Laboratorium Grafiki Komputerowej i Animacji

Ćwiczenie V

Biblioteka OpenGL - oświetlenie sceny

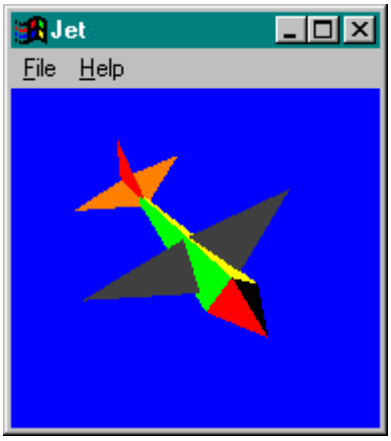
Sławomir Samolej

Rzeszów, 1999

1. Wstęp

Większość tworzonych animacji w grafice komputerowej ma za zadanie odzwierciedlanie rzeczywistości. Istotnym elementem każdego dobrego systemu projektowania trójwymiarowych scen jest możliwość generowania i modyfikowania źródeł światła, które oświetlając skonstruowane obiekty pozwalają na osiągnięcie złudzenia trójwymiarowości. Biblioteka OpenGL umożliwia zdefiniowanie i nadanie określonych właściwości kilku źródłom światła (podstawowy standard zapewnia możliwość utworzenia do 8 źródeł oświetlenia). Wraz z możliwością kreowania oświetlenia sceny, standard OpenGL pozwala na określenie sposobu reagowania na światło obiektów zdefiniowanych na scenie.

Opracowanie prezentuje zdefiniowane w OpenGL model światła, model materiału pokrywającego oświetlane obiekty, a także metodologię tworzenia i określania właściwości źródeł światła. Dołączona do opracowania dyskietka zawiera kody źródłowe i skompilowane wersje programów służących do ilustrowania omawianych zagadnień. Podstawowym obiektem, dla którego w ramach opracowania rozbudowywany będzie model oświetlenia jest model samolotu jak na rysunku 1.1 (program JET). Model samolotu skonstruowano zgodnie z zasadami omawianymi w opracowaniach do ćwiczeń 2 i 3 z przedmiotu Grafika Komputerowa i Animacja.



Rys 1.1 Model samolotu

2. Model światła OpenGL

Standard OpenGL stara się przybliżyć światło występujące w świecie rzeczywistym w postaci trzech składników: światła otaczającego (ang. ambient light), światła rozproszonego (ang. diffuse light), światła kierunkowego (ang. specular light).

Światło otaczające nie biegnie do obiektu z jakiegoś konkretnego kierunku. Posiada ono źródło, ale promienie zanim dochodzą do obiektu kilkakrotnie uległy odbiciu i nie biegną w jedna stronę. Obiekty oświetlone przez ten rodzaj światła są równomiernie oświetlone na wszystkich powierzchniach i ze wszystkich kierunków. Można założyć, że wszystkie prezentowane do tej pory przykłady aplikacji z wykorzystaniem OpenGL zawierały obiekty oświetlone w ten sposób (obiekty były zawsze widoczne i kolory wszystkich powierzchni były równomiernie nasycone niezależnie od orientacji, czy położenia).

Światło rozproszone biegnie z określonego kierunku ale od powierzchni, na którą pada odbija się równomiernie we wszystkich kierunkach. Pomimo tego, że światło rozproszone odbija się równomiernie we wszystkich kierunkach powierzchnie obiektów są jaśniejsze, gdy światło pada na nie prostopadle i ciemniejsze, gdy powierzchnia oświetlana ustawiona jest

pod innym kątem niż prosty. Dobrym przykładem światła rozproszonego (w notacji OpenGL) jest światło wpadające przez okno w pokoju.

Światło kierunkowe, podobnie jak rozproszone biegnie z określonego kierunku. Ale odbija się w ściśle określonym kierunku. Skupione światło kierunkowe powoduje pojawienie się jasnej plamy na oświetlonej powierzchni. Odpowiednikami światła kierunkowego w rzeczywistości są reflektory lub Słońce.

Kombinacja trzech wymienionych typów światła pozwala na przybliżanie rzeczywistych zjawisk świetlnych występujących w przyrodzie. Przykładowo snop czerwonego światła laserowego w laboratorium składa się prawie w całości z czerwonego światła kierunkowego. Obecność w powietrzu drobin kurzu powoduje rozpraszanie światła i w taki sposób w powietrzu widoczna jest smuga światła znacząca bieg promieni lasera. Smuga ta odpowiada światłu rozproszonemu. Jeśli w pomieszczeniu nie ma żadnego innego źródła światła, to wszystkie przedmioty znajdujące się w laboratorium mają niewielką czerwoną poświatę. Efekt takiego oświetlenia można uznać za bardzo niewielki składnik światła otaczającego.

Do skomponowania modelu oświetlenia sceny należy podać założone proporcje intensywności poszczególnych składników światła, a także określić składowe RGB koloru światła (składową Alpha światła przyjmujemy równą 1.0). Przykładowe określenie właściwości źródła światła w omawianej scenie z czerwonym laserem w laboratorium można podać jak w tabeli poniżej.

	Czerwony (R)	Zielony (G)	Niebieski (B)	Alpha
Św. kierunkowe	0,99	0,0	0,0	1,0
Św. rozproszone	0,10	0,0	0,0	1,0
Św. otaczające	0,05	0,0	0,0	1,0

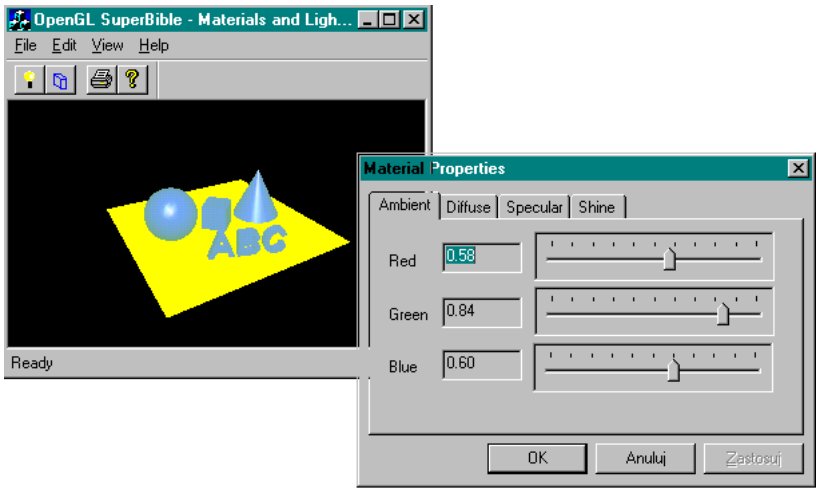
3. Model materiału OpenGL

Kolor światła określa tylko część właściwości oświetlanych obiektów. W rzeczywistym świecie obiekty posiadają swój własny kolor. Kolor przedmiotu można określić jako zdolność odbijania fotonów reprezentujących określoną długość fali świetlnej przy równoczesnym pochłanianiu większości pozostałych fotonów. Przykładowo niebieska piłka odbija większość niebieskich fotonów, a pochłania większość pozostałych. Jeśli światło oświetlające piłkę zawiera niebieskie fotony, to zostaną one odbite od piłki i będzie ona widoczna przez obserwatora sceny. Większość scen odzwierciedlających rzeczywistość oświetlanych jest białym światłem będącym równomierną mieszaniną wszystkich długości fal świetlnych. Poddane białemu oświetleniu wszystkie obiekty posiadają swój "naturalny" kolor. Jeśli jednak wspomnianą niebieską piłkę umieścimy w ciemnym pokoju i oświetlimy czystym żółtym światłem, dla obserwatora piłka będzie koloru czarnego (żółte światło zostanie przez piłkę w całości pochłonięte).

Od momentu rozpoczęcia posługiwania się oświetleniem w OpenGL nie opisujemy wielokątów poprzez ich kolor, a raczej poprzez właściwości materiału, którym są pokryte. Mówiąc, że dany wielokąt jest czerwony powinniśmy mieć na myśli, że wielokąt zbudowany jest z materiału, który odbija większość czerwonego światła. Oprócz tego, że zdefiniowaliśmy kolor materiału, należy określić w jaki sposób dany materiał reagować będzie na poszczególne składniki modelu światła.

Dobranie odpowiednich właściwości źródeł światła i cech materiałów wymaga doświadczenia. Pomocnym narzędziem do eksperymentowania nad doborem parametrów oświetlenia sceny może być załączony na dyskietce program MATLIGHT. Umożliwia on

interaktywne modyfikowanie składników światła i własności materiałów oraz obserwację zachodzących na scenie zmian (rys 3.1).



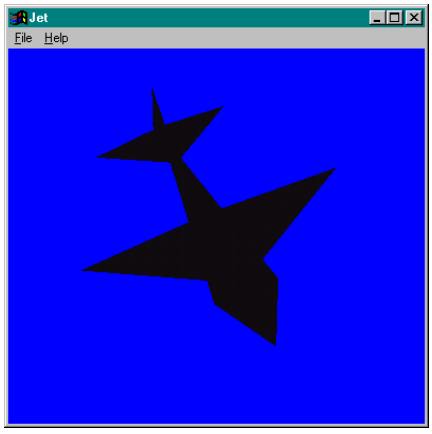
Rys 3.1 Okna programu MATLIGHT.

4. Kreowanie oświetlenia sceny

Aby włączyć do sceny OpenGL obliczenia związane z oświetleniem należy wywołać funkcję glEnable() z parametrem **GL_LIGHTING**:

```
glEnable(GL_LIGHTING);
```

Jeśli jednak do kodu programu nie zostaną dodane linie określające własności materiałów oraz parametry oświetlenia, dotychczasowo rysowane kolorowe obiekty widoczne będą w oknie programu jako ciemne zarysy (rys 4.1)



Rys 4.1 Okno programu JET z włączonym obliczaniem oświetlenia
ale z nie określonymi właściwościami materiałów i światła

Po uaktywnieniu obliczania światła w scenie należy określić w programie model oświetlenia. Do ustawiania modelu oświetlenia służy w standardzie OpenGL funkcja **glLightModel*()**. Może ona przyjmować kilka predefiniowanych stałych jako parametry.

Ważniejsze z nich omówione zostaną w dalszej części opracowania. Jednym z częstszych parametrów wywołania funkcji glLightModel*() jest GL_LIGHT_MODEL_AMBIENT pozwalający na zdefiniowanie światła otaczającego na scenie. Fragment kodu:

```
// Jasne białe światło otaczające - pełna intensywność
// wartości RGB
GLfloat ambientLight[] = { 1.0f, 1.0f, 1.0f, 1.0f };

glEnable(GL_LIGHTING);              // Uaktywnij obliczanie
                                    // oświetlenia

// Ustaw model oświetlenia światła otaczającego zdefiniowany w
// wektorze ambientLight
glLightModelfv(GL_LIGHT_MODEL_AMBIENT,ambientLight);
```

definiuje jasne białe światło otaczające na scenie. Wariant wywołania funkcji glLightModel*() zaprezentowany powyżej (glLightModel**v**()) pobiera jako pierwszy parametr stałą określającą modyfikowany, bądź ustawiany model oświetlenia, zaś jako drugi - wektor określający model. Inne parametry określające model oświetlenia pozwalają ustalić, które z powierzchni wielokątów (przednie czy tylne) mają być brane pod uwagę przy obliczaniu oświetlenia, oraz sposób obliczania kątów odbicia światła kierunkowego.

Kolejnym etapem tworzenia oświetlenia sceny jest ustalenie własności materiałów. Są dwa sposoby zdefiniowania własności materiałów. Pierwszy z nich polega na wykorzystaniu funkcji **glMaterial*()** przed definiowaniem każdego z wielokątów lub zbioru wielokątów. Przeanalizujmy następujący fragment kodu:

```
GLfloat gray[]={0.75f, 0.75f, 0.75f, 1.0f,};
...
...
glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, gray);
glBegin(GL_TRIANGLES);
      glVertex3f(-15.0f, 0.0f, 30.0f);
      glVertex3f(0.0f, 15.0f, 30.0f);
      glVertex3f(0.0f, 0.0f, -56.0f);
glEnd();
```

Pierwszy parametr funkcji glMaterialfv() ustala którą ze stron wielokąta dotyczyć będzie wywołanie funkcji: przednią (GL_FRONT), tylnią (GL_BACK), czy obu (GL_FRONT_AND_BACK). Drugi parametr określa, która z właściwości materiału będzie definiowana (W prezentowanym fragmencie kodu określany jest sposób reakcji materiału na światło otaczające i rozproszone: GL_AMBIENT_AND_DIFFUSE). Ostatnim parametrem funkcji jest wektor wartości RGBA decydujący o sposobie reakcji materiału na określony kolor światła padającego na niego. Do momentu ponownego wywołania funkcji glMaterial*() wszystkie tworzone prymitywy posiadają określone przez tę funkcję właściwości.

Drugim, preferowanym sposobem ustalania własności materiałów na scenie jest tak zwane śledzenie kolorów (ang. color tracking). Śledzenie kolorów w OpenGL polega na określaniu własności materiałów na podstawie ich kolorów zdefiniowanych przy pomocy znanej funkcji **glColor()**. Aby uaktywnić w systemie OpenGL ten typ obliczania własności materiałów należy wywołać funkcję glEnable() z parametrem **GL_COLOR_MATERIAL**:

```
glEnable(GL_COLOR_MATERIAL);
```

Funkcja **glColorMaterial()** umożliwia ustawienie parametrów materiału, które w dalszej części kodu programu określane będą przy pomocy komendy glColor(). Wywołanie:

```
glColorMaterial(GL_FRONT,GL_AMBIENT_AND_DIFFUSE);
```

oznacza, że do obliczania własności materiałów, na które będzie padać światło otaczające i rozproszone służyć będą kolory ustalone przy pomocy funkcji glColor(). Poniższy fragment kodu prezentuje fragment programu wykorzystujący śledzenie kolorów do określania własności materiałów:

```
// Uaktywnij śledzenie kolorów
glEnable(GL_COLOR_MATERIAL);

// Ustal właściwości materiałów, które będą wyliczane na podstawie
// parametrów funkcji glColor()
glColorMaterial(GL_FRONT, GL_AMBIENT_AND_DIFFUSE);

...
...

glColor3f(0.75f, 0.75f, 0.75f);
glBegin(GL_TRIANGLES);
      glVertex3f(-15.0f, 0.0f, 30.0f);
      glVertex3f(0.0f, 15.0f, 30.0f);
      glVertex3f(0.0f, 0.0f, -56.0f);
glEnd();
```

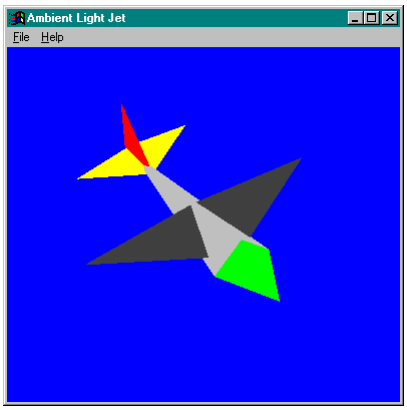
Fragment kodu, który zostanie podany poniżej zawiera funkcję SetupRC() dołączoną do kodu programu JET. Funkcja ta wywołana w momencie inicjalizacji pracy programu (obsługa komunikatu WM_CREATE) definiuje jasne światło otaczające na scenie oraz ustawia własności materiałów w taki sposób, że każdy wielokąt jest widoczny i może odbijać światło.

```
void SetupRC()
      {
      GLfloat ambientLight[] = { 1.00f, 1.00f, 1.00f, 1.0f };

      glEnable(GL_DEPTH_TEST);
      glEnable(GL_CULL_FACE);
      glFrontFace(GL_CCW);

      glEnable(GL_LIGHTING);

      glLightModelfv(GL_LIGHT_MODEL_AMBIENT,ambientLight);

      glEnable(GL_COLOR_MATERIAL);

      glColorMaterial(GL_FRONT,GL_AMBIENT_AND_DIFFUSE);

      glClearColor(0.0f, 0.0f, 0.5f,1.0f);
      }
```

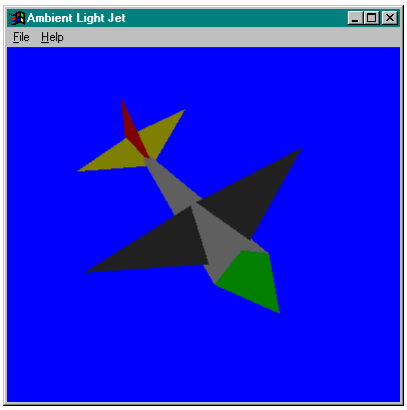
Zmodyfikowany program JET o nazwie AMBIENT znajduje się na załączonej do opracowania dyskietce. W porównaniu z programem JET, program AMBIENT ma przedefiniowane kolory poszczególnych części samolotu. Otrzymany na ekranie efekt nie różni się od programów, które nie miały w ogóle zdefiniowanego światła (rys. 4.2). Wystarczy jednak przedefiniować tablicę określającą parametry oświetlenia:

```
GLfloat ambientLight[] = { 0.50f, 0.50f, 0.50f, 1.0f };
```

aby otrzymać efekt jak na rysunku 4.3. Obraz obiektu jest zaciemniony. Efekt powyższy stosować można do symulowania zmierzchu oraz zacienionych fragmentów sceny.



Rys. 4.2 Okno programu AMBIENT



Rys 4.3 Okno programu AMBIENT po zredukowaniu intensywności światła otaczającego o połowę

Jeśli scena graficzna ma symulować rzeczywistość, konieczne jest zdefiniowanie źródła światła, które posiada swoje położenie i wysyła promieniowanie w określonym kierunku. OpenGL w podstawowej wersji dopuszcza zdefiniowanie do 8 niezależnych źródeł światła. Istnieje możliwość zdefiniowania zarówno świateł rozsyłających promienie we wszystkie strony jak i świateł o wąskim snopie - reflektorów. W efekcie pojawienia się na scenie światła padającego z określonego kierunku na poszczególne ściany trójwymiarowych obiektów pada różna ilość promieniowania świetlnego. Powoduje to, że bardziej oświetlone ściany są jaśniejsze, a mniej - ciemniejsze. Istotne znaczenie dla obliczeń efektów świetlnych ma obliczenie kąta padania promieni świetlnych na poszczególne powierzchnie obiektu. Do obliczenia tego kąta należy dla każdej ściany obiektu zdefiniować normalną do niej (W zwykłych przypadkach normalną definiuje się jako prostą prostopadłą do płaszczyzny zawierającej wielokąt, możliwe jest jednak dla osiągnięcia specjalnych efektów oświetleniowych podanie systemowi OpenGL współrzędnych dowolnej prostej jako "normalnej" do powierzchni). Standard OpenGL dopuszcza określenie normalnej dla każdego wierzchołka wielokąta osobno. Normalną do wierzchołka definiuje się w OpenGL przy pomocy funkcji **glNormal*()**, której trzema parametrami są współrzędne wektora wyznaczającego kierunek poszukiwanej prostej prostopadłej. W programie JET dla jednego z trójkątów normalna określona została w następujący sposób:

```
glBegin(GL_TRIANGLES);
      glNormal3f(0.0f, -1.0f, 0.0f);
      glVertex3f(0.0f, 0.0f, 60.0f);
      glVertex3f(-15.0f, 0.0f, 30.0f);
      glVertex3f(15.0f,0.0f,30.0f);
glEnd();
```

Aby obliczenia dotyczące oświetlenia odbywały się prawidłowo wszystkie wektory normalne powinny posiadać długość 1. Przekształcenie dowolnego wektora w wektor jednostkowy skierowany w ten sam sposób polega na obliczeniu długości wektora a następnie podzieleniu każdej z jego współrzędnych przez długość. Proces redukcji długości wektorów do jedności często nazywa się normalizacją. W standardzie OpenGL można zlecić środowisku

automatyczne normalizowanie wszystkich zdefiniowanych wektorów normalnych przy pomocy komendy:

```
glEnable(GL_NORMALIZE);
```

Powoduje to jednak zwykle pewne spowolnienie szybkości przeprowadzanych obliczeń sceny. Dla poprawienia wydajności pracy programu warto stworzyć własną funkcję redukującą długość wektora, której zasada działania omówiona została powyżej. Przykładowa realizacja funkcji może wyglądać w następujący sposób:

```
// Funkcja normalizująca wektor podany jako zbiór trzech współrzędnych
void ReduceToUnit(float vector[3])
      {
      float length;

      // Oblicz długość wektora
      length = (float)sqrt(   (vector[0]*vector[0]) +
                              (vector[1]*vector[1]) +
                              (vector[2]*vector[2]));

      // Zabezpieczenie przed podziałem przez 0
      if(length == 0.0f)
            length = 1.0f;

      // Podziel każdą ze współrzędnych przez długość wektora
      vector[0] /= length;
      vector[1] /= length;
      vector[2] /= length;
      }
```

Do znajdowania normalnych do wielokątów usytuowanych w dowolny sposób w przestrzeni posłużyć się można definicją iloczynu wektorowego dwu wektorów. Jeśli w standardzie OpenGL zdefiniowany jest jakikolwiek wielokąt, to zawsze znamy współrzędne jego wierzchołków. Trzy z nich wyznaczają zawsze dwa wektory. Podana poniżej funkcja wyznacza znormalizowany wektor normalny do wielokąta:

```
// Punkty p1, p2 i p3 zdefiniowane w odwrotnym do wskazówek zegara
// porządku
void calcNormal(float v[3][3], float out[3])
      {
      float v1[3],v2[3];
      static const int x = 0;
      static const int y = 1;
      static const int z = 2;

      // Oblicz 2 wektory na podstawie trzech punktów
      v1[x] = v[0][x] - v[1][x];
      v1[y] = v[0][y] - v[1][y];
      v1[z] = v[0][z] - v[1][z];

      v2[x] = v[1][x] - v[2][x];
      v2[y] = v[1][y] - v[2][y];
      v2[z] = v[1][z] - v[2][z];

      // Oblicz współrzędne wektora normalnego na podstawie
      // iloczynu wektorowego
      out[x] = v1[y]*v2[z] - v1[z]*v2[y];
      out[y] = v1[z]*v2[x] - v1[x]*v2[z];
      out[z] = v1[x]*v2[y] - v1[y]*v2[x];
```

```
    // Normalizuj wektor
    ReduceToUnit(out);
    }
```

Prezentowany poniżej fragment kodu pochodzi z funkcji SetupRC() programu LIJET:

```
void SetupRC()
    {
    // Parametry i koordynaty źródła światła
    GLfloat  ambientLight[] = { 0.3f, 0.3f, 0.3f, 1.0f };
    GLfloat  diffuseLight[] = { 0.7f, 0.7f, 0.7f, 1.0f };
    GLfloat      lightPos[] = { -50.f, 50.0f, 100.0f, 1.0f };

    glEnable(GL_DEPTH_TEST);
    glFrontFace(GL_CCW);
    glEnable(GL_CULL_FACE);

    // Uaktywnienie obliczania oświetlenia
    glEnable(GL_LIGHTING);

    // Ustawienie i uaktywnienie źródła światła 0
    glLightfv(GL_LIGHT0,GL_AMBIENT,ambientLight);
    glLightfv(GL_LIGHT0,GL_DIFFUSE,diffuseLight);
    glLightfv(GL_LIGHT0,GL_POSITION,lightPos);
    glEnable(GL_LIGHT0);

    // Uaktywnienie śledzenia kolorów
    glEnable(GL_COLOR_MATERIAL);

    // ustawienie własności materiałów
    glColorMaterial(GL_FRONT, GL_AMBIENT_AND_DIFFUSE);

    // Kolor tła
    glClearColor(0.0f, 0.0f, 1.0f, 1.0f );
    }
```

Kolejne linie programu tworzą źródło światła, umieszczają je za obserwatorem nieco na lewo i do góry. Źródło światła zawiera składowe otaczającą i rozproszoną o intensywnościach określonych w tablicach ambientLight[] i diffuseLight[] (umiarkowane białe światło).

```
    GLfloat  ambientLight[] = { 0.3f, 0.3f, 0.3f, 1.0f };
    GLfloat  diffuseLight[] = { 0.7f, 0.7f, 0.7f, 1.0f };

    ...
    // Ustaw i uaktywnij źródło światła
    glLightfv(GL_LIGHT0,GL_AMBIENT,ambientLight);
    glLightfv(GL_LIGHT0,GL_DIFFUSE,diffuseLight);
```

Pozycję światła ustalają linie kodu:

```
    GLfloat      lightPos[] = { -50.f, 50.0f, 100.0f, 1.0f };

    ...
    glLightfv(GL_LIGHT0,GL_POSITION,lightPos);
```

Tablica lightPos[] zawiera położenie źródła światła. Ostatnia wartość w tablicy wynosi 1.0. Oznacza to, że poprzednie wartości określają współrzędne położenia źródła światła. Jeśli ostatnią wartością w tablicy byłoby 0.0, to oznaczałoby to, że źródło światła jest w nieskończoności, a wektor określa kierunek świecenia. Linia kodu:

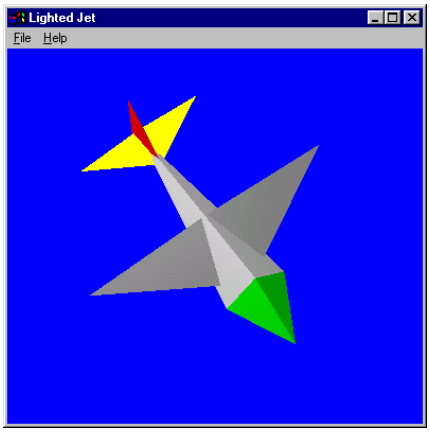
```
glEnable(GL_LIGHT0);
```

uaktywnia zdefiniowane na scenie światło.

Włączenie obliczeń dotyczących światła na scenie powoduje, że modyfikacji ulegają fragmenty kodu programu dotyczące definiowania wielokątów:

```
float normal[3];  // Do przechowywania obliczonych współrzędnych
                  // wektora normalnego
...

// Ustal kolor materiału
glRGB(0, 255, 0);
glBegin(GL_TRIANGLES);
      glNormal3f(0.0f, -1.0f, 0.0f);
      glVertex3f(0.0f, 0.0f, 60.0f);
      glVertex3f(-15.0f, 0.0f, 30.0f);
      glVertex3f(15.0f,0.0f,30.0f);
//glEnd();


{
// Wierzchołki kolejnego trójkąta
float v[3][3] =   {     { 15.0f, 0.0f, 30.0f},
                        { 0.0f, 15.0f, 30.0f},
                        { 0.0f, 0.0f,     60.0f}};

// Oblicz normalną do powierzchni
calcNormal(v,normal);

// Rysuj trójkąt wyznaczając tę samą normalną dla
// wszystkich wierzchołków
//glBegin(GL_TRIANGLES);
      glNormal3fv(normal);
      glVertex3fv(v[0]);
      glVertex3fv(v[1]);
      glVertex3fv(v[2]);
//glEnd();
}
```

Do obliczeń wektorów normalnych do poszczególnych powierzchni wyznaczających model samolotu wykorzystywana jest omówiona wcześniej funkcja calcNormal. Ponieważ każde trzy kolejne wierzchołki w kodzie tworzyć mają trójkąt zrezygnowano z linii glBegin() i glEnd() przed definicjami każdego z wielokątów. Okno programu LIJET wyświetlającego model samolotu oświetlony przy pomocy światła biegnącego z konkretnego punktu w przestrzeni pokazuje rysunek 4.4.



Rys. 4.4 Okno programu LIJET

Zdefiniowanie składowej otaczającej i rozproszonej światła umożliwia tworzenie podstawowych efektów świetlnych, takich jak cieniowanie. Nie jest możliwe jednak zdefiniowanie przy ich pomocy metalicznego połysku wybranych elementów sceny. Do tworzenia tego rodzaju efektów służy trzeci składnik modelu światła - światło kierunkowe. Efekt odbłysku osiąga się wtedy, kiedy na obiekt pada skierowany strumień światła i odbija się od niego w kierunku, z którego patrzy obserwator sceny. Odpowiednie właściwości nadane muszą także być materiałowi, z którego tworzony jest obiekt. Podany poniżej kod pokazuje, jak w przykładowy sposób uzupełnić można program LIJET o źródło światła kierunkowego:

```
GLfloat  ambientLight[] = { 0.3f, 0.3f, 0.3f, 1.0f };
GLfloat  diffuseLight[] = { 0.7f, 0.7f, 0.7f, 1.0f };
GLfloat  specular[] = { 1.0f, 1.0f, 1.0f, 1.0f};
GLfloat lightPos[] = { 0.0f, 150.0f, 150.0f, 1.0f };
...
glEnable(GL_LIGHTING);

glLightfv(GL_LIGHT0,GL_AMBIENT,ambientLight);
glLightfv(GL_LIGHT0,GL_DIFFUSE,diffuseLight);
glLightfv(GL_LIGHT0,GL_SPECULAR,specular);
glLightfv(GL_LIGHT0,GL_POSITION,lightPos);
glEnable(GL_LIGHT0);
```

Tablica specular[] definiuje białe jasne źródło światła kierunkowego, a linia:

```
glLightfv(GL_LIGHT0,GL_SPECULAR,specular);
```

dodaje do światła LIGHT0 komponent kierunkowy. Do uzyskania efektu odblasku należy zmodyfikować właściwości materiału, z którego stworzony jest oświetlany obiekt:

```
GLfloat  specref[] =  { 1.0f, 1.0f, 1.0f, 1.0f };
...

glEnable(GL_COLOR_MATERIAL);

glColorMaterial(GL_FRONT, GL_AMBIENT_AND_DIFFUSE);

// Wszystkie elementy obiektu odbijają światło kierunkowe
// i posiadają najwyższy współczynnik odblasku
glMaterialfv(GL_FRONT, GL_SPECULAR,specref);
glMateriali(GL_FRONT,GL_SHININESS,50);
```

Podobnie jak w poprzednich przykładach zdefiniowane jest śledzenie kolorów w przypadku obliczania efektów oświetlenia dla składowej otaczającej i rozproszonej. Tablica specref[] definiuje kolor odbijanego światła, a linia kodu:

```
glMaterialfv(GL_FRONT, GL_SPECULAR,specref);
```

uaktywnia nową własność materiału.

Jeśli właściwości światła kierunkowego i materiału zdefiniowane zostały jak powyżej, to oświetlone powierzchnie obiektu stawałyby się automatycznie białe. Linia kodu:

```
glMateriali(GL_FRONT,GL_SHININESS,50);
```

pozwala zmieniać stopień skupienia padającego światła kierunkowego i przez to uzyskiwać rzeczywiste efekty odbłysków. Ostatni parametr wywołania funkcji może przyjmować wartości od 0 do 128. Wartość 0 oznacza maksymalnie rozszerzoną wiązkę (uzyskuje się wtedy niepożądany efekt białych powierzchni), parametr 128 - maksymalnie skupioną. Rysunek 4.5 przedstawia okno programu SHINYJET wykorzystującego zaprezentowane elementy oświetlenia do animowania połyskującego w świetle modelu samolotu.

Rys. 4.5 Okno programu SHINYJET

Poniżej podany jest pełny kod funkcji SetupRC() programu SHINYJET:

```
void SetupRC()
      {
      GLfloat  ambientLight[] = { 0.3f, 0.3f, 0.3f, 1.0f };
      GLfloat  diffuseLight[] = { 0.7f, 0.7f, 0.7f, 1.0f };
      GLfloat  specular[] = { 1.0f, 1.0f, 1.0f, 1.0f};
      GLfloat      lightPos[] = { 0.0f, 150.0f, 150.0f, 1.0f };
      GLfloat  specref[] =  { 1.0f, 1.0f, 1.0f, 1.0f };


      glEnable(GL_DEPTH_TEST);
      glFrontFace(GL_CCW);
      glEnable(GL_CULL_FACE);

      glEnable(GL_LIGHTING);

      glLightfv(GL_LIGHT0,GL_AMBIENT,ambientLight);
      glLightfv(GL_LIGHT0,GL_DIFFUSE,diffuseLight);
      glLightfv(GL_LIGHT0,GL_SPECULAR,specular);
      glLightfv(GL_LIGHT0,GL_POSITION,lightPos);
      glEnable(GL_LIGHT0);

      glEnable(GL_COLOR_MATERIAL);

      glColorMaterial(GL_FRONT, GL_AMBIENT_AND_DIFFUSE);

      glMaterialfv(GL_FRONT, GL_SPECULAR,specref);
      glMateriali(GL_FRONT,GL_SHININESS,128);

      glClearColor(0.0f, 0.0f, 1.0f, 1.0f );
      }
```

5. Reflektory

Tworzenie reflektorów zbliżone jest do kreowania innych skierowanych źródeł światła. Poniżej podany zostanie kod funkcji SetupRC() programu SPOT. Program umieszcza niebieską sferę w centrum okna. Reflektor zbudowany jest w taki sposób, że może być poruszany przy pomocy klawiszy strzałek.

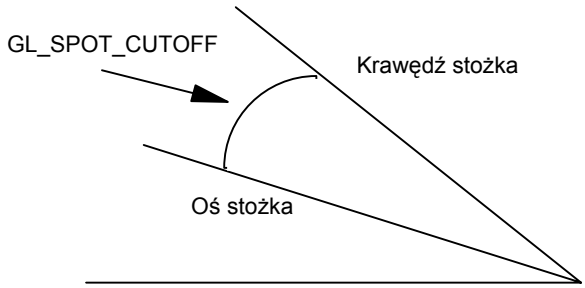
```
// właściwości i koordynaty oświetlenia
GLfloat      lightPos[] = { 0.0f, 0.0f, 75.0f, 1.0f };
GLfloat  specular[] = { 1.0f, 1.0f, 1.0f, 1.0f};
GLfloat  specref[] =  { 1.0f, 1.0f, 1.0f, 1.0f };
GLfloat  ambientLight[] = { 0.5f, 0.5f, 0.5f, 1.0f};
GLfloat  spotDir[] = { 0.0f, 0.0f, -1.0f };

void SetupRC()
      {
      glEnable(GL_DEPTH_TEST);
      glFrontFace(GL_CCW);
      glEnable(GL_CULL_FACE);

      // Uaktywnij tworzenie oświetlenia
      glEnable(GL_LIGHTING);

      // Ustaw i uaktywnij źródło światła 0, dodaj trochę
      // światła otaczającego aby obiekt był widoczny
      glLightModelfv(GL_LIGHT_MODEL_AMBIENT, ambientLight);

      // Światło komponowane jest tylko ze składników rozproszonego
      // i kierunkowego
      glLightfv(GL_LIGHT0,GL_DIFFUSE,ambientLight);
      glLightfv(GL_LIGHT0,GL_SPECULAR,specular);
      glLightfv(GL_LIGHT0,GL_POSITION,lightPos);

      // Tworzenie efektu reflektora
      // kąt strumienia światła: 60 stopni
      glLightf(GL_LIGHT0,GL_SPOT_CUTOFF,60.0f);

      // Współczynnik rozchodzenia się światła
      glLightf(GL_LIGHT0,GL_SPOT_EXPONENT,100.0f);

      // Uaktywnienie światła
      glEnable(GL_LIGHT0);

      // Uaktywnienie śledzenia kolorów
      glEnable(GL_COLOR_MATERIAL);

      // Ustalenie własności materiałów
      glColorMaterial(GL_FRONT, GL_AMBIENT_AND_DIFFUSE);

      // Wszystkie materiał będą miał pełną zdolność odblasku
      glMaterialfv(GL_FRONT, GL_SPECULAR,specref);
      glMateriali(GL_FRONT, GL_SHININESS,128);

      // Czarne tło
      glClearColor(0.0f, 0.0f, 0.0f, 1.0f );
      }
```

Komendy tworzące reflektor to:

```
      glLightf(GL_LIGHT0,GL_SPOT_CUTOFF,60.0f);
```

```
glLightf(GL_LIGHT0,GL_SPOT_EXPONENT,100.0f);
```

Wartość GL_SPOT_CUTOFF definiuje kąt rozwarcia stożka rozchodzenia się promieni świetlnych (do 180 stopni). Jeśli fragment oświetlanego obiektu znajduje się poza stożkiem oświetlenia, nie jest on oświetlany. Rysunek 5.1 przedstawia definicję kąta rozwarcia stożka.



Rys 5.1 Definicja kąta rozwarcia stożka stosowana w OpenGL

Włączenie jako drugiego parametru wywołania funkcji glLight*() stałej GL_SPOT_EXPONENT pozwala na zdefiniowanie stopnia skupienia wiązki światła padającej z reflektora. Im trzeci parametr wywołania funkcji jest większy (może przyjmować wartości 0-128) tym wiązka światła padająca z reflektora jest bardziej skupiona.

Jeśli stosuje się na scenie reflektory, to należy w jakiś sposób zaznaczyć ich obecność. W programie SPOT w miejscu źródła światła umieszczono czerwony stożek, a na końcu stożka żółtą sferę imitującą żarówkę. Podany poniżej fragment programu zawiera kod funkcji RenderScene() programu SPOT. Należy zwrócić uwagę na linię:

```
glPushAttrib(GL_LIGHTING_BIT);
```

po której następuje chwilowe wyłączenie obliczania światła na scenie i narysowanie jasnej sfery imitującej żarówkę. A następnie na linię:

```
glPopAttrib();.
```

Pierwsza z wyróżnionych funkcji zachowuje ustalone parametry oświetlenia sceny. Następnie wykonywane jest wyłączenie liczenia oświetlenia sceny na czas wyrysowania "żarówki". Funkcja glPopAttrib() przywraca poprzednie parametry oświetlenia sceny.

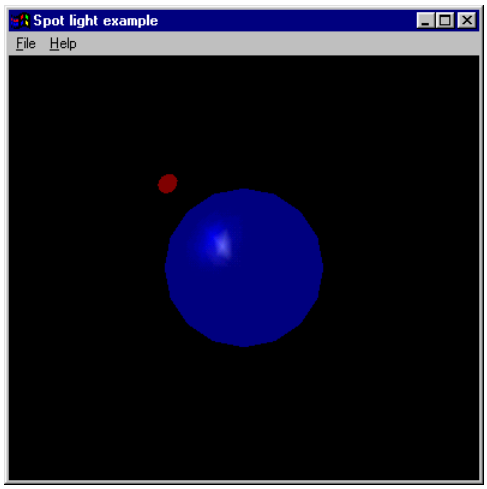
```
void RenderScene(void)
{
// Wyczyść okno bieżącym kolorem tła
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

// Ustal kolor materiału i wyświetl sferę w środku okna
glRGB(0, 0, 255);
auxSolidSphere(30.0f);

// Umieść źródło oświetlenia
// Zachowaj bieżące położenie lokalnego uk. współrzędnych
glPushMatrix();
      // Obróć lokalny układ współrzędnych
      glRotatef(yRot, 0.0f, 1.0f, 0.0f);
      glRotatef(xRot, 1.0f, 0.0f, 0.0f);

      // Podaj nowe położenie światła
      glLightfv(GL_LIGHT0,GL_POSITION,lightPos);
```

```
        glLightfv(GL_LIGHT0,GL_SPOT_DIRECTION,spotDir);

        // Wyrysuj czerwony stożek do oznaczenia położenia reflektora
        glRGB(255,0,0);

        // Przesuń układ w celu dostosowania położenia światła do
        //stożka
        glTranslatef(lightPos[0],lightPos[1],lightPos[2]);
        auxSolidCone(4.0f,6.0f);

        // Wyrysuj "żarówkę", zachowaj stan oświetlenia sceny
        glPushAttrib(GL_LIGHTING_BIT);

              // Wyłącz oświetlenie sceny
              glDisable(GL_LIGHTING);
              glRGB(255,255,0);
              auxSolidSphere(3.0f);

        // Odzyskaj poprzednie parametry oświetlenia
        glPopAttrib();

// Odzyskaj zachowane położenie lokalnego układu współrzędnych
glPopMatrix();

glFlush();
}
```

Funkcja RenderScene() programu SPOT zawiera dodatkowe wywołanie komendy glLight*() postaci:

```
glLightfv(GL_LIGHT0,GL_SPOT_DIRECTION,spotDir);
```

Parametr GL_SPOT_DIRECTION wskazuje, że dane zawarte w wektorze podanym jako ostatni parametr wywołania komendy definiowały będą kierunek promieni światła wychodzących z reflektora. Rysunek 5.2 prezentuje przykładowe okno programu SPOT.



Rys. 5.2 Przykładowe okno programu SPOT

Bibliografia:

[1] R. S. Wright Jr., M. Sweet, *OpenGL Superbible*, The Waite Group Press, 1996
[2] *OpenGL Programming Guide*, Addison-Wessley, 1993
[3] E. Angel, *Interactive Computer Graphics*, Addison-Wesley, 1997
[4] R. Fosner, *OpenGL Programming for Windows and Windows NT*, Addison-Wesley, 1997
[5] R. Leniowski, *Wykłady z przedmiotu Grafika Komputerowa i Animacja*
[6] *Guide to OpenGL® on Windows® From Silicon Graphics®*, 1997