Grafika komputerowa i komunikacja człowiek - komputer

Open GL – interakcja z użytkownikiem

25.11.2022r.

Maciej Radecki
album: 253257

Prowadzący: dr inż. Jan Nikodem

Termin: Piątek TN, 16:25

1. Wstęp

Celem laboratorium było zapoznanie się ze sposobami interakcji użytkownika z obrazem, pozwalającej na sterowanie ruchem obiektu, jak i położeniem obserwatora, za pomocą funkcji biblioteki OpenGL z rozszerzeniem GLUT.

Do takiej manipulacji potrzebne jest pojęcie rzutowania perspektywistycznego. Poprzednio zetknęliśmy się z szczególnym przypadkiem rzutowania równoległego – rzutem ortograficzny, w OpenGL realizowanym za pomocą funkcji glOrtho(). W rzucie ortograficznym rzutnia była równoległa do płaszczyzny tworzonej przez osie x i y, a proste rzutowania biegły równolegle do osi z. Wskutek tego oś z była niewidoczna, a przesuwanie obiektu wzdłuż niej nie dawało widocznego rezultatu. Aby to naprawić, należy wykorzystać rzutowanie perspektywistyczne, które ponadto umożliwia na lepszą prezentację na płaszczyźnie geometrii trójwymiarowego obiektu.

Implementacja rzutowania perspektywistycznego wygląda następująco: należy określić 3 współrzędne x, y i z obserwatora, określające położenie obserwatora. Następnie, za pomocą funkcji gluLookAt() definiujemy położenie obserwatora trzymającego hipotetyczną kamerę (pierwsze trzy argumenty – GLdouble eyeX, GLdouble eyeY, GLdouble eyeZ) określają współrzędne punktu, w którym znajduje się obserwator. Trzy kolejne (GLdouble centerX, GLdouble centerY, GLdouble centerZ) definiują punkt, na który patrzy obserwator, natomiast ostatnie trzy (GLdouble upX, GLdouble upY, GLdouble upZ) pozwala na podanie skrócenia trzymanej przez obserwatora kamery. Hipotetyczną kamerę ustawiamy tak, że jej oś y pokrywa się z osią y układu współrzędnych (więc upX, upY, upZ) = (0, 1, 0). Aby użyć rzutu perspektywistycznego, w miejscu funkcji glOrtho() zastosowujemy funkcję gluPerspective(), służącej do zdefiniowania obszaru (bryły) widoczności dla rzutu perspektywistycznego. Argumenty tej funkcji (GLdouble fovy, GLdouble aspect, GLdouble zNear, GLdouble zFar) służą do określenia granic, w jakich elementy wchodzące w skład opisu sceny będą widoczne na widocznym obrazie. Wyobraźmy sobie, że mamy punkt, z którego patrzymy na dwa prostokąty – bliższy, mniejszy oraz dalszy, większy. Zmienna zNear opisuje dystans do bliższego prostokąta, zFar do dalszego, aspect opisuje stosunek długości szerokości prostokąta (długości dolnego boku) do jego wysokości (długości prawego boku), natomiast zmienna fovy oznacza pole widzenia obserwatora w kierunku y.

Podsumowując, aby na przykład obrócić jakiś obiekt o 45◦, należy zmienić wartości zmiennych (upX, upY, upZ) z (0, 1, 0) na (1, 1, 0). Aby zmienić położenie kamery, należy zmienić wartości pierwszych 3 zmiennych (eyeX, eyeY, eyeZ), np. aby lekko unieść i przesunąć kamerę w prawo, należy zmienić wartości opisanych wcześniej zmiennych z (0, 0, 10) na (3.0, 3.0, 10.0).

Z pomocą tych informacji możliwe jest wprowadzenie interakcji użytkownika z narysowanym obrazem przy użyciu myszy. Warto również wspomnieć, że ten ruch można wykonać na dwa sposoby: można obracać obiekt wokół osi przy ustalonym położeniu przy ustalonym położeniu obserwatora lub można obracać obserwatora wokół obiektu.

2. Realizacja ćwiczenia

Podczas laboratorium należało wykonać dwa zadania. Pierwsze z nich polegało na modyfikacji kodu z instrukcji laboratoryjnej w taki sposób, by:
- po uruchomieniu, czajnik powinien być rysowany czajnik na herbatę w położeniu początkowym,
- przy wciśniętym lewym klawiszu myszy, ruch kursora myszy w kierunku poziomym powinien

powodować obroty czajnika wokół osi y,
- przy wciśniętym lewym klawiszu myszy, ruch kursora myszy w kierunku pionowym powinien powodować obroty czajnika wokół osi x,
- przy wciśniętym prawym klawiszu myszy, ruchy kursora myszy w kierunku pionowym powinien wywoływać zbliżanie i oddalanie się obserwatora od czajnika (swoisty rodzaj funkcji zoom).

Drugie zadanie polegało na zmodyfikowaniu kodu z pierwszego zadania w taki sposób, aby możliwe było oglądanie modelu jajka z poprzedniego laboratorium (zamiast czajnika) ze zmieniającego się punktu widzenia, sterując myszą. Powinno być możliwe oddalenie się obserwatora od odległego o R obserwowanego punktu. Przy wciśniętym lewym klawiszu myszy ruch kursora w poziomie powinien obrócić obserwatora poziomo wokół jajka, natomiast ruch kursora w pionie powinien obrócić obserwatora pionowo wokół jajka. Oglądanie jajka ze zmieniającego się punktu widzenia nieco różni się od obracania samego jajka. Będziemy manipulować dwoma kątami, azymutem (Θ) oraz kątem elewacji (Φ). Pierwszy określa kierunek patrzenia obserwatora na obiekt. Drugi określa pośrednio wysokość obserwatora nad hipotetycznym horyzontem.

Uwzględniając te niuanse, dokładna treść zadania jest następująca:
- po uruchomieniu programu, jajko powinno zostać narysowane w położeniu początkowym,
- przy wciśniętym lewym klawiszu myszy, ruch kursora myszy w kierunku poziomym powinien powodować proporcjonalną zmianę azymutu kąta Θ,
- przy wciśniętym lewym klawiszu myszy, ruch kursora myszy w kierunku pionowym powinien powodować proporcjonalną zmianę kąta elewacji Φ,
- przy wciśniętym prawym klawiszu myszy, ruchy kursora myszy w kierunku pionowym winien realizować zmianę promienia R.

Aby w ogóle możliwa była interakcja myszki z obrazem, należy wprowadzić parę elementów. Po pierwsze, należy wprowadzić kilka funkcji zmiennych globalnych – kąt obrotu obiektu, przelicznik pikseli na stopnie, zmienną przechowującą aktualny status myszy (1 – lewy klawisz myszy wciśnięty, 2 – prawy klawisz myszy wciśnięty), poprzednią pozycję kursora myszy na osi x oraz poprzednią pozycję kursora myszy na osi y, różnicę pomiędzy pozycją bieżącą i poprzednią kursora myszy na osi x, różnicę pomiędzy pozycją bieżącą i poprzednią kursora myszy na osi y. Zmienne te zostaną wykorzystane w funkcjach zwrotnych: Mouse() i Motion(). Pierwsza funkcja sprawdza, czy wciśnięto lewy lub prawy przycisk myszy – jeżeli tak, to zaktualizowana zostanie zmienna przechowująca status myszy o wartość 1 lub 2. W przeciwnym razie status myszy zostanie ustawiony na 0 (czyli nie został wciśnięty żaden klawisz myszy). Druga funkcja analizuje położenie myszy na osiach x i y w porównaniu z poprzednim stanem i aktualizuje zmienne globalne przechowujące ich wartości, a następnie rysuje na nowo obraz sceny. Po dodaniu tych funkcji do kodu, należy jeszcze funkcję ChangeSize() wzbogacić o zmienną pix2angle, która przelicza piksele na stopnie, a w funkcji RenderScene() – dodać kod, wykonywujący się tylko, jeżeli jest aktywny dany status myszy (np. tylko gdy wciśnięty jest prawy klawisz myszy). Będziemy za jego pomocą aktualizować kąt obrotu o kąt proporcjonalny do różnicy położeń kursora myszy, po czym obracać obiekt o nowy kąt theta_x lub theta_y za pomocą funkcji glRotatef(theta_x, 0.0, 1.0, 0.0) oraz glRotatef(theta_y, 0.0, 1.0, 0.0).

Zoom tutaj został zaimplementowany za pomocą modyfikacji jednej ze zmiennych odpowiedzialnych za położenie obserwatora – viewer[2]. Można tak zrobić, ponieważ w tym zadaniu położenie obserwatora i tak nigdy się nie zmieni – zmieniamy jedynie położenie obiektu, a oś Z jest w tym wypadku niewidoczna. Zmiana wyłącznie zmiennej viewer[2] jest

więc równoważna z przybliżaniem/oddalaniem od siebie obrazu. Zabezpieczyliśmy się przed zoomem poza czajnik za pomocą warunków (delta_y > 0) oraz (delta_y <= 0). Czajnik do herbaty jest rysowany za pomocą funkcji wbudowanej OpenGL z rozszerzeniem GLUT glutWireTeapot(3.0). Po wprowadzeniu do kodu z instrukcji takich zmian jesteśmy w stanie obracać czajnik za pomocą lewego przycisku myszy oraz przybliżać/oddalać się od niego za pomocą prawego przycisku myszy.

Aby wykonać drugie zadanie, musimy dodać nowe zmienne (np. rozdzielić pix2angle na pix2angle_x i pix2angle_y, zmiennej R (odległość obserwatora od obserwowanego punktu) oraz zestawu zmiennych theta_x1 i theta_y1). Oprócz nowych zmiennych, należało jeszcze wprowadzić zmiany w istniejących funkcjach. Przykładowo, zmodyfikowano determinujące położenie obserwatora zmienne viewer[], według opisanych wcześniej zależności: viewer[0] = observerXS(R, theta_x1, theta_y1); viewer[1] = observerYS(R, theta_y1); viewer[2] = observerZS(R, theta_x1, theta_y1); gdzie funkcje observerXS, observerYS, observerZS są autorskie i po prostu obliczają wartości xs, ys i zs, natomiast zmienne theta_x1 i theta_y1 określają o ile stopni w kierunku 5 poziomym/pionowym powinien być przesunięty obserwator (w zależności od tego, jak mocno użytkownik przesunie myszą w danym kierunku). Dzięki temu po każdym ruchu myszą położenie obserwatora jest odpowiednio aktualizowane. Zoom został ograniczony w taki sposób, by R (odległość od obserwowanego punktu do obserwatora) nie mogło spaść do wartości poniżej 1.0. Dzięki temu nie jest możliwe „przebicie się” na drugą stronę jajka.

```
float observerXS(float R, float azymut, float elewacja)
{
    return R * cos(azymut) * (float)cos(elewacja);
}

float observerYS(float R, float elewacja)
{
    return R * sin(elewacja);
}

float observerZS(float R, float azymut, float elewacja)
{
    return R * sin(azymut) * (float)cos(elewacja);
}
```

```
if (status == 1 && tryb)                    // jeś
{
    theta_x += delta_x * pix2angle; // modyfikacja
    theta_y += delta_y * pix2angle;
}  // do różnicy położeń kursora myszy
else if (status == 2 && tryb) {
    if (delta_y > 0) {
        viewer[2] += 0.1;
        punkt_obserwacji[2] += 0.25;
    }
    else {
        viewer[2] -= 0.1;
        punkt_obserwacji[2] -= 0.25;
    }
}
else if (status == 1 && !tryb) {
    theta_x1 += (delta_x * pix2angle/ 100.0f);    /
    theta_y1 += (delta_y * pix2angle / 100.0f);
    viewer[0] = observerXS(R, theta_x1, theta_y1);
    viewer[1] = observerYS(R, theta_y1);
    viewer[2] = observerZS(R, theta_x1, theta_y1);

}
else if (status == 2 && !tryb) {
    R += (delta_y * pix2angle) / 25.0f; //zabezpiec
    if (R < 1.0f) R = 1.0f;
    viewer[0] = observerXS(R, theta_x1, theta_y1);
    viewer[1] = observerYS(R, theta_y1);
    viewer[2] = observerZS(R, theta_x1, theta_y1);
}
```

Ponieważ obydwa zadania realizowane są z poziomu jednej aplikacji dodano zmienną tryb typu bool. Stanowi ona przełączalną flagę w którym trybie chcemy się obecnie znajdować – obrót obiektem czy ruch obserwatorem.