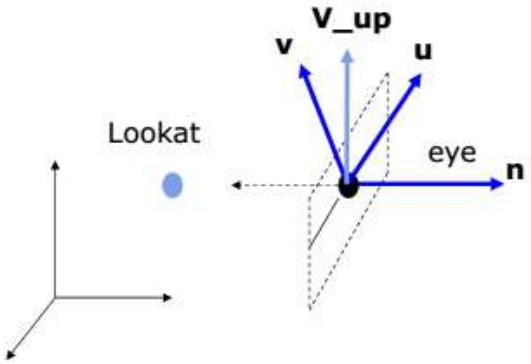
# BÀI THỰC HÀNH SỐ 6 - PHÉP CHIẾU

## A- HƯỚNG DẪN

1. **Hàm LookAt** ( trong mat.h)

**a) Mục đích:**

- Thiết lập camera
- Chuyển tọa độ từ hệ tọa độ thế giới thực sang hệ tọa độ camera (hoặc mắt - eye)




**b) Thiết lập camera**

- Eye: vị trí đặt camera trong hệ tọa độ thế giới thực.
- At (hoặc LookAt): điểm camera (eye) nhìn tới, tọa độ trong hệ tọa độ thế giới thực.
- Up ( hoặc V_up): tọa độ chỉ ra up-vector của camera.

**c) Chuyển từ hệ tọa độ thế giới thực sang hệ tọa độ camera (eye)**

- Gốc tọa độ: Eye.x, Eye.y, Eye.z
- Các vector cơ sở:
  - n = (eye – Lookat) / | eye – Lookat|
  - u = (V_up x n) / | V_up x n|
  - v = n x u

- Ma trận biến đổi:

$$Mw2e = \begin{vmatrix} ux & uy & ux & 0 \\ vx & vy & vz & 0 \\ nx & ny & nz & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} 1 & 0 & 0 & -ex \\ 0 & 1 & 0 & -ey \\ 0 & 0 & 1 & -ez \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

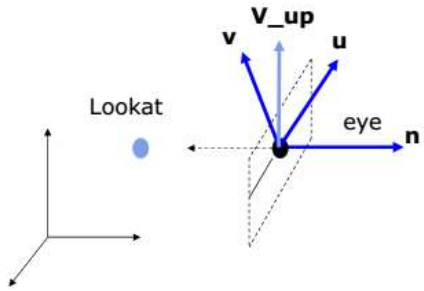
Rotation Translation

$$= \begin{vmatrix} ux & uy & uz & -\mathbf{e}\cdot\mathbf{u} \\ vx & vy & vz & -\mathbf{e}\cdot\mathbf{v} \\ nx & ny & nz & -\mathbf{e}\cdot\mathbf{n} \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

Multiplied together = lookAt transform

### d) Code hàm LookAt

```
mat4 LookAt( const vec4& eye, const vec4& at, const vec4& up )
{
    vec4 n1 = eye - at;
    n1.w = 0;
    vec4 n = normalize(n1);
    vec4 u = normalize(vec4(cross(up,n), 0));
    vec4 v = normalize(vec4(cross(n,u), 0));
    vec4 t = vec4(0.0, 0.0, 0.0, 1.0);
    mat4 c = mat4(u, v, n, t);
    return c * Translate( -eye );
}
```



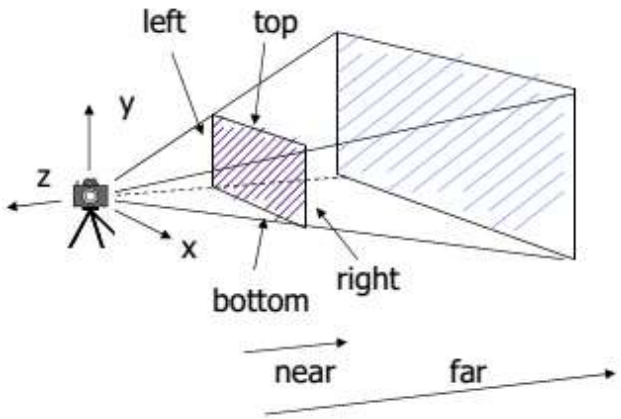

### 2. Hàm orthor

$$\mathbf{P} = \mathbf{ST} = \begin{bmatrix} \frac{2}{right-left} & 0 & 0 & -\frac{right+left}{right-left} \\ 0 & \frac{2}{top-bottom} & 0 & -\frac{top+bottom}{top-bottom} \\ 0 & 0 & \frac{2}{near-far} & -\frac{far+near}{far-near} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

```
mat4 Ortho( const GLfloat left, const GLfloat right,
            const GLfloat bottom, const GLfloat top,
            const GLfloat zNear, const GLfloat zFar )
{
    mat4 c;
    c[0][0] = (GLfloat) 2.0/(right - left);
    c[1][1] = (GLfloat)2.0/(top - bottom);
    c[2][2] = (GLfloat)2.0/(zNear - zFar);
    c[3][3] = (GLfloat)1.0;
    c[0][3] = -(right + left)/(right - left);
    c[1][3] = -(top + bottom)/(top - bottom);
    c[2][3] = -(zFar + zNear)/(zFar - zNear);
    return c;
}

//---------------------------
mat4 Ortho2D( const GLfloat left, const GLfloat right,
              const GLfloat bottom, const GLfloat top )
{
    return Ortho( left, right, bottom, top, -1.0, 1.0 );
}
```

### 3. Hàm Frustum




- **Ma trận biến đổi:**

**Scale** **Translate** **Previous Perspective Transform Matrix**

$$\begin{pmatrix} \frac{2}{right-left} & 0 & 0 & 0 \\ 0 & \frac{2}{top-bottom} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} 1 & 0 & 0 & -(right+left)/2 \\ 0 & 1 & 0 & -(top+bottom)/2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} N & 0 & 0 & 0 \\ 0 & N & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

$$\Rightarrow \begin{pmatrix} \frac{2N}{right-left} & 0 & \frac{right+left}{right-left} & 0 \\ 0 & \frac{2N}{top-bottom} & \frac{top+bottom}{top-bottom} & 0 \\ 0 & 0 & \frac{-(F+N)}{F-N} & \frac{-2FN}{F-N} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

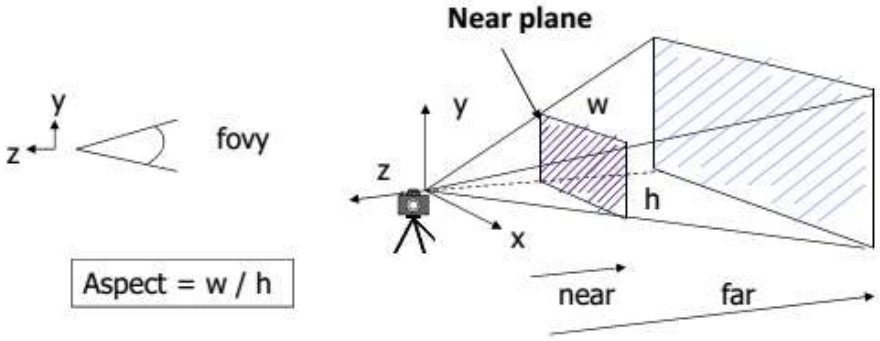
**Final Perspective Transform Matrix**

**glFrustum(left, right, bottom, top, N, F)** N = near plane, F = far plane

- **Code:**

```
mat4 Frustum( const GLfloat left, const GLfloat right,
              const GLfloat bottom, const GLfloat top,
              const GLfloat zNear, const GLfloat zFar )
{
    mat4 c;
    c[0][0] = (GLfloat) 2.0*zNear/(right - left);
    c[0][2] = (right + left)/(right - left);
    c[1][1] = (GLfloat) 2.0*zNear/(top - bottom);
    c[1][2] = (top + bottom)/(top - bottom);
    c[2][2] = -(zFar + zNear)/(zFar - zNear);
    c[2][3] = (GLfloat)-2.0*zFar*zNear/(zFar - zNear);
    c[3][2] = (GLfloat)-1.0;
    return c;
}
```

## 4. Hàm Perspective




- **Ma trận biến đổi:**

$$\mathbf{P} = \mathbf{NSH} = \begin{bmatrix} \frac{near}{right} & 0 & 0 & 0 \\ 0 & \frac{near}{top} & 0 & 0 \\ 0 & 0 & \frac{-far+near}{far-near} & \frac{-2far*near}{far-near} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

- **Code:**

```
mat4 Perspective( const GLfloat fovy, const GLfloat aspect,
                  const GLfloat zNear, const GLfloat zFar)
{
    GLfloat top   = tan(fovy*DegreesToRadians/2) * zNear;
    GLfloat right = top * aspect;

    mat4 c;
    c[0][0] = zNear/right;
    c[1][1] = zNear/top;
    c[2][2] = -(zFar + zNear)/(zFar - zNear);
    c[2][3] = (GLfloat)-2.0*zFar*zNear/(zFar - zNear);
    c[3][2] = (GLfloat)-1.0;
    return c;
}
```

## B- VÍ DỤ

Thiết lập phép chiếu phối cảnh bằng hàm Frustum để quan sát hình lập phương đơn vị.

### a) Chương trình file .cpp

```
//Chương trình vẽ 1 hình lập phương đơn vị theo mô hình lập trình OpenGL hiện đại

#include "Angel.h"
```

```
// remember to prototype
void generateGeometry( void );
void initGPUBuffers( void );
void shaderSetup( void );
void display( void );
//void keyboard( unsigned char key, int x, int y );


typedef vec4 point4;
typedef vec4 color4;
using namespace std;

// Số các đỉnh của các tam giác
const int NumPoints = 36;

point4 points[NumPoints]; /* Danh sách các đỉnh của các tam giác cần vẽ*/
color4 colors[NumPoints]; /* Danh sách các màu tương ứng cho các đỉnh trên*/


point4 vertices[8]; /* Danh sách 8 đỉnh của hình lập phương*/
color4 vertex_colors[8]; /*Danh sách các màu tương ứng cho 8 đỉnh hình lập
phương*/

GLuint program;

// Các tham số cho viewing
GLfloat radius = 1, theta = 0, phi = 0;

const GLfloat dr = 5.0 * DegreesToRadians;
GLuint model_view_loc;

// Các tham số cho projection
GLfloat l = -1.0, r = 1.0;
GLfloat bottom = -1.0, top = 1.0;
GLfloat zNear = 0.5, zFar = 3.0;
GLuint projection_loc;

void initCube()
{
    // Gán giá trị tọa độ vị trí cho các đỉnh của hình lập phương
    vertices[0] = point4(-0.5, -0.5, 0.5, 1.0);
    vertices[1] = point4(-0.5, 0.5, 0.5, 1.0);
    vertices[2] = point4(0.5, 0.5, 0.5, 1.0);
    vertices[3] = point4(0.5, -0.5, 0.5, 1.0);
    vertices[4] = point4(-0.5, -0.5, -0.5, 1.0);
    vertices[5] = point4(-0.5, 0.5, -0.5, 1.0);
    vertices[6] = point4(0.5, 0.5, -0.5, 1.0);
    vertices[7] = point4(0.5, -0.5, -0.5, 1.0);

    // Gán giá trị màu sắc cho các đỉnh của hình lập phương
    vertex_colors[0] = color4(0.0, 0.0, 0.0, 1.0); // black
    vertex_colors[1] = color4(1.0, 0.0, 0.0, 1.0); // red
    vertex_colors[2] = color4(1.0, 1.0, 0.0, 1.0); // yellow
    vertex_colors[3] = color4(0.0, 1.0, 0.0, 1.0); // green
    vertex_colors[4] = color4(0.0, 0.0, 1.0, 1.0); // blue
    vertex_colors[5] = color4(1.0, 0.0, 1.0, 1.0); // magenta
    vertex_colors[6] = color4(1.0, 0.7, 0, 1.0); // orange
    vertex_colors[7] = color4(0.0, 1.0, 1.0, 1.0); // cyan
}
int Index = 0;
void quad(int a, int b, int c, int d)  /*Tạo một mặt hình lập phương = 2 tam giác,
gán màu cho mỗi đỉnh tương ứng trong mảng colors*/
```

```
{
    colors[Index] = vertex_colors[a]; points[Index] = vertices[a]; Index++;
    colors[Index] = vertex_colors[a]; points[Index] = vertices[b]; Index++;
    colors[Index] = vertex_colors[a]; points[Index] = vertices[c]; Index++;
    colors[Index] = vertex_colors[a]; points[Index] = vertices[a]; Index++;
    colors[Index] = vertex_colors[a]; points[Index] = vertices[c]; Index++;
    colors[Index] = vertex_colors[a]; points[Index] = vertices[d]; Index++;
}
void makeColorCube(void)  /* Sinh ra 12 tam giác: 36 đỉnh, 36 màu*/

{
    quad(1, 0, 3, 2);
    quad(2, 3, 7, 6);
    quad(3, 0, 4, 7);
    quad(6, 5, 1, 2);
    quad(4, 5, 6, 7);
    quad(5, 4, 0, 1);
}
void generateGeometry( void )
{
    initCube();
    makeColorCube();
}


void initGPUBuffers( void )
{
    // Tạo một VAO - vertex array object
    GLuint vao;
    glGenVertexArrays( 1, &vao );
    glBindVertexArray( vao );

    // Tạo và khởi tạo một buffer object
    GLuint buffer;
    glGenBuffers( 1, &buffer );
    glBindBuffer( GL_ARRAY_BUFFER, buffer );
    glBufferData( GL_ARRAY_BUFFER, sizeof(points)+sizeof(colors), NULL,
GL_STATIC_DRAW );

    glBufferSubData(GL_ARRAY_BUFFER, 0, sizeof(points), points);
    glBufferSubData(GL_ARRAY_BUFFER, sizeof(points), sizeof(colors), colors);

}


void shaderSetup( void )
{
    // Nạp các shader và sử dụng chương trình shader
    program = InitShader( "vshader1.glsl", "fshader1.glsl" );   // hàm InitShader
khai báo trong Angel.h
    glUseProgram( program );

    // Khởi tạo thuộc tính vị trí đỉnh từ vertex shader
    GLuint loc_vPosition = glGetAttribLocation( program, "vPosition" );
    glEnableVertexAttribArray( loc_vPosition );
    glVertexAttribPointer(loc_vPosition, 4, GL_FLOAT, GL_FALSE, 0,
BUFFER_OFFSET(0) );

    GLuint loc_vColor = glGetAttribLocation(program, "vColor");
    glEnableVertexAttribArray(loc_vColor);
    glVertexAttribPointer(loc_vColor, 4, GL_FLOAT, GL_FALSE, 0,
BUFFER_OFFSET(sizeof(points)));
```

```
    model_view_loc = glGetUniformLocation(program, "model_view");
    projection_loc = glGetUniformLocation(program, "projection");

    glEnable(GL_DEPTH_TEST);
    glClearColor( 1.0, 1.0, 1.0, 1.0 );        /* Thiết lập màu trắng là màu xóa
màn hình*/
}



void display( void )
{

    glClear( GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT );

/*Thiết lập phép chiếu*/
/*point4 eye(radius*sin(theta)*cos(phi),
                radius*sin(theta)*sin(phi),
                radius*cos(theta),
                1.0);*/

    point4 eye(0.5, 1, 1.5, 1);
    point4 at(0.0, 0.0, 0.0, 1.0);
    vec4 up(0.0, 1.0, 0.0, 1.0);

    mat4 mv = LookAt(eye, at, up);
    glUniformMatrix4fv(model_view_loc, 1, GL_TRUE, mv);

    mat4 p = Frustum( l, r, bottom, top, zNear, zFar);
    glUniformMatrix4fv(projection_loc, 1, GL_TRUE, p);

/*Vẽ hình*/
    glDrawArrays( GL_TRIANGLES, 0, NumPoints );    /*Vẽ các tam giác*/
    glutSwapBuffers();
}


/*void
keyboard(unsigned char key, int x, int y)
{
    switch (key) {
    case 033: // Escape Key
    case 'q': case 'Q':
          exit(EXIT_SUCCESS);
          break;
    case 'x': l *= 1.1; r *= 1.1; break;
    case 'X': l *= 0.9; r *= 0.9; break;
    case 'y': bottom *= 1.1; top *= 1.1; break;
    case 'Y': bottom *= 0.9; top *= 0.9; break;
    case 'z': zNear *= 1.1; zFar *= 1.1; break;
    case 'Z': zNear *= 0.9; zFar *= 0.9; break;
    case 'r': radius *= 2.0; break;
    case 'R': radius *= 0.5; break;
    case 't': theta += dr; break;
    case 'T': theta -= dr; break;
    case 'p': phi += dr; break;
    case 'P': phi -= dr; break;
    case ' ' : // reset values to their defaults
          l = -1.0;
          r = 1.0;
          bottom = -1.0;
```

```
            top = 1.0;
            zNear = 0.5;
            zFar = 3.0;
            radius = 1.0;
            theta = 0.0;
            phi = 0.0;
            break;
        }
        glutPostRedisplay();
    }
    */

    void reshape(int width, int height)
    {
        glViewport(0, 0, width, height);
    }

    int main( int argc, char **argv )
    {
        // main function: program starts here

        glutInit( &argc, argv );
        glutInitDisplayMode( GLUT_DOUBLE|GLUT_RGBA);
        glutInitWindowSize( 640, 640 );
        glutInitWindowPosition(100,150);
        glutCreateWindow( "Drawing a Cube" );


        glewInit();

        generateGeometry( );
        initGPUBuffers( );
        shaderSetup( );

        glutDisplayFunc( display );
        //glutKeyboardFunc( keyboard );
        glutReshapeFunc(reshape);

        glutMainLoop();
        return 0;
    }
```

## b) Vertex shader file

```
#version 400
in vec4 vPosition;
in vec4 vColor;
out vec4 color;

uniform mat4 model_view;
uniform mat4 projection;

void main()
{
     gl_Position = projection * model_view * vPosition / vPosition.w;
       //gl_Position = vPosition;
       color=vColor;
}
```

## c) Fragment shader file

```
#version 400
in vec4 color;
out vec4  fColor;

void main()
{
   fColor=color;

}
```

C- **BÀI TẬP**

1. Cài đặt chương trình thiết lập vị trí quan sát và chiếu phối cảnh cho hình lập phương đơn vị. Thay đổi các thuộc tính về vị trí quan sát và nhận xét.

2. Cài đặt phép chiếu cho mô hình cái bàn và điều khiển thay đổi các tham số của phép chiếu bằng bàn phím.

3. Cài đặt các phép biến đổi cho cho camera: yaw, pitch, roll,…

**Lưu ý:** Chuyển từ hệ toạ độ Decartes sang hệ toạ độ cầu

```
point4 eye(radius*sin(theta)*cos(phi),

                    radius*sin(theta)*sin(phi),
                    radius*cos(theta),
                    1.0);*/
```

xe= R.sinθ.cosφ
ye= R.sinθ.sinφ
ze= R.cosθ