# Grafika komputerowa i komunikacja człowiek computer

# Laboratorium 6

# Model Układu Słonecznego z teksturowaniem

*Wykonał:*

*Ivan Hancharyk 264511*

*Termin zajęć:*

*WT/TP/8:00*

# 1. Cel ćwiczenia

Celem ćwiczenia było stworzenie modelu Układu Słonecznego z wykorzystaniem teksturowania w bibliotece OpenGL oraz GLUT. Model symuluje ruch planet, ich obrót wokół własnej osi oraz umożliwia interakcję użytkownika poprzez manipulację kamerą.

# 2. Wstęp teoretyczny

Układ Słoneczny, będący głównym tematem symulacji, to zbiór planet, księżyców i innych ciał niebieskich orbitujących wokół Słońca na eliptycznych orbitach. Planety posiadają różne okresy orbitalne, a ich obroty wokół własnej osi determinują długość dnia. W praktyce graficznej, jak w tej symulacji, realistyczne odwzorowanie takiego systemu wymaga zastosowania zaawansowanych technik matematycznych i graficznych.

W projekcie wykorzystano wiele funkcji i mechanizmów biblioteki **OpenGL**, z których najważniejsze to:

1. **Modelowanie sfer planet i ich orbit**:

   - Funkcja **gluSphere** pozwala na generowanie kul (planet i księżyca), gdzie promień oraz liczba segmentów definiują dokładność odwzorowania.
   - Ruch po orbitach jest symulowany za pomocą funkcji obliczających współrzędne eliptyczne planet w danym momencie, np. **radiusAngle** i **translateAngle**.

2. **Manipulacja kamerą i widokiem**:

   - **gluLookAt** – służy do definiowania pozycji obserwatora w przestrzeni oraz kierunku patrzenia. Dzięki temu możliwa jest nawigacja po scenie, w tym przybliżanie i oddalanie widoku.
   - Mechanizm zoomowania oraz obracania kamery jest realizowany za pomocą zmiennych azymuth, elevation i odpowiednich funkcji obsługi myszy.

3. **Nakładanie tekstur**:

   - Funkcja **glTexImage2D** umożliwia przypisanie tekstur obiektom. Wczytane tekstury (np. obrazy planet i Słońca) nadają realistyczny wygląd obiektom 3D.
   - W celu załadowania plików z teksturami wykorzystano funkcję **LoadTGAImage**, przetwarzającą obrazy w formacie **TGA**.

4. **Oświetlenie**:

   - Funkcje **glLightfv** i **glEnable** umożliwiają dodanie źródeł światła w scenie. Światło podkreśla bryłowatość obiektów i dodaje głębi symulacji.
   - Oświetlenie słoneczne jest wzbogacone o wielokrotne źródła światła ustawione wokół sceny, co poprawia jej estetykę.

5. **Renderowanie orbit**:

   - Ruch planet po elipsach oraz rysowanie ich orbit jest realizowane przez **glBegin(GL_LINE_LOOP)** i obliczenia geometryczne. Zapewnia to dokładne odwzorowanie torów ruchu planet.

6. **Interakcja z użytkownikiem**:

   - Funkcje takie jak **Mouse** czy **keys** obsługują wejście od użytkownika. Pozwalają na manipulację sceną w czasie rzeczywistym, np. resetowanie symulacji, regulację prędkości oraz sterowanie kamerą.

Dzięki zastosowaniu tych mechanizmów **OpenGL**, symulacja wiernie odwzorowuje dynamikę Układu Słonecznego, łącząc aspekty matematyczne z wizualizacją graficzną. W dalszej części sprawozdania przedstawiono szczegółowy opis realizacji, wyniki oraz wnioski.

# 3. Realizacja zadania

- **solarSystem()**

```
void solarSystem() {
    if (statusRight == 1) {
        zoom(delta_y > 0);//wykonanie przyblizenia
    }

    if (statusLeft == 1) {//obrot kamera "glowa" obserwatora
        azymuth += ((float)delta_x * pix2angle) * 0.01;
        elevation -= ((float)delta_y * pix2angle) * 0.01;
        if (sin(elevation) >= 0.99) {
            elevation = 1.44;
        }
        if (sin(elevation) <= -0.99) {
            elevation = -1.44;
        }
    }
    viewer[3] = 10 * cos(azymuth) * cos(elevation) + viewer[0];
    viewer[4] = 10 * sin(elevation) + viewer[1];
    viewer[5] = 10 * sin(azymuth) * cos(elevation) + viewer[2];


    orbits();//rysowanie obiektow
    sun();
    planets();
    saturn();
    moon();
}
```

Jest to główna funkcja odpowiedzialna za zarządzanie całym systemem planetarnym. Wywołuje inne funkcje zajmujące się rysowaniem poszczególnych elementów symulacji, takich jak orbity, Słońce, planety i Księżyc.

**Kluczowe wywołania**:

- `orbits()` – rysuje orbity wszystkich planet.
- `sun()` – renderuje Słońce.
- `planets()` – renderuje wszystkie planety z wyjątkiem Saturna.
- `saturn()` – renderuje Saturna wraz z jego pierścieniami.
- `moon()` – renderuje Księżyc orbitujący wokół Ziemi.

Obsługuje również sterowanie kamerą oraz zbliżanie i oddalanie widoku za pomocą zmiennych `azymuth` i `elevation`.

- **orbits()**

```
void orbit(int planet) {
    glColor3f( red: 1,  green1,  blue1);
    glBegin( modeGL_LINE_LOOP);
    for (int i = 0; i < 3600; i++) {
        GLdouble r, x, y;
        GLdouble ratio = 1.0 * GLdouble(i) / GLdouble(360);
        GLdouble angle = orbitPoint(planet, [&] r, ratio);
        translateAngle(r, angle, [&] x, [&] y);
        glVertex3d(x, y: 0, z: y);
    }
    glEnd();
}

void orbits() {
    for (int i = 0; i < 9; i++) {
        orbit(i);
    }
}
```

Wywołuje funkcję `orbit(int planet)` dla każdej planety, aby narysować jej eliptyczną orbitę. Rysowanie odbywa się za pomocą pętli rysującej punkty na orbicie eliptycznej.

- **planetRotation(int planetID)**

```
void planetRotation(int planetID) {
    glRotated( angle: -90.0,  x: 1.0,  y: 1.0,  z: 0.0);//ustawienie odpowiedniego kantu wzgledewm pionu
    glRotated( angle: -planetTilt[planetID],  x: 1.0,  y: 0.0,  z: 0.0);//kat nachylenia planety
    glRotated( angle: 360 * ((((double)day * 24) + hour) / rotation[planetID]),  x: 0.0,  y: 0.0,  z: 1.0);//obrot wokol osi
    gluSphere(sphere, planetSize[planetID],  slicessegments,  stackssegments);
    glRotated( angle: -360 * (((double)day * 24 + hour) / rotation[planetID]),  x: 0.0,  y: 0.0,  z: 1.0);
    glRotated( angle: planetTilt[planetID],  x: 1.0,  y: 0.0,  z: 0.0);
    glRotated( angle: 90.0,  x: 1.0,  y: 1.0,  z: 0.0);
}
```

Realizuje obrót planety wokół własnej osi, uwzględniając jej kąt nachylenia osi (z tablicy `planetTilt`) oraz czas obrotu (z tablicy rotation).

Nachylenie planety jest ustawiane przez:

```
glRotated(-planetTilt[planetID], 1.0, 0.0, 0.0);
```

Obrót wokół osi jest symulowany poprzez przekształcenie czasu (day i hour) na kąt obrotu w stopniach:

```
glRotated(360 * ((((double)day * 24) + hour) /
rotation[planetID]), 0.0, 0.0, 1.0);
```

- **radiusAngle(int planetID, GLdouble& r, GLdouble& angle)**

```
void radiusAngle(int planetID, GLdouble& r, GLdouble& angle) {
    GLdouble time = (double)(day + ((1.0 * hour) / 24));
    GLdouble timeMax = (days[planetID]);
    GLdouble ratio = time / timeMax;
    angle = 2 * M_PI * ratio;//okreslenia kanta obrotu wokol slonca

    double e = orbitsEccentrity[planetID];//obliczenie parametrow elipsy
    double baRatio = sqrt( X1 - pow(e,  y: 2));
    double a = radius[planetID];
    double b = a * baRatio;

    double top = a * (1 - pow(e,  y: 2));
    double bottom = 1 + e * cos(angle);

    r = top / bottom;//policzenie odleglosci plenty od slonca
}
```

Oblicza aktualną odległość planety od Słońca i jej kąt na orbicie eliptycznej. Używa ekscentryczności orbity (z tablicy `orbitsEccentrity`) oraz półosi wielkiej (z tablicy `radius`) do obliczenia kształtu orbity. Wyznacza odległość od Słońca r i kąt angle dla danego momentu symulacji. Algorytm uwzględnia parametry orbity eliptycznej:

```
r = top / bottom;
```

- **planets()**

```
void planets() {
    for (int i = 0; i < 8; i++) {
        if (i == 5) {
            continue;
        }

        GLdouble r, elipseAngle;
        radiusAngle(i,  [&] r,  [&] elipseAngle);

        GLdouble x, y;
        translateAngle(r, elipseAngle,  [&] x,  [&] y);

        glTranslated(x,  y: 0,  z: y);
        texture(i);//wgranie tekstury
        planetRotation(i);//obrot
        glTranslated(-x,  y: 0,  z: -y);
    }
}
```

Renderuje planety (z wyjątkiem Saturna), ustawiając ich pozycję w przestrzeni na podstawie obliczeń z funkcji `radiusAngle`. Obliczenie współrzędnych planety na orbicie:

```
translateAngle(r, elipseAngle, x, y);
glTranslated(x, 0, y);
```

Nałożenie tekstury na planetę:

```
texture(i);
```

Renderowanie planety jako kuli:

```
planetRotation(i);
```

- **saturn()**

```
void saturn() {
    GLdouble r, elipseAngle;
    radiusAngle( planetID: 5,  [&] r,  [&] elipseAngle);

    GLdouble x, y;
    translateAngle(r, elipseAngle,  [&] x,  [&] y);

    glTranslated(x,  y: 0,  z: y);
    texture(5);//wgranie tekstury
    planetRotation( planetID: 5);//obrot

    glRotated( angle: 1.2 * planetTilt[5],  x: -1.0,  y: 0.0,  z: 1.0);//nachylenie pierscieni
    for (double i : rings) {
        glColor3f( red: 1,  green1,  blue1);
        glBegin( mode GL_LINE_LOOP);
        for (int ii = 0; ii < 3600; ii++) {//rysowanie pierscienia
            float angle = 1.0 * float(ii) / float(3600);
            float xr = i * cos( X2 * M_PI * angle);
            float yr = i * sin( X2 * M_PI * angle);

            glVertex3f( x: xr,  y: 0,  z: yr);
        }
        glEnd();
    }

    glRotated( angle: -1.2 * planetTilt[5],  x: -1.0,  y: 0.0,  z: 1.0);
    glTranslated(-x,  y: 0,  z: -y);
}
```

Specjalna funkcja renderująca Saturna wraz z jego pierścieniami. Planeta jest renderowana podobnie jak w funkcji `planets()`.Pierścienie są rysowane w pętli dla każdej wartości promienia z tablicy `rings`:

```
for (double i : rings) {
    glBegin(GL_LINE_LOOP);
    for (int ii = 0; ii < 3600; ii++) {
        float xr = i * cos(2 * M_PI * angle);
        float yr = i * sin(2 * M_PI * angle);
        glVertex3f(xr, 0, yr);
    }
    glEnd();
}
```

- **moon()**

```
void moon() {
    GLdouble earthRadius, earthAngle;
    radiusAngle( planetID: 2,  r: [&] earthRadius,  [&] earthAngle);

    GLdouble earthX, earthY;
    translateAngle(earthRadius, earthAngle,  [&] earthX,  [&] earthY);

    GLdouble moonRadius = 21.5;
    GLdouble moonAngle = 2 * M_PI * (hour % 24) / 24.0;

    // Oblicz pozycję Księżyca z nachyleniem orbity
    GLdouble moonX = earthX + moonRadius * cos(moonAngle);
    GLdouble moonY = earthY + moonRadius * sin(moonAngle);
    GLdouble moonZ = 5.0 * sin(moonAngle); // Nachylenie orbity Księżyca

    glTranslated(moonX,  y: moonZ,  z: moonY);

    texture(9); // 9 to indeks tekstury Księżyca w tablicy `textures`
    gluSphere(sphere,  radius2.77,  slicessegments,  stackssegments);

    glTranslated(-moonX,  y: -moonZ,  z: -moonY);
}
```

Renderuje Księżyc orbitujący wokół Ziemi. Obliczane są współrzędne Ziemi na orbicie, a następnie pozycja Księżyca wokół Ziemi, uwzględniając nachylenie orbity:

```
GLdouble moonX = earthX + moonRadius * cos(moonAngle);
GLdouble moonY = earthY + moonRadius * sin(moonAngle);
GLdouble moonZ = 5.0 * sin(moonAngle);
```

Tekstura Księżyca jest nakładana, a on sam renderowany jako kula:

```
texture(9);
gluSphere(sphere, 2.77, segments, segments);
```

- **texture(int textureID)**

```
void texture(int textureID) {
    glTexImage2D( targetGL_TEXTURE_2D,  level: 0,
            internalformat: ImComponents[textureID],
            width: ImWidth[textureID],
            height: ImHeight[textureID],
            border: 0,  format: ImFormat[textureID],
            type: GL_UNSIGNED_BYTE,
            textures[textureID]);
}
```

Ładuje odpowiednią teksturę na aktualnie renderowany obiekt. Korzysta z wcześniej załadowanych danych tekstur:

```
glTexImage2D(GL_TEXTURE_2D, 0, ImComponents[textureID],
ImWidth[textureID], ImHeight[textureID], 0, ImFormat[textureID],
GL_UNSIGNED_BYTE, textures[textureID]);
```

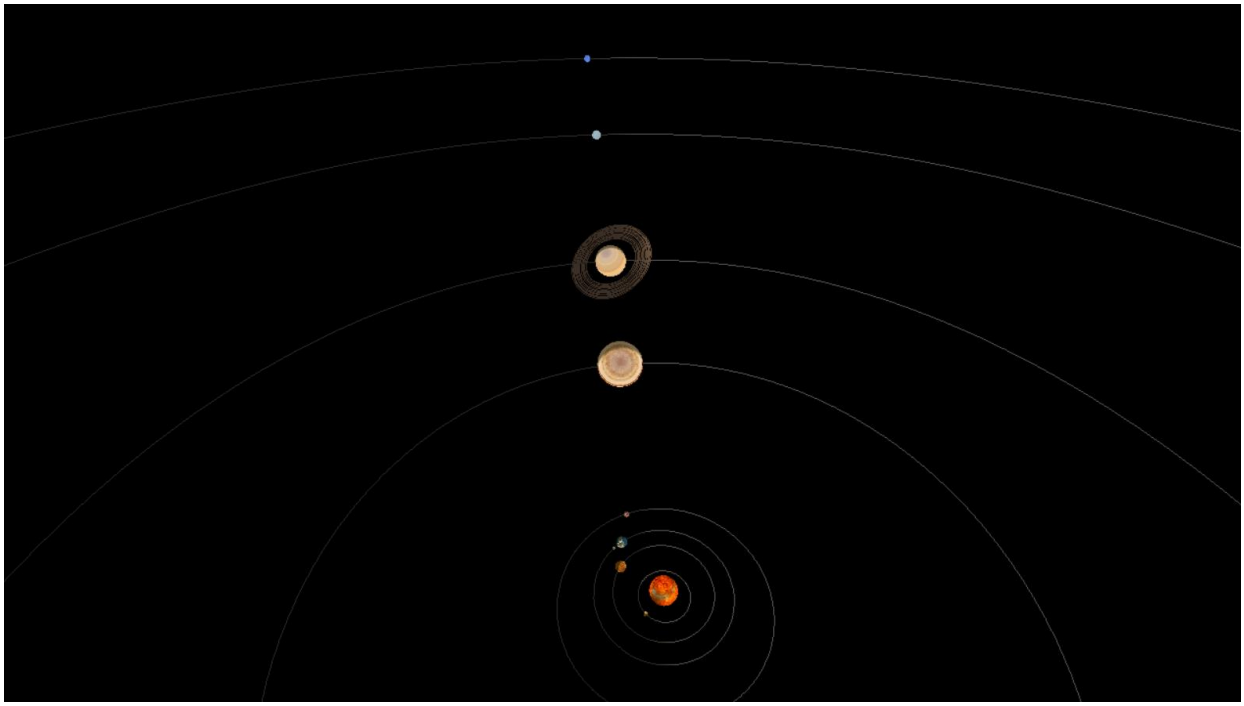
- **light() i lightInit()**

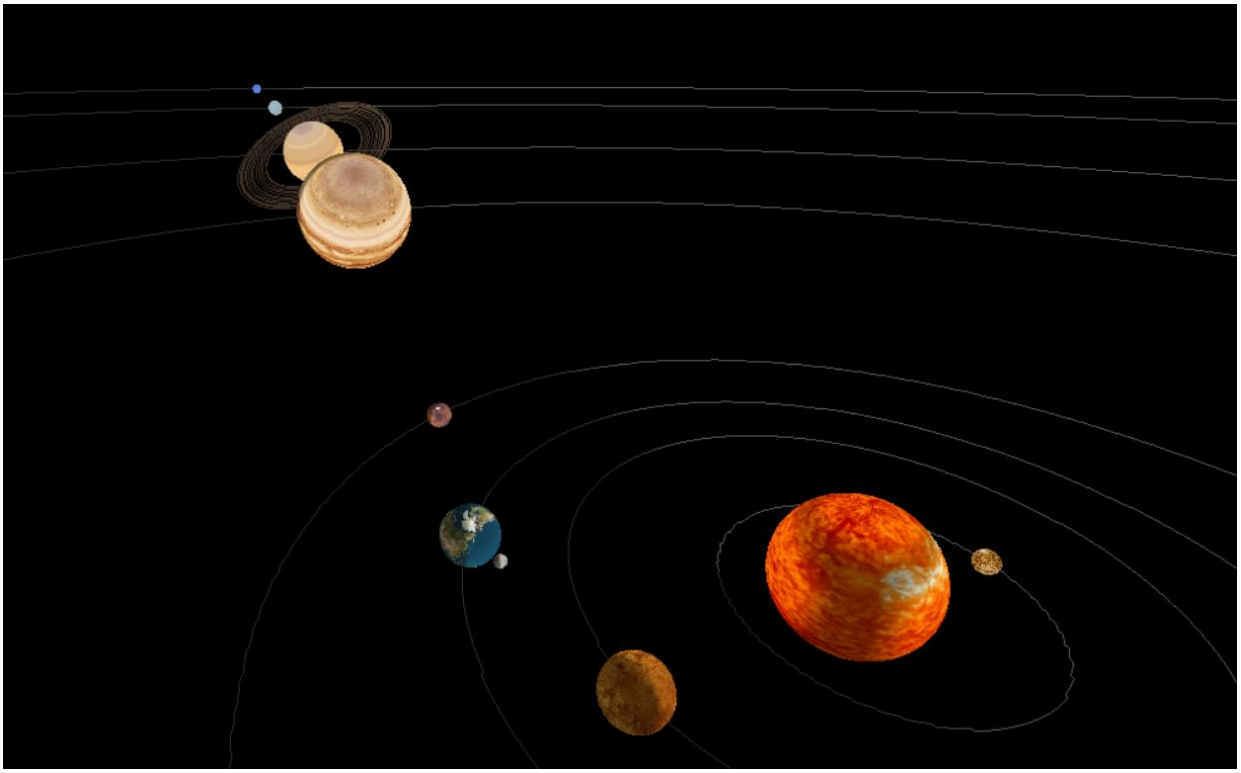
```
void light() {
    int lightRadius = 50;//ustawienie zrodel swiatla
    int lightAngle = 60;
    for (int i = 0; i <= 5; i++) {

        GLfloat x = cos( x: i * lightAngle) * lightRadius;//obliczenie pozycji zrodel swiatla
        GLfloat y = -1 * sin( x: i * lightAngle) * lightRadius;
        GLfloat LightPosition[] = { x, 0.0f, y, 1.0f };
        glLightfv( light: GL_LIGHT0 + i,  pname: GL_POSITION, LightPosition); //swiatla wokol
    }

    GLfloat topLightPosition[] = { 0.0f, 50.0f, 0.0f, 1.0f };//gorne swiatlo
    glLightfv( light: GL_LIGHT6,  pname: GL_POSITION, topLightPosition);
    GLfloat bottomLightPosition[] = { 0.0f, -50.0f, 0.0f, 1.0f };//dolne swiatlo
    glLightfv( light: GL_LIGHT7,  pname: GL_POSITION, bottomLightPosition);
}

void lightInit() {
    GLfloat att_constant = { 1.0 };//parametry swiatla
    GLfloat att_linear = { 0.0000005 };
    GLfloat att_quadratic = { 0.000000000001 };
    float LightAmbient[] = { 0.1f, 0.1f, 0.05f, 1.0f };
    float LightEmission[] = { 1.0f, 1.0f, 0.8f, 1.0f };
    float LightDiffuse[] = { 1.0f, 1.0f, 0.8f, 1.0f };
    float LightSpecular[] = { 1.0f, 1.0f, 1.0f, 1.0f };

    for (int i = 0; i <= 7; i++) {//ustawienie zrodel
        glLightfv( light: GL_LIGHT0 + i,  pname: GL_AMBIENT, LightAmbient);
        glLightfv( light: GL_LIGHT0 + i,  pname: GL_DIFFUSE, LightDiffuse);
        glLightfv( light: GL_LIGHT0 + i,  pname: GL_SPECULAR, LightSpecular);
        glLightf( light: GL_LIGHT0 + i,  pname: GL_CONSTANT_ATTENUATION, att_constant);
        glLightf( light: GL_LIGHT0 + i,  pname: GL_LINEAR_ATTENUATION, att_linear);
        glLightf( light: GL_LIGHT0 + i,  pname: GL_QUADRATIC_ATTENUATION, att_quadratic);
        glEnable( cap: GL_LIGHT0 + i);
    }

    glEnable( cap: GL_LIGHTING);
    glEnable( cap: GL_DEPTH_TEST);
}
```

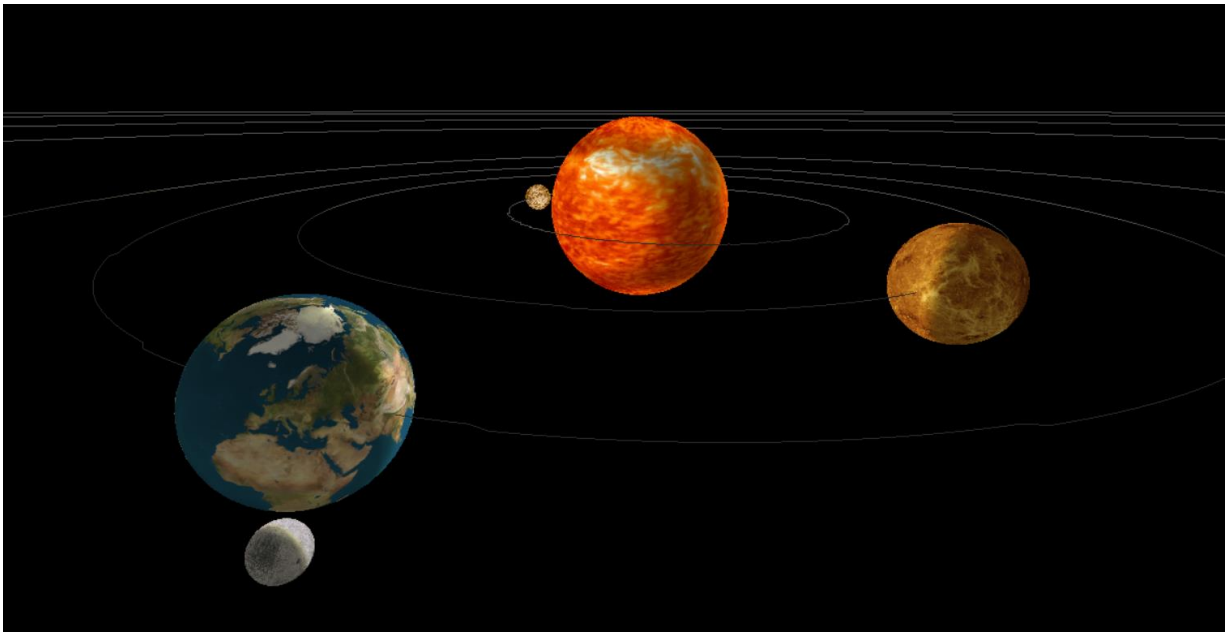
**`lightInit()`**: Inicjalizuje parametry źródeł światła (ambient, diffuse, specular) oraz dodaje ich pozycje w przestrzeni.

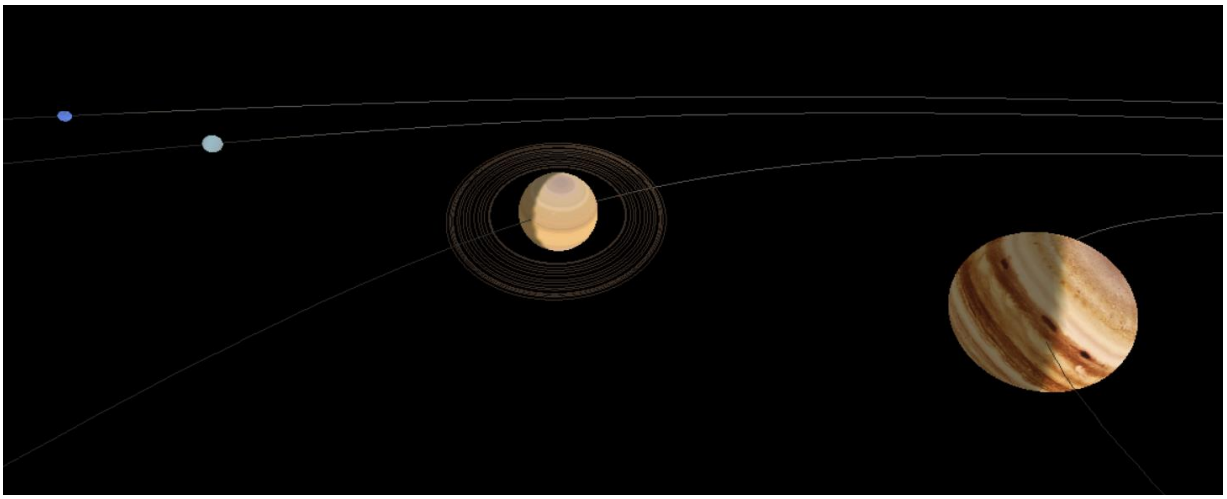
**`light()`**: Wykorzystywana w każdej klatce do ustawienia dynamicznych pozycji świateł wokół Słońca oraz na górze i dole sceny.

**Efekt działania programu (układ słoneczny z różnych pozycji kamery):**









# 4. Wnioski

Zrealizowana symulacja Układu Słonecznego za pomocą biblioteki **OpenGL** umożliwia realistyczne odwzorowanie ruchu planet, ich obrotów oraz interakcję użytkownika. Zastosowanie tekstur, oświetlenia i zaawansowanych funkcji graficznych poprawiło estetykę oraz realizm wizualizacji. Projekt ukazuje znaczenie matematyki w grafice komputerowej oraz potencjał **OpenGL** w tworzeniu dynamicznych scen 3D.