WOJSKOWA AKADEMIA TECHNICZNA

Grafika komputerowa

Sprawozdanie z pracy laboratoryjnej Nr 3

Temat: Przekształcenia geometryczne.

Prowadzący: dr inż. Marek Salamon

Autor: Mateusz Jasiński WCY20IJ1S1

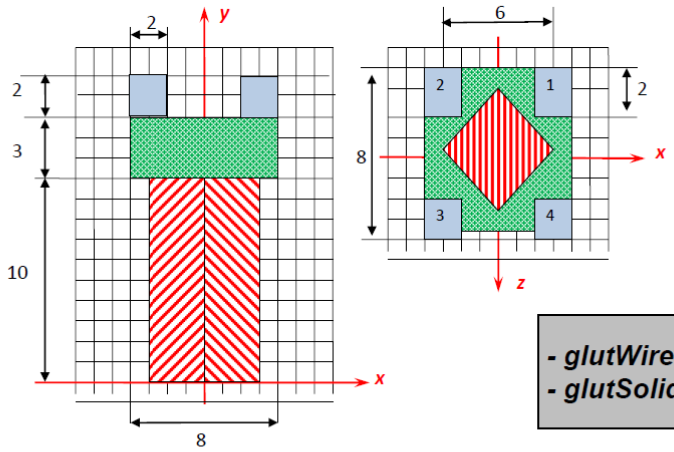
Data wykonania: 07.12.2022

1. Treść zadania

Lab 3 - Zestaw 5b:

Zadanie 1

Wykorzystując funkcję ***glutSolidCube*** oraz funkcje działające na stosie macierzy modelowania napisać fragment programu odpowiedzialny za utworzenie obiektu, którego rzuty przedstawiono na rysunkach.

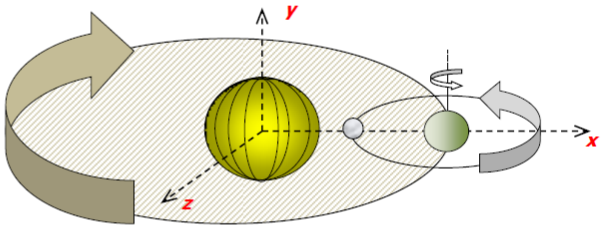


***- glutWireCube**(GLdouble size);*
***- glutSolidCube**(GLdouble size);*

Uwaga: Zadanie realizowane jest od stanu:
1. Po wykonaniu rzutu perspektywicznego
2. Po ustawieniu aktywnego stosu macierzowego przeznaczonego do modelowania.
3. Po załadowaniu układu jednostkowego i ustawieniu położenia obserwatora

Zadanie 2

Wykorzystując funkcję **glutWireSphere** oraz funkcje działające na stosie macierzy modelowania napisać program odpowiedzialny za dynamiczne tworzenie sceny przedstawiającej układ słoneczny złożony ze słońca oraz jednej planety i jej księżyca.



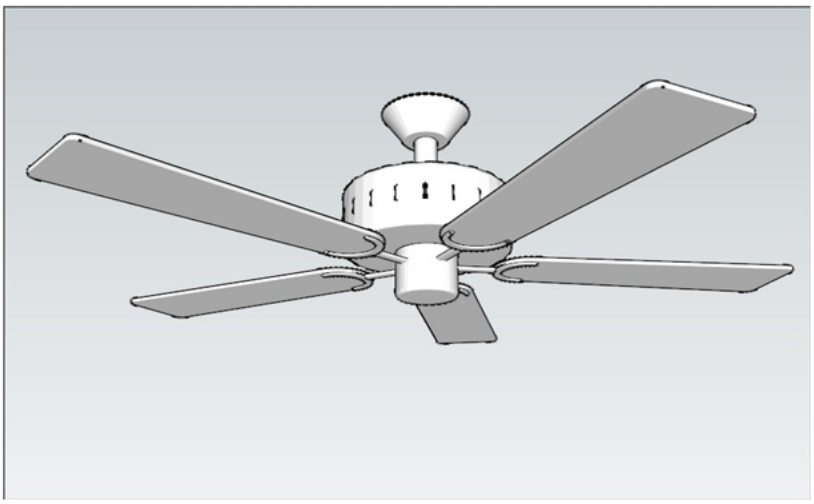
Dane:
- promień słońca = 5;
- promień planety = 2;
- promień księżyca = 0.5;
- promień orbity planety = 20;
- promień orbity księżyca = 5;
- prędkość kątowa planety = 0.25 stopnia/klatkę (kierunek CW [-])
- prędkość kątowa księżyca = 0.5 stopni/klatkę (kierunek CCW [+])
- spin planety = 1 stopień na klatkę (CCW [+]);
- obie orbity leżą w płaszczyźnie ***xz***.

***glutWireSphere**(GLdouble radius, GLint slices, GLint stacks);*
***glutSolidSphere**(GLdouble radius, GLint slices, GLint stacks);*
parametry: *radius* – promień sfery, *slices* - liczba południków, *tacks* – liczba równoleżników.
Bieguny sfery leżą na osi Z.

1. Wykorzystując projekt „**robot**” napisać fragment programu odpowiedzialny za:
   a) zmianę odległości obserwatora od obiektu w zakresie (*odlmin, odlmax*); 0.5 pkt.
   b) zmianę orientacji obserwatora (bez ograniczeń) względem osi *0X, 0Y i 0Z*. 0.5 pkt.
2. Przesunąć obiekt z projektu „**robot**” w położenie (0.0, 60.0, 0.0) i wykorzystując funkcję ***glutSolidCube*** oraz funkcje działające na stosie macierzy modelowania napisać program odpowiedzialny za utworzenie obiektu z Zad.1 (ćw.). 1 pkt.
3. Przesunąć obiekt z Zad.1 w położenie (0.0, 80.0, 0.0) i wykorzystując funkcję ***glutWireSphere*** napisać program przedstawiający układ planetarny zgodnie z parametrami przedstawionymi w Zad.2 (ćw.). 2 pkt.
   Układ uzupełnić o:
   a) planetę o promieniu 6 poruszająca się wokół słońca po orbicie o promieniu 60 z prędkością kątową 0.15 stopnia/klatkę animacji w kierunku CCW w płaszczyźnie nachylonej do osi OX pod kątem 30 stopni
   b) planetę o promieniu 4 poruszająca się wokół słońca po orbicie o promieniu 80 z prędkością kątową 0.25 stopnia/klatkę animacji w kierunku CW w płaszczyźnie nachylonej do osi OX pod kątem -15 stopni;
   a) 0.5 pkt., b) 0.5pkt.
   Zaznaczyć orbity poruszających się obiektów. Bieguny generowanych sfer powinny leżeć w osi pionowej. 1 pkt.
4. Napisać program przedstawiający obiekt zbudowany z prymitywów przestrzennych udostępnianych przez biblioteki ***GLU*** i ***GLUT***. /zadanie indywidualne oceniane na podstawie sprawozdania/ 4 pkt.

Zadanie indywidualne - wiatrak:

Napisać program przedstawiający obiekt z rysunku poniżej zbudowany z prymitywów przestrzennych udostępnianych przez biblioteki *GLU* i *GLUT*.
Użytkownik za pomocą klawiatury powinien mieć możliwość wprowadzania zmian następujących parametrów:
1. Prędkości obrotu śmigieł zakresie [0–1] stopni/klatkę animacji z krokiem 0.25:
   - w kierunku CW;
   - w kierunku CCW;



W programie uwzględnić możliwość interakcyjnej zmiany położenia obserwatora poprzez podanie następujących parametrów:
1. Odległości obserwatora od obiektu,
2. Orientacji obserwatora w zakresie [0, 360] stopni względem osi 0X, 0Y i 0Z

UWAGA: Obserwator jest zawsze zwrócony przodem w kierunku obiektu.

2. Cel ćwiczenia i sposób rozwiązania

Laboratorium 3

Cel ćwiczenia

Celami ćwiczenia było:
- dodanie możliwości zmiany położenia obserwatora,
- przesunięcie projektu „robot” do innego położenia,
- stworzenie wieży zgodnie z zadaniem 1, a następnie przesunięcie jej,
- stworzenie, rozbudowanie i animacja układu planetarnego zgodnie z zadaniem 2.

Zmiana odległości i orientacji obserwatora

Na początku programu deklarujemy sobie zmienne odpowiedzialne za przechowanie położenia obserwatora względem osi 0X, 0Y, 0Z, zmienne które będą ograniczały odległość oraz przechowującą obecną odległość obserwatora.

```
GLfloat rotObsZ = 0.0;
GLfloat rotObsY = 40.0;
GLfloat rotObsX = 40.0;
GLfloat min_distance = -100.0;
GLfloat max_distance = -300.0;
GLfloat distance = -150.0;
```

Następnie implementujemy funkcje odpowiedzialne za zmianę położenia obserwatora, gdy zostanie wciśnięty konkretny przycisk.

Kod programu

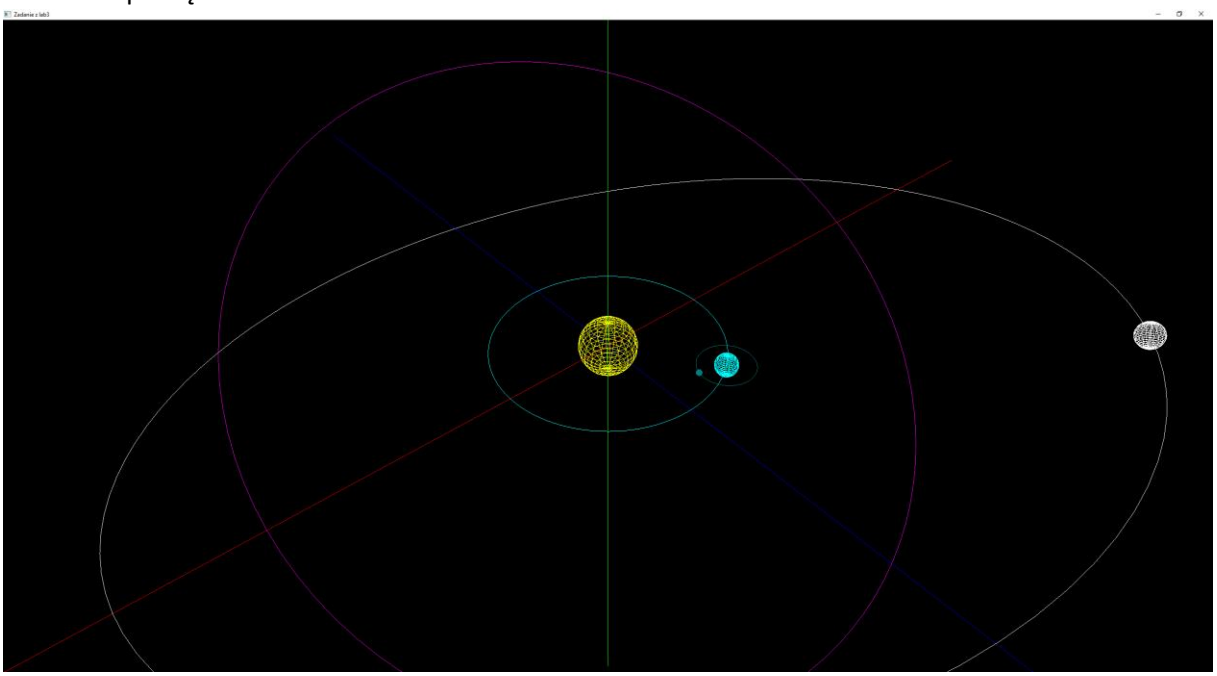
```
// Funkcja klawiszy specjalnych
void ObslugaKlawiszySpecjalnych(int klawisz, int x, int y)
{
  switch (klawisz)
  {
  case GLUT_KEY_UP:
    rotObsX = rotObsX + 1.0;
    break;

  case GLUT_KEY_DOWN:
    rotObsX = rotObsX - 1.0;
    break;

  case GLUT_KEY_LEFT:
    rotObsY = rotObsY - 1.0;
    break;

  case GLUT_KEY_RIGHT:
    rotObsY = rotObsY + 1.0;
    break;

  case GLUT_KEY_HOME:
    rotObsZ = rotObsZ + 1.0;
```

```
      break;

   case GLUT_KEY_END:
      rotObsZ = rotObsZ - 1.0;
      break;
   }
}

// Funkcja obslugi klawiatury
void ObslugaKlawiatury(unsigned char klawisz, int x, int y)
{

   switch (klawisz)
   {
   case '2':
      rotRamienia1 = (rotRamienia1 < 90.0) ? rotRamienia1 + 1.0 : rotRamienia1;
      break;

   case '@':
      rotRamienia1 = (rotRamienia1 > 0.0) ? rotRamienia1 - 1.0 : rotRamienia1;
      break;

   case '3':
      rotRamienia2 = (rotRamienia2 < 0.0) ? rotRamienia2 + 1.0 : rotRamienia2;
      break;

   case '#':
      rotRamienia2 = (rotRamienia2 > -90.0) ? rotRamienia2 - 1.0 : rotRamienia2;
      break;

   case '4':
      rotGlowicy = (rotGlowicy < 360.0) ? rotGlowicy + 1.0 : rotGlowicy;
      break;

   case '$':
      rotGlowicy = (rotGlowicy > 0.0) ? rotGlowicy - 1.0 : rotGlowicy;
      break;

   case '5':
      rozUchwytow = (rozUchwytow < 1.5) ? rozUchwytow + 0.1 : rozUchwytow;
      break;

   case '%':
      rozUchwytow = (rozUchwytow > 0.5) ? rozUchwytow - 0.1 : rozUchwytow;
      break;

   case '-':
      if (distance - 1 < max_distance) break;
```

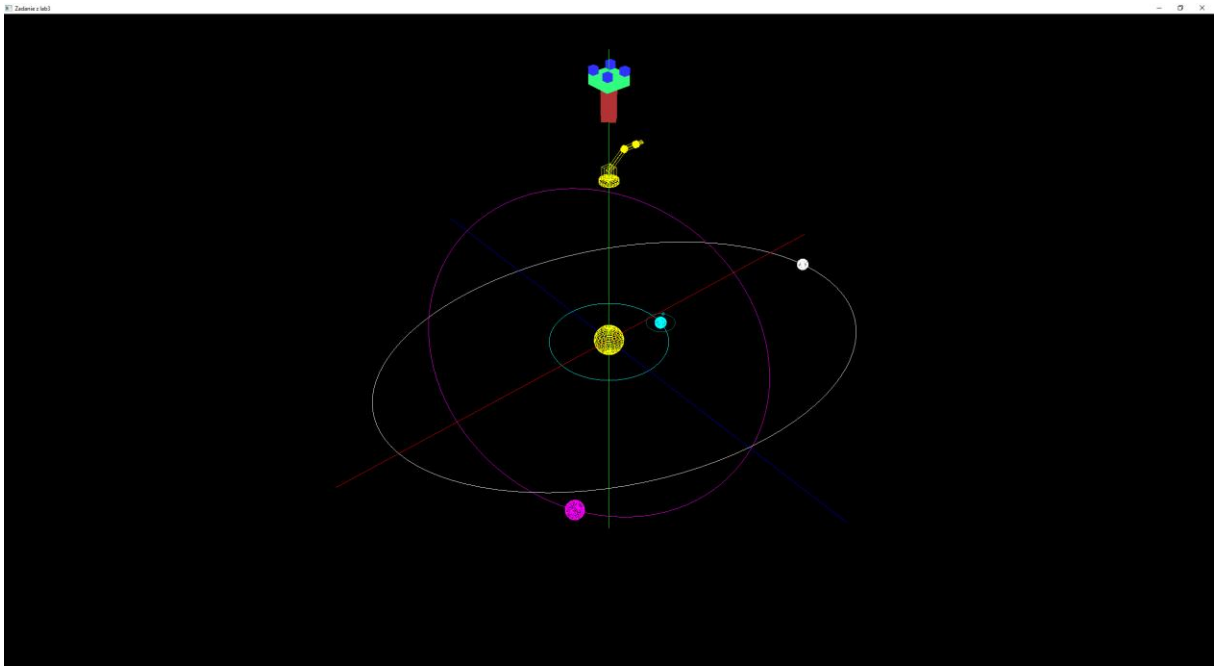
```
        else distance--;
        break;

    case '+':
        if (distance + 1 > min_distance) break;
        else distance++;
        break;
    }

    if (klawisz == 27)
        exit(0);
}
```

Efekt

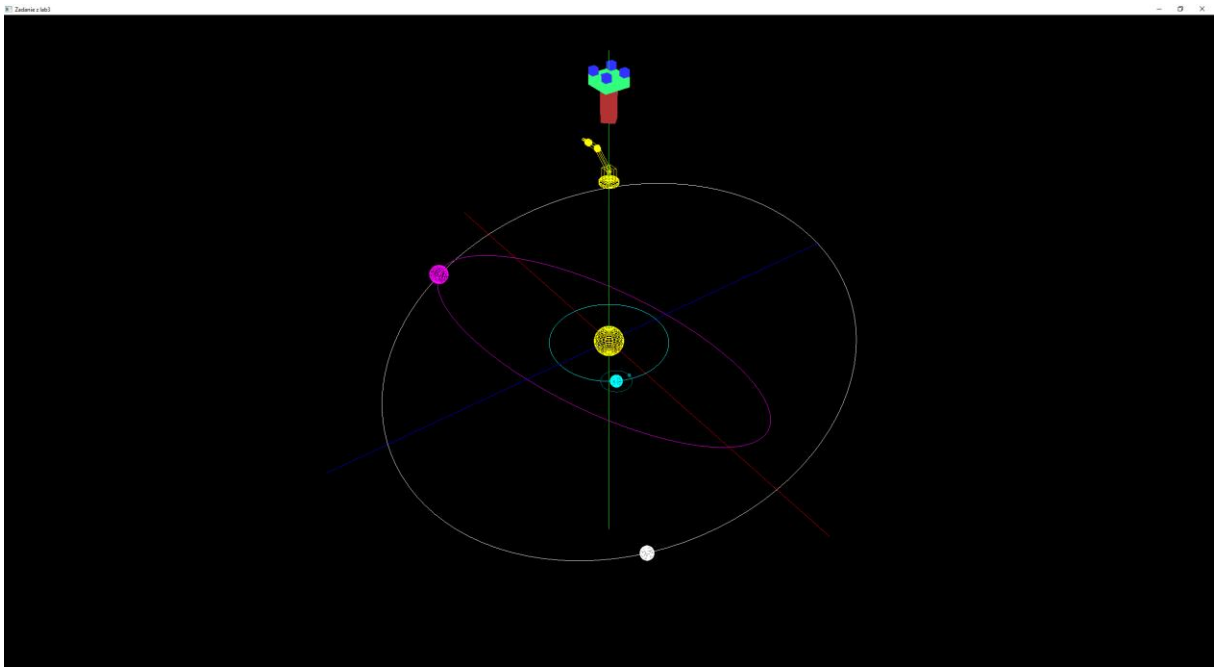
Położenie początkowe:



Maksymalne oddalenie:



Zmiana orientacji względem osi 0X:



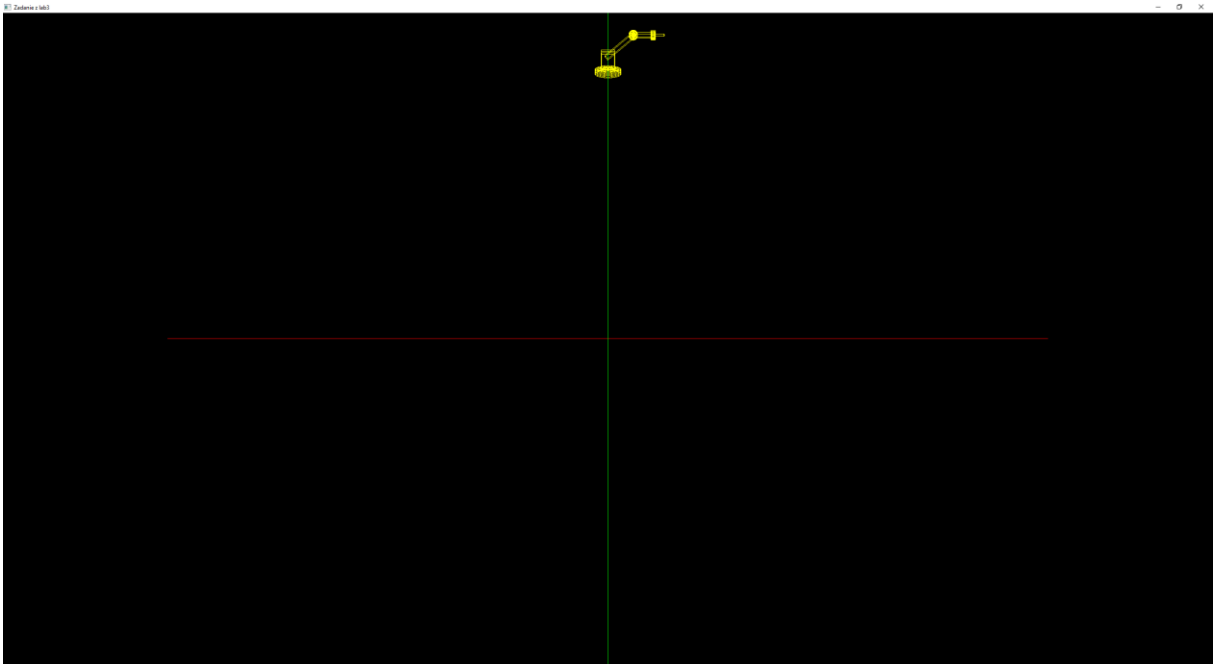
Zmiana pozycji projektu „robot”

W celu przesunięcia projektu „robot” na współrzędne (0, 60, 0), należy zmodyfikować funkcję RysujRamieRobota() dodając w niej glTranslate(0.0, 60.0, 0.0), która przesunie tworzony obiekt wzdłuż osi 0Y o 60.

Kod programu

```
void RysujRamieRobota(GLfloat obrotPodstawy, GLfloat obrotRamienia1,
   GLfloat obrotRamienia2, GLfloat obrotGlowicy,
   GLfloat rozstawUchwytow)
{
   // Poczatek tworzenia ukladu wspolrzednych
   glBegin(GL_LINES);

   // Os X
   glColor3f(1.0, 0.0, 0.0);
   glVertex3f(-100.0, 0.0, 0.0);
   glVertex3f(100.0, 0.0, 0.0);

   // Os Y
   glColor3f(0.0, 1.0, 0.0);
   glVertex3f(0.0, -100.0, 0.0);
   glVertex3f(0.0, 100.0, 0.0);

   // Os Z
   glColor3f(0.0, 0.0, 1.0);
   glVertex3f(0.0, 0.0, -100.0);
   glVertex3f(0.0, 0.0, 100.0);

   // Koniec tworzenia ukladu wspolrzednych
   glEnd();

   glColor3f(1.0, 1.0, 0);

   // Przygotowanie stosu macierzy modelowania
   glPushMatrix();
   // Przesuniecie robota
   glTranslatef(0.0, 60.0, 0.0);
   // Rysowanie podstawy ramienia (cylinder bez dolnej podstawy)
   glPushMatrix();
   // - sciany boczne
   glRotatef(-90.0, 1, 0, 0);
   gluCylinder(podstawaSciany, 3.0, 3.0, 1.0, 20, 4);
...
```

Efekt



Wieża

Przy tworzeniu wieży korzystamy z obiektów glutSolidCube(), czyli wypełnionych prostopadłościanów. Do zmiany koloru obiektu wykorzystujemy funkcję glColor3f(r, g, b), w której argumenty przyjmują wartości od 0.0 do 10.0 i determinują zawartość każdego z trzech kolorów RGB. Do zmiany skali glScalef(0X, 0Y, 0Z), dzięki tej funkcji możemy zmienić wymiary prostopadłościanu względem trzech osi. Do obrócenia glRoratef(alpha, 0X, 0Y, 0Z), ta funkcja obraca obiekt o kąt alfa tyle razy ile zostało podane w argumentach względem danej osi. Na koniec konieczne będzie przesunięcie wierzy na współrzędne (0, 80, 0), ponownie wykorzystana zostaje do tego funkcja glTranslatef(), została ona pogrubiona w kodzie programu.

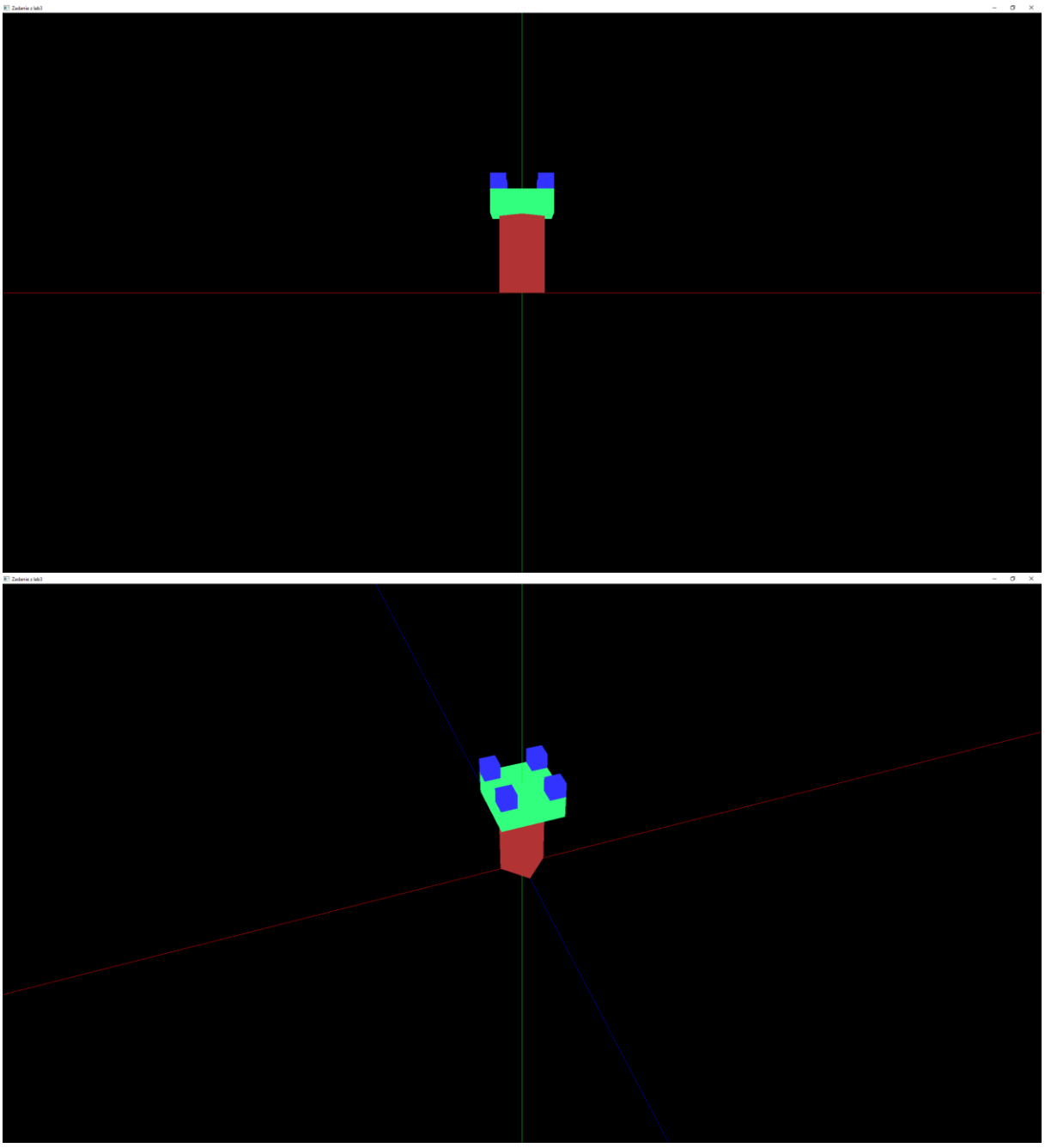
Kod programu

```
// Rysowanie wieży
void Wieza()
{
   // Przygotowanie stosu macierzy modelowania
   glPushMatrix();
   // Ustawienie wiezy w położeniu
   glTranslatef(0.0, 80.0, 0.0);
   glColor3f(0.7, 0.2, 0.2);

   // Modelowanie podstawy wieży
   glPushMatrix();
   glTranslatef(0.0, 5.0, 0.0);
   glRotatef(45.0, 0, 1, 0);
   glScalef(4.2, 10.0, 4.2);
   glutSolidCube(1);
   glPopMatrix();

   // Modelowanie nadbudówki
   glColor3f(0.2, 1.0, 0.5);
   glPushMatrix();
   glTranslatef(0.0, 11.5, 0.0);
   glScalef(8.0, 3.0, 8.0);
   glutSolidCube(1);
   glPopMatrix();

   glColor3f(0.2, 0.2, 1.0);

   // Modelowanie wieżyczek
   glPushMatrix();
   glScalef(2.0, 2.0, 2.0);
   glTranslatef(1.5, 7.0, -1.5);
   glutSolidCube(1);
   glTranslatef(-3.0, 0.0, 0.0);
   glutSolidCube(1);
   glTranslatef(0.0, 0.0, 3.0);
```

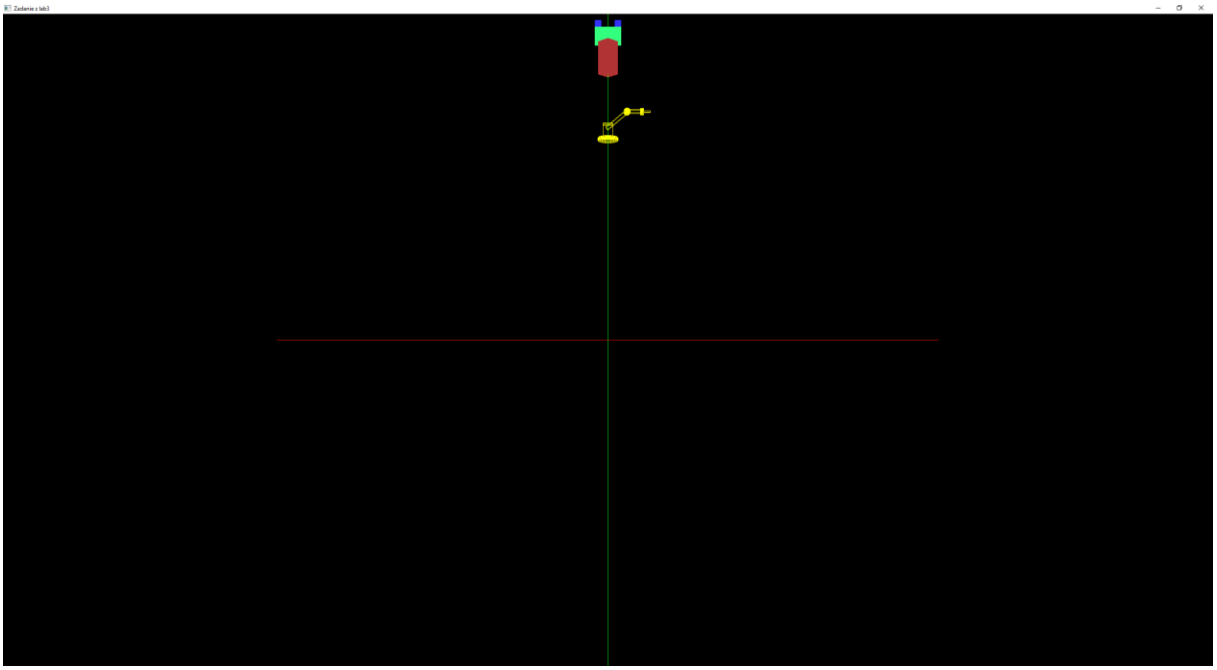
```
    glutSolidCube(1);
    glTranslatef(3.0, 0.0, 0.0);
    glutSolidCube(1);
    glPopMatrix();
    glPopMatrix();
}
```

Efekt

Utworzona wieża:



Po przesunięciu:



Układ planet

Przy tworzeniu planety korzystamy z obiektów glutWireSphere(r, p, rw), czyli sfery, złożonej z linii, o promieniu r, p południkach i rw równikach. Do zmiany koloru obiektu wykorzystujemy funkcję glColor3f(r, g, b), w której argumenty przyjmują wartości od 0.0 do 10.0 i determinują zawartość każdego z trzech kolorów RGB. Do obrócenia glRoratef(alpha, 0X, 0Y, 0Z), ta funkcja obraca obiekt o kąt alfa tyle razy ile zostało podane w argumentach względem danej osi. Do tworzenia orbit wykorzystujemy funkcje gluDisk(orbita, pw, pz, p, rw) gdzie:
orbita – obiekt typu quadrics (gluNewQuadric()),
pw – promień wewnętrzny,
pz – promień zewnętrzny,
p – liczba południków,
rw – liczba równików.

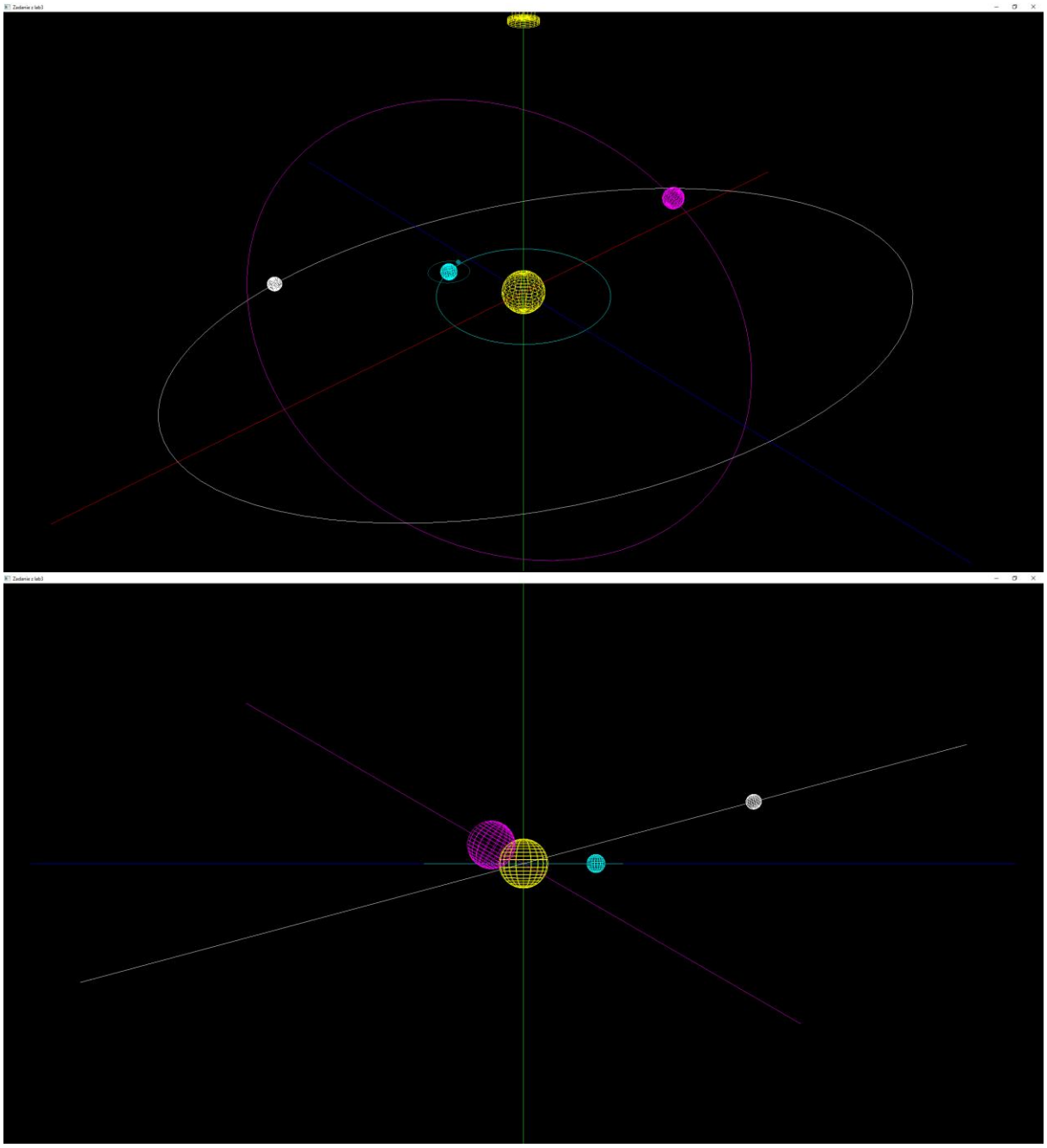
Kod programu

```
GLUquadricObj* orbita;
GLfloat rotacjaZiemia = 0;
GLfloat rotacjaKsiezyc = 0;
GLfloat rotacjaPl1 = 0;
GLfloat rotacjaPl2 = 0;

void Uklad_Planet() {
   //Zadanie 3
   glPushMatrix();
   // Słońce
   glColor3f(1.0, 1.0, 0.0);
   glRotatef(90, 1, 0, 0);
   glutWireSphere(5.0, 20.0, 20.0);
   orbita = gluNewQuadric();
   gluQuadricDrawStyle(orbita, GLU_LINE);

   //orbita Ziemi
   glColor3f(0.0, 1.0, 1.0);
   gluDisk(orbita, 20, 20, 100, 100);

   // Zadanie 3a
   // orbita Planety1
   glPushMatrix();
   glColor3f(1.0 , 0.0, 1.0);
   glRotatef(30, 1, 0, 0);
   gluDisk(orbita, 60, 60, 100, 100);
   glPopMatrix();

   // Zadanie 3b
   // orbita Planety2
   glPushMatrix();
   glColor3f(1.0, 1.0, 1.0);
   glRotatef(-15, 1, 0, 0);
   gluDisk(orbita, 80, 80, 100, 100);
   glPopMatrix();

   glRotatef(-90, 1, 0, 0);
```

```
// Ziemia
glPushMatrix();
glRotatef(rotacjaZiemia, 0, 1, 0);
// rotacja CW[-]
rotacjaZiemia -= 0.25;
glRotatef(90, 1, 0, 0);

glTranslatef(20, 0, 0);
glColor3f(0.0, 1.0, 1.0);
glutWireSphere(2, 20, 20);
glColor3f(0.0, 0.5, 0.5);
gluDisk(orbita, 5, 5, 100, 100);

glRotatef(-90, 1, 0, 0);

//Ksiezyc
glPushMatrix();
glRotatef(rotacjaKsiezyc, 0, 1, 0);
//rotacja CCW[+]
rotacjaKsiezyc += 1;
glRotatef(90, 1, 0, 0);
glTranslatef(5, 0, 0);
glutWireSphere(0.5, 20, 20);
glRotatef(-90, 1, 0, 0);
glPopMatrix();

glPopMatrix();


//Zadanie 3a
//Planeta1
glPushMatrix();
glRotatef(30, 1, 0, 0);
glRotatef(rotacjaPl1, 0, 1, 0);
//rotacja CCW[+]
rotacjaPl1 += 0.15;
glRotatef(90, 1, 0, 0);
glTranslatef(60, 0, 0);
glColor3f(1.0, 0.0, 1.0);
glutWireSphere(3, 20, 20);

glRotatef(-90, 1, 0, 0);
glPopMatrix();


// Zadanie 3b
//Planeta2
glPushMatrix();
glRotatef(-15, 1, 0, 0);
glRotatef(rotacjaPl2, 0, 1, 0);
//rotacja CW[-]
rotacjaPl2 -= 0.25;
```

```
    glRotatef(90, 1, 0, 0);

    glTranslatef(80, 0, 0);
    glColor3f(1.0, 1.0, 1.0);
    glutWireSphere(2, 20, 20);

    glRotatef(90, 1, 0, 0);
    glPopMatrix();

    glPopMatrix();
}
```

Efekt



Zadanie indywidualne

Cel zadania

Głównym celem zadania było stworzenie obiektu przypominającego wiatrak, używając prymitywów przestrzennych z biblioteki GLU i GLUT, a następnie stworzenie animacji skrzydeł tego obiektu. Dodatkowo należało dodać możliwość zmiany pozycji obserwatora.

Zmiana odległości i orientacji obserwatora

Została dodana podczas laboratorium 3.

Tworzenie i animacja wiatraka

Do tworzenia wiatraka wykorzystałem funkcję używane podczas laboratorium 3 takich jak: gluDisk(), gluWireCube(), glTranslatef(), glRotatef(), itd. Dodatkowo użyłem funkcji gluCylinder() do tworzenia cylindrów.
W celu modyfikacji prędkości obracania wiatraka należało zmodyfikować funkcje ObslugaKlawiszySpecjalnych() dodając do switch() następujące warunki:

```
case GLUT_KEY_PAGE_UP:
   speed = speed < 1 ? speed + 0.25 : speed;
   break;

case GLUT_KEY_PAGE_DOWN:
   speed = speed > -1 ? speed - 0.25 : speed;
   break;
```

Przy wciśnięciu klawisza PageDown płaty wiatraka zaczynają kręcić się w kierunku zgodnym z ruchem wskazówek zegara, a po wciśnięciu klawisza PageUp w kierunku przeciwnym. Kilkukrotne wciskanie jednego klawisza przyśpiesza obrót w daną stronę, a wciśnięcie drugiego spowalnia lub zatrzymuje.

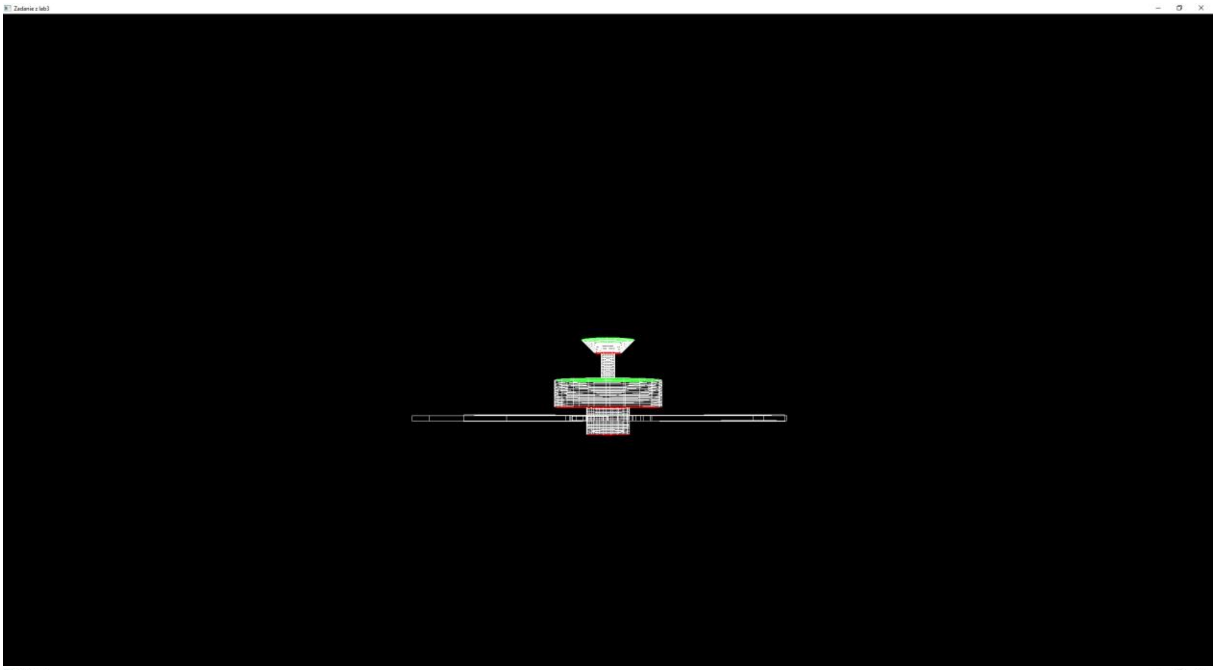
Kod programu

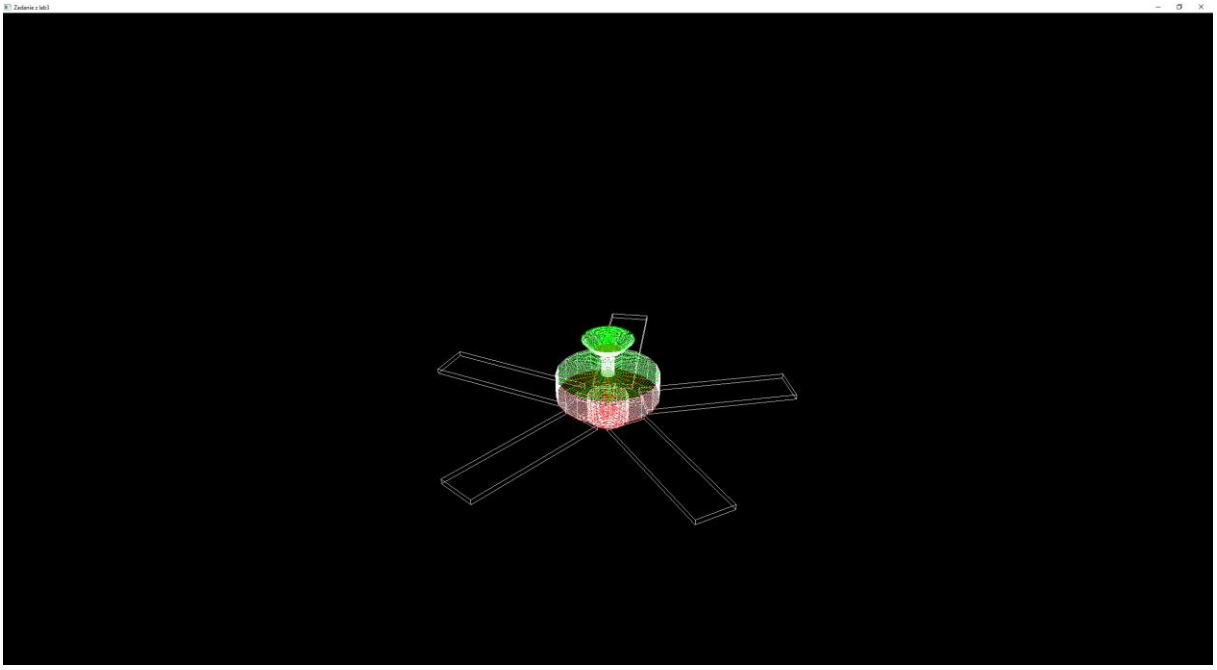
```
GLUquadricObj* cyliderWiatraka;
GLUquadricObj* dyskWiatraka;
GLfloat speed = 0.0;
GLfloat angle = 0.0;

void wiatrak() {
   angle += speed;
   glPushMatrix();
   glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
   dyskWiatraka = gluNewQuadric();
   gluQuadricDrawStyle(dyskWiatraka, GLU_LINE);
   cyliderWiatraka = gluNewQuadric();
   gluQuadricDrawStyle(cyliderWiatraka, GLU_LINE);

   glColor3f(1.0, 1.0, 1.0);
   glScalef(0.3, 0.3, 0.3);

   glPushMatrix();
   glRotatef(90, 1, 0, 0);
```

```
gluCylinder(cyliderWiatraka, 10, 5, 5, 20, 20);

glColor3f(0.0, 1.0, 0.0);
gluDisk(dyskWiatraka, 0, 10, 20, 20);
glTranslatef(0, 0, 5);

glColor3f(1.0, 0.0, 0.0);
gluDisk(dyskWiatraka, 0, 5, 20, 20);

glColor3f(1.0, 1.0, 1.0);
gluCylinder(cyliderWiatraka, 2.5, 2.5, 10, 20, 20);
glTranslatef(0, 0, 10);

glColor3f(0.0, 1.0, 0.0);
gluDisk(dyskWiatraka, 0, 20, 20, 20);

glColor3f(1.0, 1.0, 1.0);
gluCylinder(cyliderWiatraka, 20, 20, 10, 20, 20);
glTranslatef(0, 0, 10);

glColor3f(1.0, 0.0, 0.0);
gluDisk(dyskWiatraka, 0, 20, 20, 20);
glColor3f(1.0, 1.0, 1.0);
glPopMatrix();

glTranslatef(0, -25, 0);
glRotatef(angle, 0, 1, 0);

glPushMatrix();
glRotatef(90, 1, 0, 0);
gluCylinder(cyliderWiatraka, 8, 8, 10, 20, 20);
glTranslatef(0, 0, 10);
glColor3f(1.0, 0.0, 0.0);
gluDisk(dyskWiatraka, 0, 8, 20, 20);
glColor3f(1.0, 1.0, 1.0);
glPopMatrix();
for (int i = 0; i < 5; i++) {
   glPushMatrix();
   glRotatef(i * 72, 0, 1, 0);
   glTranslatef(0, -4, 11);
   // łącznik
   glPushMatrix();
   glScalef(2, 2, 6);
   glutWireCube(1);
   glPopMatrix();
   // płat
   glPushMatrix();
   glTranslatef(0, 0, 33);
   glScalef(16, 2, 60);
   glutWireCube(1);
   glPopMatrix();
   glPopMatrix();
}
```

```
    glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
    glPopMatrix();
}
```

Efekt





3. Podsumowanie

Wszystkie postawione przede mną zadania zostały pomyślnie zrealizowane.
W czasie wykonywania poleceń wiele nauczyłem się m. in. tego, że przekształcenia geometryczne to zmiany położenia, rozmiaru i orientacji obiektów w przestrzeni 3D. Można wyobrazić sobie przekształcenia geometryczne jako operacje matematyczne, takie jak translacja, obrót i skalowanie wykonywane na obiektach w programie. Zrozumienie tego przyśpieszyło moją pracę oraz uświadomiło, że praca w przestrzeni 3D nie jest taka skomplikowana.
Zdobyta wiedza i umiejętności z pewnością przydadzą mi się w przyszłych projektach np. podczas tworzenia prostych gier lub animacji.