Listing 11.25. Przekazywanie wektora normalnego do shadera fragmentów

```
...
varying vec3 vOrgNormal;
void main(void) {
    ...
   vOrgNormal = aNormal;
}
```

Shader fragmentów oblicza współrzędne sferyczne na podstawie otrzymanego wektora i wykorzystuje je jako współrzędne tekstury. W sferycznym układzie współrzędnych punkt jest opisywany przez odległość radialną oraz dwa kąty, zwykle oznaczane greckimi literami θ (*theta*) i Φ (*phi*). Zależność między współrzędnymi w układzie kartezjańskim i sferycznym przyjmuje postać równania:

```
radius = sqrt(x*x + y*y + z*z)
theta = acos(y / radius)
phi = atan(z / x)
```

Wszystkie punkty na powierzchni sfery są oddalone od jej środka o jednakową odległość, więc ważne są jedynie dwa kąty. Z uwagi na to, że wykorzystaliśmy wektor normalny zamiast współrzędnych wierzchołka, wiemy, że jest on równy 1, więc równanie zostanie uproszczone do postaci:

```
theta = acos(y)
phi = atan(z / x)
```

Jeżeli użyjesz tych działań w shaderze fragmentów — zgodnie z treścią listingu 11.26 — będziesz mógł wykorzystać zmienne `theta` i `phi` w roli współrzędnych tekstury.

Listing 11.26. Shader fragmentów i tekstura sferyczna

```
...
varying vec3 vOrgNormal;
uniform sampler2D uTexture;
void main(void) {
    ...
    float theta = acos(vOrgNormal.y);
    float phi = atan(vOrgNormal.z, vOrgNormal.x);
    vec2 texCoord = vec2(-phi / 2.0, theta) / 3.14159;
    vec4 texColor = texture2D(uTexture, texCoord);
    vec3 color = texColor.rgb * (uAmbient + diffuse + specular);
    gl_FragColor = vec4(color, 1.0);
}
```

Współrzędne tekstur mieszczą się w przedziale od 0 do 1, ale kąty `theta` i `phi` są podane w radianach. Kąt `theta` znajduje się w przedziale od 0 do Π, podczas gdy kąt `phi` w przedziale od 0 do 2Π. Z uwagi na tę rozbieżność współrzędne dzielone są przez Π, a zmienna `phi` dodatkowo dzielona jest przez 2. Wartość pikseli jest pobierana z samej tekstury za pomocą funkcji `texture2d()` i właśnie wyliczonych współrzędnych tekstury. Plik *05-texture.html* zawiera pełen kod odpowiedzialny za renderowanie oteksturowanej kuli, którą możesz podziwiać na rysunku 11.6.

Na tym kończy się nasza wycieczka po API standardu WebGL. Mam nadzieję, że pomogła Ci ona zrozumieć jego działanie i ułatwi Ci tworzenie trójwymiarowej grafiki w grach i aplikacjach. Następny podrozdział poświęcę opisowi konstrukcji modułu graficznego WebGL dla *Brylantowego wojownika*.