CG Practice 4

COLLEGE OF COMPUTING

HANYANG ERICA CAMPUS

Q YOUN HONG (홍규연)

Practice 4 Review

Rotating cube: x,y,z축을 회전축으로 삼아 회전하는 큐브

Practice 4 Review

- 회전 변환에 의해서 바뀌는 점들의 위치 업데이트 방법

① Method 1: 애플리케이션(c-code)에서 큐브를 회전시키고 회전된 점들의 위치를 VBO의 buffer에서 업데이트

② Method 2: 애플리케이션에서 회전 변환 행렬을 계산하고 계산된 4x4 행렬을 uniform 변수로 vertex shader에 전달

③ Method 3: 애플리케이션에서 각 축의 회전 각도만 업데이트하고 (x,y,z)-축의 회전각도를 vertex shader로 전달, shader 내부에서 4x4 행렬 계산

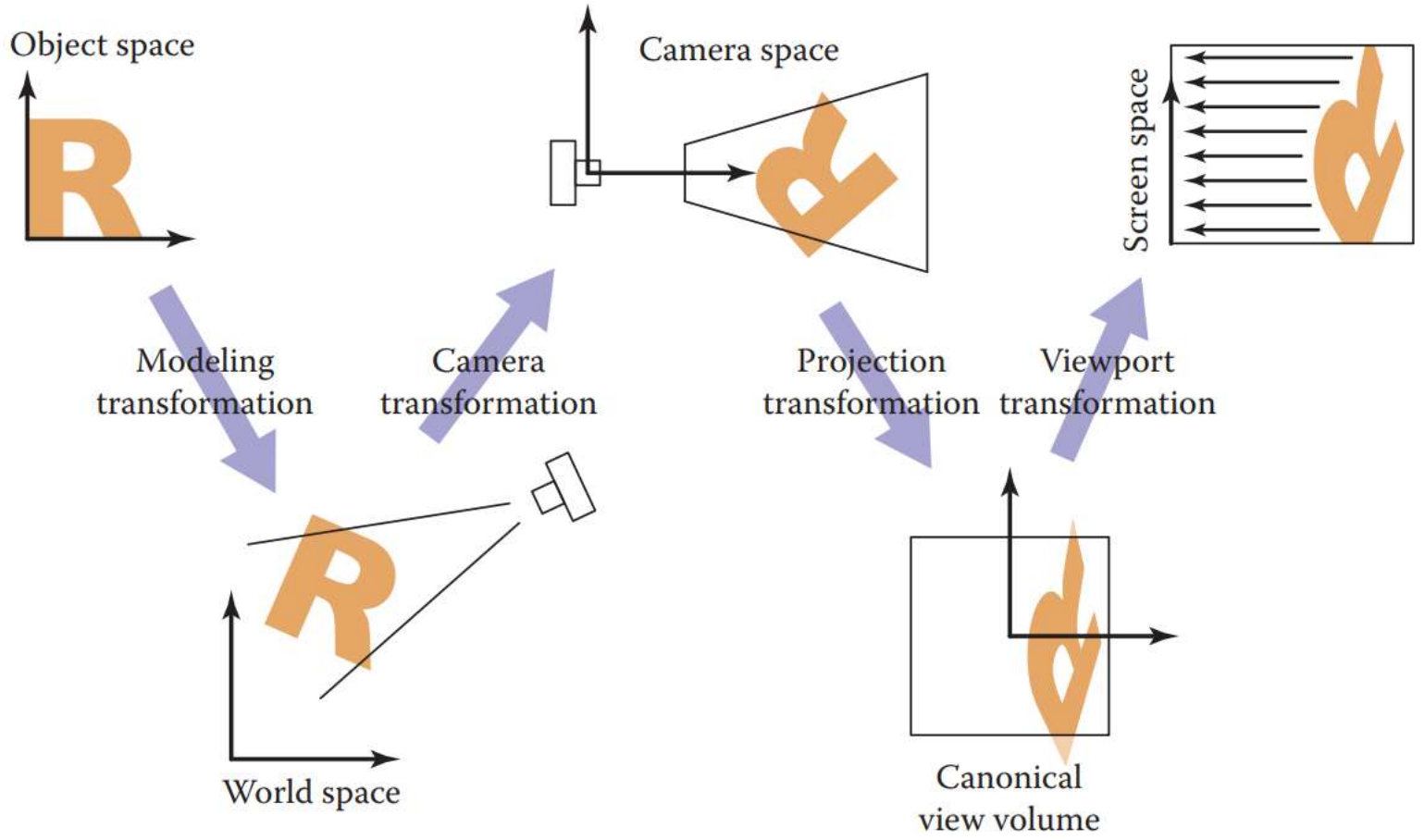
(See practice3/main_colorcuberot1.cpp,

practice3/main_colorcube_method2.cpp,

practice3/main_colorcube_method3.cpp)

Viewing in OpenGL + Shaders

뷰잉 변환(Viewing Transformation)



뷰잉 변환 행렬

- orthographic projection 적용시 world coordinate → screen coordinate로 변환시키는 행렬

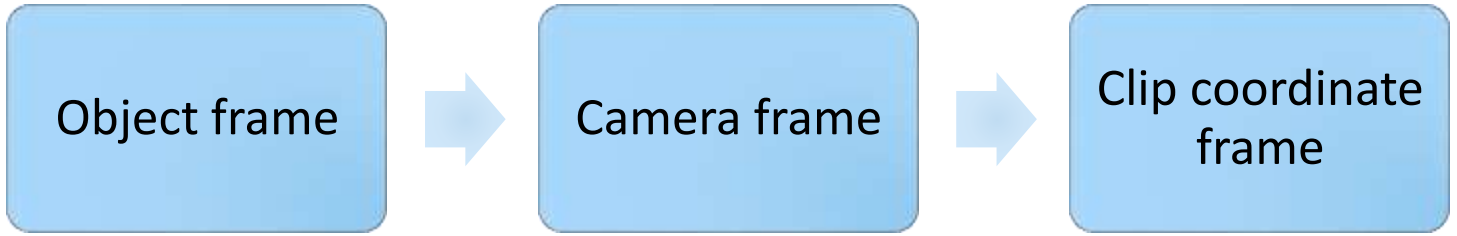
$$\mathbf{M} = \mathbf{M}_{\text{vp}}\mathbf{M}_{\text{ortho}}\mathbf{M}_{\text{cam}}$$

- perspective projection 적용시 world coordinate → screen coordinate로 변환시키는 행렬

$$\mathbf{M} = \mathbf{M}_{\text{vp}}\mathbf{M}_{\text{ortho}}\mathbf{P}\mathbf{M}_{\text{cam}}$$

(Old) OpenGL Viewing Process

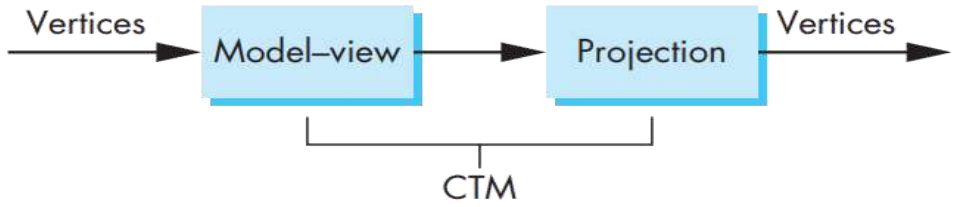
- (Old) OpenGL에서 viewing은 크게 세 frame 사이의 변환으로 볼 수 있음



- Transformation matrix은 OpenGL에서 상태 변수(state variable)
  - Current Transformation Matrix (CTM)
  - Pipeline은 CTM을 이용하여 어플리케이션에서 입력 받은 점을 변환

- Transformation matrix in OpenGL (in OLD OpenGL)
  - model-view matrix: model frame (geometric object representation)에서 eye frame으로
  - projection matrix: 투영변환 + clip coordinate

```
glMatrixMode (GL_MODELVIEW);
glMatrixMode (GL_PROJECTION);
```



Current Transform Matrix

- CTM은 수행하고자 하는 순서의 반대로 업데이트 한다
  - ‘Stack’ like operation

(예시) (4,5,6)를 지나고 방향이 (1,2,3)인 회전축을 중심으로 45도만큼 회전하기 위한 CTM

$$\mathbf{C} \leftarrow \mathbf{I},$$

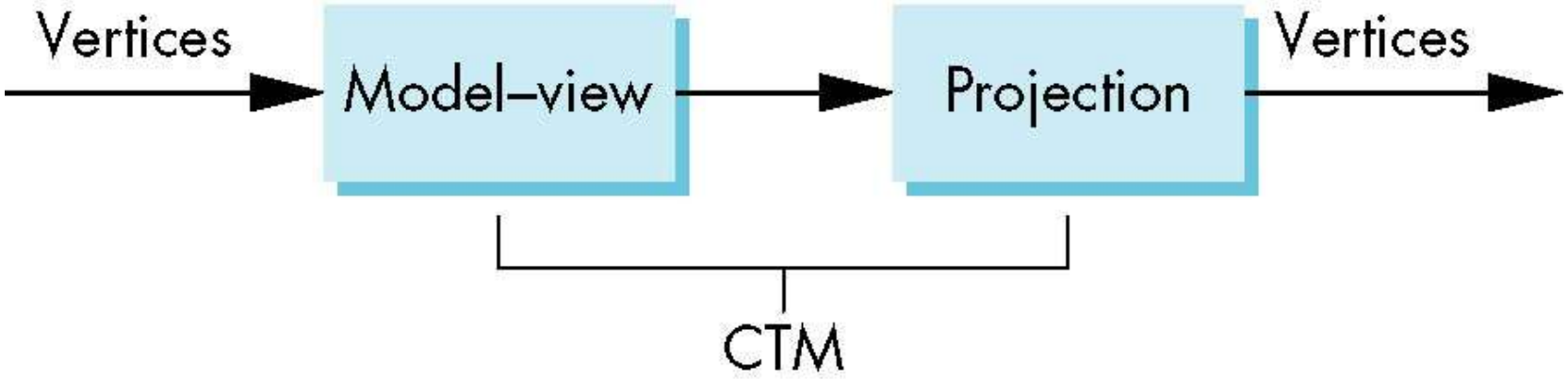
$$\mathbf{C} \leftarrow \mathbf{CT}(4.0, 5.0, 6.0),$$

$$\mathbf{C} \leftarrow \mathbf{CR}(45.0, 1.0, 2.0, 3.0),$$

$$\mathbf{C} \leftarrow \mathbf{CT}(-4.0, -5.0, -6.0).$$

OpenGL Viewing Process

- Shader 프로그램 안에서 old OpenGL의 CTM을 통한 viewing 변환을 구현

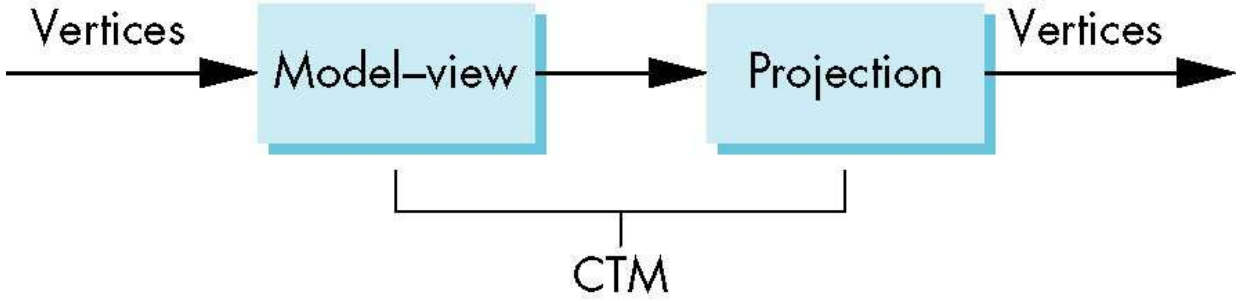


CTM: Current Transform Matrix

Program: Orthographic Projection

OpenGL Viewing Process

- Shader 프로그램 안에서 old OpenGL의 CTM을 통한 viewing 변환을 구현
  - CTM = projection * model-view
  - Projection, model_view: uniform mat4 변수로 애플리케이션에서 전달



CTM: Current Transform Matrix

OpenGL Viewing Process

- model-view, projection행렬을 uniform 변수로 shader에 전달

(step 1) Gluint model_view, projection 선언 (main.cpp)

(step 2) uniform variabl들의 shader에서의 주소 얻기 (init())

```
Model_view = glGetUniformLocation(program, "model_view");
Projection = glGetUniformLocation(program, "projection");
```

(step 3) model-view, projection 행렬 업데이트 하기(display())

```
mat4 mv = LookAt(eye, at, up);
glUniformMatrix4fv(model_view, 1, GL_TRUE, mv);
mat4 p = Ortho(left, right, bottom, top, zNear, zFar);
glUniformMatrix4fv(projection, 1, GL_TRUE, p);
```

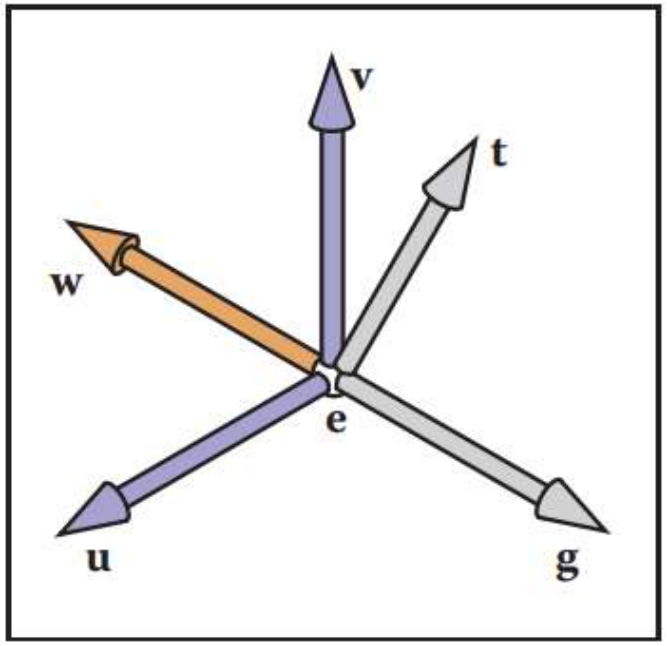
OpenGL Viewing Process

- model-view, projection 변수의 값을 애플리케이션에서 전달받아 좌표변환 (in vshader.glsl)

```
#version 330

in vec4 vPosition;
in vec4 vColor;
out vec4 color;

uniform mat4 model_view;
uniform mat4 projection;

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

Camera Transformation

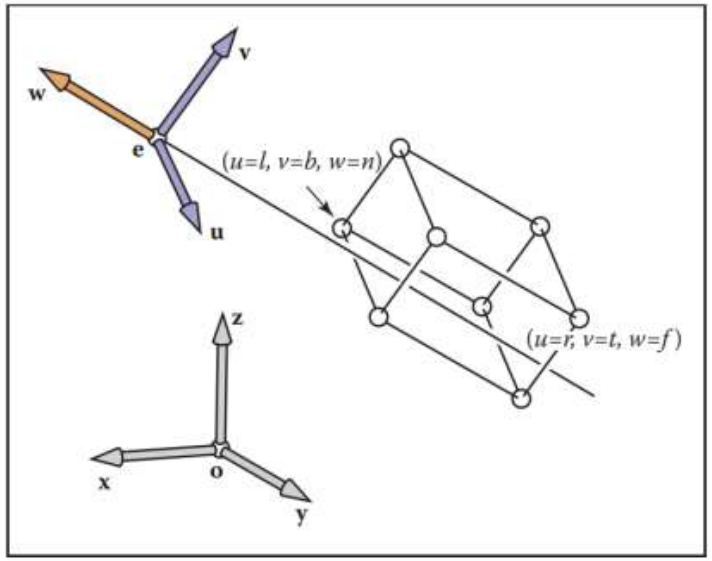
- 카메라 좌표계로 object들의 좌표를 변환시킴
- 카메라 변환 = Scene 변환



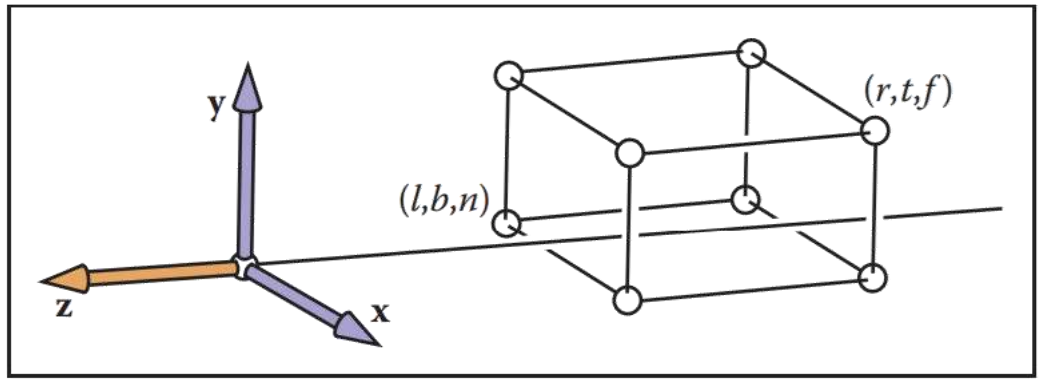
- e: 카메라의 위치
- g: 카메라가 바라보는 방향
  (viewing direction = gazing direction)
- t: 카메라의 위쪽 방향 (view-up vector)

Camera Transformation

- 카메라 좌표계로 object들의 좌표를 변환시킴
- 카메라 변환 = Scene 변환
- model-view matrix로 표현됨
- 카메라 변환은 임의의 view volume을 orthographic view volume으로 변환

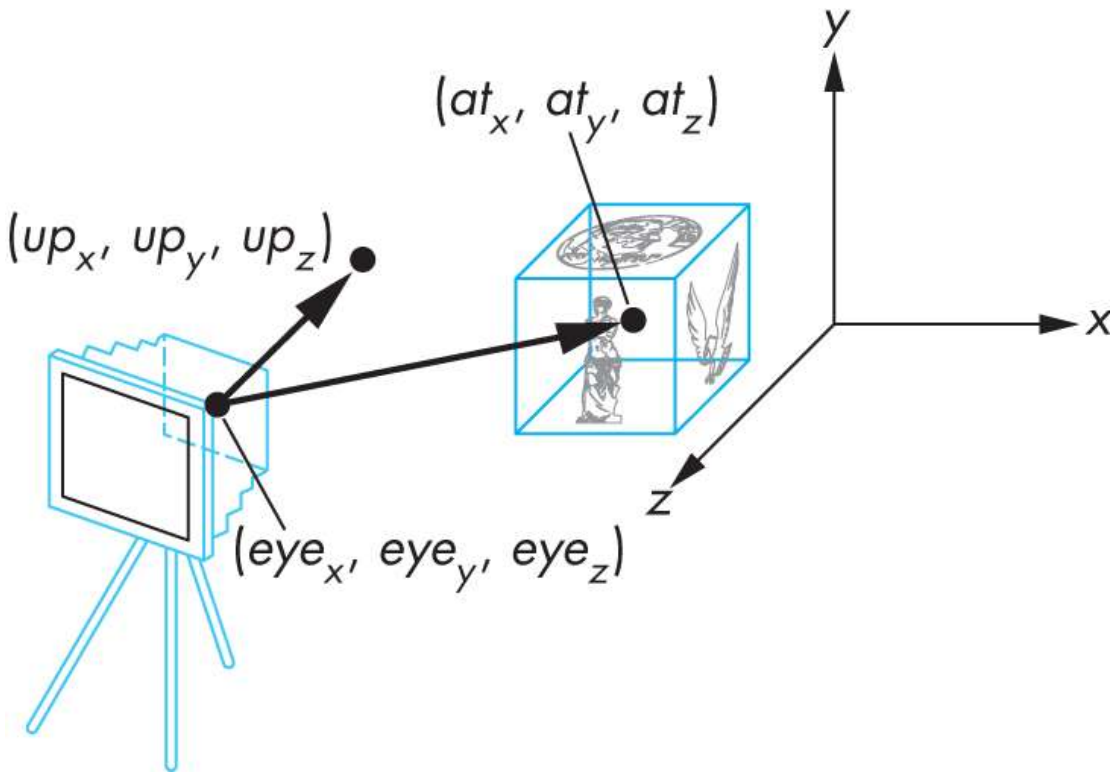


Camera Transformation



Look-At Function

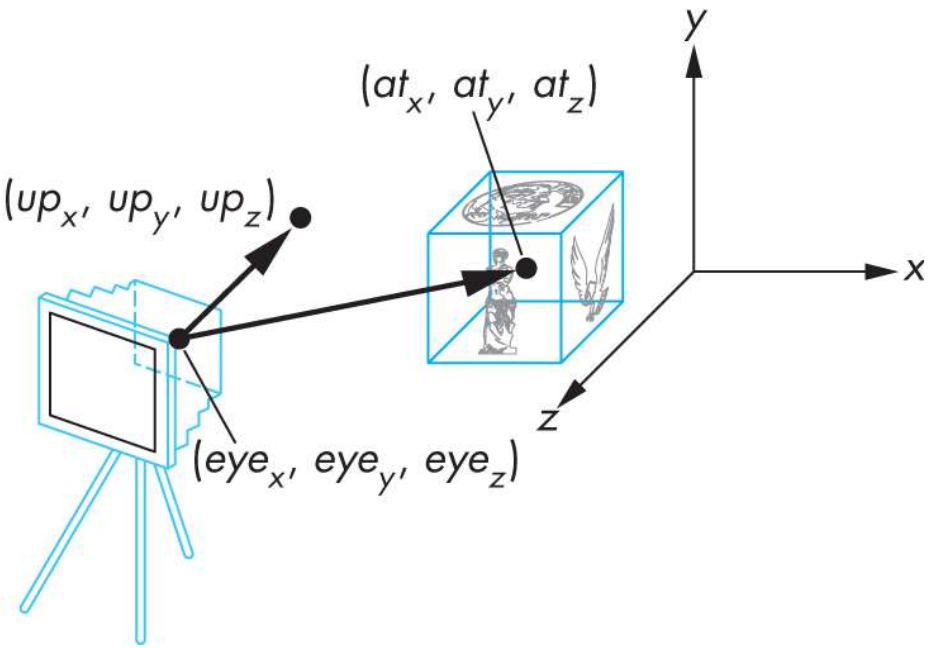
- 2 point와 1 vector로 카메라 변환 결정



Look-At Function

- **`gluLookAt (ex,ey,ez, fx,fy,fz, ux,uy,uz);`**
- e = eye point (eye)
- f = focus point (at)
- u = up vector (up)

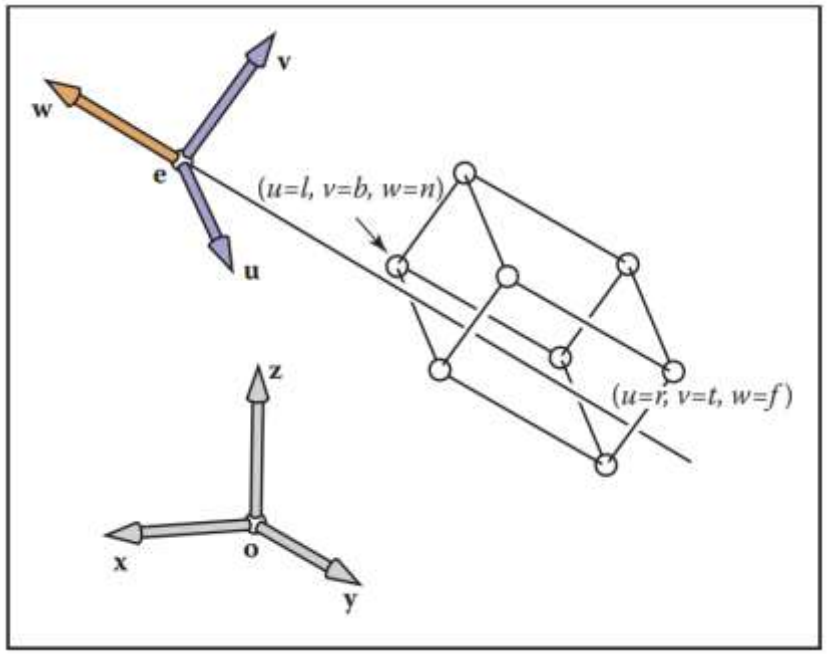
```
void display()
{
        glClear (GL_COLOR_BUFFER_BIT |
                 GL_DEPTH_BUFFER_BIT);
        glMatrixMode (GL_MODELVIEW);
        glLoadIdentity ();
        gluLookAt (ex,ey,ez,fx,fy,fz,ux,uy,uz);
        ...
        renderObjects();
        glutSwapBuffers();
}
```



Camera transformation in old OpenGL

Look-At Function


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

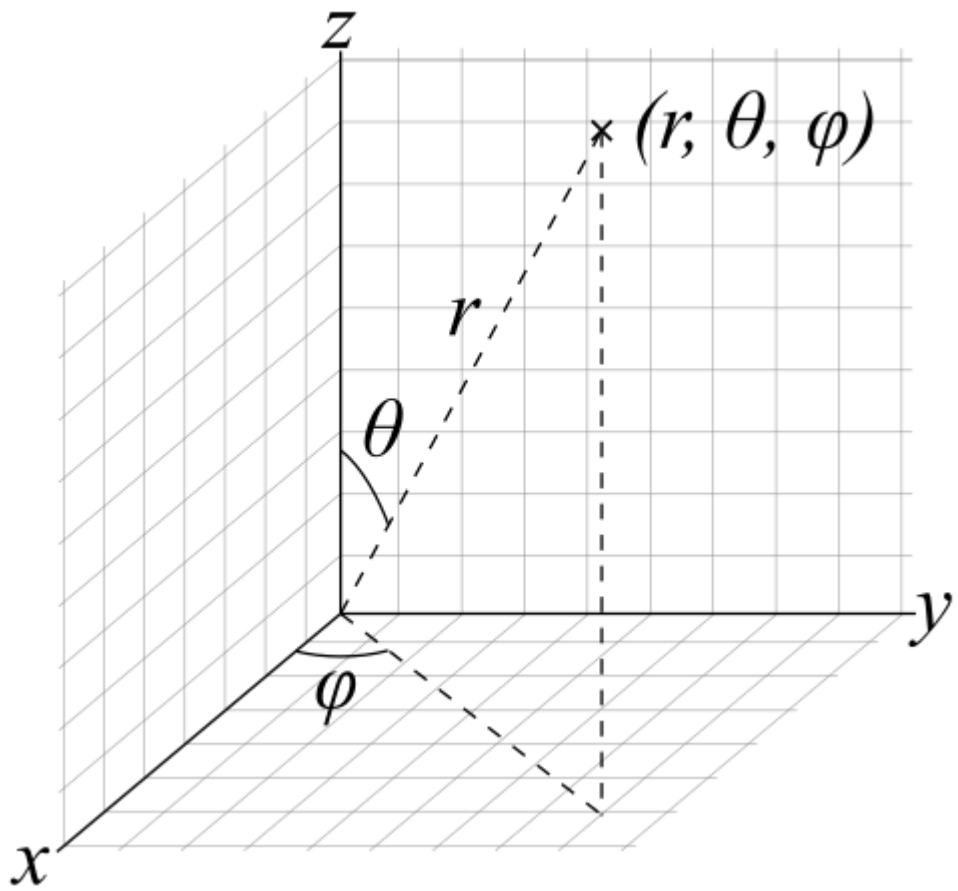


$$\mathbf{M}_{\text{cam}} = \begin{bmatrix} \mathbf{u} & \mathbf{v} & \mathbf{w} & \mathbf{e} \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} x_u & y_u & z_u & 0 \\ x_v & y_v & z_v & 0 \\ x_w & y_w & z_w & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -x_e \\ 0 & 1 & 0 & -y_e \\ 0 & 0 & 1 & -z_e \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Look-At Function

- Look-At Function 사용해서 카메라 변수 세팅하기 (display())

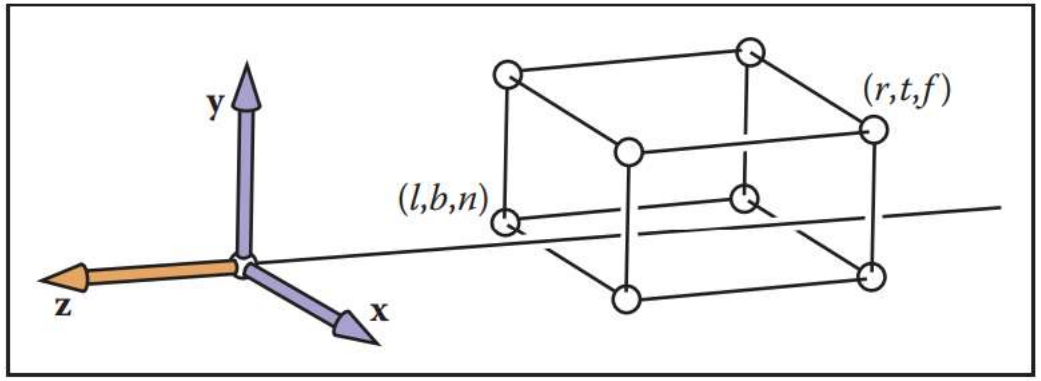
```
vec4 eye( radius * sin(theta) * cos(phi),
          radius * sin(theta) * sin(phi),
          radius * cos(theta), 1.0);
vec4 at(0.0, 0.0, 0.0, 1.0);
vec4 up(0.0, 1.0, 0.0, 0.0);

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



Orthographic Projection

- Orthographic View Volume
  - -Z 방향으로 바라봄(viewing direction)
  - +y이 항상 위 (up direction)
  - Orthographic view volume: [l,r]x[b,t]x[f,n]

$$x = l \equiv \text{left plane},$$
$$x = r \equiv \text{right plane},$$
$$y = b \equiv \text{bottom plane},$$
$$y = t \equiv \text{top plane},$$
$$z = n \equiv \text{near plane},$$
$$z = f \equiv \text{far plane}.$$



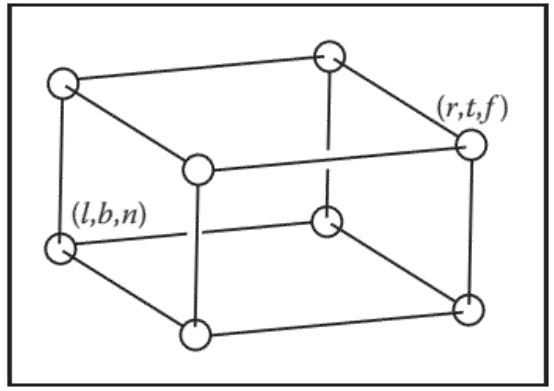
Orthographic Projection Transformation

- Orthographic Projection Transformation

  ⇒ [l,r]x[b,t]x[f,n]를 [-1,1]x[-1,1]x[-1,1]로 변환

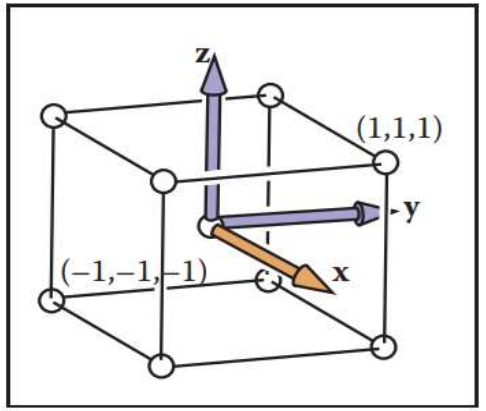
  ⇒ Translate(-l, -b, -f) → Scale($\frac{2}{r-l}, \frac{2}{t-b}, \frac{2}{n-f}$) → Translate(-1, -1, -1)

$$\mathbf{M}_{\text{orth}} = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{2}{n-f} & -\frac{n+f}{n-f} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Orthographic View Volume

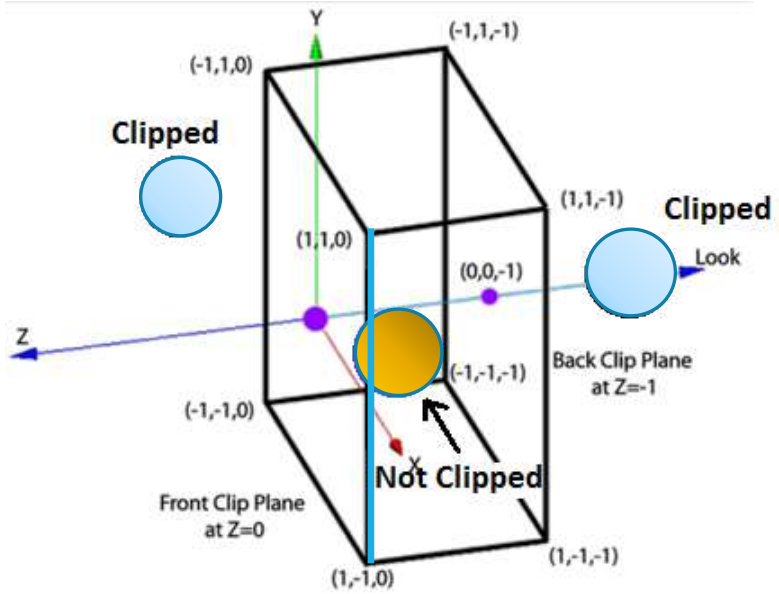
Orthographic Projection Transformation

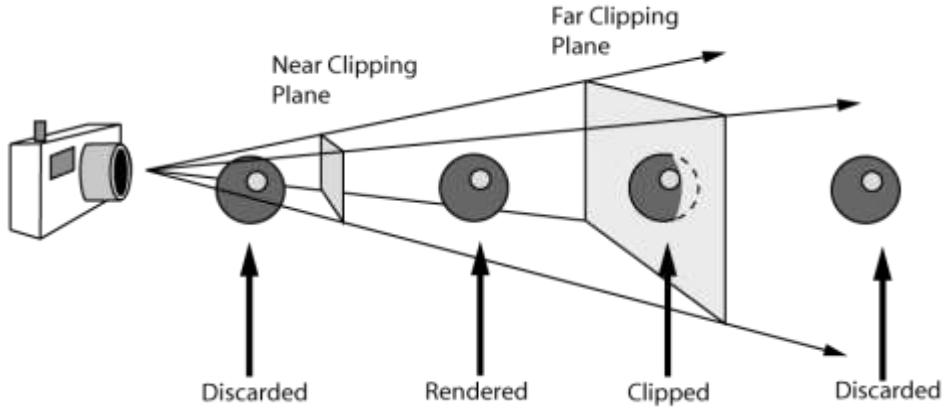


Canonical View Volume

Clipping against Canonical View Volume

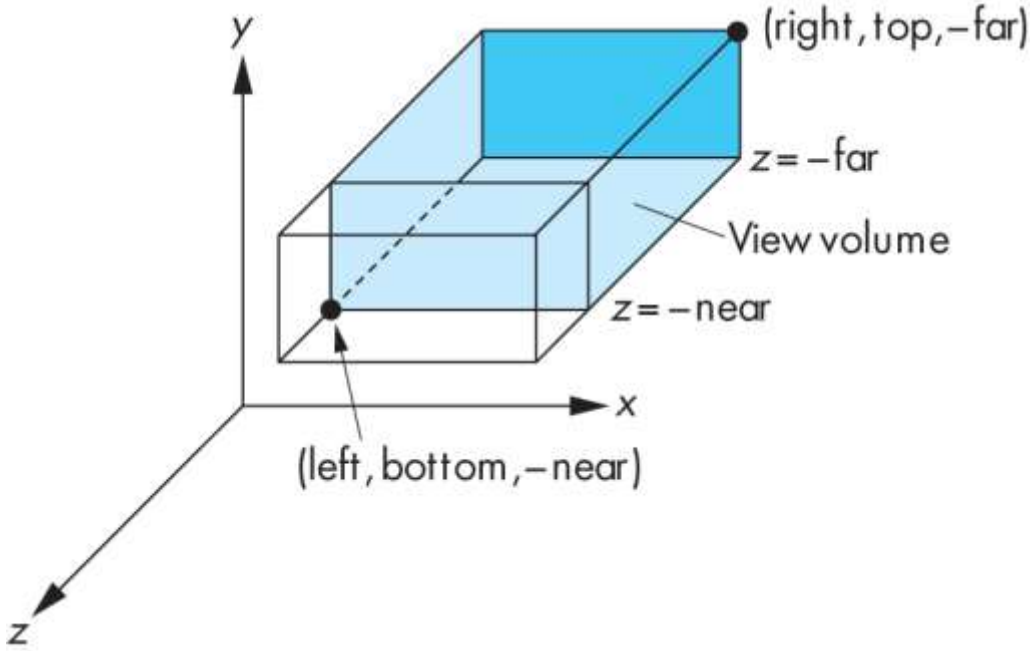
- Orthographic projection은 Object들을 canonical view volume (=normalized view volume)으로 변환시킴
- 이때, 평면 $x = \pm 1, y = \pm 1, z = \pm 1$ 바깥에 있는 object들은 잘림
- OpenGL 포함, 대부분의 graphics package에서 지원되는 기능





Orthographic Viewing in Old OpenGL


```
glOrtho(xmin, xmax, ymin, ymax, near, far);
```



Ortho() Function

(in mat.h)

```
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] = 2.0f/(right - left);
    c[1][1] = 2.0f/(top - bottom);
    c[2][2] = 2.0f/(zNear - zFar);
    c[3][3] = 1.0f;
    c[0][3] = -(right + left)/(right - left);
    c[1][3] = -(top + bottom)/(top - bottom);
    c[2][3] = -(zFar + zNear)/(zFar - zNear);
    return c;
}
```

$$\mathbf{M}_{\text{orth}} = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{2}{n-f} & -\frac{n+f}{n-f} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Ortho() Function

- Ortho() function을 이용해서 orthographic view volume을 설정하고 orthographic projection 적용하기 (display())

```
mat4 p = Ortho(left, right, bottom, top, zNear, zFar);

glUniformMatrix4fv(projection, 1, GL_TRUE, p);
```

- (in vshader.glsl)

```
gl_Position = projection * model_view * vPosition;
```

Keyboard callback

- Keyboard 입력으로 camera transformation, orthographic view volume 조절 (in keyboard())

```
…
case 'x': left *= 1.1; right *= 1.1; break;
case 'X': left *= 0.9; right *= 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 'o': theta += dr; break;
case 'O': theta -= dr; break;
case 'p': phi += dr; break;
case 'P': phi -= dr; break;
…
```

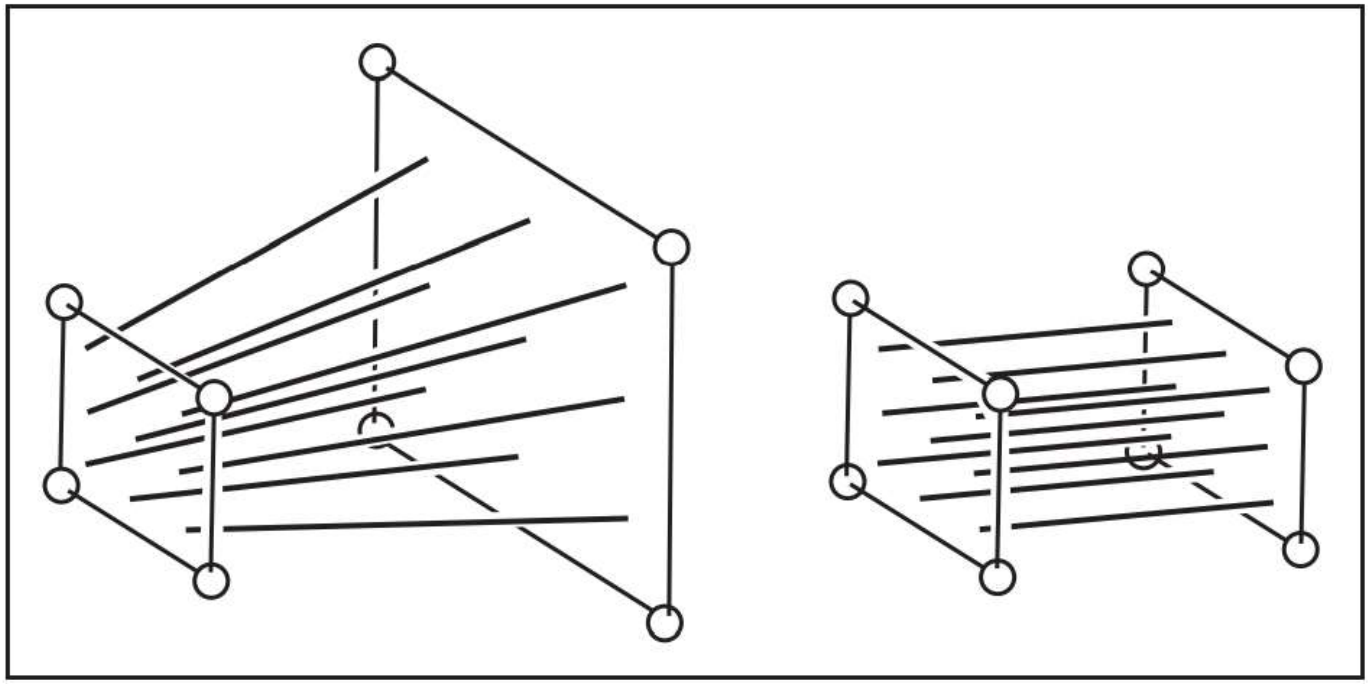
Execution Result

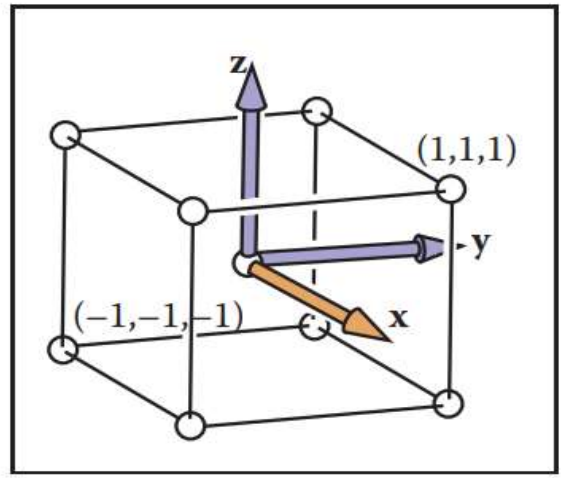
Program: Perspective Projection

Perspective Transformation

- Perspective Transformation: Perspective projection(원근 투영) 변환 후 계산된 orthographic view volume을 canonical view volume으로 다시 변환

$$\mathbf{M}\_\text{per} = \mathbf{M}_{\text{ortho}}\mathbf{P}$$





Canonical View Volume

Frustrum()


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

$$\mathbf{M}_{\text{per}} = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{l+r}{l-r} & 0 \\ 0 & \frac{2n}{t-b} & \frac{b+t}{b-t} & 0 \\ 0 & 0 & \frac{f+n}{n-f} & \frac{2fn}{f-n} \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Perspective Transformation

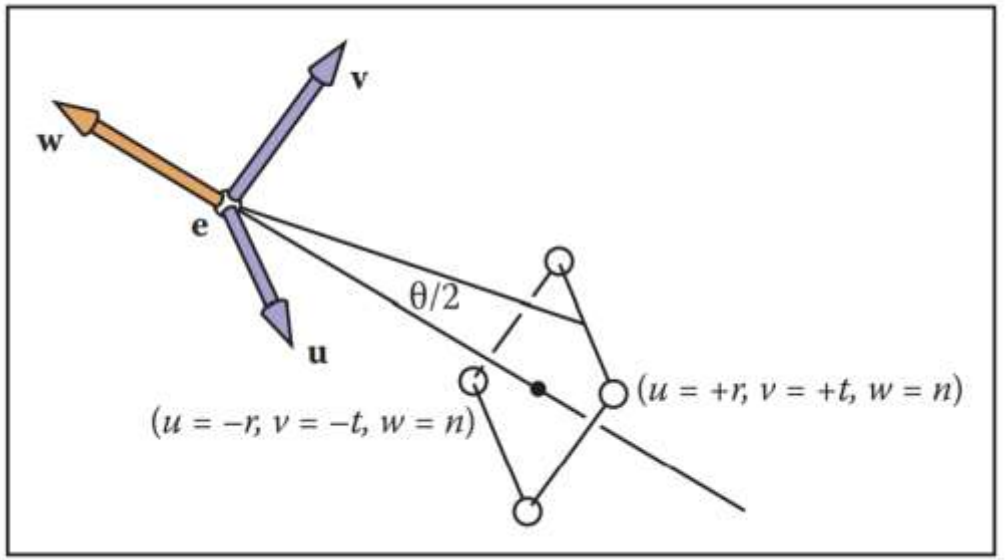
- Frustum() 함수를 이용해서 perspective transformation matrix 계산하고 vertex shader에 넘겨주기 (display())

```
mat4 mv = LookAt(eye, at, up);
glUniformMatrix4fv(model_view, 1, GL_TRUE, mv);
mat4 p = Frustum(left, right, bottom, top, zNear, zFar);
glUniformMatrix4fv(projection, 1, GL_TRUE, p);
```

Execution Result

Perspective()


```
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] = -2.0f*zFar*zNear/(zFar - zNear);
    c[3][2] = -1.0f;
    return c;
}
```



Perspective()

- 물체의 왜곡을 최소화하기 위해 view frustrum의 x와 y의 비율을 스크린의 width와 height의 비율과 같게 함
- Projection matrix의 변수

```
GLfloat fovy = 45.0; //field-of-view in y direction angle (in degrees)
GLfloat aspect; //Viewport aspect ratio
GLfloat zNear = 0.5, zFar = 3.0;
```

- (in reshape(int width, int height))

```
glViewport(0, 0, width, height);
aspect = GLfloat(width) / height;
```

- (in display())

```
mat4 p = Perspective(fovy, aspect, zNear, zFar);
glUniformMatrix4fv(projection, 1, GL_TRUE, p);
```

Execution Result

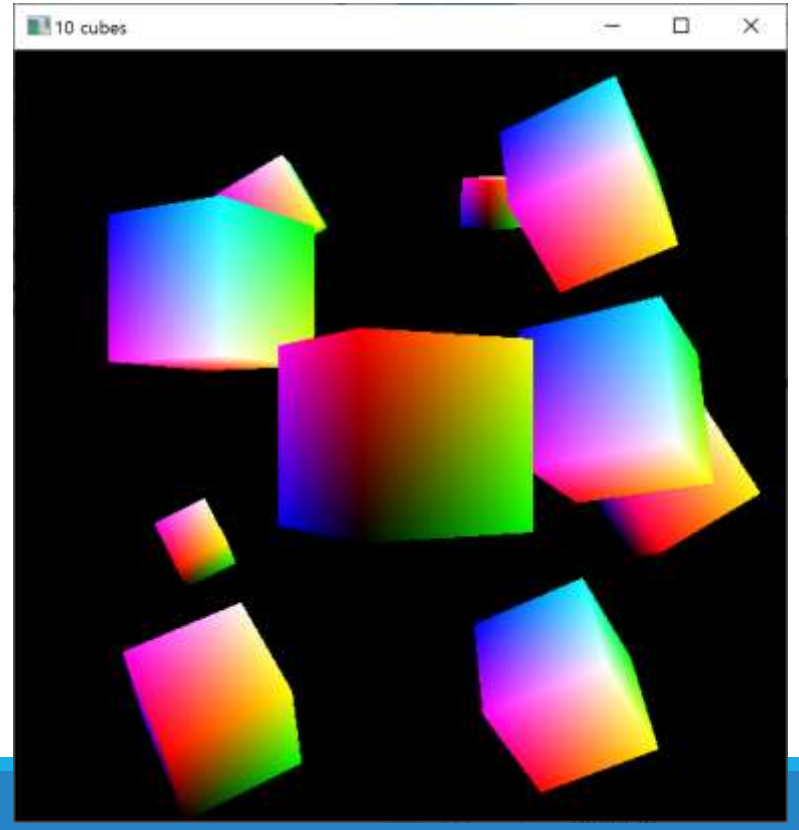
Review Task 2023-10-06

한양대학교 ERICA
소프트웨어융합대학
COLLEGE OF COMPUTING

- main_persepctive2.cpp를 변경하여 서로 다른 위치에 있는 10개의 cube 그리기
  - Vertex buffer object(VBO)에는 1개의 cube 위치, 색상만 저장함
  - 각각의 박스들은 다음의 instance (modeling) transformation 가짐
    - Cube의 position (translation)

```
vec3 cubePositions[] = {
    vec3(0.0f,  0.0f,  0.0f),
    vec3(2.0f,  5.0f, -15.0f),
    vec3(-1.5f, -2.2f, -2.5f),
    vec3(-3.8f, -2.0f, -12.3f),
    vec3(2.4f, -0.4f, -3.5f),
    vec3(-1.7f,  3.0f, -7.5f),
    vec3(1.3f, -2.0f, -2.5f),
    vec3(1.5f,  2.0f, -2.5f),
    vec3(1.5f,  0.2f, -1.5f),
    vec3(-1.3f,  1.0f, -1.5f)
};
```

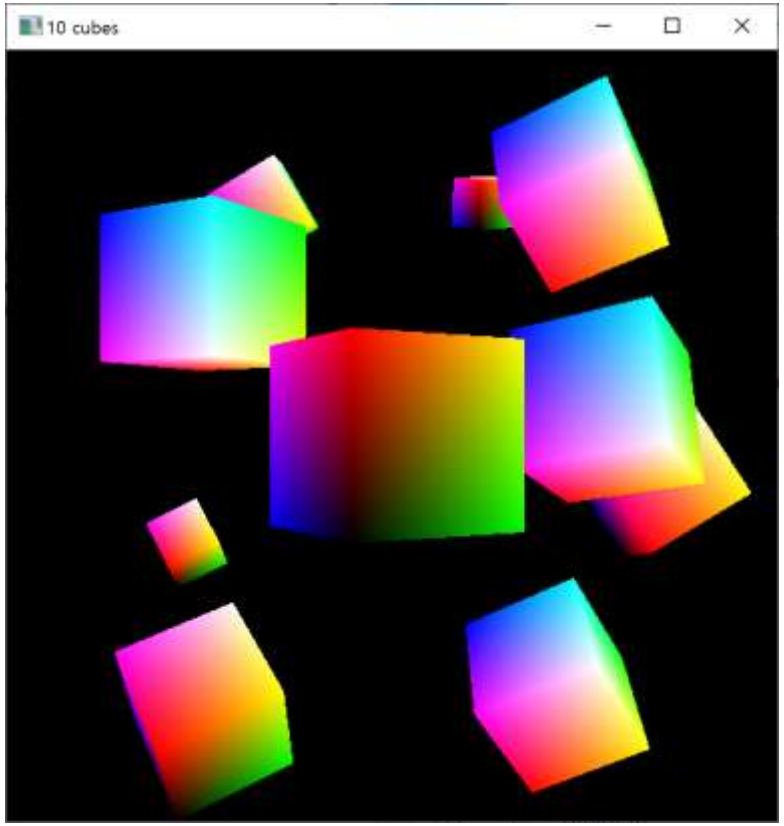
    - Cube의 rotation
      - X축으로는 0~360°까지 20° 씩 증가, Y축으로 30°씩 회전



Review Task 2023-10-06

한양대학교 ERICA
소프트웨어융합대학
COLLEGE OF COMPUTING

- Hint
  - Modelview matrix는 view matrix와 model matrix의 합성으로 나타낼 수 있음
  - glDrawArrays()가 호출될 때마다 현재의 modelview, projection을 이용하여 cube의 vertex들의 위치를 계산한다.



- 제출물
  - main.cpp, vshader.glsl, fshader.glsl + 스크린샷