Advanced Computer Graphics
Lab 5

Today’s roadmap

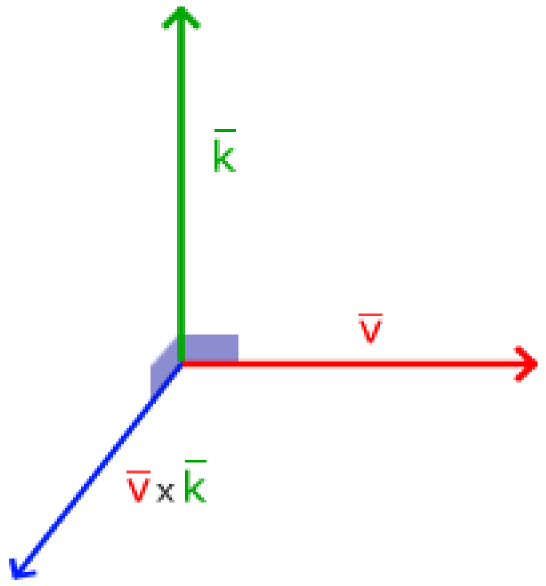
- From 2D -> 3D
- Transforms in 3D (scaling, translating, rotating, combining them)
- MVP matrix, perspective projection

3D Transformations – recap

- Vectors, matrices
- Dot product:
  - $\bar{v} \cdot \bar{k} = ||\bar{v}|| \cdot ||\bar{k}|| \cdot \cos\theta$

- Cross product

$$\begin{pmatrix} A_x \\ A_y \\ A_z \end{pmatrix} \times \begin{pmatrix} B_x \\ B_y \\ B_z \end{pmatrix} = \begin{pmatrix} A_y \cdot B_z - A_z \cdot B_y \\ A_z \cdot B_x - A_x \cdot B_z \\ A_x \cdot B_y - A_y \cdot B_x \end{pmatrix}$$



3D Transformations – matrices

- Scaling

$$\begin{bmatrix} S_1 & 0 & 0 & 0 \\ 0 & S_2 & 0 & 0 \\ 0 & 0 & S_3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} S_1 \cdot x \\ S_2 \cdot y \\ S_3 \cdot z \\ 1 \end{pmatrix}$$

- Translation

$$\begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x + T_x \\ y + T_y \\ z + T_z \\ 1 \end{pmatrix}$$

- Rotation

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta \cdot x - \sin\theta \cdot y \\ \sin\theta \cdot x + \cos\theta \cdot y \\ z \\ 1 \end{pmatrix}$$

Class assignment – part 1

1) Have a look at our example that draws multiple rotating squares into 3D. Change it so that it draws cubes.

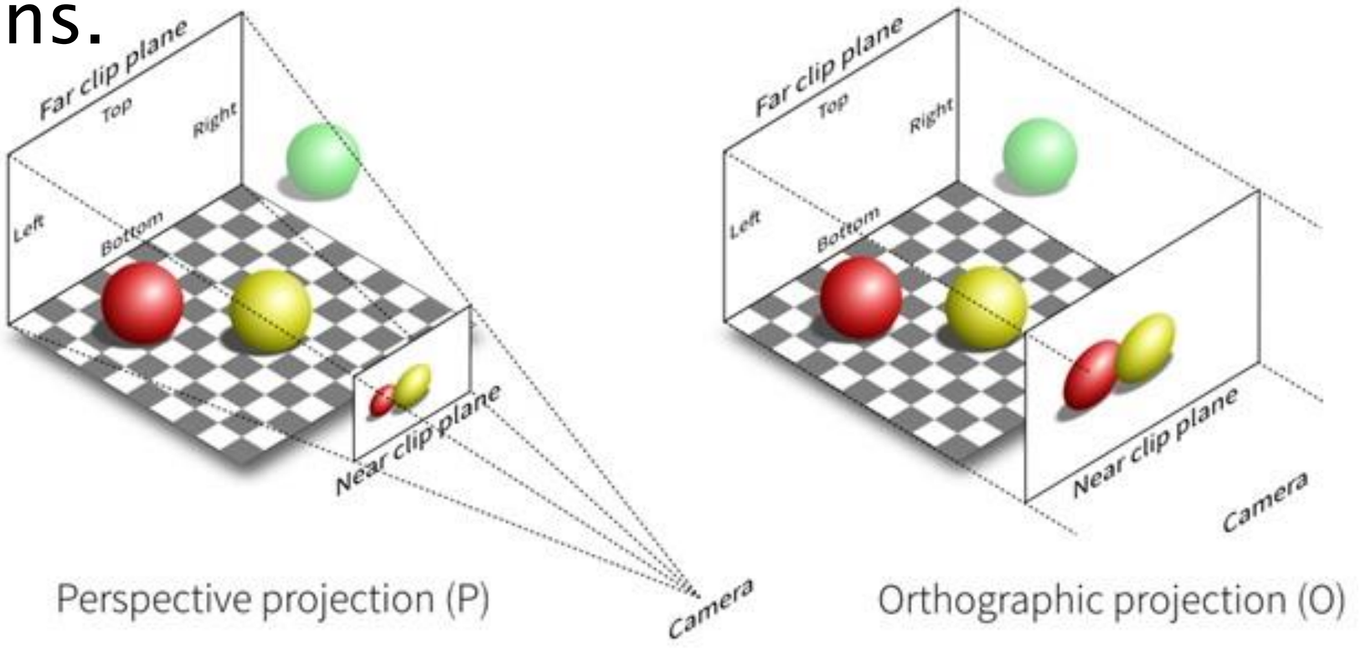
You can use the following (or anything else of your choice):

Vertices:
```
// front
0.0, 0.0,  0.05,
0.05, 0.0,  0.05,
0.0,  0.05,  0.05,
0.05,  0.05,  0.05,
// back
0.0, 0.0, 0.0,
0.05, 0.0, 0.0,
0.0,  0.05, 0.0,
0.05,  0.05, 0.0
```

Indices:
```
0, 1, 2,
1, 3, 2,
2, 3, 7,
2, 7, 6,
1, 7, 3,
1, 5, 7,
6, 7, 4,
7, 5, 4,
0, 4, 1,
1, 4, 5,
2, 6, 4,
0, 2, 4
```

Weird…looking result

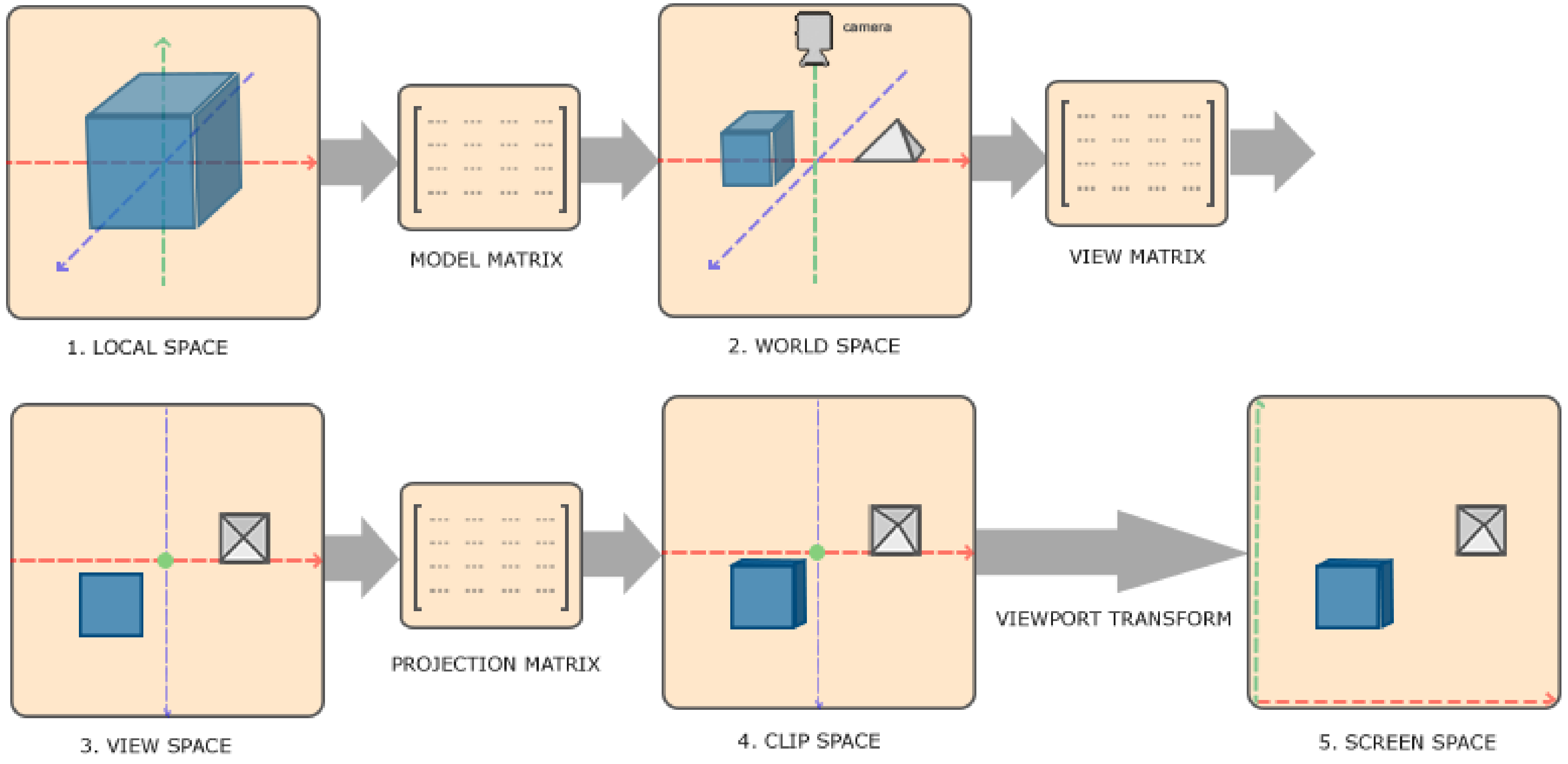
- Why? By default, OpenGL uses orthographic projections (usually used for 2D)
- We need to add depth by using perspective projections.



Perspective projection (P)

Orthographic projection (O)

Coordinate systems

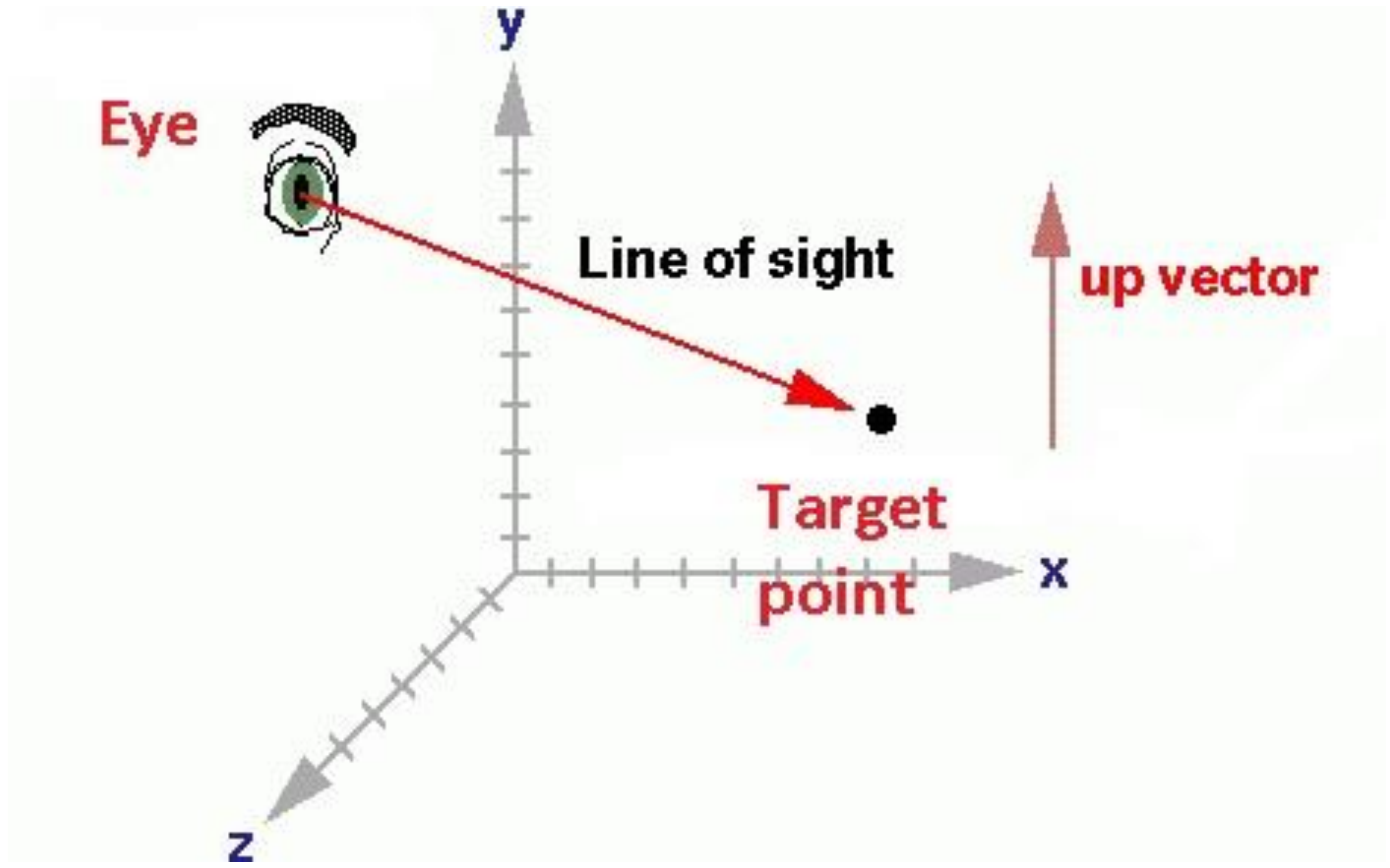


MVP – 3 types of matrices

- **Model matrix** – transforms of current object:
  - glm::mat4 model;
  - model = glm::rotate(model, 45.0f, glm::vec3(0.0f, 1.0f, 0.0f));

- **View matrix** (observer position, view direction, up vector) – transforms all the world coordinates into view coordinates that are relative to the viewer's position and direction
  - glm::mat4 view;
  - view = glm::lookAt(glm::vec3(0.0f, 30.0f, 100.0f), glm::vec3(0.0f, 0.0f, -1.0f), glm::vec3(0.0f, 1.0f, 0.0f));

View matrix



MVP – 3 types of matrices

- **Projection matrix**
  - Perspective projection
    - //glm::perspective(fov, aspect, near plane, far plane);
    - Ex: glm::mat4 projection = **glm::perspective**(glm::radians(45.0f), screenWidth / screenHeight, 0.1f, 10000.0f);
  - Orthographic projection
    - //projection = **glm::ortho**(left, right, bottom, top, zNear, zFar)

- Question: In which order we should multiply these matrices? Why?

Class assignment – part 2

2) Create the View and Projection matrices. Calculate the MVP matrix. Send it to the shader as the new transform matrix and observe the result.

3) Play with the parameters of the **lookAt** and **perspective** functions and observe the differences. For instance, try to make the view direction Oy and change the up vector accordingly. Or move the viewer closer to the scene.

For better observing the results, color the cubes differently.

Ex: `color = glm::vec4(0.5f + i / 10.0f, 1 - i / 10.0f, 0.5f, 1.0)`

Obs: you should take a first screenshot here for the Moodle upload, using your own parameters for view & projection ☺

Class assignment – part 2

4) Leave 3 cubes static (cubes 0, 4, 8) and rotate just the others.

5) Add a second set of shaders for a white light source. Use the light position and light color that are already defined.

a) See if the light is rendered by changing the light position to a visible position

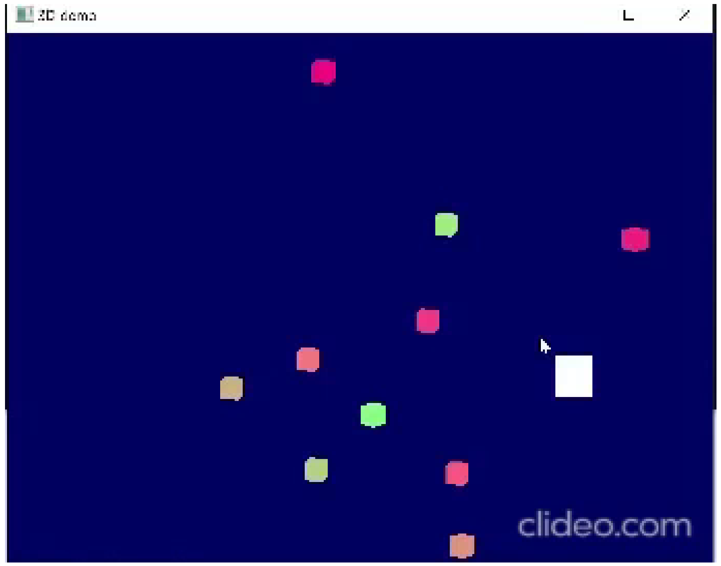
b) Add a nice animation to it by changing its position at each frame. The animation should start AFTER the user presses the "A" key. Ex of animation:

```
lightPos.x = sin(glfwGetTime());
lightPos.y = sin(glfwGetTime() / 8.0f) * 1.0f + 0.5f;
```

Obs: you should take a second screenshot here for the Moodle upload, with the light source ☺

Class assignment – bonus

Make the light source “dance” on the screen (move it based on your mouse dragging). The move should happen only when holding the left mouse button pressed and stop in the current position when released.


3D demo
clideo.com