Politechnika Wrocławska

Sprawozdanie 6

Ćwiczenie 6.Układ słoneczny

Krzysztof Zalewa

7.1.2025

Spis treści

1 Wstęp teoretyczny **2**
1.1 Układ słoneczny 2
1.2 Wykorzystane tekstury 2

2 Zadanie laboratoryjne **7**
2.1 Treść zadania 7
2.2 Opis działania programu 8
2.3 Kod programu 8

3 Wnioski **17**

4 Źródła **17**

1 Wstęp teoretyczny

1.1 Układ słoneczny

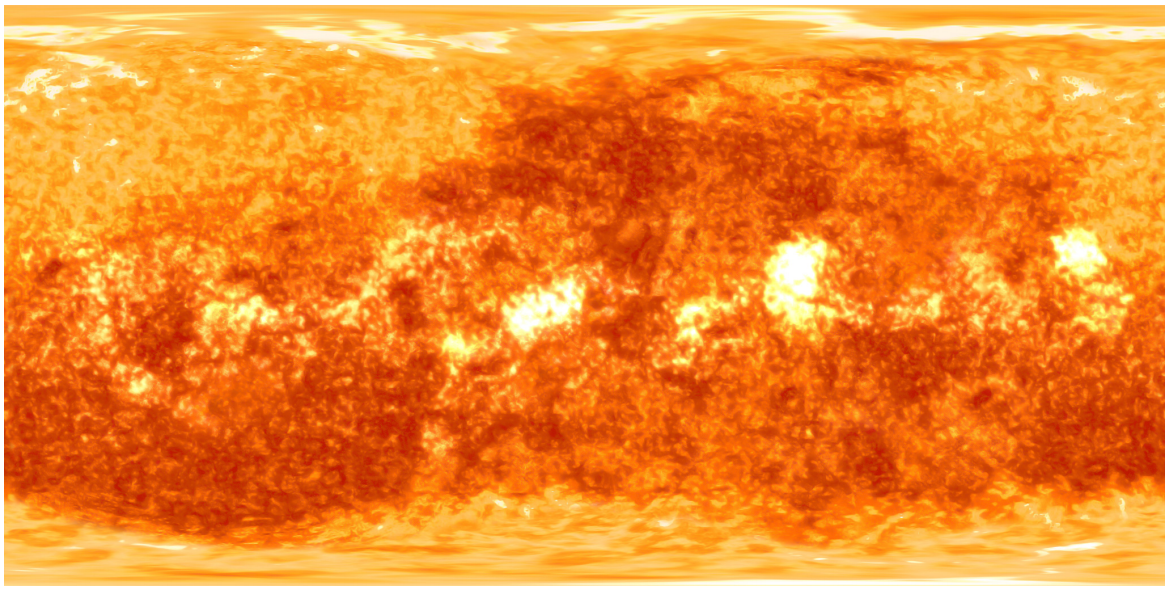
Wielkości planet zostały dobrane eksperymnetacyjnie ”tak żeby wyglądały dobrze”. Wyjątkiem są wartości pochylenia osiowego planet (dokładność do 1 miejsca po przecinku). Wysoki stopień

Planeta	Moja wartość	Wartość rzeczywista
Merkury	0.03	0.034
Wenus	177.4*	177.36
Ziemia	23.5	23.44
Mars	25.2	25.19
Jowisz	3.1	3.13
Saturn	26.7	26.73
Uran	97.8	97.77
Neptun	28.3	28.32
Pluton	122.5*	122.53

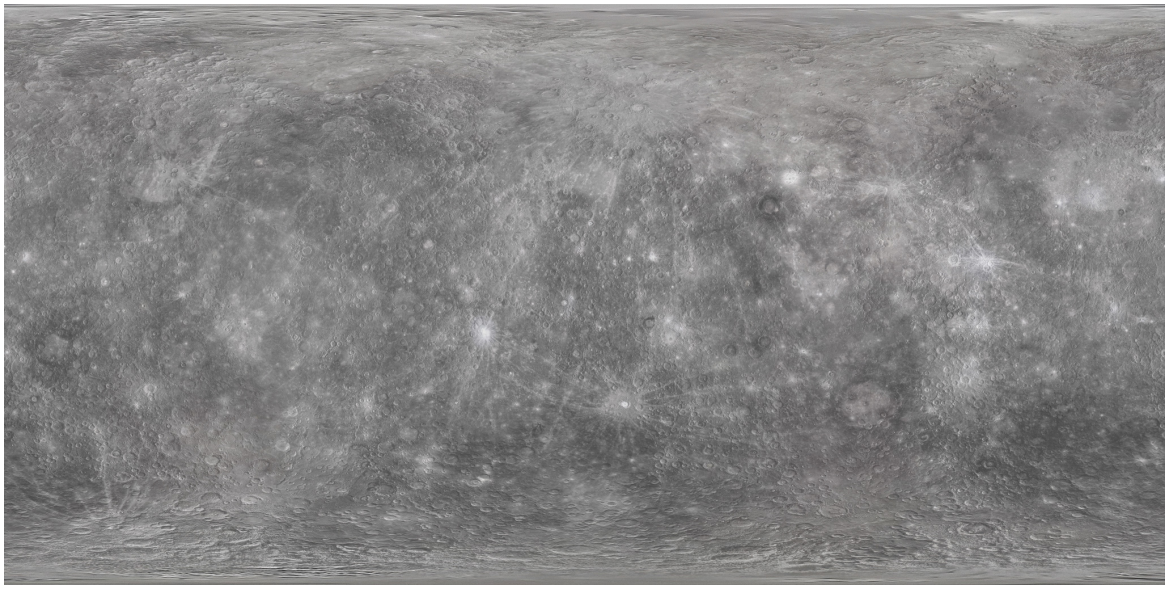
Table 1: Pochylenie osiowe planet

nachylenia sprawia że Wenus i Pluton obracają się zgodnie z ruchem wskazuwek zegara. Pozostałe planety obracają się w przeciwną stronę.

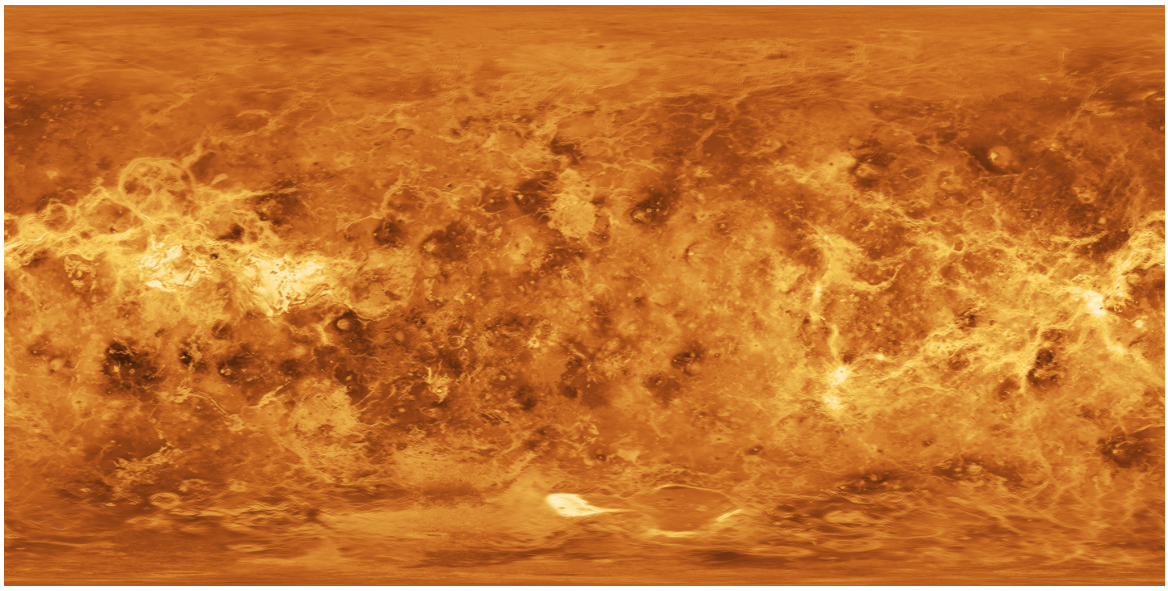
1.2 Wykorzystane tekstury



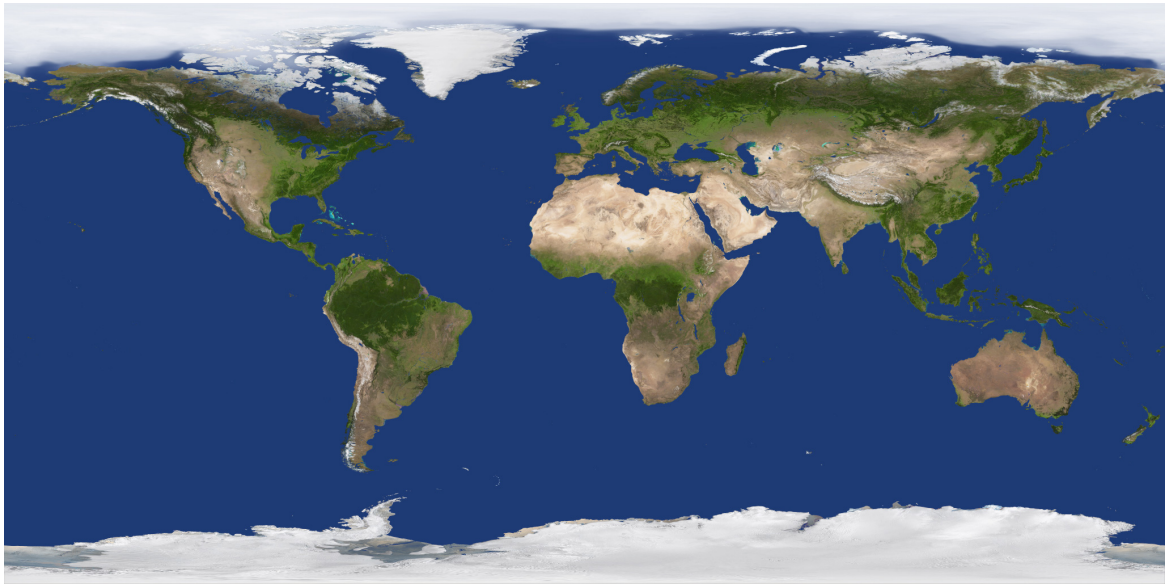
Rysunek 1: Tekstura słońca[1]



Rysunek 2: Tekstura merkurego[1]

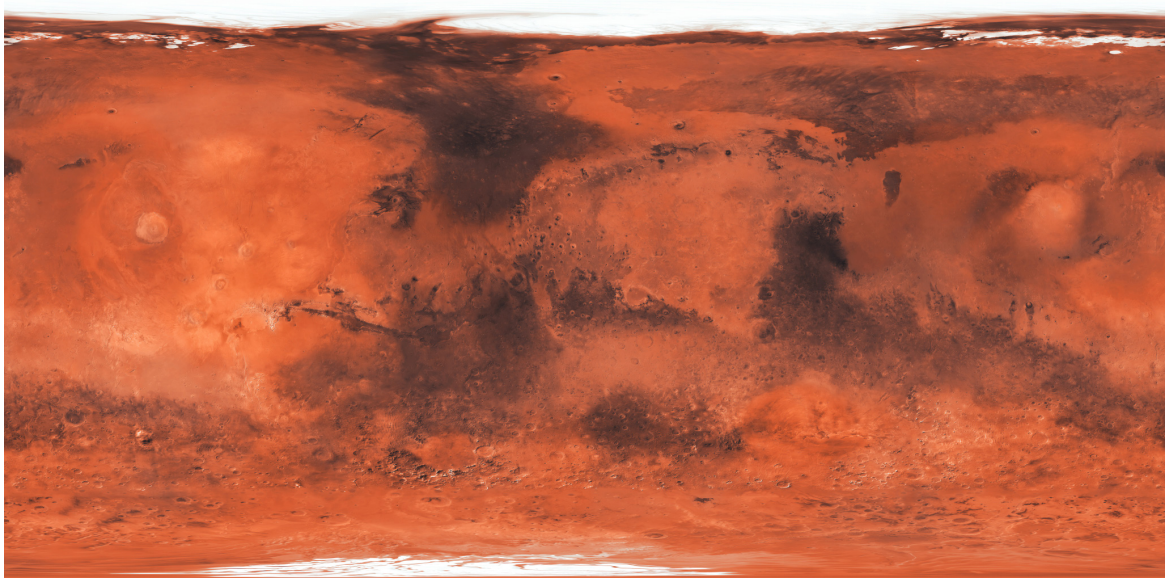


Rysunek 3: Tekstura wenus[1]

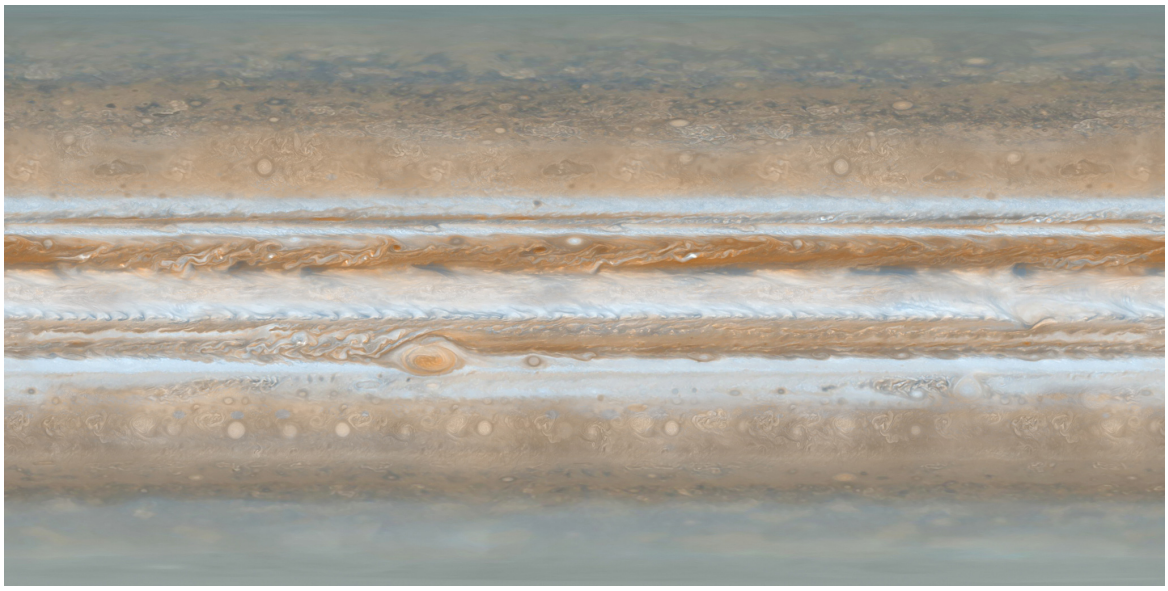


Rysunek 4: Tekstura ziemi[1]

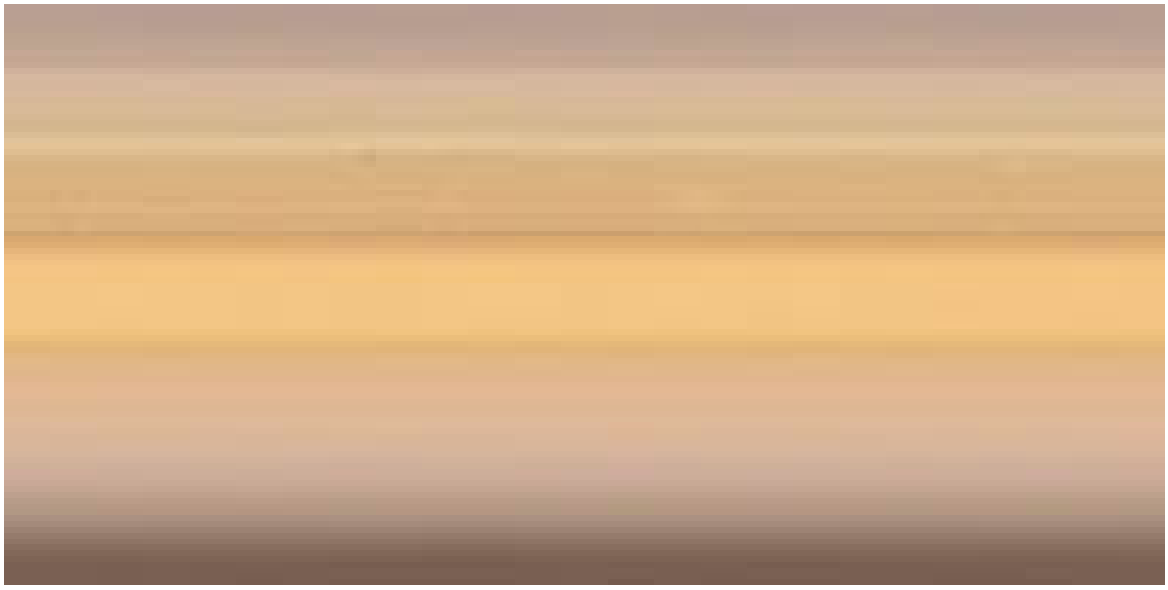
Rysunek 5: Tekstura księżyca[1]



Rysunek 6: Tekstura marsa[1]

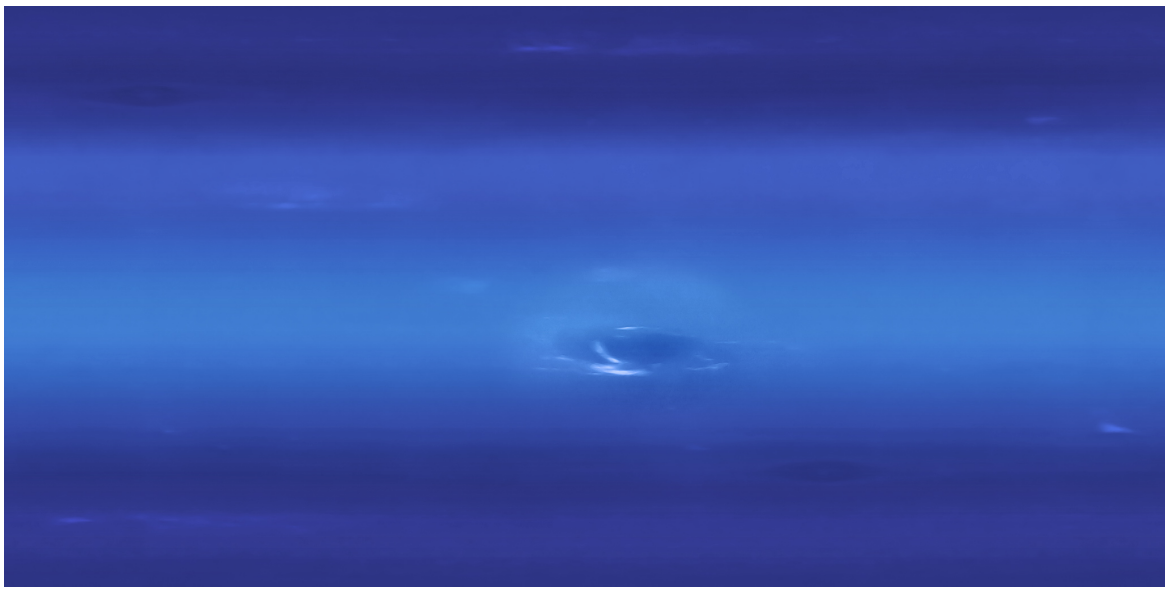


Rysunek 7: Tekstura jowisza[1]



Rysunek 8: Tekstura saturna[2]

Rysunek 9: Tekstura urana[1]



Rysunek 10: Tekstura neptuna[1]

Rysunek 11: Tekstura pluta[2]

2 Zadanie laboratoryjne

2.1 Treść zadania

W ramach zadania laboratoryjnego należało napisać program symulujący układ słoneczny. Symulacja miała składać się z planet wirujących wokół własnej osi Y i jednocześnie orbitujących wokół słońca. Orbita powinna być eliptyczna ze słońcem w jednym z ognisk. Ruch po orbicie powinien uwzględniać 3 prawo Keplera. Słońce powinno być źródłem światła.

2.2 Opis działania programu

Zgodnie z treścią zadania program rysuje 12 obiektów (10 planet,słońce i Księżyc).Wszystkie planety obracają się wokół własnej osi i orbitują wokół słońca. **F1** - Zmiana kamery
F2 - Poprzednia planeta
F3 - Następna planeta
F4 - Wypisanie listy planet
ESC - Wyjście z programu
Ruch myszy w osi X - Obrót kamery w osi X
Ruch myszy w osi Y - Obrót kamery w osi Y
Scroll up - Przybliżenie obiektu
Scroll down - Oddalenie obiektu

2.3 Kod programu

.

```
#include <windows.h>
#include <iostream>
#include <GL/glu.h>
#include <vector>
#define FREEGLUT_STATIC
#include <GL/freeglut.h>
#define STB_IMAGE_IMPLEMENTATION
#include "stb_image.h"
```

```
9  //My .h files
10 #include "Planet.h"
11 #include "Sun.h"
12
13 using namespace std;
14 //Constants
15 const int TEXT_HEIGHT = 13;
16 const int PLANET_NUM = 9;
17 const int ALL = 11;
18 const int CAMERA_NUM = 2;
19 //Global variables
20 HWND consoleWindow;
21 HWND glutWindow;
22 GLuint textureIDs[ALL];
23 void *font = GLUT_BITMAP_8_BY_13;
24 int currentPlanet = 3;
25 Planet planets[PLANET_NUM];
26 string planetNames[PLANET_NUM+1] = {"Mercury","Venus","Earth","Moon","Mars","Jupit
   ↪  er","Saturn","Uranus","Neptune","Pluto"};
27 const char* fileNames[ALL] = {"textures\\2k_sun.tga","textures\\2k_mercury.tga","t
   ↪  extures\\2k_venus_surface.tga","textures\\2k_earth_daymap.tga",
28     "textures\\2k_moon.tga","textures\\2k_mars.tga","textures\\2k_jupiter.tga","te
       ↪  xtures\\saturnmapthumb.tga","textures\\2k_uranus.tga",
29     "textures\\2k_neptune.tga","textures\\plutomapthumb.tga"};
30 float planetSizes[PLANET_NUM] = {2,6.5,7,6.5,14,10,8,8,5};
31 float planetDistances[PLANET_NUM] =
   ↪  {15.0,25.0,35,50.0f,70.0f,80.0f,90.0f,100.0f,110.0f};
32 float planetAxialTilts[PLANET_NUM] =
   ↪  {0.03f,177.4f,23.5f,25.2f,3.1f,26.7f,97.8f,28.3f,122.5f};
33 float xAxis[PLANET_NUM] = {1.0f, 1.0f, 1.0f, 1.0f, 1.0f, 1.0f, 1.0f, 1.0f, 1.0f};
34 float yAxis[PLANET_NUM] = {1.3f, 1.3f, 1.2f, 1.21f, 1.25f, 1.15f, 1.25f, 1.2f,
   ↪  1.4f};
35 float planetOrbitalTilts[PLANET_NUM] =
   ↪  {7.0,3.39,0.0,1.85,1.31,2.49,0.77,1.77,17.16} ;
36 int currentCamera = 0;
37 string cameraNames[CAMERA_NUM] = {"Wolna","Na planecie"};
38 float pix2angle = 360.0/800,theta = 0.0f,phi = 0.0f;
39 int radius = 35,lastX = 0,lastY = 0;
40 float cameraRotationX = radius * cosf((theta*(M_PI/180))) * cosf((phi*(M_PI/180)));
41 float cameraRotationY = radius * sinf((phi*(M_PI/180)));
42 float cameraRotationZ = radius * sinf((theta*(M_PI/180))) * cosf((phi*(M_PI/180)));
43 float cameraX = 35,cameraY = 35,cameraZ = 35;
44 Sun sun;
45 //Window controls
46 void toggleFocusToConsole() {
47     ShowWindow(glutWindow, SW_HIDE);
48     ShowWindow(consoleWindow, SW_SHOWNORMAL);
49     SetForegroundWindow(consoleWindow);
50 }
51 void toggleFocusToGLUT() {
52     ShowWindow(consoleWindow, SW_HIDE);
53     ShowWindow(glutWindow, SW_SHOWNORMAL);
54     SetForegroundWindow(glutWindow);
55 }
56 //UI
57 void drawString(const char *str, int x, int y, float color[4], void *font){
```

```
58      glPushAttrib(GL_LIGHTING_BIT | GL_CURRENT_BIT);
59      glDisable(GL_LIGHTING);
60      glDisable(GL_TEXTURE_2D);
61      glDepthFunc(GL_ALWAYS);
62      glColor4fv(color);
63      glRasterPos2i(x, y);
64      while(*str){
65          glutBitmapCharacter(font, *str);
66          ++str;
67      }
68      glEnable(GL_TEXTURE_2D);
69      glEnable(GL_LIGHTING);
70      glDepthFunc(GL_LEQUAL);
71      glPopAttrib();
72  }
73  void showInfo(){
74      glPushMatrix();
75      glLoadIdentity();
76      glMatrixMode(GL_PROJECTION);
77      glPushMatrix();
78      glLoadIdentity();
79      gluOrtho2D(0, 800, 0, 800);
80      float color[4] = {1, 1, 1, 1};
81      string s = "Kamera: "+cameraNames[currentCamera];
82      drawString(s.c_str(),2,800-TEXT_HEIGHT,color,font);
83      s = "F1 - Zmien kamere";
84      drawString(s.c_str(),2,800-(2*TEXT_HEIGHT),color,font);
85      s = "Planeta: "+planetNames[currentPlanet];
86      drawString(s.c_str(),2,800-(3*TEXT_HEIGHT),color,font);
87      s = "F2 - Poprzednia planeta || F3 Nastepna planeta || F4 - Lista Planet";
88      drawString(s.c_str(),2,800-(4*TEXT_HEIGHT),color,font);
89      s = "ESC - Wyjdz z programu";
90      drawString(s.c_str(),2,800-(5*TEXT_HEIGHT),color,font);
91      glPopMatrix();
92      glMatrixMode(GL_MODELVIEW);
93      glPopMatrix();
94  }
95  //Keyboard control
96  void normalKey(u_char key,int x,int y){
97      float moveSpeed = 10.0f;
98      switch (key){
99      case 'W':
100     case 'w':
101         if(currentCamera == 0){
102             cameraY -= 10.0f;
103         }
104         break;
105     case 'A':
106     case 'a':
107         if(currentCamera == 0){
108             cameraX -= 10.0f;
109         }
110         break;
111     case 'S':
112     case 's':
113         if(currentCamera == 0){
```

```
114                 cameraY += 10.0f;
115             }
116             break;
117         case 'D':
118         case 'd':
119             if(currentCamera == 0){
120                 cameraX += 10.0f;
121             }
122             break;
123         case 27:
124             exit(0);
125             break;
126         default:
127             break;
128         }
129     }
130     void specialKey(int key,int x,int y){
131         switch (key){
132         //F1 - Change camera
133         case GLUT_KEY_F1:
134             if(currentCamera==0){
135                 currentCamera = 1;
136             }else{
137                 currentCamera = 0;
138             }
139             break;
140         //F2 - Prevoius planet
141         case GLUT_KEY_F2:
142             planets[currentPlanet].setCamera();
143             currentPlanet--;
144             planets[currentPlanet].setCamera();
145             if(currentPlanet == 0){
146                 currentPlanet = PLANET_NUM-1;
147             }
148             break;
149         //F3 - Next planet
150         case GLUT_KEY_F3:
151             planets[currentPlanet].setCamera();
152             currentPlanet++;
153             planets[currentPlanet].setCamera();
154             if(currentPlanet == PLANET_NUM){
155                 currentPlanet = 1;
156             }
157             break;
158         //F4 - Print plantets
159         case GLUT_KEY_F4:
160             for(int i=0;i<PLANET_NUM;i++){
161                 cout<<i<<". "<<planetNames[i]<<"\n";
162             }
163             toggleFocusToConsole();
164             cout<<"Nacisnij Enter zeby kontynuowac\n"<<flush;
165             cin.get();
166             cin.get();
167             toggleFocusToGLUT();
168             break;
169         default:
```

```
170             break;
171         }
172         glutPostRedisplay();
173     }
174     void mouse(int x, int y){
175         float dY = y - lastY;
176         lastY = y;
177         float dX = x - lastX;
178         lastX = x;
179         theta += dX * pix2angle;
180         phi += dY * pix2angle;
181         if (phi > 89.0f) {phi = 89.0f;}
182         if (phi < -89.0f) {phi = -89.0f;}
183         switch(currentCamera){
184             case 0:
185                 cameraRotationX = radius * cosf((theta*(M_PI/180.0f))) *
                    ↪  cosf((phi*(M_PI/180.0f)));
186                 cameraRotationY = radius * sinf((phi*(M_PI/180.0f)));
187                 cameraRotationZ = radius * sinf((theta*(M_PI/180.0f))) *
                    ↪  cosf((phi*(M_PI/180.0f)));
188                 break;
189             case 1:
190 
191                 break;
192         }
193         lastX = x;
194         lastY = y;
195         glutPostRedisplay();
196     }
197     void mouseWheel(int button, int dir, int x, int y){
198         if (dir > 0){
199             radius -= 10;
200         }else{
201             radius += 10;
202         }
203         if(radius<20){
204             radius=20;
205         }
206         glutPostRedisplay();
207     }
208     void timer(int value) {
209         glutPostRedisplay();
210         glutTimerFunc(16, timer, 0);
211     }
212     void animate(){
213         for(int i = 0;i<PLANET_NUM;i++){
214             planets[i].animateSpin();
215             planets[i].move(0.1f);
216         }
217         planets[2].animateSpinMoon();
218         planets[2].moveMoon(0.1f);
219         glutPostRedisplay();
220     }
221     void display(){
222         GLfloat lPos[] = {0,0,0,1};
223         glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

```
224     glLoadIdentity();
225     glTranslatef(-cameraX,0.0f,-cameraY);
226     if(currentCamera==0){
227         gluLookAt(cameraX + cameraRotationX,cameraY + cameraRotationY,cameraZ +
            ↪  cameraRotationZ , cameraX, cameraY, cameraZ,0,1,0);
228     }else{
229         planets[currentPlanet].setCamera();
230     }
231     glLightfv(GL_LIGHT0,GL_POSITION,lPos);
232     sun.draw(textureIDs[0]);
233     for(int i = 0;i<PLANET_NUM;i++){
234         glPushMatrix();
235         glRotatef(planetOrbitalTilts[i], 1.0f, 0.0f, 0.0f);
236         planets[i].drawOrbit();
237         planets[i].draw(textureIDs[i+1]);
238         glPopMatrix();
239     }
240     showInfo();
241     glutSwapBuffers();
242 }
243 void loadTexture(const char* fileName,GLuint texID){
244     glGenTextures(1,&textureIDs[texID]);
245     glBindTexture(GL_TEXTURE_2D, textureIDs[texID]);
246     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
247     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
248     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
249     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
250     int width,height,nrChannels;
251     unsigned char *data = stbi_load(fileName, &width, &height, &nrChannels, 0);
252     if (data){
253         glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB,
            ↪  GL_UNSIGNED_BYTE, data);
254     }else{
255         cout << "Failed to load texture! "<< fileName <<" "
            ↪  <<stbi_failure_reason()<< endl;
256         system("pause");
257         exit(1);
258     }
259     stbi_image_free(data);
260 }
261 void init(){
262     sun = Sun(GL_LIGHT0,0,16);
263     float pos = 10;
264     for(int i = 0;i<PLANET_NUM;i++){
265         if(i!=3){
266             planets[i] = Planet(i,planetSizes[i]);
267             pos += planetDistances[i];
268             planets[i].setDistance(pos);
269             planets[i].setElipse(xAxis[i],yAxis[i]);
270             planets[i].setTilt(planetAxialTilts[i]);
271         }
272     }
273     moon newMoon = {
274         25,2,0,13,planets[2].getPosition(),6.7
275     };
276     planets[2].addMoon(newMoon);
```

```
277     glEnable(GL_DEPTH_TEST); //bez tego frontalna sciana nadpisuje tylnią
278     glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
279     glMatrixMode(GL_PROJECTION);
280     gluPerspective(45,1,0.01,5000);
281     glMatrixMode(GL_MODELVIEW);
282     glShadeModel(GL_SMOOTH);
283     glEnable(GL_LIGHTING); //Włączenie oświetlenia
284     glEnable(GL_LIGHT0); //Dodanie źródła światła
285     glEnable(GL_TEXTURE_2D); //Włącza teksturowanie
286     glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
287     for(int i = 0;i<ALL;i++){
288         loadTexture(fileNames[i],i);
289     }
290     planets[2].setMoonTexture(textureIDs[4]);
291 }
292 //Na 7 stycznia
293 //TODO - 2 Kamery sterowane przy użyciu myszy
294 //TODO - Swobodna na sferze (biegunowa,azymut)
295 //TODO - Umiejscowiona na planecie
296 int main(int argc, char** argv){
297     consoleWindow = GetConsoleWindow();
298     glutInit(&argc, argv);
299     glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
300     glutInitWindowSize(800,800);
301     glutCreateWindow("Lab 6 - Uklad sloneczny");
302     glutWindow = FindWindowW(NULL,L"Lab 6 - Uklad sloneczny");
303     init();
304     glutDisplayFunc(display);
305     glutKeyboardFunc(normalKey);
306     glutSpecialFunc(specialKey);
307     glutMotionFunc(mouse);
308     glutMouseWheelFunc(mouseWheel);
309     glutIdleFunc(animate);
310     glutTimerFunc(25, timer, 0);
311     glutMainLoop();
312     return 0;
313 }
```

Fragment kodu 1: Fragment kodu z programu

.

```
1  #include "Planet.h"
2  #include <math.h>
3  #define FREEGLUT_STATIC
4  #include <GL/freeglut.h>
5  void Planet::axis(){
6      glDisable(GL_LIGHTING);
7      glBegin(GL_LINES);
8      glColor3f(1.0, 0.0, 0.0);
9      glVertex3f(-15.0, 0.0, 0.0);
10     glVertex3f(15.0, 0.0, 0.0);
11     glColor3f(0.0, 1.0, 0.0);
12     glVertex3f(0.0, -15.0, 0.0);
13     glVertex3f(0.0, 15.0, 0.0);
14     glColor3f(0.0, 0.0, 1.0);
```

```
15      glVertex3f(0.0, 0.0, -15.0);
16      glVertex3f(0.0, 0.0, 15.0);
17      glEnd();
18      glEnable(GL_LIGHTING);
19  }
20  void Planet::animateSpin(){
21      spin += planetSpeed;
22      if (spin > 360.0f) {
23          spin -= 360.0f;
24      }
25  }
26  void Planet::move(float dTime){
27      float orbitalPeriod = sqrtf(powf(distance, 3));
28      float orbitalSpeed = 2.0f * 3.1415926f / orbitalPeriod;
29      planetTheta += orbitalSpeed*dTime;
30      if (planetTheta >= 2.0f * 3.1415926f) {
31          planetTheta -= 2.0f * 3.1415926f;
32      }
33  }
34  void Planet::moveMoon(float dTime){
35      moonTheta += moonOrbitSpeed*dTime;
36      if (spin > 360.0f) {
37          spin -= 360.0f;
38      }
39  }
40  void Planet::draw(u_int texture){
41      glPushMatrix();
42      glColor3f(1.0, 1.0, 1.0);
43      float planetX = xAxis * distance * cosf(planetTheta) + ellipseCenterX;
44      float planetY = yAxis * distance * sinf(planetTheta) - ellipseCenterY;
45      glTranslatef(planetX, 0.0f, planetY);
46      if(hasMoon){
47          float x = myMoon.distance * cosf(moonTheta);
48          float y = myMoon.distance * sinf(moonTheta);
49          glPushMatrix();
50          drawMoonsOrbit();
51          glRotatef(moonTheta, 0.0f, 1.0f, 0.0f);
52          glTranslatef(x, 0.0f, y);
53          glRotatef(tilt, 1.0f, 0.0f, 0.0f);
54          glRotatef(myMoon.spin, 0.0f, 1.0f, 0.0f);
55          glEnable(GL_TEXTURE_2D);
56          glBindTexture(GL_TEXTURE_2D, moonTexture);
57          GLUquadric *quad = gluNewQuadric();
58          gluQuadricTexture(quad, GL_TRUE);
59          gluSphere(quad, myMoon.size,  myMoon.points,  myMoon.points);
60          gluDeleteQuadric(quad);
61          glDisable(GL_TEXTURE_2D);
62          glPopMatrix();
63      }
64      positon = {planetX, 0.0f, planetY};
65      glRotatef(90.0, 1.0f, 0.0f, 0.0f);
66      glRotatef(tilt, 1.0f, 0.0f, 0.0f);
67      glRotatef(spin, 0.0f, 0.0f, 1.0f);
68      // axis();
69      if(camera){
70          gluLookAt(planetX, 0.0f, planetY,0,0,0,0,1,0);
```

```
71      }
72      glEnable(GL_TEXTURE_2D);
73      glBindTexture(GL_TEXTURE_2D, texture);
74      GLUquadric *quad = gluNewQuadric();
75      gluQuadricTexture(quad, GL_TRUE);
76      gluSphere(quad,size,points,points);
77      gluDeleteQuadric(quad);
78      glDisable(GL_TEXTURE_2D);
79      glPopMatrix();
80  }
81  void Planet::drawOrbit(){
82      glPushMatrix();
83      glDisable(GL_LIGHTING);
84      glPointSize(2.0f);
85      glBegin(GL_POINTS);
86      glColor3f(1.0, 1.0, 1.0);
87      for(int i = 0;i<points *2;i++){
88          float theta = 2.0f * 3.1415926f * float(i) / float(points * 2);
89          float x = xAxis * distance * cosf(theta);
90          float y = yAxis * distance * sinf(theta);
91          glVertex3f(x + ellipseCenterX, 0.0f, y - ellipseCenterY);
92      }
93      glEnd();
94      glEnable(GL_LIGHTING);
95      glPopMatrix();
96  }
97  void Planet::drawMoonsOrbit(){
98      glPushMatrix();
99      glDisable(GL_LIGHTING);
100     glPointSize(2.0f);
101     glBegin(GL_POINTS);
102     glColor3f(1.0, 1.0, 1.0);
103     for(int i = 0;i<myMoon.points *2;i++){
104         float theta = 2.0f * 3.1415926f * float(i) / float(myMoon.points * 2);
105         float x = myMoon.distance * cosf(theta) ;
106         float y = myMoon.distance * sinf(theta) ;
107         glVertex3f(x,0.0f,y);
108     }
109     glEnd();
110     glEnable(GL_LIGHTING);
111     glPopMatrix();
112 }
113 void Planet::animateSpinMoon(){
114     myMoon.spin += moonSpeed;
115     if (myMoon.spin > 360.0f) {
116         myMoon.spin -= 360.0f;
117     }
118 }
119 void Planet::setTilt(float degree){
120     tilt = degree;
121 }
122 Planet::Planet(float newDistance,float newSize){
123     distance = newDistance;
124     size = newSize;
125 }
126 Planet::Planet() : points(50), size(1.0f), distance(0.0f) {}
```

```
127 void Planet::setDistance(float newDistance){
128     distance = newDistance;
129     positon = {distance, 0.0f, 0.0f};
130 }
131 void Planet::setElipse(float newX,float newY){
132     xAxis = newX;
133     yAxis = newY;
134     c = sqrtf(yAxis * yAxis - xAxis * xAxis);
135     ellipseCenterX = 0.0f;
136     ellipseCenterY = c* distance/3;
137 }
138 point Planet::getPosition(){
139     return positon;
140 }
141 void Planet::addMoon(moon newMoon){
142     myMoon = newMoon;
143     hasMoon = true;
144 }
145 void Planet::setCamera(){
146     camera = !camera;
147 }
148 void Planet::setMoonTexture(u_int newTexture){
149     moonTexture = newTexture;
150 }
```

Fragment kodu 2: Fragment kodu z programu

.

```
1 #include "Sun.h"
2 void Sun::initLight(){
3 
4 }
5 void Sun::draw(u_int texture){
6     glDisable(GL_LIGHTING);
7     Planet::draw(texture);
8     glEnable(GL_LIGHTING);
9 }
```

Fragment kodu 3: Fragment kodu z programu

3 Wnioski

4 Źródła

1. https://www.solarsystemscope.com/textures/
2. https://planetpixelemporium.com/pluto.html