Grafika komputerowa i komunikacja człowiek-komputer

Sprawozdanie z laboratorium

Data	Tytuł zajęć	Uczestnicy
23.10.2017 8:00	OpenGL - renderowanie 3D	Iwo Bujkiewicz (226203)

Zadania

Na zajęciach należało napisać na podstawie instrukcji laboratoryjnej programy realizujące trójwymiarowo:

1. wyświetlanie kierunków osi układu współrzędnych,
2. wyświetlanie krawędzi czajnika Newella,
3. transformację czajnika Newella,
4. wyświetlanie punktów należących do powierzchni modelowego jajka,
5. wyświetlanie siatki oraz wypełnionych trójkątów powierzchni jajka,
6. animację obracania się jajka.

Kolejne etapy realizacji

Zadania 1, 2, 3

Jako, że zadania 2 i 3 kolejno rozszerzały program do zadania 1, zadania 1, 2 oraz 3 zrealizowano w postaci jednego programu.

Program miał wyrenderować obraz osi trójwymiarowego układu współrzędnych, a następnie narysować obraz siatki modelu czajnika Newella (szerzej znanego jako *Utah Teapot*), dokonując transformacji obracającej ten model.

```
void render_scene() {
    // Clear the stage using the current clear colour
    // Note the additional GL_DEPTH_BUFFER_BIT
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    // Reset the current transform matrix
    glLoadIdentity();

    draw_axes();
    draw_teapot();

    // Flush draw calls to execution
    glFlush();

    glutSwapBuffers();
}
```

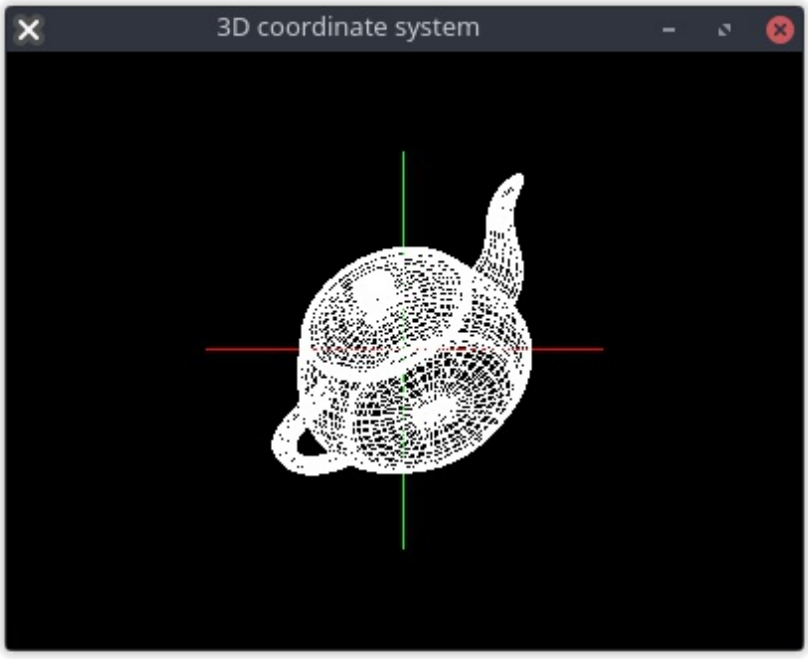
Funkcja `draw_axes()` rysowała obraz osi układu współrzędnych w promieniu od środka układu zadanym stałą `AXIS_RADIUS`. Obraz osi X rysowany był kolorem czerwonym, Y - zielonym, Z - niebieskim.

```
const float AXIS_RADIUS = 5.0f;

void draw_axes() {
    point3f x_axis_start = { -AXIS_RADIUS, 0.0f, 0.0f };
    point3f x_axis_end = { AXIS_RADIUS, 0.0f, 0.0f };
    point3f y_axis_start = { 0.0f, -AXIS_RADIUS, 0.0f };
    point3f y_axis_end = { 0.0f, AXIS_RADIUS, 0.0f };
    point3f z_axis_start = { 0.0f, 0.0f, -AXIS_RADIUS };
    point3f z_axis_end = { 0.0f, 0.0f, AXIS_RADIUS };

    glColor3f(1.0f, 0.0f, 0.0f);
    glBegin(GL_LINES);
        glVertex3fv(x_axis_start);
        glVertex3fv(x_axis_end);
    glEnd();

    glColor3f(0.0f, 1.0f, 0.0f);
    glBegin(GL_LINES);
        glVertex3fv(y_axis_start);
        glVertex3fv(y_axis_end);
    glEnd();

    glColor3f(0.0f, 0.0f, 1.0f);
    glBegin(GL_LINES);
        glVertex3fv(z_axis_start);
        glVertex3fv(z_axis_end);
    glEnd();
}
```

Za rysowanie czajnika Newella odpowiedzialna była funkcja `draw_teapot()`, która dokonywała jednocześnie transformacji widoku modelu w taki sposób, że model czajnika widoczny był obrócony o 60 stopni wokół osi wyznaczonej przez wektor `{1.0, 1.0, 1.0}` względem swojego początkowego położenia.

```
void draw_teapot() {
    glRotated(60.0, 1.0, 1.0, 1.0);
    glColor3f(1.0f, 1.0f, 1.0f);
    glutWireTeapot(3.0);
    glLoadIdentity();
}
```

Rezultat działania programu prezentuje poniższy zrzut ekranu.



Zadania 4, 5

Program realizujący zadania 4 i 5 rozpoczynał działanie od obliczenia współrzędnych oraz kolorów poszczególnych wierzchołków (punktów) składających się na powierzchnię jajka. Używał w tym celu funkcji `compute_egg_vertices()`.

```
void compute_egg_vertices() {
    egg_vertices = realloc(egg_vertices, EGG_SUBDIVISIONS * EGG_SUBDIVISIONS *
sizeof(point3f));
    vertex_colors = realloc(vertex_colors, EGG_SUBDIVISIONS * EGG_SUBDIVISIONS *
sizeof(point3f));

    for (int i = 0; i < EGG_SUBDIVISIONS; ++i) {
        float u = (1.0f / EGG_SUBDIVISIONS) * i;
        for (int k = 0; k < EGG_SUBDIVISIONS; ++k) {
            float v = (1.0f / EGG_SUBDIVISIONS) * k;
            point3f * vertex = get_vertex(i, k);
            create_vertex(u, v, vertex);
            create_vertex_color(vertex, get_vertex_color(vertex));
        }
    }
}
```

Wywoływana tu funkcje `create_vertex()` zapisywała w matrycy wierzchołków współrzędne pojedynczego wierzchołka, generowane na podstawie wzorów matematycznych podanych w instrukcji ćwiczenia.

```
void create_vertex(float u, float v, point3f * result) {
    (*result)[0] = (-  90.0f * powf(u, 5.0f)
            + 225.0f * powf(u, 4.0f)
            - 270.0f * powf(u, 3.0f)
            + 180.0f * powf(u, 2.0f)
            -  45.0f * u) * cosf (M_PI * v);

    (*result)[1] =    160.0f * powf(u, 4.0f)
            - 320.0f * powf(u, 3.0f)
            + 160.0f * powf(u, 2.0f)
            - 5.0f;

    (*result)[2] = (-  90.0f * powf(u, 5.0f)
            + 225.0f * powf(u, 4.0f)
            - 270.0f * powf(u, 3.0f)
            + 180.0f * powf(u, 2.0f)
            -  45.0f * u) * sinf(M_PI * v);
}
```

Funkcja `get_vertex_color()` natomiast zapisywała w matrycy kolorów wartości RGB, obliczone przez liniowe przeskalowanie odpowiadających im współrzędnych (x, y, z) z zakresu `[-5.0f, 5.0f]` na zakres `[0.0f, 1.0f]`. W rezultacie takiego obliczenia kolorów na narysowanym później jajku powinien być możliwy do zaobserwowania efekt trójwymiarowej tęczy.

```
void create_vertex_color(point3f * vertex, point3f * result) {
    for (int i = 0; i < 3; ++i) {
        (*result)[i] = ((*vertex)[i] + 5.0f) / 10.0f;
    }
}
```

Po wygenerowaniu matryc wierzchołków oraz kolorów tworzone było okno aplikacji i inicjalizowane było środowisko renderowania. Służyła do tego funkcja `render_scene()`.

```
void render_scene() {
    // Clear the stage using the current clear colour
    // Note the additional GL_DEPTH_BUFFER_BIT
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    // Reset the current transform matrix
    glLoadIdentity();

    draw_axes();

    if (viewmode & 1<<2)
        draw_egg_faces();
    if (viewmode & 1<<1)
        draw_egg_edges();
    if (viewmode & 1<<0)
        draw_egg_vertices();

    // Flush draw calls to execution
    glFlush();

    glutSwapBuffers();
}
```

Funkcja `draw_axes()` rysująca obraz osi układu współrzędnych wyglądała dokładnie tak samo, jak w programie do zadań 1, 2, 3. Po narysowaniu obrazu osi funkcja `render_scene()` decydowała, na podstawie informacji o aktualnym trybie wyświetlania (przechowywanym w zmiennej `viewmode`), które części powierzchni jajka narysować.

Wypełnione trójkąty powierzchni jajka rysowane były przez funkcję `draw_egg_faces()`. W pętlach pobierane były z macierzy odpowiednie współrzędne wierzchołków, a następnie używana była funkcja `face()`, która podawała do definicji trójkątów współrzędne odpowiednio dobranych wierzchołków wraz z ich kolorami.

```
void draw_egg_faces() {
    glColor3f(0.0f, 0.6f, 0.4f);
    glBegin(GL_TRIANGLES);
    for (int i = 0, i_next = 1; i < EGG_SUBDIVISIONS; ++i, ++i_next) {
        if (i_next == EGG_SUBDIVISIONS)
            i_next = 0;
        for (int k = 0, k_next = 1; k < EGG_SUBDIVISIONS; ++k, ++k_next) {
            point3f * v00 = get_vertex(i, k);
            point3f * v01 = k_next == EGG_SUBDIVISIONS ?
                      get_vertex(EGG_SUBDIVISIONS - i, 0)
                    : get_vertex(i, k_next);
            point3f * v10 = get_vertex(i_next, k);
            point3f * v11 = k_next == EGG_SUBDIVISIONS ?
                      get_vertex(EGG_SUBDIVISIONS - i_next, 0)
                    : get_vertex(i_next, k_next);
            face(v00, v01, v11);
            face(v00, v10, v11);
        }
    }
    glEnd();
}
```

Linie krawędzi jajka rysowane były przez funkcję `draw_egg_edges()`, która, podobnie jak `draw_egg_faces()`, używała funkcji pomocniczej - tym razem funkcji `edge()`, definiującej krawędzie wraz z odpowiadającymi ich wierzchołkom kolorami.

```
void draw_egg_edges() {
    glColor3f(1.0f, 1.0f, 0.5f);
    glBegin(GL_LINES);
    for (int i = 0, i_next = 1; i < EGG_SUBDIVISIONS; ++i, ++i_next) {
        if (i_next == EGG_SUBDIVISIONS)
            i_next = 0;
        for (int k = 0, k_next = 1; k < EGG_SUBDIVISIONS; ++k, ++k_next) {
            point3f * vertex = get_vertex(i, k);
            edge(vertex, get_vertex(i_next, k));
            if (k_next == EGG_SUBDIVISIONS) {
                edge(vertex, get_vertex(EGG_SUBDIVISIONS - i, 0));
                edge(vertex, get_vertex(EGG_SUBDIVISIONS - i_next, 0));
            }
            else {
                edge(vertex, get_vertex(i, k_next));
                edge(vertex, get_vertex(i_next, k_next));
            }
        }
    }
    glEnd();
}
```

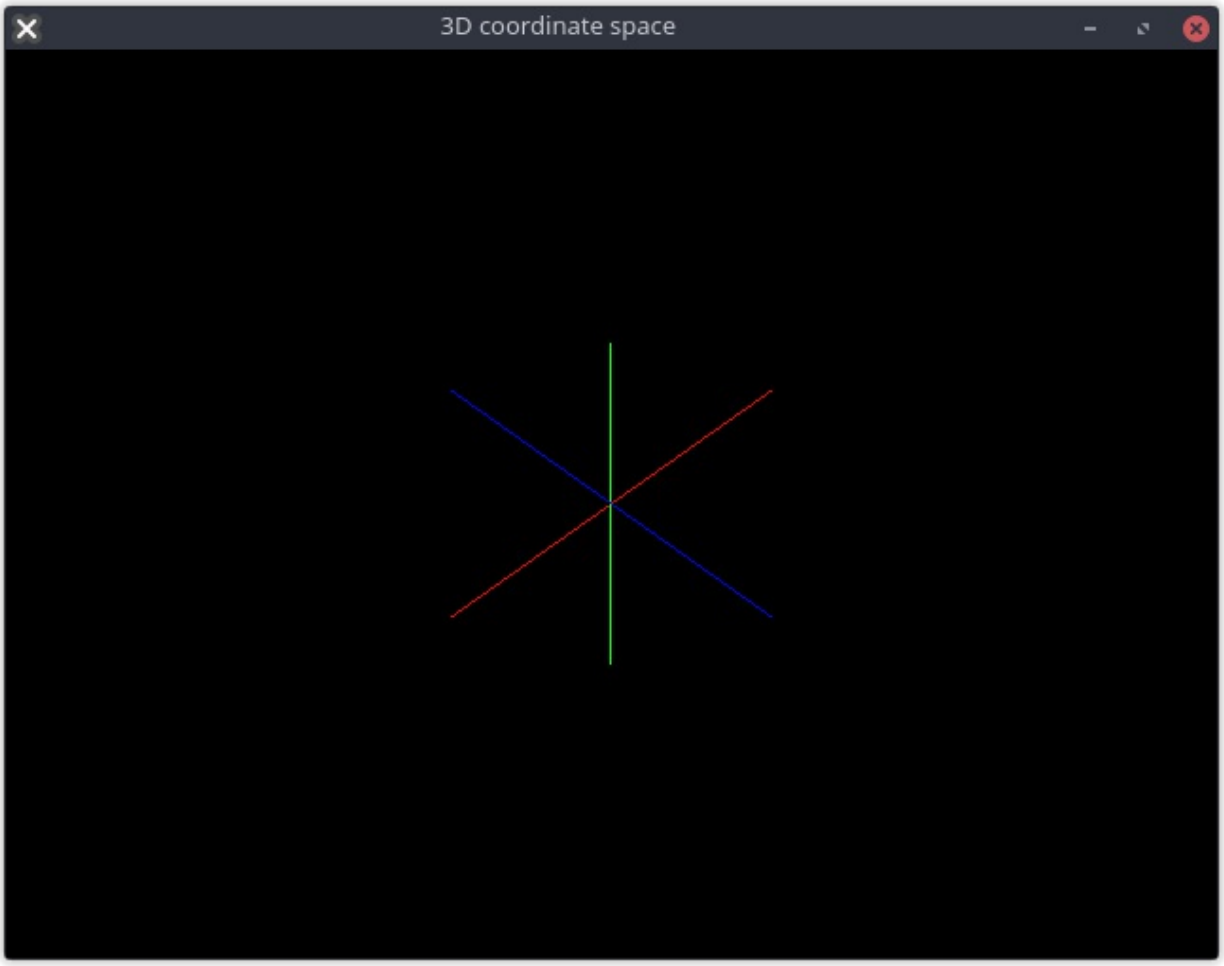
Najprostsza w konstrukcji była funkcja `draw_egg_vertices()`, rysująca obraz punktów (wierzchołków) jajka. Wszystkie punkty rysowane przy użyciu tej funkcji były jednakowego koloru, aby zachować czytelność obrazu.

```
void draw_egg_vertices() {
    glColor3f(1.0f, 1.0f, 1.0f);
    glBegin(GL_POINTS);
    for (int i = 0; i < EGG_SUBDIVISIONS; ++i) {
        for (int k = 0; k < EGG_SUBDIVISIONS; ++k) {
            point3f * vertex = get_vertex(i, k);
            glVertex3f((*vertex)[0], (*vertex)[1], (*vertex)[2]);
        }
    }
    glEnd();
}
```

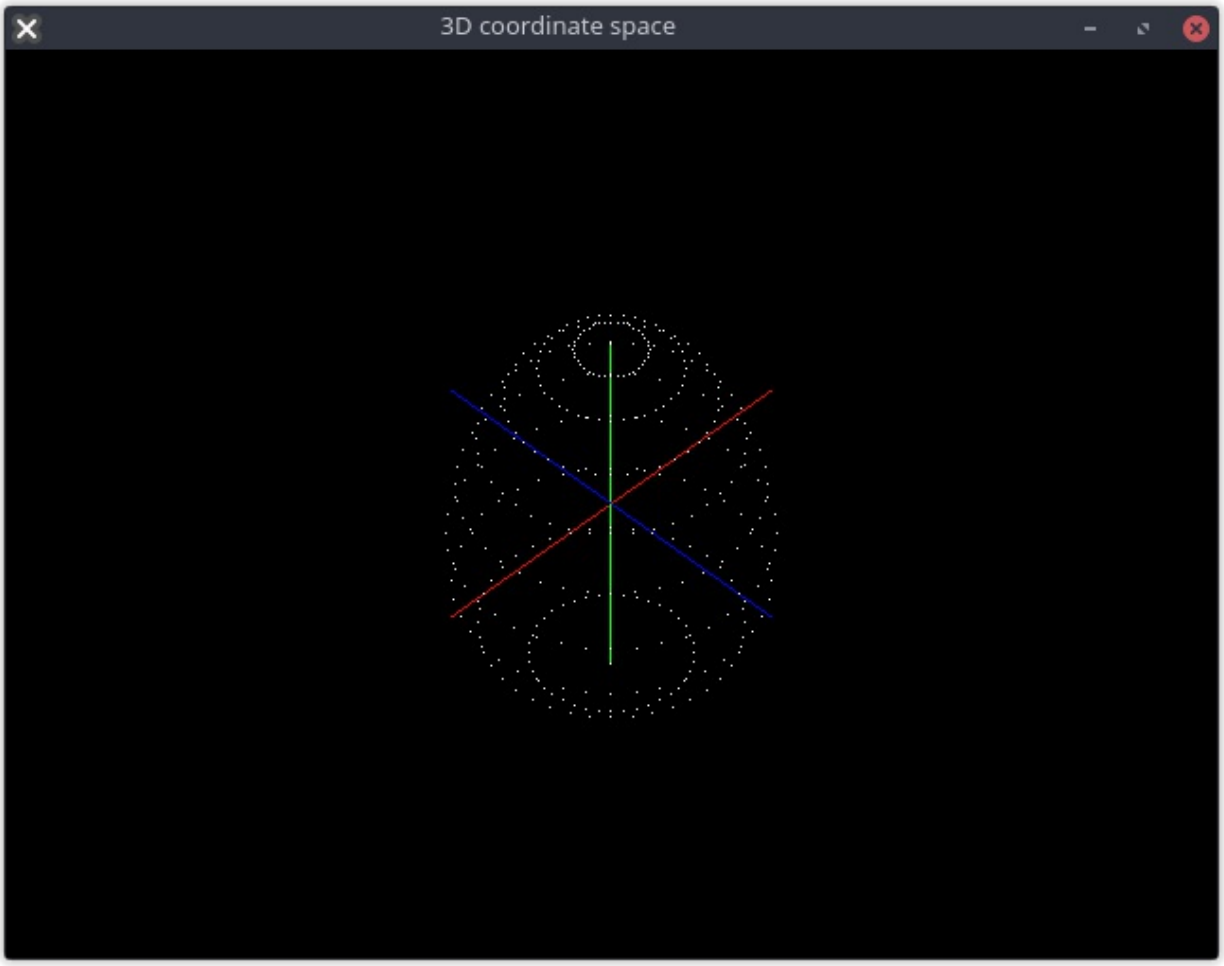
W celu umożliwienia zmiany trybu wyświetlania, program zawierał funkcję `key_pressed()`, przypisaną w GLUT do obsługi naciśnięć klawiszy na klawiaturze. Naciśnięcie klawisza `p` włączało lub wyłączało wyświetlanie punktów (wierzchołków), `w` - krawędzi, a `s` - wypełnionych trójkątów.

```
void key_pressed(unsigned char keycode, int x, int y) {
    if (keycode == 'p')
        viewmode ^= 1<<0;
    else if (keycode == 'w')
        viewmode ^= 1<<1;
    else if (keycode == 's')
        viewmode ^= 1<<2;

    render_scene();
}
```

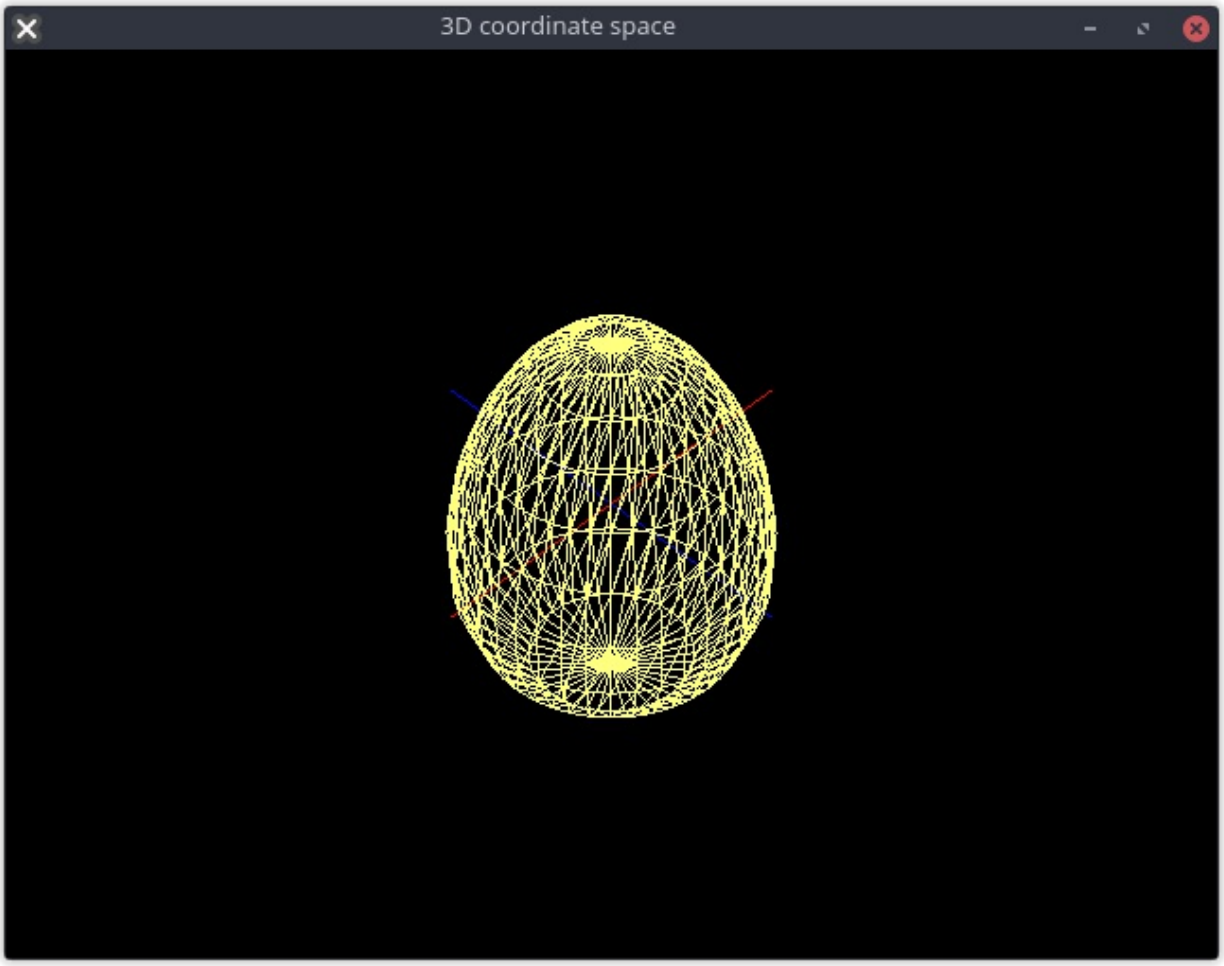
Rezultat działania programu prezentują poniższe zrzuty ekranu.



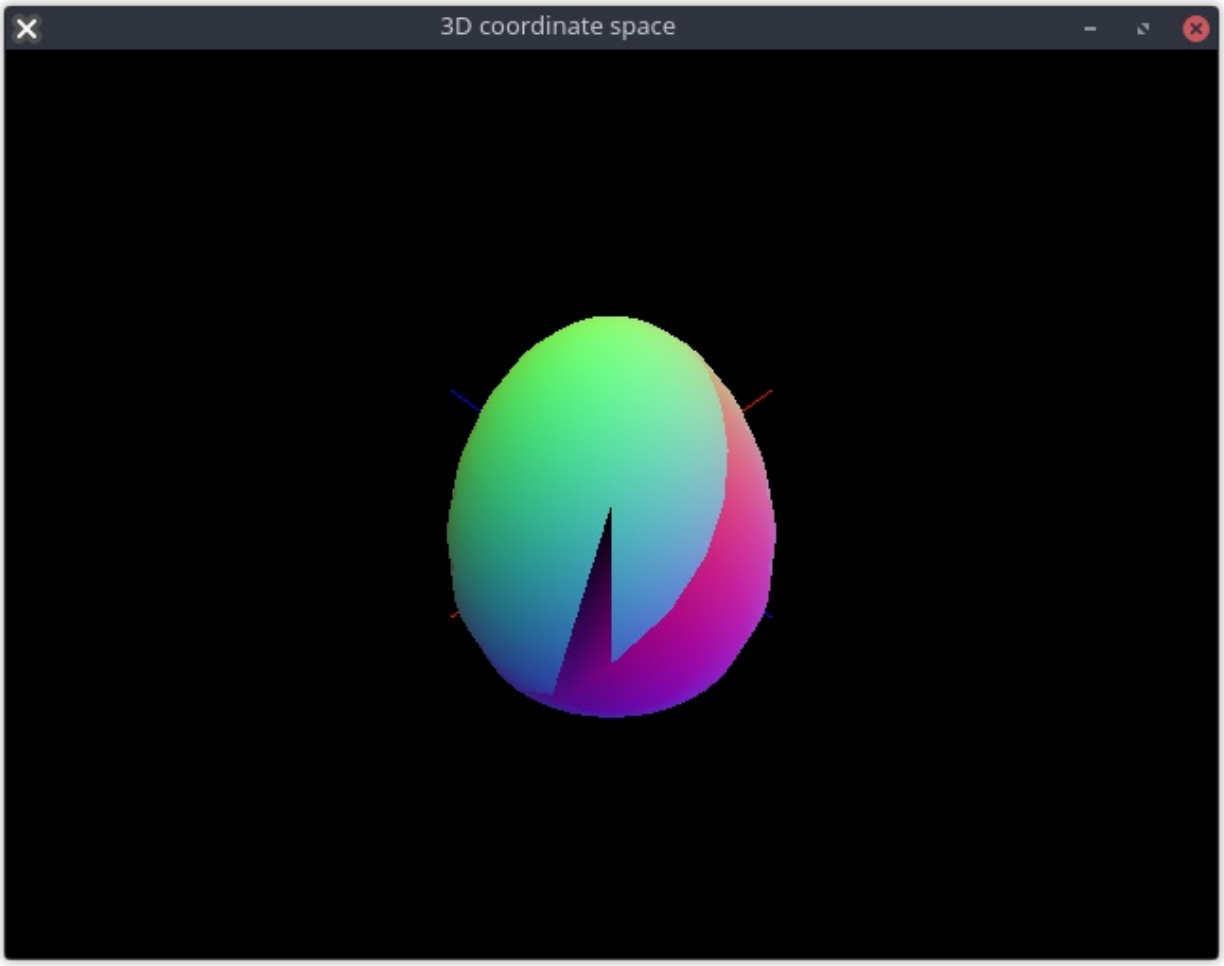
Rysowanie jajka wyłączone



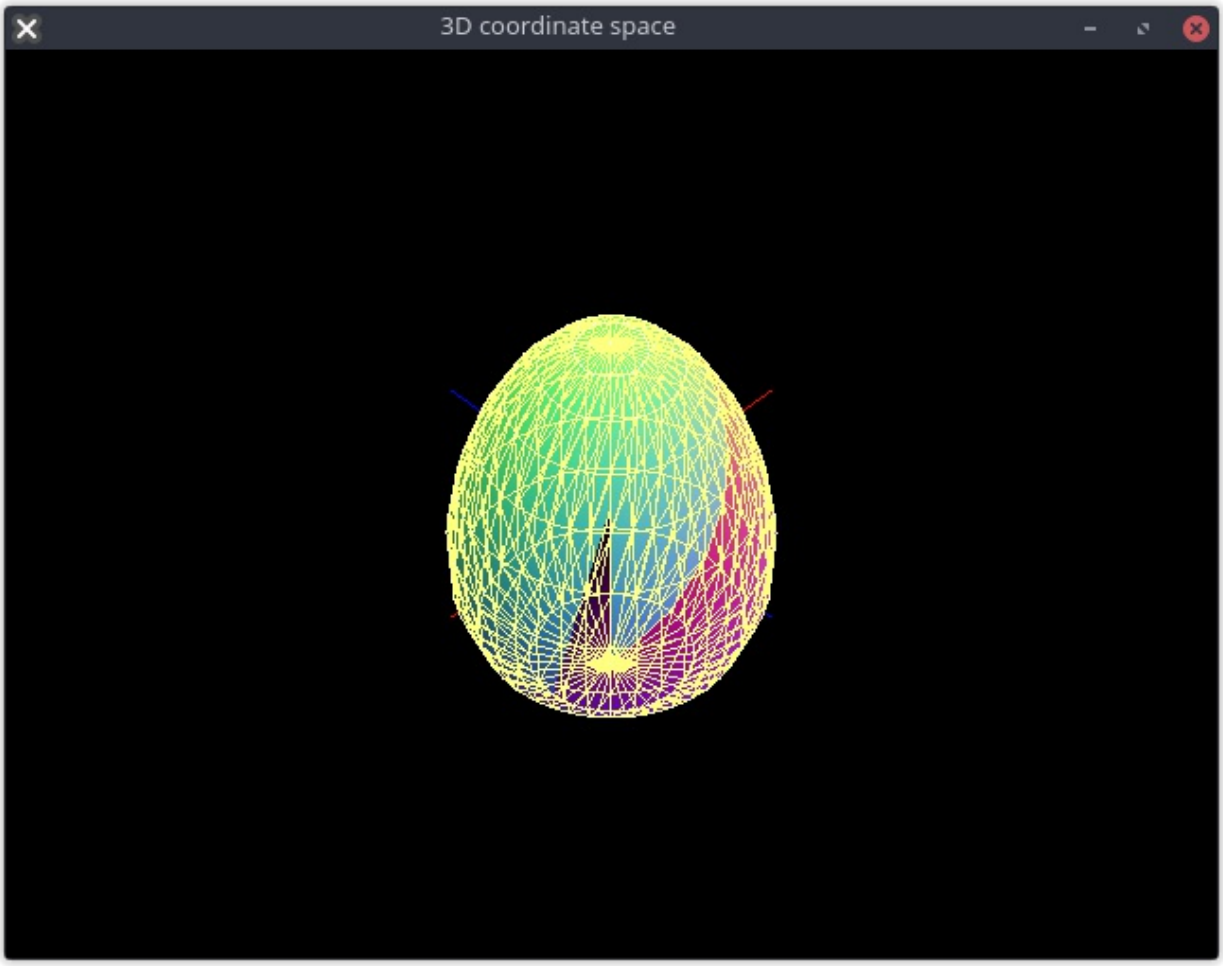
Rysowanie wierzchołków jajka



Rysowanie krawędzi jajka



Rysowanie wypełnionej powierzchni jajka



Rysowanie wypełnionej powierzchni, krawędzi oraz wierzchołków jajka

Kod źródłowy

Kompletny kod opisanych tu programów został załączony do sprawozdania w osobnych plikach.