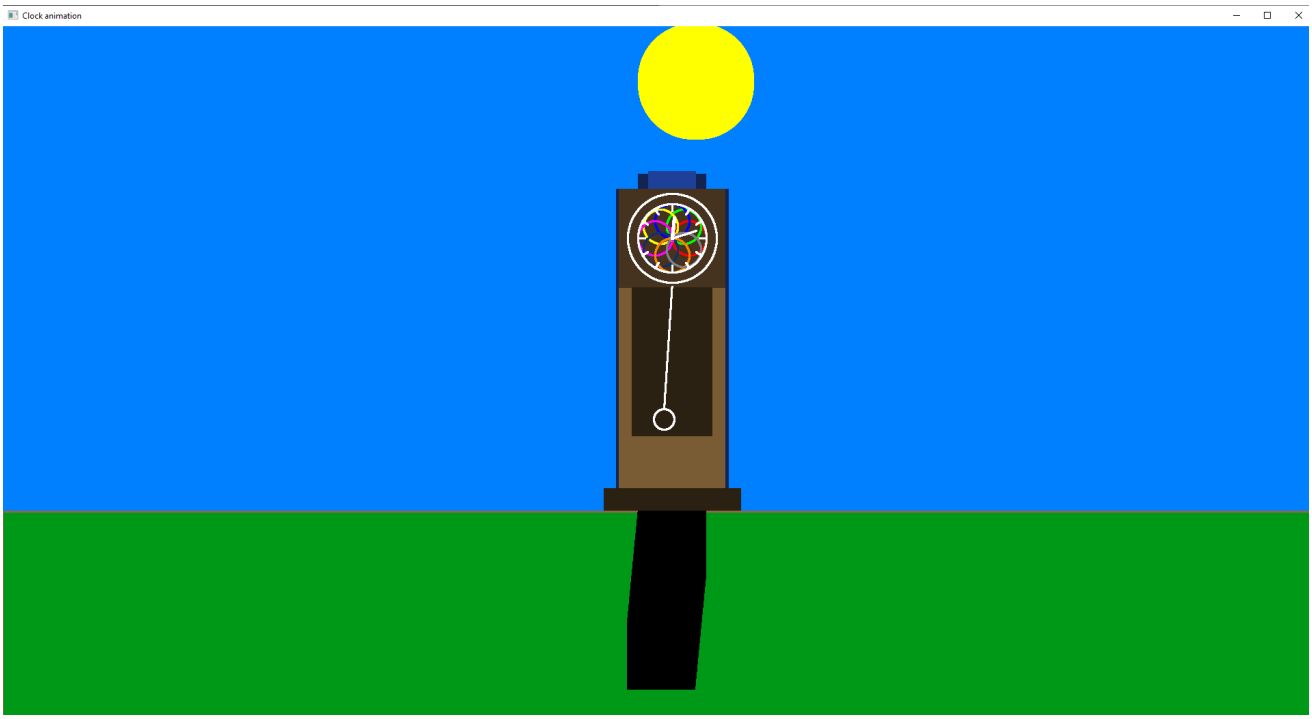
Project : Animated Clock with Day and Night Cycle

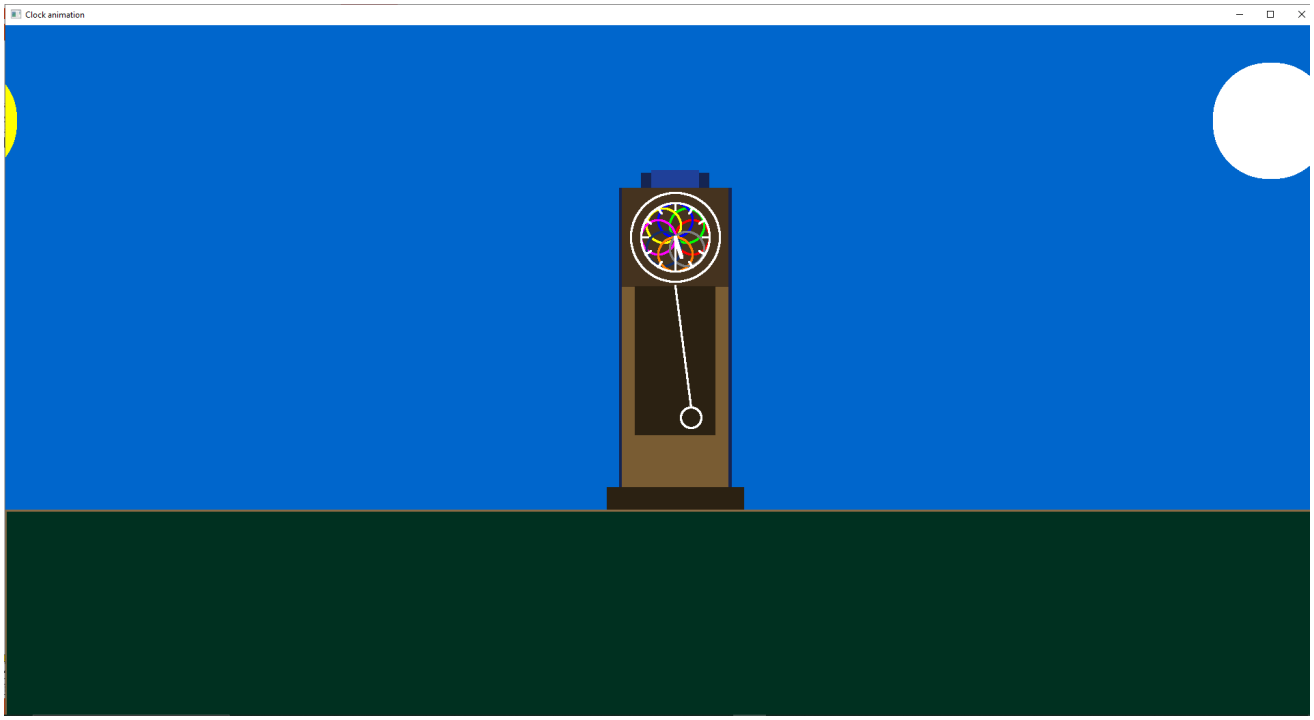
Explanation : The code is an implementation of an animated clock with a Day and Night cycle. The function draw_hour_hand() is used to draw the hour hand of the clock with a given center, minute, and radius. It also has logic to transition between two different colors, c1 and c2, and to change the background color of the clock, which includes the sun, moon and sky. The function draw_minute_hand() is used to draw the minute hand of the clock with a given center, second, and radius.

The program initializes the clock hands' parameters, including the hour, minute, and second. The milliseconds variable is used to keep track of time in milliseconds. The theta variable is used to determine the pendulum's angle, which swings back and forth underneath the clock.

The program then enters an infinite loop where it first clears the screen, and then calls the draw_hour_hand() and draw_minute_hand() functions to draw the hands of the clock. It also calls the draw_circle() function to draw a circle representing the pendulum's ball. Finally, it increments the time by a small amount of milliseconds and updates the pendulum's angle accordingly. The program then repeats this process until the 24 hour time that the user inputs has been reached, resulting in an animated clock with a swinging pendulum for a set amount of time.




Clock animation


Clock animation

Code :

```
import math


from OpenGL.GL import *
from OpenGL.GLUT import *

c1, c2 = 1, 0
bc1, bc2 = 0.1, 1
bc3, bc4 = 0, 0.5
day = False
x1,x2=0,0


def draw_point(x, y, point_size=5):
    glPointSize(point_size)
    glBegin(GL_POINTS)
    glVertex2f(x, y)
    glEnd()


def draw_line(x1, y1, x2, y2, point_size=3):
    pixels = mid_point_line_with_8way_symmetry(x2, y2, x1, y1)
    for pixel in pixels:
        draw_point(pixel[0], pixel[1], point_size)


def get_zone(x1, y1, x2, y2):
    dx = x2 - x1
    dy = y2 - y1
    if abs(dx) > abs(dy):
        if dx >= 0:
            return 0 if dy >= 0 else 7
        else:
            return 3 if dy >= 0 else 4
    else:
        if dx >= 0:
            return 1 if dy >= 0 else 6
        else:
            return 2 if dy >= 0 else 5


def to_zone_zero(x, y, zone):
    if zone == 0:
        return x, y
    elif zone == 1:
        return y, x
    elif zone == 2:
        return y, -x
    elif zone == 3:
        return -x, y
    elif zone == 4:
        return -x, -y
    elif zone == 5:
        return -y, -x
    elif zone == 6:
        return -y, x
    elif zone == 7:
        return x, -y


def to_original_zone(x, y, zone):
    if zone == 0:
        return x, y
    elif zone == 1:
        return y, x
    elif zone == 2:
        return -y, x
    elif zone == 3:
        return -x, y
    elif zone == 4:
        return -x, -y
    elif zone == 5:
        return -y, -x
    elif zone == 6:
        return y, -x
    elif zone == 7:
        return x, -y
```

```
def mid_point_line_with_8way_symmetry(x1, y1, x2, y2):
    zone = get_zone(x1, y1, x2, y2)

    x1_zone0, y1_zone0 = to_zone_zero(x1, y1, zone)
    x2_zone0, y2_zone0 = to_zone_zero(x2, y2, zone)

    delta_y = y2_zone0 - y1_zone0
    delta_x = x2_zone0 - x1_zone0
    decision_var = 2 * delta_y - delta_x
    delta_E = 2 * delta_y
    delta_NE = 2 * (delta_y - delta_x)

    x = x1_zone0
    y = y1_zone0

    original_x, original_y = to_original_zone(x, y, zone)

    pixels = []

    while x <= x2_zone0:
        pixels.append((original_x, original_y))

        if decision_var < 0:
            x += 1
            decision_var += delta_E
        else:
            x += 1
            y += 1
            decision_var += delta_NE

        original_x, original_y = to_original_zone(x, y, zone)

    return pixels


def iterate():
    glViewport(0, 0, 1920, 1080)
    glMatrixMode(GL_PROJECTION)
    glLoadIdentity()
    glOrtho(0.0, 1920, 0.0, 1080, 0.0, 1.0)
    glMatrixMode(GL_MODELVIEW)
    glLoadIdentity()


def draw_circle(x_centre, y_centre, radius, point_size=3):
    # Initialize the starting point on the circle
    x, y = 0, radius
    d = 1 - radius  # Calculate initial decision parameter
    while x <= y:
        # Draw points in each octant
        draw_point(x_centre + x, y_centre + y, point_size)
        draw_point(x_centre + y, y_centre + x, point_size)
        draw_point(x_centre + y, y_centre - x, point_size)
        draw_point(x_centre + x, y_centre - y, point_size)
        draw_point(x_centre - x, y_centre - y, point_size)
        draw_point(x_centre - y, y_centre - x, point_size)
        draw_point(x_centre - y, y_centre + x, point_size)
        draw_point(x_centre - x, y_centre + y, point_size)
        # Update x and y based on decision parameter
        if d <= 0:
            d = d + 2 * x + 3
        else:
            d = d + 2 * (x - y) + 5
            y = y - 1
        x = x + 1
```

```
def draw_hour_hand(x, y, minute, radius):
    angle = math.pi * minute / 30.0
    length = radius * 0.6

    # ------------------------ Sun-moon and background color transition logic--------------------
    global c1, c2, milliseconds, bc1, bc2, bc3, bc4, day, x1, x2
    x1 = x + length * math.sin(angle) + angle * 305
    y1 = y + length * math.cos(angle) + 200
    if milliseconds % 3600 == 0:
        c1, c2 = c2, c1
    if milliseconds == 0 or milliseconds == 1800 or (milliseconds - 1800) % 3600 == 0:
        day = not day
        bc1, bc2 = bc2, bc1
        bc3, bc4 = bc4, bc3
        glClearColor(0, bc3, bc1, 1)
        glClear(GL_COLOR_BUFFER_BIT)

    if(x1<1620 and day==True):
        alu=1
        # glClearColor(0, 0.1, 0.3, 1)
        # glClear(GL_COLOR_BUFFER_BIT)
    elif(x1<1720 and day==True):
        glClearColor(0, 0.4, 0.8, 1)
        glClear(GL_COLOR_BUFFER_BIT)
    elif(x1<1870 and day==True):
        glClearColor(0, 0.3, 0.6, 1)
        glClear(GL_COLOR_BUFFER_BIT)
    elif(x1<1920 and day==True):
        glClearColor(0, 0.2, 0.4, 1)
        glClear(GL_COLOR_BUFFER_BIT)

    if(x1<1620 and day==False):
        alu=1
    elif(x1<1720 and day==False):
        glClearColor(0, 0.2, 0.4, 1)
        glClear(GL_COLOR_BUFFER_BIT)
    elif(x1<1820 and day==False):
        glClearColor(0, 0.3, 0.6, 1)
        glClear(GL_COLOR_BUFFER_BIT)
    elif(x1<1920 and day==False):
        glClearColor(0, 0.4, 0.8, 1)
        glClear(GL_COLOR_BUFFER_BIT)

    draw_everything()

    glColor3f(1, 1, c1) # Circle 1 ----
    draw_circle(x1, y1, 80, 10)
    draw_circle(x1, y1, 50, 50)
    draw_circle(x1, y1, 20, 30)
    glColor3f(1, 1, c2) # Circle 2 ----
    draw_circle(x1 - 1920, y1, 80, 10)
    draw_circle(x1 - 1920, y1, 50, 50)
    draw_circle(x1 - 1920, y1, 20, 30)
    glColor3f(1, 1, 1)

    draw_line(x, y, x + length * math.sin(angle), y + length * math.cos(angle), 5)


def draw_minute_hand(x, y, second, radius):
    angle = 2 * math.pi * second / 60.0
    length = radius * 0.75
    draw_line(x, y, x + length * math.sin(angle), y + length * math.cos(angle), 3)


# Clock hand parameters
hour = 0
minute = 0
second = 0

milliseconds = 0

# Pendulum parameters
length = 250  # Length of the pendulum in meters
theta_max = 10  # Maximum angle in degrees
theta_min = -10  # Minimum angle in degrees
theta = 0  # Initial angle in degrees
theta_speed = (theta_max / 60) * 2  # Angular speed in degrees per frame
```

```
def draw_clock(x, y, radius, duration):
    # draw hour, minute and second hands
    draw_hour_hand(x, y, minute, radius)
    draw_minute_hand(x, y, second, radius)
    # draw_hour_hand(x, y, hour, minute, radius)

    # Draw clock face
    draw_point(x, y)
    draw_circle(x, y, radius)
    draw_circle(x, y, radius + 15)
    for i in range(12):
        angle = i * (2 * math.pi / 12)
        x1 = x + (radius - 10) * math.cos(angle)
        y1 = y + (radius - 10) * math.sin(angle)
        x2 = x + (radius) * math.cos(angle)
        y2 = y + (radius) * math.sin(angle)
        draw_line(x1, y1, x2, y2)

    # Draw the pendulum
    angle = math.pi * theta / 225
    new_pendulum_x = x + length * math.sin(angle)
    new_pendulum_y = y - length * math.cos(angle)
    draw_line(x, y - 70, new_pendulum_x, new_pendulum_y, 3)
    draw_circle(
        new_pendulum_x,
        new_pendulum_y - 15,
        15,
        3,
    )

    # minute_num = duration
    # print(minute_num)


def calculate_minutes_from_noon(time_24h):
    hours = time_24h // 100
    # minutes = time_24h % 100
    total_minutes = (hours - 12) * 60 # + minutes
    return total_minutes


def animate(value):
    global second
    global minute

    global hour
    global theta
    global theta_speed
    global milliseconds

    milliseconds = milliseconds + 1

    theta += theta_speed
    if theta > theta_max or theta < theta_min:
        theta_speed *= -1

    second = second + 1
    if second == 60:
        second = 0
        minute = minute + 1
        if minute == 60:
            minute = 0
            hour = hour + 1
    glutPostRedisplay()
    # if minute == duration-5:
    #     glutTimerFunc(0, None, 0)
    # else:
    #     glutTimerFunc(10, animate, 0)
    glutTimerFunc(10, animate, 0)
```

```
def draw_everything():
    glColor3f(0.08, 0.14, 0.32)  # color for point
    # ---------------------------------------- Standing Clock Layout

    draw_line(900, 820, 1060, 820, 3)
    draw_line(900, 820, 900, 680, 5)
    draw_line(1060, 820, 1060, 680, 5)
    draw_line(900, 680, 1060, 680, 3)

    # Roof
    # draw_line(900, 820, 1060, 820, 5)
    # draw_line(900, 820, 980, 900, 5)
    # draw_line(1060, 820, 980, 900, 5)

    draw_line(900, 680, 900, 380, 5)
    draw_line(1060, 680, 1060, 380, 5)
    draw_line(900, 680, 1060, 680, 5)
    draw_line(900, 380, 1060, 380, 5)

    draw_line(900, 380, 880, 350, 3)
    draw_line(1060, 380, 1080, 350, 3)
    draw_line(880, 350, 1080, 350, 3)
    draw_line(900, 380, 1060, 380, 3)

    # ---------------------------------------- Standing Clock Color Fill

    # Roof  box 1
    glColor3f(0.08, 0.14, 0.32)
    draw_line(980, 745, 980, 795, 100)
    glColor3f(1, 1, 1)
    # Roof  box 2
    glColor3f(0.12, 0.25, 0.6)
    draw_line(980, 810, 980, 814, 70)
    glColor3f(1, 1, 1)

    # 1st box
    glColor3f(0.27, 0.2, 0.12)
    draw_line(980, 745, 980, 710, 156)
    glColor3f(1, 1, 1)
    # 2nd box
    glColor3f(0.475, 0.36, 0.2)
    draw_line(980, 460, 980, 600, 156)
    glColor3f(1, 1, 1)
    # 3rd box
    glColor3f(0.17, 0.13, 0.07)
    draw_line(980, 520, 980, 620, 118)
    glColor3f(1, 1, 1)
    # 4th box
    glColor3f(0.17, 0.13, 0.07)
    draw_line(900, 365, 1061, 365, 40)
    glColor3f(1, 1, 1)
    # Ground
    if (day and not (x1>1750 and x1<2000)):
        glColor3f(0, 0.6, 0.09)
    elif(day):
        glColor3f(0, 0.35, 0.09)
    else:
        glColor3f(0, 0.188, 0.125)
    draw_line(0, 175, 1920, 175, 350)
    glColor3f(1, 1, 1)
    # ---------------------------------------- Clock Face Design
    x = 980
    y = 750
    radius = 50
    draw_circle(x, y, radius)

    colors = [  # list of colors randomly generated for each of the 8 circles
        (1.0, 0.0, 0.0),
        (0.0, 1.0, 0.0),
        (0.0, 0.0, 1.0),
        (1.0, 1.0, 0.0),
        (1.0, 0.0, 1.0),
        (0.08, 0.18, 0.42),
        (1.0, 0.5, 0.0),
        (0.5, 0.5, 0.5),
    ]

    for i, color in enumerate(
            colors
    ):  # There are 8 colors so i will also increment till 8
        glColor3f(*color)
        angle = i * (360 / 8)
        offset_x = int(radius * 0.5 * math.cos(math.radians(angle)))
        offset_y = int(radius * 0.5 * math.sin(math.radians(angle)))
        draw_circle(x + offset_x, y + offset_y, radius / 2)

# draw_circle(200,900,80,10)
    # draw_circle(200,900,50,50)
    # draw_circle(200,900,20,30)

    # ---------------------------------------- Border for Ground
    glColor3f(0.54, 0.42, 0.27)
    draw_line(0, 350, 1920, 350, 3)
    draw_line(0, 0, 0, 350, 3)
    draw_line(1920, 0, 1920, 350, 3)
    draw_line(0, 0, 1920, 0, 3)
    # draw_circle(200,900,80,10)
    # draw_circle(200,900,50,50)
    # draw_circle(200,900,20,30)
    # ---------------------------------------- Shadow
    if day:
        angle1 = math.pi * minute / 30.0
        length1 = radius * 0.6
        glColor3f(0, 0, 0)
        draw_line(980, 303, 980 - (length1 * math.sin(angle1)) * 5, (170 - length1 * math.cos(angle1)), 100)
        glColor3f(1, 1, 1)



def show_screen():
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
    glLoadIdentity()
    iterate()

    # ---------------------------------------- Clock animation
    glColor3f(1, 1, 1)
    global duration
    time = 2100
    duration = calculate_minutes_from_noon(time)/10
    draw_clock(980, 750, 50, duration)

    glutSwapBuffers()


glutInit()
glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH)
glutInitWindowSize(1920, 1080)
glutCreateWindow(b"Clock animation")
glutDisplayFunc(show_screen)
glutTimerFunc(10, animate, 0)
glutMainLoop()
```