Zachowanie proporcji obiektów oraz tablice wierzchołków

1. *Funkcja* `glutReshapeFunc()`

   Z pewnością każdy zauważył, że zmieniając rozmiar okna przy pomocy myszki scena może stracić proporcje. Generalnie powinno być jasne, że ponieważ po zrzutowaniu sceny na okno rzutni okno to jest przekształcane na okno ekranowe (liniowo), więc w przypadku gdy proporcje wymiarów tych okien nie są takie same, to proporcje obiektów w oknie ekranowym są inne niż w oknie rzutni. Dla ustalenia uwagi załóżmy, że okno rzutni ma wymiary `SizeX` i `SizeY`, odpowiednio w poziomie i w pionie, natomiast okno ekranowe ma w tych kierunkach odpowiednio `M` oraz `N` pikseli. Aby obiekty w oknie ekranowym miały takie same proporcje jak w oknie rzutni musi zajść warunek

   $$\frac{M}{N} = \frac{SizeX}{SizeY}. \tag{1}$$

   Aby zapewnić zachodzenie tego warunku w każdej chwili gdy scena jest odrysowywana (więc również przy pierwszym jej rysowaniu) posłuzymy się funkcją `glutReshapeFunc()`. Posiada ona jeden parametr, który określa nazwę funkcji odpowiedzialnej za odrysowanie okna. Zatem zasada działania tej funkcji jest taka sama jak znanych już funkcji `glutDisplayFunc()` i `glutIdleFunc()`. Różnica leży w tym, że musi ona posiadać dwa określone parametry, oznaczajace aktualne podane w pikselach wartości szerokości i wysokości okna ekranowego, przy czym funkcja pobiera te wartości automatycznie, tzn. nie trzeba ich specyfikować przy jej wywołaniu. Załóżmy, że funkcja odpowiedzialna za odrysowanie okna nazywa się `odrysuj()`. Funkcja ta będzie wywoływana za każdym razem gdy zostanie wygenerowane zdarzenie *odrysuj okno*, zatem również przy pierwszym rysowaniu sceny.

   Należy jednak przypomnieć, że znana już funkcja `glutInitWindowSize` (ćwiczenia nr 1) pozwala na jawne ustalenie wymiarów okna ekranowego, niezależnie od wymiarów okna rzutni. Wtedy oczywiście warunek (1) nie zawsze będzie zachodził. Wyjścia z tej sytuacji są dwa

   (a) Wydzielenie podokna w oknie ekranowym takiego, że jego proporcje będą zgodne z proporcjami okna rzutni i narysowanie sceny w tym podoknie.

   (b) Zmiana proporcji okna rzutni, tak aby dopasowało się do proporcji okna ekranowego.

   W obu przypadkach potrzebna będzie funkcja definiująca aktualne rozmairy podokna w oknie ekranowym - w przypadku (b) wygeneruje ona podokono równe calemu oknu. Funkcja ta nazywa się `glViewport()` i posiada cztery parametry

   `glViewport(x, y, width, height)`,

gdzie `x,y` oznaczają przesunięcie dolnego, lewego narożnika podokna względem dolnego, lewego narożnika okna, który ma wartość `(0,0)`; natomiast `width,height` oznaczają odpowiednio szerokość i wysokość podokna. Wszystkie te parametry są wyrażone w pikselach.

- **Przypadek (a)**

  Szerokość i wysokość podokna powinna być taka, żeby dla tych wymiarów zachodził warunek (1), tzn.

  $$\frac{width}{height} = \frac{SizeX}{SizeY}. \tag{2}$$

  Ponadto względy estetyczne powodują, że podokno powinno być wycentrowane względem okna i powinno zajmować w nim maksymalny obszar taki, że warunek (2) zachodzi. Ostatnie żądanie implikuje, że w przypadku, gdy $\frac{M}{N} > \frac{SizeX}{SizeY}$, to użyjemy całej wysokości okna ekranowego, tzn. `height=N` oraz części jego szerokości. Parametr `width` musi spełniać warunek

  $$width = \left[\frac{SizeX}{SizeY}\ height\right] = \left[\frac{SizeX}{SizeY}\ N\right], \tag{3}$$

  gdzie $[x]$ oznacza część całkowitą z liczby x. Wycentrowanie podokna względem okna oznacza, że musimy się przesunąć w poziomie o wartość $[(M - witdth)/2]$. Analogicznie rozpatrujemy przypadek, gdy $\frac{M}{N} < \frac{SizeX}{SizeY}$. Trzeba jeszcze tylko dopowiedzieć, że w przypadku używania funkcji `gluPerspective()`, stosunek $\frac{SizeX}{SizeY}$ jest równy drugiemu parametrowi tej funkcji (ćwiczenia nr 1). Przykład pokazujący działanie tak skonstruowanej funkcji `odrysuj()` jest zawarty w pliku `ogl41.cpp`.

- **Przypadek (b)**

  Tym razem rozwiązanie jest prostsze: parametry `width, height` są równe aktualnej szerokości i wysokości okna ekranowego, natomiast parametry `x,y` są równe zero. Dopasowanie proporcji okna rzutni do proporcji okna ekranowego polega na ustawieniu drugiego parametru funkcji `gluPerspective()` tak, aby był on równy stosunkowi aktualnej szerokości okna ekranowego do jego aktualnej wysokości. Oczywiście oznacza to, że funkcja `gluPerspective()` musi być wywołana w funkcji `odrysuj()`. Proszę zwrócić uwagę, że ponieważ funkcja `odrysuj()` będzie wywołana przed funkcją `display()`, więc wszystkie operacje, które muszą zostać wykonane przed funkcją `gluPerspective()` muszą się również znaleźć w funkcji `odrysuj()`. Przykład pokazujący działanie tak skonstruowanej funkcji `odrysuj()` jest zawarty w pliku `ogl42.cpp`.

2. *Tablice wierzchołków*

Rozpatrzmy jeszcze raz kod rysujący sześcian zawarty w pliku `ogl21.cpp`. Każda ściana jest tam zbudowana z czterych wierzchołków, więc jej kod zawiera cztery wywołania `glVertex3f()`. Z konstrukcji sześcianu wynika, że każdy wierzchołek nalezy do trzech ścian, więc kod każdego wierzchołka pojawia się trzy razy. Daje to 24 wywołania funkcji `glVertex3f()`. Ponadto dodatkowo specyfikowany jest kolor dla każdej ściany, co zwiększa ilość wywołań o 6. W ogólności moglibyśmy wyspecyfikować kolor dla każdego wierzchołka, co zwiększyłoby ilość wywołań o 24 zamiast 6. Ponadto w przyszłości będziemy specyfikowali dla danego wierzchołka inne dane (wektory normalne, współrzędne tekstury, parametry materiałów itd.). W efekcie ilość wywołań pomiędzy funkcjami `glBegin()` i `glEnd()` jeszcze wzrośnie. Pewnym rozwiązaniem tego problemu są tzw. *tablice wierzchołków*. Pojęcie to obejmuje więcej niż tylko specyfikowanie ich geometrii: będziemy tworzyć tablice, z których jedna będzie zawierała dane geometryczne dla układu wierzchołków (np. tych konstruujących sześcian), w drugiej będą zawarte ich kolory, w kolejnych (dalsze ćwiczenia) dane innych typów. Następnie przez użycie funkcji indeksujących te tablice będzimy mogli przez jedno wywołanie pobrać wszystkie dane dla określonego wierzchołka. Dla wyjaśnienia zagadnienia rozpatrzmy obiekt dużo prostszy niż sześcian, a mianowicie kwadrat. Przykładowy kod może wyglądać tak:

```
void kwadrat()
{
  glBegin(GL_QUADS);
    glColor3f(1.0, 0.0, 0.0);
    glVertex3f( 0.5,  0.5,  0.5);
    glColor3f(0.0, 1.0, 0.0);
    glVertex3f( 0.5,  0.5, -0.5);
    glColor3f(0.0, 0.0, 1.0);
    glVertex3f(-0.5,  0.5, -0.5);
    glColor3f(1.0, 1.0, 0.0);
    glVertex3f(-0.5,  0.5,  0.5);
  glEnd();
}
```

Zdefiniujmy teraz dwie tablice: jedną dla wierzchołków tego kwadratu, drugą dla kolorów tych wierzchołków:

```
float wierzcholki[] = {-0.5,  -0.5, 0.0,
                        0.5,  -0.5, 0.0,
                        0.5,  0.5, 0.0,
                        -0.5,  0.5, 0.0
```

```
                    };

float kolory[] = {1.0, 0.0, 0.0,
                  0.0, 1.0, 0.0,
                  0.0, 0.0, 1.0,
                  1.0, 1.0, 0.0
                 };
```

Zasada używania takich tablic jest następująca:

(a) Najpierw trzeba włączyć możliwość używania każdej z nich. Mówiąc dokładniej umożliwiamy przekazywanie geometrii wierzchołków i ich kolorów przez struktury tablicowe. Potrzebna jest do tego funkcja

`glEnableClientState(tablica)`,

gdzie parametr `tablica` będzie w naszym przypadku przyjmował wartości `GL_VERTEX_ARRAY` oraz `GL_COLOR_ARRAY` (na dalszych ćwiczeniach wprowadzimy kolejne wartości tego parametru).
W momencie gdy chcemy zakończyć używanie danych tablic wystarczy wywołać

`glDisableClientState(tablica)`.

(b) Następnie trzeba wskazać która tablica zawiera dane dla geometrii, a która dla kolorów oraz jaka jest struktura tych tablic i które dane bedą z niej pobierane (w naszym przypadku uzyjemy wszystkich danych, ale oczywiście mogłyby one być składowymi dużo większych tablic, w których nie musiałyby być podane jedna za drugą). Aby to zrobić trzeba wywołać inną funkcję dla każdego typu danych, ale wszystkie takie funkcje mają ten sam typ parametrów. Dla zdefiniowania tablicy geometrii wierzchołków wywołujemy funkcję

`glVertexPointer(rozmiar, typ, offset, wskaznik_do_tablicy )`,

gdzie `rozmiar` oznacza ilość danych na wierzchołek (w naszym przypadku `3`), `typ` określa typ danych (`GL_SHORT, GL_INT, GL_FLOAT` lub `GL_DOUBLE`) (w naszym przypadku `GL_FLOAT`), `offset` oznacza przesunięcie pomiędzy danymi dla kolejnych wierzchołków (jeżeli mają być wczytane wszystkie dane z danego bloku, to przesunięcie wynosi zero i tak jest w naszym przypadku), wreszcie

`wskaznik_do_tablicy` oznacza wskaźnik do elementu tablicy, od którego mają być czytane dane (w naszym przypadku wartością tego parametru będzie wskaźnik na początek tablicy `wierzchołki`).

Analogiczna funkcja dla kolorów nosi nazwę `glColorPointer()`. Potrzebujemy zatem wywołać:

```
glVertexPointer(3, GL_FLOAT, 0, wierzcholki);
glColorPointer(3, GL_FLOAT, 0, kolory);
```

(c) Ostatnim etapem jest odwoływanie się do tych tablic. Istnieją tu trzy warianty:

- *Odwołanie się do konkretnego elementu* - funkcja `glArrayElement(indeks)`, gdzie `indeks` oznacza numer elementu do którego się odwołujemy. Po zdefiniowaniu powyższych tablic kod funkcji `kwadrat()` mógłby wyglądać następująco:

  ```
  void kwadrat()
  {
    glBegin(GL_QUADS);
      glArrayElement(0);
      glArrayElement(1);
      glArrayElement(2);
      glArrayElement(3);
    glEnd();
  }
  ```

  Każde z powyższych wywołań jednocześnie pobiera dane z obu zdefiniowanych tablic. Istotne jest tu, że przy pomocy funkcji `glArrayElement()` możemy wybrać dowolny podzbiór wierzchołków dla określenia daego obiektu (niekoniecznie kolejnych i niekoniecznie w kolejności rosnącej). Przykład ten zawarty jest w pliku `ogl43.cpp`.

- *Odwołanie się do listy indeksów w tablicach danych* - funkcja `glDrawElements(tryb, ilosc, typ, wskaznik_do_tablicy_indeksow)`, gdzie `tryb` przyjmuje takie same wartości jak parametr funkcji `glBegin()` (ćwiczenia nr 1), a `ilosc` oznacza ilość kolejnych wierzchołków użytych do konstrukcji obiektu określonego przez `tryb`. Dwa ostatnie parametry wymagają dodatkowego komentarza. Otóż funkcja `glDrawElements()` nie odwołuje się bezpośrednio do tablic wierzchołków lub kolorów, ale do *tablic zawierających indeksy do tych tablic*. Oznacza to, że oprócz tablic z danymi musimy wyspecyfikować tablice zawierające indeksy do podzbiorów tych tablic. Przykład tablicy czteroelementowej dla kwadratu nie jest tu zbyt poglądowy, ale trywialnie można by to w tym przypadku zrealizować następująco:

  ```
  unsigned short indeksy[]={0,1,2,3};
  ```

Wówczas `wskaznik_do_tablicy_indeksow` będzie miał wartość `indeksy`, natomiast parametr `typ` przyjmuje jedną z wartości typów całkowitoliczbowych bezznakowych: `GL_UNSIGNED_BYTE, GL_UNSIGNED_SHORT, GL_UNSIGNED_INT` (w naszym przypadku `GL_UNSIGNED_SHORT`). W tym przypadku funkcja `kwadrat()` ma postać

```
void kwadrat()
{
  glDrawElements(GL_QUADS, 4, GL_UNSIGNED_SHORT, indeksy);
}
```

Równoważnie można to zapisać jako

```
void kwadrat()
{
  glBegin(GL_QUADS);
    glArrayElement(indeksy[0]);
    glArrayElement(indeksy[1]);
    glArrayElement(indeksy[2]);
    glArrayElement(indeksy[3]);
  glEnd();
}
```

Przykład pokazujący to rozwiązanie jest zwaraty w pliku `ogl44.cpp`. Znacznie bardziej sugestywnym przykładem może być tutaj konstrukcja sześcianu: sześć tablic indeksów oznaczających numery wierzchołków z tablic danych - każda z tablic indeksów definiuje jedną ścianę.

- *Odwołanie się do listy kolejnych elementów* - funkcja `glDrawArrays(tryb, pierwszy, ilosc)`, gdzie `tryb` jest taki jak w poprzednim wariancie, `pierwszy` oznacza indeks elementu startowego w tablicach danych, a `ilość` jest ilością kolejnych (począwszy od startowego) elementów użytych z danej tablicy. W tym przypadku funkcja `kwadrat()` ma postać:

  ```
  void kwadrat()
  {
    glDrawArrays(GL_QUADS, 0,4);
  }
  ```

  Pokazuje to kod w pliku `ogl45.cpp`.

3. *Uwaga:* Istnieje jeszcze jeden sposób konstruowania tablic danych - zamiast kilku tablic (po jednej dla każdego rodzaju danych) konstruuje się jedną wspólną (tzw. *tablicę z przeplotem*), zawierającą dane róznego rodzaju, np. jednocześnie geometrię wierzchołków i ich kolory. Zajmiemy się nimi na kolejnych ćwiczeniach.

4. *Zadanie*

   Zdefiniować funkcje `czworościan()` oraz `prostopadłościan()` konstruującą te bryły przy pomocy tablic zawierających dane o geometrii wierzchołków i ich kolorach. Przetestować wszystkie warianty odwoływania się do tablic. Obie bryły mają jeden wierzchołek w środku układu i trzy krawędzie rozpięte wzdłuż dodatnich osi układu współrzędnych. Uwzględnić funkcję `glutReshapeFunc()`.