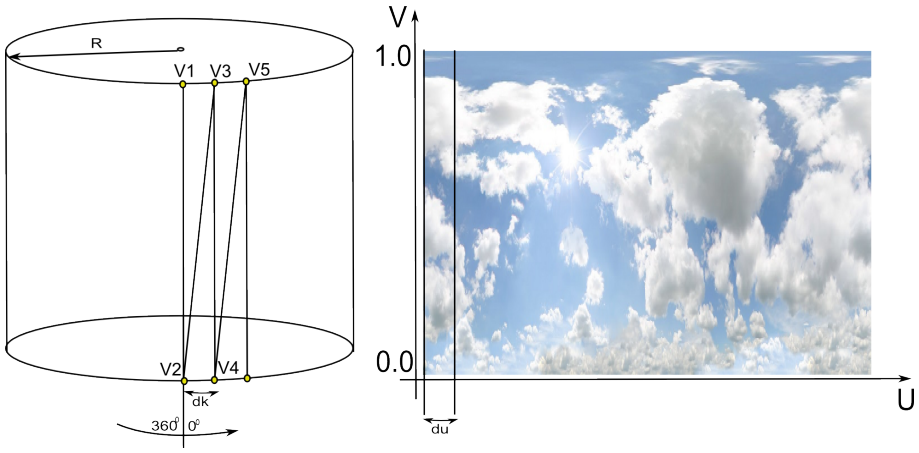
1 Mieszanie kolorów

Proces mieszania kolorów (ang. blending) pozwala uzyskiwać kilka ciekawych i realistycznych efektów wizualnych.Tematem bieżących ćwiczeń będzie implementacja mieszania kolorów dla uzyskania efektu przezroczystości oraz zamglenia. Ponadto omówione zostanie zagadnienie teksturowania obiektów przestrzennych.

1.1 Teksturowanie obiektów przestrzennych

Punktem wyjścia do teksturowania obiektów innych niż płaskie prostokąty jest zapewnienie możliwości podziału powierzchni obiektu na zbiór przystających do siebie, płaskich prymitywów (nazywanych dalej kafelkami [1]) oraz naklejenie na każdego z nich fragmentu tekstury pobranego z obrazu 2D. Fragment jest wybierany poprzez współrzędne tekstury tak, aby obraz tekstury został ówinięty"wokół przestrzennego obiektu. Dla zilustrowania tego zagadnienia omówimy proces zawijania dwuwymiarowej tekstury na powierzchnię walca. Szczegóły geometryczne prezentuje rys. 1



Rysunek 1: Nakładanie tekstury na powierzchnię walca

Rozważmy teksturę przedstawiającą panoramiczne ujęcie nieba. Po nałożeniu jej na powierzchnię walca o promieniu R oraz wysokości H powstanie najprostsza realizacja tzw. skyboxa. Po umieszczeniu obserwatora wewnątrz walca uzyskujemy wrażenie realistycznego obsadzenia w pewnej przestrzeni. Tworzenie obiektu typu skybox sprowadza się wówczas do wygenerowania geometrii walca o zadanym promieniu i wysokości. Walec składa się z przystających do siebie prostokątów (generowanych z użyciem prymitywów `GL_TRIANGLE_STRIP`). Zauważmy, że tworzenie kolejnych przystających trójkątów w tym prymitywie wymaga opisania geometrii wierzchołków $V1$ $V2$ dla aktualnej wartości kąta i promienia wodzącego. Następnie należy zwiększyć wartość kąta o obrany przyrost i ponowić generacje wierzchołków $V3$ $V4$. Proces powtarza się, aż aktualna wartość kąta osiągnie 360^o. Dla każdego z wierzchołków $V\cdot$ przypisujemy współrzędne (u, v) tekstury. Nakładanie tekstury zaczynamy od lewej krawędzi obrazu gdy $u = 0.0$ (jest to współrzędna skojarzona z kątem 0^o) a kończymy na $u = 1.0$ gdy kąt osiągnie wartość 360^o.

1.1.1 Przebieg ćwiczenia

1. Utwórz nowy projekt programistyczny i podłącz do niego pobrane pliki
2. Skompiluj i uruchom projekt. Na bieżącym etapie rozwoju aplikacja posiada zaimplementowane zmienne `Cam_r` oraz `Cam_angle`, którymi można sterować z użyciem klawiszy `WSAD`

[1] ang. tiles

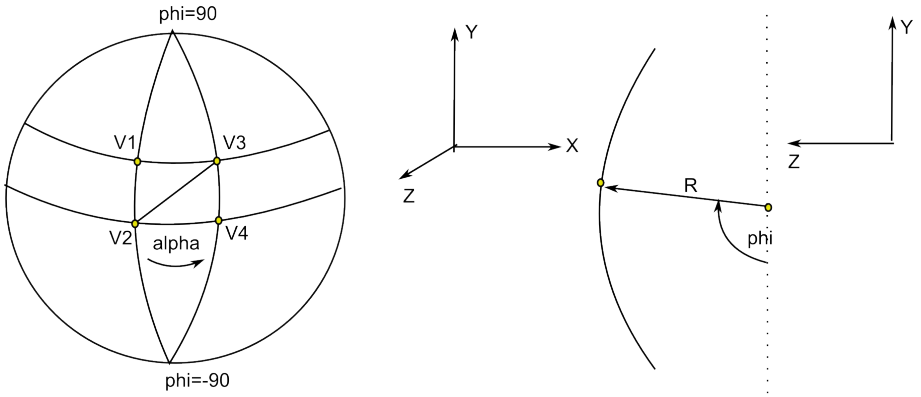
3. Rozbuduj wywołanie funkcji `LookAt` w kodzie rysowania sceny aby zapewnić możliwość przemieszczania kamery po okręgu o promieniu `Cam_r` o kąt orientacji `Cam_angle`. Kamera winna znajdować się na wysokości $y = 0.0$ i obracać wokół osi Y układu odniesienia

4. Zapoznaj się z implementacją klasy `glSkyBox` dziedziczącej z klasy `glObject`. Zwróć uwagę na sposób wywołania konstruktora tej klasy w kodzie `PrepareObjects`.

5. Odsuń obserwatora (kamerę) tak, aby zobaczyć obiekt rysowany przez `SkyBox`.

6. Przejdź do kodu konstruktora klasy `glSkyBox`. Zmień rodzaj prymitywu używanego do rysowania na `GL_TRIANGLE_STRIP` Zaimplementuj nakładanie tekstury `tex` na powierzchnię generowanego walca. Zwróć uwagę na sposób wykorzystania wysokości H do generowania współrzędnych wierzchołków w ramach pętli w konstruktorze klasy `glSkyBox`

W podobny sposób można wygenerować powierzchnię teksturowanej sfery. W tym przypadku pojedynczy walec reprezentowany przez kod generujący skybox jako wycinek w powierzchni równoleżnikowej sfery. Złożenie walców o zmieniającym się promieniu spowoduje utworzenie powierzchni aproksymującej kulę. Rozważmy geometrię sfery zaprezentowaną na rys. 2



Rysunek 2: Geometria sfery

W pasie równoleżnikowym sfery generowana jest wstęga trójkątów z wykorzystaniem prymitywu `TRIANGLE_STRIP`. Jest ona wytwarzana z wykorzystaniem pętli o iteratorze będącym kątem `alpha`. Każda pojedyncza wstęga wymaga wygenerowania zadanej liczby przystających prymitywów i obiegu wartości `alpha` od 0 do 360 stopni. Zauważmy, że współrzędne y generowanych wierzchołków będą uzależnione od kąta `phi`, który w ujęciu geograficznym miałby wymiar szerokości geograficznej. Kąt `phi` wpływa również na wartość promienia używanego do tworzenia walca. Tym samym, dla uzyskania powierzchni sfery warto założyć sferyczny układ współrzędnych, dokonać iteracji dla wartości katów `alpha` oraz `phi` oraz stałego promienia sfery i przeliczyć aktualne wartości współrzędnych sferycznych na kartezjańskie z użyciem ogólnie dostępnych zależności trygonometrycznych. Proces nakładania tekstury na powierzchnię sfery odbywa się wówczas analogicznie jak w przypadku teksturowania walców, z tą różnicą, że z obrazu tekstury wybierane są zarówno wycinki względem zmiennej v jak i u

1.1.2 Przebieg ćwiczenia

1. Przy pomocy polecenia `include` włącz do pliku `scene.h` plik `sphere.h`

2. Zapoznaj się z kodem pliku `sphere.cpp` jest on zrealizowany analogicznie do pliku generującego SkyBox

3. Uzupełnij kod konstruktora klasy `glSphere` o tworzenie geometrii sfery zgodnie z powyższym opisem i z uwzględnieniem sposobu przeliczania współrzędnych sferycznych na kartezjańskie.

4. Utwórz instancję klasy `glSphere` o nazwie `Moon`. Uzupełnij stosowne wywołania w metodach `PrepareObjects` oraz `Draw` klasy implementującej scenę. Podłącz do konstruktora ładowanie pliku tekstury `moon.bmp`

5. Przy testowaniu poprawności rysowania geometrii użyj funkcji zmieniających kolor fragmentów. Po upewnieniu się, że obiekt jest rysowany prawidłowo, zapewnij możliwość teksturowania

1.2 Dodawanie przezroczystości do obiektu

Uzyskanie przezroczystych powierzchni w procesie renderowania sceny polega na pobraniu koloru obiektu leżącego pod powierzchnią i zmieszaniu do z kolorem rysowanej powierzchni. Algorytm realizacji procesu mieszania sprowadza się do obliczania iloczynów składowych R, G, B koloru powierzchni i obiektu pod nią. Intensywność procesu mieszania kontrolowana jest skalarem określanym jako wartość kanału alpha. Składowa alpha może być wprowadzona jako dodatkowy atrybut wierzchołka. Może mieć też bardziej globalny charakter i odnosić się np. do całego rysowanego obiektu.

1.2.1 Przebieg ćwiczenia

1. Wprowadź do shadera wierzchołków zmienną jednorodną typu float o nazwie `Alpha`

2. Rozbuduj kod shadera wierzchołków do następującej postaci

```
void main()
{
	gl_Position = projectionMatrix*modelViewMatrix*vec4(inPosition, 1.0);
	vec4 vRes = normalMatrix*vec4(inNormal, 0.0);
	vNormal = vRes.xyz;
	kolorek = vec4(inColor,Alpha);
	texCoord = vec2(inTexCoord[0],inTexCoord[1]);
}
```

   Spowoduje to załączenie składowej alpha jako czwartej współrzędnej wektora opisującego kolor przekazywany do shadera fragmentów.

3. Wprowadź jako składową prywatną sceny zmienną typu float o nazwie `Alpha`. Zapewnij ustawienie jej wartości początkowej np. 0.5 w konstruktorze klasy. Zapewnij sterowanie wartością zmiennej z inkrementem 0.1 w ramach metody `KeyPressed`. Wykorzystaj w tym celu klawisze F3 oraz F4

4. W kodzie rysowania sceny zapewnij przekazywanie wartości zmiennej Alpha z aplikacji do shadera poprzez mechanizm zmiennych jednorodnych. Zapis w kodzie jest analogiczny do sposobu przekazania zmiennej `LightAmbient`

5. Uzupełnij kod rysowania sceny do następującej postaci:

```
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
Cube->Draw();
glDisable(GL_BLEND);
```

6. Skompiluj i uruchom program. Sprawdź jak sterowanie klawiszami wpływa na poziom przezroczystości obiektu. Obejrzyj obiekt pod różnymi kątami. Czy można dostrzec chmury z tekstury SkyBox poprzez podwójną ścianę sześcianu?

7. Porównaj efekty mieszania kolorów dla różnych algorytmów mieszania. Przygotuj zrzuty ekranowe. Wykaz stałych możliwych do zadania w funkcji `glBlendFunc` znajdziesz w dokumentacji OpenGL.

1.3 Generowanie mgły

Efekt zamglenia jest realizowany w podobny sposób do przezroczystości. Kolory fragmentów obiektów są mieszane ze stałym kolorem mgły. Poziom intensywności mieszania jest uzależniony od odległości obiektu od obserwatora. Odległość obiektu od obserwatora w najprostszym ujęciu można wyznaczyć jako składową z wierzchołka po jego przemnożeniu przez macierz transformacji Model-Widok. W implementacji pomija się macierz transformacji perspektywy, aby zniekształcenie widoku dające wrażenie perspektywy nie zakłócało obliczenia odległości obiektu od obserwatora. Im obiekt jest dalej, tym mniej przezroczysta jest mgła. Związek pomiędzy wartością przezroczystości mgły może być opisany przy pomocy jednego z trzech tzw. równań mgły. Jednym z najbardziej popularnych jest równanie wykładnicze:

$$FogFactor = 1.0 - e^{-Density \cdot FogCoord} \tag{1}$$

w którym $Density$ oznacza współczynnik gęstości mgły, a $FogCoord$ jest odległością obiektu od obserwatora.

Drugie z równań uwzględnia zależność kwadratową w wykładniku:

$$FogFactor = 1.0 - e^{-(Density \cdot FogCoord)^2} \tag{2}$$

Trzecie równanie ma charakter liniowy. Mgła zaczyna się w pewnej odległości od obserwatora $FogStart$ i kończy po osiągnięciu odległości $FogEnd$. Położenie obiektu $FogCoord < FogStart$ daje 0.0, co powoduje całkowite odsłonięcie obiektu. Dla sytuacji $FogCoord > FogEnd$ obiekt pozostanie całkowicie zamglony. W sytuacjach pośrednich $FogStart \leqslant FogCoord \leqslant FogEnd$ stosuje się interpolację liniową:

$$FogFactor = 1.0 - \frac{FogEnd - FogCoord}{FogEnd - FogStart} \tag{3}$$

1.3.1 Przebieg ćwiczenia

1. Rozbuduj shader wierzchołków do postaci z listingu poniżej:

```
#version 330

uniform mat4 projectionMatrix;
uniform mat4 modelViewMatrix;
uniform mat4 normalMatrix;

uniform float ColorAlpha;

layout (location = 0) in vec3 inPosition;
layout (location = 1) in vec3 inColor;
layout (location = 2) in vec3 inNormal;

layout (location = 3) in vec3 inTexCoord;
```

```
out vec3 vNormal;
out vec4 kolorek;

out vec2 texCoord;
out float fogCoord;

void main()
{
        vec4 EyeSpaceCoord = modelViewMatrix*vec4(inPosition, 1.0);
        gl_Position = projectionMatrix*EyeSpaceCoord;
        vec4 vRes = normalMatrix*vec4(inNormal, 0.0);
        vNormal = vRes.xyz;
        kolorek = vec4(inColor,ColorAlpha);
        texCoord = vec2(inTexCoord[0],inTexCoord[1]);
        fogCoord = abs(EyeSpaceCoord.z);
}
```

2. Rozbuduj shader fragmentów do postaci z listingu poniżej:

```
#version 330

uniform vec3 LightColor;
uniform vec3 LightDirection;
uniform float LightAmbient;
uniform sampler2D gSampler;

in vec4 kolorek;
in vec3 vNormal;
in vec2 texCoord;
in float fogCoord;

out vec4 outputColor;

uniform float FogDensity;

vec3 FogColor = vec3(0.6,0.6,0.6);
vec4 FragmentColor;
void main()
{
        float FogFactor = 1.0 - clamp(exp(-FogDensity*fogCoord),0.0,1.0);
        float LightDiffuse = max(0.0, dot(normalize(vNormal), -LightDirection));
        FragmentColor = kolorek*texture2D(gSampler, texCoord)
            *vec4(LightColor*(LightAmbient+LightDiffuse),1.0);
        outputColor = mix(FragmentColor,vec4(FogColor,1.0),FogFactor);
}
```

3. Zapewnij sterowanie i przekazywanie zmienną `FogDensity` z poziomu aplikacji. Wykorzystaj klawisze F5 oraz F6. Jako początkową wartość dla zmiennej przyjmij np. 0.05 podobną wartość wykorzystaj jako inkrement w sterowaniu wartością zmiennej

4. Skompiluj i uruchom program. Sprawdź, jak współdziałają efekty nakładania oświetlenia, przezroczystości, teksturowania i zamglenia
5. Wyjaśnij działanie funkcji `clamp` oraz `mix` użytych w implementacji zamglenia
6. Zmodyfikuj równanie mgły do postaci kwadratowej.
7. Rozbuduj kod tworzący mgłę wykorzystując równanie liniowe.