Grafika komputerowa i komunikacja człowiek komputer

Laboratorium 3

Interakcja z użytkownikiem

Wykonał:

Ivan Hancharyk 264511

Termin zajęć:

WT/TP/8:00

1. Cel ćwiczenia

Celem ćwiczenia jest zapoznanie się z metodami realizowania prostej interakcji z użytkownikiem przy pomocy funkcji biblioteki OpenGL, polegającej na sterowaniu ruchem obiektu i położeniem obserwatora w przestrzeni 3D.

2. Wstęp teoretyczny

Do realizacji zadania wykorzystano biblioteki OpenGL oraz GLUT, które umożliwiają tworzenie interaktywnej grafiki trójwymiarowej. Kluczowymi funkcjami są:

- **gluLookAt**(eyex, eyey, eyez, centerx, centery, centerz, upx, upy, upz) – służy do ustawiania pozycji kamery oraz określenia kierunku patrzenia.
- **gluPerspective**(fovy, aspect, zNear, zFar) – definiuje obszar widoczności w rzucie perspektywicznym.
- **glEnable**(GL_DEPTH_TEST) – włącza mechanizm usuwania elementów niewidocznych na scenie 3D.

3. Realizacja zadania

3.1. Zadanie główne

W ramach zadania głównego zaimplementowano możliwość sterowania widokiem oraz ruchem obiektu przy użyciu myszy:

- Obracanie obiektu wokół osi przy ustalonym położeniu obserwatora.
- Sterowanie obserwatorem, umożliwiając przybliżenie oraz oddalenie obiektu

Ta funkcja obsługuje kliknięcia przycisków myszy. Sprawdza, który przycisk został wciśnięty oraz jego stan (wciśnięty lub zwolniony). Jeśli wciśnięto lewy przycisk myszy, zapisuje pozycję kursora i ustawia status `statusLeft` na 1, co oznacza, że wciśnięto lewy przycisk. Ponadto, funkcja obsługuje kółko myszy do zoomowania (przybliżania i oddalania widoku). W zależności od kierunku ruchu kółka (przycisk 3 lub 4), zmienia wartość `rViewer`, co wpływa na odległość obserwatora od obiektu.

```
void MyInit(void) {
    glClearColor( red: 0.1f,  green: 0.1f,  blue: 0.1f,  alpha: 1.0f);
    glEnable( cap: GL_DEPTH_TEST);
    glEnable( cap: GL_LIGHTING);
    glEnable( cap: GL_LIGHT0);
    glEnable( cap: GL_COLOR_MATERIAL);
    glPointSize( size: 1.0);
    glLineWidth( width: 2.0);

    GLfloat light_diffuse[] = { [0]: 1.0,  [1]: 1.0,  [2]: 1.0,  [3]: 1.0 };
    GLfloat light_ambient[] = { [0]: 0.2,  [1]: 0.2,  [2]: 0.2,  [3]: 1.0 };
    glLightfv( light: GL_LIGHT0,  pname: GL_DIFFUSE,  params: light_diffuse);
    glLightfv( light: GL_LIGHT0,  pname: GL_AMBIENT,  params: light_ambient);
}
```

Funkcja obsługuje także kółko myszy, które pozwala na przybliżanie (`zoom in`) lub oddalanie (`zoom out`) kamery względem obiektu. Jeżeli kółko myszy jest przesuwane do góry (`btn == 3`), zmniejszana jest wartość zmiennej `rViewer`, co przybliża widok. Natomiast jeśli kółko jest przesuwane w dół (`btn == 4`), `rViewer` jest zwiększane, co powoduje oddalenie widoku. Dodatkowo wprowadzono ograniczenia (`MIN_VIEWER_DISTANCE` oraz `MAX_VIEWER_DISTANCE`), aby zapobiec zbyt dużemu zbliżeniu lub oddaleniu od obiektu. Po każdej zmianie wartości `rViewer`, scena jest przerysowywana za pomocą `glutPostRedisplay()`.

```
void Mouse(int btn, int state, int x, int y) {
    if (btn == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {
        x_pos_old = x; // przypisanie aktualnie odczytanej pozycji kursora
        y_pos_old = y;
        statusLeft = 1; // wciśnięty został lewy klawisz myszy
    } else {
        statusLeft = 0; // lewy klawisz nie jest wciśnięty
    }

    // Obsługa kółka myszy do zoomowania
    if (btn == 3) { // Scroll up (zoom in)
        rViewer -= 1.0f;
        if (rViewer < MIN_VIEWER_DISTANCE) {
            rViewer = MIN_VIEWER_DISTANCE;
        }
        glutPostRedisplay();
    }
    if (btn == 4) { // Scroll down (zoom out)
        rViewer += 1.0f;
        if (rViewer > MAX_VIEWER_DISTANCE) {
            rViewer = MAX_VIEWER_DISTANCE;
        }
        glutPostRedisplay();
    }
}
```

Funkcja ta jest wywoływana podczas ruchu myszy, gdy wciśnięty jest lewy przycisk. Oblicza różnicę między bieżącą a poprzednią pozycją kursora, a następnie aktualizuje odpowiednie kąty (`azymuth` i `elevation`), które odpowiadają za obrót widoku w płaszczyźnie poziomej i pionowej. Ograniczenia są stosowane na kąt elewacji, aby kamera nie mogła obracać się zbyt daleko poza zakres.

```
void Motion(GLsizei x, GLsizei y) {
    delta_x = x - x_pos_old;
    delta_y = y - y_pos_old;
    x_pos_old = x;
    y_pos_old = y;

    if (statusLeft == 1) { // Obrót obiektu
        azymuth += delta_x * pix2angle_x / 20.0;
        elevation += delta_y * pix2angle_y / 20.0;

        // Ograniczenia obrotu w pionie, aby kamera nie obracała się zbyt daleko
        if (elevation > 1.5) elevation = 1.5;
        if (elevation < -1.5) elevation = -1.5;
    }

    glutPostRedisplay();
}
```

Funkcja RenderScene odpowiada za rysowanie sceny i aktualizację widoku. Najpierw scena jest czyszczona (glClear()) i resetowana (glLoadIdentity()). Pozycja kamery (camX, camY, camZ) jest obliczana na podstawie azymutu, elewacji i odległości (rViewer), a gluLookAt() ustawia widok kamery na środek sceny.

Światło jest ustawiane za pomocą glLightfv(), a osie współrzędnych rysowane przez Axes(). Funkcje glScalef() i glRotatef() są używane do skalowania i obracania obiektu, po czym Draw() rysuje model. Na koniec glutSwapBuffers() zapewnia płynne wyświetlanie grafiki.

```
void RenderScene(void) {
    glClear( mask: GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();

    // Obliczenie pozycji kamery na podstawie azymutu, elewacji i odległości
    float camX = rViewer * cos( x: azymuth) * cos( x: elevation);
    float camY = rViewer * sin( x: elevation);
    float camZ = rViewer * sin( x: azymuth) * cos( x: elevation);

    // Ustawienie kamery z użyciem gluLookAt() - zawsze skierowana na środek sceny
    gluLookAt( eyex: camX,  eyey: camY,  eyez: camZ,
               centerx: 0.0,  centery: 0.0,  centerz: 0.0,
               upx: 0.0,  upy: 1.0,  upz: 0.0);

    // Ustawienie światła
    GLfloat light_position[] = {  [0]: 0.0,  [1]: 10.0,  [2]: 10.0,  [3]: 1.0 };
    glLightfv( light: GL_LIGHT0,  pname: GL_POSITION,  params: light_position);

    Axes(); // Narysowanie osi
    glScalef( x: scale,  y: scale,  z: scale);
    glRotatef( angle: theta[0],  x: 1.0,  y: 0.0,  z: 0.0);
    glRotatef( angle: theta[1],  x: 0.0,  y: 1.0,  z: 0.0);
    glRotatef( angle: theta[2],  x: 0.0,  y: 0.0,  z: 1.0);
    Draw();
    glFlush();
    glutSwapBuffers();
}
```

3.2. Poprawa zadania poprzedniego

W poprzedniej wersji kodu występowały anomalie w wyświetlaniu jajka, szczególnie na jego górnej i dolnej części, co powodowało, że obiekt nie wyglądał gładko i spójnie. W nowej wersji kodu wprowadzono poprawki polegające na odzwierciedleniu (lustrzanym odbiciu) trójkątów wzdłuż osi Z, co pozwoliło uzyskać symetryczny i pełny model jajka bez błędów wizualnych. Dzięki temu górna i dolna część jajka są teraz poprawnie zdefiniowane, a cała powierzchnia obiektu jest rysowana w sposób równomierny, co eliminuje wcześniejsze defekty.

```
for (int i = 0; i < N - 1; i++) {
    for (int j = 0; j < N / 2; j++) {
        glBegin(GL_TRIANGLES);

        // Pierwszy trójkąt
        glColor3f(colors[i][j].x, colors[i][j].y, colors[i][j].z);
        glVertex3f(points[i][j].x, points[i][j].y, points[i][j].z);

        glColor3f(colors[i + 1][j].x, colors[i + 1][j].y, colors[i +
1][j].z);
        glVertex3f(points[i + 1][j].x, points[i + 1][j].y, points[i +
1][j].z);
```

```
        glColor3f(colors[i][j + 1].x, colors[i][j + 1].y, colors[i][j +
1].z);
        glVertex3f(points[i][j + 1].x, points[i][j + 1].y, points[i][j +
1].z);

        glEnd();

        glBegin(GL_TRIANGLES);

        // Drugi trójkąt
        glColor3f(colors[i][j + 1].x, colors[i][j + 1].y, colors[i][j +
1].z);
        glVertex3f(points[i][j + 1].x, points[i][j + 1].y, points[i][j +
1].z);

        glColor3f(colors[i + 1][j].x, colors[i + 1][j].y, colors[i +
1][j].z);
        glVertex3f(points[i + 1][j].x, points[i + 1][j].y, points[i +
1][j].z);

        glColor3f(colors[i + 1][j + 1].x, colors[i + 1][j + 1].y, colors[i +
1][j + 1].z);
        glVertex3f(points[i + 1][j + 1].x, points[i + 1][j + 1].y, points[i +
1][j + 1].z);

        glEnd();

        // Lustrzane odbicie trójkątów
        glBegin(GL_TRIANGLES);

        // Pierwszy trójkąt odbity
        glColor3f(colors[i][j].x, colors[i][j].y, colors[i][j].z);
        glVertex3f(points[i][j].x, points[i][j].y, -points[i][j].z);

        glColor3f(colors[i + 1][j].x, colors[i + 1][j].y, colors[i +
1][j].z);
        glVertex3f(points[i + 1][j].x, points[i + 1][j].y, -points[i +
1][j].z);

        glColor3f(colors[i][j + 1].x, colors[i][j + 1].y, colors[i][j +
1].z);
        glVertex3f(points[i][j + 1].x, points[i][j + 1].y, -points[i][j +
1].z);

        glEnd();

        glBegin(GL_TRIANGLES);

        // Drugi trójkąt odbity
        glColor3f(colors[i][j + 1].x, colors[i][j + 1].y, colors[i][j +
1].z);
        glVertex3f(points[i][j + 1].x, points[i][j + 1].y, -points[i][j +
1].z);

        glColor3f(colors[i + 1][j].x, colors[i + 1][j].y, colors[i +
1][j].z);
        glVertex3f(points[i + 1][j].x, points[i + 1][j].y, -points[i +
1][j].z);

        glColor3f(colors[i + 1][j + 1].x, colors[i + 1][j + 1].y, colors[i +
1][j + 1].z);
        glVertex3f(points[i + 1][j + 1].x, points[i + 1][j + 1].y, -points[i
+ 1][j + 1].z);
```

```
            glEnd();
        }
}
```

4. Wnioski

W trakcie wykonywania ćwiczenia zapoznano się z technikami obracania obiektów w przestrzeni 3D, z funkcjami umożliwiającymi interakcję z użytkownikiem oraz zastała poprawiona implementacja rysowania jajka trójkątami. Wykorzystanie bibliotek OpenGL oraz GLUT znacząco ułatwiło implementację funkcji związanych z obsługą myszy i rzutowaniem perspektywicznym. Dzięki tym narzędziom możliwe jest wygodne manipulowanie obiektami 3D oraz obserwatorem, co jest kluczowe w tworzeniu interaktywnej grafiki komputerowej.