Grafika Komputerowa i Komunikacja Człowiek-Komputer

Projekt- Kosmiczny Imbryczek

Dawid Piechota

27 stycznia 2019

Spis treści

1 Omówienie tematu

Wykonanym przeze mnie projektem jest symulacja poruszania się w ruchomym układzie planetarnym. Użytkownik steruje kosmicznym imbryczkiem dryfującym między planetami za pomocą klawiatury. Zostały zaimplementowane: sterowanie imbryczkiem, grawitacja, kolizje, oświetlenie, teksturowanie oraz dźwięki. Wiedza z zajęć laboratoryjnych umożliwiła mi wykonanie projektu w większości bezproblemowo. Dodatkowo rozbudowałem symulator o obręcze obok każdej planety przez które należy 'przelecieć' żeby zakończyć grę. Dokładny opis gry znajduje się w instrukcji dołączone jdo programu jako dokument pdf

2 Omówienie klas

Zastosowałem podejście obiektowe i wyróżniłem następujące klasy.

2.1 Klasa Planet

Klasa Planet zawiera zmienne opisujące parametry planety oraz metody wyznaczające jej pozycje. Najważniejsze elementy klasy Planet:

2.1.1 Zmienne

- posX, posZ- pozycja plnaety w układzie,
- theta- zmienna używana do wyliczenia pozycji,
- rotationSpeed- prędkość kątowa planety wokół punktu 0,0,0,
- orbitRadius- odległość od punktu 0,0,0,
- mass- zmienna używana do obliczenia siły grawitacji danej planety.

2.1.2 Metoda PositionCalc()

Obliczanie pozycji planety w zależności od stale zmieniającej się wartości theta.

```
1 void Planet::positionCalc()
2 {
3         posX = orbitRadius * sin(theta);
4         posZ = orbitRadius * cos(theta);
5 }
```

2.1.3 Metoda getGravityForceX(...)

Obliczanie siły grawitacji planety na dany punkt w przestrzeni w osi X.

```
float Planet::getGravityForceX(float shipX, float shipZ)
{
return mass / (distance2(posX, posZ, shipX, shipZ)) *
(shipX - posX) / (fabs(shipX - posX) + fabs(shipZ - posZ));
//pozycja statku relatywna do planety to (xs-xp,ys-yp)
}
```

2.2 Klasa Spaceship

Klasa spaceship zawiera zmienne opisujące parametry statku oraz metody wyznaczające jej pozycje. Najważniejsze elementy klasy Spaceship:

2.2.1 Zmienne

- struct velocity- zmienna opisująca kierunek prędkości obiektu,
- struct position- zmienna opisująca pozycję obiektu,
- struct orientation- zmienna opisująca orientację obiektu,
- thrustPower- zmienna opisująca szybkość zmiany prędkości.

2.2.2 Metoda idleDrifting()

Aktualizacja położenia w zależności od kierunku prędkości.

```
void Spaceship::idleDrifting()
{
        position.X += velocity.X;
        position.Z += velocity.Z;
}
```

2.2.3 Metoda propulsionRight()

Metoda aktywna podczas trzymania klawisza 'D'.

```
void Spaceship::propulsionRight()
{
velocity.X -= thrustPower *  0.000005 * cos(orientation.Y * PI / 180 + PI / 2);
velocity.Z -= thrustPower * 0.000005 * sin(orientation.Y * PI / 180 + 3 * PI / 2);
}
```

2.2.4 Metoda gravityEffect(...)

Zmiana prędkości w zależności od zadanej siły grawitacji.

```
1 void Spaceship::gravityEffect(float gravityForceX, float gravityForceZ)
2 {
3         velocity.X -= 0.00002 * gravityForceX;
4         velocity.Z -= 0.00002 * gravityForceZ;
5 }
```

2.3 Klasa Image

Klasa obsługująca wczytywanie i przechowywanie tekstur w postaci tablicy pikseli.

3 Kolizje

Wykrywanie kolizji realizowane jest przez funkcję zwracającą kwadrat odległości pomiędzy dwoma punktami:

```
1 float distance2(float x1, float y1, float x2, float y2)
2 {
3         return (x2 - x1) * (x2 - x1) + (y2 - y1) * (y2 - y1);
4 }
```

Wykorzytuję kwadrat odległości ponieważ pierwiastkowanie jest kosztowne obliczeniowo. Poza tym, przy obliczaniu siły grawitacji potrzebny jest kwadrat odległości co jest kolejnym argumentem za pominięciem pierwiastkowania. W przypadku kolizji następuje reset pozycji imbryczka.

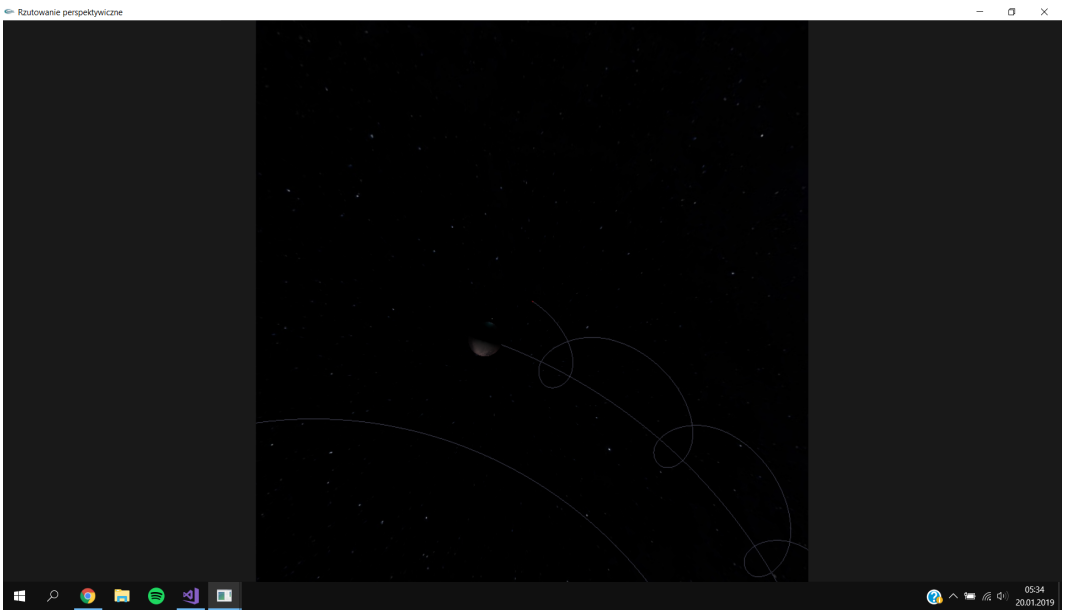
4 Warunek zakończenia gry

Warunkiem zakończenia gry jest 'przelecenie' przez wszystkie aktywne obręcze co aktywuje obręcz przy planecie Ziemia. Po zaliczeniu ziemskiej obręczy następuje akcja zakończenia gry- odegranie dźwięku i zmiana kolorów imbryczka.

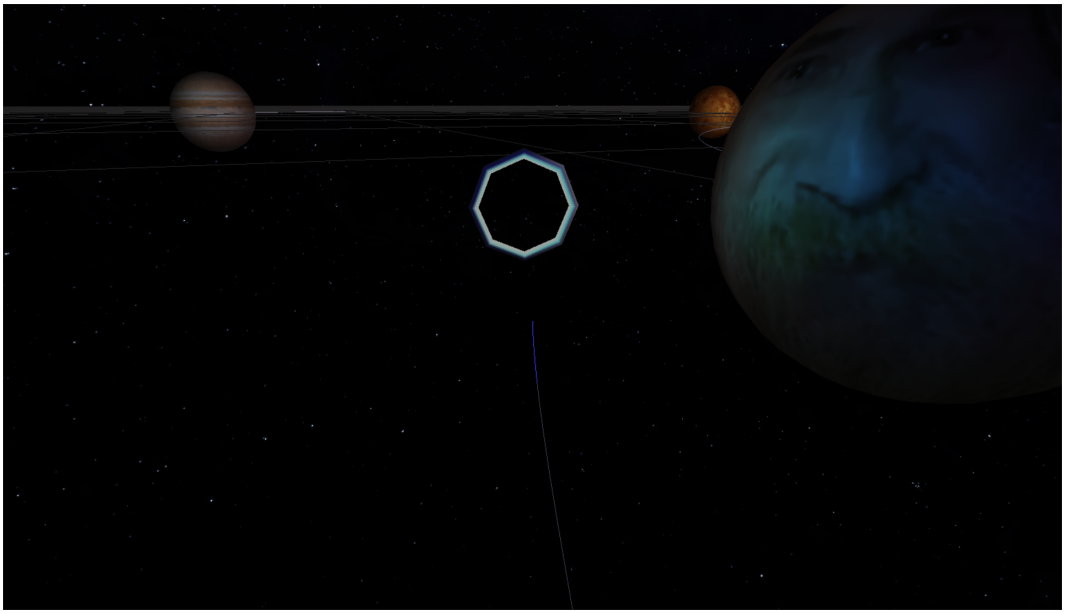
Funkcja donutColission():

```
void donutCollision()
{
if (flyingThroughDonut == 0)
{
for (int i = 0; i < N; i++)
{
        if (!lastDonut && i == earthPosition) continue;
        if ((planets[i].donutOn) && (distance2(spaceship.position.X,
        spaceship.position.Z, planets[i].donutPosX, planets[i].donutPosZ) < 0.5f))
        {
flyingThroughDonut = 1;
currentDonut = i;
if (lastDonut) {
        point = engine->play2D("audio/congratulations.mp3", true, true);
        music->setIsPaused(true);
        point->setIsPaused(false);
        point->setVolume(1.0); }
else point = engine->play2D(
"audio/point.wav", false, true); point->setIsPaused(false); point->setVolume(0.7);
points++;
teapotSize+=0.7;
pointCheck();
break;
        }
}
}
else
{
if (distance2(spaceship.position.X, spaceship.position.Z,
planets[currentDonut].donutPosX, planets[currentDonut].donutPosZ) > 0.7f)
{
        flyingThroughDonut = 0;
        planets[currentDonut].donutOn = 0;
}
}
}
```

5 Rezultat prac



Rysunek 1: Orbitowanie wokół planety orbitującej wokół słońca



Rysunek 2: Wynik działania programu po zmienieniu rozmiarów rysowanego okna i włączeniu trybu fullscreen

6 Potencjalna rozbudowa

- Każda planeta może być realizowana przez klasę Spaceship, co spowoduje ruch planet zależny od grawitacji zamiast ruchu po zadanym okręgu. Powoduje to jednak problemy z szybką destabilizacją układu przy małych odległościach,
- Dodanie większej ilości imbryczków. Np w postaci wrogich statków kosmicznych,
- Dodanie trzeciej osi do symulacji (trudne),
- Dodanie GUI, które pozwoli na prostą i szybką zmianę parametrów takich jak gravityStrenght.