WebGL, proste oświetlenie

1. Cel ćwiczenia:

Zapoznanie funkcjami realizującymi oświetlenie sceny. Ustawienie właściwości źródeł światła.

2. Wstęp:

W celu zbudowania sceny wyglądającej maksymalnie realistycznie konieczne jest zastosowanie oświetlenia. Światła oraz materiały posiadają właściwości odpowiedzialne za wygląd oświetlanych obiektów. Dobranie światła i właściwości materiałów tak, aby uzyskać pożądany efekt, wymaga testów oraz praktyki.

Rozróżniamy trzy rodzaje świateł:

1. **światła kierunkowe –** ich promienie są równoległe, natężenie jest jednakowe,
2. **światła punktowe** – nie posiadają kierunku, oświetlają całą scenę, mają określone źródło, natężenie zależy od odległości,
3. **reflektory** – świecą w danym kierunku z określonej pozycji. Snop światła mieści się w ściśle określonym stożku.

Każdy rodzaj światła posiada składowe:

- **światło otaczające** – nie ma kierunku, równomiernie oświetla całą scenę,
- **światło rozproszone** – promienie pochodzą z konkretnego kierunku i oświetlają obiekt równomiernie z jednej strony,
- **światło odbłysków** – promienie nie są odbijane równomiernie tylko ostro w jednym z kierunków.

3. Zadanie:

Należy przygotować projekt zawierający:

1. zaimportowane obiekty z wektorami normalnymi, przygotować dwie wersje z cieniowaniem płaskim i gładkim (opcja w Blender 3D),
2. na scenie umieścić źródło światła,
3. scena ma składać się z obiektów umieszczonych w zamkniętym pomieszczeniu.(aby było oświetlone od wewnątrz, należy odwróć wektory normalne – opcja w Blender 3D)

Wskazówki:

- **zaimportować obiekty do VBO wraz z informacjami o położeniu punktów, koordynatami tekstur oraz wektorami normalnymi,**

- **przekazać informacje o wektorach normalnych określając specyfikację formatu danych wierzchołkowych (zamiast o kolorze)**

```
const normalAttrib = gl.getAttribLocation(program, "aNormal");
gl.enableVertexAttribArray(normalAttrib);
gl.vertexAttribPointer(normalAttrib, 3, gl.FLOAT, false, 8*4, 3*4);
```

- **ustalić położenie światła np.;**

```
const lightPos = [-2.2,2.0,10.0];
var lightPos_loc = gl.getUniformLocation(program, 'lightPos');
gl.uniform3fv(lightPos_loc,new Float32Array(lightPos));
```

- **przekazać położenie kamery do zmiennej uniform we fragment shader (do funkcji obsługi klawiszy):**

```
let cameraPosTmp = [cameraPos.x,cameraPos.y,cameraPos.z];
gl.uniform3fv(uniCamPos,new Float32Array(cameraPosTmp));
```

  wcześniej pobrać lokalizację zmiennej uniform w programie głównym:

```
var uniCamPos = gl.getUniformLocation(program, 'camPos');
```

- **w kodzie vertex shader'a dodać kod:**

```
in vec3 aNormal;
out vec3 Normal;
```

  - **odebrać informację o wektorach normalnych i przekazać do fragment shadera(w funkcji main):**

```
Normal = aNormal;
```

  - **zadeklarować zmienną do przekazania pozycji fragmentów do fragment shadera z vertex shadera:**

```
out vec3 FragPos;
```

  - **przekazać do fragment shader'a informację o pozycji fragmentów(w funkcji main):**

```
FragPos = vec3(model * vec4(position, 1.0));
```

- **w kodzie fragment shadera:**
  - **odebrać informację o wektorach normalnych i pozycji fragmentów z vertex shader'a:**

```
in vec3 Normal;
in vec3 FragPos;
```

  - **odebrać informację o położeniu kamery i pozycji światła:**

```
uniform vec3 camPos;
uniform vec3 lightPos;
```

  - **dodać moc i kolor światła otoczenia:**

```
 // ambient
float ambientStrength = 0.1;
vec3 ambientlightColor = ambientStrength* vec3(1.0,1.0,1.0);
vec4 ambient = vec4(ambientlightColor,1.0);
```

  - **dodać moc i kolor światła rozproszonego:**

```
// diffuse
float diffuseStrength = 1.0;
vec3 diffuselightColor = vec3(1.0,1.0,1.0);
```

  - **wyznaczyć wektor kierunku między źródłem światła, a pozycją fragmentu i znormalizować zarówno wektor normalny jak i wynikowy wektor kierunkowy:**

```
vec3 norm = normalize(Normal);
vec3 lightDir = normalize(lightPos - FragPos);
```

  - **ustalić wpływ składowej rozproszonej światła na bieżący fragment, biorąc iloczyn skalarny wektora normalnego i wektora lightDir (dla kąta pomiędzy wektorami > 90° wynik jest ujemny dlatego zastosowano funkcję max)**

```
float diff = max(dot(norm, lightDir), 0.0);
vec3 diffvec = diff * diffuselightColor*diffuseStrength;
vec4 diffuse = vec4(diffvec,1.0);
```

  - **dodać moc i kolor składowej rozbłysków:**

```
//specular
float specularStrength = 1.0;
vec3 specularlightColor = vec3(1.0,1.0,1.0);
```

  - **wyznaczyć wektor kierunku widoku z kamery do fragmentu, odejmując od wektora położenia kamery wektor położenia fragmentu:**

```
vec3 viewDir = normalize(camPos - FragPos);
```

- **wyznaczyć wektor określający kierunek rozbłysku korzystając z funkcji reflect**
  ```
  vec3 reflectDir = reflect(-lightDir, norm);
  ```

- **wyznaczyć współczynnik decydujący o zanikaniu efektu wraz z kątem padania światła (wielkość plamki)**
  ```
  float spec = pow(max(dot(viewDir, reflectDir), 0.0), 128.0);
  ```

- **wyznaczyć ostateczną postać wektora składowej rozbłysków:**
  ```
  vec3 spec_final = specularStrength * spec * specularlightColor;
  vec4 specular = vec4(spec_final,1.0);
  ```

- **dodatkowo można uwzględnić wpływ odległości światła na jasność fragmentów:**
  ```
  //disstance
  float dist= distance(lightPos,FragPos);
  dist = (30.0-dist)/30.0;
  dist = max(dist, 0.0);
  ```

- **ostatecznie po wyznaczeniu trzech składowych światła otoczenia, rozproszonego, rozbłysków oraz współczynnika odległości, należy wykonać ich sumowanie i mnożenie przez bieżący kolor fragmentu:**
  ```
  frag_color = (ambient + dist*diffuse+dist*spec) * texture(texture1, TexCoord);
  ```

- **dodatkowe opcje wpływające na jakość i wydajność renderingu:**
  - **zastosowanie MipMap**
    ```
    gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR_MIPMAP_LINEAR);
    gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.LINEAR_MIPMAP_LINEAR);
    ```

  - **ukrywanie niewidocznych stron wielokątów tzw. Face Culling (prymitywów):**
    ```
    gl.enable(gl.CULL_FACE);
    ```

  - **zastosowanie filtrowania anizotropowego**
    ```
    var ext = gl.getExtension("EXT_texture_filter_anisotropic");
    ```

    dodanie filtra do każdej tekstury
    ```
    gl.texParameterf(gl.TEXTURE_2D, ext.TEXTURE_MAX_ANISOTROPY_EXT, 4);
    ```