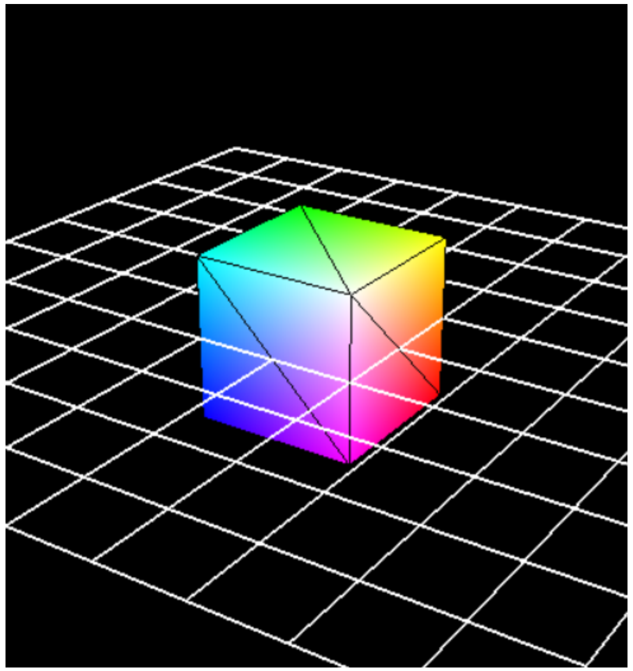
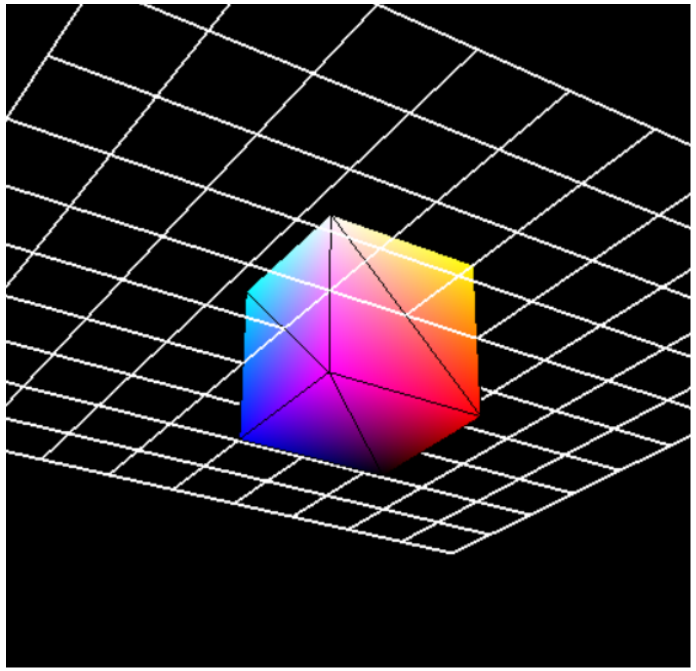
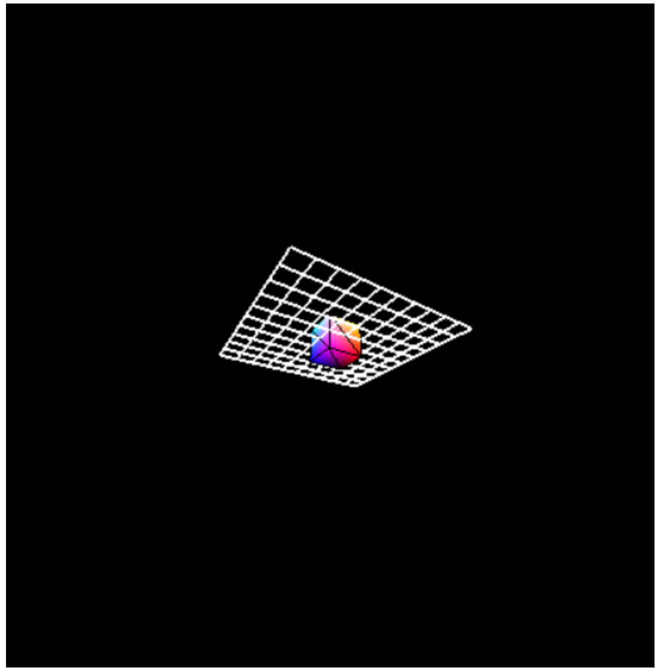
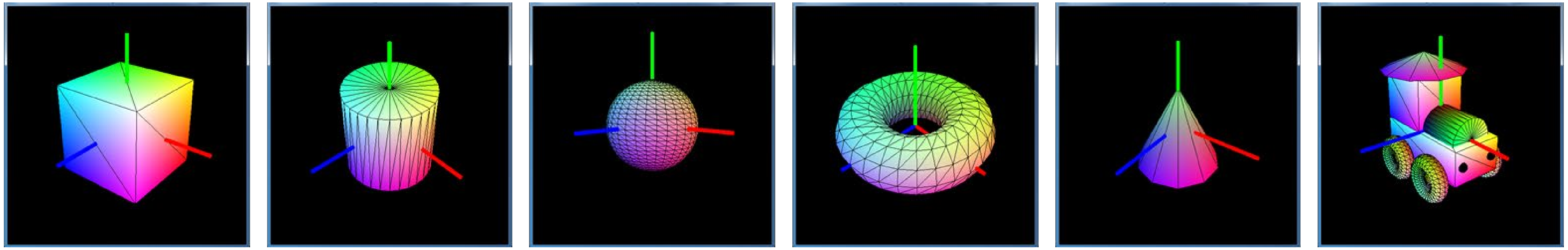
Homework 9: Navigating into the 3D World

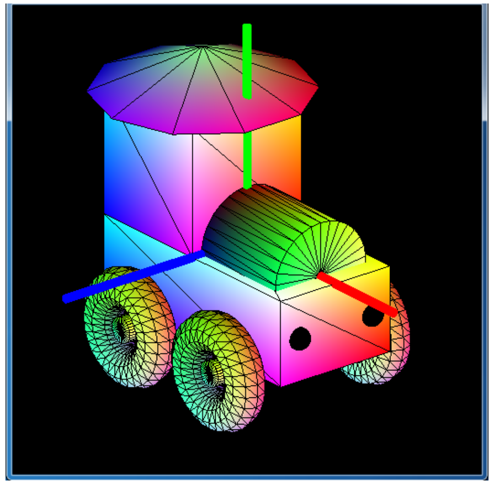
- In most 3D graphics applications, interactive viewing control is provided in order for a user to conveniently navigate through the 3D world. In this homework, you should extend the last programming assignment to allow the user to interactively control the view for the 3D world with a mouse device.

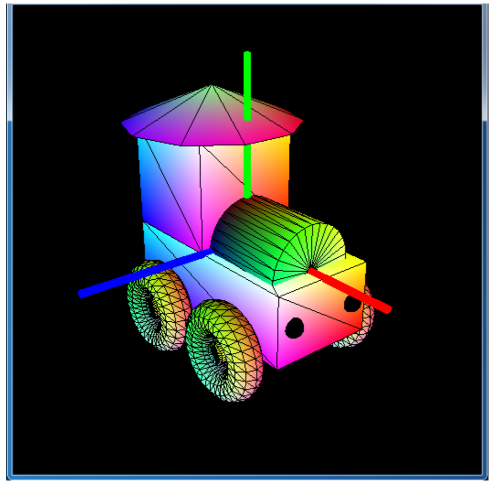




Requirement 1

- All the requirements of the last homework must be satisfied.

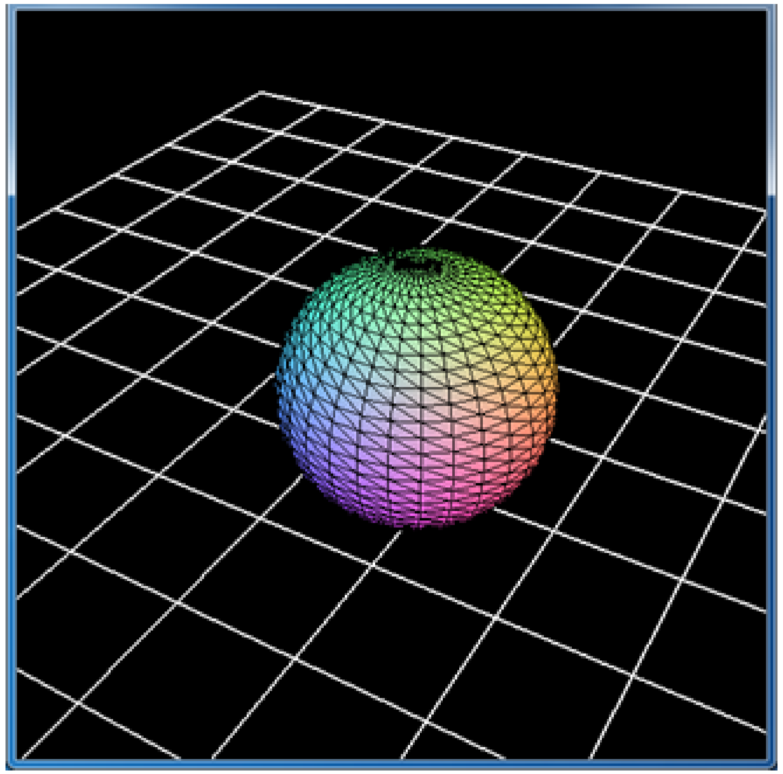

5 primitives and 1 composite model

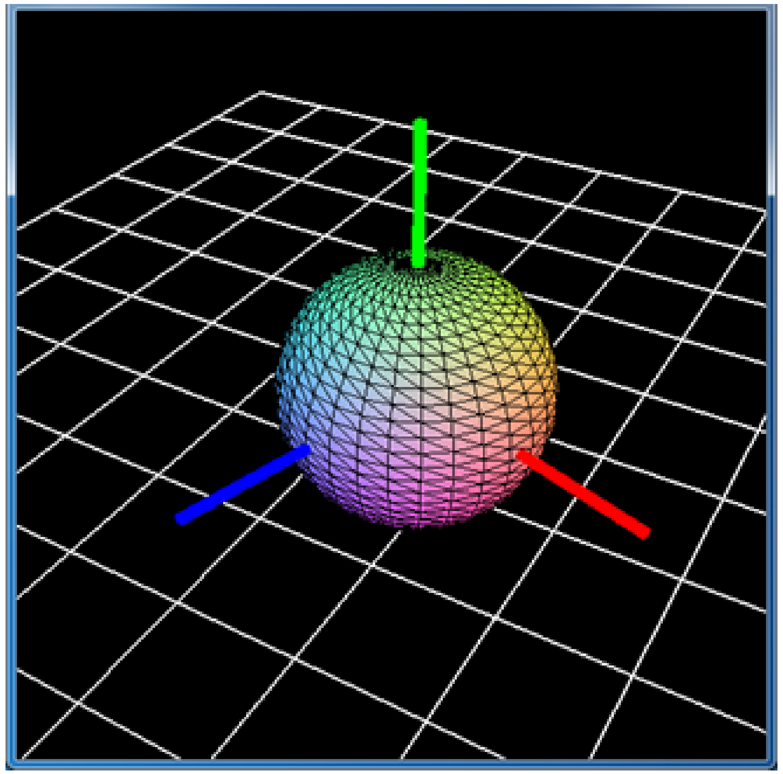

Orthographic projection


Perspective projection

Requirement 2

- If the user presses the 'a' key, the application should toggle on/off the visibility for drawing three axes of the local coordinate system using three cylinders. The x-axis, y-axis, and z-axis must be drawn in red, green, and blue, respectively.



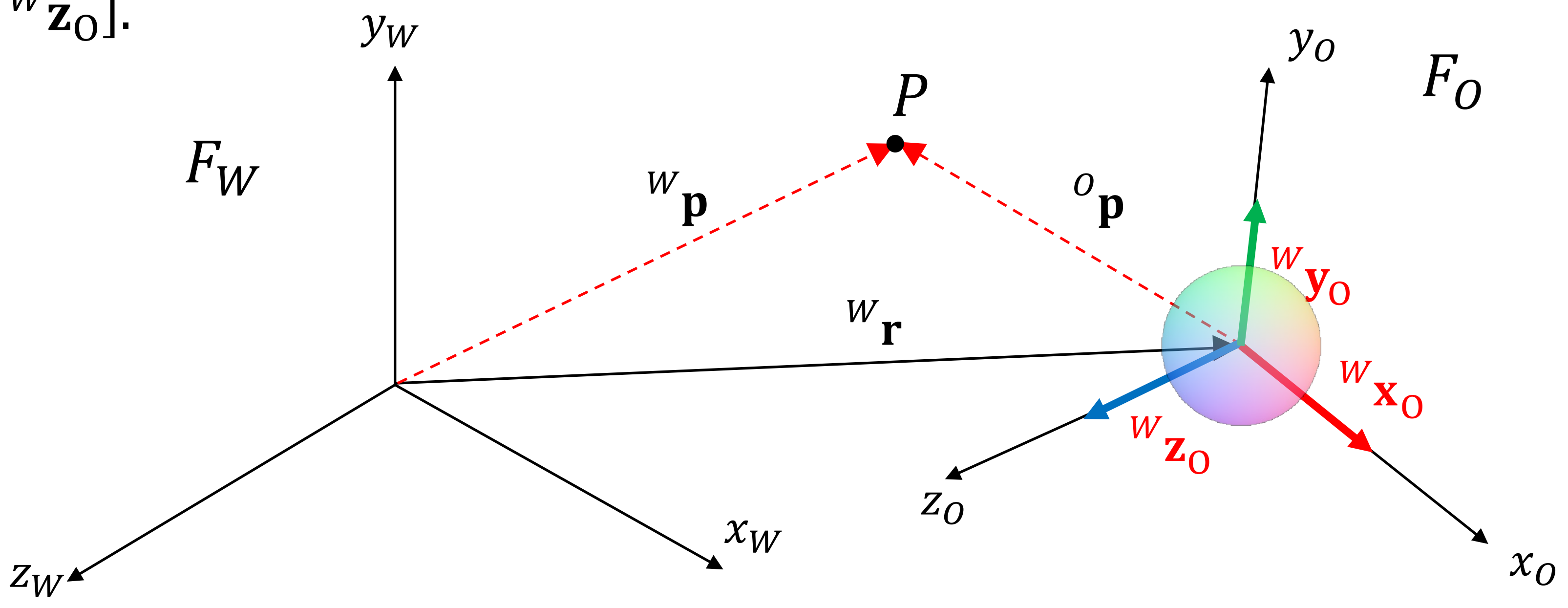


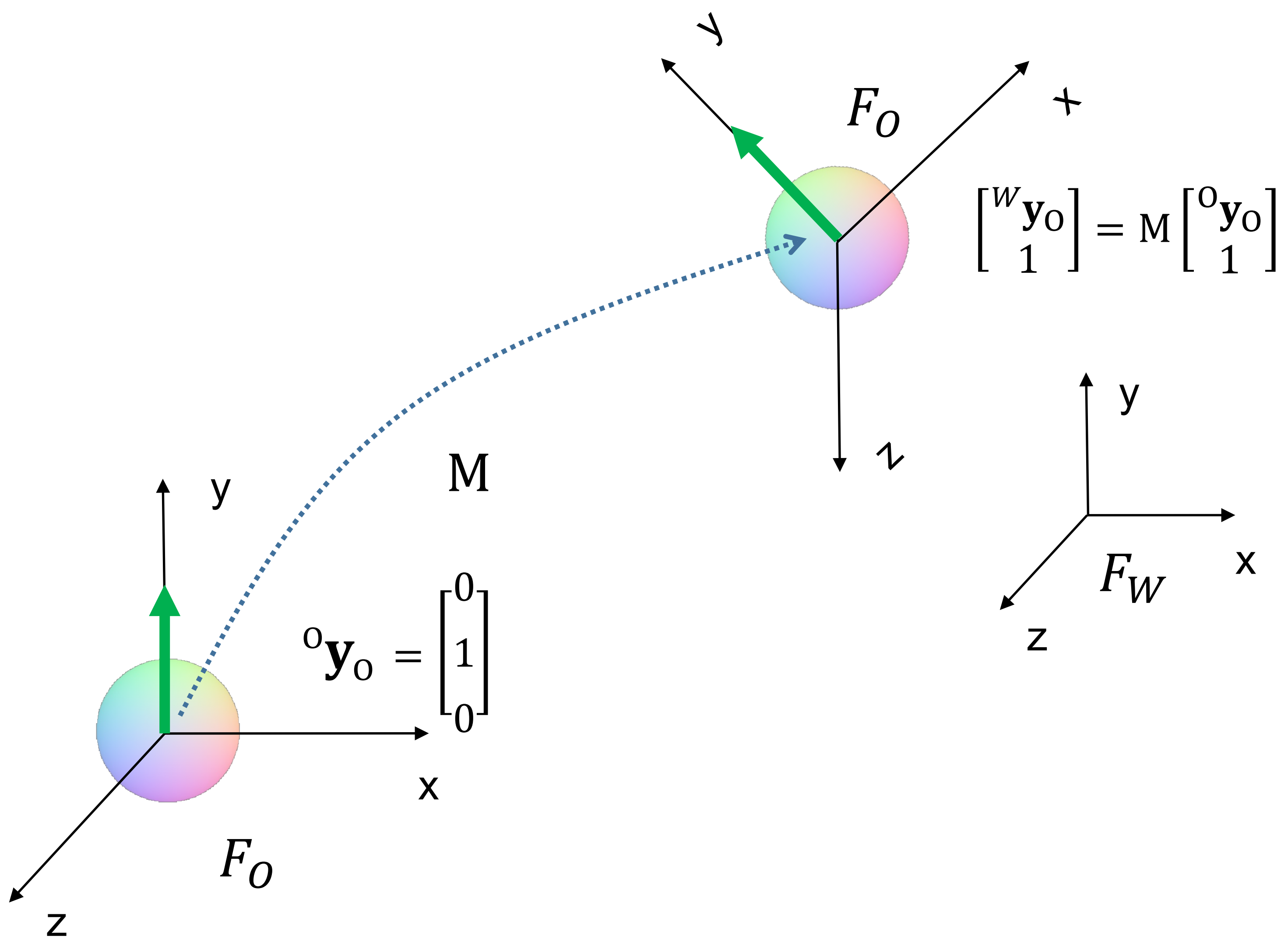
- How to know the three axes of the local coordinate system?

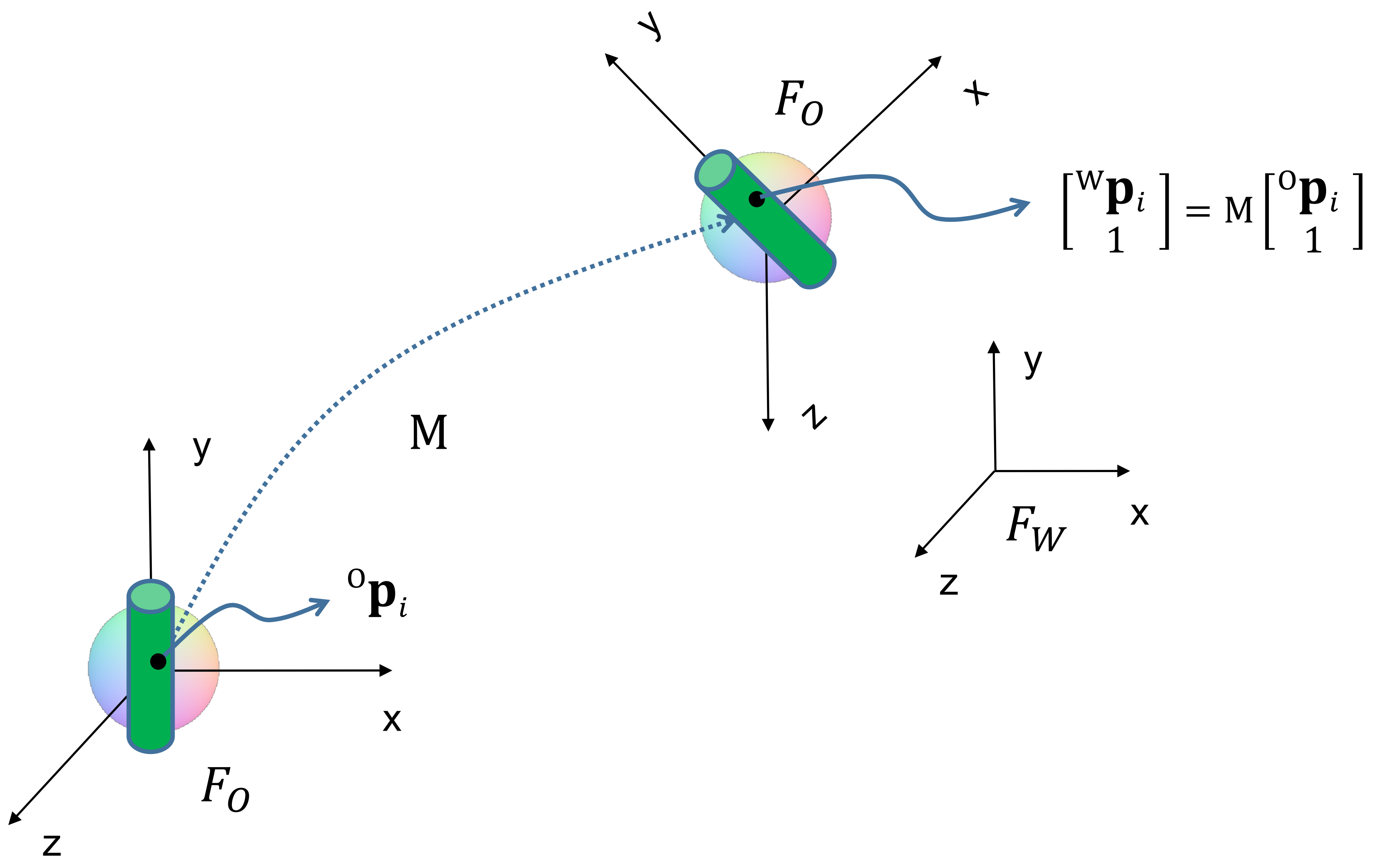
Recall: Modeling Transformation matrix:

$$\mathrm{M} = {}^{W}\mathrm{T}_{O} = \begin{bmatrix} {}^{W}\mathrm{R}_{\mathrm{O}} & {}^{W}\mathbf{r} \\ \mathbf{0}^{\mathrm{T}} & 1 \end{bmatrix}$$

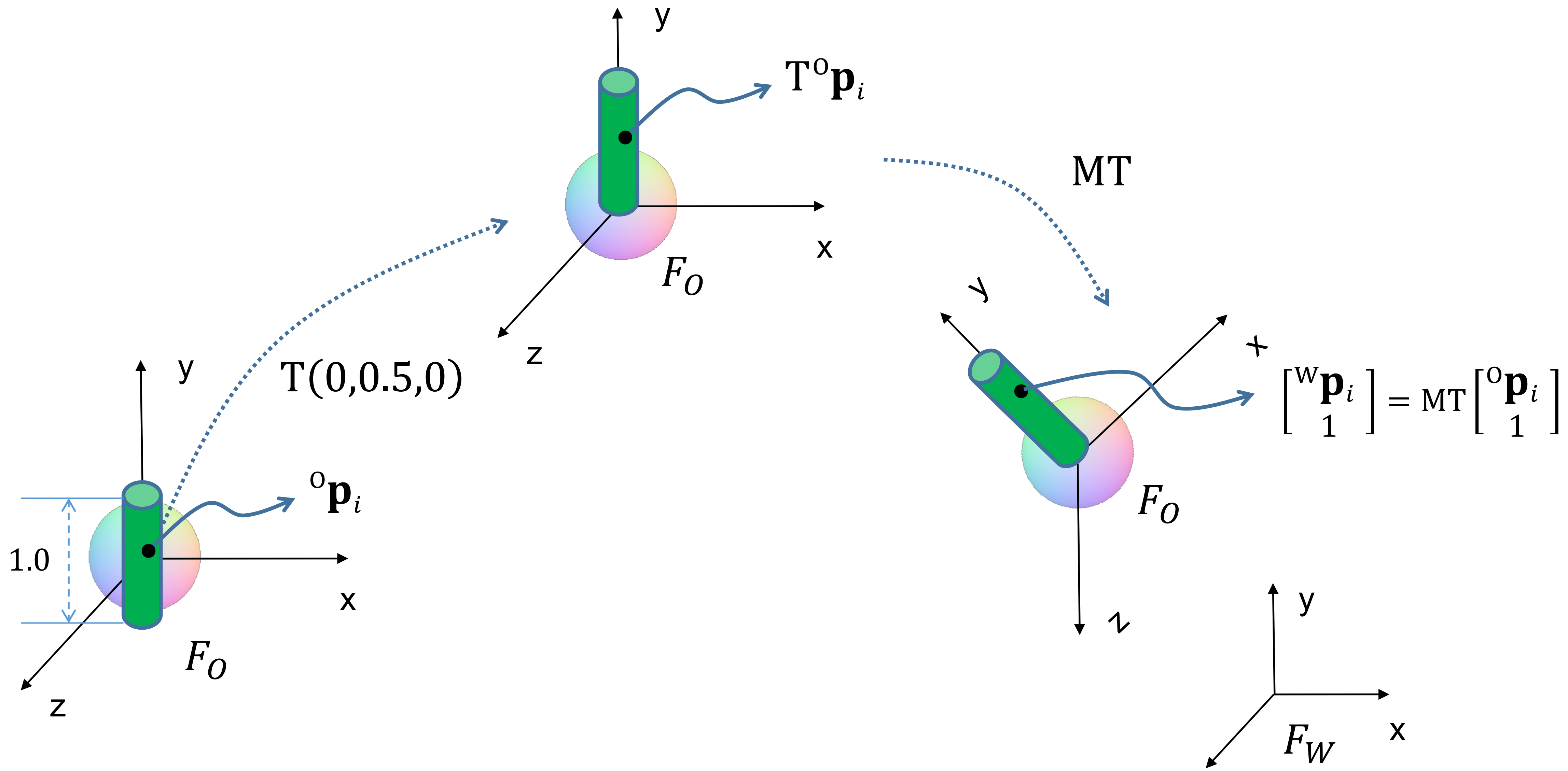
Here, ${}^{W}\mathrm{R}_{\mathrm{O}} \triangleq [{}^{W}\mathbf{x}_{\mathrm{O}} \quad {}^{W}\mathbf{y}_{\mathrm{O}} \quad {}^{W}\mathbf{z}_{\mathrm{O}}]$.




y
x
z
F_O
$\begin{bmatrix} {}^W\mathbf{y}_O \\ 1 \end{bmatrix} = \mathrm{M} \begin{bmatrix} {}^O\mathbf{y}_O \\ 1 \end{bmatrix}$
M
y
x
z
F_W
y
${}^O\mathbf{y}_O = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$
x
z
F_O



$$\begin{bmatrix} {}^{W}\mathbf{p}_i \\ 1 \end{bmatrix} = \mathrm{M} \begin{bmatrix} {}^{O}\mathbf{p}_i \\ 1 \end{bmatrix}$$

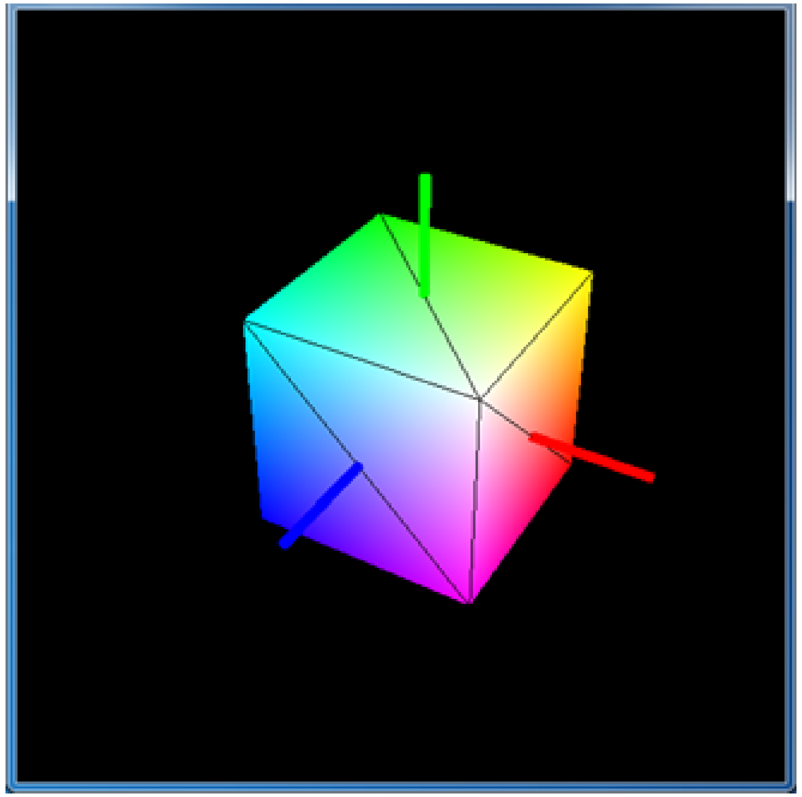


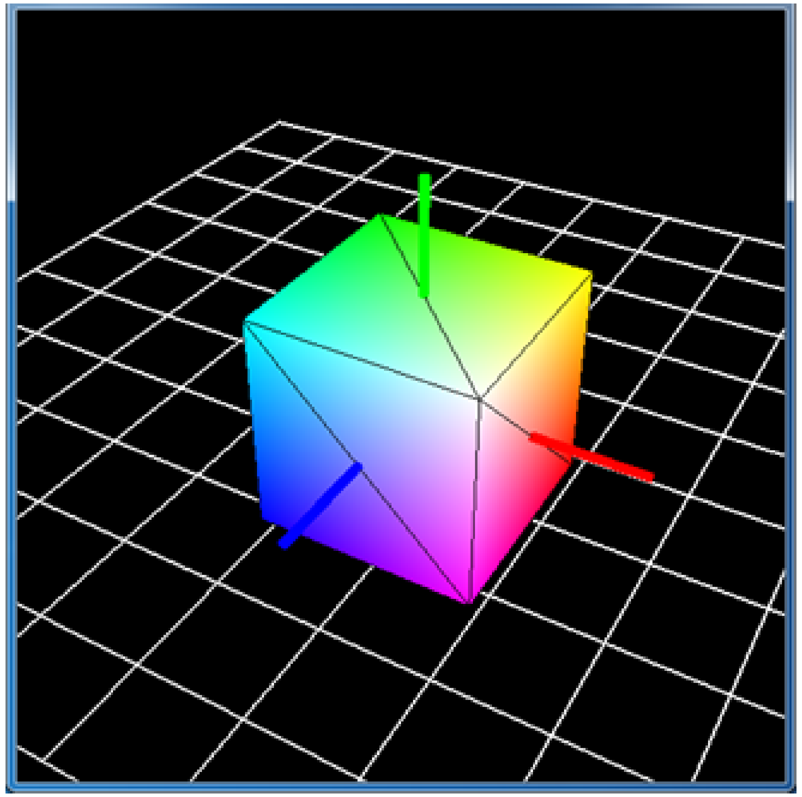
1.0
y
x
z
F_O
${}^{O}\mathbf{p}_i$
T(0,0.5,0)
y
x
z
F_O
$\mathrm{T}\,{}^{O}\mathbf{p}_i$
MT
y
x
z
F_O
$\begin{bmatrix} {}^{W}\mathbf{p}_i \\ 1 \end{bmatrix} = \mathrm{MT} \begin{bmatrix} {}^{O}\mathbf{p}_i \\ 1 \end{bmatrix}$
y
x
z
F_W

You may do the similar transformations to draw the other axes.

Requirement 3

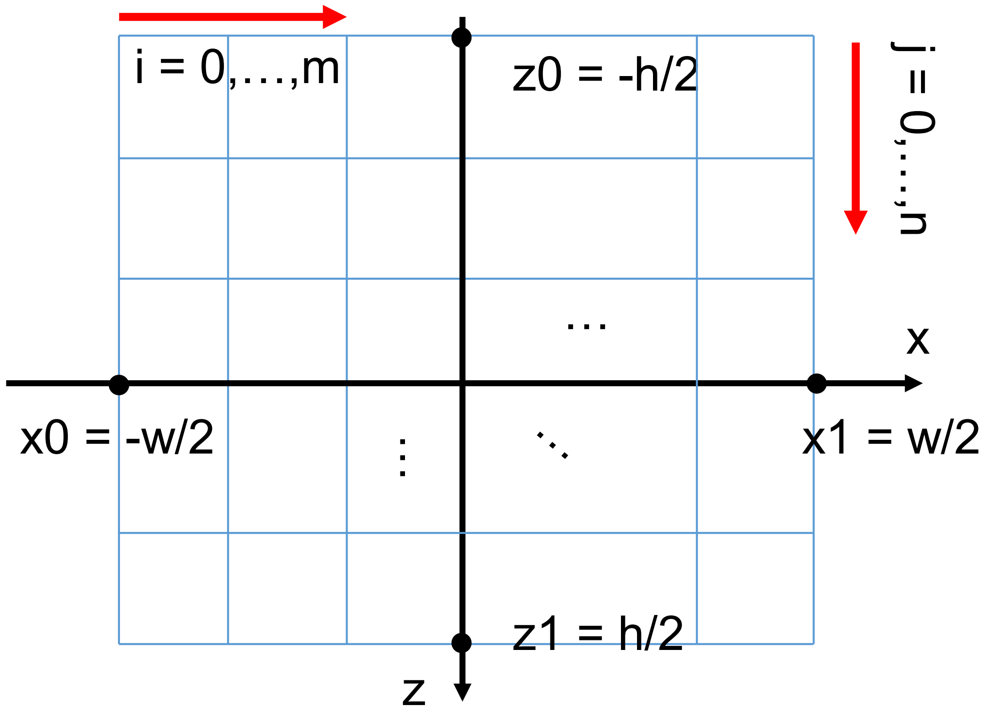
- If the user presses the ‘g’ key, the application should toggle on/off the visibility for a 10 x 10 grid on the x-z plane. You can draw a grid using GL_LINES primitives in OpenGL.





- How to make a grid geometry

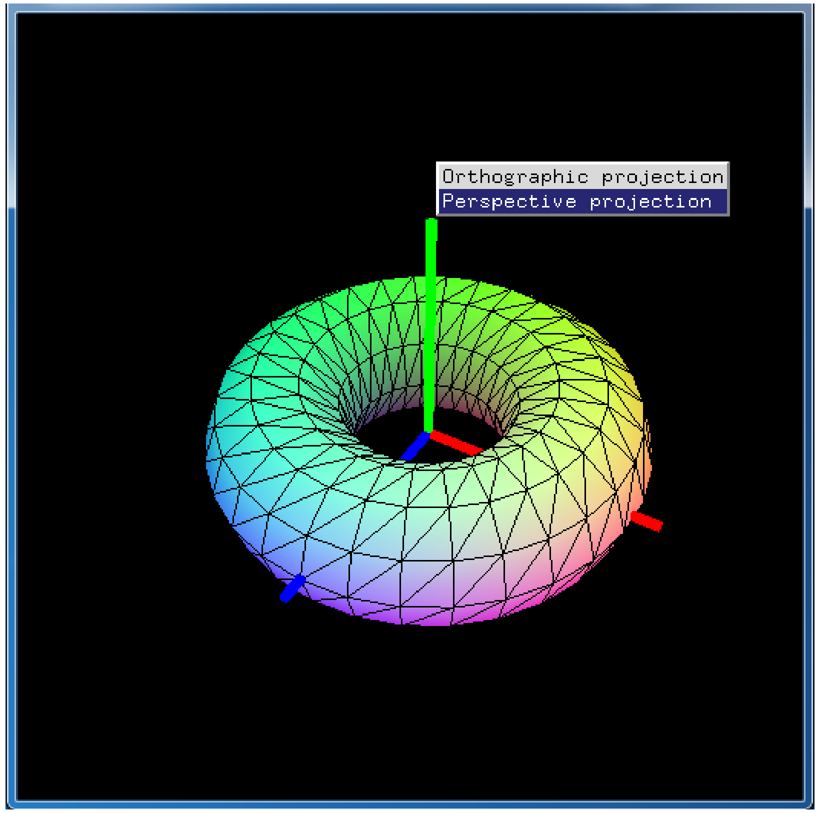
- Parameters
  - w: width of the grid
  - h: height of the grid
  - m: # of cells along the x-axis
  - n: # of cells along the z-axis



```
void get_grid(std::vector<GLfloat>& p, GLfloat w,
              GLfloat h, int m, int n)
{
    GLfloat x0 = -0.5f * w;
    GLfloat x1 = +0.5f * w;
    GLfloat z0 = -0.5f * h;
    GLfloat z1 = +0.5f * h;
    for (int i = 0; i <= m; ++i) {
        GLfloat x = x0 + w*i / m;
        FPUSH_VTX3(p, x, 0, z0);
        FPUSH_VTX3(p, x, 0, z1);
    }
    for (int i = 0; i <= n; ++i) {
        GLfloat z = z0 + h*i / n;
        FPUSH_VTX3(p, x0, 0, z);
        FPUSH_VTX3(p, x1, 0, z);
    }
}
```

Requirement 4

- When the user clicks the right mouse button, the application should show a pop-up menu to let the user select one of the orthographic and the perspective projection modes.



* Use the GLUT menu callback functions.

Requirement 5

- According to the mouse interaction, the viewing (camera) control must be done as follows:
  - (a) **Mouse Left Button + Alt + Drag**: Works as the **Tumble tool** does
  - (b) **Mouse Middle Button + Alt + Drag**: Works as the **Track tool** does
  - (c) **Scroll up / down**: Works as the **Zoom tool** does
  - (d) **Scroll up / down + Alt**: Works as the **Dolly tool** does

- How to track mouse movement and button clicks.

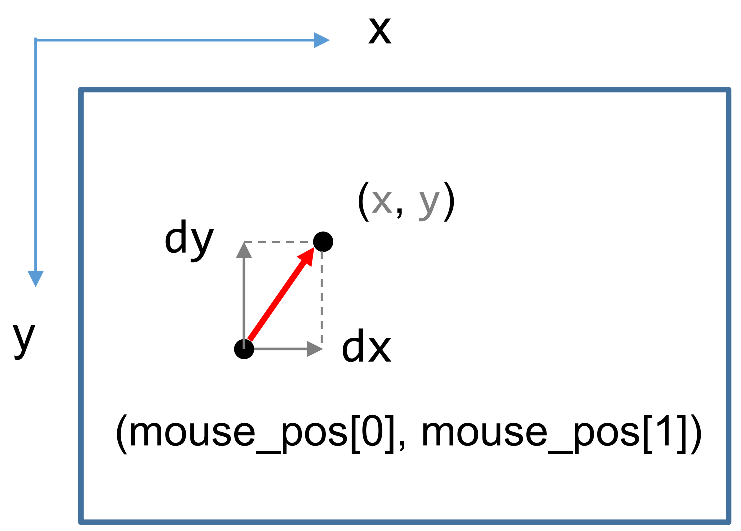
Global variables:

```
int button_pressed[3] = {GLUT_UP, GLUT_UP, GLUT_UP};
int mouse_pos[2] = { 0, 0 };
```

Mouse callbacks:

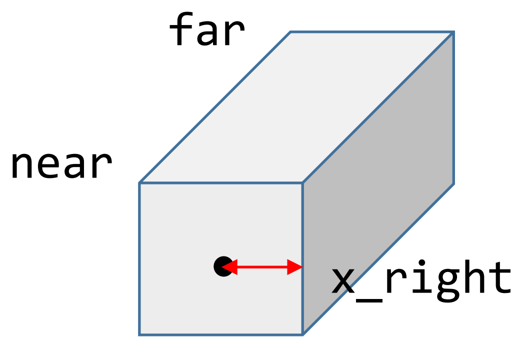
```
void mouse(int button, int state, int x, int y)
{
    button_pressed[button] = state;
    mouse_pos[0] = x;
    mouse_pos[1] = y;
}
```

```
void motion(int x, int y)
{
    using namespace glm;

    int modifiers = glutGetModifiers();
    int is_alt_active = modifiers & GLUT_ACTIVE_ALT;
    int is_ctrl_active = modifiers & GLUT_ACTIVE_CTRL;
    int is_shift_active = modifiers & GLUT_ACTIVE_SHIFT;

    int w = glutGet(GLUT_WINDOW_WIDTH);
    int h = glutGet(GLUT_WINDOW_HEIGHT);
    GLfloat dx = 1.f*(x - mouse_pos[0]) / w;
    GLfloat dy = -1.f*(y - mouse_pos[1]) / h;

    ... Add code here to deal with mouse motion ...

     mouse_pos[0] = x;
     mouse_pos[1] = y;
     glutPostRedisplay();

}
```



- Camera structure

```
struct Camera
{
    enum { ORTHOGRAPHIC, PERSPECTIVE };
    glm::vec3 eye;
    glm::vec3 center;
    glm::vec3 up;
    float zoom_factor;
    int projection_mode;
    float z_near;
    float z_far;
    float fovy;
    float x_right;

    ... See the next slides ...
};
```



View volume of orthographic projection

- Constructor

```
Camera() :
    eye(0, 0, 8),
    center(0, 0, 0),
    up(0, 1, 0),
    zoom_factor(1.0f),
    projection_mode(ORTHOGRAPHIC),
    z_near(0.01f),
    z_far(100.0f),
    fovy((float)(M_PI/180.0*(30.0))),
    x_right(1.2f)
{}
```

- Member functions

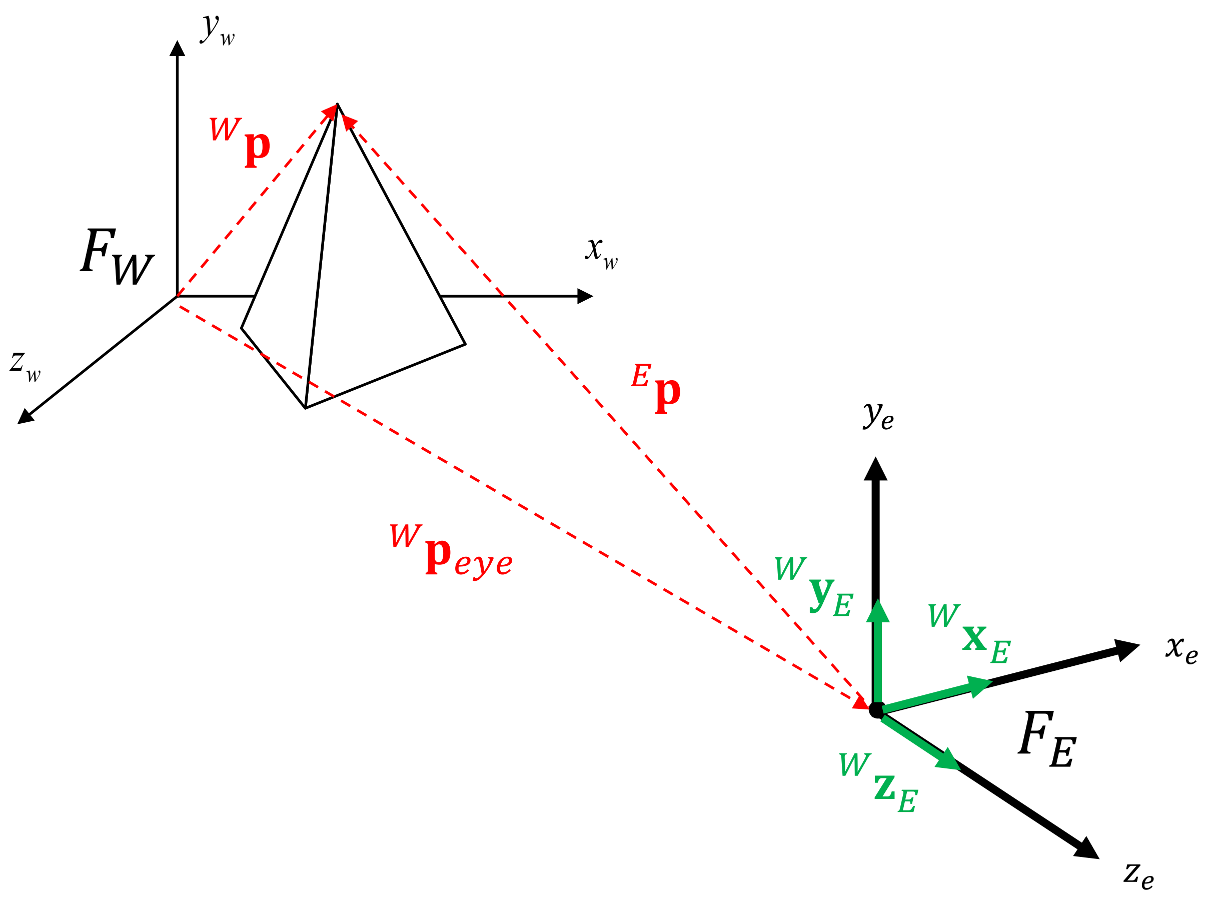
```
glm::mat4 get_viewing() { return glm::lookAt(eye, center, up); }
```

```
glm::mat4 get_projection(float aspect)
{
    glm::mat4 P;
    switch (projection_mode)
    {
    case ORTHOGRAPHIC:
        P = parallel(zoom_factor*x_right, aspect, z_near, z_far);
        break;

    case PERSPECTIVE:
        P = glm::perspective(zoom_factor*fovy, aspect, z_near, z_far);
        break;
    }
    return P;
}
```

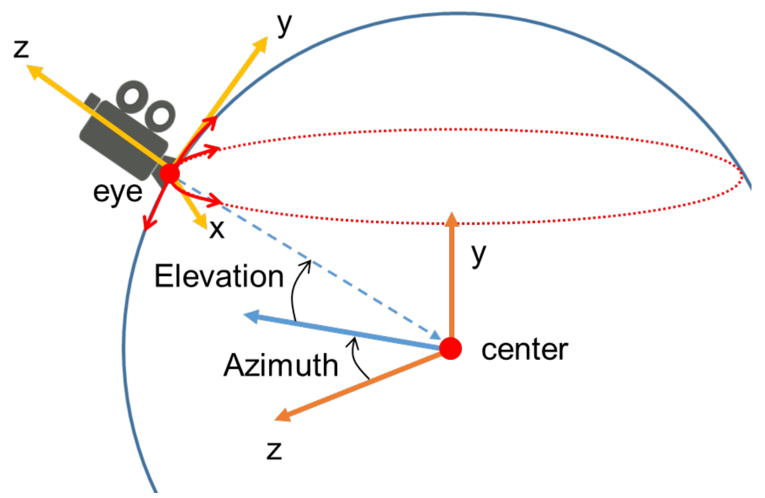
- Recall: Viewing transformation matrix

$$\mathrm{V} = {}^{E}\mathrm{T}_{W} = \begin{bmatrix} {}^{E}\mathrm{R}_{W} & {}^{E}\mathrm{R}_{W}\left(-{}^{W}\mathbf{p}_{eye}\right) \\ \mathbf{0}^{\mathrm{T}} & 1 \end{bmatrix}$$

$${}^{E}\mathrm{R}_{W} = [{}^{W}\mathbf{x}_{E} \quad {}^{W}\mathbf{y}_{E} \quad {}^{W}\mathbf{z}_{E}]^{\mathrm{T}}$$



(a) **Mouse Left Button + Alt + Drag**: Works as the **Tumble tool** does



Global variable: `Camera camera;`

In the motion callback:

```
if (button_pressed[GLUT_LEFT_BUTTON] == GLUT_DOWN)
{
     if (is_alt_active)
     {
        vec4 disp(eye - center, 1);

        GLfloat alpha = 2.0f;
        mat4 V = camera.get_viewing();
        mat4 Rx = rotate(mat4(), alpha*dy, vec3(transpose(V)[0]));
        mat4 Ry = rotate(mat4(), -alpha*dx, vec3(0,1,0));
        mat4 R = Ry*Rx;
        camera.eye = camera.center + vec3(R*disp);
        camera.up = mat3(R)*camera.up;

      }
 }
```

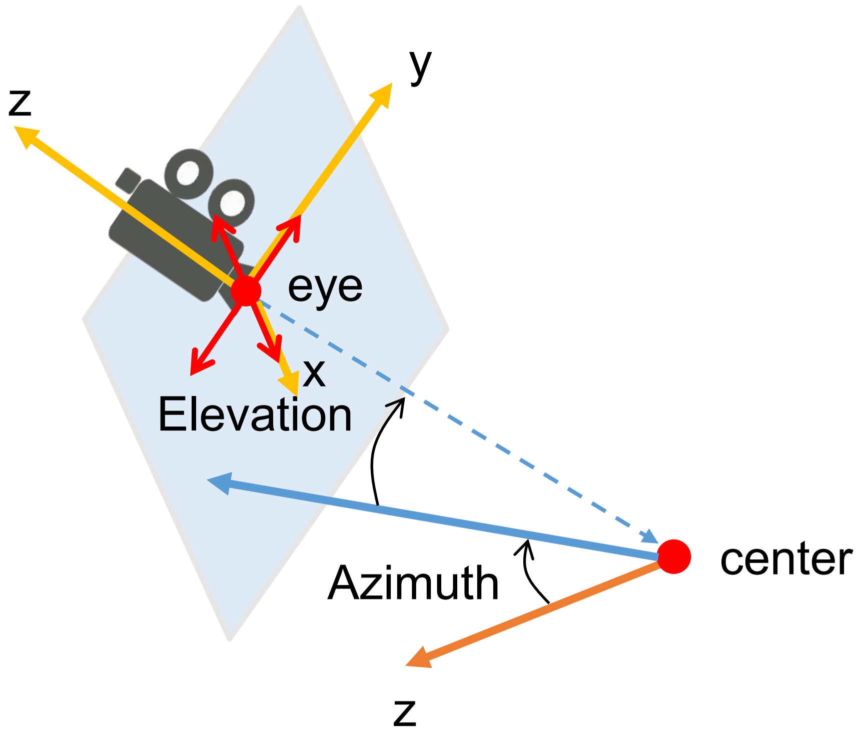
$\boxed{\texttt{vec3(transpose(V)[0])}} \rightarrow {}^{W}\mathbf{x}_{\mathrm{E}}$

Why?

$$V = {}^{E}T_{W} = \begin{bmatrix} {}^{E}R_{W} & {}^{E}R_{W}(-{}^{W}\mathbf{p}_{eye}) \\ \mathbf{0}^{T} & 1 \end{bmatrix}$$

$$({}^{E}R_{W} = [{}^{W}\mathbf{x}_{E} \quad {}^{W}\mathbf{y}_{E} \quad {}^{W}\mathbf{z}_{E}]^{T})$$

(b) **Mouse Middle Button + Alt + Drag**: Works as the **Track tool** does



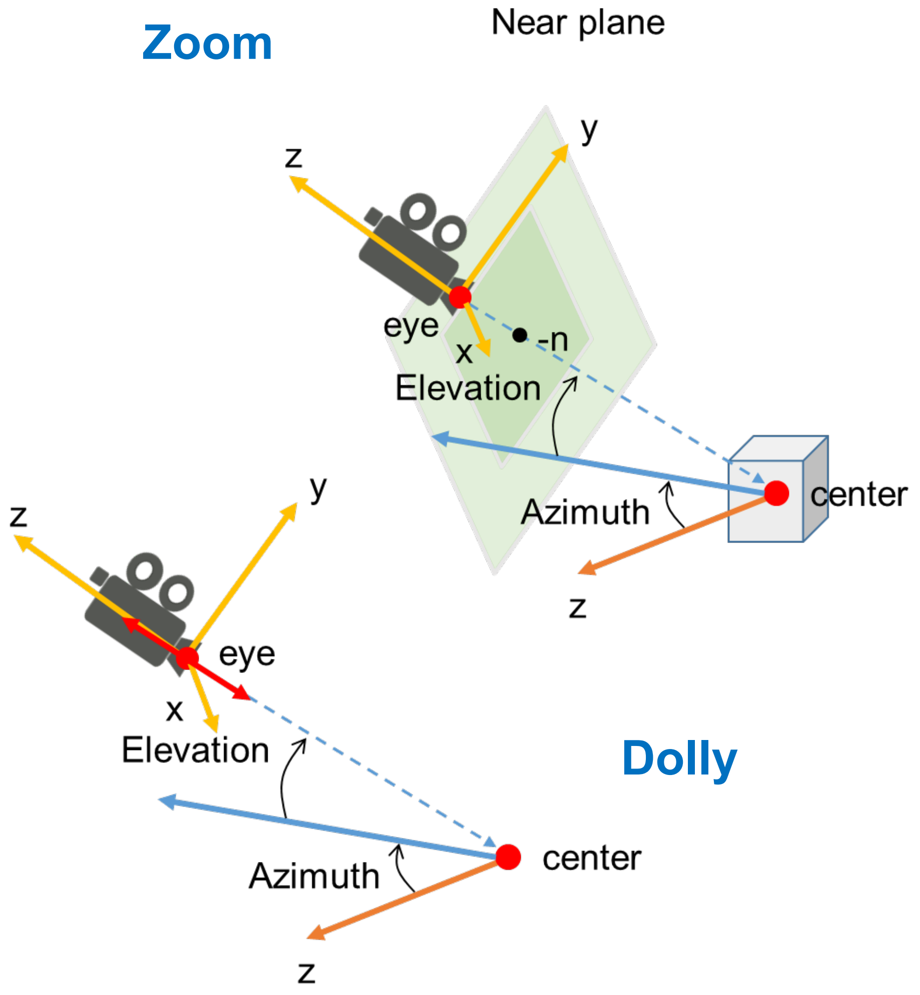
In the motion callback:

```
if (button_pressed[GLUT_MIDDLE_BUTTON] == GLUT_DOWN)
{
      if (is_alt_active) {
            mat4 VT = transpose(camera.get_viewing());
            camera.eye += vec3(-dx* VT[0] + -dy * VT[1]);
            camera.center += vec3(-dx* VT[0] + -dy * VT[1]);
      }
}
```

$\boxed{\text{VT[0]}} \rightarrow {}^{w}\mathbf{x}_E \qquad \boxed{\text{VT[1]}} \rightarrow {}^{w}\mathbf{y}_E$

(c) **Scroll up / down**: Works as the **Zoom tool** does

(d) **Scroll up / down + Alt**: Works as the **Dolly tool** does



```
void mouse_wheel(int wheel, int dir, int x, int y)
{
	int is_alt_active = glutGetModifiers() & GLUT_ACTIVE_ALT;

	if (is_alt_active) {
		glm::vec3 disp = camera.eye - camera.center;
		if (dir > 0)
			camera.eye = camera.center + 0.95f*disp;
		else
			camera.eye = camera.center + 1.05f*disp;
	}
	else
	{
		if (dir > 0)
			camera.zoom_factor *= 0.95f;
		else
			camera.zoom_factor *= 1.05f;
	}

	glutPostRedisplay();
}
```

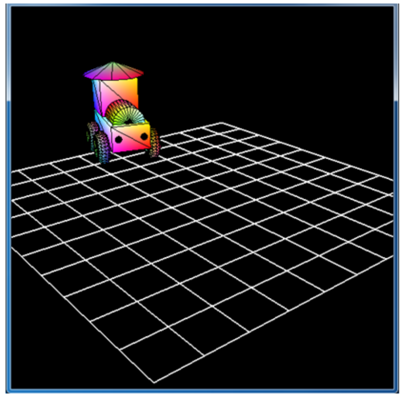
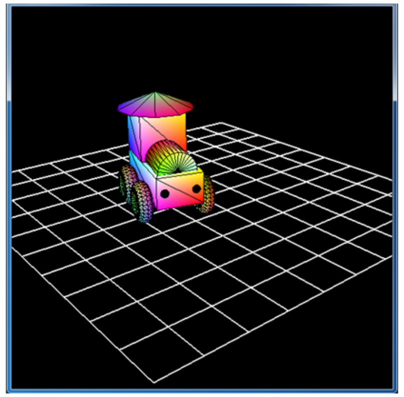
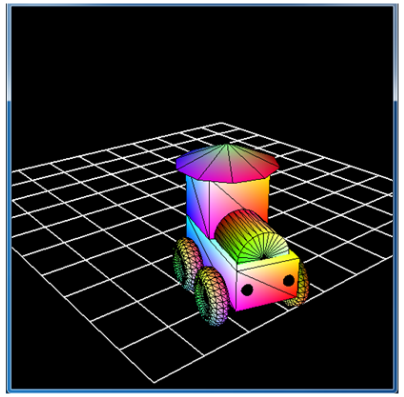
Requirement 6

- According to the user's keyboard input, the position and rotation of the model must be interactively changed as follows:

  (a) **Right arrow / Left arrow**:
  Add a positive / negative offset to the position of the model along the x-axis of the local coordinate system.

  (b) **Up arrow / Down arrow**:
  Add a positive / negative offset to the y coordinate of the position of the model.

  (c) **Home key / End key**:
  Rotate the model by a positive / negative offset angle around the y-axis of the world coordinate system, i.e., $[0 \quad 1 \quad 0]^T$.

Requirement 7

- As the car model's position and rotation are changed by the keyboard input above, the tires of the car must be rotated as well:

  (a) **Right arrow / Left arrow**:
  All the tires are rotated in the same direction to move backward / forward.

  (b) **Home key / End key**:
  The two left tires are rotated in an opposite direction to their respective right tires while turning right / left.

- What to submit:
  - A **zip file** that compresses the following files:
    - **Project source files** except libraries.
      - Clean your project before compression by selecting **Build → Clean Solution** in the main menu.
  - File name format
    - **hw9_000000.zip**, where 000000 must be replaced by your own student ID.
- Due date: **To be announced later**