[0][1] == -map\_half\_length){**

**exit(0);**

**}**

**if(part\_coords[0][0] == food\_coords[0] &&**

**part\_coords[0][1] - 2.0f == food\_coords[1]){**

**growth\_stage++;**

**food\_available = false;**

**}**

**new\_head[1] = part\_coords[0][1] - 2.0f;**

**} else {**

**new\_head[1] = part\_coords[0][1];**

**}**

**if(direction == LEFT){**

**for(unsigned int a = 0; a < part\_coords.size(); a++){**

**if(part\_coords[0][0] - 2.0f == part\_coords[a][0] &&**

**part\_coords[0][1] == part\_coords[a][1]){**

**exit(0);**

**}**

**}**

**if(part\_coords[0][0] == -map\_half\_length){**

**exit(0);**

**}**

**if(part\_coords[0][0] - 2.0f == food\_coords[0] &&**

**part\_coords[0][1] == food\_coords[1]){**

**growth\_stage++;**

**food\_available = false;**

**}**

**new\_head[0] = part\_coords[0][0] - 2.0f;**

**} else if(direction == RIGHT){**

**for(unsigned int a = 0; a < part\_coords.size(); a++){**

**if(part\_coords[0][0] + 2.0f == part\_coords[a][0] &&**

**part\_coords[0][1] == part\_coords[a][1]){**

**exit(0);**

**}**

**}**

**if(part\_coords[0][0] == map\_half\_length){**

**exit(0);**

**}**

**if(part\_coords[0][0] + 2.0f == food\_coords[0] &&**

**part\_coords[0][1] == food\_coords[1]){**

**growth\_stage++;**

**food\_available = false;**

**}**

**new\_head[0] = part\_coords[0][0] + 2.0f;**

**} else {**

**new\_head[0] = part\_coords[0][0];**

**}**

**part\_coords.push\_front(new\_head);**

**if(!growth\_stage){**

**part\_coords.pop\_back();**

**} else if(growth\_stage == growth){**

**growth\_stage = 0;**

**} else {**

**growth\_stage++;**

**}**

**glutPostRedisplay();**

**}**

**void keyboard(int key, int x, int y){**

**UNUSED(x);**

**UNUSED(y);**

**switch(key){**

**case GLUT\_KEY\_UP:{**

**if(direction == LEFT || direction == RIGHT){**

**moved = true;**

**moveSnake(UP);**

**}**

**break;**

**}**

**case GLUT\_KEY\_DOWN:{**

**if(direction == LEFT || direction == RIGHT){**

**moved = true;**

**moveSnake(DOWN);**

**}**

**break;**

**}**

**case GLUT\_KEY\_LEFT:{**

**if(direction == UP || direction == DOWN){**

**moved = true;**

**moveSnake(LEFT);**

**}**

**break;**

**}**

**case GLUT\_KEY\_RIGHT:{**

**if(direction == UP || direction == DOWN){**

**moved = true;**

**moveSnake(RIGHT);**

**}**

**break;**

**}**

**}**

**glutPostRedisplay();**

**}**

**void initGL(){**

**glMatrixMode(GL\_PROJECTION);**

**gluPerspective(75.0f, 1, 0.0f, 35.0f);**

**glClearColor(0.0f, 0.0f, 0.0f, 1.0f);**

**}**

**void moveSnakeAuto(int value){**

**if(!moved){**

**UNUSED(value);**

**if(direction == UP){**

**moveSnake(UP);**

**} else if(direction == DOWN){**

**moveSnake(DOWN);**

**} else if(direction == LEFT){**

**moveSnake(LEFT);**

**} else if(direction == RIGHT){**

**moveSnake(RIGHT);**

**}**

**} else {**

**moved = false;**

**}**

**glutTimerFunc(move\_speed, moveSnakeAuto, 0);**

**}**

**void reshape(GLsizei width, GLsizei height){**

**UNUSED(width);**

**UNUSED(height);**

**// co dinhh kich thuoc cua so**

**glutReshapeWindow(600, 600);**

**}**

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

**glutInit(&argc, argv);**

**glutInitWindowSize(600, 600);**

**glutCreateWindow(title);**

**glutDisplayFunc(display);**

**glutReshapeFunc(reshape);**

**glutSpecialFunc(keyboard);**

**glutTimerFunc(move\_speed, moveSnakeAuto, 0);**

**int initSize = 3;**

**// chi dinh toa do tung phan con ran**

**for(int a = 1; a <= initSize; a++){**

**std::deque<float> row;**

**row.push\_back(0.0f);**

**row.push\_back((map\_half\_length + 2.0f + (initSize \* 2)) - (a \* 2));**

**part\_coords.push\_front(row);**

**}**

**srand(time(NULL));**

**initGL();**

**glutMainLoop();**

**return 0;**

**}**