**TRƯỜNG ĐẠI HỌC ĐIỆN LỰC**

**KHOA CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO CHUYÊN ĐỀ HỌC PHẦN**

**ĐỒ HỌA MÁY TÍNH**

**ĐỀ TÀI:**

**MÔ PHỎNG XE Ô TÔ BÁN TẢI**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | **Sinh viên thực hiện** | **: NGUYỄN ĐÌNH MẠNH** | | | **Giảng viên hướng dẫn** | **: NGÔ TRƯỜNG GIANG** | | **Ngành** | **: CÔNG NGHỆ THÔNG TIN** | | **Chuyên ngành** | **: CÔNG NGHỆ PHẦN MỀM** | | **Lớp** | **: D14CNPM8** | |  |

**Hà Nội, tháng 6 năm 2022**

**PHIẾU CHẤM ĐIỂM**

Sinh viên thực hiện:

|  |  |  |
| --- | --- | --- |
| **Họ và tên** | **Chữ ký** | **Ghi chú** |
| Nguyễn Đình Mạnh |  |  |

|  |  |  |
| --- | --- | --- |
| **Họ và tên giảng viên** | **Chữ ký** | **Ghi chú** |
| Giảng viên chấm 1: |  |  |
| Giảng viên chấm 2: |  |  |

**MỤC LỤC**

|  |  |
| --- | --- |
|  | Trang |
| LỜI NÓI ĐẦU | 1 |
| PHẦN 1: TỔNG QUAN VỀ CƠ SỞ LÝ THUYẾT | 2 |
| CHƯƠNG1: GIỚI THIỆU THƯ VIỆN OPENGL | 2 |
| * 1. . Lịch sử phát triển | 2 |
| 1.2. Khái niệm | 2 |
| 1.3. Cài đặt thư viện mới trên môi trường MS Visual Studio 2019 | 5 |
| CHƯƠNG 2: TÌM HIỂU THƯ VIỆN OPENGL | 6 |
| 2.1 Cấu trúc lệnh trong OpenGL | 6 |
| 2.2 Một số thuật toán vẽ hình học 2D | 6 |
| 2.2.1 Vẽ điểm, đường, đa giác(POINT, LINES,POLYGON) | 6 |
| 2.2.2 Hàm vẽ hình học phức tạp | 7 |
| 2.3 Kỹ thuật tô màu | 8 |
| 2.4 Phép biến đổi đồ họa 2d 3d | 10 |
| 2.5 Phép chiếu | 11 |
| 2.6 Kỹ thuật gán cấu trúc (texture mapping) | 12 |
| PHẦN 2: VẬN DỤNG | 15 |
| 1. ĐẶT VẤN ĐỀ | 15 |
| * 1. Yêu cầu | 15 |
| 2. PHÂN TÍCH YÊU CẦU | 15 |
| 2.1.1 Phân tích yêu cầu | 15 |
| 2.2. Định hướng giải quyết vấn đề | 15 |
| 3. CÀI ĐẶT CHƯƠNG TRÌNH | 16 |
| 3.1. Cấu trúc chương trình | 16 |
| 3.2. CODE | 16 |

**LỜI NÓI ĐẦU**

Sự phát triển của khoa học, kĩ thuật, kinh doanh và công nghệ luôn luôn thuộc vào khả năng truyền đạt thông tin của chúng ta, hoặc thông qua các bit dữ liệu lưu trữ trong microchip hoặc thông qua giao tiếp bằng tiếng nói. Câu châm ngôn từ xa xưa “một hình ảnh có giá trị hơn cả vạn lời” hay “trăm nghe không bằng một thấy” cho thấy ý nghĩa rất lớn của hình ảnh trong việc truyền tải thông tin. Hình ảnh bao giờ cũng cảm nhận nhanh và dễ dàng hơn, đặc biệt là trong trường hợp bất đồng ngôn ngữ. Do đó không có gì ngạc nhiên khi ngay mà từ khi xuất hiện máy tính, các nghiên cứu đã cố gắng sử dụng nó để phát minh ra các hinh ảnh sử dụng trên màn hình. Trong suốt gần 50 năm phát triển của máy tính, khả năng phát sinh hình ảnh của chúng ta đã đạt tới mức bây giờ hầu như tất cả máy tính đều có khả năng đồ họa.

Đồ họa máy tính là một trong những lĩnh vực lí thú và phát triển nhanh nhất của tin học. Ngay từ khi xuất hiện, đồ họa máy tính đã có sức lôi cuốn mãnh liệt, cuốn hút rất nhiều người ở nhiều lĩnh vực khác nhau như: khoa học, nghệ thuật, kinh doanh, quản lí, … Tính hấp dẫn và đa dạng của đồ họa máy tính có thể được minh họa rất trực quan thông qua việc khảo sát các ứng dụng của nó.

Và để hiểu rõ hơn về kỹ thuật đồ họa máy tính chúng em đã tìm hiểu và hoàn thành bài tập lớn về đề tài: “Xây dựng chương trình mô phỏng ô tô bán tải”.

Trong quá trình làm bài chúng em đã hết sức cố gắng, song chắc chắn không thể tránh khỏi thiếu sót, vì vậy rất mong có sự đóng góp ý kiến của thầy.

Em chân thành cảm ơn giảng viên -------- đã giúp chúng em hoàn thành bài tập này!

**PHẦN 1: TỔNG QUAN VỀ CƠ SỞ LÝ THUYẾT**

**CHƯƠNG1: GIỚI THIỆU THƯ VIỆN OPENGL**

* 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 GL với các cấu hình phần cứng khác thì có vấn đề phát sinh
* OpenGL là kết nối nổ lực của SGL nhằm cải ithieejn 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 loại cấu hình phần cứng, cũng như các hệ điều hành khác nhau.
* Version 1.0 của OPENGL được giới thiệu vào ngày 01/7/1992. Để bảo đảm tính “mở”, mọi sự nâng cấp OpenGL phải thông qua Ủy Ban Xem Xét Kiến Trúc OpenGL (OpenGL Architecture Review Board AEB) gồm các thành viên sang lập là SGI, Digittal Equipment Corporation, IBM, Intel và Microsoft. ARB họp mỗi năm hai lần. (Các công ty khác cũng có thể tham gia thảo luận nhưng không có quyền bỏ phiếu). OpenGL version 1.1 được ARB thông qua vào tháng 12/1995.
  1. **Khái niệm**
* OpenGL được định nghĩa là “giao diện phần mềm cho phần cứng đồ họa”. Thực chất, OpenGL là một thư viện 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 3D 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 dung 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 loại 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 lí dữ liệu và tạo ra hình ảnh tương ứng lên bộ đệm khung.
* 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 hợp với vertex. Khi xử lí primitive, mỗi cặp vertex và dữ liệu kết hợp với nó được xử lí độc lập với cặp khác, theo thứ tự và cùng phương pháp. Ngoại lệ duy nhất là trong trường hợp khử phần khuất của primitive (clipping). Khi đó, dữ liệu vertex được sửa vào các vertex khác được tạo ra. Loại clipping tùy thuộc loại primitive mà nhóm vertex biểu diễn.
  + Các lệnh luôn luôn được xử lí theo thứ tự mà nó tiếp nhận, mặc dù có sự trì hoãn không xác định trước khi lệnh có hiệu lực. Nghĩa là mỗi primitive được vẽ trọn vẹn trước khi lệnh tiếp theo có hiệu lực.
* Cách làm việc của OpenGL:
  + OpenGL là ngôn ngữ đồ họa theo thủ tục chứ không phải ngôn ngữ mô tả. Thay vì 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 sỉ 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 sang, tô bóng, gán cấu trúc, tạo ảo giác chuyển động và hiệu quả đặc biệt khác.
  + OpenGL không có các chức năng quản lí của sổ, tương tác với người dung hay xuất nhập file. Môi trường chủ (tức hệ điều hành) có các chức năng này và chịu trách nhiệm thực hiện các biện pháo quản lí cho OpenGl.
* 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 lí 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 m ạng 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 dụng trên máy độc lập hay môi trường mạng). Không có lệnh OpenGL nào tiếp nhận việc nhận việc nhập dữ liệc 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 suất 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 khỏi 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 tạp OpenGL được tiến hành khi hệ thống cấp phát cửa sổ cho việc biểu diễn.

**Khối lệnh OPENGL**

**OpenGL DLL**

**Sever DLL**

**Win DLL**

**Video Driver**

*Hình 1.1 Mô hình hoạt động cơ bản của OpenGL*

**Phía Client**

**Phía Sever**

* Thao tác cơ bản OpenGL

Khối lệnh

**Danh sách**

**Bước ước lượng**

**Thao tác trên Vertex và Primitive**

**Bộ đệm khung**

**Rasteri zation**

**Thao tác trên fragme**

**Bộ nhớ texture**

**Các thao tác pixel**

*Hình 1.2 Sơ đồ sữ lý dữ liệu của OpenGL*

* 1. **Cài đặt thư viện mới trên môi trường MS Visual Studio 2019**

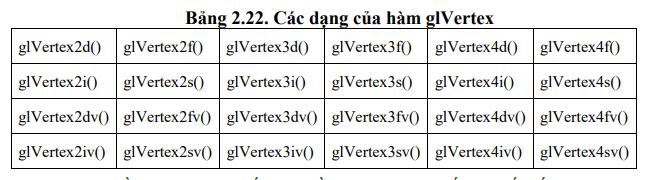
**CHƯƠNG 2: TÌM HIỂU THƯ VIỆN OPENGL**

**2.1 Cấu trúc lệnh trong OpenGL**

**2.2 Một số thuật toán vẽ hình học 2D**

**2.2.1 Vẽ điểm, đường, đa giác(POINT, LINES,POLYGON)**

* OpenGL không có sẵn các hàm để xây dựng các đối tượng hình học phức tạp, người dùng phải tự xây dựng chúng từ các đối tượng hình học cơ bản mà OpenGL hỗ trợ: điểm, đoạn thẳng, đa giác. Khai báo một điểm, dùng hàm glVertexXY với X là số chiều (2,3,4)



*Hình 1.4 Các dạng của hàm glVertex*

* Ví dụ khi ta chỉ định một điểm trong không gian 2d và 3d:
* glVertex2s(2,3);
* glVertex3d(0.0,0.0,1.0
* Các đỉnh được liệt kê giữa hai hàm:
* glBegin(tham số) /\* xác định tọa độ và màu sắc của các điểm của hình\*/
* glEnd();
* Tham số đưa vào cho hàm glBegin(thamso) sẽ giúp OpenGL quyết định vẽ gì từ các Vertex khai báo bên trong ví dụ:

glBegin(GL\_POLYGON);

glVertex3f (0.25, 0.25, 0.0);

glVertex3f (0.75, 0.25, 0.0);

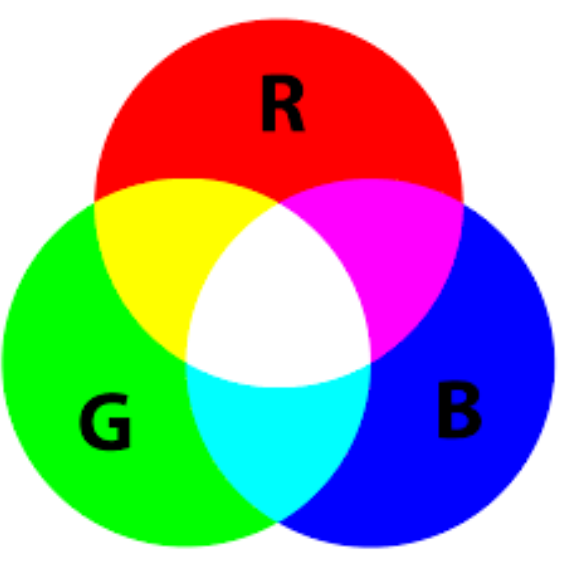
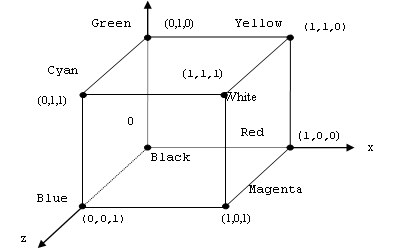
glVertex3f (0.75, 0.75, 0.0);

glVertex3f (0.25, 0.75, 0.0);

glEnd();

* + 1. **Hàm vẽ hình học phức tạp**
* Tên các hàm của GLUT đều có tiền tố glut.
* GLUT hỗ trợ sẵn một số hàm để vẽ các đối tượng hình học phức tạp hơn:
* - Vẽ hình lập phương
* + Vẽ khung: **glutWireCube** (Gldouble size);
* + Vẽ đặc: **glutSolidCube** (Gldouble size);
* - Vẽ hình cầu:
* **glutWireSphere** (GLdouble radius, GLdouble slices, GLdouble stacks);
* **glutSolidSphere** (GLdouble radius, GLdouble slices, GLdouble stacks);
* Trong đó: Radius: là bán kính hình cầu; Slices: Số đường tựa như vĩ tuyến trên hình cầu; Stacks: Số đường tựa như kinh tuyến trên hình cầu. - Vẽ hình đế hoa glutWireTorus (GLdouble innerRadius, GLdoubleouterRadius, GLdouble nsides, GLdouble rings); glutSolidTorus (GLdouble innerRadius, GLdouble outerRadius, GLdouble nsides, GLdouble rings);
* - Vẽ hình ấm pha trà **glutWireTeapot**(GLdouble size); glutSolidTeapot(GLdouble size);
* - Vẽ hình nón **glutWireCone** (GLdouble Radius, GLdouble height, GLint slices, GLint stacks); glutSolidCone (GLdouble Radius, GLdouble height, GLint slices, Glint stacks);
* - Vẽ đối tượng Quadric: đối tượng đối xứng trục và quay quanh trục, ví dụ như hình cầu, hình trụ, chóp cầu...
* + Bước 1: Khai báo đối tượng GLUquadric obj;
* + Bước 2: Tạo đối tượng 66 obj= gluNewQuadric();
* + Bước 3: Vẽ đối tượng

**2.3 Kỹ thuật tô màu**



*Hình 1.5 Hệ màu RGB*

Không gian màu RGB mô tả màu sắc bằng 3 thành phần chính là Red - Green và Blue. Không gian này được xem như một khối lập phương 3 chiều với màu red là trục x, màu Green là trục y, và màu Blue là trục z. Mỗi màu trong không gian này được xác định bởi 3 thành phần R, G, B. Ứng với các tổ hợp khác nhau của 3 màu này sẽ cho ta một màu mới.

* Gốc (0,0,0) biểu diễn màu đen (black), tọa độ (1,1,1) biểu diễn màu trắng (white), tọa độ trên các cạnh trục biểu diễn các màu cơ sở, các cạnh còn lại biểu diễn màu bù cho mỗi màu cơ sở.
* Biểu đồ RGB thuộc mô hình cộng: phát sinh màu mới bằng cách cộng cường độ màu cơ sở. Gán giá trị từ 0 đến 1 cho R,G,B.
* Màu Red là (1, 0, 0);
* Màu Blue là (0, 0, 1);
* Màu Yellow (1, 1, 0);
* Màu White (1, 1, 1);

Red + Green = Yellow Red + Green + Blue = White

* Nhận xét: Trong hình lập phương trên (hình 3.1), mỗi màu gốc (R,G,B) có các gốc đối diện là các màu bù với nó. Hai màu được gọi là bù nhau khi kết hợp hai màu này lại với nhau ra màu trắng.
* Ví dụ : Green - Magenta, Red - Cyan, Blue - Yellow. Mô hình RGB được sử dụng rộng rãi, đủ cho các ứng dụng máy tính: cảm nhận thị giác, màn hình máy tính và TV. Tuy nhiên, mô hình này không thể biểu diễn mọi màu trong phổ nhìn thấy.

**Không gian màu CMY (Cyan – Magenta – Yellow)**

Tương tự như không gian màu RGB nhưng 3 thành phần chính là Cyan – Magenta

* - Yellow. Do đó, tọa độ các màu trong không gian CMY trái ngược với không gian RGB. Ví dụ: màu White có các thành phần là (0,0,0), màu Black (1,1,1), màu Cyan (1,0,0) v.v... Không gian màu CMY là bù của không gian màu RGB.
* Tô màu trong OpenGL
* OpenGL hỗ trợ 2 chế độ màu: RGBA và Color – Index.
* Trong tài liệu này, chúng ta chỉ quan tâm đến chế độ màu RGBA. Trong chế độ màu RGBA, RGB lần lượt thể hiện màu Red, Green, Blue.
* Còn thành phần A (tức alpha) không thực sự ảnh hưởng trực tiếp lên màu pixel, người ta có thể dùng thành phần A để xác định độ trong suốt hay thông số nào đó cần quan tâm. Lệnh xóa và đặt màu là: glClearColor( ), tham số của lệnh là bộ 4 màu thành phần: đỏ, xanh lá cây, xanh dương và độ mờ (Red, Green, Blue, Alpha blending - RGBA), giá trị màu thay đổi từ: 0.0 đến 1.0.
* Ví dụ:

glClearColor(0.0, 0.0, 0.0, 0.0);

Để thiết lập màu vẽ hiện hành trong chế độ RGBA, chúng ta sử dụng các hàm sau:

glColor3{b s i f d ub us ui} (TYPEr, TYPEg, TYPEb);

glColor4{b s i f d ub us ui} (TYPEr, TYPEg, TYPEb, TYPEa);

glColor3{b s i f d ub us ui}v (const TYPE\*v);

glColor4{b s i f d ub us ui}v (const TYPE\*v);

**2.4 Phép biến đổi đồ họa 2d 3d**

* **Phép tịnh tiến (phép dịch)**
* Để di chuyển một điểm từ vị trí A tới một vị trí B trong tọa độ không gian 3d.
* Người ta có hai cách thực hiện.

+ Giữ nguyên hệ trục tọa độ và thay đổi giá trị của điểm.

+ Giữ nguyên giá trị của điểm nhưng thay đổi vị trí của hệ tọa độ. Gọi là dời hệ quy chiếu.

* Thay đổi giá trị của điểm thì không có gì khó, chúng ta chỉ việc tính toán giá trị muốn thay đổi và vẽ bình thường.
* Nhưng khi muốn thay đổi hệ trục tọa độ sang một vị trí khác thì làm thế nào.
* Opengl hỗ trợ câu lệnh sau để thực hiện

glTranslatef(x, y, z);

* Phép quay
* Phép quay trong opengl cho phép một đối tượng quay quanh một trục.
* Về bản chất, cũng là cách thay đổi hệ trục tọa độ hiện hành, nhưng không phải là phép dời trục mà là phép xoay trục.
* Xoay theo X => Mặt phẳng YOZ  quay | Xoay theo  Y => Mặt phẳng XOZ quay | Xoay theo  Z => Mặt phẳng XOY quay.
* Và câu lệnh dưới đây sẽ thực hiên việc quay.

glRotatef(angle, x, y, z);

* Ví dụ quay theo trục Z  một góc 450 thì sẽ là:   glRotatef(45.0, 0.0, 0.0, 1.0);

**2.5 Phép chiếu**

* Chiếu sáng hay còn gọi là Lighting trong OpenGL là một kỹ thuật không thể thiếu, và nó vô cùng đa dạng, sử dụng phổ biến trong mô phỏng đồ họa.

**+ Trong Hàm Init. Bật chế độ lighting**

* glEnable(GL\_LIGHTING);  
  glEnable(GL\_LIGHT0);

**Chọn vị trí ánh sáng chiếu vào:**

* GLfloat light\_pos [] = {0.0, 0.0, 1.0, 0.0};  
  glLightfv(GL\_LIGHT0, GL\_POSITION, light\_pos);
* Như thế này nghĩa là: ánh sáng chiếu theo chiều ngược với trục Z của OpenGL. Z đâm ra ngoài màn hình thì ánh sáng chiếu từ ngoài màn hình vào.
* Tương tự nếu đổi sang thành trục X và Y.

**+ Kết hợp với glMaterial tạo tính chất vật liệu cho đối tượng khi được chiếu sáng.**

* 4 Tham số chính như sau:
* **GL\_AMBIENT:**    Chiếu sáng toàn phần cho đối tượng của vật
* **GL\_DIFFUSE:**     Tạo ánh sáng khuếch tán cho đối tượng
* **GL\_SPECULAR**: Tạo ánh sáng phản xạ trong đối tượng.

**GL\_SHININESS:** Điều chỉnh cường độ điểm chiếu sáng phản xạ.

**2.6 Kỹ thuật gán cấu trúc (texture mapping)**

* Trong lập trình đồ họa thường phải giải quyết các ảnh có nhiều chi tiết hơn là ta có thể vẽ với OpenGL. Ví dụ như mặc dù có thể một hình chữ nhật thể hiện mặt bàn, nhưng đa giác được điền đầy không có các chi tiết như là thớ gỗ, để có thể thể hiện đó là mặt bàn. Để cung cấp các chi tiết như vậy, ta sử dụng các ảnh OpenGL để thực hiện gán cấu trúc (texture mapping) cho các đối tượng hay trong một số tài liệu còn được gọi kỹ thuật texture hình ảnh.
* **Các bước cơ bản trong texture mapping**
* Gán cấu trúc (texture mapping) là quá trình phức tạp, đòi hỏi sự hiểu biết về cách lưu trữ ảnh trong bộ nhớ và cách OpenGL thao tác chúng.

Để gán cấu trúc, trước hết phải khởi tạo các thông số gán cấu trúc, thiết lập môi trường gán cấu trúc, và kích hoạt việc gán cấu trúc. Ngoài ra phải truyền cấu trúc cần gán cùng các thông tin thể hiện cấu trúc đó cho OpenGL. Cũng cần phải liên kết các tọa độ cấu trúc với các vertex của đa giác được gán cấu trúc.

Như vậy, các bước để gán cấu trúc gồm có:

* Bước 1: Khai báo texture: Gluint texture\_name.
* Bước 2: Tải ảnh Bitmap và đọc thông tin Bitmap vào bộ nhớ sử dụng hàm LoadBitmap(). Kích hoạt texture sử dụng hàm glBindTexture();
* Bước 3: Định nghĩa các vertex đa giác hoặc vẽ các đối tượng.
* Bước 4: Ngừng kích hoạt: glDisable (GL\_TEXTURE\_2D).
* **Cấu trúc BITMAPINFO**

Theo sau cấu trúc BITMAPFILEHEADER là cấu trúc BITMAPINFO, được Windows địnhg nghĩa như sau:

Typef struct tagBITMAPINFO

{

BITMAPINFOHEADER bmiHeader

RGBQUAD bmiColors[1];

}

BITMAPINFO;

Cấu trúc này gồm một header, thể hiện bởi cấu trúc BITMAPINFOHEADER, và một bảng danh sách màu (color table), thể hiện bởi một mảng của cấu trúc RGBQUAD.

* **Cấu trúc BITMAPINFOHEADER**

Cấu trúc BITMAPINFOHEADER được Windows định nghĩa như sau:

Typedef struct tagBITMAPFINFOHEADER

{

DWORD biSize;

DWORD biWidth;

DWORD biHeight;

DWORD biPlanes;

DWORD biBitcount;

DWORD biCompression;

DWORD biSizeImage;

DWORD biXPelsPerMeter;

DWORD biYPelsPerMeter;

DWORD biClrUsed;

DWORD biClrImportant;

}

BITMAPFINFOHEADER;

* **Cấu trúc RGBQUAD**

Cấu trúc RGBQUAD được Windows định nghĩa như sau:

Typef struct tagRGBQUAD

{

BYTE rgbBlue;

BYTE rgbGreen;

BYTE rgbRed;

BYTE rgbReserve;

}

RGBQUAD;

* Cấu trúc này chứa cường độ các thành phần màu đỏ, xanh dương, xanh lục. Mỗi màu trong DIB được thể hiện bởi một cấu trúc RGBQUAD. Tức là bitmap 16 màu (4 bit) có một bảng danh sách màu gồm 16 cấu trúc màu RGBQUAD. Trong khi bitmap 256 màu (8 bit) có một danh sách bảng màu gồm 256 cấu trúc màu RGBQUAD. Ngoại trừ ảnh 24 bit màu thì không có bảng danh sách màu.

**PHẦN 2: VẬN DỤNG**

1. **ĐẶT VẤN ĐỀ**
   1. **Yêu cầu**

* Dựa vào kiến thức đã học, ứng dụng thư viện OpenGL hẫy xây dựng chương trình mô phỏng xe “ Ô TÔ BÁN TẢI ”

1. **PHÂN TÍCH YÊU CẦU**
   1. **Phân tích yêu cầu**

Yêu cầu đặt ra ở đây là xây dựng chương trình mô phỏng “ Ô TÔ BÁN TẢI ”

Có các thành phần cấu tạo như: Khung xe, bánh xe, kính, gương, thùng xe, đèn

* 1. **Định hướng giải quyết vấn đề**
* Xây dựng ý tưởng:
* Tạo mô hình Ô TÔ bằng cách ghép nối các khối hình riêng lẻ lại với nhau.
* Các chuyển động được dựa trên 2 nguyên tắc chính là tịnh tiến và quay quanh trục.
* Chia yêu cầu thành nhiều phần nhỏ hơn để giải quyết.
* Sử dụng các hàm đồ họa cơ bản được thư viện hỗ trợ để công việc trở nên dễ dàng hơn như: hàm glutSoliSphere để vẽ khối cầu.
* Về mặt cảm quan cần xử lý các vấn đề về phối cảnh, màu sắc, ánh sáng.

1. **CÀI ĐẶT CHƯƠNG TRÌNH**
   1. **Cấu trúc chương trình**

Cấu trúc chương trình gồm 3 phần chính:

Phần 1: Khai báo.

Phần 2: Xây dựng các hàm con và các thục tục hỗ trợ.

Phần 3: Chương trình chính

* 1. **CODE**

1. #include<Windows.h>
2. #include<iostream>
3. #include<GL/GL.h>
4. #include<GL/GLU.h>
5. #include<GL/glut.h>
6. #include<math.h>
7. void cylinder(float alas, float atas, float tinggi);
8. void kerucut(float ngisor, float nduwur, float dowo);
9. void blok(float tebal, int ratiol, int ratiop);
10. GLfloat source\_light[] = { 0.30, 0.30, 0.30, 0.50 };
11. double Loop, LoopAll = 0;
12. bool on = false;
13. void Mobil()
14. {
15. glClearColor(1.0, 1.0, 1.0, 0.0);
16. glShadeModel(GL\_SMOOTH);
17. glMatrixMode(GL\_PROJECTION);
18. glLoadIdentity();
19. gluPerspective(50.0, 1.5, 10.0, 1000.0);
20. glEnable(GL\_DEPTH\_TEST);
21. glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_FILL);
22. glEnable(GL\_LIGHTING);
23. glLightfv(GL\_LIGHT7, GL\_DIFFUSE, source\_light);
24. glEnable(GL\_LIGHT7);
25. glEnable(GL\_COLOR\_MATERIAL);
26. glColorMaterial(GL\_FRONT\_AND\_BACK, GL\_AMBIENT);
27. }
28. void keboad\_s(int key, int x, int y)
29. {
30. if (key == GLUT\_KEY\_F1)
31. {
32. if (on == false)
33. {
34. on = true;
35. }
36. else
37. {
38. on = false;
39. }
40. }
41. }
42. void display()
43. {
44. glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);
45. glMatrixMode(GL\_MODELVIEW);
46. glLoadIdentity();
47. glTranslatef(0, 0, -100);
48. Loop = +Loop + 0.1;
49. glRotatef(Loop, 1, 0, 0);
50. glPushMatrix();
51. glColor3f(0.0, 0.0, 0.0);
52. blok(10, 7, 3);
53. glTranslatef(0, 9, 0);
54. blok(10, 3, 3);
55. glPopMatrix();
56. glPushMatrix();
57. glColor3f(0.0, 0.0, 0.0);
58. glTranslatef(20, -8, -7);
59. cylinder(5, 5, 3);
60. glColor3f(0.0, 0.0, 0.0);
61. glTranslatef(-20, 8, 7);
62. glTranslatef(-5, -8, -7);
63. cylinder(5, 5, 3);
64. glColor3f(0.0, 0.0, 0.0);
65. glTranslatef(5, 8, 7);
66. glRotatef(180, 0, 180, 0);
67. glTranslatef(3, -8, -27);
68. cylinder(5, 5, 3);
69. glColor3f(0.0, 0.0, 0.0);
70. glTranslatef(-3, 8, 17);
71. glTranslatef(-22, -8, -17);
72. cylinder(5, 5, 3);
73. glColor3f(1.0, 1.0, 1.0);
74. glRotatef(90, 1, 0, 0);
75. glTranslatef(8, 2, -15);
76. blok(2, 3, 2);
77. glRotatef(90, 0, 1, 0);
78. glTranslatef(0, -0.2, 7);
79. blok(2, 4, 8);
80. // thùng xe
81. glColor3f(1.0, 0.0, 0.0);
82. glRotatef(0, 1, 0, 0);
83. glTranslatef(0, 1.2, 23);
84. blok(2, 3, 10);
85. glColor3f(0.0, 1.0, 0.0);
86. glRotatef(0, 1, 0, 0);
87. glTranslatef(0, 28, -1);
88. blok(2, 3, 10);
89. glColor3f(0.0, 0.0, 1.0);
90. glRotatef(90, 1, 0, 0);
91. glTranslatef(0, 19, 0);
92. blok(2, 3, 15);
93. // thùng xe
94. //gương
95. glRotatef(0, 1, 0, 0);
96. glTranslatef(0, -48, -5.5);
97. blok(2, 3, 5);
98. glRotatef(0, 1, 0, 0);
99. glTranslatef(0, 0, 35);
100. blok(2, 3, 2);
101. //gương
102. //kính
103. glColor3f(1.0, 1.0, 1.0);
104. glRotatef(90, 1, 0, 0);
105. glTranslatef(0, -31, -23);
106. blok(2, 4, 8);
107. glRotatef(0, 1, 0, 0);
108. glTranslatef(0, 0, 21);
109. blok(2, 3, 3);
110. // tấm để đèn
111. glColor3f(1.0, 1.0, 1.0);
112. glRotatef(90, 1, 0, 0);
113. glTranslatef(-7.5, 23, -23.999);
114. blok(2.19, 3, 9.9);
115. //kính to đầu xe
116. glColor3f(1.0, 1.0, 1.0);
117. glRotatef(0, 1, 0, 0);
118. glTranslatef(10.2, -20, -1);
119. blok(2, 3, 11);
120. //kính
121. // đèn
122. glColor3f(9.9, 9.9, 0.0);
123. glRotatef(90, 1, 0, 0);
124. glTranslatef(-10, -3, -20);
125. cylinder(2, 2, 3);
126. glColor3f(9.9, 9.9, 0.0);
127. glRotatef(0, 0, 0, 0);
128. glTranslatef(0, 25, 0);
129. cylinder(2, 2, 3);
130. // đèn
131. glPopMatrix();
132. glFlush();
133. glutSwapBuffers();
134. }
135. void Kerucut(float ngisor, float nduwur, float dowo)
136. {
137. float i;
138. glPushMatrix();
139. glTranslatef(1.0, 0.0, ngisor / 24);
140. glutSolidCone(ngisor, 0, 32, 4);
141. for (i = 0; i <= dowo; i += ngisor / 24)
142. {
143. glTranslatef(0.0, 0.0, ngisor / 24);
144. glutSolidTorus(ngisor / 4, ngisor - ((i \* (ngisor - nduwur)) / dowo), 16, 16);
145. }
146. glTranslatef(0.0, 0.0, ngisor / 4);
147. glutSolidCone(nduwur, 0, 20, 1);
148. glColor3f(0., 1., 1.);
149. glPopMatrix();
150. }
151. void cylinder(float alas, float atas, float tinggi)
152. {
153. float i;
154. glPushMatrix();
155. glTranslatef(1.0, 0.0, -alas / 8);
156. glutSolidCone(alas, 0, 32, 4);
157. for (i = 0; i <= tinggi; i += alas / 24)
158. {
159. glTranslatef(0.0, 0.0, alas / 24);
160. glutSolidTorus(alas / 4, alas - ((i \* (alas - atas)) / tinggi), 16, 16);
161. }
162. glTranslatef(0.0, 0.0, alas / 4);
163. glutSolidCone(atas, 0, 20, 1);
164. glColor3f(1., 0., 0.);
165. glPopMatrix();
166. }
167. void blok(float tebal, int ratiol, int ratiop)
168. {
169. float i, j;
170. glPushMatrix();
171. for (i = 0; i < ratiop; i++)
172. {
173. glTranslatef(-(ratiol + 1) \* tebal / 2, 0.0, 0.0);
174. for (j = 0; j < ratiol; j++)
175. {
176. glTranslatef(tebal, 0.0, 0.0);
177. glutSolidCube(tebal);
178. }
179. glTranslatef(-(ratiol - 1) \* tebal / 2, 0.0, tebal);
180. }
181. glPopMatrix();
182. }
183. void tampil()
184. {
185. display();
186. }
187. int main(int argc, char\*\* argv)
188. {
189. glutInit(&argc, argv);
190. glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB);
191. glutInitWindowPosition(10, 10);
192. glutInitWindowSize(1000, 700);
193. glutCreateWindow("");
194. glClearColor(0.5, 0.5, 0.0, 0.0);
195. glutDisplayFunc(display);
196. gluOrtho2D(-320., 320., -240.0, 240.0);
197. glutIdleFunc(tampil);
198. glutSpecialFunc(keboad\_s);
199. Mobil();
200. glutMainLoop();
201. return(0);
202. }