Janusz Ganczarski

OpenGL

Podstawy programowania grafiki 3D

Helion

Opieka redakcyjna: Ewelina Burska
Projekt okładki: Studio Gravite/Olsztyn
Obarek, Pokoński, Pazdrijowski, Zaprucki
Materiały graficzne na okładce zostały wykorzystane za zgodą Shutterstock.

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/opglwp
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

ISBN: 978-83-283-0193-1

Printed in Poland.

Spis treści

Rozdział 3.
Pierwszy program

Po wstępnych informacjach dotyczących biblioteki OpenGL przyszedł czas na napisanie pierwszego programu, który z niej korzysta. Nie jest to zadanie banalne, ponieważ wymaga zapoznania się z kilkoma fundamentalnymi elementami OpenGL, bez których praktycznie nie ma programu w OpenGL. Po lekturze tego rozdziału Czytelnik powinien mieć już podstawową wiedzę, jak wygląda program korzystający z tej biblioteki.

Obiekty bufora

Biblioteka OpenGL grupuje większość funkcjonalności w obiektach. Obiekty w bibliotece OpenGL są oznaczane unikatowymi identyfikatorami reprezentowanymi przez liczby całkowite bez znaku. Każdy obiekt w bibliotece OpenGL posiada swój własny zestaw zmiennych stanu, który jest inicjowany podczas tworzenia obiektu.

Obiekty bufora (ang. *buffer object*) służą do przechowywania różnego rodzaju danych w pamięci (a dokładniej w przestrzeni adresowej) serwera OpenGL. Tworząc nowy obiekt, w pierwszej kolejności musimy wygenerować jego identyfikator, a następnie dokonać tzw. **powiązania** (ang. *bind*) wybranego typu obiektu z jego identyfikatorem. Czynność ta tworzy wybrany rodzaj obiektu. OpenGL przechowuje dane w obiekcie bufora jako dane binarne o nieokreślonej strukturze. Dopiero podczas użycia obiektu bufora odpowiednie polecenia OpenGL określają, jak należy interpretować dane w nich zawarte.

Tworzenie i usuwanie obiektów bufora

Do generowania identyfikatorów obiektów bufora służy funkcja:

```
void glGenBuffers( GLsizei n, GLuint *buffers );
```

która umożliwia jednoczesne utworzenie `n` identyfikatorów umieszczanych w tablicy wskazywanej w parametrze `buffers`. W analogiczny sposób można grupowo usuwać obiekty bufora (nie tylko same identyfikatory), do czego służy funkcja:

```
void glDeleteBuffers( GLsizei n, const GLuint *buffers );
```

Utworzenie samego obiektu bufora wymaga wywołania funkcji:

```
void glBindBuffer( GLenum target, GLuint buffer );
```

która w pierwszym parametrze określa rodzaj obiektu bufora, a w drugim parametrze zawiera jego wcześniej utworzony identyfikator. W programach będziemy wykorzystywać **obiekty bufora wierzchołków VBO** (ang. *vertex buffer object*), których zadaniem jest przechowywanie atrybutów wierzchołków. Aby utworzyć obiekty typu VBO, parametr `target` musi przyjąć wartość równą `GL_ARRAY_BUFFER`. Następne wywołania funkcji `glBindBuffer` z danym identyfikatorem pozwalają na przełączanie się pomiędzy różnymi obiektami bufora, a ostatnio tak wybrany obiekt bufora jest nazywany bieżącym obiektem bufora.

Obiekty bufora można także tworzyć jednocześnie z generowaniem ich identyfikatorów, do czego służy funkcja:

```
void glCreateBuffers( GLsizei n, GLuint *buffers );
```

Tak utworzone obiekty bufora nie mają określonego rodzaju, który jest ustalany przy pierwszym wywołaniu funkcji `glBindBuffer`.

Ładowanie danych do obiektu bufora

Nowo utworzony obiekt bufora nie zawiera żadnych danych. Ładowanie danych do obiektów bufora można realizować różnie, ale najczęściej wykonuje się jedną z poniższych operacji:

- kopiowanie całości lub części danych z przestrzeni adresowej klienta, np. z tablicy lub innej struktury danych;
- kopiowanie danych z innego obiektu bufora;
- zmapowanie danych bufora do przestrzeni adresowej klienta OpenGL poprzez pobranie wskaźnika.

Dane obiektu bufora mogą również być ładowane bezpośrednio z potoku renderingu na etapie transformacji sprzężonych zwrotnie, a także zapisywane z poziomu shadera.

Najprostszym i w praktyce najczęściej wykorzystywanym sposobem załadowania danych do bieżącego obiektu bufora jest ich skopiowanie bezpośrednio z pamięci aplikacji OpenGL. Służy do tego funkcja:

```
void glBufferData( GLenum target, GLsizeiptr size, const GLvoid *data,
↪GLenum usage );
```

Parametr `target` określa rodzaj obiektu bufora i powinien przyjąć taką samą wartość jak w funkcji `glBindBuffer`. Drugi parametr — `size` — ustala rozmiar danych obiektu bufora w bajtach, a trzeci parametr — `data` — jest wskaźnikiem na źródło danych bufora znajdujące się w pamięci klienta. Jeśli parametr `data` nie jest pusty, to funkcja `glBufferData` kopiuje wskazane dane do obiektu bufora. W przeciwnym wypadku zawartość danych obiektu bufora jest niezdefiniowana. W każdym przypadku usuwane są wszelkie istniejące dane znajdujące się w obiekcie bufora.

Przy określaniu rozmiarów pamięci, odstępów pomiędzy elementami pamięci i podobnych wielkościach operujących na pamięci specyfikacja OpenGL używa pojęcia **podstawowych jednostek maszyny BMU** (ang. *basic machine unit*), które w pewnym uproszczeniu będziemy w niniejszym tekście utożsamiać z pojęciem 8-bitowego bajta.

Ostatni parametr `usage` jest wskazówką wydajności, informującą OpenGL o spodziewanym sposobie wykorzystania danych obiektu bufora (faktyczny sposób wykorzystania bufora może być inny). Dostępne są następujące wartości tego parametru:

- `GL_STREAM_DRAW` — zawartość danych obiektu bufora zostanie określona raz przez aplikację i będzie używana co najwyżej kilka razy jako źródło dla poleceń OpenGL do rysowania i specyfikacji obrazu.
- `GL_STREAM_READ` — zawartość danych obiektu bufora zostanie określona raz przez pobranie danych z OpenGL i będzie pobierana co najwyżej kilka razy przez aplikację.
- `GL_STREAM_COPY` — zawartość danych obiektu bufora zostanie określona raz przez pobranie danych z OpenGL i będzie używana co najwyżej kilka razy jako źródło dla poleceń OpenGL do rysowania i specyfikacji obrazu.
- `GL_STATIC_DRAW` — zawartość danych obiektu bufora zostanie określona raz przez aplikację i będzie używana wiele razy jako źródło dla poleceń OpenGL do rysowania i specyfikacji obrazu.
- `GL_STATIC_READ` — zawartość danych obiektu bufora zostanie określona raz przez pobranie danych z OpenGL i będzie pobierana wiele razy przez aplikację.

- `GL_STATIC_COPY` — zawartość danych obiektu bufora zostanie określona raz przez pobranie danych z OpenGL i będzie używana wiele razy jako źródło dla poleceń OpenGL do rysowania i specyfikacji obrazu.
- `GL_DYNAMIC_DRAW` — zawartość danych obiektu bufora będzie określana wielokrotnie przez aplikację i używana wiele razy jako źródło dla poleceń OpenGL do rysowania i specyfikacji obrazu.
- `GL_DYNAMIC_READ` — zawartość danych obiektu bufora będzie określana wielokrotnie poprzez pobranie danych z OpenGL i pobierana wiele razy przez aplikację.
- `GL_DYNAMIC_COPY` — zawartość danych obiektu bufora będzie określana wielokrotnie poprzez pobranie danych z OpenGL i używana wiele razy jako źródło dla poleceń OpenGL do rysowania i specyfikacji obrazu.

Wybrany obszar danych bieżącego obiektu bufora można także modyfikować przy użyciu funkcji:

```
void glBufferSubData( GLenum target, GLintptr offset,
↪GLsizeiptr size, const void *data );
```

która określa początek (`offset`) i rozmiar (`size`) modyfikowanych danych obiektu bufora typu określonego w parametrze `target`. Modyfikowane dane przechowywane są w tablicy w pamięci klienta wskazywanej w ostatnim parametrze `data`.

Transfer danych pomiędzy programem korzystającym z OpenGL a pamięcią GPU jest procesem stosunkowo wolnym. Częsty transfer dużych ilości danych może być jednym z tzw. wąskich gardeł aplikacji graficznych, zatem należy takie operacje w programie dobrze zoptymalizować.

Wierzchołki i ich atrybuty

Jak wspomnieliśmy w poprzednim rozdziale, OpenGL operuje na prymitywach geometrycznych, które są opisane przez wierzchołki i ich atrybuty. Każdy wierzchołek prymitywu jest określony przez jeden lub większą liczbę atrybutów, z kolei każdy atrybut jest określony przez wartość skalarną bądź wektor dwu-, trzy- lub czteroelementowy. Skalary oraz składowe wektora mogą być: liczbami zmiennoprzecinkowymi (pojedynczej lub podwójnej precyzji), liczbami stałoprzecinkowymi (typ `GLfixed`) lub liczbami całkowitymi o różnej liczbie bitów. Możliwe jest także definiowanie atrybutów zawierających macierze, ale w tym wypadku OpenGL traktuje taki atrybut jak zbiór atrybutów wektorowych liczący tyle atrybutów, ile kolumn ma macierz. Typowymi atrybutami wierzchołków są: współrzędne położenia, składowe kolorów, wektory normalne i współrzędne tekstury.

Atrybuty wierzchołków przechowywane są w obiektach bufora wierzchołków VBO, skąd pobierane są i przetwarzane przez shadery wierzchołków w celu wykorzystywania w późniejszych etapach potoku renderingu.

Tablice wierzchołków

Tablice wierzchołków VA (ang. *vertex arrays*) opisują, w jaki sposób są zorganizowane dane atrybutów wierzchołków zawarte w obiektach VBO. Każda tablica wierzchołków zawiera m.in. unikatowy numer indeksu atrybutu wierzchołka (początek numeracji indeksów zaczyna się od 0), przy czym maksymalnie dostępnych jest `GL_MAX_VERTEX_ATTRIBS` indeksów atrybutów wierzchołków. Stała ta jest zależna od implementacji biblioteki OpenGL, ale nie może być mniejsza niż 16.

Definiowanie tablic wierzchołków

Typowo do zdefiniowania tablicy wierzchołków z atrybutami zmiennoprzecinkowymi pojedynczej precyzji używana jest funkcja:

```
void glVertexAttribPointer( GLuint index, GLint size, GLenum type,
↪GLboolean normalized, GLsizei stride, const GLvoid *pointer );
```

Parametr `index` zawiera numer indeksu definiowanego atrybutu wierzchołka. Numery indeksów wierzchołków generowane są automatycznie przez OpenGL lub mogą być wskazane wprost w shaderze wierzchołków albo w samym programie. Drugi parametr `size` określa, ile składowych ma pojedynczy atrybut wierzchołka przechowywany w obiekcie VBO. Możliwe są wartości 1, 2, 3 i 4, które oznaczają odpowiednio atrybut w postaci skalara oraz dwu-, trzy- lub czteroelementowego wektora. Poszczególne atrybuty wierzchołków pobierane są kolejno z pamięci obiektu VBO, w przypadku atrybutów wektorowych składowe wektorów są także pobierane kolejno z pamięci obiektu.

Następny parametr `type` określa typ danych atrybutów wierzchołka przechowywanych w obiekcie VBO. Najważniejsze wartości parametru `type` i ich powiązanie z typami danych dostępnymi w bibliotece OpenGL zawiera tabela 3.1. Poza typami prezentowanymi w tabeli atrybuty wierzchołków mogą także występować w tzw. formatach upakowanych, których jednak nie będziemy używać w przykładowych programach.

Kolejny parametr `normalized` wskazuje, czy dane zawarte w obiekcie VBO będące liczbami całkowitymi (`GL_BYTE`, `GL_UNSIGNED_BYTE`, `GL_SHORT`, `GL_UNSIGNED_SHORT`, `GL_INT` i `GL_UNSIGNED_INT`) mają być konwertowane do liczb zmiennoprzecinkowych przez normalizowanie do przedziału $\langle 0;1 \rangle$ dla typów bez znaku

Tabela 3.1. *Wybrane typy danych atrybutów w tablicach wierzchołków i odpowiadające im typy danych OpenGL*

Parametr type	Odpowiadający typ OpenGL
GL_BYTE	GLbyte
GL_UNSIGNED_BYTE	GLubyte
GL_SHORT	GLshort
GL_UNSIGNED_SHORT	GLushort
GL_INT	GLint
GL_UNSIGNED_INT	GLuint
GL_FIXED	GLfixed
GL_FLOAT	GLfloat
GL_HALF_FLOAT	GLhalf
GL_DOUBLE	GLdouble

lub przedziału $\langle -1;1 \rangle$ dla typów ze znakiem (wartość parametru równa GL_TRUE). Możliwe jest także bezpośrednie przeliczenie wartości takiego atrybutu do liczby zmiennoprzecinkowej (wartość parametru równa GL_FALSE). Normalizacja jest typową operacją wykonywaną w grafice komputerowej na liczbach całkowitych przechowujących np. składowe RGB(A) kolorów w plikach graficznych. Na przykład dla składowej koloru opisanej 8-bitową liczbą całkowitą bez znaku 0 przyjmuje po normalizacji wartość 0,0, a 255 przyjmuje po normalizacji wartość 1,0.

W pojedynczym obiekcie VBO można przechowywać dane wielu atrybutów wierzchołków. Dane te możemy np. całkowicie od siebie oddzielić, np. umieszczając kolejno: współrzędne wszystkich wierzchołków, współrzędne wektorów normalnych i na końcu składowe kolorów. Można także umieścić w obiekcie VBO dane atrybutów z przeplotem: współrzędne położenia pierwszego wierzchołka, współrzędne wektora normalnego pierwszego wierzchołka, składowe koloru pierwszego wierzchołka, współrzędne położenia drugiego wierzchołka itd. Sposób ułożenia danych atrybutów w obiekcie VBO określa kolejny parametr stride, który definiuje określone w bajtach odstępy pomiędzy danymi poszczególnych atrybutów wierzchołków. Jeżeli stride jest równy zero, to dane definiowanej tablicy atrybutu są kolejno zapisane w obiekcie VBO, w przeciwnym wypadku dane kolejnych atrybutów są odpowiednio od siebie oddzielone.

Ostatni parametr pointer określa w bajtach położenie w pamięci obiektu bufora VBO (wskaźnik) pierwszego atrybutu definiowanej tablicy wierzchołków. Wskaźnik ten odnosi się do danych bieżącego obiektu VBO, czyli obiektu wybranego przy ostatnim wywołaniu funkcji glBindBuffer. Odpowiednie wykorzystanie tego parametru umożliwia wspomniane wyżej przechowywanie danych różnych atrybutów wierzchołków w odrębnych częściach jednego obiektu VBO.

Jeżeli posiadamy atrybuty zawierające liczby całkowite, to typowo przy definiowaniu tablicy wierzchołków używamy funkcji:

```
void glVertexAttribIPointer( GLuint index, GLint size, GLenum type,
➥GLsizei stride, const GLvoid *pointer );
```

która w stosunku do funkcji `glVertexAttribPointer` ma listę dopuszczalnych wartości parametru `type` ograniczoną do liczb całkowitych: `GL_BYTE`, `GL_UNSIG➥NED_BYTE`, `GL_SHORT`, `GL_UNSIGNED_SHORT`, `GL_INT` i `GL_UNSIGNED_INT`. W przypadku użycia tej funkcji dane atrybutów wierzchołków zawsze przekazywane są bezpośrednio w wartościach całkowitych.

Biblioteka OpenGL zawiera jeszcze funkcję:

```
void glVertexAttribLPointer( GLuint index, GLint size, GLenum type,
➥GLsizei stride, const GLvoid *pointer );
```

która służy wyłącznie do definiowania tablic wierzchołków atrybutów zawierających liczby zmiennoprzecinkowe podwójnej precyzji (`GLdouble`). Jedyna dopuszczalna wartość parametru `type` powyższej funkcji to `GL_DOUBLE`. Zauważmy, że analogiczny typ danych obsługuje także funkcja `glVertexAttribPointer`, ale przy jej użyciu dane zmiennoprzecinkowe podwójnej precyzji będą konwertowane do liczb zmiennoprzecinkowych pojedynczej precyzji (`GLfloat`).

Włączanie i wyłączanie tablicy wierzchołków

Pojedyncza tablica atrybutów wierzchołków jest włączana lub wyłączana przez wywołanie jednego z poniższych poleceń:

```
void glEnableVertexAttribArray( GLuint index );
void glDisableVertexAttribArray( GLuint index );
```

których parametr `index` określa numer indeksu atrybutu w tablicy wierzchołków.

Domyślne bieżące atrybuty wierzchołków

Gdy tablica wierzchołków zawierająca wartości atrybutów wierzchołka nie jest aktywna, OpenGL stosuje domyślne bieżące wartości ogólnych atrybutów wierzchołków. Początkowe wartości domyślne dla wszystkich atrybutów ogólnych wierzchołka wynoszą odpowiednio: 0 dla atrybutów skalarnych, (0,0) dla wektorów dwuelementowych, (0,0,0) dla wektorów trójelementowych i (0,0,0,1) dla wektorów czteroelementowych.

OpenGL umożliwia zmianę wartości domyślnych atrybutów wierzchołków. Służy do tego bardzo obszerna grupa funkcji `glVertexAttrib`, które wskazują wartości

domyślnych atrybutów dla wybranego indeksu atrybutu. Przykładowo do określenia wartości domyślnych dla atrybutu będącego wektorem czteroelementowym z liczbami zmiennoprzecinkowymi pojedynczej precyzji najlepiej użyć jednej z następujących funkcji:

```
void glVertexAttrib4f( GLuint index, GLfloat x, GLfloat y, GLfloat z,
↪GLfloat w );
void glVertexAttrib4fv( GLuint index, const GLfloat *v );
```

OpenGL zapewnia odpowiednią wersję funkcji `glVertexAttrib` dla każdej możliwej kombinacji rodzaju atrybutu wierzchołka i typu danych przechowujących wartości atrybutów.

Obiekty tablic wierzchołków

Tablice wierzchołków grupowane są w zbiorczych **obiektach tablic wierzchołków VAO** (ang. *Vertex Array Objects*). Obiekty tablic wierzchołków przechowują informacje o danych tablic wierzchołków i wszystkich związanych z nimi zmiennych stanu używanych w procesie renderingu.

Obsługa obiektów VAO jest analogiczna do obsługi innych obiektów biblioteki OpenGL. Każdy obiekt VAO zawiera unikatowy identyfikator, który należy wygenerować, a następnie utworzyć sam obiekt. Niepotrzebny obiekt VAO należy usunąć. Całość realizują trzy funkcje:

```
void glGenVertexArrays( GLsizei n, GLuint *arrays );
void glDeleteVertexArrays( GLsizei n, const GLuint *arrays );
void glBindVertexArray( GLuint array );
```

Po wywołaniu funkcji `glBindVertexArray` określony w funkcji obiekt VAO jest bieżącym obiektem tablic wierzchołków i wszystkie operacje związane z tablicami wierzchołków (np. `glVertexAttribPointer` i `glEnableVertexAttribArray`) oraz polecenia renderingu, które korzystają z tablic wierzchołków (np. `glDraw↪Arrays` i `glDrawElements`), odnoszą się do wybranego obiektu VAO. Z poleceniami renderingu zapoznamy się bliżej w dalszej części niniejszego rozdziału.

Prymitywy geometryczne

Biblioteka OpenGL obsługuje siedem podstawowych typów prymitywów geometrycznych: punkty, łamane, łamane zamknięte, odcinki, paski trójkątów, wachlarze trójkątów i trójkąty. Każdy rodzaj prymitywu jest identyfikowany przez stałą używaną w poleceniach renderingu. Są to odpowiednio: `GL_POINTS`, `GL_LINE_↪STRIP`, `GL_LINE_LOOP`, `GL_LINES`, `GL_TRIANGLE_STRIP`, `GL_TRIANGLE_FAN` i `GL_TRIANGLES`.

Skorowidz

A

B

C

D

E

F

G

H

I

J

K

N

O

P

R

S

Ś

T

U

V

W

Z

Ź

Darmowa biblioteka OpenGL jest świetnym narzędziem do tworzenia grafiki trójwymiarowej na różnych platformach sprzętowych i w różnych systemach operacyjnych. Jej legendarne możliwości, doskonalone od 1992 roku do dziś, znakomicie sprawdzają się na przykład w grach komputerowych. Z kolei otwarty interfejs programistyczny pozwolił na zbudowanie wokół niej mnóstwa narzędzi uzupełniających i rozszerzeń, które przydadzą się każdemu użytkownikowi pragnącemu wykreować swój własny świat w formacie 3D.

Już niedługo zobaczysz na ekranie to, co na razie pozostaje jedynie w świecie Twojej wyobraźni! Ta książka omawia budowę biblioteki OpenGL, zasady pisania programów i wykorzystania obiektów. Nauczysz się programować grafikę — przekształcać scenę, budować modele oświetlenia i obsługiwać tekstury. Poznasz różne rodzaje buforów i zobaczysz, na czym polega antyaliasing. Odkryjesz też, jak przetwarzać geometrię obiektów i jak tworzyć realistyczne cienie. W dodatkach do książki znajdziesz informacje, które pomogą Ci rozpocząć pracę z tą biblioteką. Poznaj OpenGL i zachwyć świat swoimi grafikami!

- Co to jest OpenGL?
- Historia i perspektywy OpenGL
- Pierwszy program
- Rzutowanie i transformacje
- Cieniowanie i oświetlenie
- Tekstury i antyaliasing
- Zaawansowane przekształcanie geometrii
- Cienie
- Odwzorowanie środowiska i nierówności powierzchni
- Rozszerzenia
- Macierze i wektory
- Język GLSL

Wykreuj swój świat z OpenGL!

Helion
31685 numer katalogowy
księgarnia internetowa
http://helion.pl
zamówienia telefoniczne
0 801 339900
0 601 339900

Sprawdź najnowsze promocje:
http://helion.pl/promocje
Książki najchętniej czytane:
http://helion.pl/bestsellery
Zamów informacje o nowościach:
http://helion.pl/nowosci

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
http://helion.pl

ISBN 978-83-283-0193-1
9 788328 301931

Informatyka w najlepszym wydaniu
cena: 89,00 zł