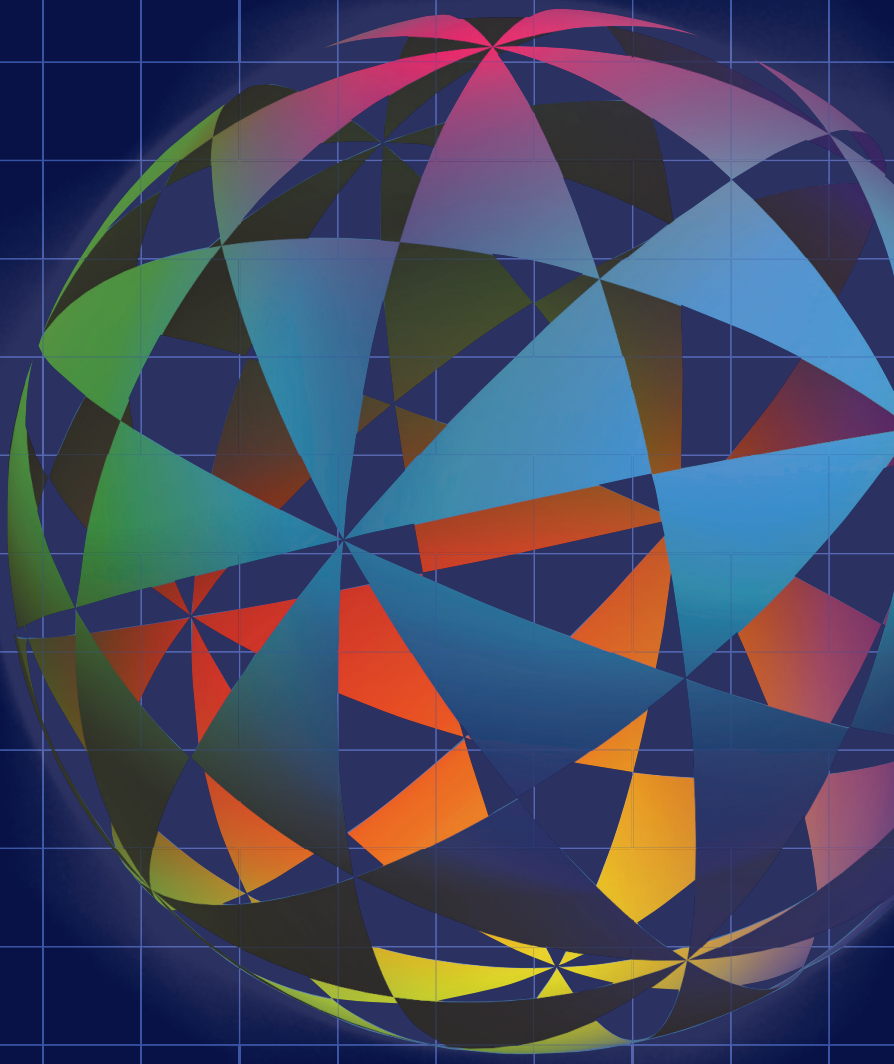
Przemysław Kiciak

OpenGL i GLSL

(nie taki krótki kurs)

Część III



PWN

OpenGL i GLSL

Przemysław Kiciak

OpenGL i GLSL

(nie taki krótki kurs)

Część III

PWN

Projekt okładki i stron tytułowych Anna Ludwicka
Wydawca Wioleta Szczygielska-Dybciak
Redaktor prowadzący Jolanta Kowalczuk
Redaktor Maria Kasperska
Skład systemem TEX Przemysław Kiciak
Koordynator produkcji Anna Bączkowska

Recenzenci
prof. dr hab. Krzysztof Diks
prof. dr hab. inż. Przemysław Rokita

ISBN 978-83-01-20864-6 cz. III
ISBN 978-83-01-20866-0 cz. I–III

Wydanie I
Warszawa 2019

Wydawnictwo Naukowe PWN
02-460 Warszawa, ul. Gottlieba Daimlera 2
tel. (0-22) 69 54 321, faks 22 69 54 288
infolinia 801 33 33 88
e-mail: pwn@pwn.com.pl, reklama@pwn.pl
www.pwn.pl

Druk i oprawa: Partner Poligrafia Andrzej Kardasz

Spis treści części III

26

Graficzny interfejs użytkownika w X Window

Interfejs użytkownika opisanych dotąd aplikacji, mówiąc delikatnie, pozostawia co nieco do życzenia: wszystkie polecenia oprócz zmieniania wymiarów okna i położenia obserwatora użytkownik wydaje, naciskając jakiś klawisz. Nie da się w ten sposób wygodnie wprowadzać wielkości analogowych, takich jak parametry oświetlenia lub parametry artykulacji, a zresztą klawiatura bywa potrzebna do wprowadzania napisów (liczb, nazw plików itp.), a wtedy użytkownik powinien na bieżąco widzieć, co pisze. Dlatego w bardziej skomplikowanych aplikacjach potrzebny jest **graficzny interfejs użytkownika** (*GUI, graphical user interface*), czyli rozmaite **wihajstry** (*widgets*), które użytkownik widzi w oknie i za których pośrednictwem może wprowadzać dane i wydawać polecenia. Niestety, biblioteka FreeGLUT ma tylko bardzo ograniczony i niedziałający poprawnie z nowym OpenGL-em (zobacz p. 3.1.1) zestaw procedur realizujących GUI, a w bibliotece GLFW nie ma nawet tego.

Mój kłopot polega na tym, że nie chcę zbytnio oddalać się od kursu OpenGL-a, a jednocześnie nie chcę narażać Czytelników na studiowanie kiepskiego opisu jakiejś biblioteki GUI, której akurat nie mają i z rozmaitych powodów nie mogą sobie zainstalować. Oczywiście, można stworzyć znakomity GUI w aplikacji FreeGLUT-a lub GLFW, w którym wihajstry rysuje OpenGL, ale (wobec konieczności dostarczenia odpowiednich szaderów i utworzenia buforów z danymi opisującymi wihajstry) jest to dużo bardziej pracochłonne niż pouczające. Jeśli więc obrazy wihajstrów nie przedstawiają skomplikowanych obiektów trójwymiarowych, to łatwiej jest użyć dostępnych procedur grafiki dwuwymiarowej i rysować wihajstry w (znacznie prostszym do użycia) trybie natychmiastowym. Obrazy większości wihajstrów są na tyle nieskomplikowane, że czas ich rysowania będzie niezauważalny.

Ryzykując utratę zainteresowania Czytelników, których komputery nie znają dobrodziejstw systemu X Window, a także tych, którzy już mają swoją ulubioną bibliotekę GUI, postanowiłem napisać swój zestaw procedur, rysujący wihajstry za pomocą biblioteki Xlib. Decydując się na użycie tej biblioteki do rysowania, muszę też użyć zawartych w niej procedur obsługi komunikatów o zdarzeniach wejściowych i użyć biblioteki GLX do utworzenia kontekstu OpenGL-a. Mam nadzieję, że żaden Czytelnik nie będzie przez tę decyzję czuł się

pokrzywdzony bardziej niż inni[1]. Opisany tu mechanizm komunikowania się aplikacji z wihajstrami może być użyty do zaimplementowania wihajstrów rysowanych przez OpenGL-a. W razie potrzeby można też dorabiać nowe rodzaje wihajstrów — trzeba tylko wymyślić dla każdego z nich wygląd i sposób działania.

26.1. Struktury danych i procedury podstawowe

Do zrealizowania wihajstra potrzebne są dwie procedury[2]. Pierwsza z nich przetwarza wejście (tj. reaguje na komunikaty o działaniach użytkownika), a druga wyświetla odpowiedni obraz w oknie, aby użytkownik widział, gdzie ma umieścić kursor przed naciśnięciem przycisku myszy albo w jakim wihajster jest stanie (np. czy wihajster — przełącznik — jest w danej chwili włączony).

Listing 26.1 przedstawia typy danych zdefiniowane w celu zaimplementowania GUI. Każdy wihajster jest opisany przez strukturę typu `xwidget`, której pola to: `id` — identyfikator wihajstra, `r` — opis prostokąta zajmowanego przez wihajster w oknie, `state` — stan wihajstra, `input` i `redraw` — wskaźniki procedury przetwarzającej komunikaty wejściowe i procedury rysującej wihajster, `wm` — wskaźnik struktury menu okna, w którym wihajster ma się pojawić, `link` — para wskaźników tworzących listę wihajstrów tego menu, oraz `data0`, `data1` — wskaźniki danych specyficznych dla wihajstra konkretnego rodzaju.

Struktura typu `xwinmenu` reprezentuje zbiór wihajstrów należących do danego okna (lub podokna) utworzonego przez system X Window. Pole `window` jest identyfikatorem okna. Pole `pixmap` zawiera identyfikator **kanwy** (*pixmap*), na której odbywa się rysowanie wihajstrów; można by je rysować bezpośrednio w oknie, ale choć to zabiera znikomą ilość czasu, byłoby widoczne migotanie (spowodowane wyświetlaniem w oknie niedokończonych obrazów). Dlatego wihajstry mają być rysowane na tej kanwie, a jej zawartość będzie kopiowana do okna, gdy obrazy wszystkich wihajstrów będą gotowe. Pole `r` opisuje wymiary okna (i kanwy). W polach `prevx`, `prevy` i `prevmask` będą pamiętane położenia kursora w oknie i stan przycisków myszy *po* zakończeniu obsługi komunikatu, aby można ich było użyć podczas obsługi następnego komunikatu. Pole `changed` ma przypisywaną wartość niezerową, gdy któryś wihajster lub aplikacja sygnalizuje potrzebę odświeżenia obrazu w oknie. Pole `expose_sent` służy do tego, aby zapobiegać wysyłaniu do okna niepotrzebnych komunikatów `Expose` podczas przetwarzania komunikatów o przesunięciu kursora, co będzie wyjaśnione dalej. Wskaźnik `data` jest przeznaczony do użytku aplikacji. Pole `wlist` jest nagłówkiem listy dwukierunkowej wihajstrów. Pole `redraw` jest wskaźnikiem procedury rysującej zawartość okna, czyli tło i na nim wszystkie wihajstry. Procedura ta ma używać do rysowania *albo* procedur systemu X Window (z biblioteki Xlib), *albo* OpenGL-a, przy czym ten sam sposób rysowania w oknie obowiązuje procedury rysujące *wszystkie* wihajstry w tym oknie. W tym rozdziale opisałem tylko najprostsze przykłady wihajstrów rysowanych przez

[1] Użytkownikom Windowsów pozostaje użycie jakiegoś gotowego oprogramowania lub własnoręczne napisanie działającej w tym środowisku biblioteki, naśladującej w ostateczności moje rozwiązanie dla systemu X Window. To też może być dobra zabawa, życzę powodzenia.

[2] W języku C++ wihajster powinien być obiektem z dwiema metodami wirtualnymi.

Listing 26.1. Typy danych dla systemu wihajstrów

```C
#define WDGSTATE_DEFAULT  0
#define WDGSTATE_INACTIVE 1

typedef struct {
    struct xwidget *prev, *next;
  } xwlink;

typedef struct xwidget {
    int             id;
    XRectangle      r;
    int             state;
    char            (*input)(struct xwidget *wdg,
                             int msg, int key, int x, int y);
    void            (*redraw)(struct xwidget *wdg);
    struct xwinmenu *wm;
    xwlink          link;
    void            *data0, *data1;
  } xwidget;

typedef struct xwinmenu {
    Window       window;
    Pixmap       pixmap;
    XRectangle   r;
    int          prevx, prevy;
    unsigned int prevmask;
    char         changed, expose_sent;
    void         *data;
    xwidget      *empty, *focus;
    XEvent       *ev;
    xwlink       wlist;
    void         (*redraw)(struct xwinmenu *wm);
    void         (*callback)(struct xwidget *wdg,
                             int msg, int key, int x, int y);
  } xwinmenu;
```

procedury Xlib, ale trzecia aplikacja otworzy okno z wihajstrami, którego cała zawartość jest rysowana przez OpenGL-a.

Wihajstry w oknie są połączone w listę dwukierunkową, której uporządkowanie odpowiada kolejności rysowania: pierwszy element jest „na samym spodzie", a ostatni „na samym wierzchu" stosu wihajstrów, a zatem jeśli poszczególne wihajstry nakładają się, to element „wyżej" (czyli położony dalej w liście) zasłania wihajster pod spodem. Podczas odświeżania obrazu w oknie wihajstry są rysowane „od dołu do góry", czyli od pierwszego do ostatniego. Natomiast kolejność wyszukiwania wihajstra, do którego ma trafić komunikat o zdarzeniu, które miało miejsce, gdy kursor był w pewnym punkcie okna, jest „od góry do dołu", bo komunikat ma trafić do wihajstra, który we wskazanym punkcie jest widoczny.

Procedura przetwarzania wejścia wihajstra ma poinformować, czy komunikat został przetworzony, czy nie, podając odpowiednio niezerową wartość powrotną albo zero. W tym ostatnim przypadku lista wihajstrów będzie przeszukiwana dalej, w celu znalezienia innego wihajstra zainteresowanego tym komunikatem.

Pierwszym elementem listy wihajstrów w menu jest pusty wihajster o zerowych wymiarach. Jest on dodatkowo wskazywany przez pole `empty` i przydaje się jako parametr opisanej dalej procedury wskazywanej przez pole `callback`. Pole `focus` ma zwykle wartość `NULL`, ale wihajstry mogą na pewien czas przypisywać mu swój adres. Wtedy kolejne komunikaty (do odwołania, czyli do ponownego przypisania wartości `NULL`) będą wysyłane do tego wihajstra. Jeśli na przykład użytkownik manipuluje suwakiem i przesunie kursor poza jego obszar, to suwak nadal ma otrzymywać komunikaty do chwili, gdy użytkownik zwolni przycisk myszy, w odpowiedzi na co suwak wyłączy tryb manipulowania sobą.

Parametry procedur przetwarzania wejścia wihajstrów opisują uproszczony komunikat, przy czym opis ten w większości przypadków jest wystarczający dla wykonania właściwej reakcji wihajstra na zdarzenie. Pole `ev` wskazuje strukturę typu `XEvent` z pełną informacją o komunikacie dostarczoną przez system X Window, na wypadek gdyby taka informacja była potrzebna.

Pole `callback` wskazuje procedurę aplikacji, która ma być wywoływana przez wihajstry w celu powiadomienia na przykład o naciśnięciu guzika lub przesunięciu suwaka. Procedura ta jest wywoływana także w razie nieprzetworzenia komunikatu wejściowego przez żaden wihajster w oknie albo w razie otrzymania komunikatu takiego jak `ClientMessage`. W takich przypadkach pierwszy parametr tej procedury ma wartość pola `empty`.

Listing 26.2 przedstawia procedury tworzenia, rysowania zawartości i likwidacji menu, tj. listy wihajstrów dla okna. Procedura `WinMenuRedraw`, posługując się procedurami z biblioteki Xlib, rysuje tło, a następnie wywołuje procedurę rysowania po kolei dla wszystkich wihajstrów z wyjątkiem nieaktywnych w danym momencie. Tło i wihajstry są rysowane na kanwie, której identyfikator jest wartością pola `pixmap`.

Listing 26.2. Procedury tworzenia, rysowania i likwidacji menu okna

C

```
1: typedef void (*xcallback)(struct xwidget *wdg,
2:                           int msg, int key, int x, int y);
3: typedef void (*xmredraw)(struct xwinmenu *wm);
4:
5: void WinMenuRedraw ( xwinmenu *wm )
6: {
7:   xwidget *wdg;
8:
9:   XSetForeground ( xdisplay, xgc, XWP_MENU_BACKGROUND_COLOUR );
10:  XFillRectangle ( xdisplay, wm->pixmap, xgc, 0, 0,
11:                   wm->r.width, wm->r.height );
12:  for ( wdg = wm->wlist.next;  wdg;  wdg = wdg->link.next )
13:    if ( wdg->state != WDGSTATE_INACTIVE )
14:      wdg->redraw ( wdg );
```

```
15: } /*WinMenuRedraw*/
16:
17: xwinmenu *NewWinMenu ( Window window, int w, int h, int x, int y,
18:                        void *data, xmredraw redraw, xcallback callback )
19: {
20:   xwinmenu *wm;
21:
22:   if ( (wm = malloc( sizeof(xwinmenu) )) ) {
23:     memset ( wm, 0, sizeof(xwinmenu) );
24:     wm->window = window;
25:     wm->r.width = w;  wm->r.height = h;  wm->r.x = x;  wm->r.y = y;
26:     wm->data = data;
27:     wm->redraw = redraw ? redraw : WinMenuRedraw;
28:     if ( !redraw )
29:       wm->pixmap = XCreatePixmap ( xdisplay, window, w, h, 24 );
30:     wm->callback = callback;
31:     wm->empty = NewEmptyWidget ( wm, 0 );
32:     wm->changed = true;
33:   }
34:   return wm;
35: } /*NewWinMenu*/
36:
37: void DestroyWinMenu ( xwinmenu *wm )
38: {
39:   xwidget *wdg, *w;
40:
41:   for ( wdg = wm->wlist.next; wdg; ) {
42:     w = wdg;  wdg = w->link.next;
43:     free ( w );
44:   }
45:   if ( wm->pixmap )
46:     XFreePixmap ( xdisplay, wm->pixmap );
47:   free ( wm );
48: } /*DestroyWinMenu*/
```

Procedura `NewWinMenu` rezerwuje strukturę danych menu i zapisuje w jej polach odpowiednie informacje, w szczególności tworzy pusty wihajster, który staje się pierwszym elementem listy. Wywołując tę procedurę, aplikacja może podać parametr `redraw` pusty (`NULL`) i wtedy procedurą rysującą w tym oknie stanie się procedura `WinMenuRedraw`. Aplikacja może też podać adres innej procedury rysującej, która jeśli korzysta z OpenGL-a, to wszystkie wihajstry w tym menu muszą być rysowane za jego pomocą. Parametr `callback` musi być adresem procedury w aplikacji, która będzie wywoływana za każdym razem, gdy wihajster ma dla aplikacji komunikat, albo gdy menu przekazuje aplikacji komunikat od systemu X Window, taki jak `ConfigureNotify` lub `ClientMessage`.

Procedura `DestroyWinMenu` likwiduje kolejno wszystkie wihajstry (tj. zwalnia zajmowaną przez nie pamięć), a potem likwiduje kanwę (nieobecną w oknach z zawartością rysowaną przy użyciu OpenGL-a) i menu.

Procedura przedstawiona na listingu 26.3 rezerwuje pamięć na strukturę opisującą wihajster i inicjalizuje jej pola wspólne dla wszystkich wihajstrów. Nowy wihajster jest dołączany *na koniec* listy wihajstrów menu okna (zatem kolejność tworzenia wihajstrów będzie kolejnością ich rysowania). Ponadto zapamiętywane są wymiary i położenie prostokąta zajmowanego przez wihajster w oknie i nadany przez aplikację identyfikator wihajstra. Początkowa wartość pola `state` określa stan, w którym wihajster niczego szczególnego nie robi. Inne wartości będzie temu polu przypisywać procedura wskazywana przez parametr `input` lub dowolna inna procedura aplikacji. W szczególności od stanu wihajstra może zależeć jego wygląd na ekranie.

Listing 26.3. Procedura `NewWidget`

C

```
typedef char (*xwinput)(struct xwidget *wdg,
                        int msg, int key, int x, int y);
typedef void (*xwredraw)(struct xwidget *wdg);

xwidget *NewWidget ( struct xwinmenu *wm, int size, int id,
                 int w, int h, int x, int y,
                 xwinput input, xwredraw redraw, void *data0, void *data1 )
{
  xwidget *wdg;

  if ( size < sizeof(xwidget) )
    size = sizeof(xwidget);
  if ( (wdg = malloc ( size )) ) {
    memset ( wdg, 0, size );
    if ( !wm->wlist.prev )
      wm->wlist.prev = wm->wlist.next = wdg;
    else {
      wdg->link.prev = wm->wlist.prev;
      wdg->link.prev->link.next = wm->wlist.prev = wdg;
    }
    wdg->id = id;
    wdg->r.width = w;  wdg->r.height = h;  wdg->r.x = x;  wdg->r.y = y;
    wdg->input = input;  wdg->redraw = redraw;
    wdg->data0 = data0;  wdg->data1 = data1;
    wdg->wm = wm;
    wdg->state = WDGSTATE_DEFAULT;
  }
  return wdg;
} /*NewWidget*/
```

26.2. Procedury przekazujące komunikaty

Zadaniem procedury przedstawionej na listingu 26.4 jest tworzenie uproszczonych informacji na temat komunikatów otrzymanych od systemu X Window, pochodzących od urządzeń wejściowych (myszy i klawiatury), co umożliwia pisanie prostszych procedur obsługi tych komunikatów. Takie informacje są przekazywane w parametrach procedury wejścia wihajstra i procedury przyjmującej polecenia wydane przez wihajstry (wskazywanej przez pole `callback` struktury typu `xwinmenu`). Informacja zawiera rodzaj komunikatu (`msg`), informację dodatkową (`key`) i współrzędne położenia kursora w oknie (`x`, `y`). Jeśli komunikat opisuje naciśnięcie lub zwolnienie przycisku myszy, to informacja dodatkowa określa, który to jest przycisk. Jeśli komunikat opisuje przesunięcie myszy, to informacja dodatkowa opisuje stan wszystkich przycisków. Jeśli został naciśnięty klawisz, to informacja dodatkowa podaje kod ASCII napisanego znaku, lub w przypadku klawisza specjalnego (strzałki, `F1` itp.) **symbol klawisza** (*KeySym*) przekazany w oryginalnym komunikacie. Makrodefinicje opisujące symbole klawiszy można znaleźć w pliku `/usr/include/X11/keysymdef.h`. Komunikaty inne niż pochodzące od urządzeń wejściowych otrzymują rodzaj `XWMSG_UNKNOWN`, ale aplikacja ma dostęp do oryginalnego komunikatu od systemu X Window, a dokładniej do zmiennej (wskazywanej przez pole `ev` struktury typu `xwinmenu`) opisującej ostatni otrzymany od systemu komunikat, który aplikacja właśnie przetwarza.

Listing 26.4. Procedura upraszczania komunikatów

C

```
#define XWMSG_UNKNOWN              1
#define XWMSG_ENTERING             2
#define XWMSG_LEAVING              3
#define XWMSG_BUTTON_PRESS         4
#define XWMSG_BUTTON_RELEASE       5
#define XWMSG_MOUSE_MOTION         6
#define XWMSG_KEY_PRESS            7
#define XWMSG_KEY_RELEASE          8
#define XWMSG_SPECIAL_KEY_PRESS    9
#define XWMSG_SPECIAL_KEY_RELEASE 10
#define XWMSG_CLIENT_MESSAGE      11

void TranslateEventMsg ( XEvent *ev, int *msg, int *key, int *x, int *y )
{
  char   chr;
  KeySym ks;

  switch ( ev->xany.type ) {
case ButtonPress:
    *msg = XWMSG_BUTTON_PRESS;
    *key = ev->xbutton.button;
    *x = ev->xbutton.x;  *y = ev->xbutton.y;
    break;
```

```
24: case ButtonRelease:
25:     *msg = XWMSG_BUTTON_RELEASE;
26:     *key = ev->xbutton.button;
27:     *x = ev->xbutton.x;  *y = ev->xbutton.y;
28:     break;
29: case MotionNotify:
30:     *msg = XWMSG_MOUSE_MOTION;
31:     *key = ev->xmotion.state;
32:     *x = ev->xmotion.x;  *y = ev->xmotion.y;
33:     break;
34: case KeyPress:
35:     *msg = XWMSG_KEY_PRESS;
36:     goto decode_key;
37: case KeyRelease:
38:     *msg = XWMSG_KEY_RELEASE;
39: decode_key:
40:     XLookupString ( &ev->xkey, &chr, 1, &ks, NULL );
41:     if ( !chr ) {  /* nie ASCII */
42:       *msg = ev->xany.type == KeyPress ?
43:                 XWMSG_SPECIAL_KEY_PRESS : XWMSG_SPECIAL_KEY_RELEASE;
44:       *key = ks;
45:     }
46:     else
47:       *key = chr;
48:     *x = ev->xkey.x;  *y = ev->xkey.y;
49:     break;
50: default:
51:     *msg = XWMSG_UNKNOWN;
52:     *x = *y = -1;
53:     break;
54:   }
55: } /*TranslateEventMsg*/
```

W odpowiedzi na komunikat `Expose` pokazana na listingu 26.5 procedura `WinMenuInput` rysuje wihajstry. Jeśli jest w użyciu kanwa X Window (pole `pixmap` ma wartość niezerową), to wihajstry są rysowane na niej, a następnie cały obraz z kanwy jest kopiowany do okna, przy czym rysowanie jest zbędne, jeśli pole `wm->changed` ma wartość `false`, co oznacza, że ostatnio wykonany obraz na kanwie jest aktualny. Procedura rysowania zawartości okna jest wywoływana zawsze, gdy kanwa nie jest używana (w oknie, którego zawartość ma rysować OpenGL). Komunikat `ConfigureNotify` powoduje zapamiętanie nowych wymiarów okna i utworzenie nowej kanwy, której wymiary są równe nowej szerokości i wysokości okna, po czym następuje wywołanie procedury `callback`, która może spowodować zmianę wielkości i rozmieszczenia wihajstrów w oknie. Potem do okna jest wysyłany (za pośrednictwem opisanej dalej procedury `PostMenuExposeEvent`) komunikat `Expose`, aby spowodować odświeżenie jego zawartości. Komunikat `ClientMessage` jest przesyłany od razu do procedury `callback`.

Komunikaty `EnterNotify` i `LeaveNotify`, otrzymywane, gdy kursor pojawia się w obszarze okna lub go opuszcza, są „tłumaczone" na komunikat o wejściu kursora do obszaru wihajstra lub o opuszczeniu tego obszaru. Zmienna `lastinput` jest wskaźnikiem wihajstra, do którego są kierowane komunikaty; jeśli kolejny komunikat wejściowy ma odbierać inny wihajster, to oba wihajstry są zawiadamiane o tej zmiane.

Komunikaty `GraphicsExpose` i `NoExpose`, dla porządku, są przekazywane aplikacji, ale może ona je ignorować.

Inne komunikaty są wstępnie dekodowane przez procedurę `TranslateEventMsg`. Jeśli pole `focus` nie ma wartości `NULL`, to komunikat jest przekazywany wskazywanemu przez to pole wihajstrowi. W przeciwnym razie lista wihajstrów jest przeglądana („od góry do dołu") w poszukiwaniu wihajstra, którego prostokąt zawiera punkt wskazywany przez kursor. Jeśli wihajster nie przetworzył komunikatu, to przeglądanie listy jest kontynuowane. Jeśli żaden wihajster nie przetworzył komunikatu, to jest on przesyłany do aplikacji, tj. do procedury

Listing 26.5. Procedura przesyłania komunikatów do wihajstrów

C

```
1: static xwidget *lastinput;
2:
3: char XYInside ( xwidget *wdg, int x, int y )
4: {
5:   return x >= wdg->r.x && x < wdg->r.x+wdg->r.width &&
6:          y >= wdg->r.y && y < wdg->r.y+wdg->r.height;
7: } /*XYInside*/
8:
9: void WinMenuInput ( xwinmenu *wm, XEvent *ev )
10: {
11:   int     msg, key;
12:   int     x, y;
13:   xwidget *wdg;
14:   char    inp;
15:   Window  root, child;
16:
17:   wm->ev = ev;
18:   switch ( ev->xany.type ) {
19: case Expose:
20:     if ( ev->xexpose.count == 0 ) {
21:       if ( wm->changed || !wm->pixmap ) {
22:         wm->redraw ( wm );
23:         wm->changed = wm->expose_sent = false;
24:       }
25:       if ( wm->pixmap )
26:         XCopyArea ( xdisplay, wm->pixmap, wm->window, xgc,
27:                     0, 0, wm->r.width, wm->r.height, 0, 0 );
28:     }
29:     return;
30: case ConfigureNotify:
```

```
31:     wm->r.x = ev->xconfigure.x;      wm->r.y = ev->xconfigure.y;
32:     wm->r.width = ev->xconfigure.width;
33:     wm->r.height = ev->xconfigure.height;
34:     if ( wm->pixmap ) {
35:       XFreePixmap ( xdisplay, wm->pixmap );
36:       wm->pixmap = XCreatePixmap ( xdisplay, wm->window,
37:                                    wm->r.width, wm->r.height, 24 );
38:     }
39:     wm->callback ( wm->empty, WDGMSG_RECONFIGURE, 0,
40:                    wm->r.width, wm->r.height );
41:     wm->changed = true;
42:     PostMenuExposeEvent ( wm );
43:     break;
44: case ClientMessage:
45:     wm->callback ( wm->wlist.next, XWMSG_CLIENT_MESSAGE,
46:                    ev->xclient.message_type, -1, -1 );
47:     break;
48: case EnterNotify:
49:     for ( wdg = wm->wlist.prev;  wdg;  wdg = wdg->link.prev )
50:       if ( XYInside ( wdg, ev->xcrossing.x, ev->xcrossing.y ) ) {
51:         wdg->input ( wdg, XWMSG_ENTERING, 0,
52:                      ev->xcrossing.x, ev->xcrossing.y );
53:         lastinput = wdg;
54:         break;
55:       }
56:     break;
57: case LeaveNotify:
58:     if ( lastinput ) {
59:       lastinput->input ( lastinput, XWMSG_LEAVING, 0,
60:                          ev->xcrossing.x, ev->xcrossing.y );
61:       lastinput = NULL;
62:     }
63:     break;
64: case GraphicsExpose:
65: case NoExpose:
66:     wm->callback ( wm->wlist.next, XWMSG_UNKNOWN, 0, 0, 0 );
67:     break;
68: default:
69:     inp = found = false;
70:     TranslateEventMsg ( ev, &msg, &key, &x, &y );
71:     if ( (wdg = wm->focus) ) {
72:       inp = wdg->input ( wdg, msg, key, x, y );
73:       if ( !wm->focus && !XYInside ( wdg, x, y ) ) {
74:         wdg->input ( wdg, XWMSG_LEAVING, 0, x, y );
75:         lastinput = NULL;
76:       }
77:     }
```

```
78:     else {
79:       for ( wdg = wm->wlist.prev;  wdg;  wdg = wdg->link.prev ) {
80:         if ( XYInside ( wdg, x, y ) ) {
81:           found = true;
82:           if ( wdg != lastinput ) {
83:             if ( lastinput )
84:               lastinput->input ( lastinput, XWMSG_LEAVING, 0, x, y );
85:             wdg->input ( wdg, XWMSG_ENTERING, 0, x, y );
86:             lastinput = wdg;
87:           }
88:           if ( (inp = wdg->input ( wdg, msg, key, x, y )) )
89:             break;
90:         }
91:       }
92:       if ( !found && lastinput ) {
93:         lastinput->input ( lastinput, XWMSG_LEAVING, 0, x, y );
94:         lastinput = NULL;
95:       }
96:     }
97:     if ( !inp )
98:       wm->callback ( wm->wlist.next, msg, key, x, y );
99:     if ( wm->changed )
100:      PostMenuExposeEvent ( wm );
101:  }
102:  XQueryPointer ( xdisplay, wm->window, &root, &child,
103:                  &x, &y, &wm->prevx, &wm->prevy, &wm->prevmask );
104:  return;
105: } /*WinMenuInput*/
106:
107: void PostMenuExposeEvent ( xwinmenu *wm )
108: {
109:  if ( !wm->expose_sent ) {
110:    PostExposeEvent ( wm->window, wm->r.width, wm->r.height );
111:    wm->expose_sent = true;
112:  }
113: } /*PostMenuExposeEvent*/
114:
115: void GrabInput ( xwidget *wdg )
116: {
117:  wdg->wm->focus = wdg;
118: } /*GrabInput*/
119:
120: void UngrabInput ( xwidget *wdg )
121: {
122:  wdg->wm->focus = NULL;
123: } /*UngrabInput*/
```