OpenGL

Zaawansowana grafika komputerowa

Czym jest OpenGL

- OpenGL można zdefiniować jako "programowy interfejs sprzętu graficznego". Jest to biblioteka przeznaczona do tworzenia trójwymiarowej grafiki, bardzo szybka i łatwo przenaszalna pomiędzy różnymi systemami.
- OpenGL jest używany do różnych celów, od programów typu CAD, przez aplikacje architektoniczne aż do generowania komputerowych dinozaurów w najnowszych filmach.

Czym jest OpenGL

- OpenGL jest względnie nowym standardem przemysłowym, opracowanym zaledwie kilka lat temu, jednak już powszechnie zaakceptowanym. Przodkiem OpenGL był własny język **GL** firmy Silicon Graphics dla stacji roboczych IRIS.
- OpenGL jest rezultatem prac firmy SGI nad poprawą przenaszalności biblioteki IRIS GL. Nowy język oferuje moc biblioteki GL, lecz jest przy tym otwarty, pozwalając na łatwą adaptację dla różnych platform sprzętowych i systemów operacyjnych.

Jak działa OpenGL

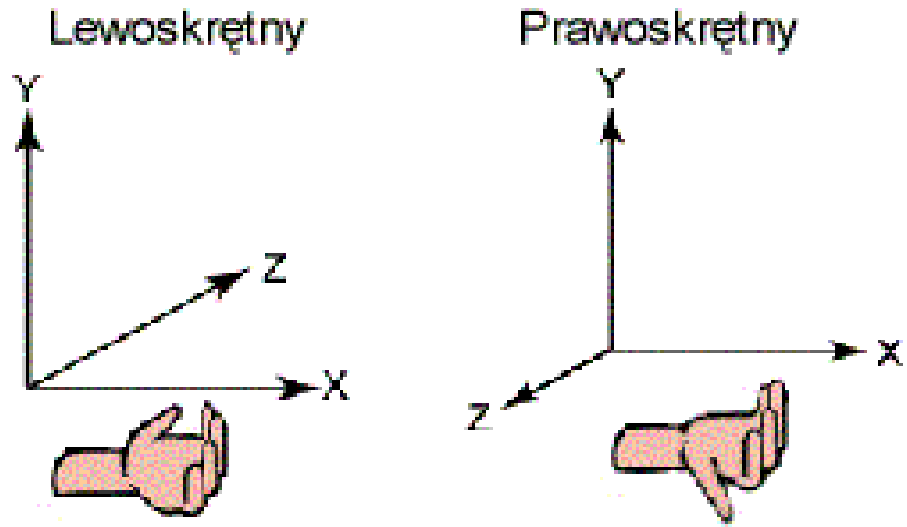
- OpenGL jest biblioteką procedur, za pomocą których programista zamiast opisywać samą scenę oraz jej wygląd, opisuje kroki konieczne do osiągnięcia określonego wyglądu lub efektu.
- OpenGL nie zawiera żadnych funkcji przeznaczonych do zarządzania oknami, interakcji z użytkownikiem czy operacji wejścia/wyjscia

Układ współrzędnych

Istnieją różne układy współrzędnych, które są stosowane w zależności od tego jakie istnieją potrzeby i jakie dane są dostępne. Kartezjański układ współrzędnych to taki, który na płaszczyźnie tworzą dwie, a w przestrzeni trzy wzajemnie prostopadle proste - noszą nazwę osi układu, a miejsce ich przecięcia jest jego początkiem. Położenie punktu określa się przez podanie odległości od początku układu do punktów otrzymanych przez rzutowanie prostopadłe danego punktu na poszczególne osie.

Układ współrzędnych

Aplikacje generujące grafikę 3D używają zasadniczo dwóch rodzajów układów współrzędnych kartezjańskich: prawo- i lewoskrętnego.



Układ współrzędnych

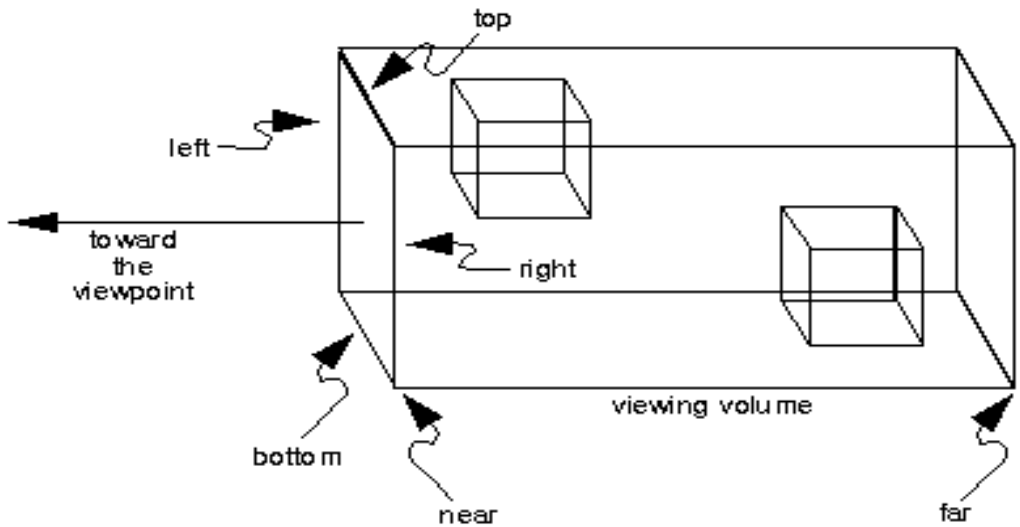
W obu przypadkach oś X leży poziomo i jest skierowana w prawo. Oś Y jest osią pionową i jest skierowana w górę. Różnica istnieje w kierunku osi Z. Oś ta jest zawsze położona prostopadle do powierzchni ekranu (jeśli dla uproszczenia założymy, że jest on całkowicie płaski, a jej kierunek możemy określić na podstawie ułożenia lewej lub prawej ręki).

Układ współrzędnych

Dla lewoskrętnego układu współrzędnych, jeśli wszystkie palce lewej ręki skierujemy w dodatnią stronę osi X i zagniemy je tak aby pokazywały w kierunku dodatniej osi Y to odchylony kciuk pokaże nam dodatni kierunek osi Z (w tym przypadku "w głąb" monitora). Dla prawoskrętnego układu robimy to samo, tylko z wykorzystaniem prawej ręki. Wszystkie nasze przykłady będą używać lewoskrętnego układu współrzędnych.

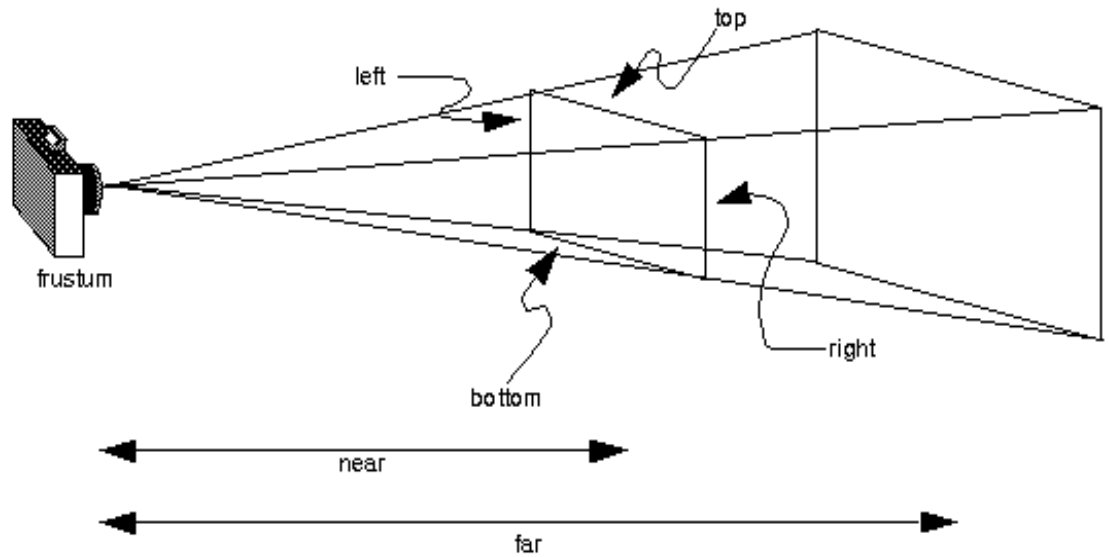
Rzutowanie

W OpenGL mamy do czynienia z dwoma głównymi rodzajami rzutów. Pierwszy rodzaj to rzutowanie równoległe. Tego typu rzutowanie określa się stosując prostokątną lub sześcienną bryłę rzutowania. Wszystkie obiekty są rysowane z zachowaniem swoich rozmiarów bez względu na to w jakiej są odległości.



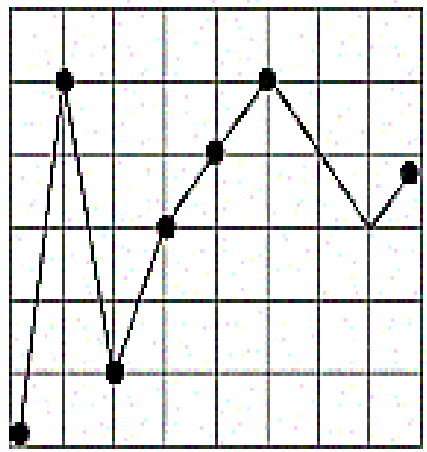
Rzutowanie

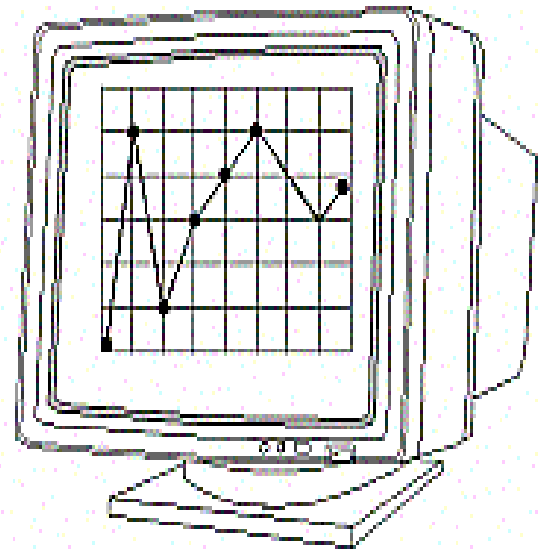
Drugim popularniejszym typem rzutowania są rzuty perspektywiczne. To rzutowanie powoduje powstanie efektu zmniejszenia rozmiarów bardziej odległych obiektów. Bryła widzenia przypomina piramidę ze ściętym wierzchołkiem.



Widoki

Odwzorowanie logicznych współrzędnych kartezjańskich na fizyczne współrzędne pikseli okna nazywane jest widokiem. Zwykle widok jest definiowany jako całe okno.





Rysowanie prymitywów

- Zarówno w dwóch jak i w trzech wymiarach, gdy rysujemy obiekt, w rzeczywistości składamy go z serii mniejszych kształtów, zwanych prymitywami. Prymitywy to proste obiekty, takie jak punkty, odcinki i płaskie wielokąty składane w przestrzeni 3D w celu utworzenia trójwymiarowych obiektów.
- Każda operacja rysowania w OpenGL zawarta jest pomiędzy wywołaniami funkcji **glBegin** oraz **glEnd**

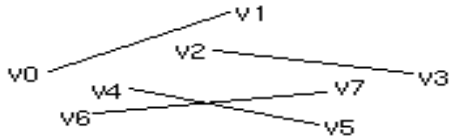
Rysowanie prymitywów

W openGL dostępne są następujące prymitywy:

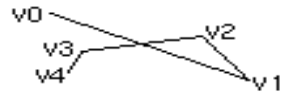
- GL_POINTS
- GL_LINES
- GL_LINE_STRIP
- GL_LINE_LOOP
- GL_POLYGON
- GL_QUADS
- GL_QUAD_STRIP
- GL_TRIANGLES
- GL_TRIANGLE_STRIP
- GL_TRIANGLE_FAN

Rysowanie prymitywów

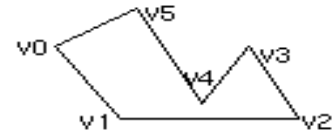
GL_POINTS



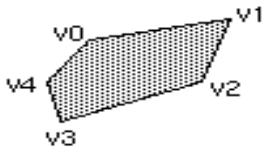
GL_LINES



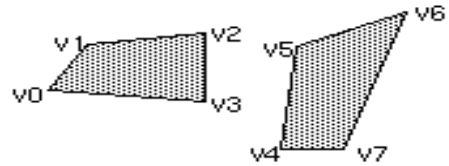
GL_LINE_STRIP



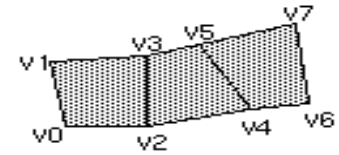
GL_LINE_LOOP



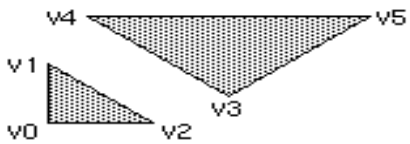
GL_POLYGON



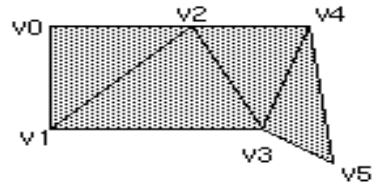
GL_QUADS



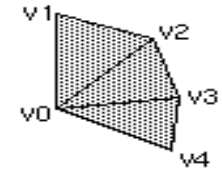
GL_QUAD_STRIP



GL_TRIANGLES



GL_TRIANGLE_STRIP



GL_TRIANGLE_FAN

Typy danych OpenGL

Aby ułatwić przenoszenie kodu OpenGL pomiędzy różnymi platformami, OpenGL definiuje własne typy danych. Typy te odnoszą się do normalnych typów danych języka C, których także można używać

Typ danych OpenGL	Wewnętrzna reprezentacja	Zdefiniowane jako typ C	Przyrostek nazwy zmiennej w C
GLbyte	8-bitowa liczba całkowita	signed char	b
GLshort	16-bitowa liczba całkowita	short	s
GLint, GLsizei	32-bitowa liczba całkowita	long	l
GLfloat, GLclampf	32-bitowa liczba zmiennoprzecinkowa	float	f
GLdouble, GLclampd	64-bitowa liczba zmiennoprzecinkowa	double	ub

Typy danych OpenGL

Typ danych OpenGL	Wewnętrzna reprezentacja	Zdefiniowane jako typ C	Przyrostek nazwy zmiennej w C
GLubyte, GLboolean	8-bitowa liczba całkowita bez znaku	unsigned char	us
GLushort	16-bitowa liczba całkowita bez znaku	unsigned short	ui
GLuint, GLenum, GLbitfield	32-bitowa liczba całkowita bez znaku	unsigned long	

Wszystkie typy danych rozpoczynają się od GL, co oznacza zmienną OpenGL. Większość posiada także przyrostek określający odpowiedni typ danych języka C (byte, short, int, float itd.)

Konwencje nazw funkcji

Wszystkie funkcje OpenGL zostały nazwane zgodnie z określoną konwencją, informującą o bibliotece, z której pochodzi funkcja, oraz o rodzaju i typie jej argumentów.

gl Color 3f (...)

- **gl** Boblioteka gl
- **Color** Rdzeń nazwy
- **3** Liczba argumentów
- **f** Typ argumentów

Rysowanie w trzech wymiarach

- Rysowanie punktów:

```
glBegin(GL_POINTS);                          // wybranie punktow
   glVertex3f(0.0f, 0.0f, 0.0f);             // pierwszy punkt
   glVertex3f(50.0f, 50.0f, 50.0f);          // drugi punkt
glEnd();                                     // koniec rysowania
```

- Rysowanie linii:

```
glBegin(GL_LINES);                           // wybranie lini
   glVertex3f(0.0f, 0.0f, 0.0f);             // pierwszy wierzcholek
   glVertex3f(50.0f, 50.0f, 50.0f);          // drugi wierzcholek
glEnd();                                     // koniec rysowania
```

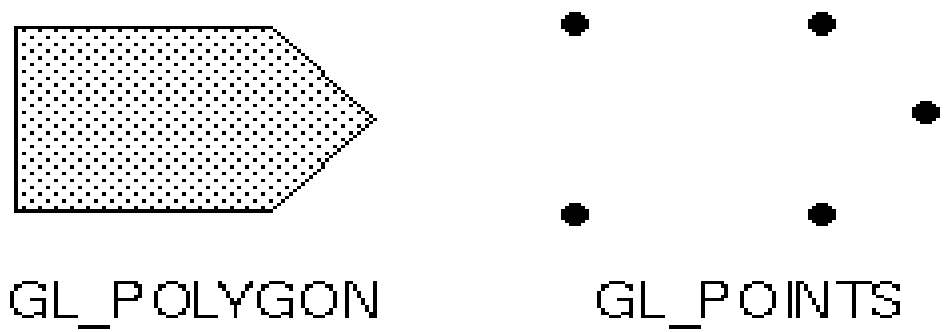
- Rysowanie trójkątów:

```
glBegin(GL_TRIANGLES);                       // wybranie trojkatow
   glVertex3f(0.0f, 0.0f, 0.0f);             // pierwszy wierzcholek
   glVertex3f(25.0f, 25.0f, 0.0f);           // drugi wierzcholek
   glVertex3f(50.0f, 50.0f, 0.0f);           // trzeci wierzcholek
glEnd();                                     // koniec rysowania
```

Rysowanie w trzech wymiarach

- Tryb wielokątów:

```
glBegin(GL_POLYGON);                    // wybranie wielokatow
   glVertex2f(0.0f, 0.0f);              // P1
   glVertex2f(0.0f, 3.0f);              // P2
   glVertex2f(3.0f, 3.0f);              // P3
   glVertex2f(4.0f, 1.5f);              // P4
   glVertex2f(3.0f, 0.0f);              // P5
glEnd();                                // koniec rysowania
```



Przekształcenia OpenGL

Przekształcenia układu współrzędnych

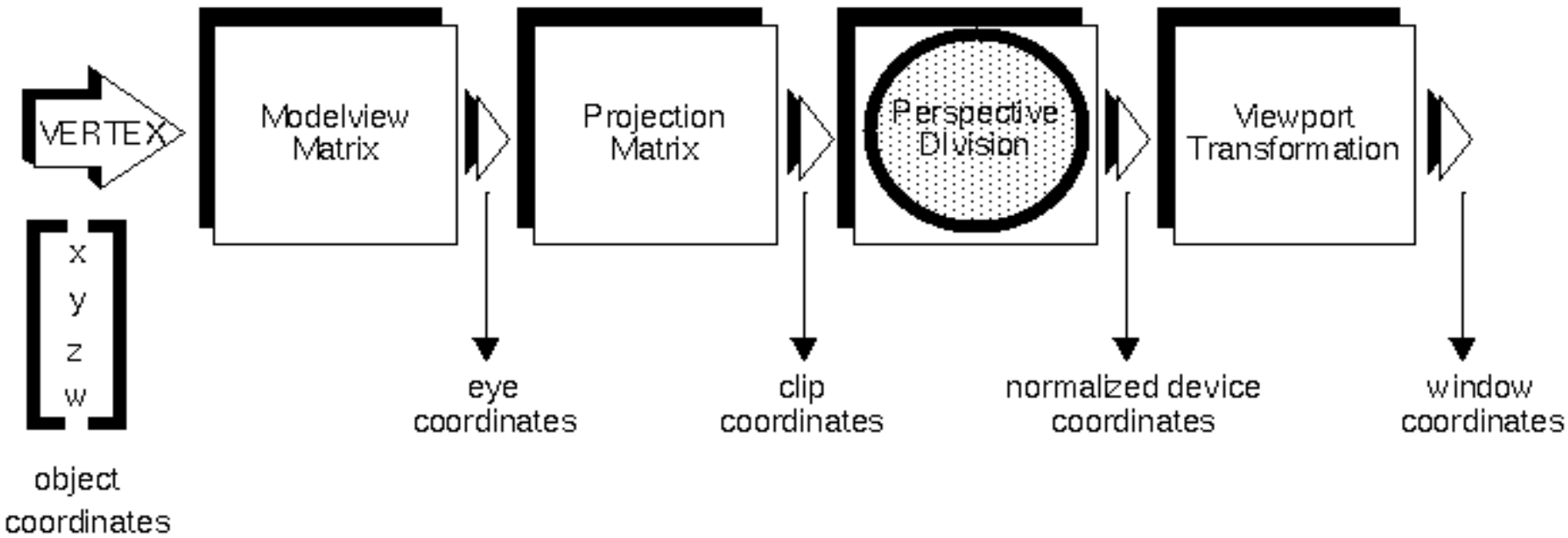
Pozwalają nam przemieszczać i manipulować obiektami w wirtualnym świecie. Podczas tworzenia trójwymiarowej grafiki wierzchołki obiektów poddawane są trzem rodzajom przekształceń, zanim zostaną narysowane na ekranie:

- przekształceniom widoku, które określają położenie kamery
- przekształceniom modelowania, które przemieszczają obiekt na scenie
- przekształceniom rzutowania, które definiują bryłę widoku oraz płaszczyzny obcięcia

Przekształcenie	Opis
Widoku	Określa położenie kamery
Modelowania	Przemieszcza obiekt na scenie
Rzutowanie	Definiuje bryłę widoku oraz płaszczyzny obcięcia
Okienkowe	Odwzorowuje dwuwymiarowy rzut sceny w oknie
Widoku modelu	Stanowi kombinację przekształcenia widoku i przekształcenia modelowania

Przekształcenia OpenGL

Przekształcenia muszą być zawsze wykonywane w odpowiedniej kolejności. przekształcenie widoku musi zawsze poprzedzać przekształcenie modelowania. Rzutowanie i przekształcenia okienkowe muszą natomiast poprzedzać tworzenie grafiki na ekranie.



Macierz modelowania

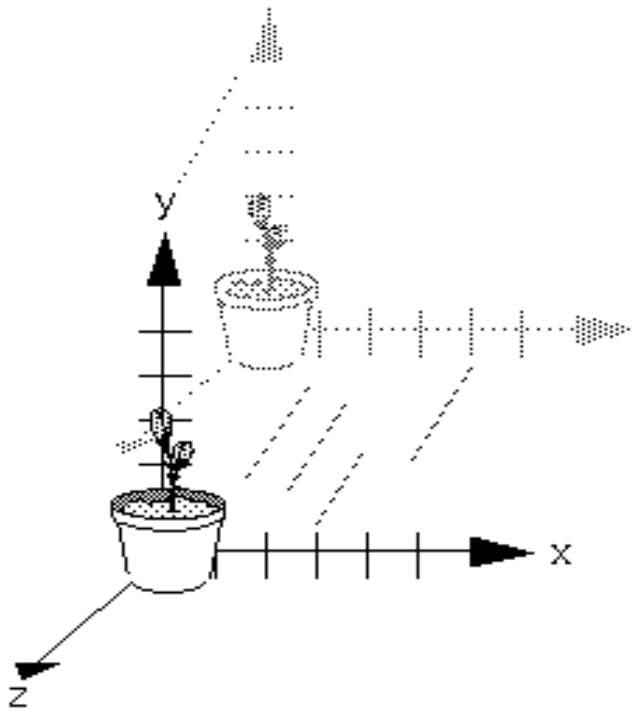
Macierz modelowania definiuje układ współrzędnych wykorzystywany podczas tworzenia obiektów. Macierz ta posiada wymiar 4x4 i mnożona jest o wektory reprezentujące wierzchołki oraz macierze przekształceń

- **Przesunięcie**
  Pozwala przemieścić obiekt z jednego punktu przestrzeni do innego punktu. OpenGL umożliwia wykonanie przesunięć za pomocą funkcji `glTranslatef()` i `glTranslated()` zdefiniowanych jako:
  ```
  void glTranslatef(GLfloat x, GLfloat y, GLfloat z);
  void glTranslated(GLdouble x, GLdouble y, GLdouble z);
  ```
- **Obrót**
  Również funkcja umożliwiająca wykonywanie obrotów w OpenGL posiada dwie wersje rózniace się typem parametrów:
  ```
  void glRotatef(GLfloat angle, GLfloat x, GLfloat y, GLfloat z);
  void glRotated(GLdouble angle, GLdouble x,
                 GLdouble y, GLdouble z);
  ```

Przesunięcie

przykładowy fragment kodu realizujący przesunięcie obiektu na scenie może wyglądać następująco:

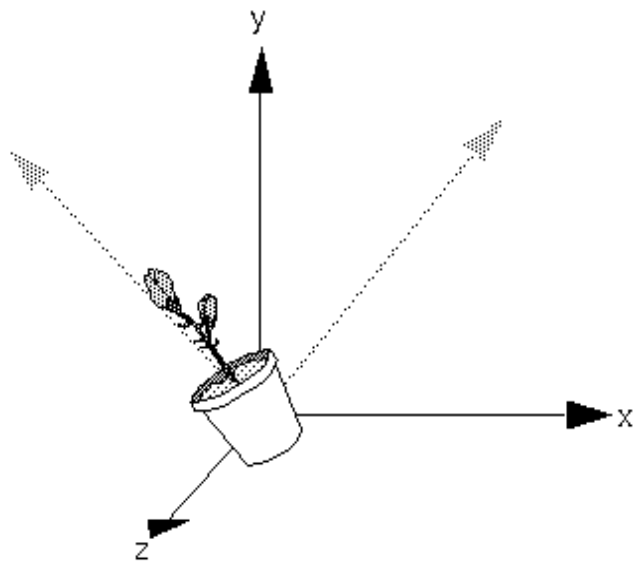
```
glMatrixModel(GL_MODELVIEW)      // wybranie macierzy modelowania
glLoadIdentity();                // resetuje macierz modelowania
glTranbslatef(0.0f, 0.0f, 5.0f); // przesuniecie do punktu (0,0,5)
RysujObiekt();                   // rysuje obiekt
```



Obrót

Wykonanie obrotu dookoła dowolnej osi wymaga określenia wszystkich wartości parametrów x, y, z. Przykładowy kod realizujący obrót:

```
glMatrixModel(GL_MODELVIEW)          // wybranie macierzy modelowania
glLoadIdentity();                    // resetuje macierz modelowania
glRotatef(45.0f, 0.0f, 0.0f, 1.0f); // obrot o 45 stopni dookoła osi z
RysujObiekt();                       // rysuje obiekt
```



Skalowanie

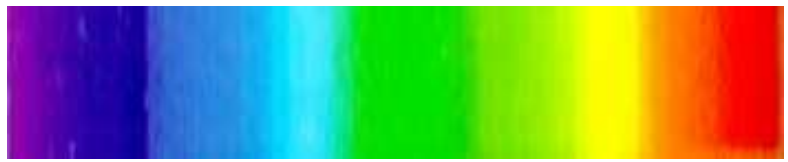
Skalowanie polega na zmianie rozmiarów obiektu. W jego wyniku wierzchołki obiektu zbliżają się lub oddalają od siebie wzdłuż osi osi układu współrzędnych zgodnie z wartościami odpowiadającymi współczynnikowi skalowania.

```
void glScalef(GLfloat x, GLfloat y, GLfloat z);
void glScaled(GLdouble x, GLdouble y, GLdouble z)
```

Podanie wartości ujemnej jako współczynnika skalowania spowoduje efekt odbicia obiektu względem podanej osi układu współrzędnych.

Kolory w OpenGL

Widzialny zakres światła tworzą fale o długości od 390 nanometrów (nm) do 720 nm. W przedziale tym mieści się pełne spektrum tęczy: począwszy od fioletu, przez błękit, zieleń, żółcień, pomarańcz aż do czerwieni.



Głębia koloru

określa maksymalną liczbę kolorów, które może posiadać piksel i zależy od rozmiaru bufora koloru. Rozmiar ten może wynosić 4,8,16 i więcej bitów. 8-bitowy bufor koloru może przechowywać informacje o jednym z 2^8, czyli 256 kolorów.

Model RGBA w OpenGL

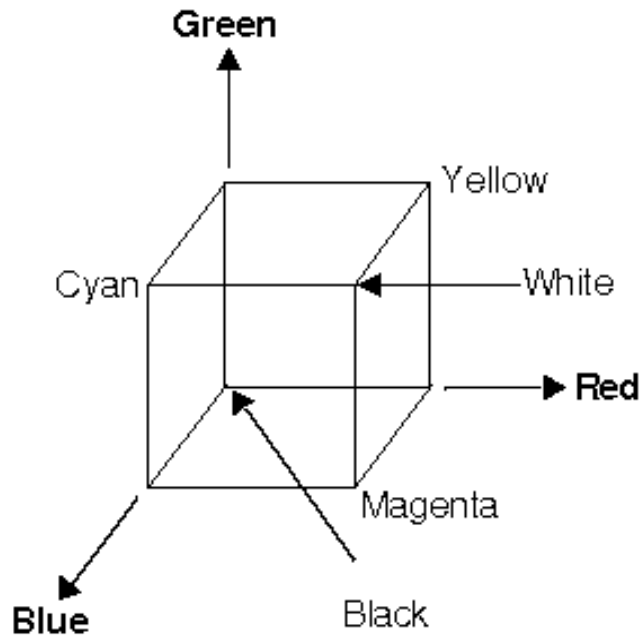
Specyfikując kolor w modelu RGBA przekazuje się fukcji `glColor*()` wartości składowej czerwonej, zielonej i niebieskiej. Funkcja `glColor*()` posiada kilkanaście wersji różniących się przyrostkami dodawanymi do jej nazwy. Do najczęściej używanych wersji należą:

```
void glColor3f(GLfloat r, GLfloat g, GLfloat b);
void glColor4f(GLfloat r, GLfloat g, GLfloat b, GLfloat a);
void glColor3fv(const GLfloat *v);
void glColor4fv(const  GLfloat *v);
```

Pierwsza z funkcji wybiera bieżący kolor na podstawie wartości składowej czerwonej reprezentowanej przez parametr *r*, zielonej *g* i niebieskiej *b*. Druga z nich posiada dodatkowy parametr *a* reprezentujący wartość współczynnika alfa określającego sposób łączenia kolorów. Wartości parametrów funkcji muszą należeć do przedziału od 0.0 do 1.0

Sześcian kolorów

OpenGL opisuje kolory za pomocą intensywności ich składowej czerwonej, zielonej i niebieskiej. Intensywność poszczególnych składowych może przyjmować wartości z przedziału 0 do 1. Rysunek prezentuje za pomocą sześcianu kolorów to, w jaki sposób kombinacje różnej intensywności poszczególnych składowych tworzą spektrum kolorów.



Cieniowanie

Posługując się przykładem odcinka łączącego dwa wierzchołki o różnych kolorach, dla uproszczenia przyjmiemy, że pierwszy z nich jest w kolorze czarnym, a drugi jest biały. Jaki będzie kolor łączącego je odcinka ?. Można to ustalić posługując się modelem cieniowania.

Cieniowanie może być płaskie lub gładkie. Cieniowanie płaskie wykonywane jest za pomocą pojedynczego koloru. Zwykle jest to kolor ostatniego wierzchołka (jedynie w przypadku trybu rysowania wielokątów o dowolnej liczbie wierzchołków wybranego za pomocą stałej `GL_POLYGON` jest to kolor pierwszego z wierzchołków). Cieniowanie gładkie zwane też cieniowaniem `Gouraud` stosuje natomiast interpolację koloru pikseli odcinka.

Jeśli dla naszego odcinka zastosujemy cieniowanie płaskie, to będzie on miał kolor biały, ponieważ ostatni z wierzchołków odcinka ma taki kolor. Natomiast w przypadku cieniowania gładkiego kolor pikseli będzie zmieniać się począwszy od koloru czarnego pierwszego wierzchołka poprzez coraz jaśniejsze odcienie szarości aż do koloru białego drugiego z wierzchołków.

Oświetlenie

Można teraz zastanowić się nad tym, w jaki sposób oświetlenie wpływa na wygląd obiektów w rzeczywistym świecie. Obraz widziany w oku powstaje na skutek podrażnienia siatkówki przez fotony. Fotony te muszą mieć określone źródło, a do oka trafiają na skutek odbicia przez oglądane obiekty. Nie wszystkie fotony, które padają na obiekt zostają odbite, ponieważ część ich zostaje pochłonięta. Stopień odbicia padającego światła przez obiekt zależy od materiału z jakiego jest on wykonany. Obiekty, które widziane są jako połyskujące, odbijają większość światła. Natomiast obiekty, które pochłaniają znaczną część padającego na nie światła lub odbijają ją w takim kierunku, że nie trafiają do oka, odbierane są jako ciemniejsze.

OpenGL wyznacza efekt oświetlenia obiektów w podobny sposób. Kolor obiektu zależy od stopnia odbicia przez niego składowej czerwonej, zielonej i niebieskiej padającego światła. Stopień ten OpenGL określa na podstawie rodzaju materiału, na który pada światło.

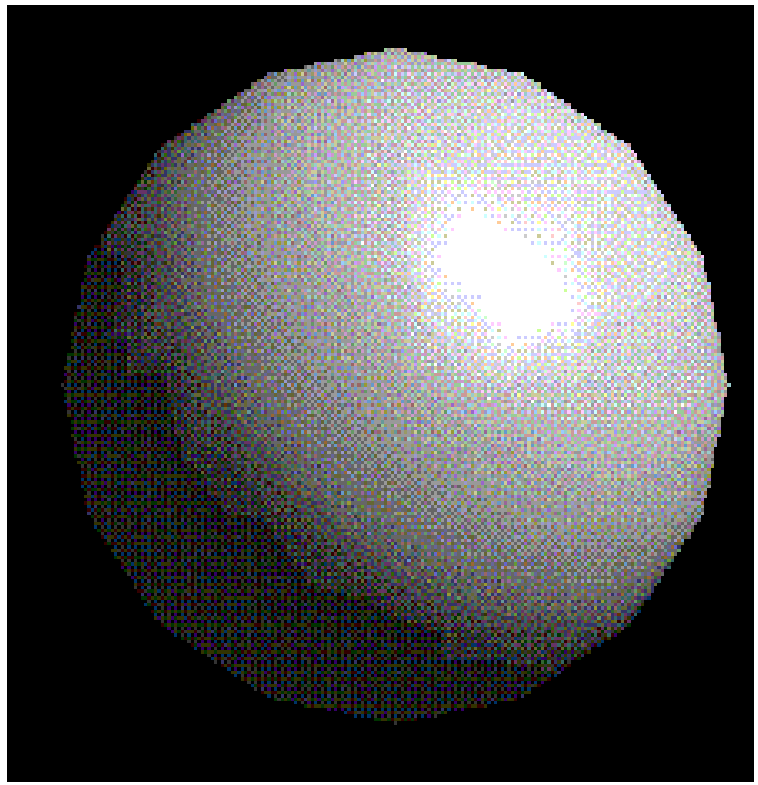
Oświetlenie

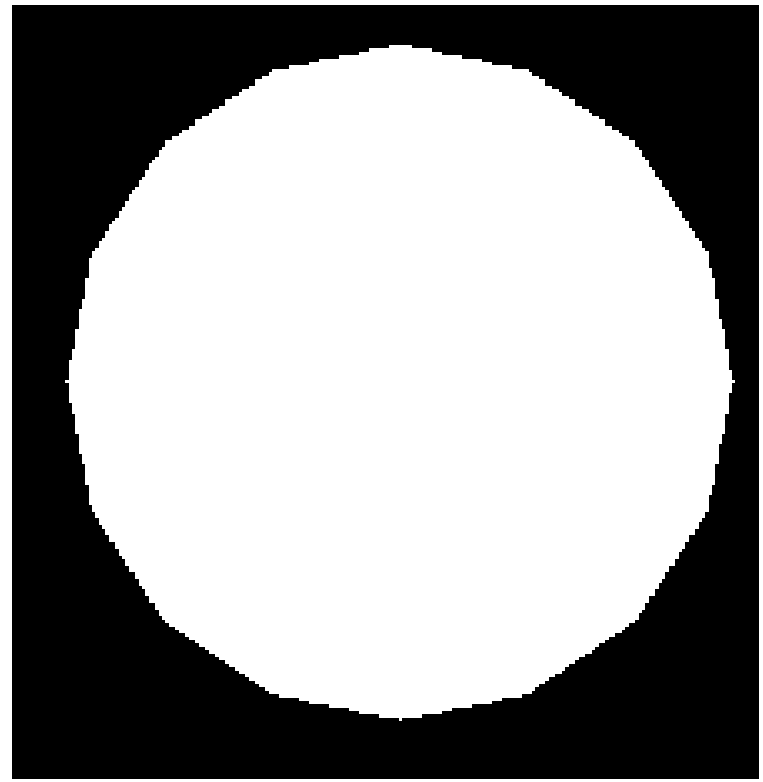
OpenGL umożliwia modelowanie oświetlenia za pomocą czterech komponentów:

- **światła otoczenia** - światło to przedstawia się tak, jakby nie posiadało źródła (zostało rozproszone w takim stopniu, że ustalenie jego źródła nie jest możliwe)
- **światła rozproszonego** - światło to pada z określonego kierunku, ale zostaje odbite przez obiekt równomiernie we wszystkich kierunkach, dzięki czemu niezależnie od położenia obserwatora obiekt wydaje się oświetlony równomiernie
- **światło odbicia** - światło pada z określonego kierunku i odbijane jest także w jednym kierunku tworząc efekt odbicia
- **światło emisji** - obiekty wydzielające światło emisji posiadają większą intensywność kolorów (światło emisji nie ma wpływu na wygląd pozostałych obiektów, a inne źródła światła nie mają wpływu na światło emisji danego obiektu)

Oświetlenie

Światło ma również wpływ na to, że większość nie oświetlonych obiektów trójwymiarowych wygląda jak płaskie obiekty jeśli nie są oświetlone.



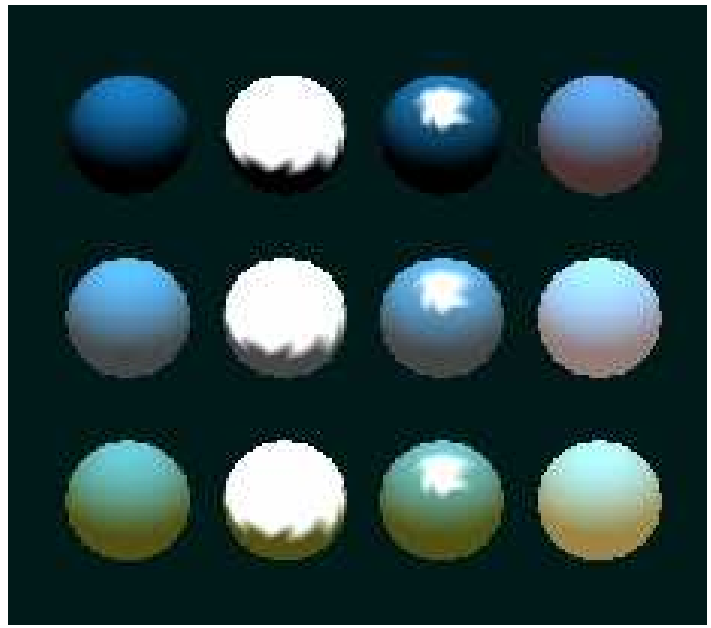


Materiały

OpenGL wyznacza kolor obiektu na podstawie stopnia odbicia przez jego materiał składowej czerwonej, zielonej i niebieskiej światła. Na przykład powierzchnia w kolorze zielonym odbija w całości składowa zieloną światła, a pochłania składową czerwoną i niebieską. Jeśli więc będzie padać na nią światło posiadające wyłącznie składową czerwoną, to będzie można zobaczyć powierzchnię w kolorze czarnym, ponieważ oświetlające ją światło zostanie pochłonięte w całości. Natomiast jeżeli oświetlona zostanie zielona powierzchnia białym światłem, to będzie ona miała kolor zielony, ponieważ składowa zielona zostanie odbita, a składowe niebieska i czerwona zostaną pochłonięte. Podobnie będzie się działo, jeśli oświetlona zostanie ona za pomocą źródła emitującego kolor zielony, ponieważ odbije w całości składową zieloną padającego na nią światła.

Materiały

Materiał opisuje się za pomocą tych samych właściwości co światło. Właściwości te opisują stopień odbicia poszczególnych rodzajów światła przez dany materiał. Jedne materiały będą na przykład dobrze odbijać jedynie światło otoczenia, a pochłaniać światło rozproszone i światło odbicia. Inne będą odbijać światło odbicia stwarzając wrażenie połysku, a absorbować pozostałe rodzaje światła. Kolor materiału zależy od stopnia odbicia światła i światła rozproszonego, który zwykle jest jednakowy.



Normalne

Normalna do powierzchni jest wektorem prostopadłym do danej powierzchni. Wyznaczenie normalnej do powierzchni jest niezbędne w celu ustalenia odcienia koloru, którym wypełniony zostanie wielokąt oświetlony pod określonym kątem. Jak wiadomo źródła światła (w zależności od ich rodzaju) promieniują w określonym kierunku lub we wszystkich kierunkach. Promienie światła padają więc na obiekt pod pewnym kątem. Znając wartość tego kąta oraz kąt odbicia światła można, biorąc pod uwagę także rodzaj materiału i rodzaj światła, wyznaczyć kolor danej powierzchni. Wyznaczenie normalnej ma więc na celu określenie kąta padania światła na daną powierzchnię.

Gdy jednak działania ograniczy się jedynie do wyznaczenia oświetlenia powierzchni na podstawie normalnej, to uzyska się mało realistyczny efekt wypełnienia powierzchni jednolitym kolorem. Dlatego też normalną, a tym samym kolor powierzchni, należy wyznaczyć w odniesieniu do wszystkich wierzchołków wielokąta. Wielokąt będzie natomiast wypełniany przy użyciu modelu cieniowania gładkiego, co pozwoli uzyskać bardziej realistyczny efekt.

Użycie normalnych

Normalną zapisuje się jak każdy inny wektor. Na przykład dla wektora leżącego na płaszczyźnie xz (którego wszystkie wierzchołki maja taką samą współrzędną y) można narysować linie prostopadłą do płaszczyzny trójkąta w wierzchołku o współrzędnych $(x, y, 0)$. Linia ta będzie wyznaczać prostopadłą i normalną do trójkąta.

OpenGL udostępnia funkcję `glNormal3f()`, która pozwala wyznaczyć normalną w kolejnym rysowanym wierzchołku lub normalną do płaszczyzny przez zbiór rysowanych wierzchołków. jej prototyp wygląda następująco:

```
void glNormal3f(GLfloat nx, GLfloat ny, GLfloat nz);
```

Parametry funkcji reprezentuja składowe wektora normalnego do płaszczyzny. Funkcje tę wywołuje się przed utworzeniem wierzchołków:

```
glBegin(GL_TRIANGLES);
   glNormal3f(0.0f, 1.0f, 0.0f);
   glVertex3f(0.0f, 5.0f, 0.0f);
   glvertex3f(4.0f, 5.0f, 0.0f);
   glvertex3f(0.0f, 5.0f, 4.0f);
glEnd();
```

Normalne jednostkowe

Aby przeprowadzić obliczenia związane z wyznaczeniem oświetlenia obiektu, OpenGL musi przekształcić normalne przekazane za pomocą funkcji `glNormal3f()` w *normalne jednostkowe*. Normalne jednostkowe są poprostu wektorami normalnymi o długości równej 1.

Aby OpenGL używał normalnych jednostkowych, należy wywołać funkcję `glEnable()` z parametrem `GL_NORMALIZE`

```
glEnable(GL_NORMALIZE);
```

Zastosowanie tego parametru powoduje, że OpenGL wyznacza wektor jednostkowy. Ułatwia to pracę programisty, który nie musi samodzielnie wyznaczać normalnych jednostkowych. jednak powoduje wolniejsze działanie OpenGL. Dlatego też lepiej jest zawsze samodzielnie wyznaczać normalne jednostkowe.

Korzystanie z oświetlenia

OpenGL umożliwia umieszczenie w tworzonej scenie do ośmiu źródeł światła. Liczba ta powinna być wystarczająca w zdecydowanej większości przypadków. W pozostałych przypadkach można zastosować odpowiednie przekształcenia, tak by obciąć źródła światła które nie są widoczne. W celu dodania oświetlenia do sceny należy wykonać cztery podane niżej polecenia:

- Należy wyznaczyć wektory normalne w każdym wierzchołku każdego obiektu. Pozwolą one określić orientację obiektów względem źródła światła.
- Należy utworzyć, wybrać i umieścić źródła światła
- Należy utworzyć i wybrać model oświetlenia. Model ten definiuje światło otoczenia oraz położenie obserwatora, dla którego wyznaczane jest oświetlenie.
- Należy zdefiniować właściwości materiałów obiektów znajdujących się na scenie.

Korzystanie z oświetlenia

Definiujemy tablice określające właściwości oświetlenia:

```
// swiatlo otoczenia
float ambientLight[] = {0.3f, 0.5f, 0.8f, 1.0f};
// swiatlo rozproszone
float diffuseLight[] = {0.25f, 0.25f, 0.25f, 1.0f};
// pozycja zrodla swiatla
float lightPosition[] = {0.0f, 0.0f, 0.0f, 1.0f};
```

Zmienne te określają właściwości oświetlenia. Tablica `ambientLight` deklaruje światło otoczenia o wartościach składowej czerwonej, zielonej i niebieskiej wynoszących odpowiednio 0.3, 0.5, 0.8. Oznacza to, że światło padające na obiekt ze wszystkich kierunków będzie miało odcień niebieski, ponieważ ta składowa ma największą intensywność. Zmienna `diffuseLight` nadaje wszystkim składowym światła rozproszonego tę sama wartość 0.25. oznacza to, że każda ściana oświetlona bezpośrednio przez to światło będzie jaśniejsza od pozostałych ścian obiekt. Ostatnia z wartości podana dla obu rodzajów światła opisuje współczynnik alfa.

Korzystanie z oświetlenia

Ostatnia ze zmiennych określa pozycję źródła światła. Pierwsze trzy wartości maja takie samo znaczenie jak w przypadku funkcji `glTranslate()`. W tym przykładzie źródło światła zostało umieszczone w początku układu współrzędnych. czwarta wartość informuje OpenGL, czy pierwsze trzy wartości reprezentują współrzędne źródła światła czy wektor. Wartość 1.0 oznacza, że reprezentują one współrzędne źródła światła.

Jeśli zmieniona zostanie ostatnia wartość na 0.0, to OpenGL zinterpretuje pierwsze trzy wartości jako składowe wektora określającego kierunek z którego pada światło. Wektor $(0, 0, 0)$ nie reprezentuje żadnego kierunku, wobec czego należy zmienić jego składowe. Jeśli przyjmie się założenie, że obserwator znajduje się w środku układu współrzędnych i spogląda w kierunku ujemnej części osi z, wtedy można ustalić kierunek, z którego pada światło jako dodatni kierunek osi z. Wektor ten należy zdefiniować następująco:

```
float lightPosition[] = {0.0f, 0.0f, 1.0f, 0.0f};
```

Korzystanie z oświetlenia

Kod inicjalizujący właściwości oświetlenia i materiału może wyglądać następująco:

```
void Initialize()
{
   // kolor czarny
   glClearColor(0.0f, 0.0f, 0.0f, 0.0f);

   glShadeModel(GL_SMOOTH);    // cienowanie gladkie
   glEnable(GL_DEPTH_TEST);    // usuwa ukryte powierzchnie

   glEnable(GL_LIGHTING);      // aktywuje oswietlenie
   // okresla wlasciwosci materialu
   glMaterialfv(GL_LIGHT0, GL_AMBIENT, ambientLight);
   glMaterialfv(GL_LIGHT0, GL_DIFFUSE, diffuseLight);
   glLightfv(GL_LIGHT0, GL_POSITION, lightPosition);

   glEnable(GL_LIGHT0);        // wlacza oswietlenie
}
```

Korzystanie z oświetlenia

Za pomocą wartości `GL_SMOOTH` włącza się model cieniowania Gouraud.

Kolejną z użytych wartością jest `GL_LIGHTING`. przekazana jako parametr funkcji `glEnable()` informuje OpenGL, że będą przeprowadzane obliczenia związane z wyznaczeniem oświetlenia obiektów.

Funkcja `glMaterialfv()` określa właściwości źródeł światła. trzeba w tym miejscu przypomnieć o tym, że OpenGL pozwala umieścić na tworzonej scenie do ośmiu źródeł światła oznaczonych za pomocą wartości `GL_LIGHTX`, gdzie `X` jest liczba z przedziału 0 do 7. Aby więc zmodyfikować właściwość na przykład piątego źródła, należy podać jako parametr funkcji `glLightfv()`wartość `GL_LIGHT4`

Na końcu należy jeszcze włączyć źródło światła przekazując odpowiednia wartość `GL_LIGHTX` funkcji `glEnable()`. W przeciwnym razie scena nie zostanie oświetlona.

Łączenie kolorów

Łączenie kolorów pozwala uzyskać efekt przezroczystości. Dzięki temu grafika OpenGL może symulować obiekty świata rzeczywistego, przez które można widzieć inne obiekty na przykład szkło lub wodę.

Przy łączeniu kolorów znajduje zastosowanie wartość współczynnika alfa, który pojawił się wiele razy przy omawianiu funkcji OpenGL. Łączenie kolorów polega na utworzeniu nowego koloru na podstawie kolorów obiektu przesłanianego i przesłaniającego. Wykonywane jest zwykle na składowych modelu RGB przy wykorzystaniu współczynnika alfa reprezentującego przesłanianie. Im mniejsza jego wartość, tym większe będzie wrażenie przezroczystości obiektu przesłaniającego.

Aby korzystać z łączenia kolorów w OpenGL, należy je uaktywnić przekazując funkcji `glEnable()` parametr `GL_BLEND`. następnie wywołuje się funkcję `glBlendFunc()` z parametrem `GL_ONE` dla źródła i `GL_ZERO` dla celu.

Łączenie kolorów

Dostępne funkcje łączenia źródła:

Funkcja	Opis
GL_ZERO	kolorem źródła jest (0,0,0,0)
GL_ONE	wykorzystuje bieżący kolor źródła
GL_DST_COLOR	mnoży kolor źródła przez kolor celu
GL_ONE_MINUS_DST_COLOR	mnoży kolor źródła przez dopełnienie koloru celu, czyli przez $[(1,1,1,1) - kolorcelu]$
GL_SRC_ALPHA	mnoży kolor źródła przez wartość alfa źródła
GL_ONE_MINUS_SRC_ALPHA	mnoży kolor źródła przez dopełnienie wartości alfa źródła, czyli przez (1-wartość alfa celu)
GL_DST_ALPHA	mnoży kolor źródłą przez wartość alfa celu
GL_ONE_MINUS_DST_ALPHA	mnoży kolor źródła przez dopełnienie wartości alfa celu, czyli przez (1-wartość alfa celu)

Przezroczystość

Efekt przezroczystości uzyskuje się stosując kombinację funkcji źródła `GL_SRC_ALPHA` z funkcją celu `GL_ONE_MINUS_SRC_ALPHA`. Poniższy fragment kodu konfiguruje sposób łączenia kolorów tak, by można było uzyskać efekt przezroczystości:

```
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```

Przy tak określonych funkcjach łączenia kolor źródła nakładany jest w stopniu określonym przez współczynnik alfa na kolor pikseli celu.

Odwzorowania tekstur

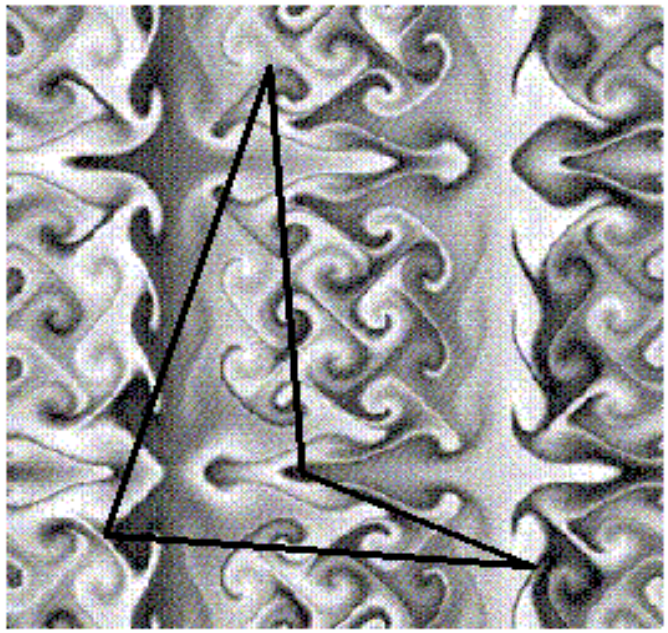
Odwzorowanie tekstur pozwala pokrywać wielokąty za pomocą realistycznych obrazów. Na przykład kulę można pokryć mapą powierzchni Ziemi uzyskując w ten sposób trójwymiarową reprezentację naszej planety. tekstury są powszechnie wykorzystywane w grafice trójwymiarowej. Zwłaszcza w grach umożliwiając uzyskanie tak pożądanego realizmu tworzonych scen.

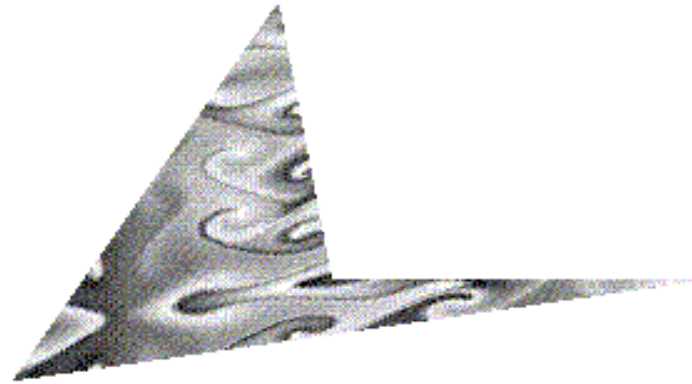
Mapy tekstur są dwuwymiarowymi tablicami zawierającymi elementy nazywane *tekselami*. Mimo, że reprezentują zawsze obszar prostokąta, to mogą być odwzorowane na dowolnych trójwymiarowych obiektach, takich jak kule, walce i tym podobne.

Zwykle stosowane są tekstury dwuwymiarowe, ale używa się także tekstur jedno i trójwymiarowych. tekstura dwuwymniarowa posiada szerokość i wysokość.

Odwzorowania tekstur

Proces mapowania tekstury:





Odwzorowania tekstur

Po wykonaniu odwzorowania tekstury na wielokąt będzie ona przekształcana razem z wielokątem. W przypadku wspomnianej już reprezentacji Ziemi tekstura będzie obracać się razem z kulą, na której została umieszczona stwarzając w ten sposób wrażenie obrotu ziemi. Także w przypadku, gdy do jej animacji dodany zostanie kolejny obrót i przesunięcie na orbicie wokół innej kuli reprezentującej Słońce, tekstura pozostanie na powierzchni kuli. Odwzorowanie tekstur można wyobrazić sobie jako zmianę sposobu wyglądu powierzchni obiektu lub wielokąta. Niezależnie od wykonywanych przekształceń pokrywać będzie ona zawsze jego powierzchnię.

Tekstury praktyka

Po załadowaniu obrazu tekstury z pliku graficznego stosuje się funkcję `glGenTextures()`, aby nadać nazwę obiektowi tekstury. Funkcja `glBindTexture()` wiąże obiekt tekstury z bieżącą teksturą, która będzie wykorzystywana podczas rysowania wielokątów. Aby skorzystać w programie z wielu różnych tekstur, trzeba przy zmianie bieżącej tekstury za każdym razem wywoływać funkcję `glBindTexture()`. Funkcja `glTexParameteri()` określa sposób filtrowania tekstury.

Wzór tekstury reprezentowany za pomocą wczytanego wcześniej obrazu określa się za pomocą funkcji `glTexImage2d()`. Za pomocą jej parametrów można określić także rozmiary, typ, położenie i format danych.

Mapy tekstur

Po załadowaniu z pliku obrazu tekstury, należy zadeklarować go w OpenGL jako mapę tekstury. W zależności od liczby wymiarów tekstury używa się w tym celu innej funkcji. Dla mapy tekstury dwuwymiarowej będzie to funkcja `glTexImage2d()`. Dla map tekstur jednowymiarowych użyć trzeba funkcji `glImage1D()`, a dla map tekstur trójwymiarowych funkcji `gltexImage3D()`

Tekstury dwuwymiarowe

Aby zdefiniować teksturę dwuwymiarową użyć można funkcji `glTexImage2d()` zdefiniowanej następująco:

```
void gltexture2d(GLenum target, GLint level, GLint internalFormat
GLsizei width, GLsizei height, GLint border, GLenum format,
GLenum type, const GLvoid* texels);
```

Tekstury dwuwymiarowe

Parametry :

- `target` może przyjmować wartość `GL_TEXTURE_2D` lub `GL_PROXY_TEXTURE_2D`. W naszych zastosowaniach będzie on posiadać zawsze wartość `GL_TEXTURE_2D`
- `level` określa rozdzielczość mapy tekstury. Nadanie wartości 0 oznacza tylko jedną rozdzielczość
- `internalFrame` opisuje format tekseli, które przekazywane będą funkcji. Może przyjmować on wartości z przedziału od 1 do 4 bądź jednej z 32 stałych zdefiniowanych przez OpenGL. Najbardziej przydatne to: `GL_LUMINANCE`, `GL_LUMINANCE_ALPHA`, `GL_RGB` i `GL_RGBA`
- `width` i `height` definiują szerokość i wysokość mapy tekstury. Ich wartości być potęgą liczby 2.
- `border` określa, czy dookoła tekstury znajduje się ramka. Przyjmuje wtedy wartość 1, natomiast wartość 0 oznacza brak ramki

Tekstury dwuwymiarowe

parametry:

- `format` definiuje format danych obrazu tekstury. Może on przyjmować jedną z następujących wartości: `GL_COLOR_INDEX`, `GL_RGB`, `GL_RGBA`, `GL_RED`, `GL_GREEN`, `GL_BLUE`, `GL_ALPHA`, `GL_LUMINANCE` lub `GL_LUMINANCE_ALPHA`
- `type` definiuje typ danych obrazu tekstury. Może on przyjmować jedną z następujących wartości: `GL_BYTE`, `GL_UNSIGNED`, `GL_UNSIGNED_BYTE`, `GL_SHORT`, `GL_UNSIGNED_SHORT`, `GL_INT`, `GL_UNSIGNED_INT`, `GL_FLOAT` lub `GL_BITMAP`
- `texels` jest wskaźnikiem właściwych danych obrazu tekstury, które zostały wygenerowane za pomocą pewnego algorytmu lub załadowane z pliku.

Jeżeli załadowany został obraz formatu RGBA o szerokości `textureWidth` i wysokości `textureHeight` do bufora wskazywanego przez zmienna `textureData`. Funkcję `gltexImage2d()` wywoływać się będzie wtedy w następujący sposób:

```
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, textureWidth,
textureHeight, 0, GL_RGBA, GL_UNSIGNED_BYTE, textureData
```

Obiekty tekstur

Obiekty tekstur służą do przechowywania gotowych do użycia tekstur. Umożliwiają załadowanie do pamięci wielu obrazów tekstur, a następnie odwoływanie się do nich podczas rysowania sceny. Rozwiązanie takie zwiększa efektywność tworzenia grafiki, ponieważ tekstury nie muszą być ładowane za każdym razem, gdy konieczne jest ich użycie.

Tworzenie nazwy tekstury

Zanim użyty zostanie obiekt tekstury, trzeba najpierw utworzyć jego nazwę. Nazwy tekstury są liczbami dodatnimi całkowitymi różnymi od zera. Użycie funkcji `glGentextures()` do tworzenia nazw pozwala zachować pewność, że istniejąca już nazwa nie zostanie zduplikowana:

```
void glGenTextures(GLsizei n, GLuint *textureNames);
```

parametr n określa liczbę nazw tekstur, które powinny zostać utworzone i umieszczone w tablicy `textureNames`. Na przykład poniższy fragment kodu utworzy trzy nowe nazwy tekstur:

```
unsigned int textureNames[3];
...
glGenTextures(3, texturenames);
```

Obiekty tekstur

Po utworzeniu nazwy tekstury należy związać ją z danymi opisującymi teksturę. Służy do tego funkcja `glBindTexture()`:

```
void glBindTexture(GLenum target, GLuint textureName);
```

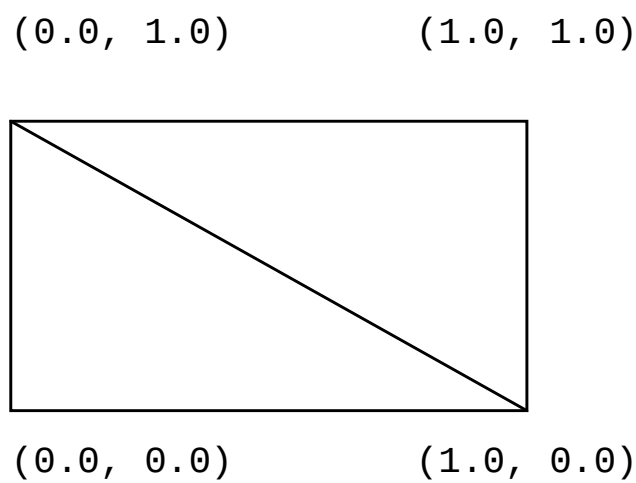
Pierwsze wywołanie tej funkcji powoduje utworzenie nowego obiektu tekstury o domyślnych właściwościach. Właściwości te można zmienić odpowiednie funkcje OpenGL związane z teksturami. Parametr `target` funkcji `glBindTexture()` może przyjmować wartości `GL_TEXTURE_1D`, `GL_TEXTURE_2D` lub `GL_TEXTURE_3D`.

Po związaniu obiektu tekstury z danymi można ponownie użyć funkcji `glBindTexture()` do zmiany, właściwości obiektu tekstury. Można założyć, że w programie utworzono wiele obiektów tekstur i powiązano je z danymi tekstur. Przy tworzeniu grafiki informuje się maszynę OpenGL za pomocą funkcji `glBindTexture()`, której tekstury należy użyć.

Współrzędne tekstury

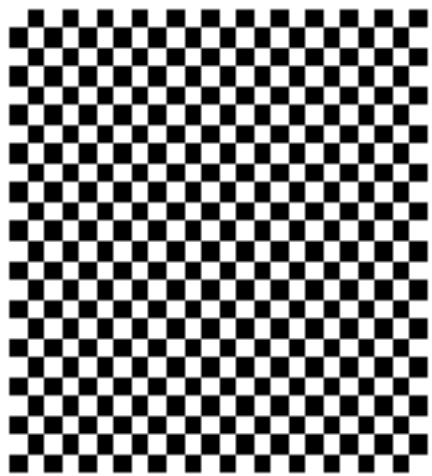
Tworząc scenę należy określić współrzędne tekstury dla wszystkich wierzchołków. Współrzędne tekstury pozwalają ustalić położenie tekseli na rysowanym obiekcie. Współrzędne $(0, 0)$ określają lewy dolny narożnik tekstury, a współrzędne $(1, 1)$ prawy, górny narożnik.

Podczas rysowania wielokąta trzeba określić współrzędne tekstury dla każdego z jego wierzchołków. W przypadku tekstur dwuwymiarowych mają one postać (s, t), gdzie s i t przyjmują wartości z przedziału od 0 do 1



Powtarzanie i rozciąganie tekstur

Jeśli współrzędne tekstury będą posiadać wartości spoza przedziału od 0 do 1, to OpenGL może powtarzać lub rozciągać teksturę. Powtarzanie tekstury określane jest czasem także mianem kafelkowania. Jeśli współrzędne tekstury będą posiadać wartość $(2.0, 2.0)$, to tekstura zostanie powtórzona czterokrotnie. Jeżeli współrzędne tekstury będą posiadać wartość $(1.0, 2.0)$, to tekstura zostanie powtórzona dwukrotnie tylko w kierunku t.



Powtarzanie i rozciąganie tekstur

Powtarzanie i rozciąganie tekstur wykonuje się w OpenGL za pomocą funkcji `glTexParameteri()`

Fragment kodu informujący OpenGL, aby powtórzył teksturę w obu kierunkach:

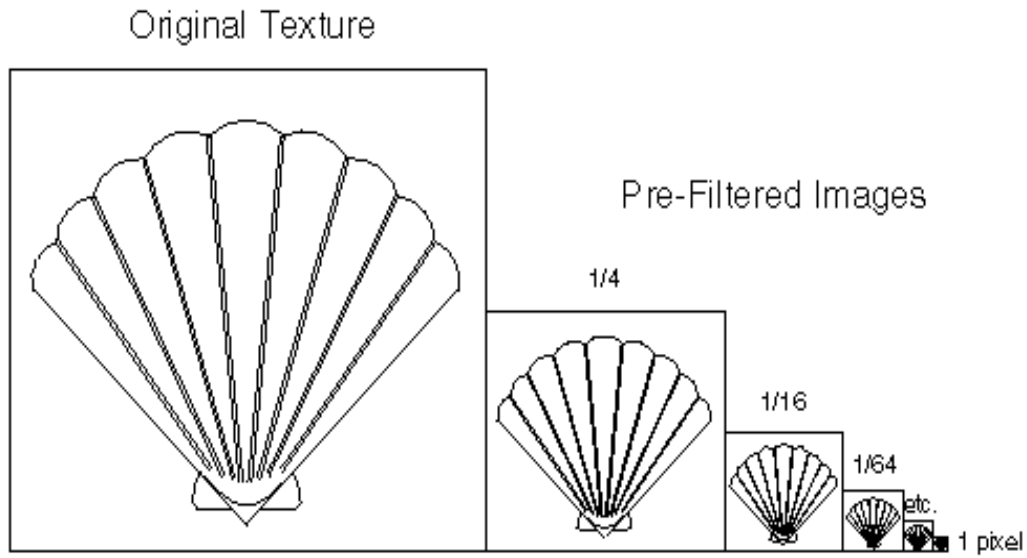
```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
```

OpenGL może równocześnie powtarzać teksturę w jednym kierunku i rozciągać ją w drugim kierunku:

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
```

Mipmapy i poziomy szczegółowości

Podczas rysowania obiektów pokrytych teksturami można zauważyć pewne obszary, w których pojawiają się nieoczekiwane wzory, zwłaszcza przy oddalaniu się lub zbliżaniu obiektów. Są one rezultatem filtrowania zastosowanego przez OpenGL w celu dopasowania tekstury do rozmiarów obiektu. Niepożądany efekt można wyeliminować za pomocą `mipmap`, które pozwalają kontrolować poziom szczegółowości tekstury.



Automatyczne mipmapy

Biblioteka GLU udostępnia zestaw funkcji pozwalający zautomatyzować tworzenie mipmap, W przypadku tekstur dwuwymiarowych wiele wywołań funkcji `glTexImage2d()` można zastąpić pojedynczym wywołaniem funkcji `gluBuild2dMipmaps()` o następującym prototypie:

```
int gluBuild2DMipmaps(GLenum target, GLint internalFormat,
GLint width, GLint height, GLenum format, GLenum type,
void *texels);
```

Funkcja ta tworzy sekwencje mipmap, a sama wywołuje funkcję `glTexImage2D()`. Trzeba przekazać jej jedynie mipmapę o największej rozdzielczości, a pozostałe zostaną wygenerowane automatycznie.

Biblioteka GLUT

GLUT - OpenGL Utility Toolkit, jest biblioteką umożliwiającą budwanie aplikacji korzystających z OpenGL w sposób całkowicie niezależny od platformy systemowej. Dzięki bibliotece GLUT możemy pisać aplikacje bez konieczności uczenia się mechanizmów tworzenia okien w środowisku X Windows lub MS Windows.

Inicjalizacja

W pierwszym kroku należy zainicjalizować bibliotekę, służy do tego funkcja `glutinit()` wyglądająca następująco:

```
void glutinit(int *argc, char **argv);
```

parametry:

- argc - wskaźnik do niezmodyfikowanego parametru argc z wywołania funkcji `main`
- argv wskaźnik do niezmodyfikowanego parametru argv funkcji **main**

Biblioteka GLUT

Następnie należy ustalić rozmiar okna wykorzystując funkcję `glutInitWindowSize()`:

```
void glutInitWindowSize(int width, int height);
```

parametry:

- width - szerokość okna
- height - wysokość okna

następnie definiujemy tryb wyświetlania z wykorzystaniem funkcji `glutInitDisplayMode()`

```
void glutInitDisplayMode(unsigned int mode);
```

parametry:

- mode - tryb wyświetlania

Biblioteka GLUT

Przykład kodu inicjalizującego okno z wykorzystaniem GLUT:

```
#include <gl/glut.h>

void main(int argc, char *aargv[]) {
   glutInit(&argc, &argv);
   glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGBA);
   glutInitWindowPossition(100,100);
   glutInitWindowSize(320,320);
   glutCreateWindow(okno GLUT");"

   glutDisplayFunc(display);

   glutmainLoop();
```

Dobry tutorial dotyczący biblioteki GLUT znajduje sie pod adresem:
http://www.lighthouse3d.com/opengl/glut/

Biblioteka GLUT

Biblioteka GLUT opiera się na mechaniźmie funkcji typu call back. Za przykład niech posłuży funkcja `glutDisplayFunc()` zadeklarowana jako:

```
void glutDisplayFunc(void(*func)(void));
```