Grafika komputerowa i komunikacja człowiek-komputer

Patryk Jurkiewicz 263896

Ćwiczenie 4 – Interakcja z użytkownikiem

1. Cel ćwiczenia

   To ćwiczenie zostało zrealizowane w celu demonstracji praktycznych zastosowań funkcji z biblioteki OpenGL oraz rozszerzenia GLUT. Głównym celem jest ukazanie, jak można łatwo osiągnąć prostą interakcję w trójwymiarowym środowisku poprzez kontrolę ruchu obiektu i położenia obserwatora za pomocą myszy. Podczas tego zadania, mysz pełni rolę narzędzia do sterowania, umożliwiając użytkownikowi aktywne uczestnictwo w przestrzeni trójwymiarowej. Poprzez manipulację myszą, użytkownik będzie mógł zarządzać ruchem obiektu oraz dostosowywać położenie obserwatora, co wpłynie na perspektywę i doświadczenie wirtualnej przestrzeni. Dodatkowo, w trakcie ćwiczenia zostaną przedstawione metody prezentacji obiektów trójwymiarowych w rzucie perspektywicznym. To obejmuje techniki wizualizacyjne, które mają na celu lepsze zrozumienie i odbiór struktury oraz układu przestrzennego obiektów.

2. Rzutowanie

   W wcześniejszym podejściu stosowano rzutowanie równoległe, zwłaszcza jego typ znany jako rzut ortograficzny. Rzut ortograficzny był osiągany za pomocą określonej funkcji. W tym kontekście płaszczyzna rzutni, na której formował się obraz, była równoległa do konkretnej płaszczyzny utworzonej przez pewne osie, a linie rzutowania biegły równolegle do innej osi. Ze względu na równoległy kierunek linii rzutowania do jednej z osi, przemieszczenie obiektu wzdłuż tej osi nie miało wpływu na ostateczny obraz. Aby uwidocznić przemieszczenia obiektu we wszystkich kierunkach, zastosowano inny rodzaj rzutowania, znany jako perspektywiczny. Rzut perspektywiczny okazał się korzystniejszy nie tylko ze względu na możliwość przedstawienia przemieszczeń, ale także umożliwił lepsze oddanie geometrii trójwymiarowego obiektu na płaszczyźnie. Istnieje kilka różnych metod definiowania rzutu perspektywicznego. Jedna z nich wymaga podania pewnej liczby parametrów, które określają sposób patrzenia na obiekt oraz właściwości kamery. Mimo pozornej złożoności parametrów, przy pewnym doświadczeniu i wyobraźni taka koncepcja pozwala intuicyjnie i wygodnie manipulować perspektywą widoku.

3. Kod

```
static GLfloat R = 10.0;  // odległość od obserwatora do sceny
static GLfloat viewer[] = { 0.0, 0.0, 10.0 };  // pozycja obserwatora
static GLfloat punkt_obserwacji[] = { 0.0, 0.0, 0.0 };  // punkt, na który jest skierowany wzrok obser
static GLfloat theta_x = 0.0;  // kąt obrotu wokół osi y
static GLfloat theta_y = 0.0;  // kąt obrotu wokół osi x
static GLfloat theta_x1 = 0.0;  // kąt obrotu wokół osi y dla trybu niezależnego od punktu obserwacji
static GLfloat theta_y1 = 0.0;  // kąt obrotu wokół osi x dla trybu niezależnego od punktu obserwacji
static GLfloat pix2angle;  // współczynnik przeliczeniowy pikseli na kąty
static GLint status = 0;  // status interakcji myszą
static int x_pos_old = 0;  // poprzednia pozycja myszy w osi x
static int delta_x = 0;  // różnica pozycji myszy w osi x
static int y_pos_old = 0;  // poprzednia pozycja myszy w osi y
static int delta_y = 0;  // różnica pozycji myszy w osi y
static int z_pos_old = 0;  // poprzednia pozycja myszy w osi z
static int delta_z = 0;  // różnica pozycji myszy w osi z
```

Zmienne takie jak viewer, punkt_obserwacji, theta_x, i theta_y są kluczowe dla kontroli widoku trójwymiarowej sceny w programie. viewer przechowuje trójwymiarową pozycję obserwatora, natomiast punkt_obserwacji określa punkt, na który jest skierowany wzrok obserwatora. Zmienne theta_x i theta_y odpowiadają za kąty obrotu wokół osi y i x, odpowiednio, co wpływa na orientację obserwatora. Te parametry są kluczowe dla interakcji z użytkownikiem za pomocą myszy oraz kontrolowania perspektywy widoku sceny trójwymiarowej. Dodatkowe zmienne, takie jak pix2angle, status, x_pos_old, delta_x, y_pos_old, delta_y, z_pos_old, i delta_z, są używane do śledzenia interakcji myszy, takich jak przesuwanie czy obracanie widoku.

```
// Funkcja obliczająca współrzędną x sceny na podstawie azymutu i elewacji
float scenaX(float R, float azymut, float elewacja)
{
    return R * cos(azymut) * (float)cos(elewacja);
}

// Funkcja obliczająca współrzędną y sceny na podstawie elewacji
float scenaY(float R, float elewacja)
{
    return R * sin(elewacja);
}

// Funkcja obliczająca współrzędną z sceny na podstawie azymutu i elewacji
float scenaZ(float R, float azymut, float elewacja)
{
    return R * sin(azymut) * (float)cos(elewacja);
}
```

Te funkcje są odpowiedzialne za obliczanie współrzędnych punktu w trójwymiarowej przestrzeni na podstawie parametrów azymutu, elewacji i odległości od obserwatora.

scenaX(float R, float azymut, float elewacja): Ta funkcja oblicza współrzędną x punktu na scenie. Parametr R to odległość od obserwatora do sceny, azymut to kąt poziomy w płaszczyźnie poziomej (kąt poziomy od osi x), a elewacja to kąt pionowy (kąt pionowy od osi y). Funkcja wykorzystuje funkcje trygonometryczne, takie jak cos, do obliczenia współrzędnej x na podstawie azymutu i elewacji.

scenaY(float R, float elewacja): Funkcja ta oblicza współrzędną y punktu na scenie. Ponownie, R to odległość od obserwatora do sceny, a elewacja to kąt pionowy. Funkcja używa funkcji trygonometrycznej sin do obliczenia współrzędnej y na podstawie elewacji.

scenaZ(float R, float azymut, float elewacja): Ta funkcja oblicza współrzędną z punktu na scenie. Podobnie jak wcześniej, R to odległość od obserwatora do sceny, azymut to kąt poziomy, a elewacja to kąt pionowy. Wykorzystuje ona zarówno funkcję sin jak i cos do obliczenia współrzędnej z na podstawie azymutu i elewacji.

$$x_s(\Theta,\Phi) = R\cos(\Theta)\cos(\Phi)$$

$$y_s(\Theta,\Phi) = R\sin(\Phi)$$

$$z_s(\Theta,\Phi) = R\sin(\Theta)\cos(\Phi)$$

$$0 \le \Theta \le 2\pi$$

$$0 \le \Phi \le 2\pi$$

Rys 1. Wzory z dokumentacji, które były wykorzystane do tworzenia funkcji

W kontekście trójwymiarowej grafiki komputerowej, azymut i elewacja są używane do opisu kierunku patrzenia obserwatora. Azymut to kąt poziomy mierzony od pewnego ustalonego punktu (najczęściej od osi x) w kierunku przeciwnym do ruchu wskazówek zegara. Elewacja to kąt pionowy mierzony od poziomej płaszczyzny. Te parametry są istotne w kontekście określenia, w jakim kierunku obserwator patrzy na scenę trójwymiarową, co ma wpływ na widok i perspektywę.

```
// Funkcja obsługująca zdarzenia myszy
void Mouse(int btn, int state, int x, int y)
{
    if (btn == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
    {
        x_pos_old = x;
        y_pos_old = y;
        status = 1;
    }
    else if (btn == GLUT_RIGHT_BUTTON && state == GLUT_DOWN)
    {
        y_pos_old = y;
        status = 2;
    }
    else
        status = 0;
}
```

Ta funkcja Mouse odpowiada za obsługę zdarzeń myszy w programie. Po wywołaniu tej funkcji, parametry btn i state zawierają informacje o przycisku myszy (lewy, prawy, itp.) oraz o stanie przycisku (naciśnięty, zwolniony, itp.). Parametry x i y to aktualne współrzędne myszy na ekranie.

W funkcji sprawdzane jest, czy lewy przycisk myszy został naciśnięty (GLUT_LEFT_BUTTON i GLUT_DOWN). Jeśli tak, aktualne współrzędne myszy (x i y) są zapisywane jako x_pos_old i y_pos_old, a zmienna status ustawiana jest na 1. To oznacza, że użytkownik rozpoczął interakcję poprzez przesuwanie myszy.

W przypadku, gdy prawy przycisk myszy jest naciśnięty (GLUT_RIGHT_BUTTON i GLUT_DOWN), tylko współrzędna y myszy (y) jest zapisywana jako y_pos_old, a zmienna status ustawiana jest na 2. To wskazuje, że użytkownik rozpoczął interakcję poprzez przesuwanie myszy w kierunku pionowym.

Jeśli żaden z przycisków nie jest naciśnięty, zmienna status zostaje ustawiona na 0, co sygnalizuje brak interakcji z myszą.

```
// Funkcja obsługująca ruch myszy
void Motion(GLsizei x, GLsizei y)
{
    delta_x = x - x_pos_old;
    x_pos_old = x;
    delta_y = y - y_pos_old;
    y_pos_old = y;
    glutPostRedisplay();
}
```

Funkcja Motion jest odpowiedzialna za obsługę ruchu myszy w trakcie interakcji użytkownika. Jest wywoływana w momencie, gdy użytkownik porusza myszą po ekranie. Poniżej znajduje się wyjaśnienie poszczególnych elementów tej funkcji:

delta_x = x - x_pos_old; i delta_y = y - y_pos_old;: Te linie obliczają różnicę między aktualnymi współrzędnymi myszy (x i y) a poprzednimi współrzędnymi myszy (x_pos_old i y_pos_old). Wyniki są zapisywane w zmiennych delta_x i delta_y. Te wartości będą wykorzystane do określenia kierunku i siły ruchu myszy.

x_pos_old = x; i y_pos_old = y;: Po obliczeniu różnic, aktualne współrzędne myszy (x i y) są zapisywane jako poprzednie współrzędne myszy (x_pos_old i y_pos_old). To umożliwia aktualizację różnic w kolejnych krokach i śledzenie ruchu myszy.

glutPostRedisplay();: Ta linia kodu informuje system GLUT, że zawartość okna została zmieniona i należy przerysować scenę. Jest to zazwyczaj używane w kontekście aktualizacji widoku w odpowiedzi na ruch myszy.

Ogólnie rzecz biorąc, funkcja Motion śledzi zmiany pozycji myszy i zapisuje różnice w delta_x i delta_y. Te wartości są później wykorzystywane do odpowiedniego dostosowania widoku sceny, na przykład do obracania lub przesuwania obiektów w zależności od interakcji użytkownika.

```
void RenderScene(void)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    gluLookAt(viewer[0], viewer[1], viewer[2], punkt_obserwacji[0], punkt_obserwacji[1], punkt_obserwacji[2], 0.0, 1.0, 0.0);
    Axes();

    if (status == 1 && tryb)
    {
        theta_x += delta_x * pix2angle;
        theta_y += delta_y * pix2angle;
    }
    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] = scenaX(R, theta_x1, theta_y1);
        viewer[1] = scenaY(R, theta_y1);
        viewer[2] = scenaZ(R, theta_x1, theta_y1);
    }
    else if (status == 2 && !tryb)
    {
        R += (delta_y * pix2angle) / 25.0f;
        if (R < 1.0f) R = 1.0f;
        viewer[0] = scenaX(R, theta_x1, theta_y1);
        viewer[1] = scenaY(R, theta_y1);
        viewer[2] = scenaZ(R, theta_x1, theta_y1);
    }

    glRotatef(theta_x, 0.0, 1.0, 0.0);
    glRotatef(theta_y, 1.0, 0.0, 0.0);
    glColor3f(1.0f, 1.0f, 1.0f);
    Egg(N);
    glFlush();
    glutSwapBuffers();
}
```

Funkcja RenderScene stanowi kluczową część programu odpowiedzialną za renderowanie trójwymiarowej sceny. W pierwszym etapie, funkcja inicjalizuje rysowanie poprzez wyczyszczenie buforów kolorów i głębokości przy użyciu glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT) oraz przywrócenie jednostkowej macierzy model-widok za pomocą glLoadIdentity(). Następnie, ustawiana jest kamera przy użyciu funkcji gluLookAt, co definiuje pozycję obserwatora (viewer) oraz punkt, na który jest skierowany wzrok obserwatora (punkt_obserwacji).

W dalszej części funkcji, realizowana jest obsługa interakcji myszy. W zależności od stanu (status) oraz aktywnego trybu widoku (tryb), funkcja reaguje na ruchy myszy, modyfikując kąty obrotu (theta_x i theta_y) lub regulując odległość kamery od obiektu (viewer[2]) oraz jej punktu obserwacji (punkt_obserwacji[2]). W przypadku nieaktywnego trybu widoku, kąty i odległość kamery są modyfikowane w sposób umożliwiający niezależne obroty i przemieszczanie obserwatora.

Kolejnym krokiem jest obrót obiektów na scenie zgodnie z aktualnymi kątami obrotu (theta_x i theta_y) przy użyciu funkcji glRotatef. Następnie, funkcja wywołuje procedurę rysującą trójwymiarowe jajko, co jest zrealizowane przez funkcję Egg(N).

Na końcu, po przetworzeniu wszystkich parametrów i narysowaniu obiektów, klatka jest wysyłana do bufora wyświetlania poprzez glFlush() i glutSwapBuffers(). Dzięki tej funkcji, użytkownik ma możliwość dynamicznej interakcji z trójwymiarowym modelem, eksplorując go w czasie rzeczywistym za pomocą myszy.

4. Wnioski

Wykonując zadanie zgłębiliśmy kod programu, który umożliwia renderowanie trójwymiarowej sceny. Analizowaliśmy funkcje matematyczne odpowiedzialne za przekształcenia trójwymiarowych współrzędnych, takie jak scenaX, scenaY, i scenaZ, które uwzględniają azymut, elewację oraz odległość od obserwatora w procesie wyznaczania współrzędnych punktów na scenie.

Rozważaliśmy również zaawansowane mechanizmy obsługi interakcji myszy, widoczne w funkcjach Mouse i Motion, co pozwala na dynamiczne poruszanie się po scenie. Omówiliśmy logikę związaną z perspektywą w funkcji RenderScene, gdzie kamera dostosowuje się do zmian parametrów, takich jak kąty obrotu, odległość od obiektu, oraz punkt obserwacji.

W trakcie analizy kodu zauważyliśmy zastosowanie biblioteki GLUT do obsługi okien i interakcji oraz implementację różnych trybów widoku oraz modeli obiektów trójwymiarowych. Nasza eksploracja tego programu pozwoliła na lepsze zrozumienie podstawowych koncepcji grafiki komputerowej, takich jak przekształcenia geometryczne, interakcje myszy, oraz perspektywa w renderowaniu trójwymiarowych scen. W rezultacie, zyskaliśmy wgląd w budowę interaktywnego programu graficznego, zdolnego do generowania trójwymiarowych renderów z użyciem prostych matematycznych transformacji.