POLITECHNIKA RZESZOWSKA
im. IGNACEGO ŁUKASIEWICZA

GRAFIKA KOMPUTEROWA - PROJEKT

DOKUMENTACJA/SPRAWOZDANIE

Wyslij na ozog.dominik@reconal.com.pl

L06

Radosław Niedbała

Wstęp

Celem projektu było zamodelowanie lądownika kosmicznego
Lądownik – część statku kosmicznego lub sondy kosmicznej, która po oddzieleniu się od statku macierzystego ląduje na planecie,księżycu, planetoidzie lub jądrze komety. Lądownik może służyć również do opuszczenia na powierzchnię ciała niebieskiego robotów lub ludzi (np. Apollo 11). Aparatura badawcza lądownika ma za zadanie dostarczenie szczegółowych danych o składzie chemicznym i właściwościach fizycznych atmosfery (o ile takowa istnieje) i powierzchni ciała niebieskiego, wykonanie zdjęć bądź filmów, badanie możliwości istnienia życia pozaziemskiego (np. lądowniki programu Viking) i inne.
Jako że w sieci znalazione lądowniki wydały mi się zbyt skomplikowane postanowiłem stworzyć statek ufo wydało mi sie to znacznie ciekawsze.
Statek następnie należało otekstorować oraz zamodelować do niego system fizyki, postanowiłem stworzyć do niego odpowiednie środowisko ponieważ tematem okazało się lądowanie na planecie (na marsie czy księżycu).

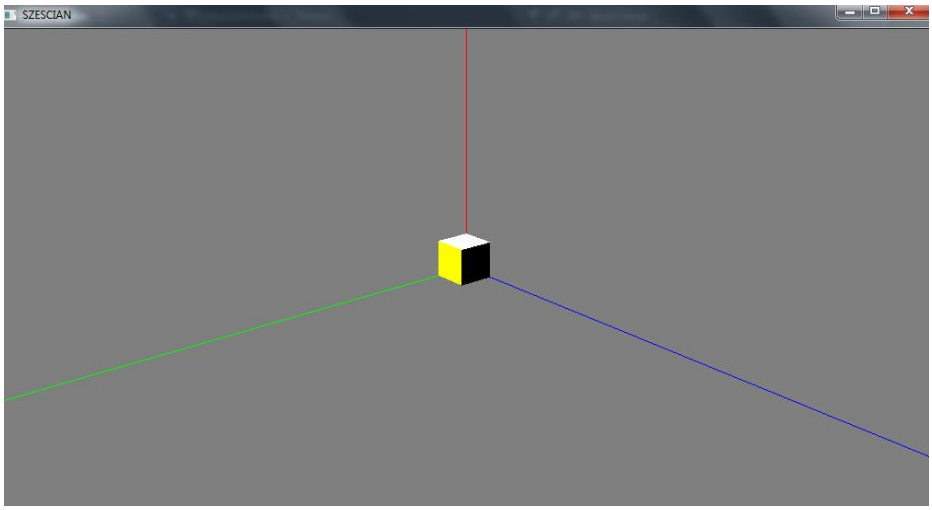
2. Użyte środowisko programistyczne oraz biblioteki:
Projekt wykonano w języku programowania C, który po zmiane rozszerzenia mógł od razu działać w c++ jako że jest to język z rodziny C. Nie było potrzeby używać obiektowości języka, cały program został napisany proceduralnie. Użyte narzędzie to Visual Studio 2013, które oferuje bardzo wygodny interfejs oraz oferuje wielkie możliwości. W
kodzie programu użyto następujących nagłówków i bibliotek:
Windows API – pozwala pisać programy wykorzystując możliwości systemu Windows, poprzez tzw uchwyty może odwoływać się do różnych elementów systemu , w tym takich podstawowych elementów jak okna, liczniki oraz standardowe przyciski, które robią za wizualne elementy czy też funkcje pośrednio wykonujące systemowe przerwania procesora.
OpenGL – przenośna multiplatformowa biblioteka do tworzenia aplikacji graficznych, można stwierdzić, że jest to dosyć „niskopiziomowy" budulec, na którym mogą być oparte różne bardziej złożone i wyspecjalizowane biblioteki, takie jak na przykład silniki graficzne pokroju Unity 3D. Na chwilę obecną jedynym znanym mi odpowiednikiem OpenGL godnym uwagi jest biblioteka DirectX opracowana przez firmę Microsoft. Natomiast warto wspomnięć iż coraz więcej aplikacjii z niej korzysta i być może czeka nas komercjalizacja OpenGL który powoli zamienia sie w Vulcan(API)
AntTweakBar – wygodny interfejs graficzny dla aplikacji opartych o biblioteki graficzne, z pewnością godny polecenia dla prostych aplikacji, został użyty do określenia pozycji obiektu oraz sprawdzenia warunku kolizjii (zmienna kolizja).
Glu – rozszerzenie do OpenGL dodające wiele funkcji zwiększających możliwości. Przykładowym elementem z tej biblioteki użytym w moim projekcie są kwadryki.
Inną funkcja którą użyłem było gluLookAt() funkcja odpowiedzalna za śledzenie kamery.

```
void uklad_xyz()
{
	glBegin(GL_LINES);

	// uklad kartezjanski os dla z ZIELONA
	glColor3f(0.0f, 255.0f, 0.0f);
	glLineWidth(5044);
	glVertex3f(0.0, 0.0, 0.0);
	glVertex3f(0.0, 0.0, 20000.0);
	glEnd();
	//glLineLength(5044);
	glBegin(GL_LINES);

	// uklad kartezjanski X NIEBIESKA
	glColor3f(0.0, 0.0f, 255.0f);

	glVertex3f(0.0, 0.0, 0.0);
	glVertex3f(100000.0, 0.0, 0.0);
	glEnd();



	glBegin(GL_LINES);

	// uklad kartezjanski   OS DLA Y ;) CZERWONA
	glColor3f(255.0f, 0.0f, 0.0f);
	glVertex3f(0.0, 0.0, 0.0);
	glVertex3f(0.0, 1000000.0, 0.0);
	glEnd();
```

3.Podstawy

3.1. Rozpoczęcie pracy z szablonem:

Na początku laboratorium dostałem kod z różnokolorowym sześcianem który można było obracać była to możliwość łatwego przyswojenia sobie w jaki sposób działa OpenGL.



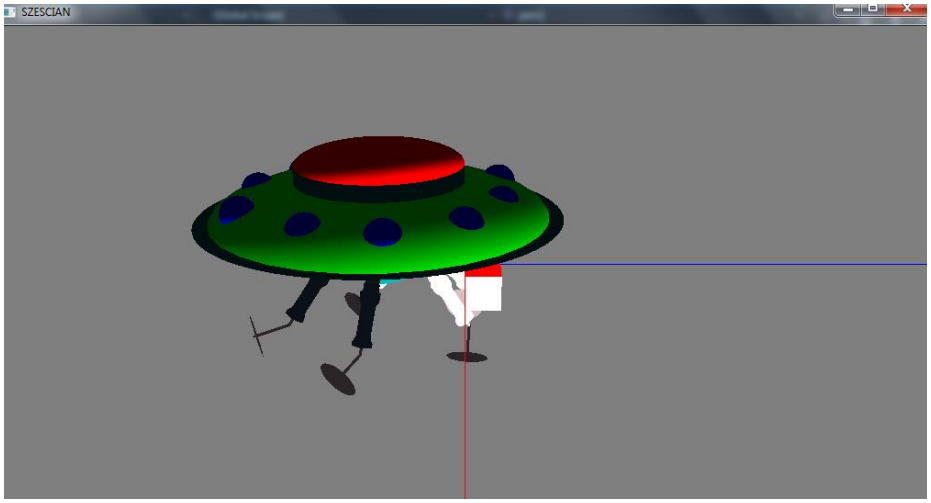
W tym momencie utworzyłem układ kartezjański dla osi xyz aby móc się łatwiej orientować w przestrzeni.

W swoim programie wykorzystałem wiele sfer które okazały się pomocne do tworzenia kul.

W tym celu napisałem funkcję tworzącą sferę.
`void sfera(double t, int czesci, int wart)`
Wkrótce jednak po poznaniu kwadryk dla kul wykorzystałem funkcje glut tworzącą sferę.

```
void sfera(double t, int czesci, int wart)
{
	int i, j;
	int pol = czesci / 2;
	for (i = 0; i <= pol; i++)
	{
		//glColor3f(0, 0, 25);
		double lat0 = GL_PI * (-0.5 + (double)(i - 1) / czesci);
		double z0 = sin(lat0);
		double zr0 = cos(lat0);

		double lat1 = GL_PI * (-0.5 + (double)i / czesci);
		double z1 = sin(lat1);
		double zr1 = cos(lat1);

		glBegin(GL_QUAD_STRIP);

		for (j = 0; j <= wart; j++)
		{
			double lng = 2 * GL_PI * (double)(j - 1) / wart;
			double x = cos(lng) * 4;
			double y = sin(lng) * 4;

			// glTexCoordf()

			glNormal3f(x * zr0, y * zr0, z0);
			glVertex3f(x * zr0, y * zr0, z0);

			// glTexCoordf()
			glNormal3f(x * zr1, y * zr1, z1);
			glVertex3f(x * zr1, y * zr1, z1);
		}
		glEnd();
	}
```

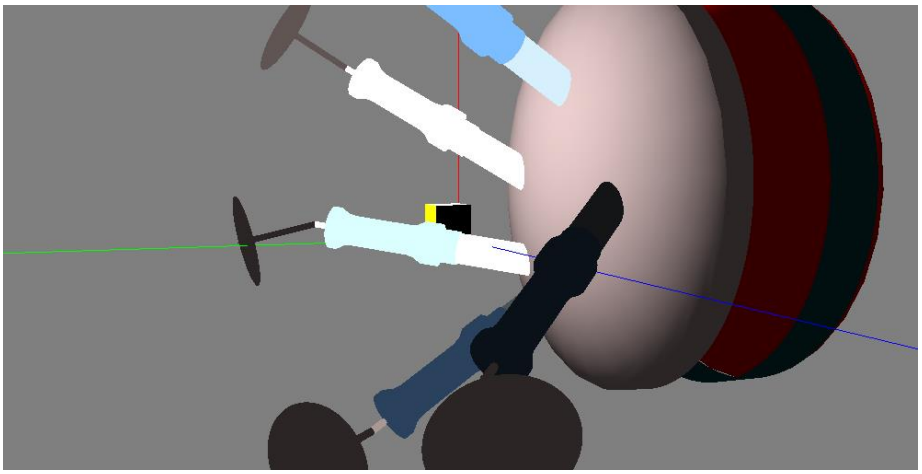
Natomiast do stworzenia kuli wykorzystany został kod który jedną sfere (pół kuli) obraca o 90 stopni i przesuwa ją w kierunku drugiej.




```
moja_sfera(30, 30, 30, 1);
                    ////////////////////////////////////kula3d();
              glRotatef(180, 1, 0, 0);
       glTranslatef(0, 0, 0.02);
       moja_sfera(30, 30, 30, 1);
```

Kod tworzy dwie sfery, jedną z nich obraca i przesywa w ten sposób tworząc kule.

```
kula3d();
glTranslatef(2, 4, 0); // przemieszcza kule
       kula3d();              s   //rysuje kule na tacy
statku i analogicznie
glTranslatef(4, -2, 0);
kula3d();
glTranslatef(4, -1, 0);
kula3d();
glTranslatef(4, 3, 0);
kula3d(); //dotoad ok
glTranslatef(0, -5, 0);
kula3d();
glTranslatef(-2, 4, 0);
kula3d();
glTranslatef(-4, -2, 0);
kula3d();
glTranslatef(-4, 0, 0);
kula3d();
glTranslatef(-4, 4, 0);
kula3d();
```



Aby nadać trochę życia postanowiłem zbudować Skybox czyli niebo (przestrzeń otaczająca) Skybox został stworzony ze zwykłego sześcianu którego każda ściana została pokryta teksturą nieba nocą:

`void drawSkybox(float size)` gdzie „size’’ oznacza wielkość sześcianu.

Tekstury zostały określone w części inicjującej okno OpenGL. Aby dodać więcej tekstur należy zmienić nazwę pliku w „`Bitmapy//nazwa.bmp`" gdzie „nazwa” nalezy wpisac odpowiednia nazwę pliku w formacie bmp z folderu Bitmapy.

```
glGenTextures(10, &texture[0]);                    // tworzy
obiekt tekstury

              // ³aduje pierwszy obraz tekstury:
              bitmapData =
LoadBitmapFile("Bitmapy//szklo.bmp", &bitmapInfoHeader);
              glBindTexture(GL_TEXTURE_2D, texture[0]);
// aktywuje obiekt tekstury
              glTexParameteri(GL_TEXTURE_2D,
GL_TEXTURE_MAG_FILTER, GL_LINEAR);
              glTexParameteri(GL_TEXTURE_2D,
GL_TEXTURE_MIN_FILTER, GL_LINEAR);
              glTexParameteri(GL_TEXTURE_2D,
GL_TEXTURE_WRAP_S, GL_CLAMP);
              glTexParameteri(GL_TEXTURE_2D,
GL_TEXTURE_WRAP_T, GL_CLAMP);
              // tworzy obraz tekstury
              glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB,
bitmapInfoHeader.biWidth,
                    bitmapInfoHeader.biHeight, 0, GL_RGB,
GL_UNSIGNED_BYTE, bitmapData);
              if (bitmapData)

                                        free(bitmapData);
```

Statek został wyrysowany poprzez wywołanie funkcji sfera z odpowiednimi parametrami i narysowaniu kilku kół w sobie potem zostały dołożone kule i nóżki.

Dla stworzenia księżyca zostałą stworzona odpowiednia kwadryka oraz przypisana do niej tekstura:

GL_SPHERE_MAP); - pozwala na mapowanie sferyczne, koordy dla tekstury każdego sferopodobnego obiektu so automatycznie obliczane przez opengl.

glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP);- gl tex geni zostało użyte juz przy teksturze, wpływa na sposób w jaki tekstura będzie mapowana.

```
void ksionzyc()
{
	glBindTexture(GL_TEXTURE_2D, texture[9]);
	glEnable(GL_TEXTURE_2D);
	GLUquadricObj *quadric;
	quadric = gluNewQuadric();
	//glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP);
	//glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP);
	glEnable(GL_TEXTURE_GEN_S);
	glEnable(GL_TEXTURE_GEN_T);
	gluQuadricDrawStyle(quadric, GLU_FILL);
	gluSphere(quadric, 50,30,30);
		glDisable(GL_TEXTURE_GEN_S);
		glDisable(GL_TEXTURE_GEN_T);



	glDisable(GL_TEXTURE_2D);

	}
```

Efekt jest zadowalający gdyż planeta została stworzona w całości w OpenGl nie jako obj z Blendera.

```
case WM_KEYDOWN: //************ OBSLUGA KLAWISZY **********************
		if (GetAsyncKeyState(VK_SHIFT) & 0x8000)
		{

		change = 1.0f;

	}
			else
				change=0.1f;

			switch ((int)wParam)
			{
			case VK_SPACE:
				if (k == 1) k = 0;
				else if (k == 0) k = 1;
				PostMessage(hWnd, WM_PAINT, 0, 0                    break;

			case VK_NUMPAD0:
				pivot_vert_angle += 5;
				if (pivot_vert_angle >= 360)
					pivot_vert_angle -= 360;
				InvalidateRect(hWnd, NULL, FALSE);
				break;
			case VK_NUMPAD1:
			{
				pivot_vert_angle -= 5;
				if (pivot_vert_angle < 0)
					pivot_vert_angle += 360;
				InvalidateRect(hWnd, NULL, FALSE);
			}break;
			case VK_UP:
			{
				if (k1 == 0)
				{
					if (gl_przesun_y <= -10 && gl_przesun_y >= -50)
					{
						gl_przesun_y -= 1;
					}
					else
					gl_przesun_y += 1;
				}

			}
					InvalidateRect(hWnd, NULL, FALSE);
				break;
```

Ruch kamery podążającej za statkiek został zaprojektowany za pomocą glulookAt()
gluLookAt(cam_x,cam_y,cam_z,obiekt_x,obiekt_y,obiekt_z,0,-1,0);
cam_x – okresli położenie punktu Z KTÓREGO patrzy kamera w osi x;
cam_y – okresli położenie punktu Z KTÓREGO patrzy kamera w osi y;
cam_z – okresli położenie punktu Z KTÓREGO patrzy kamera w osi z;
obiekt_x – określa położenie punktu na który patrzy kamera, jest to statek w osi x;
obiekt_y – określa położenie punktu na który patrzy kamera, jest to statek w osi y;
obiekt_z – określa położenie punktu na który patrzy kamera, jest to statek w osi z;

Ruch obiektu został zaprojektowany za pomocą zmiany parametrów gl_przesun_”(x,y,z)”
A ponieważ pozycja kamery jest dynamiczna jej finalna postać wygląda tak:

gluLookAt(gl_przesun_x - 30, gl_przesun_y-20 , gl_przesun_z - 50, gl_przesun_x, gl_przesun_y, gl_przesun_z, vek_x, vek_y, vek_z);

Przykład dla wcisnięcia przycisku ruchu w prawo.

```
case  VK_RIGHT:
                    {
                          if (k==0)
                          {
                    if (gl_przesun_x >= -140 && gl_przesun_x <= -
                    100)
                                {
                                      gl_przesun_x -= 1;
                                }
                                else
                                gl_przesun_x += 1;

                          }
                    }
```

Ponieważ funkcja gluLookAt() jako domyślne 3 ostatnie parametry bierze wektór górny (0, -1,0) lub dolny (0,1,0) umożliwiłem użytkownikowi zmiane wszystkich parametrów funkcjii za pomocą klawiszy F1-F5 tak by dowolnie mógł ustawic kamere.

```
                    case VK_F1:
                          vek_x += 1;
                          break;
                    case VK_F2:
                          vek_x -= 1;
                          break;
                    case VK_F3:
                          vek_y += 1;
                          break;
                    case VK_F4:
                          vek_y -= 1;
                          break;
                    case VK_F5:
```

W pętli rysującej należało wywołać odpowiednie funkcje wraz z odpowiednia ich kolejnością, aby translacje nie nakładały sie na siebie wykorzystałem glPush oraz glPop do resetowania macierzy translacjii i rotacjii.
Została użyta domyślna macierz projekcjii.

```
gluLookAt(gl_przesun_x - 30, gl_przesun_y-20 , gl_przesun_z - 50, gl_przesun_x, gl_przesun_y, gl_przesun_z, vek_x, vek_y, vek_z);
  drawSkybox(500);



  glPushMatrix();
  glTranslatef(-120, -30, 40);
  ksionzyc();
  glPopMatrix();


    // umiejscowienie kamery w odleglym miejscu
  //   glTranslatef( pivot_x, pivot_y, -z_dist ); - bezuzyteczne bo jest gluLookAt a razem nie dzialaja.


       glPushMatrix();
  glRotatef(pivot_vert_angle, 1, 0, 0);
       glTranslatef(gl_przesun_x, gl_przesun_y, gl_przesun_z);
       glRotatef(270, 1, 0, 0);
       pen();
glPopMatrix(0;
```

Wciskając F12 można chować podwozie statku(ukrywać). AntTweakBar aktualizuje sie co 4 klatki.

Kod dla AntTweakBara to głównie stworzenie obiektu typu Bar:

```
TwInit(TW_OPENGL, NULL);
TwWindowSize(800,600); //inicjacja tweak bara dla okna 800x600
myBar = TwNewBar("Interfejs"); //stworzenie obiekty typu Bar
TwAddVarRW(myBar, "pozycja-x", TW_TYPE_FLOAT, &gl_przesun_x, "");// stworzenie parametru dla paska który wyświetla wartość zmiennej gl_przesun_x poprzez wskaznik.

TwAddVarRW(myBar, "pozycja-y", TW_TYPE_FLOAT, &gl_przesun_y, "");// stworzenie parametru dla paska który wyświetla wartość zmiennej gl_przesun_y

TwAddVarRW(myBar, "gl_przesun-z", TW_TYPE_FLOAT, &gl_przesun_z, ""); //stworzenie parametru dla paska który wyświetla wartość zmiennej gl_przesun_z

TwAddVarRW(myBar, "Kolizja", TW_TYPE_BOOLCPP, &kolizja, "TAK lub NIE"); // stworzenie zmiennej typu bool które określa czy doszło do kolizji z obiektem czy nie.

TwAddButton(myBar, "przycisk - W", Callback, NULL, " 'W' "); //właściwie miało sprawdzać czy nastąpiło wciśnięcie przycisku natomiast ze względu na problem z odwołaniem do Callback nieużywana.
```

4. Podsumowanie:

Dzięki temu projektowi nauczyłem się podstaw tworzenia grafiki przy pomocy „niskopoziomowej” biblioteki jaką jest OpenGL. Poznałem istotę obiektów graficznych z niskopoziomowego punktu widzenia, nauczyłem się dokonywać transformacji obserwacji, operować oświetleniem, i dodawać proste elementy fizyki. Wykorzystałem shader jeśli chodzi o mgłe.

Właściwie mój statek zo stał stworzony z bardzo prostych figur i tu bardzo pomocną okazała sie matematyka która w momencie gdy osoba tworząca projekt ma znacznie większe umiejętności ode mnie jesli chodzi o matematykę może stworzyc coś niesamowitego.

System kolizjii budowany przeze mnie był jedynie sferowy lub punktowy gdyż próby utworzenia lepszego systemu kolizjii nie dawały efektów zapewne z powodu poziomu mojej zbyt słabej wiedzy.

Natomiast największych problemów przyszporzyło mi rozumienie timerów przez co poruszanie sie zostało we przeze mnie wykonane tylko w stopniu podstawowym, jednocześnie animacja układu słonecznego nie powiodła się z tego samego powodu, dlatego też pozostałem przy samym księżycu.

W programie oraz nauce użyto tutoriali z :

http://nehe.gamedev.net/

Nehe textures tutoriual

Zostały również użyte 3 gotowe funckje z:

http://stackoverflow.com/

Boundaring boxes, spheres intersection.