POLITECHNIKA WROCŁAWSKA **Wydział Informatyki i Telekomunikacji**	**Autor:** Michał Pakuła	Wydział: W4 Rok: 2025 Rok akadem.: 2024/2025

Grafika komputerowa i komunikacja człowiek-komputer (laboratorium)		
Data ćwiczenia: **7.01.2025**	**Temat ćwiczenia laboratoryjnego:** ***OpenGL: Obsługa Tekstur***	**Ocena:**
Nr ćwiczenia: **5**		**Podpis prowadzącego:**

Streszczenie – W trakcie zajęć należało przy użyciu języka programowania oraz oprogramowania OpenGL obsłużyć tekstury w formacie *.tga na zaprogramowane wcześniej obiekty jajka oraz czajnika.

Oświadczenie: *Przekazując to sprawozdanie do oceny prowadzącemu zajęcia Autorzy wspólnie oświadczają, że zostało ono przygotowane samodzielnie, bez udziału osób trzecich oraz że żadna jego część nie jest plagiatem.*

1. Wstęp teoretyczny.

Biblioteka OpenGL pozwala na modelowanie obiektów w przestrzeni 2D oraz 3D. Przy pomocy odpowiednich funkcji matematycznych, można prezentować obiekty przestrzenne określone funkcjami lub współrzędnymi. Obiekty te mogą posiadać tekstury wprowadzane z zewnętrznych plików, oraz można obserwować jak zmieniają się pod wpływem światła oraz jak się łączą krawędzie tekstur.

2. Cel i zakres ćwiczenia oraz opis sposobu wykonania ćwiczenia.

Celem tego zadania jest zapoznanie się z funkcjami obowiązującymi w bibliotece OpenGL, poznanie znaczenia wektorów tekstur oraz nabranie płynności w obsłudze funkcji tej biblioteki.

3. Główne zmiany w kodzie

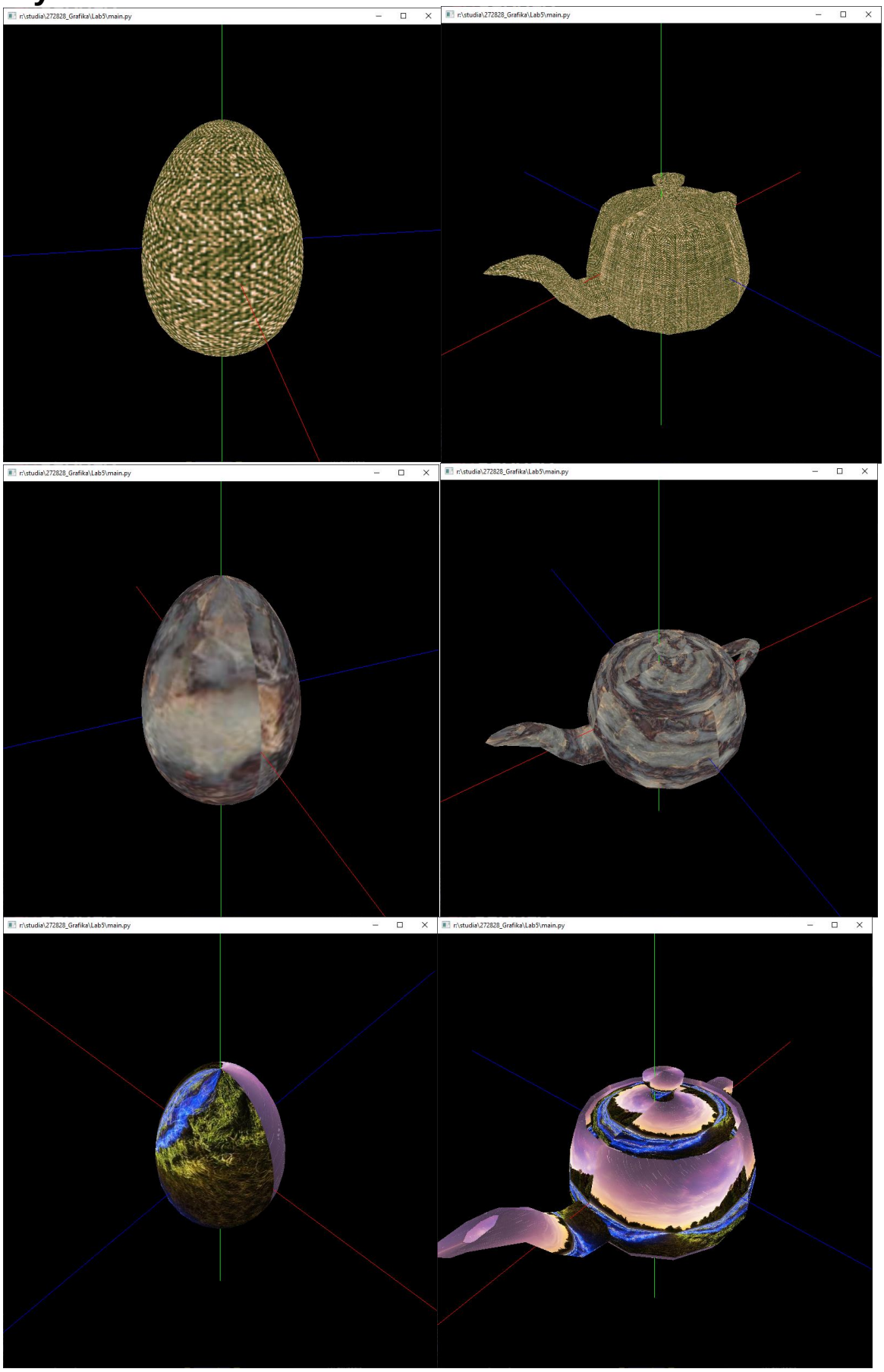
- Zmieniono plik „teapot.obj” na taki który posiada wartości mapowania tekstur oraz wektory normalne,
- Zmieniono funkcję czytającą plik *.obj*
- Dodano obsługę tekstur do programu głównego,
- Nałożono tekstury według zmapowanych punktów na jajko oraz czajnik,
- Przy pomocy klawisza „\” pozwolono na zmianę widocznej tekstury w czasie rzeczywistym
- Podpięto zewnętrzny folder do obsługi tekstur

```python
def startup():
    glClearColor(0.0, 0.0, 0.0, 1.0)
    update_viewport(None, 800, 800)

    glEnable(GL_DEPTH_TEST)
    glEnable(GL_COLOR_MATERIAL)
    glEnable(GL_CULL_FACE)
    glFrontFace(GL_CCW)
    glShadeModel(GL_SMOOTH)

    material_ambient = [0.2, 0.2, 0.2, 1.0]
    material_diffuse = [1.0, 1.0, 1.0, 1.0]
    material_specular = [0.3, 0.3, 0.3, 1.0]
    material_shininess = 50

    glMaterialfv(GL_FRONT, GL_AMBIENT, material_ambient)
    glMaterialfv(GL_FRONT, GL_DIFFUSE, material_diffuse)
    glMaterialfv(GL_FRONT, GL_SPECULAR, material_specular)
    glMaterialfv(GL_FRONT, GL_SHININESS, material_shininess)

    #glEnable(GL_LIGHTING)
    glEnable(GL_LIGHT0)
    glEnable(GL_LIGHT1)

    glEnable(GL_TEXTURE_2D)
    glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE)
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT)
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT)
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR)
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR)

```

```python
    glEnable(GL_CULL_FACE)
    glFrontFace(GL_CW)
    texture_image = Image.open(files[whatimage])
    glTexImage2D(
        GL_TEXTURE_2D, 0, 3, texture_image.size[0], texture_image.size[1], 0,
        GL_RGB, GL_UNSIGNED_BYTE, texture_image.tobytes("raw", "RGB", 0, -1)
    )
    glEnable(GL_TEXTURE_2D)
    glBegin(GL_TRIANGLES)
    for j in range(1, d):
        for i in range(1, d):
            v1 = tab[i - 1, j - 1]
            v2 = tab[i, j - 1]
            v3 = tab[i, j]
            v4 = tab[i - 1, j]

            normal1 = calculate_normal(v1, v2, v3)
            normal2 = calculate_normal(v1, v3, v4)

           # Pierwszy trójkąt
            glColor3f(1.0, 1.0, 1.0)  # white
            glNormal3f(*normal1)
            glTexCoord2f(*texture[i - 1, j - 1])  # Współrzędne tekstury dla v1
            glVertex3f(*v1)
            glTexCoord2f(*texture[i, j - 1])  # Współrzędne tekstury dla v2
            glVertex3f(*v2)
            glTexCoord2f(*texture[i, j])  # Współrzędne tekstury dla v3
            glVertex3f(*v3)

            # Drugi trójkąt
            glColor3f(1.0, 1.0, 1.0)  # white
            glNormal3f(*normal2)
            glTexCoord2f(*texture[i - 1, j - 1])  # Współrzędne tekstury dla v1
            glVertex3f(*v1)
            glTexCoord2f(*texture[i, j])  # Współrzędne tekstury dla v3
            glVertex3f(*v3)
            glTexCoord2f(*texture[i - 1, j])  # Współrzędne tekstury dla v4
            glVertex3f(*v4)
    glEnd()
    glDisable(GL_TEXTURE_2D)
```

```python
def utahTeapot():
    global obj_model
    glDisable(GL_CULL_FACE)
    glFrontFace(GL_CCW)
    if obj_model:
        texture_image = Image.open(files[whatimage])
        glTexImage2D(
        GL_TEXTURE_2D, 0, 3, texture_image.size[0], texture_image.size[1], 0,
        GL_RGB, GL_UNSIGNED_BYTE, texture_image.tobytes("raw", "RGB", 0, -1)
        )
        glEnable(GL_TEXTURE_2D)
        glBegin(GL_TRIANGLES)
        for face in obj_model['faces']:
            for vertex_data in face:
                vertex_id, texture_id, normal_id = (int(idx)- 1 if idx else -1 for idx in vertex_data.split('/'))
                if texture_id >= 0:
                    glTexCoord2fv(obj_model['textures'][texture_id])
                if normal_id >= 0:
                    glNormal3fv(obj_model['normals'][normal_id])
                glColor3f(1.0, 1.0, 1.0)
                glVertex3fv(obj_model['vertices'][vertex_id])
        glEnd()
        glDisable(GL_TEXTURE_2D)

def load_shape_from_obj(file_path):
    try:
        vertices = []
        faces = []
        textures = []
        normals = []
        with open(file_path) as f:
            for line in f:
                if line.startswith('v '):
                    vertex = list(map(float, line[2:].strip().split()))
                    vertices.append(vertex)
                elif line.startswith('vt '):
                    texture = list(map(float, line[3:].strip().split()))
                    textures.append(texture[:2])
                elif line.startswith('vn '):
                    normal = list(map(float, line[3:].strip().split()))
                    normals.append(normal)
                elif line.startswith('f '):
                    face = [entry for entry in line[2:].strip().split()]
                    faces.append(face)


        shape_data = {
            "vertices": np.array(vertices, dtype=np.float32),
            "textures": np.array(textures, dtype=np.float32),
            "normals": np.array(normals, dtype=np.float32),
            "faces": faces
        }
        return shape_data

    except FileNotFoundError:
        print(f"{file_path} not found.")
    except:
        print("An error occurred while loading the shape.")

```

4. Wynik działania kodu



5. Podsumowanie

Ćwiczenie polegało na implementacji nakładania różnych tekstur na obiekt oraz obserwacji w jaki sposób się wyświetlają, korzystając z biblioteki OpenGL. Zadanie to było skomplikowane ze względu na mapowanie wielu punktów dla jednej współrzędnej w taki sposób, aby tekstura wyglądała naturalnie. Podczas implementacji zwrócono też uwagę na możliwość zmiany wartości w linii 83 oraz 84 programu głównego z GL_LINEAR na GL_NEAREST co dawało wyraźny efekt „pikselowy”. Po raz kolejny można było zauważyć, w jaki sposób działa grafika komputerowa i jakie korzyści za sobą niesie znajomość działania grafiki komputerowej.