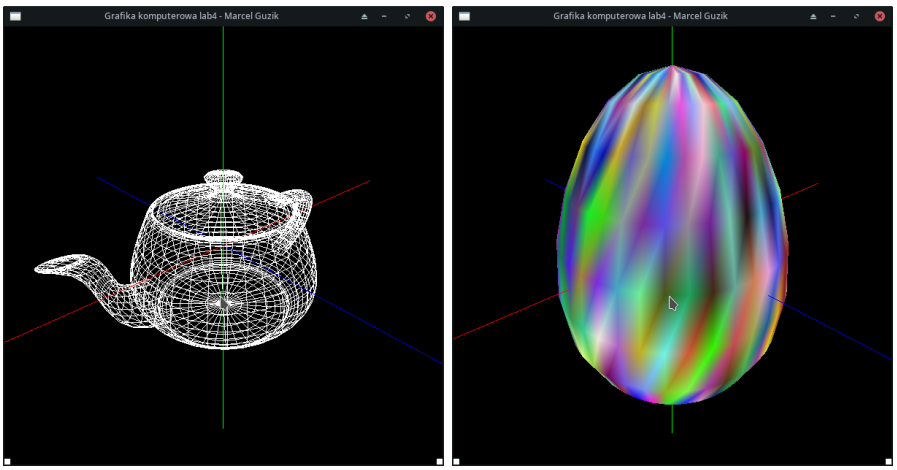
1 Wprowadzenie

Ćwiczenie ma za zadanie pokazać, jak przy pomocy funkcji biblioteki OpenGL z biblioteką GLUT można zrealizować prostą interakcję, polegające na sterowaniu ruchem obiektu i położeniem obserwatora w przestrzeni 3D. Do sterowania służyla będzie mysz. Ponadto zostaną zilustrowanie sposoby prezentacji obiektów trójwymiarowych w rzucie perspektywicznym.

Finalny program realizuje:

- Po uruchomieniu programu, w położeniu początkowym rysowane jest jajko
- 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.
- Możliwa jest zmiana rysowanego modelu pomiędzy różnymi modelami jaja oraz czajnikiem.



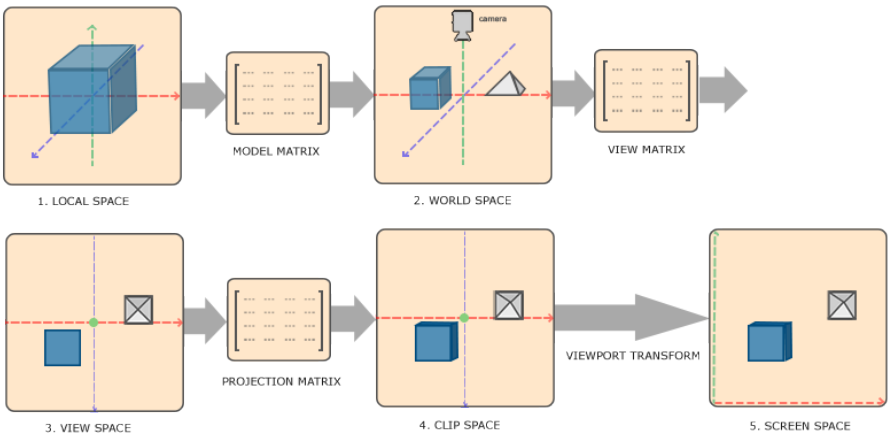
Rysunek 1: Widok czajnika oraz jaja

Uwaga: pojęcia "obserwator" oraz "kamera" są używane wymiennie.

2 Teoria

2.1 Stos transformacji macierzowych

Aby wyświetlić przestrzeń trójwymiarową na dwuwymiarowym ekranie komputera, niezbędne jest wykonanie przekształceń systemów współrzędnych, które przetransformują współrzędne rysowanych obiektów na znormalizowane współrzędne urządzenia (NDC). Możemy wykonywać te przekształcenia krokowo, za pomocą mnożenia przez siebie kolejnych macierzy. Diagram przekształceń widoczny jest na rysunku 2.



Rysunek 2: Kolejne kroki przekształceń systemu współrzędnych za pomocą transformacji macierzowych

1. Współrzędne lokalne to współrzędne obiektu relatywne dla niego samego. Np. dla naszego jajka początek układu lokalnych współrzędnych znajduje się u jego podstawy. Musimy zatem wykonać translację za pomocą macierzy modelu by wycentrować jajko w naszym układzie współrzędnych.

2. Następnym krokiem jest transformacja współrzędnych lokalnych do współrzędnych przestrzeni świata, gdzie współrzędne opisują pozycję obiektów w świecie. Są one relatywne do arbitralnie wybranego początku układu współrzędnych.

3. Następnie przekształcamy współrzędne przestrzeni świata do współrzędnych przestrzeni widoku za pomocą macierzy widoku. Jest to przestrzeń, w której początkiem układu współrzędnych jest pozycja obserwatora i kierunek w którym jest on zwrócony. W tym miejscu nie jest jeszcze zdefiniowany zakres widoczności obserwatora, więc żadne wierzchołki nie są usuwane.

4. Przestrzeń widoku chcemy następnie przyciąć tak, by tylko obiekty widziane przez obserwatora znajdowały się w zdefiniowanej przestrzeni oraz by ograniczyć zakres współrzędnych do $(-1.0, 1.0)$, czyli tak by wyrazić tą przestrzeń w znormalizowanych współrzędnych urządzenia (Normalized Device Coordinates) możliwych do wyświetlenia przez OpenGL.

5. Jako ostatni krok, przekształcamy "przycięte" współrzędne do współrzędnych ekranu (czyli upraszczając, po prostu w piksele).

Na rysunku widoczne są 3 różne macierze, jednak w trybie Fixed Function Pipeline w OpenGL mamy dostęp do tylko 2 macierzy: `GL_MODELVIEW` oraz `GL_PROJECTION`. Jak wskazuje nazwa, macierze modelu oraz widoku są ze sobą połączone.

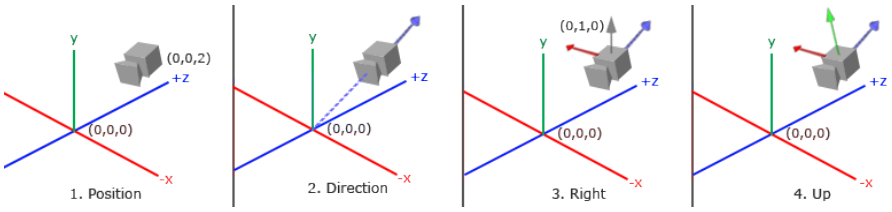
Transformacje z przestrzeni "uciętej" do przestrzeni ekranu przez funkcję `glViewport()` były opisywane w sprawozdaniu do ćwiczenia nr 2, dlatego nie będą opisywane w niniejszym sprawozdaniu.

Aby zaimplementować ruszającą się kamerę, niezbędne będzie wykonanie macierzy widoku, tudzież macierzy `GL_MODELVIEW` oraz macierzy rzutowania. Macierz modelu i widoku ustawia funkcja `gluLookAt()`, natomiast macierz rzutowania ustawia funkcja `gluPerspective()`.

2.2 Pozycja kamery i macierz widoku

Do zdefiniowania pozycji kamery potrzebujemy czterech wektorów:

- Wektor pozycji kamery w systemie współrzędnych świata: p
- Wersor definiujący kierunek w który skierowana jest kamera: $\hat{z}$
- Wersor osi x: $\hat{x}$
- Wersor osi y: $\hat{y}$



Rysunek 3: Proces uzyskiwania wektorów położenia oraz wersorów $\hat{x}$, $\hat{y}$, $\hat{z}$ kamery

Zadaniem macierzy widoku jest przetransformowanie systemu współrzędnych w taki sposób, by nowy system opisywał położenie obiektów tak, jak są one widoczne z punktu widzenia obserwatora. W tym celu należy zastosować kombinację translacji, skalowania oraz rotacji, które mają spełnić następujące warunki:

- Obserwator znajduje się w początku nowego układu współrzędnych.
- Obserwator skierowany jest w stronę danego punktu (inaczej mówiąc: oś z przebiega przez dany punkt oraz obserwatora)
- System współrzędnych obracany jest zgodnie z wartościami nowych wektorów x oraz y ustalonych przez obserwatora.

Efektem jest następująca macierz która realizuje transformację do systemu współrzędnych widoku:

$$\begin{bmatrix} \hat{x}_x & \hat{x}_y & \hat{x}_z & 0 \\ \hat{y}_x & \hat{y}_y & \hat{y}_z & 0 \\ \hat{z}_x & \hat{z}_y & \hat{z}_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & -P_x \\ 0 & 1 & 0 & -P_y \\ 0 & 0 & 1 & -P_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rysunek 4: Macierz widoku

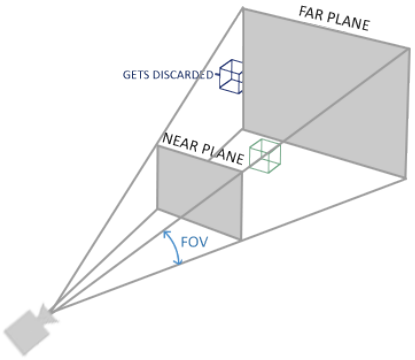
Gdzie lewa macierz jest odpowiedzialna za rotację, a prawa za translację.

2.3 Rzutowanie perspektywistyczne

W poprzednim ćwiczeniu stosowano rzutowanie równoległe, a dokładniej jego szczególny przypadek zwany rzutem ortograficznym. Do realizacji rzutowania ortograficznego służy funkcja `glOrtho()`. W rzucie ortograficznym rzutnia, czyli płaszczyzna na której powstawał obraz, była równoległa do płaszczyzny tworzonej przez osie x i y, a proste rzutowania biegły równoległe do osi z.

Ze względu na to, że proste rzutowania są równoległe do osi z, przesuwanie obiektu wzdłuż tej osi nie spowoduje żadnego efektu na obrazie. Aby umożliwić pokazanie efektów przemieszczeń obiektu we wszystkich osiach nalely zastosować rzutowanie perspektywiczne. Rzut perspektywiczny jest lepszy od równoległego nie tylko ze względu na możliwość prezentacji przemieszczeń, pozwala także lepiej pokazać na płaszczyźnie geometrii trójwymiarowego obiektu.

Poniżej znajduje się rysunek bryły widoczności zdefiniowanej przez funkcję `gluPerspective()`. Bryły znajdujące się w całości poza bryłą widoczności kamery nie są rysowane.



Rysunek 5: Bryła widoczności zdefiniowana przez macierz rzutu perspektywistycznego

2.4 Rotacja

Efektywnie, rotacja realizowana przez program składa się z dwóch elementów:

1. **Zakrzywienie ruchu obserwatora.** Obserwator pozostaje w zdefiniowanej odległości R od środka układu współrzędnych. Poprzez ruch myszą w dwóch dostępnych osiach, obserwator porusza się ruchem kątowym za pomocą dwóch kątów, azymutu Θ, oraz elewacji Φ.

   System równań określający możliwe pozycje obserwatora wygląda następująco:

$$x_s(\Theta, \Phi) = R\cos(\Theta)\cos(\Phi)$$
$$y_s(\Theta, \Phi) = R\sin(\Phi)$$
$$z_s(\Theta, \Phi) = R\sin(\Theta)\cos(\Phi)$$
$$0 \le \Theta \le 2\pi$$
$$0 \le \Phi \le 2\pi$$

Rysunek 6: System równań określający pozycję obserwatora

2. **Obrót obserwatora w kierunku początku układu współrzędnych.** Ta część zrealizowana jest w funkcji `gluLookAt()` i jest częścią macierzy widoku. Przekształca ona system współrzędnych, możliwie obracając go tak, by osie x oraz y odpowiadały osiom x oraz y powierzchni ekranu, oraz by oś z układała się "wgłąb" ekranu.

3 Wykonanie programu

Przekształcenie przestrzeni świata w przestrzeń widoku wykonuje funkcja `gluLookAt()`. Przyjmuje ona jako argumenty:

- wektor określający pozycję obserwatora
- punkt w kierunku którego zwrócony jest obserwator
- wektor określający kierunek górny z punktu widzenia obserwatora

Zakładając wywołanie `gluLookAt(p2, t, u)`, gdzie `p2` to pozycja obserwatora, `t` to obiekt w stronę którego zwrócony jest obserwator, a `u` to wektor definiujący kierunek górny we współrzędnych świata, funkcja uzyskuje argumenty w sposób następujący:

$$p = p2$$
$$\hat{z} = \frac{p - t}{|p - t|}$$
$$\hat{x} = \frac{u \times \hat{z}}{|u \times \hat{z}|}$$
$$\hat{y} = \hat{z} \times \hat{x}$$

Przekształcenie przestrzeni widoku w przestrzeń "uciętą" wykonuje funkcja `gluPerspective()`. Jest ona zdefiniowana następująco:

```
void gluPerspective(double fovy, double aspect, double zNear, double zFar)
```

Kolejne argumenty to:

- Pole widzenia w osi y
- Proporcje osi x do osi y

- Minimalna odległość rysowania w osi z
- Maksymalna odgległość rysowania w osi z

Ponadto, dalsze wytłumaczenie nowego kodu jest dostępne w postaci komentarzy. Nowy kod źródłowy:

```
#define N 20

int model = 1;
const float EPSILON = 0.000001;

// inicjalizacja położenia obserwatora
static float viewer[] = { 0.0, 0.0, 10.0 };

// kąt obrotu obiektu
static float angleY = 0.5 * M_PI;
static float angleX = 0.0;
static float zstep = 0.1;

// przelicznik pikseli na stopnie
static GLfloat pix2angle;

// stan klawiszy myszy
// 0 - nie naciśnięto żadnego klawisza
// 1 - naciśnięty zostać lewy klawisz
static GLint status = 0;

// poprzednia pozycja kursora myszy w osiach x i y
static int x_pos_old = 0;
static int y_pos_old = 0;

// różnica pomiędzy pozycją bieżącą i poprzednią kursora myszy
static int delta_x = 0;
static int delta_y = 0;

// odległość obserwatora od początku układu współrzędnych
float R = 10.0;

// Funkcja blokuje wartość d tak, by znajdowała się w zakresie (min, max), tzn.
// jeżeli d > max to d = max, jeżeli d < min to d = min
float clamp(float d, float min, float max) {
    const float t = d < min ? min : d;
    return t > max ? max : t;
}

// Dodatnia reszta z dzielenia przez siebie dwóch liczb zmiennoprzecinkowych.
// Potrzebne do "zawijania" kąta theta
float fmodfp(float a, float b) {
    if (a < 0) {
        a += b;
```

```
    }
    else if (a > b) {
        a -= b;
    }
    return a;
}

void renderScene() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    // Zdefiniowanie położenia obserwatora
    gluLookAt(viewer[0], viewer[1], viewer[2], 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);

    Axes();

    if (status == 1) {
        // tutaj `angleY` odnosi się do kąta dookoła osi Y (theta)
        // a `angleX` do kąta dookoła osi X, czyli azymutu (phi)
        // natomiast `delta_{x,y}` odnosi się do osi powierzchni ekranu
        angleY = fmodfp(angleY + delta_x * 0.02, 2 * M_PI);
        angleX = fmodfp(angleX + delta_y * 0.02, 2 * M_PI);

        // blokujemy przekroczenie osi y od górnej strony
        if (0.5 * M_PI < angleX && angleX < 1.0 * M_PI) {
            angleX = 0.5 * M_PI - EPSILON;
        }
        // blokujemy przekroczenie osi y od dolnej strony
        else if (1.0 * M_PI < angleX && angleX < 1.5 * M_PI) {
            angleX = 1.5 * M_PI + EPSILON;
        }
    }
    else if (status == 2) {
        // zmieniamy odległość obserwatora od początku układu współrzędnych
        R += delta_y * zstep;
    }

    // wyliczenie współrzędnych następnego położenia obserwatora
    viewer[0] = R * cos(angleY) * cos(angleX);
    viewer[1] = R * sin(angleX);
    viewer[2] = R * sin(angleY) * cos(angleX);

    // jeżeli rysujemy jajo, musimy je przesunąć w dół
    if (model != 4) glTranslatef(0.0f, -5.0f, 0.0f);

    glColor3f(1.0f, 1.0f, 1.0f);
    if (model == 1) drawEggPoints(N);
    else if (model == 2) drawEggLines(N);
```

```
    else if (model == 3) drawEggTriangles(N);
    else if (model == 4) glutWireTeapot(3.0);

    glFlush();
    glutSwapBuffers();
}

void changeSize(int horizontal, int vertical) {
    // przeliczenie pikseli na stopnie
    pix2angle = 360.0 / (float)horizontal;

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();

    if(vertical == 0) return;
    glViewport(0, 0, horizontal, vertical);

    // Ustawienie parametrów dla rzutu perspektywicznego:
    // 70 stopni pola widzenia w osi y
    // proporcje między osią x i y, do uzyskania pola widzenia w osi x
    // minimalna odległość od obserwatora
    // maksymalna odległość od obserwatora
    float aspectRatio = (float)horizontal / (float)vertical;
    gluPerspective(70, aspectRatio, 1.0, 30.0);

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

// Funkcja "bada" stan myszy i ustawia wartości odpowiednich zmiennych
// globalnych
void mousePressed(int btn, int state, int x, int y) {
    // przypisanie aktualnie odczytanej pozycji kursora jako pozycji poprzedniej
    x_pos_old = x;
    y_pos_old = y;

    // jeżeli wciśnięty lewy klawisz myszy, obracamy, jeżeli prawy,
    // przybliżamy/oddalamy
    if (btn == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {
        status = 1;
    }
    else if (btn == GLUT_RIGHT_BUTTON && state == GLUT_DOWN) {
        status = 2;
    }
    else
        status = 0;
}

// Wylicza poruszenie myszą i przerysowuje scenę
void mouseMoved(int x, int y) {
```

```
    // obliczenie różnicy położenia kursora myszy
    delta_x = x - x_pos_old;
    delta_y = y - y_pos_old;

    // podstawienie bieżącego położenia jako poprzednie
    x_pos_old = x;
    y_pos_old = y;

    glutPostRedisplay();
}

void keyPressed(unsigned char key, int x, int y) {
    if (key == 'p') model = 1;
    else if (key == 'w') model = 2;
    else if (key == 's') model = 3;
    else if (key == 'c') model = 4;

    renderScene();
}
```

4 Problemy

4.1 Zmiana orientacji kamery przy przekroczeniu osi y

Przez wykorzystanie funkcji `gluLookAt()`, gdy rotacja przekraczała oś y, następowała zmiana orientacji kamery tak, by górna krawędź widoku kamery była skierowana w kierunku rosnących y, tzn. tak by kamera nie była "do góry nogami". Skutkowało to skokową zmianą orientacji kamery oraz obrotu pokazywanego obiektu.

Możliwe były następujące sposoby rozwiązania tego problemu:

- Zablokować kąty rotacji tak by niemożliwe było przekroczenie osi y
- Gdy współrzędne kamery przekroczyły oś y, zmienić zwrot wektora up, tak by kamera pozostała "do góry nogami".

Do rozwiązania problemu zdecydowano się zablokowanie kątów rotacji.

4.2 Niska rozdzielczość ruchów myszą

Ponieważ funkcja obsługująca zdarzenia ruchu myszą otrzymuje od systemu operacyjnego liczbę pikseli o którą poruszył się wskaźnik myszy, obracanie widokiem może się niekiedy wydawać skokowe, szczególnie kiedy czułość myszy jest niska i uruchamiamy program na ekranie o niskiej rozdzielczości.

Rozwiązaniem tego problemu byłoby pobieranie informacji o ruchu myszy nie poprzez liczbę pikseli o którą poruszył się kursor, lecz jako bardziej "surowe" dane, określające poruszenie myszą bez względu na to czy kursor myszy poruszył się na ekranie. Byłaby to liczba zmiennoprzecinkowa, która mogłaby być arbitralnie mała dla bardzo niewielkich poruszeń myszą.