Przygoda z OpenGL na tureckim yachcie

Kamil Łopuszański, Patryk Mendrala

Rok i grupa: **EF-DI L5**
Moduł: Grafika komputerowa
Data: 2016

1 Wprowadzenie

Do realizacji projektu użyta została technologia OpenGL (ang. *Open Graphics Library*) w wersji 1.2.2.0 Microsoft Corporation, zbiór funkcji tworzących API do generowania grafiki. Zestaw funkcji umożliwia budowanie złożonych trójwymiarowych scen z podstawowych figur geometrycznych: linii, wielokątów.

OpenGL ma rolę analogiczną, jak konkurencyjny DirectX firmy Microsoft. Oprócz gier i małych programów w stylu wygaszacze ekranu, również programy do przedstawiania wyników badań naukowych oraz rozszerzonej rzeczywistości (ang. *augmented reality*) używają OpenGL.

OpenGL działa jako maszyna wirtualna. Polecenia zapisane w języku OpenGL są tłumaczone na język maszynowy sterownika konkretnego zastosowania. Pisząc programy, nie definiujemy typów zmiennych używając nazw języka C++: float, double, lecz GLfloat, GLdouble. To zapewnia przenośność między platformami. Na stan OpenGL w danym momencie składa się szereg parametrów i trybów działania, które można ustawić lub zapamiętać na stosie i później odtworzyć. Ich konfiguracja będzie miała bezpośredni lub pośredni wpływ na otrzymany rezultat renderingu. Raz ustawiony parametr lub tryb działania pozostaje zachowany aż do następnej zmiany. Przykładami takich parametrów mogą być: kolor rysowania, aktualnie używana tekstura, sposób działania bufora Z, macierz, na której wykonywane są aktualnie operacje oraz wiele innych. Czasem parametry mogą się na siebie nakładać, jak jest w przypadku macierzy transformacji. Dodawanie nowych efektów: translacji, rotacji, skalowania itd. osiąga się poprzez mnożenie aktualnej macierzy przez macierz transformacji.

1.1 Pomocnicze biblioteki

Istnieją pomocnicze biblioteki, które można wykorzystać przy okazji programowania w OpenGL. Należą do nich: GLUT, GLEW, GLAUX. Programowanie w środowisku Windows i Win32 nastręcza jednak trudności, bo tworzą się często błędy, które są trudne do zlokalizowania i naprawienia.

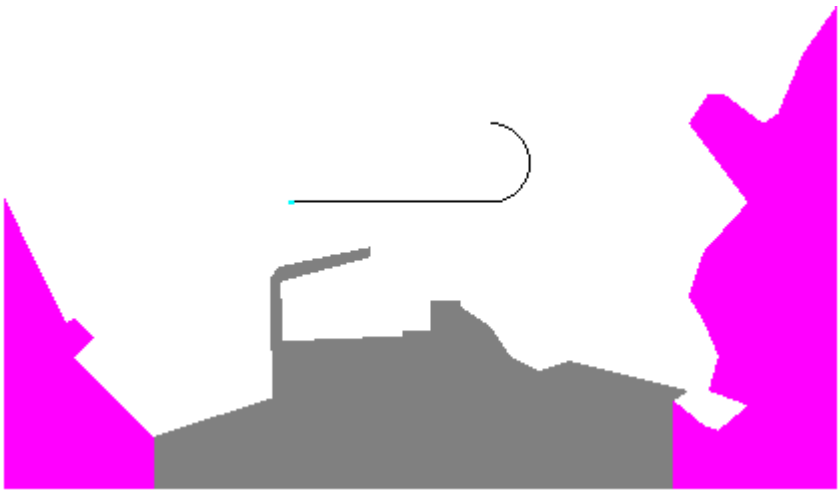
1.2 Obiekt modelowany

Celem projektu było zamodelowanie yachtu - małego statku o długości ok. 5m. Aby coś zamodelować, należy zapoznać się z terminologią i cechami fizycznymi danej rzeczy. Yacht składa się z rufy z tyłu, dziobu z przodu, kadłuba w środku. Ze względu na niedoświadczenie i brak czasu, studenci nie porywali się na tworzenie bardziej skomplikowanego modelu. Prawdziwy yacht posiada ster, olinowanie, kajutę itd.

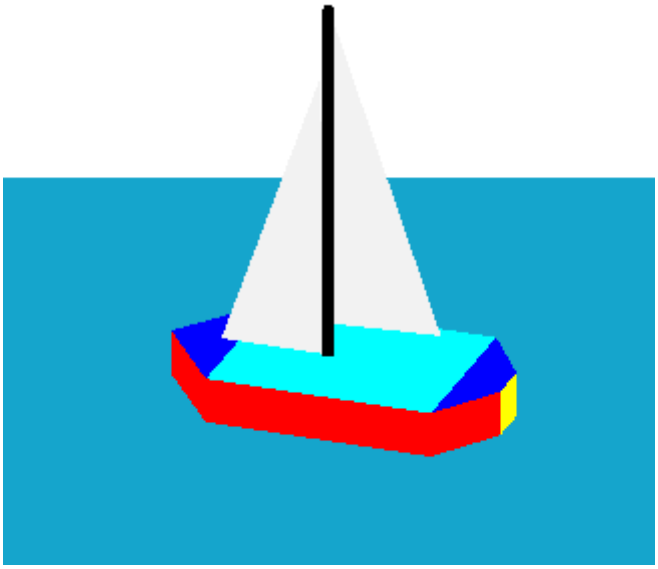
2 Historia zmian

2.1 Laboratorium 1

Co zrobiono Na 1. laboratorium do wzorca dodano 2 funkcje: *rufa* oraz *kadlub*, stanowiące dno pokładu i kształt jachtu. Kadłub jest po prawej stronie i jest trójkątna, zaś rufa z tyłu i ma kształt graniastosłupa o podstawie będącej trapezem. Globalny środek układu współrzędnych wybrano na dnie jachtu, na jego środku, aby ułatwić orientację pozostałych punktów w przestrzeni oraz późniejsze skalowanie. Bowiem jacht można w przybliżeniu uważać za symetryczny wzgl. środka dna. Wygląd yachtu przedstawia poniższy rysunek.



Rysunek 2.1: Tor yachtu w animacji



Tam gdzie kadłub lub rufa łączy się z wnętrzem, wierzchołki są zdublowane. Zdublowane są także dwie ściany.

Dla ułatwienia pisania, stworzono 2 uniwersalne funkcje: *drawCuboid* i *drawTriangle*. Za ich pomocą stworzono maszt i 2 żagle. Korzystając z mapy miejsca[1], utworzono linię brzegową, która odgradza również namalowane: *akwen* i ląd.

Początkowo układ współrzędnych był położony „po kartezjańsku", tzn. w płaszczyźnie rysunku zawarte były osie x i y, zaś z przecinała płaszczyznę. Uznano, że sprawia to problemy w projektowaniu i komplikuje rozłożenie obiektów. Zmieniono więc orientację tak, że oś z jest pionowa. Dodano również nowe features w zakresie samego okna: standardowe położenie na ekranie, tytuł oraz możliwość poruszania się za pomocą klawiszy numerycznych 2,4,6,8.

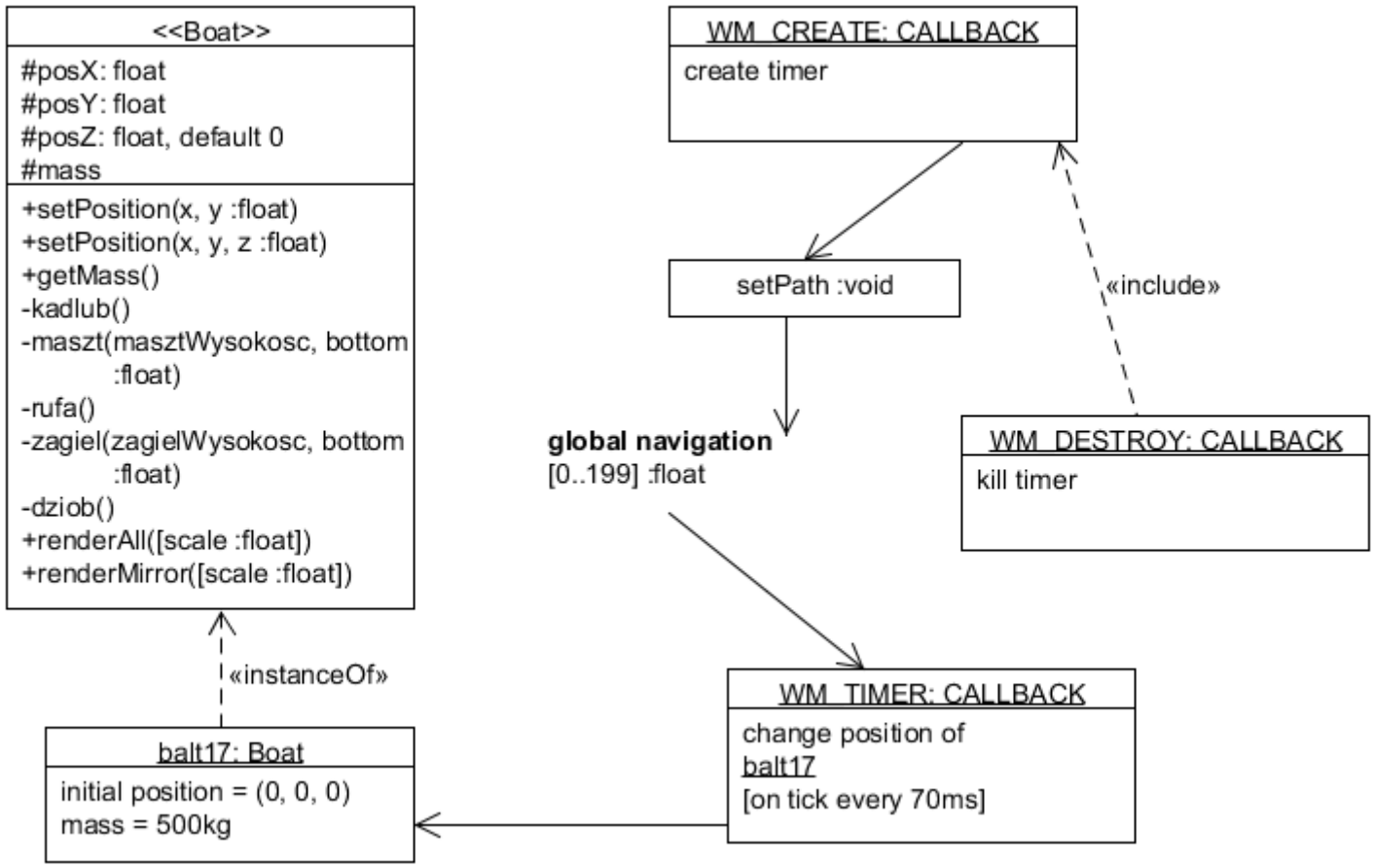
Wyznaczony tor ruchu yachtu dla animacji jest pokazany na rys. 2.1. Jest to linia krzywa, zaczynająca się w punkcie (0. 0) i kończy w (2000, 800). Pierwsza część jest linią prostą, druga łukiem okręgu o promieniu 400.

Zaprojektowano i stworzono klasę Boat, reprezentującą yacht, zob. rys. rys. 2.2. Klasa zawiera pola dotyczące pozycji oraz masy statku. Główna funkcja renderująca statek - *renderAll()* - wywołuje wszystkie pozostałe - *kadlub()*, *dziob()*, *rufa()*, *zagiel()*, *maszt()*. Klasa udostępnia oprócz tego 3 metody pochodne od *renderAll()* - *renderAll(scale)*, *renderMirror()*, *renderMirror(scale)*. Jeżeli argument scale jest podany, to

[1] wybrano marinę w Turcji, o współrzędnych geograficznych 36°49′5″N 28°18′32″E

statek jest przeskalowany. *renderMirror([...])* rysuje statek pod wodą - ma to na celu stworzenie lustrzanego odbicia, widocznego z góry. Globalną instancją klasy Boat jest zmienna balt17.

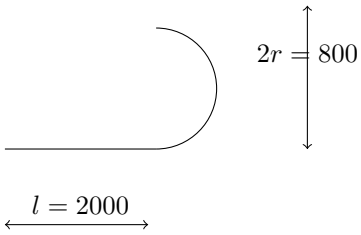
Trasa, po której pływa statek jest wyznaczana na początku działania aplikacji, w sekcji **WM_CREATE** funkcji *WinMain*. Wywoływana jest funkcja *setPath()*, która inicjalizuje wartości zmiennej globalnej navigation. Oprócz tego w tej sekcji, uruchamiany jest czasomierz - timer, który wysyła CALLBACK **WM_TIMER**. W sekcji odbierającej ostatni CALLBACK, aktualizowana jest pozycja statku - balt17.



Rysunek 2.2: Diagram klas dla Boat

Problemy Nastąpiły problemy związane z orientacją osi oraz nazewnictwem punktów SA–SH. Nie było wiadomo, który punkt w kodzie programu odpowiada któremu puntowi na renderze. Nastąpiło kilka pomyłek. Pozostaje niewyjaśnione, dlaczego oś y jest pionowa w płaszczyźnie rysunku, zaś oś z prostopadła do tej płaszczyzny. Następnie stracono dużo czasu z powodu niewłaściwego sposobu wykreowania rufy za pomocą istniejącego kadłuba. Skopiowano początkowo punkty, a następnie zamieniono wartości na osiach x oraz z na przeciwne. Doprowadziło to do całkowitej dezorientacji, ponieważ punkt SA nie odpowiadał puntowi RA, SD punktowi RD itd.

Wnioski Należy trzymać się konsekwentnego nazewnictwa punktów, a w przypadku całkiem nowego punktu, używać unikatowej nazwy.



Rysunek 2.3: Trajektoria i jej wymiary (pierwsza połowa)

2.2 Laboratorium 2

Na 3. laboratorium został zaimplementowany ruch po zadanej trajektorii (rys. 2.3). Ruch był 2-częściowy. Krzywa została podzielona na punkty, ponieważ w grafice nie da się modelować krzywych czy objętości ciągłych. Pierwszą część stanowi prosta złożona z 62 punktów, o równaniu

$$\begin{cases} x_i = i \cdot \Delta L \\ y_i = 0 \end{cases} \tag{2.1}$$

gdzie i jest numerem punktu, zaś $\Delta L = \frac{2000}{62}$. Drugą część stanowi łuk 180°, złożony z 38 punktów, o równaniu

$$\begin{cases} x = l + r \cdot \cos(\alpha) \\ y = r + r \cdot \sin(\alpha) \end{cases} \tag{2.2}$$

gdzie $\alpha \in [-\pi, \pi]$, a (l, r) stanowi współrzędne środka okręgu, w którym zawiera się łuk. Należy zwrócić uwagę na parametryczność współrzędnych. Ich wartości nie zostały obliczone jawnie, lecz obie zależały od parametru α.

Następne 100 punktów to symetria pierwszych 100 względem środka okręgu:

$$\begin{cases} x_{i+100} = 2l - x_i \\ y_{i+100} = 2r - y_i \end{cases} \tag{2.3}$$

dla $i = 100, 101, \cdots 199$

2.3 Laboratorium 3

Co zrobiono Na 3. laboratorium został zaimplementowany prosty silnik fizyczny. Silnik ten odpowiada za ruch okrętu zgodny ze wzorami:

$$a(t) = \frac{1}{m} * F(t) \tag{2.4}$$

$a(t)$ - przyspieszenie okrętu m - masa okrętu $F(t)$ - suma sił działających na okręt

$$v(t) = v(t - \Delta t) + a(t) * \Delta t \tag{2.5}$$

$v(t)$ - prędkość okrętu $v(t - \Delta t)$ - prędkość okrętu w poprzedniej jednostce czasu $a(t)$ - przyspieszenie okrętu Δt - przyrost czasu

$$x(t) = x(t - \Delta t) + v(t) * \Delta t \tag{2.6}$$

$x(t)$ - pozycja okrętu
$x(t - \Delta t)$ - pozycja okrętu w poprzedniej jednostce czasu
$v(t)$ - prędkość okrętu
Δt - przyrost czasu

Problemy ...

Wnioski ...

3 Język C++

Ponieważ początkowy kod źródłowy był napisany w języku C, autorzy zmienili go na język C++. Zaletą tego wyboru była m. in. możliwość uproszczenia kodu. Możliwe stało się wykorzystanie tzw. zmiennych anonimowych:

```
drawCuboid(new GLfloat[6] { 4, 6, -1, 1, masztDolWysokosc,
 (masztDolWysokosc + masztDlugosc) });
```

zamiast definiować tablicę, której nie używa się nigdzie poza jednym miejscem w funkcji. Takie podejście upraszcza programowanie i czyni kod bardziej czytelnym. Dzięki przejściu można było również zdefiniować klasy: Boat, Physics.

4 Matematyka i fizyka OpenGL

4.1 Światło i materiał

Aby wprowadzić światło do wygenerowanego świata, trzeba panować nad kolorem materiału, obiektów i tekstur, a także kolorem samego światła. Materiały są niezbędne do ustawienia, ponieważ domyślnie kolor materiału jest szary i po oświetleniu kamera widzi wszystko czarno-biało.

Matematyka materiału jest prosta. Jeżeli komponenty materiału oznaczymy przez (MR, MG, MB), światła zaś przez (LR, LG, LB), to po oświetleniu zobaczymy kolor posiadający komponenty (LR·MR, LG·MG, LB·MB). Podobnie, w obecności wielu źródeł światła, ich komponenty są dodawane: (R1+R2, G1+G2, B1+B2).

4.2 Transformacja geometryczna

Nie zawsze łatwo dokładnie określić współrzędne, w jakich powinien znajdować się nasz obiekt. Aby łatwo było określić i zmieniać te miejsca, używa się transformacji macierzowych. Jeśli wektor współrzędnych jest równy $[x, y, z, w]^T$ gdzie w jest globalnym czynnikiem skalującym, to żądane współrzędne uzyskuje się w następujący sposób:

$$\mathbf{M} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} \tag{4.1}$$

gdzie $\mathbf{M}$ jest macierzą transformacji. Np. translację o wektor $[t_x, t_y, t_z]^T$ uzyskuje się poprzez wykorzystanie macierzy

$$M = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}]$$

Najpierw jednak zastosowana jest rotacja. Wywołana jest funkcja *glRotatef*. Funkcja ta ma składnię:

```
glRotatef(GLfloat angle, GLfloat x, GLfloat y, GLfloat z );
```

Tablica 4.1: Parametry ustawienia kamery

eye	pozycja kamery w przestreni
center	wyznacza oś optyczną. Zwykle jest to punkt w środku wydarzeń, na który uwaga obserwatora powinna zostać zwrócona
up	razem z pozycją eye wyznacza wektor prostopadły do kamery, zwrócony w górę

i obraca obiekt o kąt *angle* wokół linii wyznaczonej przez wektor $[x, y, z]$. Kierunek obrotu jest określany w stopniach (w przedziale 0°–360°) w kierunku przeciwnym do ruchu wskazówek zegara. Należało wcześniej upewnić się, że środek statku jest w punkcie *origin* świata.

Animacja jest zaimplementowana w następujący sposób:

```
glPushMatrix();
        glTranslatef(navigation[0][i] + balt17.getPos()[0],
        navigation[1][i] + balt17.getPos()[1], navigation[2][i] + balt17.getPos()
        glRotatef(navAngle[i] * 180 / GL_PI, 0.0, 0.0, 1.0);
glPopMatrix();
```

gdzie parametry przesunięć i kąta są zapisane w zmiennej navigation.

Kamera To, co będzie widział obserwator zależy ściśle od kamery. Do regulowania jej parametrów użyto funkcji *gluLookAt* o składni

```
void gluLookAt(GLdouble eyex, GLdouble eyey, GLdouble eyez,
        GLdouble centerx, GLdouble centery, GLdouble centerz, GLdouble upx,
        GLdouble upy, GLdouble upz);
```

czyli praktycznie parametrami są współrzędne 3 punktów w przestrzeni o następującym znaczeniu:

W programie wykorzystane są 2 ustawienia kamery, jedno w planie ogólnym, drugie jest ujęciem zwróconym na ruch yachtu.

5 Opis programu

5.1 Animacja

Animacja wykorzystana w projekcie polega na wykonaniu kolejnych kroków dla yachtu:

1. wyznacznie ścieżki, po jakiej porusza się okręt
2. odbicie w wodzie statku otrzymane poprzez odbicie lustrzane wzgl. płaszczyzny xy
3. rotacja statku,
4. translacja statku

Odbicie zostało zrealizowane jako transformacja z macierzą

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

poprzez następujący kod:

```
void Boat::renderMirror()
{
    //mirror everything from top do bottom
    const GLfloat m[16] = { 1, 0, 0, 0,
                            0, 1, 0, 0,
                            0, 0, -1, 0,
                            0, 0, 0, 1 };
    glPushMatrix();
    glMultMatrixf(m);
    renderAll();
    glPopMatrix();
}
```

Warto zauważyć, że kolejność wykonania rotacji i translacji miała znaczenie i rotacja jest wykonywana jako pierwsza nieprzypadkowo.

5.2 GUI

Spośród wielu gotowych technologii tworzenia GUI (Qt, SFML, Win32 itd.) wybrano AntTweakBar, ponieważ jest on dobrze zintegrowany zarówno z językiem C/C++ jak i z technologią OpenGL.

5.3 Sterowanie klawiszami

Rozszerzono możliwości sterowania sceną za pomocą klawiatury. Do możliwości wcześniej napisanych: sterowanie za pomocą klawiszy strzałek, dodano następujące:

4 rotate left

8 rotate up

6 rotate right

2 rotate down

- zoom out

+ zoom in

a translate left

w translate forward

d translate right

s translate backward

Zoom polega na zmianie parametrów kamery, pozostałe wybory zmieniają transformują całą scenę, dając złudzenie ruchu kamery.

Tablica 5.1: Parametr funkcji *glTexParameter**

param	Opis
GL_CLAMP	poza obszarem obrazu tekstury (zakresem od 0,0 do 1,0) jako koloru tekstury używa się koloru ramki lub określonego stałego koloru
GL_REPEAT	powoduje powtarzanie tekstury na całej powierzchni wielokąta

5.4 Tekstury

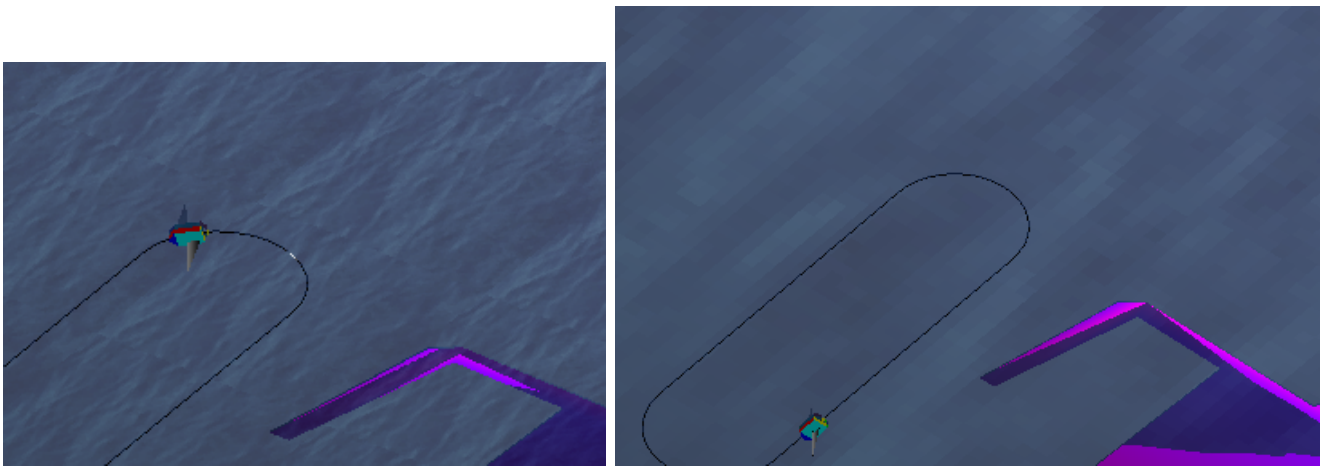
OpenGL posiada funkcje mapowania tekstur, umożliwiające nakładanie obrazów na wielokąty w scenie. O tym, jak te obrazy zostaną nałożone, decyduje programista. Opcji jest jednak tak wiele, że z łatwością mogą przytłoczyć i spowodować złe wrażenie. Początkowo wykorzystano następujący kod do inicjalizacji tekstury symulującej fale na morzu:

```
1 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
2 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
3
4 glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
```

Wynik tego jest przedstawiony na rys. 5.1. Możliwe parametry są opisane w tab. 5.1.

Wynik oteksturowania nie jest zadowalający, ponieważ fale są zbyt rzadko położone i widać skutki rasteryzacji obrazu. Użytkownikowi wydaje się, że ktoś wykonał zdjęcie aparatem o małej rozdzielczości. Po zastosowaniu drugiej odmiany parametru i przeskalowania tekstury (zmniejszenia), otrzymano lepszy wygląd grafiki (rys. 5.1). Skalowanie tam, gdzie tworzony jest akwen - zmieniono współrzędne granicznych teksli:

```
1 glBegin(GL_QUADS);
2         glTexCoord2f(0.0, 0.0); glVertex3f(-range, -range, 0.0);
3         glTexCoord2f(0.0, 9.0); glVertex3f(-range, range, 0.0);
4         glTexCoord2f(9.0, 9.0); glVertex3f(range, range, 0.0);
5         glTexCoord2f(9.0, 0.0); glVertex3f(range, -range, 0.0);
6 glEnd();
```



Rysunek 5.1: Wynik nałożenia tekstury z parametrem GL_CLAMP (po prawej) i GL_REPEAT (po lewej)

Literatura

- Leniowski, Ryszard. Wykłady dla kierunku informatyka
- Wright, R.S.; Sweet, M. *OpenGL. Ksiega Eksperta*
- Woo, Mason; Neider, Jackie; Davis, Tom. *OpenGL Programming Guide. The Official Guide to Learning OpenGL, Version 1.1.* 2nd Edition

Źródła internetowe:

- https://msdn.microsoft.com/en-us/library/ee855621.aspx
- http://stackoverflow.com/questions/8494942/why-does-my-color-go-away-when-i-enable-ligh
- http://nehe.gamedev.net/tutorial/clipping__reflections_using_the_stencil_buffer/17004/