<Class Assignment 1>

Simple 3D Viewer

Course: CSE4020 Computer Graphics (11272)

Instructor: Yoonsang Lee

Student Name: Jungtae Lee

Student ID: 2018014275

1. Requirements & Implementations
2. Manipulating camera with mouse movement (70pts)

Get input

glfw.get\_mouse\_button()

glfw.get\_cursor\_pos()

glfw.get\_key()

glfw.set\_scroll\_callback()

Process input

branch when first pressed vs kept pressed

store values and states in buffer & temp variables

(LMBbuff, RMBbuff, azbuff, evbuff, azbufftemp, etc)

store

Compute eye & center

Pass to gluLookAt()

update\_cam() computes viewspace (u, v, w) with orbit, pan, zoom data &

then computes eye & center with u, v, w

glfw.PRESS

glfw.RELEASE

xpos,ypos,…

az,ev,panUV, zoom,…

1. Orbit & Pan input

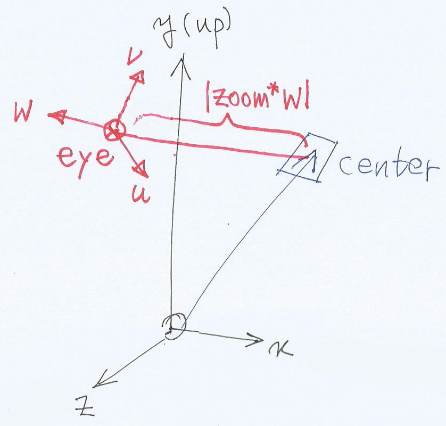
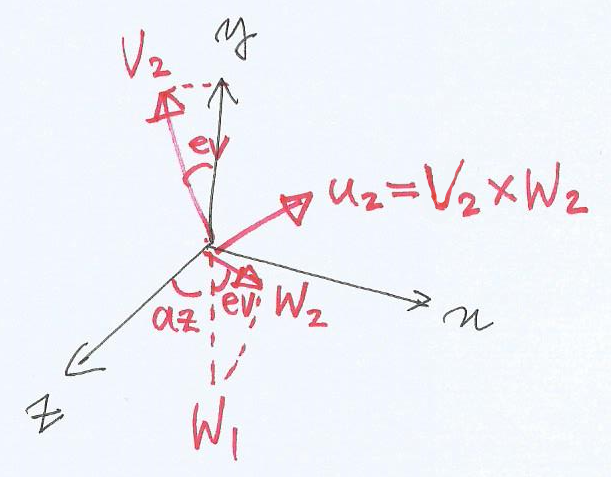
* Get input: LMB state and cursor position can be queried directly in render loop with glfw.get\_mouse\_button() and glfw.get\_cursor\_pos()
* Process input: To drag intuitively, value persistence & accumulation is implemented via buffer variables (buff, bufftemp) and if-branching (switching LMBbuff, first time vs kept pressed)
* Getting & processing RMB input is almost identical to LMB

1. Zoom input

* Scroll wheel input cannot be queried with glfw.get -> set callback function

1. Computing Eye & Center to pass into gluLookAt()

(expressed in x,y,z coords)



1. Toggle perspective / orthogonal projection by pressing ‘v’key (10pts)
2. Define global variable bool PROJMODE (default to True)
3. In render loop, get key state with glfw.get\_key(glfw.KEY\_V)
4. When pressed, toggle PROJMODE between True and False
5. Reference PROJMODE to branch to glOrtho() or gluPerspective()
6. Draw a rectangular grid with lines on xz plane as a reference ground plane (5pts)
7. Define function to draw grid: drawGrid()
8. Calculate vertex points with numpy.arange() using parameters for grid size, minor grid step, and major grid step
9. Draw vertices with glVertex3f() and for-loop
10. Miscellaneous functionalities
11. drawAxes() draws X, Y, Z axes in Red, Green, Blue
12. Press ESC to close window
13. Press 0 to reset camera
14. Set control sensitivity with LMBsens, RMBsens, zoomsens
15. Set aspect ratio and window resolution (size) with aspRatio and height
16. Source Code
17. Manipulating camera with mouse movement
18. Input handling

* mouse input (nested inside render loop):

def update\_cam():

    global ev, az, panU, panV, zoom, center, eye

    u = np.array([np.cos(az), 0, -np.sin(az)])

    v = np.array([-np.sin(ev) \* np.sin(az),

                  np.cos(ev),

                  -np.sin(ev) \* np.cos(az)])

    w = np.array([np.cos(ev) \* np.sin(az),

                  np.sin(ev),

                  np.cos(ev) \* np.cos(az)])

    center = panU \* u + panV \* v

    eye = zoom \* w + center

    return

if glfw.get\_mouse\_button(window, glfw.MOUSE\_BUTTON\_LEFT) == glfw.PRESS:

    if LMBbuff:

        azbufftemp = RMBsens \* -float(glfw.get\_cursor\_pos(window)[0])

        evbufftemp = RMBsens \*  float(glfw.get\_cursor\_pos(window)[1])

        LMBbuff = False

    else:

        az = azbuff - azbufftemp + RMBsens \* -float(glfw.get\_cursor\_pos(window)[0])

        ev = evbuff - evbufftemp + RMBsens \*  float(glfw.get\_cursor\_pos(window)[1])

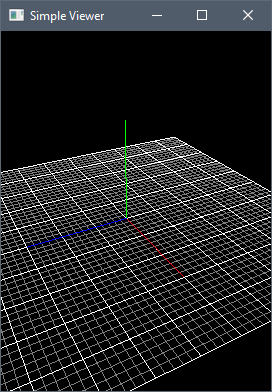
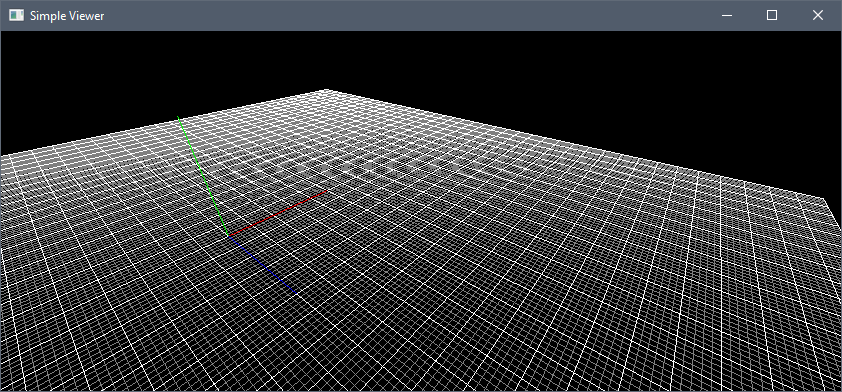
elif glfw.get\_mouse\_button(window, glfw.MOUSE\_BUTTON\_LEFT) == glfw.RELEASE:

    azbuff = az

    evbuff = ev

    LMBbuff = True

* Computing camera:

1.  Screenshots