BÀI GIẢNG ĐỒ HOẠ MÁY TÍNH

LẬP TRÌNH 3D VỚI OPENGL

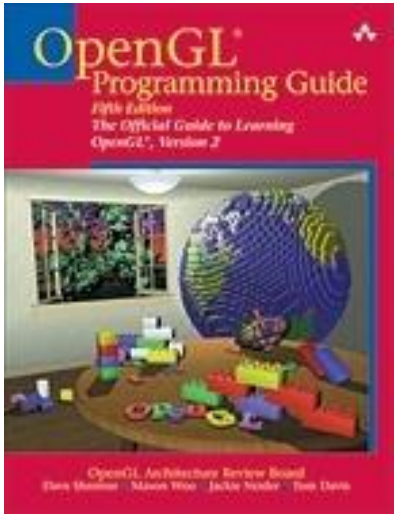
Ngô Quốc Việt - 2010

Nội dung

1. Giới thiệu OpenGL
2. Drawing
3. Hiển thị và biến đổi
4. Ánh sáng

Tài liệu tham khảo

1. Dave Shreiner, Jackie Neider, Mason Woo, Tom Davis - OpenGL Programming Guide- Addison Wesley – 1994 up to now).



2. http://glprogramming.com/red/

Mục tiêu bài giảng

- Giúp sinh viên tiếp cận với lập trình đồ hoạ 3D.
- Các bước cơ bản nhất để tạo ứng dụng 3D trên Windows.
- Hướng dẫn các khái niệm chính của lập trình 3D với OpenGL/GLUT.

OpenGL là gì

- OpenGL = Open Graphics Library
- Do Silicon Graphics (SGI) phát triển vào 1992.
- Phát triển thành chuẩn từ IrisGL (1990-cũng do SGI).
- Độc lập HĐH.
- Là dạng State Machine (mọi biến là trạng thái).
- Chỉ xử lý 3D Graphics. Không đòi hỏi platform (Windowing, Fonts, Input, GUI)

OpenGL là gì

- Giao diện phần mềm cho phần cứng đồ hoạ (~150 commands).
- Là API đồ hoạ 3 chiều
  - High-quality color images composed of geometric and image primitives
- Với OpenGL, developer phải tự tạo model thông qua các đối tượng hình học cơ bản.
- Quản lý bởi Khronos Group.

Các công nghệ 3D

- OpenGL. Phiên bản mới OpenGL 4.0.
- Scene Graphs, BSP: Open SceneGraph, Java3D, VRML, PLIB
- DirectX (Direct3D)
- Có thể kết hợp một vài phần của DirectX với OpenGL (vd: OpenGL và DirectInput trong Quake III)
- Các thư viện hỗ trợ: GL(graphics library), GLU (graphics library utilities).
- GLUT: dễ, đơn giản khi làm việc với OpenGL.

Toạ độ OpenGL

$$v = \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$$

- Sử dụng vector 4 thành phần để biểu diễn điểm.
- Được xem là hệ toạ độ thuần nhất.
- Giá trị w thường bằng 1.

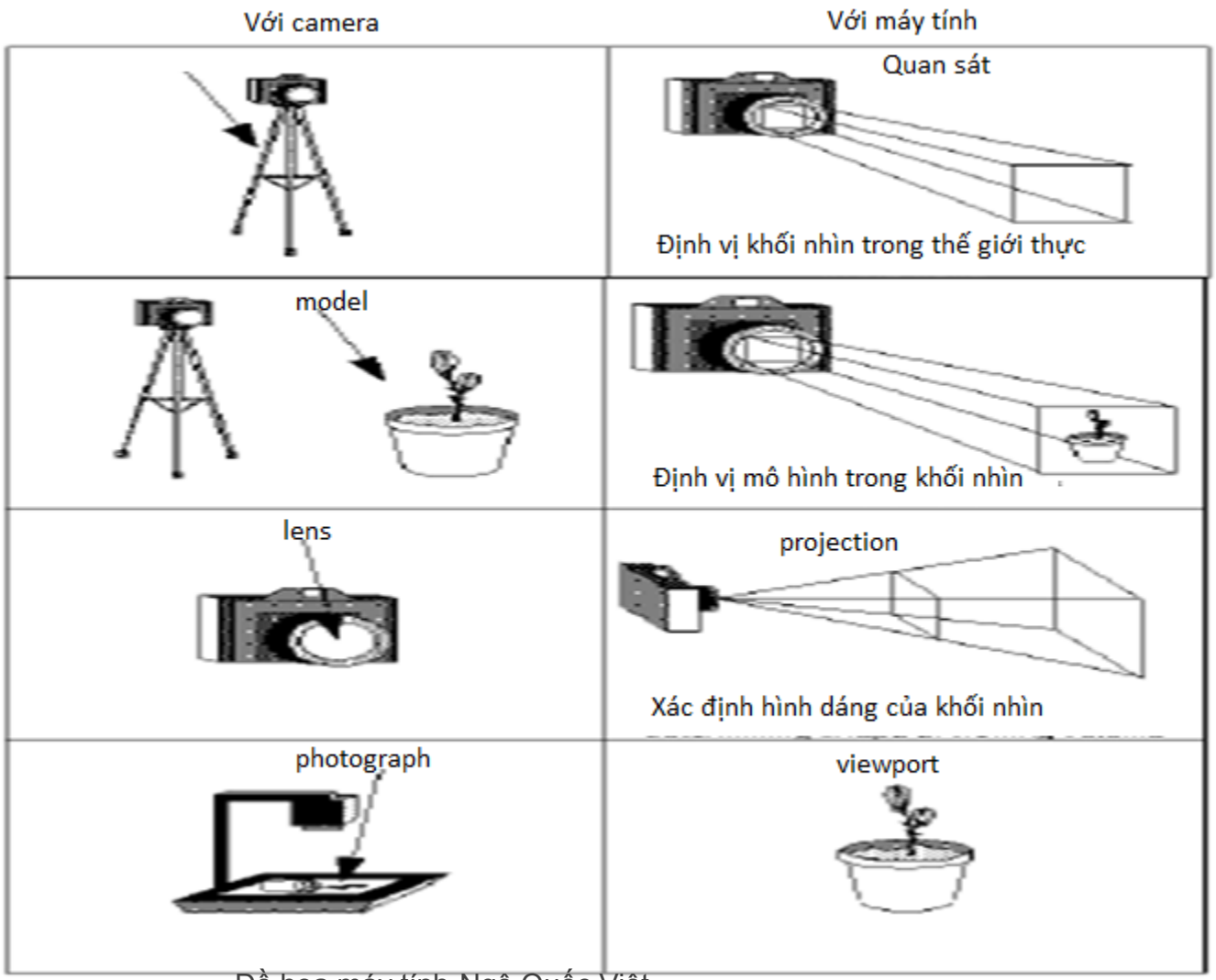
Hệ Toạ độ

- Miền giá trị màu (R, G, B, A) trong khoảng [0,1].
- Toạ độ (X, Y, Z) theo right-hand rule.
- Đơn vị hệ toạ độ: do xác lập tuỳ theo dữ liệu và ứng dụng
- Thường sử dụng World Coordinate System (miền giá trị -1 đến 1) để mô hình đối tượng.
- World Coordinate System có được do chuyển biến đổi từ toạ độ đối tượng qua ma trận ModelView.
- Toạ độ mắt được xây dựng từ phép biến đổi qua ma trận ModelView

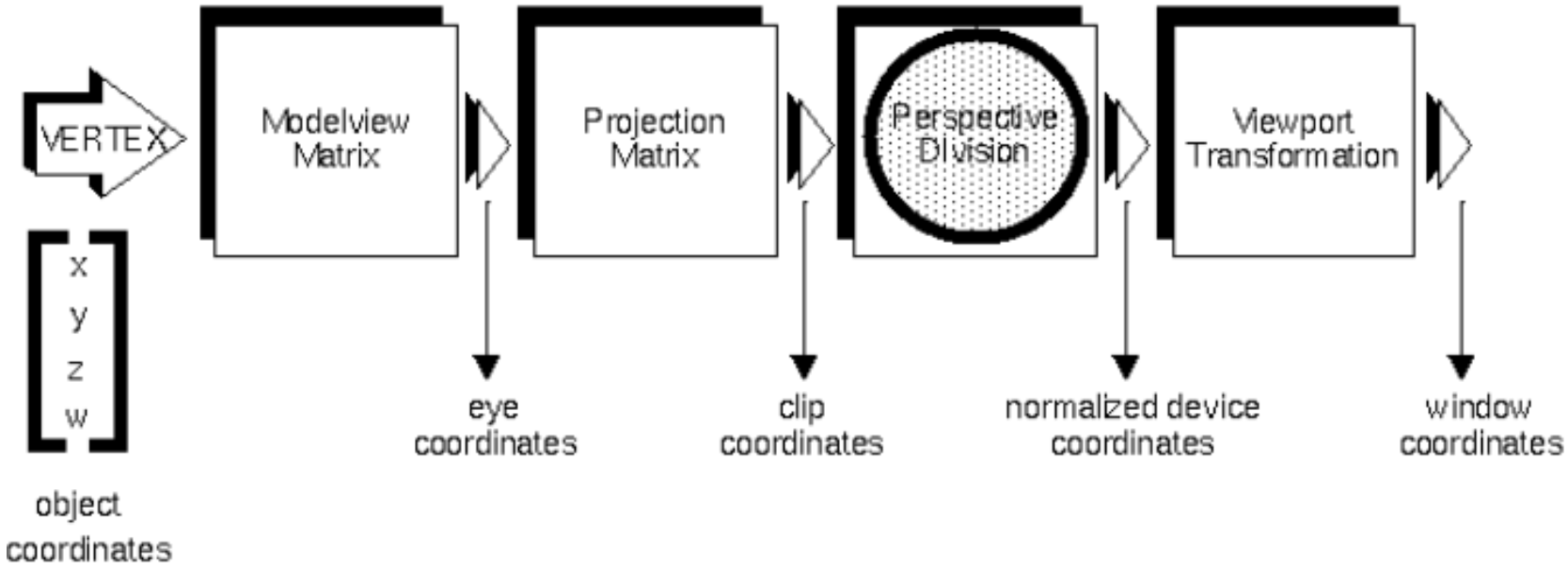
Hệ toạ độ

- Ma trận ModelView chứa cả hai phép biến đổi cho mô hình và mắt nhìn. Mắt nhìn tại gốc, với hướng nhìn dọc theo trục Z âm.
- Toạ độ xén do biến đổi toạ độ mắt nhìn qua ma trận Projection. Miền giá trị toạ độ của 3 trục từ $-W_c$ đến W_c.
- Phối cảnh thực hiện trên Clip Coordinates tạo ra Normalized Device Coordinates, với miền giá trị -1 to 1 cho cả 3 trục

Quan sát cảnh: camera



Biến đổi toạ độ



$$\mathbf{M} = \begin{bmatrix} m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ m_3 & m_7 & m_{11} & m_{15} \\ m_4 & m_8 & m_{12} & m_{16} \end{bmatrix}$$

$$v' = Mv$$

Khái niệm ma trận ModelView

- Dùng cho biến đổi vị trí camera (biến đổi vị trí quan sát)
- Dùng cho biến đổi và hướng của mô hình (vật thể cần vẽ).

```
glMatrixMode(GL_MODELVIEW);
```

Khái niệm ma trận projection

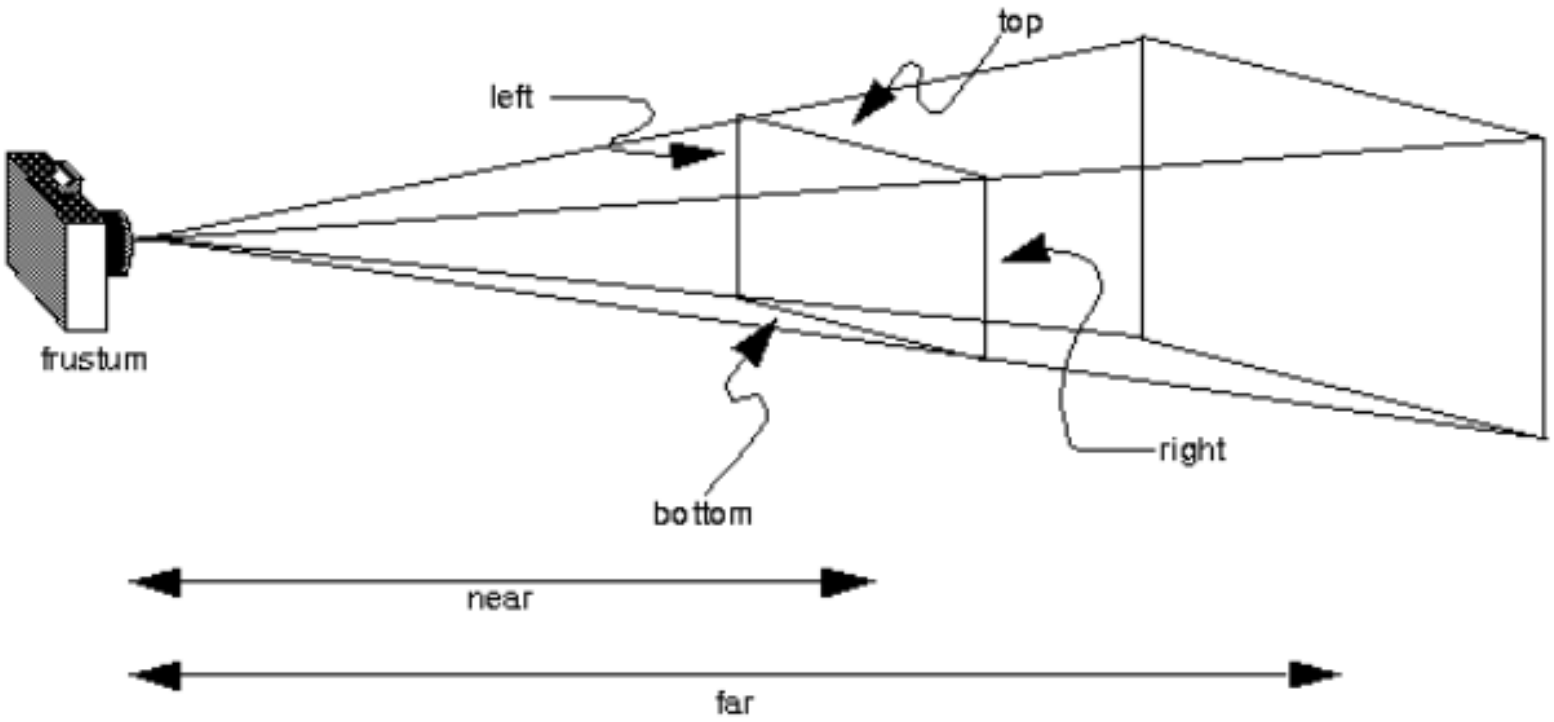
- Xác định lens cho camera.
- Xác định vùng nhìn (field of view) và các tham số khác

```
glMatrixMode(GL_PROJECTION);
```

- Có thể thực hiện dãy các biến đổi qua khái niệm matrix stack với các hàm.

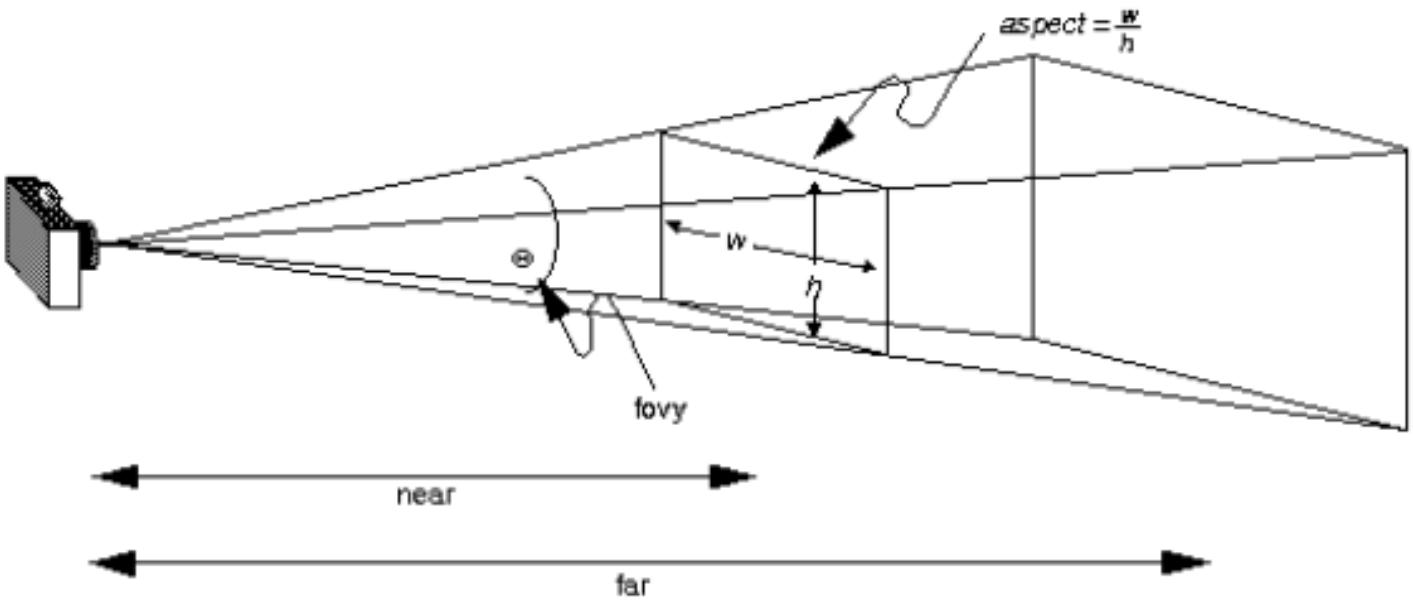
glPushMatrix() and glPopMatrix()

Chiếu phối cảnh




```
glFrustrum(left, right, bottom, top, near, far);
```

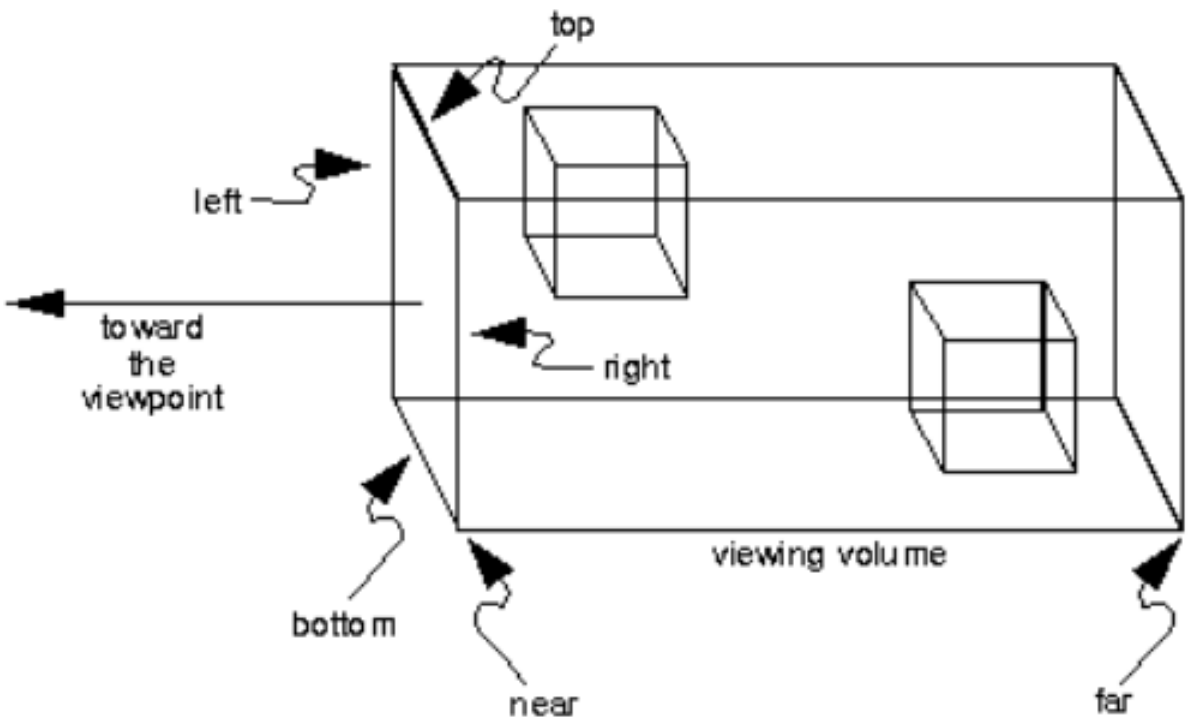
Chiếu phối cảnh với GLU




```
gluPerspective(fovy, aspect, near, far);
```

Fovy = góc (theo độ) của vùng nhìn (field of view) theo hướng y .
Aspect = tỉ lệ của vùng nhìn theo hướng x

Chiếu trực giao

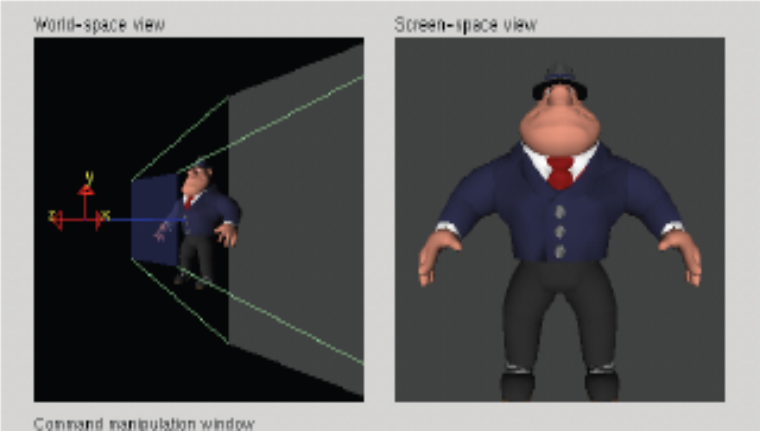



```
glOrtho(left, right, bottom, top, near, far);
```

Định vị trí camera

- Vị trí camera, điểm camera hướng đến, và vector hướng của camera

```
gluLookAt(eyex, eyey, eyez,
          centerx, centery, centerz,
          upx, upy, upz);
```



Quy ước trong OpenGL

- Functions in OpenGL start with `gl`
  - Hầu hết hàm bắt đầu **`gl`** (e.g., **`glColor()`**)
  - Một số hàm bắt đầu với **`glu`** là hàm tiện tích (e.g., **`gluLookAt()`**)
  - Hàm bắt đầu với **`glx`** là giao diện với X Windows system (e.g., in gfx.c)

Quy ước trong OpenGL

Suffix	Data Type	Typical Corresponding C-Language Type	OpenGL Type Definition
b	8-bit integer	signed char	GLbyte
s	16-bit integer	short	GLshort
i	32-bit integer	int or long	GLint, GLsizei
f	32-bit floating-point	float	GLfloat, GLclampf
d	64-bit floating-point	double	GLdouble, GLclampd
ub	8-bit unsigned integer	unsigned char	GLubyte, GLboolean
us	16-bit unsigned integer	unsigned short	GLushort
ui	32-bit unsigned integer	unsigned int or unsigned long	GLuint, GLenum, GLbitfield

Quy ước trong OpenGL

- Tên hàm chỉ ra kiểu và số lượng tham số
  - Hàm có đuôi `f` có tham số floats
  - Hàm có đuôi `i` có tham số ints
  - Hàm có đuôi `b` có tham số bytes
  - Hàm có đuôi `ub` có tham số unsigned bytes
  - Hàm có đuôi `v` có tham số array.
- Ví dụ
  - `glColor3f()` có tham số là 3 số floats
  - `glColor4fv()` có tham số mảng 4 floats

Các lệnh hợp lệ giữa glBegin và glEnd

Command	Purpose of Command
glVertex*()	set vertex coordinates
glColor*()	set current color
glIndex*()	set current color index
glNormal*()	set normal vector coordinates
glTexCoord*()	set texture coordinates
glEdgeFlag*()	control drawing of edges
glMaterial*()	set material properties
glArrayElement()	extract vertex array data
glEvalCoord*(), glEvalPoint*()	generate coordinates
glCallList(), glCallLists()	execute display list(s)

Ánh sáng trong OpenGL

- Các kiểu nguồn sáng: emit (tự phát), ambient (xung quanh), diffuse (khuếch tán), specular (phản chiếu).
  - Emit:
  - Ambient: không có nguồn cụ thể. Mọi đối tượng đều bị tác động của ánh sáng này
  - Diffuse: bởi nguồn sáng và vật liệu bề mặt.
  - Specular:
- Màu và cường độ sáng xác định bởi vector (R,G,B,A) tương tự như màu sắc.

Ánh sáng trong OpenGL

- Vị trí nguồn ánh sáng (vd: bóng đèn): xác định bởi vector (leftright, updown, frontback, 1) tương tự như một toạ độ.
  - Giá trị frontback
    - Zero: là nằm trên mặt phẳng Z.
    - Dương: nguồn sáng chiếu vào màn hình.
    - Âm: nguồn sáng chiếu từ sau vào màn hình.
  - Giá trị leftright: chạy trên trục x.
  - Giá trị updown: chạy trên trục y.
- Hàm glEnable(GL_LIGHTING) bật nguồn sáng

Ánh sáng trong OpenGL

```
GLfloat lightAmbient[] = { 0.4, 0.5, 0.0, 1.0 };
GLfloat lightDiffuse[] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat lightSpecular[] = { 1.0, 1.0, 1.0, 1.0};
GLfloat lightPosition[] = {1.0, 1.0, 1.0, 0.0};

glLightfv(GL_LIGHT0, GL_AMBIENT, lightAmbient);
glLightfv(GL_LIGHT0, GL_DIFFUSE, lightDiffuse);
glLightfv(GL_LIGHT0, GL_SPECULAR, lightSpecular);
glLightfv(GL_LIGHT0, GL_POSITION, lightPosition);

glEnable(GL_LIGHT0);
glEnable(GL_LIGHTING);
```

Vật liệu bề mặt trong OpenGL

- Độ bóng, màu phản xạ … của vật liệu ảnh hưởng tới render mô hình

```
glMaterial{if}[v](GLenum face
                  GLenum pname,
                  TYPE param);
```

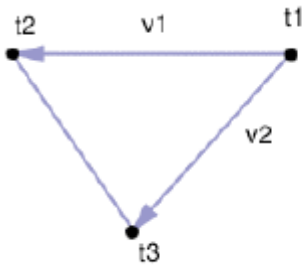
```
GLfloat material[] = {0.1, 0.5, 0.8, 1.0 };
glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE,
                                material);

GLfloat matSpecular[] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat lowShininess[] = { 5.0 };
glMaterialfv(GL_FRONT, GL_SPECULAR, matSpecular);
glMaterialfv(GL_FRONT, GL_SHININESS, lowShininess);
```

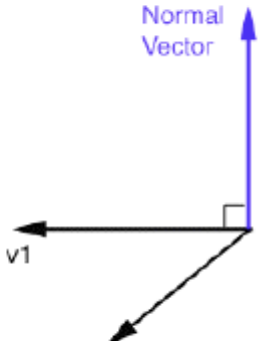
Dán texture cho bề mặt

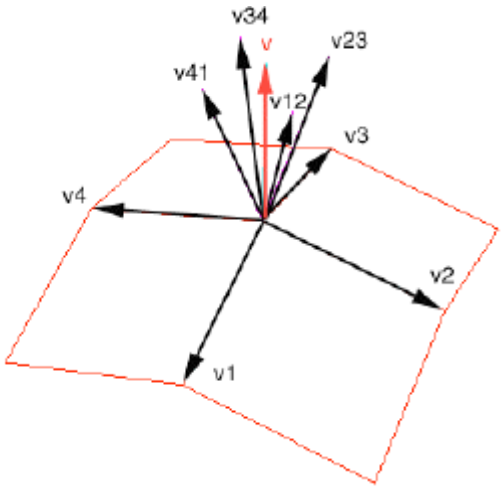
- Load bitmap làm texture. Sinh texture với glGenTextures; glBindTexture; glTexParameteri; glTexImage2D; gluBuild2DMipmaps;
- Định vị khung cho texture khi dán lên bề mặt với glTexCoord2f;

```
glNormal3f( 0.0f, 0.0f, 1.0f);
glTexCoord2f(0.0f, 0.0f); glVertex3f(-1.0f, -1.0f,  1.0f);
glTexCoord2f(1.0f, 0.0f); glVertex3f( 1.0f, -1.0f,  1.0f);
glTexCoord2f(1.0f, 1.0f); glVertex3f( 1.0f,  1.0f,  1.0f);
glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f,  1.0f,  1.0f);
// Back Face
glNormal3f( 0.0f, 0.0f,-1.0f);
glTexCoord2f(1.0f, 0.0f); glVertex3f(-1.0f, -1.0f, -1.0f);
glTexCoord2f(1.0f, 1.0f); glVertex3f(-1.0f,  1.0f, -1.0f);
glTexCoord2f(0.0f, 1.0f); glVertex3f( 1.0f,  1.0f, -1.0f);
glTexCoord2f(0.0f, 0.0f); glVertex3f( 1.0f, -1.0f, -1.0f);
// Top Face
glNormal3f( 0.0f, 1.0f, 0.0f);
glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f,  1.0f, -1.0f);
glTexCoord2f(0.0f, 0.0f); glVertex3f(-1.0f,  1.0f,  1.0f);
glTexCoord2f(1.0f, 0.0f); glVertex3f( 1.0f,  1.0f,  1.0f);
glTexCoord2f(1.0f, 1.0f); glVertex3f( 1.0f,  1.0f, -1.0f);
// Bottom Face
glNormal3f( 0.0f,-1.0f, 0.0f);
glTexCoord2f(1.0f, 1.0f); glVertex3f(-1.0f, -1.0f, -1.0f);
glTexCoord2f(0.0f, 1.0f); glVertex3f( 1.0f, -1.0f, -1.0f);
glTexCoord2f(0.0f, 0.0f); glVertex3f( 1.0f, -1.0f,  1.0f);
glTexCoord2f(1.0f, 0.0f); glVertex3f(-1.0f, -1.0f,  1.0f);
// Right face
glNormal3f( 1.0f, 0.0f, 0.0f);
glTexCoord2f(1.0f, 0.0f); glVertex3f( 1.0f, -1.0f, -1.0f);
glTexCoord2f(1.0f, 1.0f); glVertex3f( 1.0f,  1.0f, -1.0f);
glTexCoord2f(0.0f, 1.0f); glVertex3f( 1.0f,  1.0f,  1.0f);
glTexCoord2f(0.0f, 0.0f); glVertex3f( 1.0f, -1.0f,  1.0f);
// Left Face
glNormal3f(-1.0f, 0.0f, 0.0f);
glTexCoord2f(0.0f, 0.0f); glVertex3f(-1.0f, -1.0f, -1.0f);
glTexCoord2f(1.0f, 0.0f); glVertex3f(-1.0f, -1.0f,  1.0f);
glTexCoord2f(1.0f, 1.0f); glVertex3f(-1.0f,  1.0f,  1.0f);
glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f,  1.0f, -1.0f);
```

Vector chuẩn








```
glBegin(GL_POLYGON)
        glNormal3fv(n0);
        glVertex3fv(v0);
        glNormal3fv(n1);
        glVertex3fv(v1);
        glNormal3fv(n2);
        glVertex3fv(v2);
        glNormal3fv(n3);
        glVertex3fv(v30);
glEnd();
```

OpenGL và GLUT

- GLUT (OpenGL Utility Toolkit).
  - Thư viện bổ sung
    - Là windowing API
    - Không phải là thành phần của OpenGL
  - Các tác vụ chính
    - Tạo cửa sổ.
    - Đăng ký sự kiện.
      - Mouse buttons, movement, keyboard, etc…
    - Viết các hàm Callbacks xử lý sự kiện.

Cách cài đặt GLUT

- Download GLUT: http://www.opengl.org/resources/libraries/glut.html
- Kết hợp với Visual C++. Copy files vào các folder sau:
  - glut.h → VC/include/gl/
  - glut32.lib → VC/lib/
  - glut32.dll → windows/system32/
- Header Files, luôn đưa các include sau vào đầu file header có sử dụng OpenGL.
  - #include <GL/glut.h>
- Lib Files: Glu32.lib; GLaux.lib và OpenGL32.lib (nếu là GLUT cũ).

Chương trình mẫu

```
#include <GL/glut.h>
#include <GL/gl.h>

void main(int argc, char** argv)
{
    int mode = GLUT_RGB|GLUT_DOUBLE;
    glutInitDisplayMode( mode );
    glutInitWindowSize( 500,500 );
    glutCreateWindow( "Simple" );
    init();
    glutDisplayFunc( display );
    glutKeyboardFunc( key );
    glutMainLoop();
}
```

Specify the display Mode – RGB or color Index, single or double Buffer

Create a window Named "simple" with resolution 500 x 500

Your OpenGL initialization code (Optional)

Register your call back functions

Infinite loop waiting for events

Khởi tạo OpenGL

```
void init( void )
{
   glClearColor (0.0, 0.0, 0.0, 0.0);
   glViewport(0, 0, width, height);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   glOrtho(-10, 10, -10, 10, -10, 20);
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();

   glEnable( GL_LIGHT0 );
   glEnable( GL_LIGHTING );
   glEnable( GL_DEPTH_TEST );
}
```

Xác lập hình chiếu của camera. Phối cảnh.

Hệ toạ độ trực giao

Định vị và hướng của camera

Ma trận đơn vị 4x4

Ma trận 4x4 được xem như phép biến đổi

Các hàm GLUT Callback

- **Điều khiển sự kiện:** Chương trình sử dụng windows
  - Input/Output
  - Chờ sự kiện nhất định xuất hiện, sau đó thực hiện yêu cầu tuỳ theo sự kiện

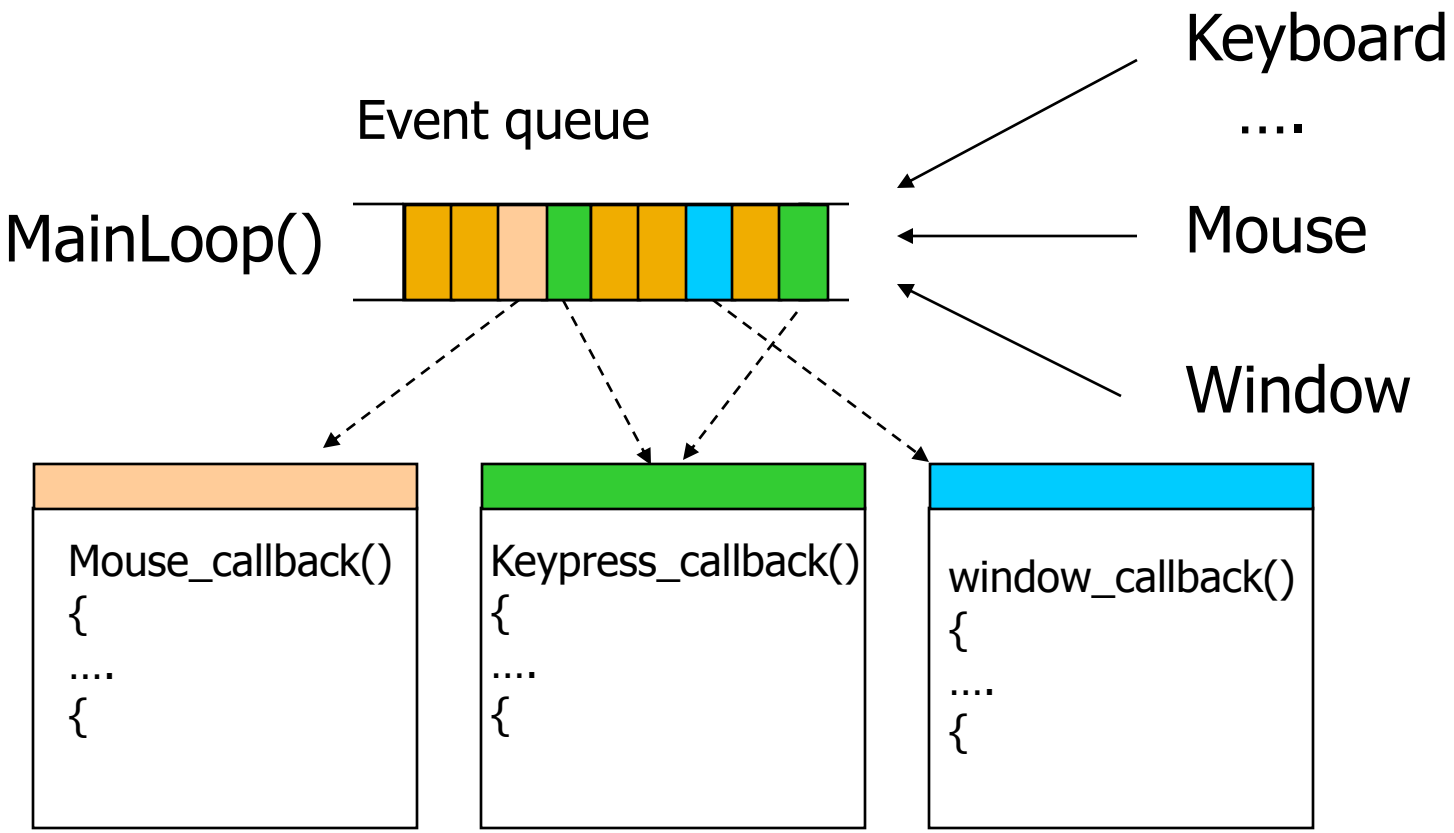
- **Sự kiện** – key press, mouse button press and release, window resize, etc.
- *Chương trình OpenGL luôn ở trong vòng lặp*

GLUT Callback Functions

- **Hàm Callback** : thủ tục được gọi khi event xuất hiện
  - Window resize hay redraw
  - User input (mouse, keyboard)
  - Animation (render many frames)

- "Đăng ký" callbacks trong GLUT với các hàm
  - glutDisplayFunc(my_display_func);
  - glutIdleFunc(my_idle_func);
  - glutKeyboardFunc(my_key_events_func);
  - glutMouseFunc (my_mouse_events_func);

Hàng chờ Event



Rendering Callback

- Hàm Callback xử lý mọi thao tác vẽ
- Mọi chương trình GLUT phải có hàm callback hiển thị.

- glutDisplayFunc(***my_display_func***); /* this part is in main.c */

```
void my_display_func (void )
{
   glClear( GL_COLOR_BUFFER_BIT );
   glBegin( GL_TRIANGLE );
     glVertex3fv( v[0] );
     glVertex3fv( v[1] );
     glVertex3fv( v[2] );
   glEnd();
   glFlush();
}
```

Idle Callback

- Dùng cho animation và cập nhật liên tục
  - Có thể dùng *glutTimerFunc* hoặc *callbacks timer* cho animations
- glutIdleFunc(***idle***);

```
void idle( void )
{
  /* change something */
  t += dt;
  glutPostRedisplay();
}
```

Hàm Callbacks cho User Input

- Xử lý user input
- glutKeyboardFunc(my_key_events);

```
void my_key_events (char key, int x, int y )
{
   switch ( key ) {
     case 'q' : case 'Q' :
        exit ( EXIT_SUCCESS);
        break;
     case 'r' : case 'R' :
        rotate = GL_TRUE;
        break;
   }
}
```

Mouse Callback

- Bắt sự kiện mouse press và release.
- glutMouseFunc(my_mouse);

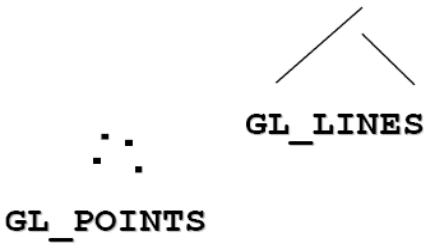
```
void myMouse(int button, int state, int x, int y)
{
    if (button == GLUT_LEFT_BUTTON && state ==
      GLUT_DOWN)
    {
    …
    }
}
```

Các sự kiện trong OpenGL

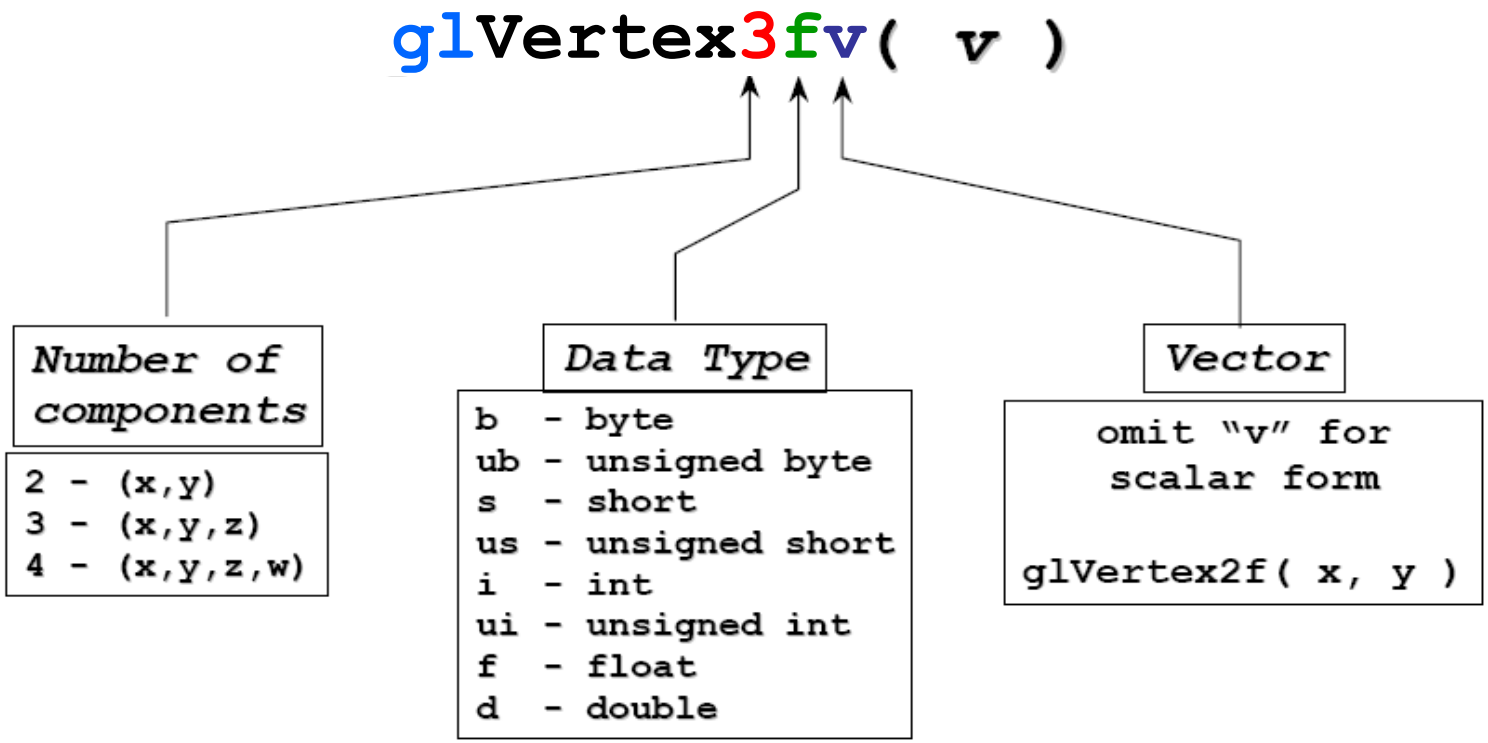
Event	Example	OpenGL Callback Function
Keypress	KeyDown KeyUp	`glutKeyboardFunc`
Mouse	leftButtonDown leftButtonUp	`glutMouseFunc`
Motion	With mouse press Without	`glutMotionFunc` `glutPassiveMotionFunc`
Window	Moving Resizing	`glutReshapeFunc`
System	Idle Timer	`glutIdleFunc` `glutTimerFunc`
Software	What to draw	`glutDisplayFunc`

Các đối tượng hình học trong OpenGL

- Thực thể hình học được xác định bởi các đỉnh 3D.
- Có 10 kiểu thực thể hình học:



Định dạng lệnh OpenGL



Các đỉnh và đối tượng

- Cách thực hiện được xác định bởi

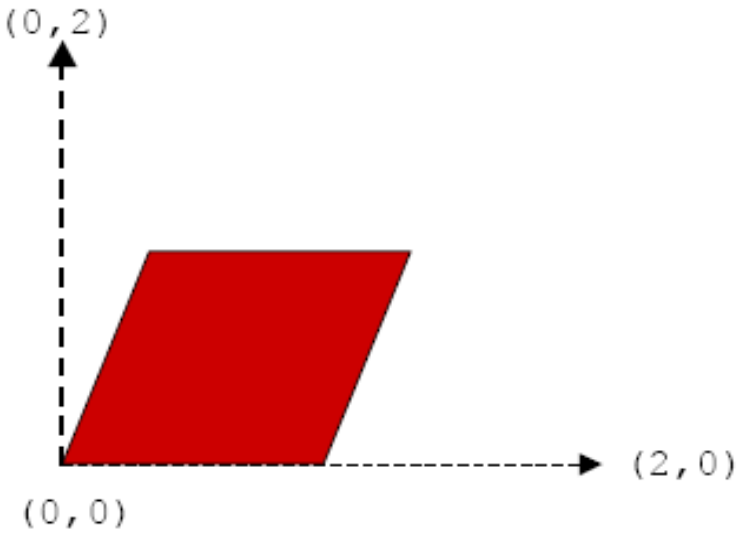
```
glBegin( primType );
…
glEnd();
```

  - *primType* xác định cách kết hợp các đỉnh

```
GLfloat red, green, blue;
Glfloat coords[nVerts][3];
/*Initialize coords and colors somewhere in program*/
glBegin( primType );
for ( i = 0; i < nVerts; ++i ) {
     glColor3f( red, green, blue );
     glVertex3fv( coords[i] );
}
glEnd();
```

Ví dụ

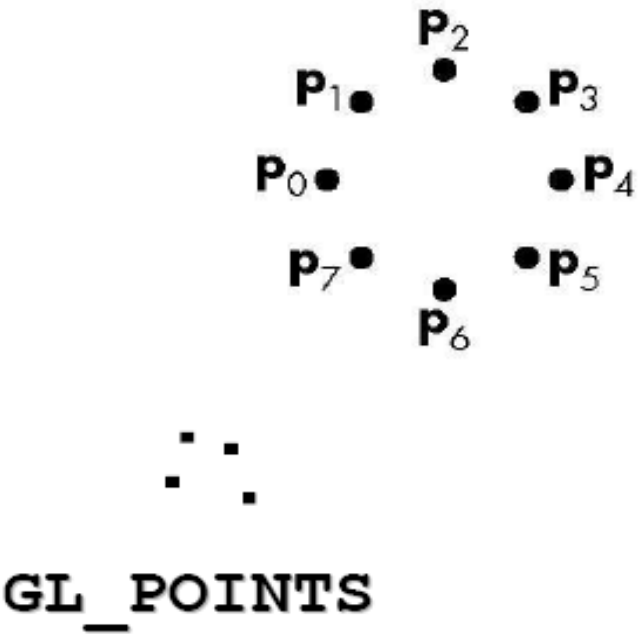
```
void drawParallelogram( GLfloat
   color[] )
{
   glBegin( GL_QUADS );
   glColor3fv( color );
   glVertex2f( 0.0, 0.0 );
   glVertex2f( 1.0, 0.0 );
   glVertex2f( 1.5, 1.118 );
   glVertex2f( 0.5, 1.118 );
   glEnd();
}
```



Các đỉnh và đối tượng

- Points, **GL_POINTS**
  - Điểm rời rạc
  - Kích thước điểm có thể thay đổi
    - *glPointSize (float size)*

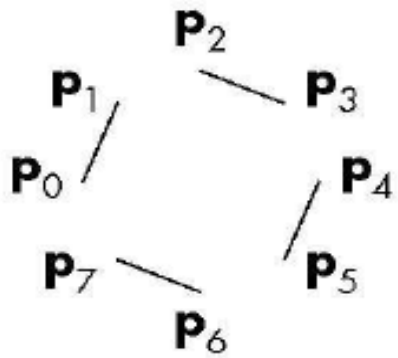
```
glBegin(GL_POINTS);
glColor3fv( color );
glVertex2f( P0.x, P0.y );
glVertex2f( P1.x, P1.y );
glVertex2f( P2.x, P2.y );
glVertex2f( P3.x, P3.y );
glVertex2f( P4.x, P4.y );
glVertex2f( P5.x, P5.y );
glVertex2f( P6.x, P6.y );
glVertex2f( P7.x, P7.y );
glEnd();
```



VD: vẽ đoạn thẳng giữa 2 đỉnh

- Lines, **GL_LINES**
  - Nối giữa 2 đỉnh: ví dụ sau vẽ 8 đoạn thẳng.
  - Có thể chỉ ra độ dày với.
    - *glLineWidth (float width)*

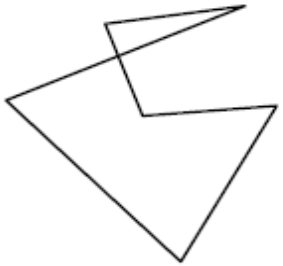
```
glBegin(GL_LINES);
glColor3fv( color );
glVertex2f( P0.x, P0.y );
glVertex2f( P1.x, P1.y );
glVertex2f( P2.x, P2.y );
glVertex2f( P3.x, P3.y );
glVertex2f( P4.x, P4.y );
glVertex2f( P5.x, P5.y );
glVertex2f( P6.x, P6.y );
glVertex2f( P7.x, P7.y );
glEnd();
```



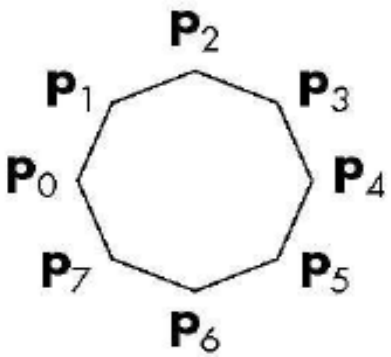
GL_LINES

Vẽ polyline khép kín

- Line Loop, **GL_LINE_LOOP**

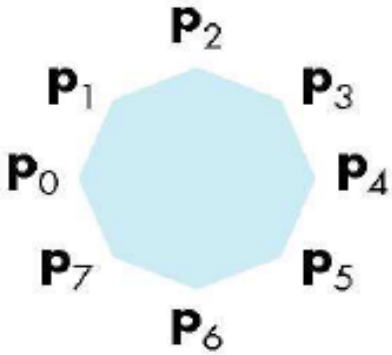


GL_LINE_LOOP



- Sử dụng **GL_POLYGON**

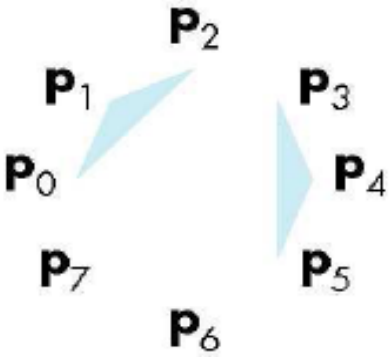
GL_POLYGON



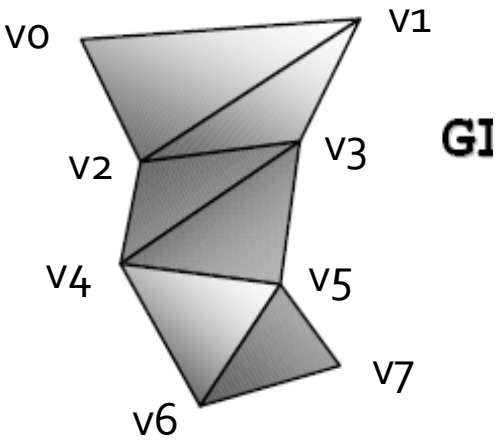
VD: vẽ tam giác

- Triangles , **GL_TRIANGLES**

GL_TRIANGLES



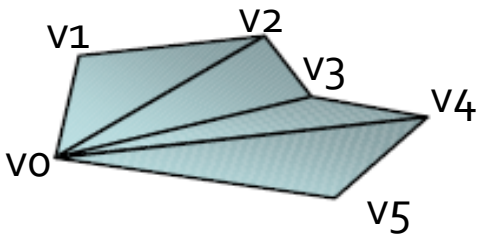
- Sử dụng **GL_TRIANGLE_STRIP**



GL_TRIANGLE_STRIP

VD: vẽ quạt tam giác

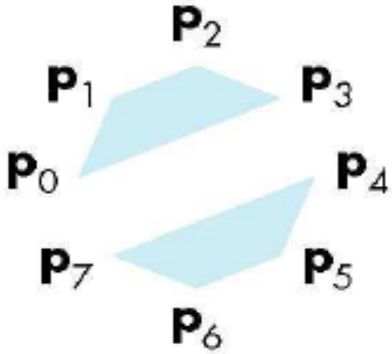
- Sử dụng, **GL_TRIANGLE_FAN**



GL_TRIANGLE_FAN

- Sử dụng **GL_QUADS với bộ 4 đỉnh**

GL_QUADS



Thay đổi màu

- Màu theo bộ 4 (Red, Green, Blue, Alpha)

```
glColor4f(red, green, blue, alpha);
glColor3f(red, green, blue);
glColor3f(0.0, 0.0, 0.0); /* Black */
glColor3f(1.0, 0.0, 0.0); /* Red */
glColor3f(0.0, 1.0, 0.0); /* Green */
glColor3f(1.0, 1.0, 0.0); /* Yellow */
glColor3f(1.0, 0.0, 1.0); /* Magenta */
glColor3f(1.0, 1.0, 1.0); /* White */)
```

gluPerspective

- Xác lập ma trận chiếu phối cảnh

  gluPerspective(GLdouble fovy, GLdouble aspect,
  GLdouble zNear, GLdouble zFar)

- fovy: liếc theo góc thẳng đứng; aspect: tỉ lệ giữa ngang và dọc; zNear, zFar: khoảng xén
- Ma trận chiếu xác định như sau

$$f = cotangent\left(\frac{fovy}{2}\right)$$

$$\begin{pmatrix} \frac{f}{aspect} & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & \frac{zFar+zNear}{zNear-zFar} & \frac{2\times zFar\times zNear}{zNear-zFar} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

Chương trình mẫu: Dán bitmap tại các vị trí khác nhau

- Tạo texture (load bitmap, sinh texture).
- Vẽ quads đã bind bitmap texture tại các vị trí texture.
- Có sử dụng hàm glColor4ub (4 bytes màu).
- Dùng các hàm glRotate, glTranslatef để vẽ quad có dán texture.

Hiển thị TEXT

- Xác lập màu

```
glColor3f(1.0f*float(cos(cnt1)),1.0f*float(sin(cnt2)),1.0f-0.5f*float(cos(cnt1+cnt2)));
```

- Định vị trí dòng text

```
glRasterPos2f(-0.45f+0.05f*float(cos(cnt1)), 0.32f*float(sin(cnt2)));
```

- Hiển thị Text

```
glPrint("Active OpenGL Text With Me - %7.2f", cnt1);
```

Câu hỏi ngắn

- Làm sao vẽ hình chữ nhật phủ toàn bộ cửa sổ viewport?
- Set cả modelView và ProjectionView là Identity
- Ví dụ sau vẽ tại zNear=-1

```
glMatrixMode (GL_MODELVIEW);
glPushMatrix ();
glLoadIdentity ();
glMatrixMode (GL_PROJECTION);
glPushMatrix ();
glLoadIdentity ();
glBegin (GL_QUADS);
glVertex3i (-1, -1, -1); glVertex3i (1, -1, -1); glVertex3i (1, 1, -1); glVertex3i (-1, 1, -1);
glEnd ();
glPopMatrix (); glMatrixMode (GL_MODELVIEW);
glPopMatrix ();
```

Câu hỏi ngắn

- Làm cách nào giữ nguyên tỉ lệ phối cảnh khi kích thước cửa sổ thay đổi.

```
glMatrixMode(GL_PROJECTION);        glLoadIdentity();
gluPerspective(fov,
(float)windowWidth/(float)windowHeight, zNear,   zFar);
```

- Hoặc

```
float cx, halfWidth = windowWidth*0.5f; float aspect =
        (float)windowWidth/(float)windowHeight;
glMatrixMode(GL_PROJECTION);        glLoadIdentity();
glFrustum(cx-halfWidth*aspect,
cx+halfWidth*aspect, bottom, top, zNear, zFar);
```