Pakiet glsl-progII

Pakiet aplikacji przykładowych opisanych w książce *OpenGL i GLSL* (*nie taki krótki kurs*) (wydanie II, w przygotowaniu) udostępniłem w dwóch wersjach.

Plik `glsl-progII.tar.gz` zawiera spakowane oryginalne drzewo katalogów, w których pisałem i uruchamiałem te aplikacje w systemie Linux. Jeśli jest zainstalowane wszystko, czego trzeba (tj. kompilator **gcc**, program **make** i biblioteki), to wystarczy rozpakować ten plik, wejść do katalogu `glsl-progII` i wydać polecenie `make`, bo są tam gotowe (napisane „odręcznie") pliki `Makefile`. W tej wersji pakietu są używane dowiązania symboliczne do plików znajdujących się w innych podkatalogach; spakowanie tego pakietu programem **zip** i rozpakowanie (np. w systemie Windows) powoduje powstanie kopii plików zamiast dowiązań, co zajmuje 10 razy więcej miejsca na dysku.

Plik `glsl-progIICM.zip` zawiera pakiet przygotowany do kompilacji z wykorzystaniem programu **cmake** i powinien dać się skompilować tak samo w systemach Linux i MS Windows. W tym pakiecie są wszystkie wersje trzech aplikacji opisanych w zasadniczej części *nie takiego krótkiego kursu*, ale nie ma w nim aplikacji przedstawionych w uzupełnieniach na końcach poszczególnych rozdziałów i w dodatkach B–G po ostatnim rozdziale.

Książka przedstawia kurs podstaw programowania grafiki komputerowej, a nie przygotowywania komercyjnych pakietów oprogramowania. Dlatego pliki `Makefile` i skrypty cmake'a w pakiecie są napisane przy założeniu, że Czytelnicy, którzy zdecydują się zostać Kursantami, będą kody źródłowe w językach C i GLSL czytać, modyfikować, przeprowadzać eksperymenty i zdobywać własne doświadczenia. W wyniku kompilacji nie mają powstać wersje „produkcyjne" aplikacji, a raczej wersje robocze, których działanie krok po kroku można podglądać przy użyciu uruchamiacza („*debuggera*"). Temu celowi jest też podporządkowany wybór miejsc, w których umieszczane są binarne pliki wykonywalne: w katalogach z plikami źródłowymi (i szaderami) i również z tego powodu pomocnicze procedury są umieszczone w dużych plikach źródłowych (takich jak `utilities.c` lub `xwidgets.c`) zamiast w osobnych krótkich plikach, z których buduje się biblioteki.

Aplikacje używają następujących ponadstandardowych bibliotek: **OpenGL**, **FreeGLUT**, **GLFW** oraz **TIFF**. Niżej opisuję, co i jak trzeba zainstalować, aby móc cieszyć się nimi.

Linux/X Window

Instalacja sterowników karty graficznej w szczególności wiąże się z zainstalowaniem biblioteki `libGL.so` (**OpenGL**), ale nie podejmuję się opisać tego procesu, bo jest zbyt wiele różnych dystrybucji Linuksa i marek i modeli karty graficznej. Osobny problem to instalacja sterowników na laptopach; mam tylko nadzieję, że każdy zainteresowany umie to zrobić lub znajdzie kogoś, kto umie i pomoże.

Oczywiście trzeba zainstalować pakiety programistyczne, tj. dowolny edytor ASCII, kompilator **gcc** i program **make**, przyda się też uruchamiacz **gdb** (ja korzystam z niego za pośrednictwem interfejsu **ddd**). Jeśli chcemy korzystać z programu **cmake**, to też go trzeba zainstalować. W różnych dystrybucjach Linuksa nazwy pakietów, w których są biblioteki, są różne,

ale składają się z nazwy biblioteki opatrzonej przyrostkiem `-dev` albo `-devel`, więc da się je odgadnąć. Polecenia instalacji w przykładowych dystrybucjach (wydawane po uzyskaniu uprawnień administratora) są takie:

Debian:

```
apt-get install gcc cmake libtiff5-dev freeglut3-dev libglfw3-dev
```

Fedora:

```
dnf install gcc cmake libtiff-devel freeglut-devel glfw-devel
```

OpenSUSE:

```
zypper install gcc cmake libtiff-devel freeglut-devel libglfw-devel
```

Ubuntu:

```
apt update; apt install gcc cmake libtiff-dev freeglut3-dev libglfw-dev
```

Może się okazać konieczne doinstalowanie innych niezbędnych pakietów, np. `xrandr`, `Xinerama` itd. Menedżer pakietów, jeśli nie zrobi tego automatycznie, to wyświetli odpowiednie informacje.

MS Windows

Żeby było jasne: mnie interesuje grafika, a nie system operacyjny Windows[1]. Wykonałem pewną (niemałą) pracę na rzecz osób zainteresowanych grafiką *i* tym systemem, ale niewiele więcej jestem w stanie dla nich uczynić.

Po długiej, ciężkiej i nie do końca wygranej walce udało mi się doprowadzić aplikacje do działania także w systemie Windows; w szczególności aplikacja 3, składająca się z części graficznej (niezależnej od systemu) i z części okienkowej napisanej pierwotnie dla X Window, ma teraz wersję z częścią okienkową przystosowaną do współpracy z Windowsami (część graficzna jest w obu wersjach ta sama). Użyty przeze mnie kompilator jest częścią zintegrowanego środowiska programistycznego **Visual Studio Community 17 2022**, którego zapewne nie trzeba instalować, jeśli ktoś jest bardziej zainteresowany pakietem **MinGW** (o którym ja wiem tylko tyle, że istnieje).

Trzeba mieć zainstalowany program **cmake** i biblioteki, przy czym do ich instalacji jest potrzebny menedżer pakietów **vcpkg**. Aby jego zainstalować, trzeba mieć zainstalowanego **Gita** (instrukcje w Internecie); jak już jest Git, to dalej w konsoli wydaje się polecenie

```
git clone https://github.com/Microsoft/vcpkg.git
```

Po wejściu do katalogu, w którym jest menedżer trzeba napisać

```
setx VCPKG_DEFAULT_TRIPLET x64
./bootstrap-vcpkg.sh
./vcpkg integrate install
```

[1] po polsku „łokna”

Wykonując ostatnie z tych poleceń, program vcpkg wyświetli tekst opcji wywołania programu cmake koniecznej do sprawienia, aby znalazł on pakiety z bibliotekami. W moim przypadku to był tekst

```
-DCMAKE_TOOLCHAIN_FILE=C:/Users/przemek/vcpkg/vcpkg/scripts/-
  buildsystems/vcpkg.cmake
```

ale najpierw trzeba zainstalować pakiety, poleceniem

```
./vcpkg install tiff freeglut glfw3
```

Dalej można postąpić na dwa sposoby. Prostszy z nich polega na wywołaniu programu Visual Studio i otwarciu tym programem katalogu `glsl-progIICM`. Znalazłszy w nim plik `CMakeLists.txt`, program (bez pytania o zgodę) wygeneruje potrzebne mu podkatalogi i pliki i wtedy można będzie wydać polecenie `Build`, które spowoduje skompilowanie wszystkich aplikacji. Podobnie jak w Linuksie, plik wykonywalny każdej aplikacji jest zapisywany w katalogu z jej plikami źródłowymi, gdzie jest gotów do działania. Niestety, nie zdołałem odkryć sposobu skłonienia Visual Studia do rozpoczęcia wykonywania skompilowanej w ten sposób aplikacji w uruchamiaczu (ale nie zamierzam twierdzić, że to jest niemożliwe).

W drugim sposobie po wejściu (w konsoli) do katalogu `glsl-progIICM` trzeba napisać

```
cmake -DCMAKE_TOOLCHAIN_FILE=C:/Users/przemek/vcpkg/vcpkg/-
  scripts/buildsystems/vcpkg.cmake .
```

Uwaga: Nie wolno pominąć tego długiego argumentu wywołania programu cmake, bo wtedy on nie znajdzie pakietów i nie zrobi tego nawet po ponownym wywołaniu z tym argumentem; pozostaje tylko cały katalog z zawartością wyrzucić do kosza, ponownie rozpakować plik `glsl-progIICM.zip` i wydać polecenie `cmake` jak należy[2].

Dwukrotne pstryknięcie ikony lub nazwy pliku `glsl-progII.sln` powoduje uruchomienie programu Visual Studio, otwarcie przezeń tego pliku i wykonanie zawartych w nim poleceń. Wtedy można wydać polecenie `Build`, aby skompilować aplikacje. Ale aby można było rozpocząć wykonywanie aplikacji w uruchamiaczu (poleceniem `Start Debugging`), trzeba usunąć cele `ALL_BUILD` i `ZERO_CHECK`, które program cmake z nieznanych mi powodów dodaje do projektu każdej aplikacji. Jak się tego nie zrobi, to pojawia się komunikat o nieznalezieniu pliku wykonywalnego `ALL_BUILD` — wygląda to tak, jak gdyby Visual Studio chciało uruchamiać wszystko, tylko nie aplikację.

Aplikacja 1

Wszystkie wersje pierwszej aplikacji korzystają z biblioteki FreeGLUT.

Pierwsza aplikacja wyświetla obraz nieruchomego dwudziestościanu foremnego i reaguje tylko na zmianę wymiarów okna i na napisanie na klawiaturze małej lub wielkiej litery `W` (po której wyświetlane są wierzchołki), `K` (powodującej wyświetlanie krawędzi) i `S` (po której wraca do wyświetlania obrazu ścian). Naciśnięcie klawisza `Esc` kończy działanie aplikacji.

[2] Odkrycie, dlaczego cmake nie umie sobie poradzić i co zrobić, aby umiał, kosztowało mnie sporo czasu.

Aplikacja 1A

Ten wariant aplikacji umożliwia oglądanie dwudziestościanu z różnych stron oraz animację, tj. samoczynne obracanie dwudziestościanu ze stałą prędkością kątową. Aby zmienić położenie obserwatora, należy nacisnąć lewy przycisk myszy i, trzymając go, przesuwać mysz. Uruchomienie i zatrzymanie animacji wykonuje się, naciskając klawisz spacji.

Aplikacja 1B

Aplikacja 1B wyświetla obraz dwudziestościanu oświetlonego przy użyciu modelu oświetlenia Lamberta. Litera `L` powoduje przełączanie obrazu ścian — z oświetleniem lub bezpośrednio w kolorach farb, jakimi jest pomalowany obiekt.

Aplikacja 1C

Aplikacja 1C w czasie, gdy jest w trybie obracania obserwatora wokół obiektu (tj. gdy lewy przycisk jest naciśnięty), wyświetla współrzędne położenia obserwatora. Litera `M` przełącza używany font, między dużym i małym.

Używane przez aplikację fonty zawierają znaki, których kody ASCII są między 32 a 127, a więc nie są w nich obecne „polskie litery” ą, ć, ę, ł, ń, ó, ś, ź, ż. W podkatalogu `utilities` są dodatkowe trzy fonty ze znakami o kodach od 32 do 255, kodowane zgodnie ze standardem ISO-8859-2 oraz font jeden font z cyrylicą, kodowaną według standardu ISO-8859-5.

Aplikacja 1D

Aplikacja 1D umożliwia rozdrabnianie krawędzi oraz trójkątnych ścian dwudziestościanu oraz zamianę ich na łuki okręgów wielkich i trójkąty sfryczne na sferze jednostkowej. Do przełączania między dwudziestościanem i otrzymaną z niego sferą służy litera `T`.

Aplikacja 1E

Aplikacja 1E wyświetla trzy sfery obrazujące Słońce i orbitujące wokół niego Ziemię i Księżyc, animowane za pomocą łańcucha kinematycznego.

Aplikacja reaguje *tylko* na literę `M`, która przełącza wielkość fontu używanego do wyświetlania położenia obserwatora, znaki + i –, które powodują przyspieszanie i spowalnianie animacji, spację, która animację uruchamia albo zatrzymuje, oraz klawisz `Esc` kończący działanie aplikacji. Położenie obserwatora można zmieniać za pomocą myszy (po naciśnięciu lewego przycisku), a ponadto można wykonywać powiększenia i zmniejszenia obrazu, obracając rolkę myszy.

Aplikacja 1F

Aplikacja 1F *zamiast* dwudziestościanu wyświetla obiekt, którego opis został przeczytany z pliku w formacie SMF. Nazwę tego pliku trzeba podać jako parametr wywołania aplikacji,

pisząc `app1f -i plik.smf`. Aplikacja reaguje na litery `W`, `K` i `S` wyświetlaniem wierzchołków, krawędzi lub ścian obiektu, litery `L` i `M` przełączające oświetlenie i wielkość fontu oraz spację i klawisz `Esc`.

Aplikacja 2

Wszystkie wersje drugiej aplikacji korzystają z biblioteki GLFW.

Druga aplikacja wyświetla model czajnika z Utah, jako przykład sposobu rysowania tensorowych płatów powierzchni Béziera. Obracanie obserwatora wokół czajnika użytkownik może wykonać za pomocą myszy, tak samo jak w pierwszej aplikacji. Klawisz spacji włącza i wyłącza animację („samoczynne" obracanie czajnika wokół pionowej osi). Ponadto aplikacja reaguje na napisanie znaku + albo -, odpowiednio zwiększając i zmniejszając wartości parametrów rozdrabniania płatów (w zakresie od 3 do 30) i na napisanie znaku `N`, który przełącza wektory normalne używane w modelu oświetlenia (Lamberta): domyślnie brane są wektory normalne powierzchni, alternatywą są wektory normalne płaszczyzn trójkątów przybliżających płaty.

Aplikacja 2A

Aplikacja 2A powstała przez dodanie możliwości rysowania siatek kontrolnych płatów, z których składa się model czajnika. Do włączania i wyłączania rysowania siatek służy klawisz z literą `C`. Ponadto napisanie litery `S` powoduje przełączenie między (domyślnym) rysowaniem trójkątów przybliżających płaty a rysowaniem tylko krawędzi tych płatów.

Aplikacja 2B

Do aplikacji 2A dodałem drugi obiekt — torus, obracający się nad wylotem dziobka czajnika — i w ten sposób powstała aplikacja 2B. Torus jest zbudowany z 9 wymiernych płatów Béziera stopnia $(2, 2)$. Aplikacja reaguje na te same klawisze, co 2A, i nie reaguje na żadne inne.

Aplikacja 2C

W aplikacji 2C jest zaimplementowany nowy model oświetlenia — Blinna-Phonga. Do przełączania modeli oświetlenia służy klawisz z literą `B`. Klawisze z cyframi `1` i `0` przełączają materiał, z którego wykonany jest czajnik, między czymś w rodzaju białej porcelany i żółtawej terakoty.

Aplikacja 2D

Aplikacja 2D demonstruje sposób nakładania tekstury na powierzchnię. Aby zobaczyć efekt, trzeba napisać cyfrę 2. Tekstura jest interpretowana jak obraz namalowany (odbijającą światło farbą) na rysowanych powierzchniach.

Ta i następne wersje drugiej aplikacji dokonują antyaliasingu obrazów.

Aplikacja 2E

Aplikacja 2E przedstawia przykład rysowania poza ekranem — dodatkowym elementem w scenie jest lustro, w którym scena się odbija. Obraz sceny odbitej w lustrze jest wykonywany poza ekranem, a następnie (podczas rysowania obrazu końcowego) nakładany na lustro jako tekstura.

Nie ma nowych elementów interakcji z użytkownikiem.

Aplikacja 2F

Aplikacja 2F zawiera przykład implementacji proceduralnej tekstury odkształceń powierzchni. Efekt „nierówności" na powierzchni jest osiągnięty przez zaburzanie wektora normalnego. Włącza się go i wyłącza, naciskając klawisz z literą `D`. Klawisz z literą `M` służy do włączania i wyłączania modyfikowania głębokości fragmentów powierzchni z nałożoną teksturą odkształceń.

Aplikacja 2G

W aplikacji 2G został zaimplementowany algorytm wyznaczania cieni. Cienie na obrazie można wyłączyć i ponownie włączyć, naciskając klawisz z literą `U`.

Aplikacja 2H

W aplikacji 2H jest obsługiwany łańcuch kinemayczny. Służy do tego ta sama biblioteczka (w pliku `../utilities/linkage.c`), która była użyta w aplikacji 1F. Tym razem przekształcaniem obiektów (punktów kontrolnych płatów, z których zbudowany jest czajnik) zajmuje się szader obliczeniowy działający na karcie graficznej. Zestaw klawiszy wydających polecenia dla tej aplikacji jest taki sam jak dla poprzedniej.

Aplikacja 2I

W aplikacji 2I jest wykonywana symulacja układu cząsteczek, którego zadaniem jest otrzymanie obrazu wylatującej z dziobka czajnika pary, która się skrapla, tworząc mgłę. Symulacja jest wykonywana przez kolejny szader obliczeniowy. Jej włączanie i wyłączanie następuje po napisaniu litery `P`. Litera `R` przywraca stan początkowy symulacji (tj. „resetuje ją").

Aplikacja 2J

W aplikacji 2J środkami dostępnymi w nowym OpenGL-u zrealizowany jest bufor akumulacji. Przy jego użyciu symulowane są efekty głębi ostrości oraz rozmycia obiektów w ruchu. Litera `X` całkowicie wstrzymuje animację albo ją wznawia. Klawisze ze znakami < i > powodują zmienianie odległości, na którą jest nastawiona ostrość „obiektywu", którym aplikacja wykonuje obrazy. Klawisz `^` i litera `V` zwiększają i zmniejszają liczbę obrazów mieszanych

w buforze akumulacji w celu otrzymania opisanych efektów. Klawisz `Z` włącza i wyłącza rozmycie obiektów w ruchu.

„Obiektyw" można też nastawiać za pomocą rolki myszy. Jej obracanie, gdy żaden przycisk nie jest naciśnięty, powoduje zmianę „długości ogniskowej", gdy naciśnięty jest lewy przycisk zmienia odległość nastawienia ostrości, a gdy naciśnięty jest przycisk prawy, zmieniana jest średnica „otworu przysłony".

Aplikacja 2K

Aplikacja 2K ma podobne możliwości jak 2J, z tym, że obrazy mieszane w buforze akumulacji są wykonywane jednocześnie (jest użyte rysowanie na wielu warstwach tekstury załączonej do pozaekranowego bufora obrazu). Nie ma tu możliwości rysowania obiektów rozmytych w ruchu, ale jest zaimplementowana stereoskopia — po naciśnięciu klawisza `F1` w oknie pojawia się (albo znika) obraz anaglifowy, do oglądania przez kolorowe okulary.

Aplikacja 3

Wszystkie wersje trzeciej aplikacji mają natywne wersje dla systemów Linux/X Window i Windows. Graficzny interfejs użytkownika jest realizowany przy użyciu biblioteki Xlib albo podsystemu GDI.

Aplikacja 3 wykonuje obraz powierzchni reprezentowanej przez siatkę nieregularną, stanowiącą model dłoni. Aplikacja wykorzystuje szader obliczeniowy, który dokonuje zagęszczania siatek, co umożliwia narysowanie mniej lub bardziej dokładnego przybliżenia gładkiej powierzchni. Wyboru dokonuje się za pomocą przełączników w menu. Za pomocą myszy można obracać obserwatora wokół obiektu. Klawisz spacji włącza i wyłącza animację, tj. obracanie obiektu. Klawisz z literą `K` przełącza między rysowaniem ścian siatki (tj. nieprzezroczystych trójkątów) a krawędzi tych ścian.

Aplikacja 3A

Dodatkowy szader obliczeniowy w aplikacji 3A oblicza, dla każdego wierzchołka zagęszczonej siatki, wektor, który może być użyty jako wektor normalny powierzchni na potrzeby modelu oświetlenia. Do przełączania między tymi wektorami a wektorami normalnymi płaszczyzn trójkątów służy klawisz z literą `N`.

Aplikacja 3B

Aplikacja 3B tworzy łańcuch kinematyczny, aby umożliwić odkształcanie siatki przed jej zagęszczaniem. Odkształcanie wykonuje jeszcze jeden szader obliczeniowy. Dodane do menu suwaki sterują parametrami artykulacji (kątami ugięcia poszczególnych stawów), co umożliwia zginanie palców rysowanej dłoni.

Aplikacja 3C

Model dłoni został rozszerzony o paznokcie wykonane z płatów Béziera. Tę aplikację obsługuje się tak samo jak 3B.

Aplikacja 3D

Aplikacja 3D ma bardziej rozbudowany graficzny interfejs użytkownika, dzięki któremu możliwe jest tworzenie animacji metodą klatek kluczowych. Dla ustalonego węzła na osi czasu można określić wartości parametrów artykulacji. Po włączeniu animacji wartości tych parametrów w chwilach rysowania kolejnych klatek animacji będą otrzymywane przez interpolację funkcjami sklejanymi wartości parametrów podanych we wspomnianych węzłach.

Można też zadawać położenia obserwatora wokół obiektu; obroty obserwatora w $\mathbb{R}^3$ są reprezentowane przez kwaterniony, a interpolacja obrotów polega na obliczeniu odpowiedniego punktu sklejanej kwaternionowej krzywej interpolacyjnej. Dokładniejszy opis jest w książce.