Grafika komputerowa - Symulacja Układu Słonecznego

Kacper Małkowski 252724

Prowadzący - dr. inż. Jan Nikodem
Zajęcia: Wtorek P 7.30-10.30

1 Wstęp

Celem miniprojektu było napisanie symulacji układu słonecznego z wykorzystaniem umiejętności i wiedzy z zakresu bliblioteki GLUT poznanej w trakcie całego semestru. Projekt wymaga umiejętności z każdego labolatorium: rysowanie brył, poruszanie obiektami, ruch kamerą, użycie klawiatury i myszy, oświetlenie i teksturowanie.

2 Plan działania

W celu napisania symulacji podzieliłem projekt na mniejsze podproblemy, stanowiące swoistą ścieżkę postępu prac:

1. Rysowanie planet w odpowiednich wielkościach.
2. Rysowanie orbit w skali.
3. Teksturowanie planet, dodanie pierścieni saturnowi.
4. Ruch obiegowy planet wokół słońca.
5. Ruch obrotowy planet i nadanie im odpowiednich kątów nachylenia względem płaszczyzny układu.
6. Zmiana orbit z okrągłych na eliptyczne i nadanie odpowiednich parametrów ruchu względem II prawa Keplera.
7. Dodanie oświetlenia.

Punkty 6 i 7 zostały zrealizowane tylko częściowo. Orbity zostały zmienione na elipsy, ale nie udało mi się zmienić ruchu planet na zgodny z II prawem Keplera i planety krążą ze stałą prędkością kątową. Oświetlenie także nie jest zrobione w sposób w jakich chciałem i w celu uzyskania oświetlenia byłem zmuszony umieścić 8 źrodeł światła wokoł słońca. Kolejnym problemem z oswietleniem jest nierównomierne oswietlenie orbit. W symulacji generuję orbity jako okregi zrobione przy użyciu LINE_LOOP, a źródła światła oświetlają tylko jedną połowę orbity.

3 Program

Z racji na wysoki stopień rozbudowania kodu nie będę opisywał całości kodu. Skupię się tutaj tylko na opisie naistotniejszych metod symulacji. Cały kod wraz z komentarzami znajduje się na githubie po linkiem https://github.com/TorpedusMaximus/Solar-System-Simulation.

3.1 Generowanie układu

```
1  void solarSystem() {
2    if (statusRight == 1) {
3      zoom(delta_y > 0);//wykonanie przyblizenia
4    }
5
6    if (statusLeft == 1) {//obrot kamera "glowa" obserwatora
7      azymuth += ((float)delta_x * pix2angle) * 0.01;
8      elevation -= ((float)delta_y * pix2angle) * 0.01;
9      if (sin(elevation) >= 0.99) {
10       elevation = 1.44;
11     }
12     if (sin(elevation) <= -0.99) {
13       elevation = -1.44;
14     }
15   }
16   viewer[3] = 10 * cos(azymuth) * cos(elevation) + viewer[0];//ustalenie pozycji
         puktu na ktory patrzymy
17   viewer[4] = 10 * sin(elevation) + viewer[1];
18   viewer[5] = 10 * sin(azymuth) * cos(elevation) + viewer[2];
19
20
21   orbits();//rysowanie obiektow
22   sun();
23   planets();
24   saturn();//saturn rysowany oddzielnie ze wzgledu na rysowanie pierscieni
25 }
```

Metoda jest wywoływana w RenderScene() i służy do wyrysowania wszystkich obiektów, oraz określenia parametrów kamery.
Linijki 2-18 są kopią wcześniej używanych rozwiazań i służą do ruchu kamery i zooma. Zastosowałem tutaj sposób poruszania polegający na rozglądaniu się przy użyciu lewego przycisku myszy i przybliżania i oddalania przy uzyciu prawego przycisku myszy. Dodatkowo poruszanie odbywa się przy użyciu klawiszy WASD.
Linijki 21-24 wywołują konkretne funkcje rysujące kolejno: orbity, słońce, planety bez saturna i na koniec saturna. Saturn jest rysowany osobno w celu ułatwienia generowania jego pierścieni.

3.2 Generowanie planet

```
void planets() {
  for (int i = 0; i < 8; i++) {
    if (i == 5) {//saturn pominiety ze wzgledu na pierscienie
      continue;
    }

    GLdouble r, elipseAngle;
    radiusAngle(i, r, elipseAngle);//obliczenie odleglosci od slonca i aktualnego
        katu wzgledem punktu starowego orbity

    GLdouble x, y;
    translateAngle(r, elipseAngle, x, y);//obliczenie pozycji x i y wzgledem
        odleglosci i katu

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

W celu wygenerowania planety potrzebne jest kilka parametrów. Metoda najpierw oblicza r i elipseAngle (funkcja radiusAngle opisana w punkcie 3.3) w celu określenia kątu wzgledem początku obiegu i odległośći planety od słońca. Następnie na podstawie tych danych obliczana jest pozycja w układzie kartezjańskim, w celu poznania miejsca, w którym wygenerowany ma być obiekt (funkcja translateAngle opisana w punkcie 3.3).
Znając współrzędne metoda wybiera odpowiednią teksturę dla planety i wywołuje metodę odpowiedzialną za rysowanie planety pod odpowiednimi kątami(funkcja planetRotation w punkcie 3.4).

3.3 Generowanie ruchu obiegowego

```
1  void translateAngle(int r, GLdouble elipseAngle, GLdouble& x, GLdouble& y) {
2    x = cos(elipseAngle) * r;
3    y = -1 * sin(elipseAngle) * r;
4  }
5
6  void radiusAngle(int planetID, GLdouble& r, GLdouble& angle) {
7    GLdouble time = (double)(day + ((1.0 * hour) / 24));
8    GLdouble timeMax = (days[planetID]);
9    GLdouble ratio = time / timeMax;
10   angle = 2 * M_PI * ratio;//okreslenia kanta obrotu wokol slonca
11
12   double e = orbitsEccentrity[planetID];//obliczenie parametrow elipsy
13   double baRatio = sqrt(1 - pow(e, 2));
14   double a = radius[planetID];
15   double b = a * baRatio;
16
17   /*GLdouble p = pow(b, 2) / a;              //proba dostosowania predkosci planet
         (II prawo Keplera)
18   GLdouble M = 2 * p * ratio;
19
20   GLdouble E[2];
21
22   E[1] = M;
23
24   do {
25     E[0] = E[1];
26     E[1] = M + e * sin(E[0]);
27   } while (abs(E[1] - E[0]) > 0.001);
28
29   GLdouble cosn = (cos(E[1]) - e) / (1 - e * cos(E[1]));
30   angle = cosh(cosn);
31   if (planetID == 0)
32     cout << angle << endl;*/
33
34   double top = a * (1 - pow(e, 2));
35   double bottom = 1 + e * cos(angle);
36
37   r = top / bottom;//policzenie odleglosci plenty od slonca
38 }
```

Metody służą do określenia miejsca, w którym generowany ma być obiekt.
Najpierw wywoływana jest metoda radiusAngle (wyjaśnione w 3.2). Na podstawie minionej liczby dni oraz dni w okresie obiegu, wyliczany jest kąt przesunięcia planety (linijki 7-10). Następnie wyliczane są parametry elipsy ze znajomości mimośrodu i półosi wielkiej (linijki 12-15). Znając wszystkie te wartosci używamy wzoru:

$$r = \frac{a * (1 - sin^2(e))}{1 + e * cos(\theta)}$$

i obliczamy odległość od słońca.
Obliczony kąt i odległość używamy w metodzie translateAngle w celu przekształcenia ich we współrzędne kartezjańskie.
W metodzie radiusAngle zauważyć można próbę dostosowania prędkości obiegu planet w odniesieniu do II prawa Keplera (linijki 17-32). Wymienione obliczenia oparłem na przykładzie ze strony https://www.bogan.ca/orbits/kepler/keplerex.html, ale metoda ta powodowała jedynie ruch oscylacyjny planet.

3.4 Generowanie ruchu obrotowego

```
void planetRotation(int planetID) {
  glRotated(-90.0, 1.0, 1.0, 0.0);//ustawienie odpowiedniego kantu wzgledewm pionu

  glRotated(-planetTilt[planetID], 1.0, 0.0, 0.0);//kat nachylenia planety

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

  gluSphere(sphere, planetSize[planetID], segments, segments);

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

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

  glRotated(90.0, 1.0, 1.0, 0.0);
}
```

Metoda jest wywoływana kiedy wiemu już w jakim miejscu wygenerować należy planetę. Metoda najpierw obraca obiekt o 90°, gdyż planety generowane są w innej orientacji niż potrzebna. Następnie pochylam planetę o kąt względem płaszczyzny układu. Ostatni obrót jaki nadajemy to obrót względem osi obrotu planety. Kąt jest obliczany wzgledem minionego czasu. Znając wszystkie obroty planety i jej pozycję, finalnie można wygenerować pożądaną bryłę.

3.5 Użycie tekstur

```
void loadTextures() {//wczytanie wszytskich tekstur do pamieci
  const char* plik[] = {
    "mercury.tga",
    "venus.tga",
    "earth.tga",
    "mars.tga",
    "jupiter.tga",
    "saturn.tga",
    "uranus.tga",
    "neptune.tga",
    "sun.tga",
    "orbits.tga"
  };
  GLbyte* pBytes;

  int i = 0;
  for (auto sciezka : plik) {
    pBytes = new GLbyte;
    pBytes = LoadTGAImage(sciezka, &ImWidth[i], &ImHeight[i], &ImComponents[i],
        &ImFormat[i]);//wczytanie tekstury
    textures[i] = pBytes;//zapis do pamieci
    i++;
  }
}
```

Istotnym elementem jest system zarządzania teskturami. Z racji na ilość wczytywań i planet postanowiłem zmniejszyć liczbę wczytań do minimum i trzymać wczytane tektury w pamięci programu. w metodzie deklaruje pliki do wczytania, a następnie wczytuje je przy użyciu funkcji ze strony labolatorium. Każda tekstura jest wczytana do pamięci razem z width, hight i components.

3.6 Mijający czas

```
void dayByDay() {//funkcja w tle zapeniajaca uplyw czasu
  hour += speed;
  Sleep(10);
  RenderScene();

  if (day >= INT_MAX - 5 || day <= INT_MIN + 5) {//zabezpieczenie przed
      przepelnieniem
    day = 0;
  }
}
```

Metoda jest wykonywana jako GLUTIdleFunc i działa w tle. Służy do inkrementacji lub dekrementacji czasu. Do aktualnej godziny dodawana jest aktualna prędkość. Akcja ta wykonuje się co 10ms. Ostatnim elementem metody jest zabezpieczenie przed wykroczeniem poza zakres wartości int. Dodatkowo w funkcji RenderScene jest warunek który zmienia ilość minionych dni w zależności od godziny, gdzie if(hour>=24){hour-=24; day++;}.

4 Wnioski

Napisanie zadanej symulacji nie było zadaniem bardzo skomplikowanym, ale za to bardzo rozbudowanym. Przez stopień złożoności nie wszystkie założenia udało się spełnić co opisałem w punkcie 2. W celu widoczności symulacji byłem zmuszony zmienić niektóre alementy układu tak by nie był zgodne ze skalą. Wielkości kośmiczne są tak ogromne, że planet by nie było widać obok ogromnego słońca. Dlatego zastosowałem nastepujace skale:

- Orbity skala 1:1
- planety wewnętrzne skala 1000:1
- planety zewnętrzne skala 500:1
- słońce skala 22:1