Computer Graphics

9 - Lab - Orientation & Rotation

Yoonsang Lee
Hanyang University

Spring 2025

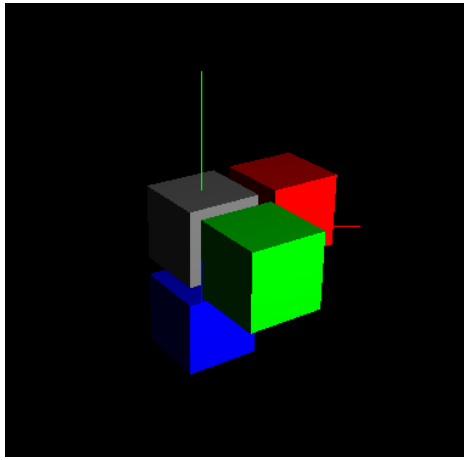
Outline

- Euler Angles
- Slerp

Euler Angles

[Code] 1-euler-angles

- This example implements ZYX Euler angles.
  - But you can easily change the type of Euler angles by changing the multiplication order.



- Rotating cubes
- Two groups to render, each moving differently:
  - World frame
  - Four cubes

[Code] 1-euler-angles

- Two VAOs
  - Frame VAO - `prepare_vao_frame()`
    - from the previous examples that draw frames
  - Cube VAO - `prepare_vao_cube()`
    - from the previous examples that draw cubes

- Two shader programs
  - Shader for frame - `shader_color`
    - use vertex color specified as a vertex attribute
    - shader code is from the previous examples that draw frames
  - Shader for cubes - `shader_lighting`
    - calculate fragment color by Phong illumination model
    - shader code is from "8-Lab-Lighing/4-all-components-phong-facenorm.py"

[Code] 1-euler-angles

```
def draw_frame(vao, MVP, unif_locs):
    glUniformMatrix4fv(unif_locs['MVP'], 1, GL_FALSE, glm.value_ptr(MVP))
    glBindVertexArray(vao)
    glDrawArrays(GL_LINES, 0, 6)

def draw_cube(vao, MVP, M, matcolor, unif_locs):
    glUniformMatrix4fv(unif_locs['MVP'], 1, GL_FALSE, glm.value_ptr(MVP))
    glUniformMatrix4fv(unif_locs['M'], 1, GL_FALSE, glm.value_ptr(M))
    glUniform3f(unif_locs['material_color'], matcolor.r, matcolor.g, matcolor.b)
    glBindVertexArray(vao)
    glDrawArrays(GL_TRIANGLES, 0, 36)

def main():
    ...
    # load shaders & get uniform locations
    shader_lighting = load_shaders(g_vertex_shader_src_lighting, g_fragment_shad...)
    unif_names = ['MVP', 'M', 'view_pos', 'material_color']
    unif_locs_lighting = {}
    for name in unif_names:
        unif_locs_lighting[name] = glGetUniformLocation(shader_lighting, name)

    shader_color = load_shaders(g_vertex_shader_src_color, g_fragment_shader_src...)
    unif_names = ['MVP']
    unif_locs_color = {}
    for name in unif_names:
        unif_locs_color[name] = glGetUniformLocation(shader_color, name)
    ...
```

[Code] 1-euler-angles

```
while not glfwWindowShouldClose(window):
    ...
    # projection matrix
    P = glm.perspective(45, 1, 1, 20)

    # view matrix
    view_pos =
glm.vec3(5*np.sin(g_cam_ang),g_cam_height,5*np.cos(g_cam_ang))
    V = glm.lookAt(view_pos, glm.vec3(0,0,0), glm.vec3(0,1,0))

    # draw world frame
    glUseProgram(shader_color)
    draw_frame(vao_frame, P*V, unif_locs_color)

    # ZYX Euler angles
    t = glfwGetTime()
    xang = t
    yang = glm.radians(30)
    zang = glm.radians(30)
    Rx = glm.rotate(xang, (1,0,0))
    Ry = glm.rotate(yang, (0,1,0))
    Rz = glm.rotate(zang, (0,0,1))
    M = glm.mat4(Rz * Ry * Rx)
```

[Code] 1-euler-angles

```
    # set view_pos uniform in shader_lighting
    glUseProgram(shader_lighting)
    glUniform3f(unif_locs_lighting['view_pos'], view_pos.x, view_pos.y,
view_pos.z)

    # draw cubes
    M = M * glm.scale((.25, .25, .25))

    Mo = M * glm.mat4()
    draw_cube(vao_cube, P*V*Mo, Mo, glm.vec3(.5,.5,.5),
unif_locs_lighting)

    Mx = M * glm.translate((2.5,0,0))
    draw_cube(vao_cube, P*V*Mx, Mx, glm.vec3(1,0,0), unif_locs_lighting)

    My = M * glm.translate((0,2.5,0))
    draw_cube(vao_cube, P*V*My, My, glm.vec3(0,1,0), unif_locs_lighting)

    Mz = M * glm.translate((0,0,2.5))
    draw_cube(vao_cube, P*V*Mz, Mz, glm.vec3(0,0,1), unif_locs_lighting)
```

Slerp

Recall: Slerp

- $\text{slerp}(\mathbf{R}_1, \mathbf{R}_2, t) = \mathbf{R}_1(\mathbf{R}_1^T\mathbf{R}_2)^t$
  $= \mathbf{R}_1\exp(\text{t} \cdot \log(\mathbf{R}_1^T\mathbf{R}_2))$

- This example implements this formula.

Alternatives

- But you can implement and test following alternatives:
  - Quaternion slerp:
    - $\text{slerp}(\mathbf{q}_1, \mathbf{q}_2, t) = \mathbf{q}_1(\mathbf{q}_1^{-1}\mathbf{q}_2)^t$
  - Geometric slerp (equivalent):
    - $\text{slerp}(\mathbf{q}_1, \mathbf{q}_2, t) = \frac{\sin((1-t)\varphi)}{\sin\varphi}\mathbf{q}_1 + \frac{\sin(t\varphi)}{\sin\varphi}\mathbf{q}_2$
      - φ: the angle subtended by the arc ($\cos\varphi = \mathbf{q}_1 \cdot \mathbf{q}_2$)
  - glm.slerp(x: quat, y: quat, a: float) -> quat

Exp & Log

- This examples uses PyGLM functions for exp() and log().

- But for log(), PyGLM does not have a function to directly convert a rotation matrix to a rotation vector.

- So we convert it like: rotation matrix → unit quaternion → rotation vector

[Code] 2-slerp

```
def ZYXEulerToRotMat(angles):
    zang, yang, xang = angles
    Rx = glm.rotate(xang, (1,0,0))
    Ry = glm.rotate(yang, (0,1,0))
    Rz = glm.rotate(zang, (0,0,1))
    return glm.mat3(Rz * Ry * Rx)

def slerp(R1, R2, t):
    return R1 * exp( t * log(glm.transpose(R1) * R2) )

eps = 1e-6
def exp(rotvec):
    angle = glm.l2Norm(rotvec)
    if angle > eps:
        axis = glm.normalize(rotvec)
        return glm.mat3(glm.rotate(angle, axis))
    else:
        return glm.mat3()

def log(rotmat):
    quat = glm.quat(rotmat)
    return glm.angle(quat) * glm.axis(quat)
```

[Code] 2-slerp

```
...
# start orientation: ZYX Euler angles - rot z by -90 deg then rot y by 90 then rot x by 0
R1 = ZYXEulerToRotMat((-np.pi*.5, np.pi*.5, 0))

# end orientation: ZYX Euler angles - rot z by 0 then rot y by 0 then rot x by 90
R2 = ZYXEulerToRotMat((0, 0, np.pi*.5))

while not glfwWindowShouldClose(window):
    ...
    # t is repeatedly increasing from 0.0 to 1.0
    t = glfwGetTime() % 3 / 3

    # slerp
    R = slerp(R1, R2, t)
    M = glm.mat4(R)


    ...
    # draw cubes
    M = M * glm.scale((.25, .25, .25))

    Mo = M * glm.mat4()
    draw_cube(vao_cube, P*V*Mo, Mo, glm.vec3(.5,.5,.5), unif_locs_lighting)

    Mx = M * glm.translate((2.5,0,0))
    draw_cube(vao_cube, P*V*Mx, Mx, glm.vec3(1,0,0), unif_locs_lighting)
    ...
```

Time for Assignment

- Let's start today's assignment.
- TA will guide you.