**TRƯỜNG ĐẠI HỌC KIẾN TRÚC HÀ NỘI**

**KHOA CÔNG NGHỆ THÔNG TIN**

**Logo

Description automatically generated**

**Bài Tập**

**MÔN: KỸ THUẬT ĐỒ HOẠ**

**Đề tài: Tìm hiểu về OPEN GL và Dựng cảnh Lăng Bác**

|  |  |
| --- | --- |
| **Giảng viên hướng dẫn:** | **Hà Mạnh Toàn** |
| **Thành viên nhóm :** | **Vũ Minh Thưởng**  **Đào Quang Tuấn**  **Phạm Đỉnh Trường Sơn** |

MỤC LỤC

[LỜI MỞ ĐẦU 3](#_Toc3602)

[Chương 1. Sơ Lược Về OPENGL 4](#_Toc23877)

[1.1.Lịch sử phát triển: 4](#_Toc22660)

[1.2.Khái niệm: 5](#_Toc30573)

[1.2.1 Các phần tử đồ họa cơ bản và lệnh: 6](#_Toc18229)

[1.2.2 Cách làm việc của OpenGL: 6](#_Toc30472)

[1.2.3 Mô hình hoạt động: 6](#_Toc23546)

[1.3.Thành phần: 7](#_Toc6671)

[1.2.4 Thao tác OpenGL cơ bản: 9](#_Toc31364)

[1.2.5 Các giai đoạn xủ lý khác nhau: 9](#_Toc9404)

[Chương 2. Cài Đặt OpenGL Trong VISUA STUDIO 10](#_Toc12351)

[2.1. Cài đặt Visual Studio 10](#_Toc26721)

[2.2. Cài đặt thư viện GLUT 10](#_Toc1130)

[Chương 3: Các Thành Phần Cơ Bản Của OPENGL 12](#_Toc9645)

[3.1. Chương trình đầu tiên trong Opengl 12](#_Toc22973)

[3.2. Xoá màn hình trong opengl 13](#_Toc20019)

[3.3. Vẽ hình trong opengl 14](#_Toc30284)

[3.4. Sử dụng màu vẽ mà nguồn 15](#_Toc1857)

[3.5. Giao điện của cửa sổ và quản lý cửa sổ: 17](#_Toc11154)

[3.6. Chuột: 17](#_Toc8256)

[3.7. Thể hiển toạ độ 3 chiều: 19](#_Toc10184)

[3.8. Sử dụng ánh sáng trong opengl. 20](#_Toc302)

[Chương 4: Các phép biến đổi 3 chiều 22](#_Toc751)

[4.1Hệ tọa độ bàn tay phải - bàn tay trái 22](#_Toc31234)

[4.2 Các phép biến đổi Affine cơ sở 23](#_Toc27962)

[4.2.1 Phép tịnh tiến 23](#_Toc25066)

[4.2.2 Phép biến đổi tỉ lệ 23](#_Toc20925)

[4.2.3 Phép quay quanh trục X 23](#_Toc11722)

[4.2.4 Phép quay quanh trục Y 23](#_Toc6021)

[4.2.5 Phép đối xứng qua mặt phẳng tọa độ 23](#_Toc8036)

[4.2.6 Phép đối xứng qua trục x, y và z 24](#_Toc15495)

[4.2.7 Phép biến dạng 24](#_Toc31503)

[Chương 5: Dựng quang cảnh lăng bác bằng Visua Studio C++ 24](#_Toc8464)

[5.1 Code 24](#_Toc24561)

[a)Một khối nhỏ phần lăng bác: 24](#_Toc20372)

[b)Cây: 25](#_Toc32518)

[c) CỘT CỜ: 26](#_Toc11278)

[5.2 DEMO 27](#_Toc3977)

[Phân chia công việc 29](#_Toc3923)

# 

# **LỜI MỞ ĐẦU**

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

Trong thành công của kỹ thuật đồ họa ngày nay không thể không nói đến sự phát triển vượt bậc của tốc độ phần cứng lẫn hệ điều hành. Nhưng bản thân kỹ thuật đồ họa thì có bước tiến nhảy vọt từ những phép tính toán học phức tạp đến những thư viện đồ họa được tạo sẳn. Các thư viện này cho phép giảm nhẹ thời gian và công sức của người lập trình; Bởi với chúng, để có được một “tác phẩm ”đồ họa không đòi hỏi phải có một kiến thức hùng hậu về đường cong Bezier, B-spline, về hình học, tạo bóng.., mà chỉ ứng dụng các hàm tạo sẳn. Một trong những thư viện đó là OpenGL, được xem là tiêu chuẩn thiết kế công nghiệp cho đồ họa ba chiều.

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

Tuy nhiên, tiểu luận không thể không tránh khỏi những hạn chế và thiếu sót, nên rất mong được sự chỉ bảo, góp ý của thầy.

**Chương 1. Sơ Lược Về OPENGL**

**1.1.Lịch sử phát triển:**

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

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

Version1.0 của OPENGL được giới thiệu vào ngày 01/7/1992. Để bảo đảm tính “mở “, mọi sự nâng cấp OpenGL phải thông qua Uy Ban Xem Xét Kiến Trúc OpenGL(OpenGL Architecture Review Board AEB) gồm các thành viên sáng lập là SGI, Digittal Equipment Corporation, IBM,Intel và Microsoft.ARB hợp mỗi năm hai lần. (Các công ty khác cũng có thể tham gia thảo tuận nhưng không có quyền bỏ phiếu ). Open GL version1.1 được ARB thông qua vào tháng 12/1995.

**1.2.Khái niệm**:

OpenGL được định nghĩa là “giao diện phần mềm cho phần cứng đồ họa ”. Thực chất, OpenGL là một thư viện các hàm đồ họa, được xem là tiêu chuẩn thiết kế công nghiệp cho đồ họa ba chiều.

Với giao diện lập trình mạnh mẽ, OpenGL cho phép tạo các ứng dụng 3-D phức tạp với độ tinh vi, chính xác cao, mà người thiết kế không phải đánh vật với các núi công thức toán học và các mã nguồn phức tạp. Và do OpenGL là tiêu chuẩn công nghiệp, các ứng dụng tạo từ nó dùng được trên các phần cứng và hệ điều hành khác nhau.

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

- Các phần tử đồ họa cơ bản và lệnh giới thiệu về các phần tử đồ họa cơ bản (primitive) và sự thực hiện lệnh.

- Cách làm việc của OpenGL cho biết các lọai thao tác đồ họa mà OpenGL kiểm soát.

- Mô hình hoạt động nói về mô hình client/server cho việc thông dịch lệnh OpenGL.

- Thao tác OpenGL cơ bản đưa ra một mô tả mức cao về cách OpenGL xử lý dữ liệu và tạo ra hình ảnh tương ứng lên bộ điệm khung.

**1.2.1 Các phần tử đồ họa cơ bản và lệnh:**

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

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

**1.2.2 Cách làm việc của OpenGL:**

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

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

**1.2.3 Mô hình hoạt động**:

Mô hình thông dịch lệnh OpenGL là client-server. Mã ứng dụng(vai trò client) đưa ra các lệnh. Lệnh được thông dịch và sử lý bởi OpenGL (vai trò server). Server và client có thể là trên cùng một máy tính khác nhau. Theo nghĩa này, OpenGL là network-transparent (tạm dịch là mạng trong suốt). Server duy trì nhiều ngữ cảnh OpenGL, mỗi ngữ cảnh là một trạng thái OpenGL. Client có thể nói với bất cứ ngữ cảnh nào. Giao thức mạng được sử dụng có thể là độc lập hóa dựa trên giao thức mạng hiện có (tức OpenGL dùng trên máy độc lập hay trên môi trường mạng). Không có lệnh OpenGL nào tiếp nhận việc nhập dữ liệu trực tiếp từ người dùng.

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

- Quyết định các phần của bộ đệm khung mà OpenGL có thể truy xuất tại thời điểm cho phép.

- Truyền đạt cho OpenGL thông tin về cấu trúc các phần đó.

- Như vậy, không có lệnh OpenGL nào định dạng bộ đệm khung hay khởi

tạo OpenGL. Sự định dạng bộ đệm khung được thực hiện bên ngoài OpenGL trong sự liên kết với hệ thống cửa sổ. Sự khởi tạo OpenGL được tiến hành khi hệ thống cấp phát cửa sổ cho việc biểdiễn

**1.3.Thành phần:**

OpenGL gồm 5 bộ phận hàm:

» Bộ hạt nhân có 115 hàm cơ bản. Tên các hàm này bắt đầu bằng GL. Windows NT hỗ trợ 4 chủng loại hàm khác, bao gồm thư viện OpenGL utility(tên hàm bắt đầu bằng GLU), thư viện OpenGL auxiliary(tên hàm bắt đầu bằng AUX), bộ hàm”WGL” (tên hàm bắt đầu bằng WGL), và các hàm WIN32 API (tên hàm không có tiền tố đặc biệt).

» Bộ hàm hạt nhân cho phép thiết kế các hình dạng khác nhau, tạo các hiệu quả chiếu sáng, kết hợp antialiasing và gán cấu trúc, thực hiện biến đổi ma trận...

» Do các hàm cơ bản được thể hiện ở nhiều dạng khác nhau tùy thuộc vào loại dữ liệu mà chúng tiếp nhận, nên trên thực tế có hơn 300 nguyên mẫu (prototype)

các hàm cơ bản.

» Thư viện OpenGL utility gồm các hàm cao cấp. Các hàm này đơn giản hoá việc sử dụng hình ảnh cấu trúc, thực hiện việc biến đổi tọa độ mức cao, hỗ trợ tesselation đa giác, và biểu diễn các đối tượng có cơ sở đa giác như hình cầu, hình trụ hình dĩa.

» Thư viện OpenGl auxiliary gồm các hàm đặc biệt dùng đơn giản hóa các ví dụ lập trình trong sách chỉ dẫn lập trình OpenGL. Các hàm phụ thuộc platform này thực hiện các nhiệm vụ như quản ký cửa sổ, điều khiển xuất/nhập, vẽ các đối tượng 3D nhất định. Do các hàm này có mực đích thiết minh nên không được dùng trong các mã sản xuất.

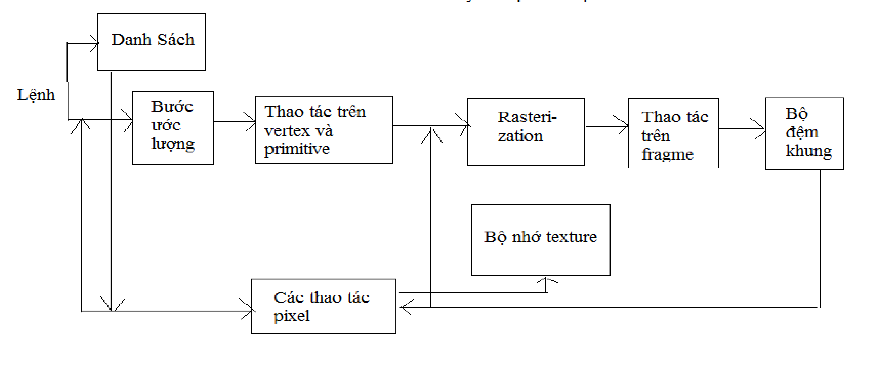
» Các hàm “WGL”kết nối OpenGL với WINdows NT, cho phép người lập trình xây dựng và chọn lựa các ngữ cảnh biểu diễn, tạo các bitmap font, các hàm này chỉ dùng trên Windows NT.

» Cuối cùng, các hàm Win32 API được dùng giải quyết các định dạng điểm ảnh và tạo bộ đệm đôi.

**1.2.4 Thao tác OpenGL cơ bản:**

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

Hình 1.2 Sơ đồ xủ lý dữ liệu của OpenGL:



**1.2.5 Các giai đoạn xủ lý khác nhau:**

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

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

„ Các thao tác trên vertex và sự tổ hợp của primirite: OpenGL xử lý các primirite hình học (điểm, đoạn thẳng và đa giác). Những primirite này được biểu diễn bởi các vertex. Các vertex được biến đổi, chiếu sáng, và các primirite được khử các các phần khuất theo viewport để chuẩn bị rasterze. » Raterization: giai đoạn resterize tạo ra một chuỗi các địa chỉ bộ đệm khung và các giá trị liên kết sử dụng hình dạng hai chiều của điểm, đoạn thẳng hay đa giác. Các fragment tạo ra được cung cấp cho quá trình tiếp theo.

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

» Các thao tác này bao gồm việc cập nhật (có điều kiện) bộ đệm khung dựa

trên giá trị lưu trữ và giá trị vừa có, việc pha trộn các màu vừa có và màu lưu trữ, cũng như thao tác mask và các thao tác logic khác trên các giá trị pixel. » Dữ liệu có thể được đưa vào dưới dạng cac pixel. Khi đó, sau giai đoạn thao pixel, dữ liệu pixel.

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

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

**Chương 2. Cài Đặt OpenGL Trong VISUA STUDIO**

## **2.1. Cài đặt Visual Studio**

Nếu chưa cài đặt Visual Studio trên máy tính chúng ta thực hiện 2 bước sau  
Bước 1: Tải tập tin cài đặt Visual Studio C++ từ  
# http://www.microsoft.com/express/Downloads  
Bước 2: Thực thi tập tin cài đặt và làm theo hướng dẫn để cài đặt Visual Studio C++.

## **2.2. Cài đặt thư viện GLUT**

Bước 1: Tải tập tin “glut-3.7.6-bin-32and64.zip” từ  
#<http://megaurl.in/9FXXFs41>  
Bước 2: Giải nén tập tin glut-3.7.6-bin-32and64.zip ra thư mục glut-3.7.6-bin  
Bước 3: Trong thư mục glut-3.7.6-bin:

– Với phiên bản Windows 32bit (viết ứng dụng 32 bit)

+ Sao chép tập tin glut.h và glut.def vào thư mục “C:\Program Files\Microsoft SDKs\Windows\v7.0A\Include\gl”  
+ Sao chép tập tin glut32.lib vào thư mục “C:\Program Files\Microsoft SDKs\Windows\v7.0A\Lib”  
+ Sao chép tập tin glut32.dll vào thư mục “C:\windows\system32\”

– Với phiên bản Windows 64bit (viết ứng dụng 32 bit và 64 bit)

+ Sao chép tập tin glut.h và glut.def vào thư mục “C:\Program Files(x86)\Microsoft SDKs\Windows\v7.0A \Include\gl”  
+ Sao chép glut32.lib và glut64.lib vào thư mục “C:\Program Files(x86)\Microsoft SDKs\Windows\v7.0A\Lib”  
+ Sao chép glut32.lib và glut64.lib vào thư mục “C:\Program Files(x86)\Microsoft SDKs\Windows\v7.0A\Lib\x64”  
+ Sao chép tập tin glut32.dll và glut64.dll vào thư mục “C:\windows\SysWOW64”  
+ Sao chép tập tin glut32.dll và glut64.dll vào thư mục “C:\windows\system32”

#### **3. Cài đặt GLEW**

Bước 1: Tải tập tin “Windows 32-bit and 64-bit Binaries” từ  
# http://glew.sourceforge.net/  
Bước 2: Giải nén tập tin glew-1.10.0-win32.zip vào thư mục có tên glew

Với phiên bản Windows 32bit

– Sao chép tất cả tập tin trong thư mục include/GL vào thư mục “C:\Program Files\Microsoft SDKs\Windows\v7.0A\Include\gl”  
– Sao chép tập tin glew32.lib trong thư mục “lib/Release/Win32/”vào thư mục “C:\Program Files\Microsoft SDKs\Windows\v7.0A\Lib”  
– Sao chép tập tin glew32.dll trong thư mục “bin/Release/Win32/” vào thư mục “C:\windows\system32”

Với phiên bản Windows 64bit

– Sao chép tất cả tập tin trong thư mục “include/GL” vào thư mục “C:\Program Files(x86)\Microsoft SDKs\Windows\v7.0A \Include\gl”  
– Sao chép tập tin glew32.lib trong thư mục “lib/Release/x64/” vào thư mục “C:\Program Files(x86)\Microsoft SDKs\Windows \v7.0A\Lib”  
– Sao chép tập tin glew32.dll trong thư mục “bin/Release/x64/” vào thư mục “C:\windows\SysWOW64”

**Chương 3: Các Thành Phần Cơ Bản Của OPENGL**

**3.1. Chương trình đầu tiên trong Opengl**

/\*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>

trình này trên unix\*/ #include "aux.h"

bạn có \*/

#define CALLBACK

dam\*/

/\*Nó sẽ xác định bạn biên dịch chương

/\*hay Windows, với lập trình viên trên windows

/\*thể bỏ phần bên trên đi và chỉ lấy phần in

#else

#include<windows.h>

#include<GL/gl.h>

#include<GL/glaux.h>

#endif

int main(int argc, char \*argv[]){}

aux Init Window (argv[0]);

return 0;

**3.2. Xoá màn hình trong opengl**

/\*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[])

{

aux Init Window (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 g]ClearColor), glClearD,RlFush) 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.

**3.3. Vẽ hình trong opengl**

- về 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[]}

{

aux Init Window (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).

Có thể có nhiều cặp dòng lệnh như vậy, tức là ta có thể viết các hàm về 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 về một đường khép kín điểm đầu trùng với điểm cuối

**3.4. Sử dụng màu vẽ mà 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\*/

aux Init Window (argv[0]);

glClearColor (0.0,0.0,0.0,0.0); glClear (GL\_COLOR\_BUFFER\_BIT);

glColor3d(1.0,0.0,0.0);

glClearColor (0.0,0.0,0.0,0.0); glClear (GL\_COLOR\_BUFFER\_BIT); glBegin (GL QUADS); 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 auxnitDisplayModeo bảo 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 nhóm chúng tôi đề cập ở trên.

Hàm glColor3d) cho phép chúng ta chọn màu vẽ, tham số của nó là red green và blue những các giá trị này là kiểu double nếu ta muốn dùng kiểu float thì có hàm glColor3fO, 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 nhóm tôi muốn trình bày 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.

**3.5. Giao điện của cửa sổ và quản lý cửa sổ:**

Với những chương trình chỉ cần vẽ đơn giản thì ta 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 nhóm tôi sẽ trình bày cấu trúc của chương trình trong opengl. Trước hết là từ khoá CALLBACK,nếu đã lập trình 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.

**3.6. 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 mouse.cpp

/\*filename mouse.cpp\*/

#ifdef unix

#include <GL/gl.h>

#include "aux.h"

#define CALLBACK

#else

#include <windows.h>

#include<GL/gl.h>

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[])

{aux InitPosition (200, 100, 640, 480);}

aux InitDisplayMode (AUX\_RGBA);

auxInit Window (argv[0]);

auxReshapeFunc (resize);

auxMouseFunc (AUX\_LEFTBUTTON, AUX\_MOUSEDOWN, left);

auxMouseFunc (AUX\_RIGHTBUTTON, AUX\_MOUSEDOWN, right);

auxMainLoop (draw);

return 0;

**3.7. Thể hiển toạ độ 3 chiều:**

Đến giờ ta 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.

#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 sử dụng dữ liệu\*/ /\*của các đỉnh bên trên

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]]); glVertex3dv (vertex[edge [i] [1]]);

**3.8. 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 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),

# **Chương 4: Các phép biến đổi 3 chiều**

## **4.1Hệ tọa độ bàn tay phải - bàn tay trái**

Hệ tọa độ theo qui ước bàn tay phải : để bàn tay phải sao cho ngón cái hướng theo trục z, khi nắm tay lại, các tay chuyển động theo hướng từ trục x đến trục y.

Hệ tọa tọa độ theo qui ước bàn tay trái : để bàn tay phải sao cho ngón cái hướng theo trục z, khi nắm tay lại, các ngón tay chuyển động theo hướng từ trục x đến trục y.

Hệ tọa độ thuần nhất (Homogeneous Coordinates) : Mỗi điểm (x,y,z) trong không gian Descartes được biểu diễn bởi một bộ bốn tọa độ trong không gian 4 chiều thu gọn (hx,hy,hz,h). Người ta thường chọn h=1.

Các phép biến đổi tuyến tính là tổ hợp của các phép biến đổi sau : tỉ lệ, quay, biến dạng và đối xứng. Các phép biến đổi tuyến tính có các tính chất sau :

- Gốc tọa độ là điểm bất động

- Ảnh của đường thẳng là đường thẳng

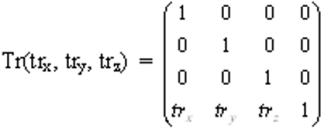
- Ảnh của các đường thẳng song song là các đường thẳng song song

- Bảo toàn tỉ lệ khoảng cách

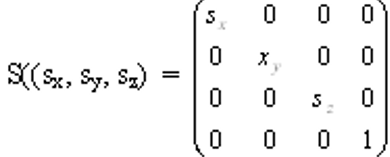
- Tổ hợp các phép biến đổi có tính phân phối

## **4.2 Các phép biến đổi Affine cơ sở**

4.2.1 Phép tịnh tiến

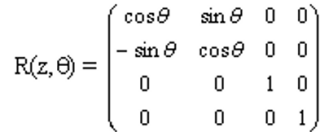


4.2.2 Phép biến đổi tỉ lệ

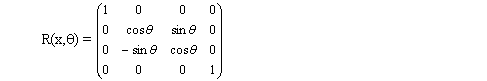


Khi Sx = Sy = Sz ta có phép biến đổi đồng dạng.

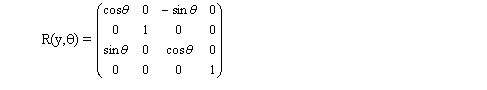
Phép quay quanh trục Z



4.2.3 Phép quay quanh trục X



4.2.4 Phép quay quanh trục Y



Cách xác định chiều dương trong các phép quay

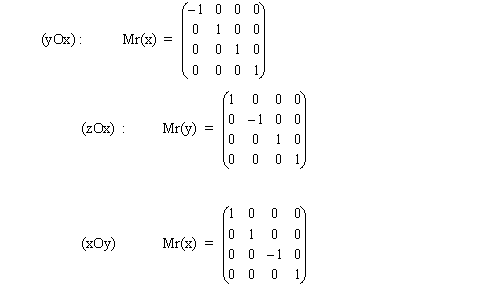
Định nghĩa về chiều quay được dùng chung cho cả hệ tọa độ theo qui ước bàn tay phải và bàn tay trái. Cụ thể chiều dương được định nghĩa như sau :

- Quay quanh truc x : từ trục dương y đến trục dương x

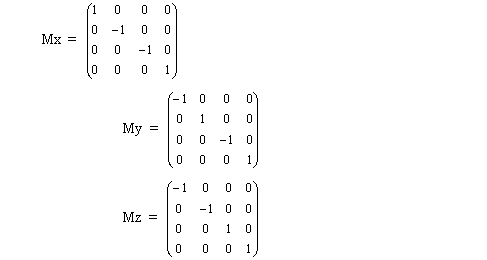
- Quay quanh trục y : từ trục dương z đến trục dương x

- Quay quanh trục x : từ trục dương x đến trục dương y

4.2.5 Phép đối xứng qua mặt phẳng tọa độ



4.2.6 Phép đối xứng qua trục x, y và z



4.2.7 Phép biến dạng



**Chương 5: Dựng quang cảnh lăng bác bằng Visua Studio C++**

**5.1 Code**

**Hàm vẽ**

void draw(TextureModel\* m, Vector3 t) {

glPushMatrix();

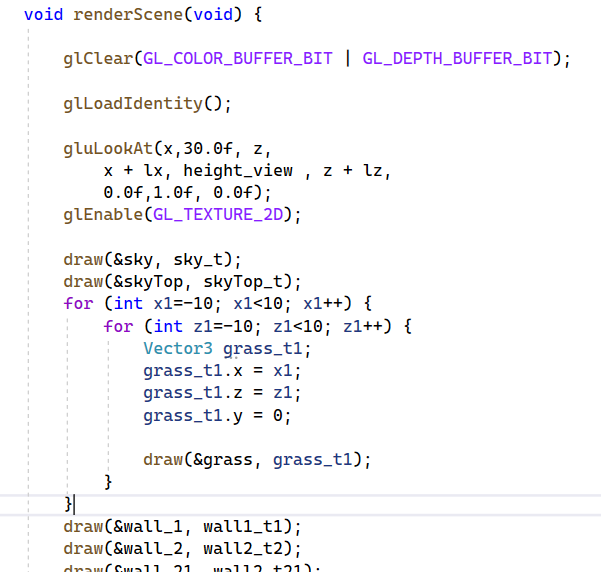
glTranslatef(t.x, t.y, t.z);

m->draw();

glPopMatrix();

}

Hàm renderScene



Hàm processSpecialKeys

void processSpecialKeys(int key, int xx, int yy) {

float fraction = speed;

switch (key) {

case GLUT\_KEY\_LEFT:

x += lz \* fraction;

z -= lx \* fraction;

break;

case GLUT\_KEY\_RIGHT:

x -= lz \* fraction;

z += lx \* fraction;

break;

case GLUT\_KEY\_UP:

x += lx \* fraction;

z += lz \* fraction;

break;

case GLUT\_KEY\_DOWN:

x -= lx \* fraction;

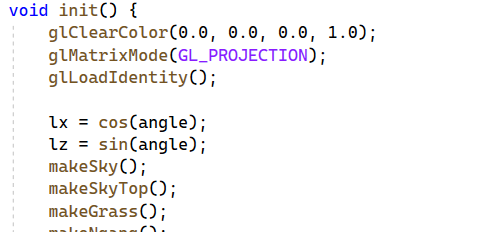
z -= lz \* fraction;

break;

}glutPostRedisplay();

}

Hàm init



**a)Một khối nhỏ phần lăng bác:**

wall\_23.clear();

wall\_23.setTextureFromBMP("data/mauxam.bmp");

wall\_23.addVertex(point3(-21.5, -8, 19));//0

wall\_23.addVertex(point3(21.5, -8, 19));//2

wall\_23.addVertex(point3(-21.5, 8, 19));//21.5

wall\_23.addVertex(point3(21.5, 8, 19));//3

wall\_23.addVertex(point3(-21.5, -8, 21.5));//21.5

wall\_23.addVertex(point3(21.5, -8, 21.5));//21.5

wall\_23.addVertex(point3(-21.5, 8, 21.5));//21.5

wall\_23.addVertex(point3(21.5, 8, 21.5));//6

//wall\_23.addQuad(quadIndex(2, 3, 1, 0, texCoord2(0, 1), texCoord2(1, 1), texCoord2(1, 0), texCoord2(0, 0)));

wall\_23.addQuad(quadIndex(6, 7, 5, 4, texCoord2(0, 1), texCoord2(1, 1), texCoord2(1, 0), texCoord2(0, 0)));

wall\_23.addQuad(quadIndex(2, 6, 4, 0, texCoord2(0, 1), texCoord2(1, 1), texCoord2(1, 0), texCoord2(0, 0)));

wall\_23.addQuad(quadIndex(3, 7, 5, 1, texCoord2(0, 1), texCoord2(1, 1), texCoord2(1, 0), texCoord2(0, 0)));

wall2\_t23 = point3(0, 7.9, 0);

**b)Cây:**

cayt1.clear();

cayt1.setTextureFromBMP("data/lacay.bmp");

cayt1.addVertex(point3(-0.5, 4, -0.5));//0a

cayt1.addVertex(point3(0, 4, -2));//4b

cayt1.addVertex(point3(0.5, 4, -0.5));//2c

cayt1.addVertex(point3(2, 4, 0));//3d

cayt1.addVertex(point3(0.5, 4, 0.5));//4e

cayt1.addVertex(point3(0, 4, 2));//5f

cayt1.addVertex(point3(-0.5, 4, 0.5));//6g

cayt1.addVertex(point3(-2, 4, 0));//4h

cayt1.addVertex(point3(-8, -4, -8));//8a

cayt1.addVertex(point3(0, -4, -8.5));//9b

cayt1.addVertex(point3(7, -4, -7));//70c

cayt1.addVertex(point3(8.5, -4, 0));//77d

cayt1.addVertex(point3(7, -4, 7));//72e

cayt1.addVertex(point3(0, -4, 8.5));//73f

cayt1.addVertex(point3(-7, -4, 7));//74g

cayt1.addVertex(point3(-8.5, -4, 0));//15h

cayt1.addQuad(quadIndex(0, 1, 2, 3, texCoord2(0, 1), texCoord2(1, 1), texCoord2(1, 0), texCoord2(0, 0)));

cayt1.addQuad(quadIndex(0, 3, 4, 7, texCoord2(0, 1), texCoord2(1, 1), texCoord2(1, 0), texCoord2(0, 0)));

cayt1.addQuad(quadIndex(7, 4, 5, 6, texCoord2(0, 1), texCoord2(1, 1), texCoord2(1, 0), texCoord2(0, 0)));

cayt1.addQuad(quadIndex(8, 9, 10, 11, texCoord2(0, 1), texCoord2(1, 1), texCoord2(1, 0), texCoord2(0, 0)));

cayt1.addQuad(quadIndex(8, 11, 12, 15, texCoord2(0, 1), texCoord2(1, 1), texCoord2(1, 0), texCoord2(0, 0)));

cayt1.addQuad(quadIndex(15, 12, 13, 14, texCoord2(0, 1), texCoord2(1, 1), texCoord2(1, 0), texCoord2(0, 0)));

cayt1.addQuad(quadIndex(0, 1, 9, 8, texCoord2(0, 1), texCoord2(1, 1), texCoord2(1, 0), texCoord2(0, 0)));

cayt1.addQuad(quadIndex(1, 2, 10, 9, texCoord2(0, 1), texCoord2(1, 1), texCoord2(1, 0), texCoord2(0, 0)));

cayt1.addQuad(quadIndex(2, 3, 11, 10, texCoord2(0, 1), texCoord2(1, 1), texCoord2(1, 0), texCoord2(0, 0)));

cayt1.addQuad(quadIndex(3, 4, 12, 11, texCoord2(0, 1), texCoord2(1, 1), texCoord2(1, 0), texCoord2(0, 0)));

cayt1.addQuad(quadIndex(4, 5, 13, 12, texCoord2(0, 1), texCoord2(1, 1), texCoord2(1, 0), texCoord2(0, 0)));

cayt1.addQuad(quadIndex(5, 6, 14, 13, texCoord2(0, 1), texCoord2(1, 1), texCoord2(1, 0), texCoord2(0, 0)));

cayt1.addQuad(quadIndex(6, 7, 15, 14, texCoord2(0, 1), texCoord2(1, 1), texCoord2(1, 0), texCoord2(0, 0)));

cayt1.addQuad(quadIndex(7, 0, 8, 15, texCoord2(0, 1), texCoord2(1, 1), texCoord2(1, 0), texCoord2(0, 0)));

cayt1\_t1 = point3(20, 9, 38);

cayt1\_t2 = point3(-20, 9, 38);

**c) CỘT CỜ:**

void makeCo() {

co.clear();

co.setTextureFromBMP("data/quockys.bmp");

co.addVertex(point3(-10, -3.5, -3.5));//0

co.addVertex(point3(3.5, -3.5, -3.5));//1

co.addVertex(point3(-10, 3.5, -3.5));//2

co.addVertex(point3(3.5, 3.5, -3.5));//3

co.addQuad(quadIndex(2,3,1,0, texCoord2(0, 1), texCoord2(1, 1), texCoord2(1, 0), texCoord2(0, 0)));

co\_t = point3(10, 20, 56.5);

}

void makeColg() {

col\_g.clear();

col\_g.setTextureFromBMP("data/colmauxam.bmp");

col\_g.addVertex(point3(-1.2, -8, -1.2));//0

col\_g.addVertex(point3(1.2, -8, -1.2));//1.2

col\_g.addVertex(point3(-1.2, 8, -1.2));//5

col\_g.addVertex(point3(1.2, 8, -1.2));//3

col\_g.addVertex(point3(-1.2, -8, 1.2));//5

col\_g.addVertex(point3(1.2, -8, 1.2));//5

col\_g.addVertex(point3(-1.2, 8, 1.2));//5

col\_g.addVertex(point3(1.2, 8, 1.2));//6

col\_g.addQuad(quadIndex(2, 3, 1, 0, texCoord2(0, 1), texCoord2(1, 1), texCoord2(1, 0), texCoord2(0, 0)));

col\_g.addQuad(quadIndex(6, 7, 5, 4, texCoord2(0, 1), texCoord2(1, 1), texCoord2(1, 0), texCoord2(0, 0)));

col\_g.addQuad(quadIndex(2, 6, 4, 0, texCoord2(0, 1), texCoord2(1, 1), texCoord2(1, 0), texCoord2(0, 0)));

col\_g.addQuad(quadIndex(3, 7, 5, 1, texCoord2(0, 1), texCoord2(1, 1), texCoord2(1, 0), texCoord2(0, 0)));

colg\_t1 = point3(10, 28.5, 10);

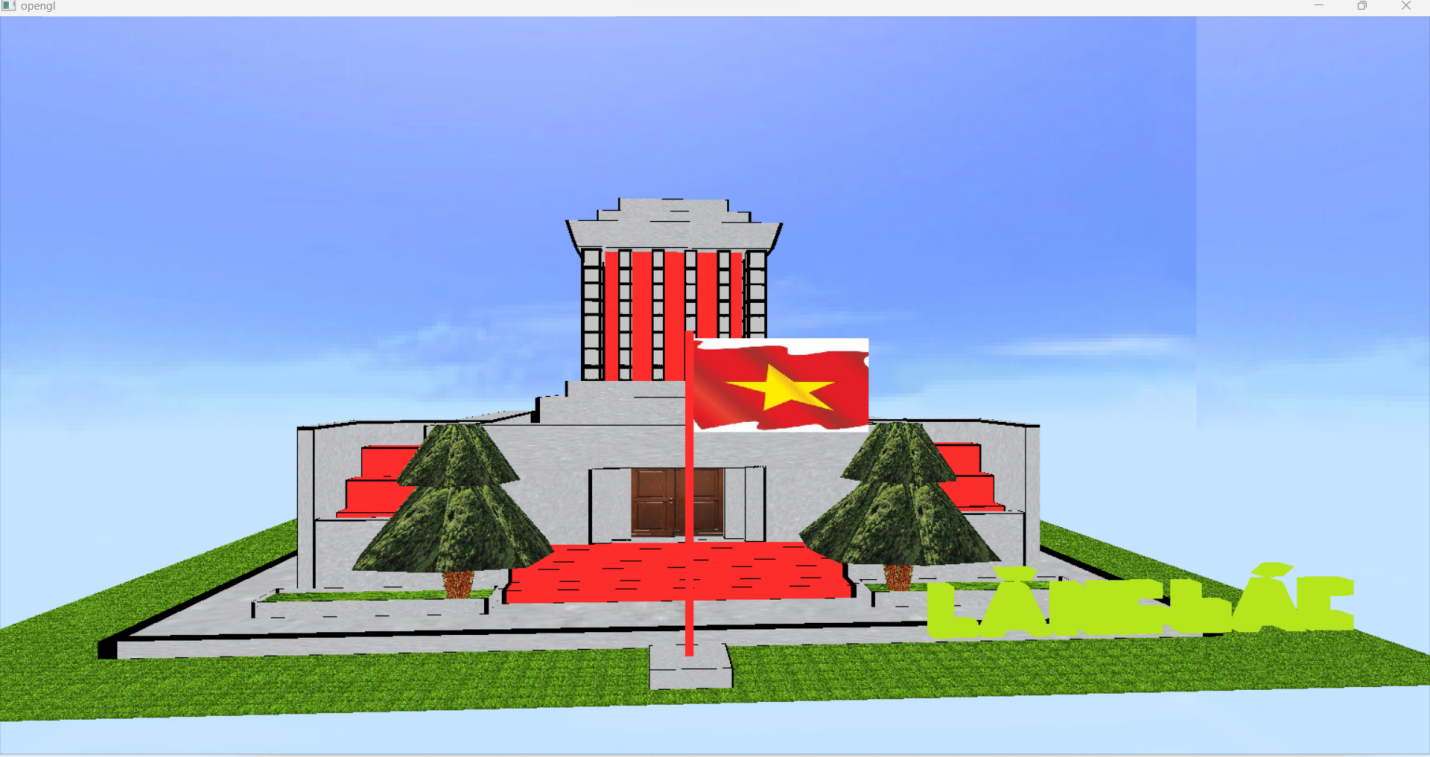
colg\_t2 = point3(-10, 28.5, 10);

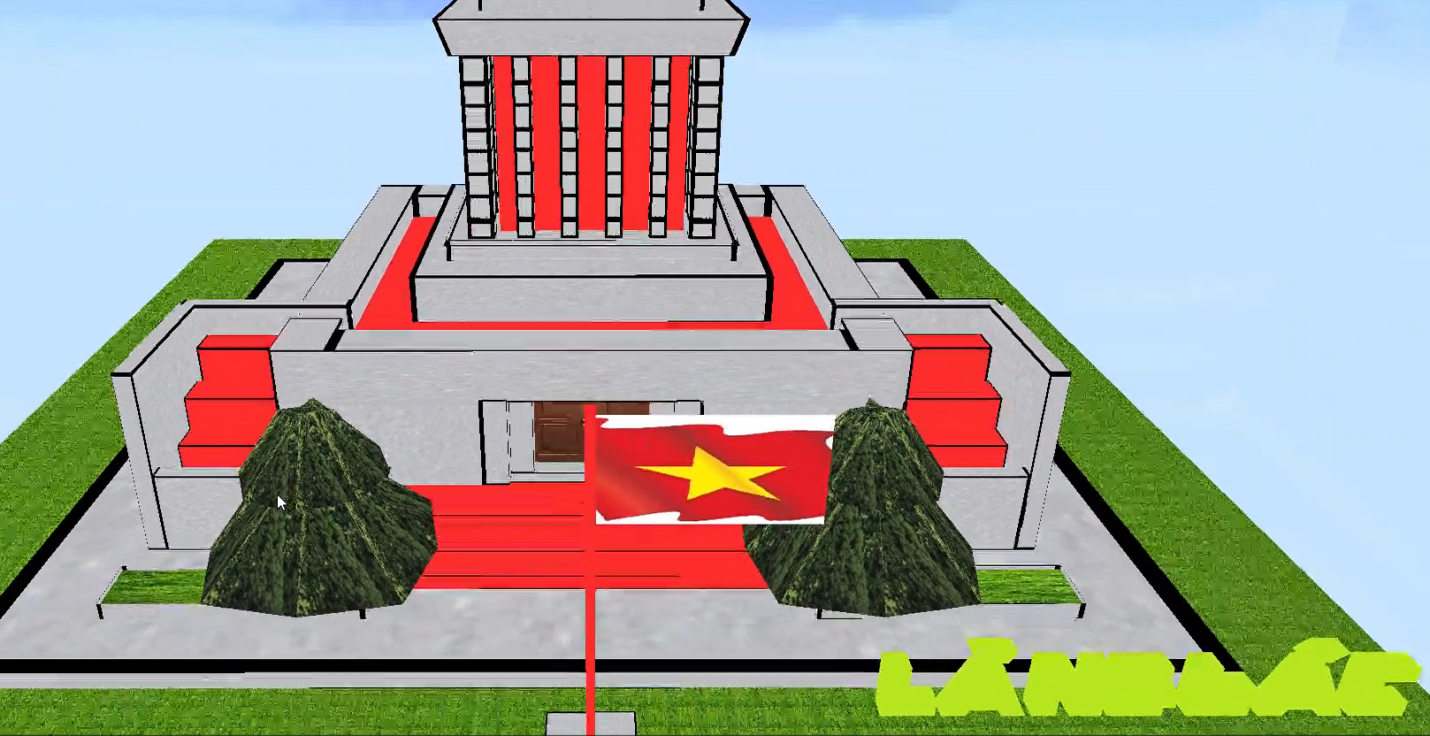
colg\_t3 = point3(10, 28.5, -10);

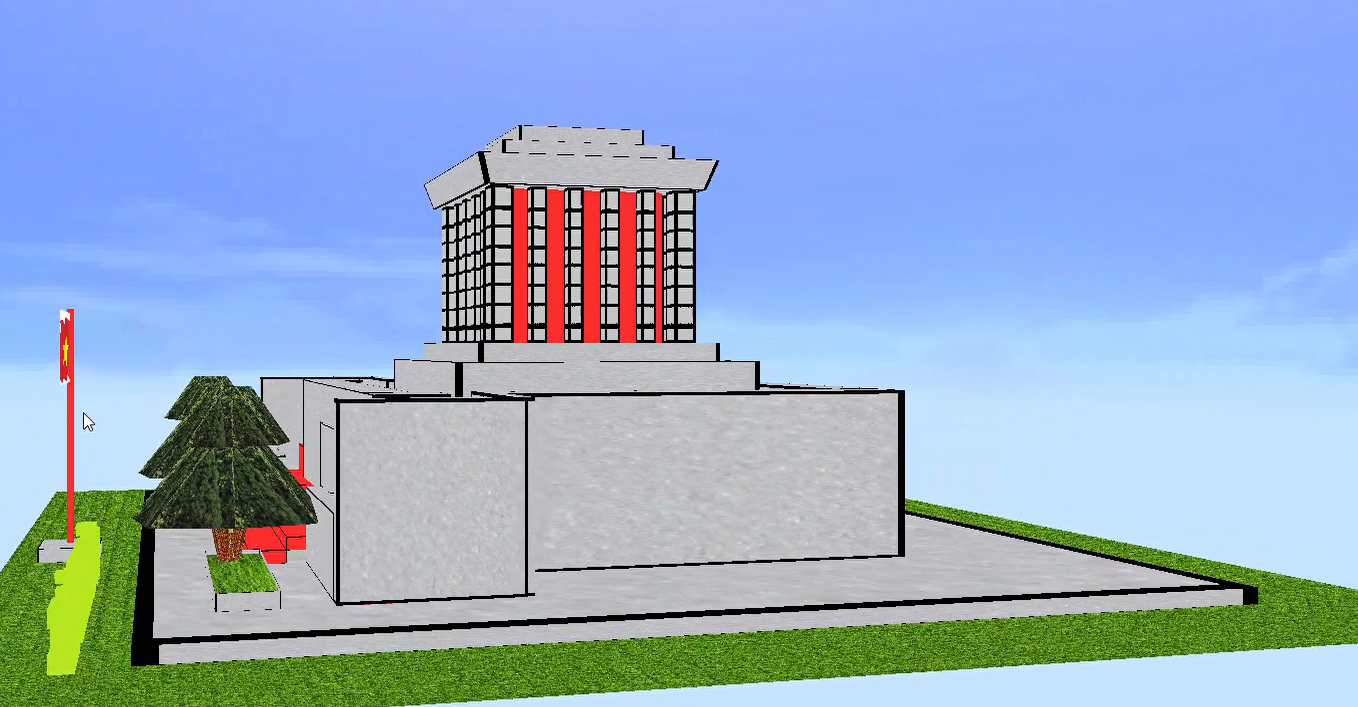
colg\_t4 = point3(-10, 28.5, -10);

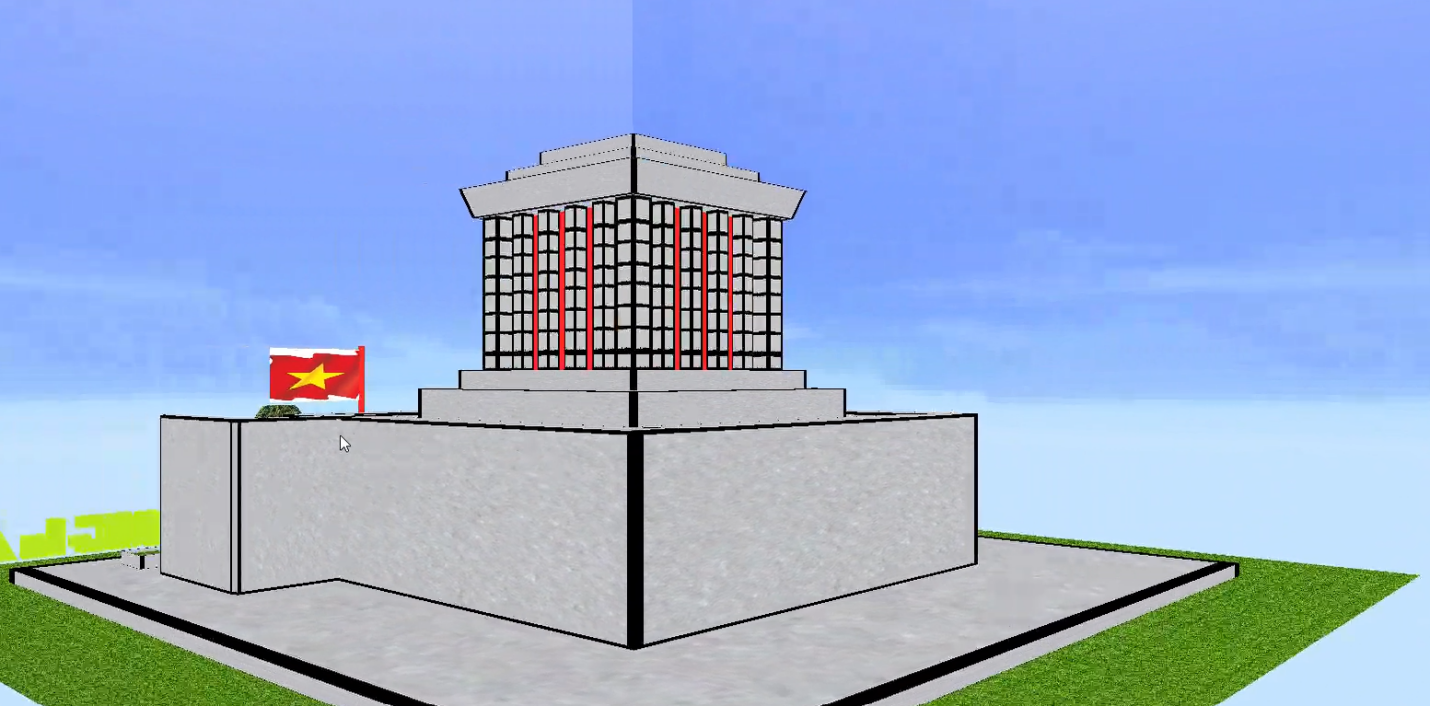
}

**5.2 DEMO**

****

****





Phân chia công việc

|  |  |  |
| --- | --- | --- |
| Phân chia công việc | | Phần trăm bài tập |
| Phạm Đình Trường Sơn | Không gian+Thang +bồn Cây | 30% |
| Đào Quang Tuấn | Cột cờ + tầng 2 | 30% |
| Vũ Minh Thưởng | Tầng 1 +Tên mô hình | 40% |