Sprawozdanie
Grafika komputerowa i komunikacja człowiek-komputer
Laboratorium 6

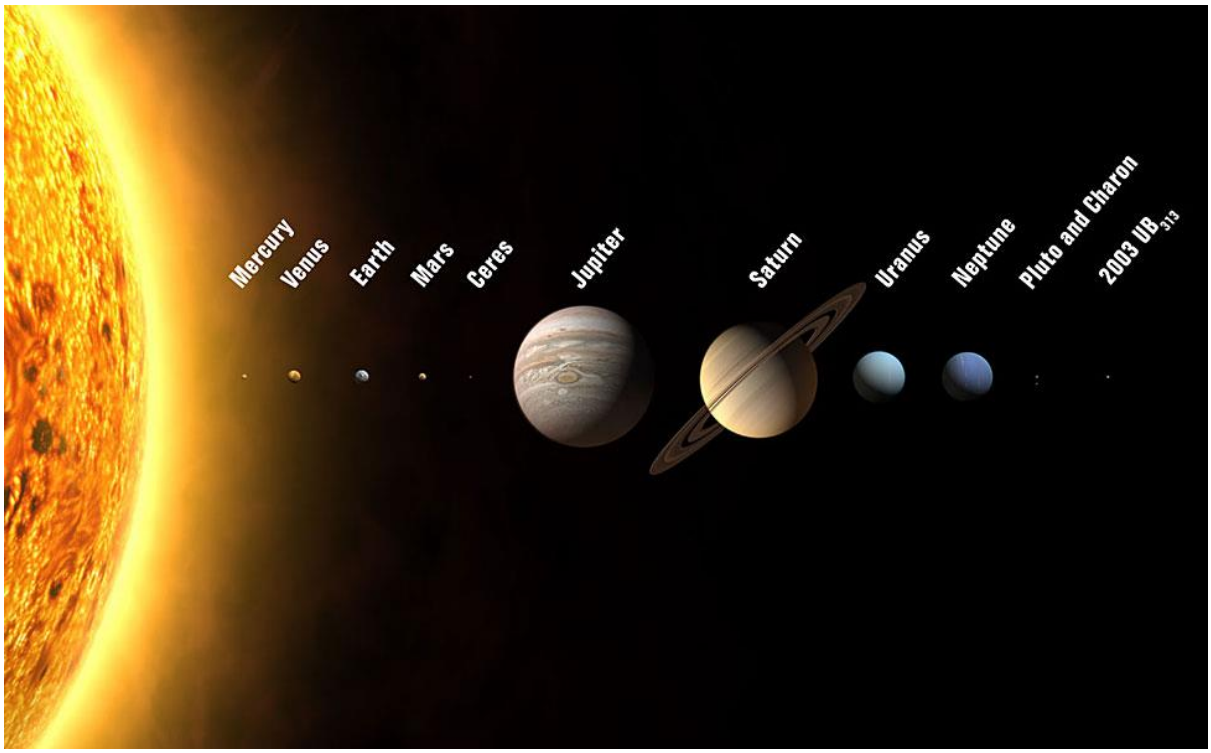
Katsiaryna Kolyshko 276708

Dr. Inż. Jan Nikodem

Wstęp teoretyczny

Zadanie laboratoryjne polega na stworzeniu układu słonecznego. Najpierw musimy przypomnieć jak on wygląda.

Na obrazku niżej jest przedstawiony układ, na który opieramy się, aby stworzyć i napisać program.



Z oficjalnej strony NASA poznaliśmy różnice w rozmiarach planet i Słońca w porównaniu do Ziemi. Ale trochę zmieniliśmy rozmiary, aby mniejsze planety też było widać. Niżej jest przestawiona informacja o rozmiarach planet w porównaniu do Ziemi:

The Solar System: Planet Sizes

- Mercury – 1,516mi (2,440km) radius; about 1/3 the size of Earth
- Venus – 3,760mi (6,052km) radius; only slightly smaller than Earth
- Earth – 3,959mi (6,371km) radius
- Mars – 2,106mi (3,390km) radius; about half the size of Earth
- Jupiter – 43,441mi (69,911km) radius; 11x Earth's size
- Saturn – 36,184mi (58,232km) radius; 9x larger than Earth
- Uranus – 15,759mi (25,362km) radius; 4x Earth's size
- Neptune – 15,299mi (24,622km) radius; only slightly smaller than Uranus

```python
# Planetary data: name, radius scale, orbit radius, color, orbital speed multiplier, texture filename
planets = [
    ("Sun", a * 30, 0, (1.0, 0.8, 0.0), 0, "Sun.jpeg"),  # Bright yellow
    ("Mercury", a, 2, (0.5, 0.5, 0.5), 4.8, "Mercury.jpeg"),  # Gray
    ("Venus", a * 3, 3, (0.8, 0.6, 0.2), 3.5, "Venus.jpeg"),  # Orangish
    ("Earth", a * 3, 4, (0.0, 0.4, 0.8), 3, "Earth.jpeg"),  # Blue
    ("Mars", a * 2, 5, (0.8, 0.2, 0.0), 2.4, "Mars.jpeg"),  # Red
    ("Jupiter", a * 15, 7, (0.7, 0.5, 0.3), 1.3, "Jupiter.jpeg"),  # Brownish
    ("Saturn", a * 14, 9, (0.8, 0.7, 0.4), 1, "Saturn.jpeg"),  # Pale yellow
    ("Uranus", a * 7, 11, (0.4, 0.8, 0.9), 0.7, "Uranus.jpeg"),  # Light blue
    ("Neptune", a * 7, 13, (0.0, 0.4, 0.8), 0.5, "Neptune.jpeg")  # Deep blue
]
```

Oprócz tego, musimy pamiętać o tym, że za pomocą 3-ego prawa Keplera, planety obracają się po orbitach okrąg Słońca w specyficzny sposób. Trzecie prawo Keplera (ang. Kepler's third law) mówi, że kwadrat okresu obiegu planety jest proporcjonalny do sześcianu wielkiej półosi jej orbity.

Też musimy pokazać orbitę okrąg których obracają się planety.

Opis parametrów programu:

```python
# Window dimensions
width, height = 1280, 720

# Base radius
a = 1.0

active_camera = 0
camera_rotation = 0.0
rotating = False
last_mouse_x = 0

theta = 180.0
phi = 0.0
pix2angle = 1.0
piy2angle = 1.0
R = 20.0
scale = 1.0
left_mouse_button_pressed = 0
right_mouse_button_pressed = 0
mouse_x_pos_old = 0
mouse_y_pos_old = 0
delta_x = 0
delta_y = 0
upY = 1.0
```

Opis programu

Funkcja key_callback:

```python
def key_callback(window, key, scancode, action, mods):  2 usages  new *
    """Handle keyboard input for camera switching and resetting."""
    global active_camera

    if action == glfw.PRESS:
        if key >= glfw.KEY_0 and key <= glfw.KEY_9:
            # Switch camera to a specific planet based on the numeric key pressed
            new_camera = key - glfw.KEY_0
            if new_camera < len(planets):
                active_camera = new_camera
        elif key == glfw.KEY_D:
            # Reset camera to default position (0th index in this case)
            active_camera = 0
```

Ta funkcja nasłuchuje danych wprowadzanych z klawiatury i obsługuje dwie określone akcje:

1. Po naciśnięciu klawisza numerycznego (0-9) przełącza kamerę, aby wyświetlić inną planetę na podstawie naciśniętego numeru. Na przykład naciśnięcie „1” przełączy kamerę na pierwszą planetę, naciśnięcie „2” na drugą planetę itd.

2. Po naciśnięciu klawisza „D” resetuje kamerę do jej domyślnej pozycji, która jest ustawiona na wyświetlanie pierwszej planety.

Funkcja get_camera_position:

```python
def get_camera_position(time):  1 usage  new *
    """Get camera position as if standing on the orbit of the active planet."""
    planet = planets[active_camera]
    name, radius_scale, orbit_radius, _, speed_multiplier, _ = planet

    if name == "Sun":
        # For sun, maintain existing rotation around center
        camera_distance = radius_scale * 0.15
        camera_x = camera_distance * math.cos(camera_rotation)
        camera_z = camera_distance * math.sin(camera_rotation)
        return [camera_x, camera_distance, camera_z], [0, 0, 0]
    else:
        # Calculate position on planet's orbit
        angle = time * (0.02 * speed_multiplier)
        view_position = [
            orbit_radius * math.cos(angle),
            2,  # Slight elevation
            orbit_radius * math.sin(angle)
        ]

        # Calculate look direction based on mouse input
        look_x = view_position[0] + R * math.cos(2 * math.pi * theta / 360) * math.cos(2 * math.pi * phi / 360)
        look_y = view_position[1] + R * math.sin(2 * math.pi * phi / 360)
        look_z = view_position[2] + R * math.sin(2 * math.pi * theta / 360) * math.cos(2 * math.pi * phi / 360)

        look_at = [look_x, look_y, look_z]
        return view_position, look_at
```

Ta funkcja oblicza pozycję kamery w przestrzeni, tak jakbyś stał na orbicie planety.

1. Jeśli aktywną planetą jest „Słońce”, kamera pozostaje w stałej odległości i obraca się wokół Słońca po orbicie kołowej.

2. W przypadku innych planet pozycja kamery jest obliczana na podstawie orbity planety wokół Słońca, a jej pozycja zmienia się w czasie (na podstawie parametru `time`). Kamera jest nieznacznie podniesiona nad orbitą, a jej pozycja na orbicie jest obliczana za pomocą trygonometrii.

3. Kamera oblicza również pozycję „patrzenia” (w którą jest zwrócona) na podstawie niektórych zmiennych wejściowych, które dostosowują kierunek widzenia kamery. Dzięki temu kamera jest zawsze prawidłowo zorientowana w przestrzeni, patrząc w określony punkt.

Ta funkcja obsługuje wprowadzanie danych za pomocą przycisków myszy i śledzi stan lewego i prawego przycisku myszy.

Funkcja mouse_button_callback:

```
83  def mouse_button_callback(window, button, action, mods):  1 usage  new *
84      global left_mouse_button_pressed, right_mouse_button_pressed, mouse_x_pos_old, mouse_y_pos_old
85
86      if button == glfw.MOUSE_BUTTON_LEFT:
87          if action == glfw.PRESS:
88              left_mouse_button_pressed = 1
89              x_pos, y_pos = glfw.get_cursor_pos(window)
90              mouse_x_pos_old = x_pos
91              mouse_y_pos_old = y_pos
92          elif action == glfw.RELEASE:
93              left_mouse_button_pressed = 0
94
95      if button == glfw.MOUSE_BUTTON_RIGHT:
96          if action == glfw.PRESS:
97              right_mouse_button_pressed = 1
98              x_pos, y_pos = glfw.get_cursor_pos(window)
99              mouse_x_pos_old = x_pos
100         elif action == glfw.RELEASE:
101             right_mouse_button_pressed = 0
```

1. Lewy przycisk myszy:

- Po naciśnięciu lewego przycisku ustawia flagę wskazującą, że przycisk jest naciśnięty i zapisuje bieżącą pozycję myszy (współrzędne x i y).

- Po zwolnieniu lewego przycisku ustawia flagę wskazującą, że przycisk nie jest już naciśnięty.

2. Prawy przycisk myszy:

- Podobnie jak w przypadku lewego przycisku, po naciśnięciu prawego przycisku ustawia flagę i zapisuje bieżącą pozycję x (y nie jest śledzone).

- Po zwolnieniu prawego przycisku czyści flagę.

Ta funkcja pomaga śledzić, czy użytkownik przytrzymuje przycisk myszy i śledzi pozycję myszy, gdy przycisk jest naciśnięty.

Funkcja cursor_position_callback:

```python
def cursor_position_callback(window, x_pos, y_pos):  1 usage  new *
    global delta_x, delta_y, mouse_x_pos_old, mouse_y_pos_old, theta, phi, R

    delta_x = x_pos - mouse_x_pos_old
    delta_y = y_pos - mouse_y_pos_old
    mouse_x_pos_old = x_pos
    mouse_y_pos_old = y_pos

    if left_mouse_button_pressed:
        theta += delta_x * pix2angle
        phi += delta_y * piy2angle

    if right_mouse_button_pressed:
        if delta_x > 0 and R < 100:  # Increase upper limit for zoom-in (can zoom closer to the planets)
            R += 0.5
        elif delta_x < 0 and R > 1:  # Decrease lower limit for zoom-out (can zoom further away)
            R -= 0.5
```

Ta funkcja śledzi ruch kursora myszy i dostosowuje pewne parametry na podstawie ruchu myszy, szczególnie gdy wciśnięty jest lewy lub prawy przycisk myszy.

1. Oblicza zmianę położenia myszy (delta_x i delta_y) poprzez porównanie bieżącej pozycji z poprzednią, a następnie aktualizuje zapisaną pozycję myszy.

2. Po naciśnięciu lewego przycisku myszy:

- Dostosowuje dwie zmienne (theta i phi), aby obrócić widok. Theta kontroluje obrót poziomy (lewo/prawo), a phi kontroluje obrót pionowy (góra/dół).

3. Po naciśnięciu prawego przycisku myszy:

- Zmienia poziom powiększenia (R), umożliwiając użytkownikowi powiększanie i pomniejszanie. Przesunięcie myszy w prawo powiększa (zwiększa R), a przesunięcie myszy w lewo pomniejsza (zmniejsza R), z ograniczeniami, aby zapobiec zbytniemu powiększaniu lub pomniejszaniu.

Ta funkcja pomaga kontrolować zarówno obrót kamery, jak i powiększenie na podstawie ruchów myszy.

Funkcja get_camera_position:

```python
def get_camera_position(time):  1 usage  new *
    """Get camera position based on spherical coordinates."""
    planet = planets[active_camera]
    name, radius_scale, orbit_radius, _, speed_multiplier, _ = planet

    if name == "Sun":
        planet_pos = [0, 0, 0]
    else:
        angle = time * (0.02 * speed_multiplier)
        planet_pos = [
            orbit_radius * math.cos(angle),
            0,
            orbit_radius * math.sin(angle)
        ]

    xeye = R * math.cos(2 * math.pi * theta / 360) * math.cos(2 * math.pi * phi / 360)
    yeye = R * math.sin(2 * math.pi * phi / 360)
    zeye = R * math.sin(2 * math.pi * theta / 360) * math.cos(2 * math.pi * phi / 360)

    camera_pos = [planet_pos[0] + xeye, planet_pos[1] + yeye, planet_pos[2] + zeye]

    return camera_pos, planet_pos
```

Ta funkcja oblicza położenie kamery i położenie planety, wokół której krąży, na podstawie współrzędnych sferycznych.

1. Położenie planety:

- Jeśli aktywną planetą jest „Słońce”, położenie planety jest ustawione na środek układu słonecznego ([0, 0, 0]).

- W przypadku innych planet położenie jest obliczane na podstawie orbity planety wokół Słońca. Położenie planety zmienia się w czasie, gdy krąży wokół Słońca, przy użyciu kąta, który zależy od promienia orbity planety i mnożnika prędkości.

2. Położenie kamery:

- Położenie kamery jest obliczane przy użyciu współrzędnych sferycznych na podstawie bieżącego poziomu powiększenia i kątów obrotu.

- Położenie kamery (`xeye`, `yeye`, `zeye`) jest przesunięte względem położenia planety, co pozwala kamerze na orbitowanie wokół planety.

- Ostateczne położenie kamery jest sumą położenia planety i względnego położenia kamery, tworząc orbitę kamery wokół planety.

3. Wartości zwracane:

- Funkcja zwraca zarówno położenie kamery, jak i położenie planety, umożliwiając innym częściom programu poznanie lokalizacji obu tych obiektów.

Funkcja load_texture:

```python
def load_texture(filename):  1 usage  new *
    """Load an image and convert it to a texture."""
    try:
        # Open the image
        image = Image.open(filename)

        # Ensure the file is valid
        if not image:
            raise ValueError("Image file could not be opened.")

        # Convert to RGBA
        image = image.convert("RGBA")

        # Get image data
        img_data = image.tobytes()

        # Create a texture
        texture = glGenTextures(1)
        glBindTexture(GL_TEXTURE_2D, texture)

        # Set texture parameters
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR)
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR)

        # Upload the texture
        glTexImage2D(GL_TEXTURE_2D, level: 0, GL_RGBA,
                     image.width, image.height,
                     border: 0, GL_RGBA, GL_UNSIGNED_BYTE, img_data)

        return texture
    except FileNotFoundError:
        print(f"File not found: {filename}")
```

Ta funkcja ładuje obraz, konwertuje go na teksturę, której można używać w OpenGL, i zwraca teksturę.

1. Otwórz obraz:

- Funkcja próbuje otworzyć plik obrazu za pomocą metody Image.open() z biblioteki Python Imaging Library (PIL). Jeśli obrazu nie można otworzyć, zgłasza błąd.

2. Konwertuj obraz:

- Obraz jest konwertowany do formatu kolorów RGBA, zapewniając, że ma kanały czerwony, zielony, niebieski i alfa (przezroczystość).

3. Wyodrębnij dane obrazu:

- Dane obrazu są konwertowane na surowe bajty za pomocą metody tobytes(), która jest wymagana dla tekstur OpenGL.

4. Utwórz i powiąż teksturę:

- Tekstura jest tworzona za pomocą glGenTextures(1) i jest powiązana z celem tekstury 2D (GL_TEXTURE_2D) za pomocą glBindTexture().

5. Ustaw parametry tekstury:

- Parametry filtrowania tekstury są ustawione na filtrowanie liniowe zarówno dla minifikacji, jak i powiększenia, zapewniając płynne skalowanie tekstury.

6. Prześlij teksturę:

- Dane tekstury są przesyłane do OpenGL za pomocą glTexImage2D(), który przesyła dane obrazu RGBA do tekstury.

7. Zwróć teksturę:

- Jeśli wszystko się powiedzie, obiekt tekstury jest zwracany. Jeśli wystąpi błąd (np. plik nie został znaleziony lub jakikolwiek inny problem), zostanie wyświetlony komunikat o błędzie, a funkcja zwróci None.

Ta funkcja umożliwia ładowanie i używanie plików obrazów jako tekstur w OpenGL.

Funkcja init:

```
197    def init():  1 usage  new *
198        """Initialize OpenGL settings with enhanced lighting."""
199        glClearColor( red: 0.0,  green: 0.0,  blue: 0.1,  alpha: 1.0)
200        glEnable(GL_DEPTH_TEST)
201        glEnable(GL_TEXTURE_2D)
202        glEnable(GL_LIGHTING)
203        glEnable(GL_LIGHT0)
204        glEnable(GL_COLOR_MATERIAL)
205        glColorMaterial(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE)
206        glShadeModel(GL_SMOOTH)
207
208        # Enhanced lighting properties
209        ambient_light = [0.0, 0.0, 0.0, 1.0]  # Minimal ambient light
210        diffuse_light = [1.0, 1.0, 1.0, 1.0]  # Bright diffuse light
211        specular_light = [1.0, 1.0, 1.0, 1.0]  # White specular highlights
212
213        glLightfv(GL_LIGHT0, GL_AMBIENT, ambient_light)
214        glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuse_light)
215        glLightfv(GL_LIGHT0, GL_SPECULAR, specular_light)
216
217        # Load textures
218        global planet_textures
219        planet_textures = [load_texture(tex_file) for _, _, _, _, _, tex_file in planets]
```

Funkcja init() ustawia środowisko OpenGL, przygotowując scenę do renderowania:

1. Ustawia kolor tła na ciemnoniebieski.
2. Włącza test głębokości, obsługę tekstur i oświetlenie.
3. Ustala, że oświetlenie będzie wygładzone i stosowane do wszystkich powierzchni.
4. Ustawia właściwości oświetlenia, takie jak minimalne światło otoczenia, jasne światło rozproszone i białe refleksy.
5. Ładuje tekstury dla planet, przygotowując je do wyświetlenia w scenie.

Funkcja draw_sun:

```python
def draw_sun(radius, texture):  1 usage  new *
    """Draw sun with emissive properties."""
    glDisable(GL_LIGHTING)  # Sun generates light, doesn't receive it

    # Make sun glow
    emission = [0.5, 0.4, 0.0, 1.0]
    glMaterialfv(GL_FRONT, GL_EMISSION, emission)

    if texture:
        glBindTexture(GL_TEXTURE_2D, texture)

    quadric = gluNewQuadric()
    gluQuadricTexture(quadric, GL_TRUE)
    gluSphere(quadric, radius, 50, 50)
    gluDeleteQuadric(quadric)

    # Reset emission
    glMaterialfv(GL_FRONT, GL_EMISSION, params: [0.0, 0.0, 0.0, 1.0])
    glEnable(GL_LIGHTING)
```

Funkcja draw_sun rysuje Słońce w scenie 3D, nadając mu właściwości emisyjne:

1. Wyłączenie oświetlenia:

- Wyłącza oświetlenie (glDisable(GL_LIGHTING)), ponieważ Słońce generuje światło, a nie je odbiera.

2. Ustawienie emisji:

- Ustala kolor emisji (światła generowanego przez Słońce) na pomarańczowo-żółty.

3. Zastosowanie tekstury:

- Jeśli przekazano teksturę, to jest ona przypisywana do Słońca (glBindTexture(GL_TEXTURE_2D, texture)).

4. Rysowanie Słońca:

- Tworzy sferę reprezentującą Słońce przy użyciu funkcji gluSphere(), z określonym promieniem i detalami powierzchni (50 segmentów w obu kierunkach).

5. Czyszczenie:

- Po narysowaniu Słońca resetuje właściwości emisji do domyślnych ([0.0, 0.0, 0.0, 1.0]).

- Ponownie włącza oświetlenie (glEnable(GL_LIGHTING)), aby przywrócić normalne działanie oświetlenia w scenie.

W skrócie, funkcja ta rysuje Słońce, sprawiając, że świeci na scenie dzięki emisji światła.

Funkcja draw_orbital_path:

```python
def draw_orbital_path(radius):  1 usage  new *
    """Draw a white circular orbit path with transparency."""
    glDisable(GL_LIGHTING)  # Disable lighting for the orbit lines
    glColor4f( red: 1.0,  green: 1.0,  blue: 1.0,  alpha: 0.03)  # White color with 30% opacity (alpha = 0.3)
    glLineWidth(1.0)  # Thin lines

    # Draw the orbit as a circle
    glBegin(GL_LINE_LOOP)
    for i in range(100):
        angle = 2 * math.pi * i / 100
        x = radius * math.cos(angle)
        z = radius * math.sin(angle)
        glVertex3f(x,  y: 0, z)
    glEnd()

    glEnable(GL_LIGHTING)
```

Funkcja draw_orbital_path() rysuje przezroczystą, białą linię okręgu, która reprezentuje orbitę planety.

1. Wyłącza oświetlenie (glDisable(GL_LIGHTING)), aby orbitę rysować niezależnie od źródeł światła.
2. Ustawia kolor linii na biały z 30% przezroczystością (glColor4f(1.0, 1.0, 1.0, 0.03)).
3. Ustawia szerokość linii na cienką (glLineWidth(1.0)).
4. Rysuje okrąg, korzystając z pętli i funkcji trygonometrycznych:
    - Dzieli okrąg na 100 punktów.
    - Oblicza współrzędne każdego punktu przy użyciu math.cos i math.sin.
    - Rysuje linię łączącą te punkty, używając glBegin(GL_LINE_LOOP) i glVertex3f().
5. Po zakończeniu rysowania włącza ponownie oświetlenie (glEnable(GL_LIGHTING)).

Funkcja tworzy wizualizację orbity planety jako delikatny, półprzezroczysty okrąg.

Funkcja draw_planet:

```python
def draw_planet(radius, texture, rotation_angle):  1 usage  new *
    glEnable(GL_LIGHTING)

    # Enable face culling for planets
    glEnable(GL_CULL_FACE)
    glCullFace(GL_BACK)
    glFrontFace(GL_CCW)

    ambient = [0.2, 0.2, 0.2, 1.0]
    diffuse = [0.8, 0.8, 0.8, 1.0]
    specular = [0.2, 0.2, 0.2, 1.0]
    shininess = 50.0

    glMaterialfv(GL_FRONT, GL_AMBIENT, ambient)
    glMaterialfv(GL_FRONT, GL_DIFFUSE, diffuse)
    glMaterialfv(GL_FRONT, GL_SPECULAR, specular)
    glMaterialf(GL_FRONT, GL_SHININESS, shininess)

    if texture:
        glBindTexture(GL_TEXTURE_2D, texture)
        glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP)
        glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP)
        glEnable(GL_TEXTURE_GEN_S)
        glEnable(GL_TEXTURE_GEN_T)

    # Apply rotation around the planet's axis (Y-axis)
    glRotatef(rotation_angle, x: 0, y: 1, z: 0)

    quadric = gluNewQuadric()
    gluQuadricTexture(quadric, GL_TRUE)
    gluQuadricOrientation(quadric, GLU_OUTSIDE)
    gluSphere(quadric, radius, 50, 50)
    gluDeleteQuadric(quadric)

    if texture:
        glDisable(GL_TEXTURE_GEN_S)
        glDisable(GL_TEXTURE_GEN_T)

    glDisable(GL_CULL_FACE)
```

Funkcja draw_planet() odpowiada za rysowanie planety z uwzględnieniem jej wyglądu, tekstury oraz obrotu wokół własnej osi.

1. Włącza oświetlenie i ustawienia dla usuwania tylnych ścian planety, dzięki czemu renderowana jest tylko widoczna część kuli. Określa, że widoczne ściany są definiowane przez przeciwny do ruchu wskazówek zegara układ wierzchołków.
2. Ustawia właściwości materiału planety, takie jak:
    - Ambient: światło otoczenia o niskiej intensywności.
    - Diffuse: światło rozproszone odpowiadające za oświetlenie powierzchni.
    - Specular: odbicie światła na powierzchni.

- Shininess: połyskliwość powierzchni, definiująca intensywność speculara.

3. Jeśli została podana tekstura, aktywuje teksturowanie sferyczne, które sprawia, że tekstura naturalnie układa się na kuli. Ustawia mapowanie tekstur w trybie sferycznym.
4. Stosuje obrót planety wokół osi Y, bazując na podanym kącie obrotu. Symuluje to naturalny obrót planety.
5. Tworzy obiekt geometryczny w postaci sfery o zadanym promieniu i odpowiedniej liczbie segmentów (dzięki funkcji OpenGL gluSphere). Jeśli tekstura jest podana, jest ona nakładana na powierzchnię planety.
6. Po narysowaniu planety dezaktywuje mapowanie tekstur oraz usuwanie tylnych ścian, aby przywrócić domyślne ustawienia.

Dzięki tym operacjom planeta jest realistycznie renderowana z uwzględnieniem oświetlenia, tekstury i ruchu obrotowego.

Funkcja draw_solar_system:

```
306    def draw_solar_system(time):  1 usage  new *
307        """Render solar system with dynamic lighting, planet rotation, and self-rotation."""
308        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
309        glLoadIdentity()
310
311        camera_pos, look_at = get_camera_position(time)
312        gluLookAt(camera_pos[0], camera_pos[1], camera_pos[2],
313                  look_at[0], look_at[1], look_at[2],
314                  0, 1, 0)
315
316        # Set light position at sun's center
317        light_position = [0.0, 0.0, 0.0, 1.0]
318        glLightfv(GL_LIGHT0, GL_POSITION, light_position)
319
320        # Draw orbital paths
321        for _, _, orbit_radius, _, _, _ in planets[1:]:
322            draw_orbital_path(orbit_radius)
323
324        # Planet rotation speeds (degrees per time unit)
325        # These values approximate real planet rotation periods relative to Earth
326        rotation_speeds = {
327            "Sun": 27,      # Sun rotates once every ~27 Earth days
328            "Mercury": 59,  # Mercury rotates once every ~59 Earth days
329            "Venus": -243,  # Venus has retrograde rotation (negative value)
330            "Earth": 1,     # Earth's rotation period is our reference (1 day)
331            "Mars": 1.03,   # Mars takes slightly longer than Earth
332            "Jupiter": 0.41,# Jupiter rotates about twice as fast as Earth
333            "Saturn": 0.45, # Saturn rotates about twice as fast as Earth
334            "Uranus": 0.72,# Uranus rotates about 17 hours
335            "Neptune": 0.67 # Neptune rotates about 16 hours
```

```
    # Draw planets
    for i, (name, radius_scale, orbit_radius, color, speed_multiplier, _) in enumerate(planets):
        glPushMatrix()

        if name == "Sun":
            glColor3f(*color)
            # Calculate sun's rotation
            rotation_angle = time * 50 * rotation_speeds[name]
            draw_sun(radius_scale * 0.05, planet_textures[i])
        else:
            # Calculate orbital position
            orbital_angle = time * (0.02 * speed_multiplier)
            x = orbit_radius * math.cos(orbital_angle)
            z = orbit_radius * math.sin(orbital_angle)

            # Move to orbital position
            glTranslatef(x, y: 0, z)

            # Calculate self-rotation angle
            rotation_angle = time * 50 * rotation_speeds[name]

            glColor3f(*color)
            draw_planet(radius_scale * 0.05, planet_textures[i], rotation_angle)

        glPopMatrix()
```

Funkcja draw_solar_system() renderuje dynamiczny model Układu Słonecznego z uwzględnieniem rotacji planet, ich ruchu orbitalnego oraz efektów świetlnych. Oto szczegółowy opis działania:

1. Przygotowanie sceny: Funkcja czyści bufory koloru i głębokości, aby przygotować ekran do nowej klatki za pomocą glClear. Resetuje macierz modelu-widoku do stanu początkowego za pomocą glLoadIdentity.

2. Pozycjonowanie kamery: Wyznacza bieżącą pozycję kamery oraz punkt, na który patrzy, korzystając z funkcji get_camera_position(time). Ustawia widok kamery za pomocą gluLookAt, co pozwala na dynamiczne zmiany perspektywy w zależności od czasu.

3. Ustawienie światła: Określa pozycję głównego źródła światła (GL_LIGHT0) w centrum Układu Słonecznego (pozycja Słońca) za pomocą glLightfv. Symuluje to Słońce jako główne źródło oświetlenia.

4. Rysowanie orbit: Iteruje przez wszystkie planety (oprócz Słońca) w liście planets[1:] i rysuje ich orbity w postaci cienkich okręgów, wywołując funkcję draw_orbital_path.

5. Prędkości rotacji: Określa prędkości rotacji dla każdej planety w stopniach na jednostkę czasu. Wartości te bazują na rzeczywistych okresach rotacji planet, uwzględniając m.in. retrogradacyjną rotację Wenus.

6. Renderowanie planet i Słońca: Iteruje przez wszystkie obiekty w liście planets, aby je narysować:

    - Słońce: Ustawia kolor Słońca, oblicza jego rotację na podstawie prędkości rotacji i czasu, a następnie wywołuje funkcję draw_sun, aby narysować sferę z odpowiednimi właściwościami emisyjnymi.

    - Planety: Oblicza aktualną pozycję planety na orbicie (x, z) w oparciu o jej promień orbity i prędkość. Przesuwa pozycję rysowania do obliczonego miejsca za pomocą glTranslatef. Następnie oblicza kąt rotacji planety wokół własnej osi, ustawia jej kolor

i wywołuje funkcję draw_planet, aby ją narysować z odpowiednim rozmiarem, teksturą i rotacją.

7. Resetowanie macierzy: Po narysowaniu każdej planety lub Słońca przywraca poprzedni stan transformacji za pomocą glPopMatrix, aby uniknąć wpływu na kolejne obiekty.

Efekt: Funkcja tworzy realistyczną symulację Układu Słonecznego. Planety poruszają się po swoich orbitach i obracają wokół własnych osi z uwzględnieniem rzeczywistych prędkości. Słońce działa jako centralne źródło światła, a orbity planet są rysowane dla lepszej przejrzystości. Dynamiczna kamera pozwala eksplorować całą scenę, co zwiększa wrażenie realizmu.

Funkcja main:

```
344  def main():  1 usage  new *
345      """Main function with added keyboard callback."""
346      if not glfw.init():
347          return
348
349      window = glfw.create_window(width, height,  title: "Solar System Simulation",  monitor: None,  share: None)
350      if not window:
351          glfw.terminate()
352          return
353
354      glfw.make_context_current(window)
355      glfw.set_framebuffer_size_callback(window, framebuffer_size_callback)
356      glfw.set_key_callback(window, key_callback)  # Add keyboard callback
357      glfw.set_mouse_button_callback(window, mouse_button_callback)
358      glfw.set_cursor_pos_callback(window, cursor_position_callback)
359      glfw.set_key_callback(window, key_callback)
360
361      init()
362      time = 0.0
363
364      while not glfw.window_should_close(window):
365          draw_solar_system(time)
366          glfw.swap_buffers(window)
367          glfw.poll_events()
368          time += 0.005
369
370      glfw.terminate()
```

Funkcja main() jest główną funkcją programu, która inicjalizuje środowisko i uruchamia symulację Układu Słonecznego.

1. Inicjalizacja GLFW: Sprawdza, czy biblioteka GLFW została poprawnie zainicjalizowana. Jeśli nie, program kończy działanie.

2. Tworzenie okna: Tworzy okno o określonych wymiarach i tytule "Solar System Simulation". Jeśli utworzenie okna się nie powiedzie, kończy działanie programu i zwalnia zasoby GLFW.

3. Ustawianie kontekstu i callbacków:

    - glfw.make_context_current(window) ustawia utworzone okno jako aktywny kontekst OpenGL.

    - Rejestruje callbacki, które odpowiadają za obsługę zmiany rozmiaru okna (framebuffer_size_callback), klawiatury (key_callback), myszy (mouse_button_callback i cursor_position_callback). Callbacki obsługują interakcję użytkownika z symulacją.

4. Inicjalizacja OpenGL: Wywołuje funkcję init(), która ustawia parametry OpenGL, takie jak światło, tekstury i głębię.
5. Główna pętla renderowania:
    - W pętli sprawdza, czy użytkownik zamknął okno.
    - Wywołuje funkcję draw_solar_system(time), aby narysować całą scenę Układu Słonecznego.
    - Wymienia bufory za pomocą glfw.swap_buffers(window), aby zaktualizować wyświetlaną zawartość okna.
    - Obsługuje zdarzenia użytkownika, takie jak naciskanie klawiszy czy poruszanie myszką, za pomocą glfw.poll_events().
    - Aktualizuje czas symulacji (time) o mały krok, aby uzyskać płynny ruch planet.
6. Zamykanie programu: Po zakończeniu działania pętli renderowania zamyka środowisko GLFW za pomocą glfw.terminate(), zwalniając wszystkie zasoby.

Funkcja uruchamia symulację w oknie graficznym, pozwalając użytkownikowi oglądać animowany model Układu Słonecznego i wchodzić z nim w interakcje przy użyciu klawiatury i myszy.

Funkcja framebuffer_size_callback:

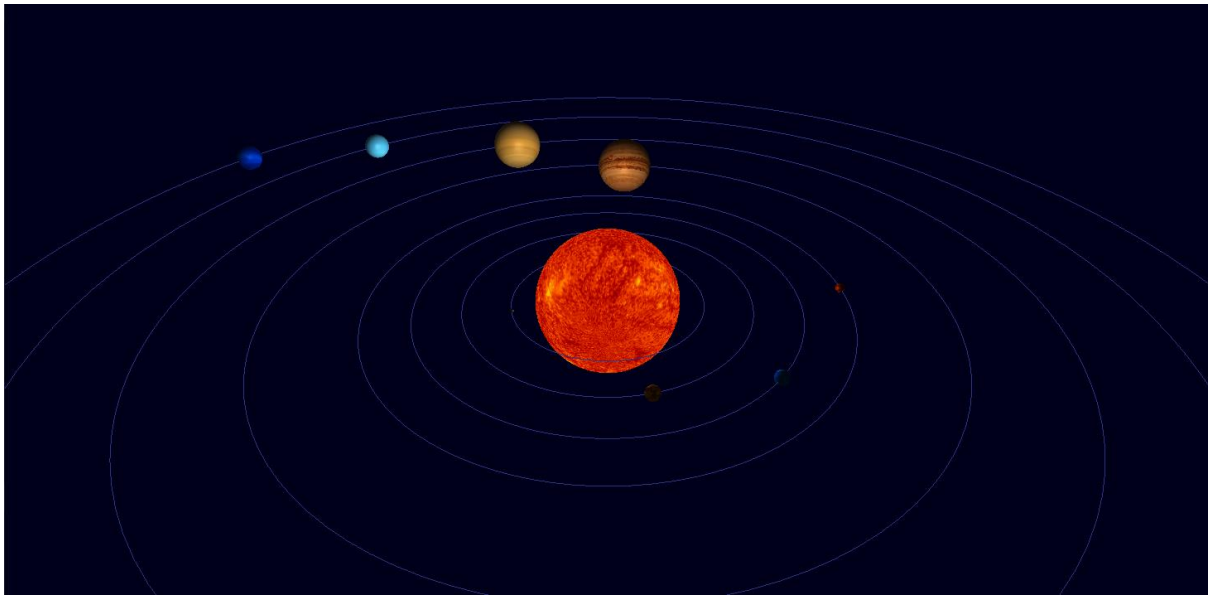
```
373  def framebuffer_size_callback(window, w, h):  1 usage  new *
374      """Adjust viewport when the window is resized."""
375      global width, height
376      width, height = w, h
377      glViewport( x: 0,  y: 0, w, h)
378      glMatrixMode(GL_PROJECTION)
379      glLoadIdentity()
380      gluPerspective(45, w / float(h), 0.1, 100.0)
381      glMatrixMode(GL_MODELVIEW)
```

Funkcja framebuffer_size_callback() dostosowuje widok i projekcję, gdy okno zmienia rozmiar.

1. Aktualizuje szerokość i wysokość okna w globalnych zmiennych.
2. Ustawia nowy obszar widoku za pomocą glViewport.
3. Przełącza na macierz projekcji, resetuje ją, i ustawia nową perspektywę przez gluPerspective.
4. Wraca do macierzy modelowania, by kontynuować renderowanie.

Zapewnia to poprawne skalowanie sceny do nowych wymiarów okna.

Wyniki działania programu:



Podsumowanie:

Program tworzy prosty model 3D układu słonecznego przy użyciu OpenGL. Obejmuje słońce w centrum i kilka planet, które krążą wokół niego. Każda planeta obraca się również wokół własnej osi.

Program inicjuje środowisko OpenGL, ustawia oświetlenie, aby obiekty wyglądały realistycznie, i definiuje materiały, aby nadać słońcu i planetom ich kolory.

Tworzy wiele planet, definiując ich położenie, rozmiary i obroty. Jeśli tekstury nie można załadować, po prostu koloruje planety według koloru podstawowego (Mars — czerwony itd.).

Każda planeta:

Porusza się po orbicie kołowej wokół słońca.

Obraca się wokół własnej osi, tak jak Ziemia obraca się każdego dnia.

Słońce: Słońce jest nieruchome w centrum układu słonecznego i służy jako źródło światła.

Ruch planet (krążących i samoobracających się) jest animowany w czasie rzeczywistym, dzięki czemu układ wygląda dynamicznie.

Użytkownik może wchodzić w interakcję z programem, zmieniając rozmiar okna lub oglądając układ słoneczny pod różnymi kątami.