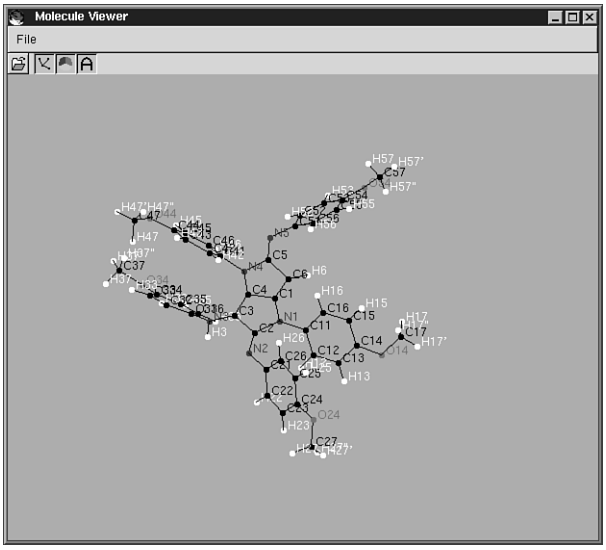
Rozdział 12

Przeglądarka cząsteczek

Wyświetlanie informacji przy użyciu trójwymiarowej grafiki jest zawsze interesującym zagadnieniem. W rozdziale wykorzystamy GDK (*Graphics Drawing Kit*) do napisania przeglądarki plików .pdb, która pozwoli oglądać cząsteczki na kontrolce obszaru rysunkowego (patrz rysunek 12.1). W programie wykorzystamy arytmetykę trójwymiarową; jeśli czytelnicy nie czują się pewnie w tym temacie, mogą po prostu zapoznać się z ogólnymi koncepcjami, ignorując mnożenie macierzy. Jeśli zaś chcieliby dowiedzieć się więcej o obliczeniach trójwymiarowych, mogą sięgnąć po jedną z wielu książek traktujących o grafice komputerowej.



Rysunek 12.1. Przeglądarka cząsteczek.

Oprócz samego wyświetlania cząsteczki, aplikacja będzie dysponowała możliwością jej obracania. Użytkownik będzie także mógł wyświetlać cząsteczkę w kolorze i z tekstem, opisującym poszczególne atomy. Aby

obracanie cząsteczki nie powodowało zakłóceń na ekranie, wykorzystamy podwójne buforowanie.

Format pliku

Aby wyświetlać pliki .pdb, musimy umieć je odczytać, a także zrozumieć format opisu cząsteczek. Format pliku .pdb jest dość skomplikowany, ponieważ zaprojektowano go do opisu złożonych cząsteczek. Na szczęście możemy zignorować większość danych, przechowywanych w pliku .pdb. Interesują nas tylko nazwy atomów, ich pozycje, oraz wiązania pomiędzy nimi. Pliki .pdb posiadają dwie części: pierwsza opisuje atomy, a druga (opcjonalna) wiązania pomiędzy atomami. Pełen opis atomów jest dość skomplikowany:

Format rekordu

KOLUMNY	TYP DANYCH	POLE	OPIS
1 -6	Nazwa rekordu	"ATOM "	
07 - 11	Liczba całkowita	serial	Numer kolejny atomu
13 -16	Atom	name	Nazwa atomu
17	Znak	altLoc	Wskaźnik alternatywnego położenia
18 - 20	Nazwa pozostałości	resName	Nazwa pozostałości
22	Znak	chainID	Identyfikator łańcucha
23 - 26	Liczba całkowita	resSeq	Numer kolejny pozostałości
27	Znak	iCode	Kod wstawiania pozostałości
31 - 38	L. rzeczywista (8.3)	x	Ortogonalna współrzędna X w angstremach.
39 - 46	L. rzeczywista (8.3)	y	Ortogonalna współrzędna Y w angstremach.
47 - 54	L. rzeczywista (8.3)	z	Ortogonalna współrzędna Z w angstremach.
55 - 60	L. rzeczywista (6.3)	occupancy	Zajętość
61 - 66	L. rzeczywista (6.3)	tempFactor	Czynnik temperaturowy
73 - 76	LString(4)	segID	Identyfikator segmentu, wyrównany do lewej
77 - 78	LString(2)	element	Symbol pierwiastka, wyrównany do prawej
79 - 80	LString(2)	charge	Ładunek atomu

Najważniejszymi informacjami są nazwa rekordu, „Atom" i współrzędne przestrzenne (x, y, z).

Druga część pliku .pdb jest opcjonalna i zawiera informacje, potrzebne do wyświetlenia wiązań atomowych. Linie CONECT opisują atom i wszystkie jego wiązania z innymi atomami. Pierwsza liczba po słowie CONECT jest numerem kolejnym atomu w cząsteczce, a pozostałe są numerami kolejnymi atomów, z którymi jest związany. Przykładowy plik .pdb z danymi o wiązaniach wygląda następująco:

```
ATOM      1 O11      1       2.227   3.257   9.904
ATOM      2 O12      1       4.387   2.116  10.202
ATOM      3 O13      1       3.470   2.116   8.123
ATOM      4 N1       1       1.032   3.192  13.498
ATOM      5 C1       1       1.135   4.580  13.046
ATOM      6 C2       1       1.192   2.116  12.739
ATOM      7 C3       1       0.762   2.796  14.797
ATOM      8 C3'      1       0.507   3.783  15.877
ATOM      9 C11      1       2.759   4.526   9.626
ATOM     10 C12      1       4.372   2.116  11.603
ATOM     11 C13      1       2.451   2.116   7.171
ATOM     12 B1       1       3.078   2.116   9.530
ATOM     13 N1*      1       1.032   1.041  13.498
ATOM     14 C3*      1       0.762   1.437  14.797
ATOM     15 O11*     1       2.227   0.976   9.904
ATOM     16 C1*      1       1.135  -0.347  13.046
ATOM     17 C3'*     1       0.507   0.450  15.877
ATOM     18 C11*     1       2.759  -0.293   9.626
TER
CONECT    1    9   12
CONECT    2   10   12
CONECT    3   11   12
CONECT    4    5    6    7
CONECT    5    4
CONECT    6    4   13
CONECT    7    4    8   14
CONECT    8    7
CONECT    9    1
CONECT   10    2
CONECT   11    3
CONECT   12    1    2    3   15
CONECT   13    6   14   16
```

```
CONECT   14      7     13     17
CONECT   15     12
CONECT   16     13
CONECT   17     14
```

Przyjrzyjmy się pierwszej linii pliku. Zaczyna się od słowa ATOM, co oznacza, że linia opisuje atom. Następny znak, 1, jest indeksem (numerem kolejnym) atomu. Atomy powinny być uporządkowane. O11 jest nazwą atomu - w tym przypadku O oznacza prawdopodobnie tlen. Położenie atomu określają trzy ostatnie liczby w linii. Są to współrzędne x, y i z atomu w cząsteczce.

```
ATOM     1 O11        1      2.227   3.257   9.904
```

Opis atomów kończy się na linii o etykiecie TER, po której następują wiązania. Poszczególne linie zaczynają się od słowa CONECT, i zawierają numer atomu oraz numery wszystkich atomów, z którymi łączą go wiązania. Pierwsza linia opisu wiązań wygląda następująco:

```
CONECT   1    9   12
```

Oznacza ona, że atom 1 ma wiązania z atomami 9 i 12. Atomy niekoniecznie muszą mieć wiązania, ale zazwyczaj je mają.

Struktury danych

Struktura danych atomu (typAtom) przechowuje nazwę atomu, dwa zbiory współrzędnych oraz listę atomów, z którymi atom jest połączony wiązaniami. Struktura posiada dwa zbiory współrzędnych, ponieważ przechowujemy w niej współrzędne pierwotne oraz współrzędne przekształcone. Potrzebujemy przekształconych współrzędnych dlatego, że cząsteczka będzie obracana wokół osi-a w miarę obrotu będzie zmieniać się położenie atomów w trójwymiarowej przestrzeni. Określają je przekształcone współrzędne, obliczane od nowa po każdym przesunięciu cząsteczki. Przekształcone współrzędne odzwierciedlają miejsce, w którym atom zostanie narysowany. Ponieważ każdy atom może mieć wiązania z jednym lub wieloma atomami, struktura typAtom przechowuje wszystkie wiązania w łączonej liście GSList.

```
typedef struct {
   char *szNazwa;     // --- Nazwa atomu
   double x;          // --- Pierwotne współrzędne
   double y;          //
   double z;          //
   double tx;         // --- Współrzędne po translacji
```

```
    double ty;          //
    double tz;          //
    GSList *listaWiazan;    // --- Atomy, z którymi atom ma wiązania
} typAtom;
```

Wiązania opisują związki, które umożliwiają połączenie atomów liniami. Dodatkowy znacznik przyspiesza rysowanie-wiązanie powinno być rysowane tylko raz dla każdej pary atomów.

```
typedef struct {
    typAtom *atom1;    /* --- Pierwszy atom we wiązaniu --- */
    typAtom *atom2;    /* --- Drugi atom we wiązaniu--- */
    int bNarysowane;   /* --- narysowane--- */
} typWiazanie;
```

Atomy i wiązania są przechowywane w statycznych tablicach (autor nad tym ubolewa, ale dynamiczne przydzielanie pamięci oznaczałoby znacznie więcej pracy). Tablice ułatwiają rysowanie atomów.

Rysowanie w trzech wymiarach

Ponieważ cząsteczka ma strukturę trójwymiarową, powinniśmy narysować ją tak, aby stworzyć iluzję trójwymiarowości. Podczas wyświetlania cząsteczek należy unikać oczywistych błędów, jak na przykład rysowania atomów bez uwzględnienia ich położenia w osi z. Spowodowałoby to, że atomy znajdujące się dalej znalazłyby na ekranie przed atomami położonymi bliżej oglądającego. Można uporać się z tym problemem, sortując atomy według przekształconej współrzędnej z. Jeśli narysujemy najpierw odległe atomy, mamy gwarancję, że atomy znajdujące się najbliżej oglądającego pozostaną na szczycie, i nie zostaną przykryte przez inne. Algorytm rysowania wyświetla najpierw najdalsze atomy, a potem przykrywa je bliższymi.

Kod źródłowy

Cały kod służący do rysowania cząsteczek (z wyjątkiem procedur matematycznych operujących na macierzach) znajduje się w pliku czasteczka.c. Umieszczono tutaj procedury umożliwiające wczytywanie plików, rysowanie cząsteczek i wykonywanie innych operacji na cząsteczkach; tego rodzaju modularyzacja kodu pozwala na łatwe wykorzystanie go w innej aplikacji. Ponieważ większość nowych funkcji znajduje się w pliku

czasteczka.c, przejrzymy teraz jego zawartość i omówimy czynności wykonywane przez poszczególne procedury.

WczytajCzasteczke

Główna procedura wczytująca plik musi zanalizować dwie jego części. Najpierw wczytuje atomy, pobierając z pliku ich nazwy oraz współrzędne. Następnie wczytuje wiązania, umieszczając je zarazem w strukturze danych atomu jak i w tablicy wiązań. Funkcja ta jest wywoływana podczas uruchamiania aplikacji, i wczytuje domyślny plik (czasteczka.pdb). Można ją także wywołać wybierając opcję menu Plik/Nowy, co umożliwia wczytanie innych plików .pdb. Kiedy wczytywany jest nowy plik, trzeba wyzerować niektóre dane (na przykład macierze). Procedura Wczytaj Czasteczke wymusza także przerysowanie ekranu po wczytaniu nowych danych.

```
/*
 * WczytajCzasteczke
 *
 * Wczytuje cząsteczkę z pliku .pdb o podanej nazwie.
 * Zachowuje informacje w zdefiniowanych strukturach
 * danych.
 */
void WczytajCzasteczke (char *sNazwaPliku)
{
   FILE *fp;
   char bufor[120];
   float x, y, z;
   int nIndeks1;
   int nIndeks2;
   char szNazwa[120];
   char *sTmcz;
   typAtom *atom;
   char szTmcz[20];
   Inicjuj3d ();

   nAtomy = 0;
   nWiazania = 0;

   /* --- Przed wczytywaniem pliku zerujemy macierze --- */
   if (macierz) {
      jednostka (macierz);
      jednostka (amacierz);
      jednostka (tmacierz);
```

```
}

nPromienCzasteczki = 2;

/* --- Otwieramy plik do odczytu --- */
fp = fopen (sNazwaPliku, "r");

/* --- Wczytujemy linię z pliku --- */
while (fgets (bufor, sizeof (bufor), fp)) {

   /* --- Jeśli jest to atom --- */
   if (strncmp (bufor, "ATOM", 4) == 0) {

      /* --- Wczytujemy dane atomu, znając
       *     strukturę pliku .pdb
       */
      strncpy (szNazwa, &bufor[12], 4);
      szNazwa[4] = 0;
      strncpy (szTmcz, &bufor[30], 8);
      szTmcz[8] = 0;

      x = atof (szTmcz);
      strncpy (szTmcz, &bufor[38], 8);
      szTmcz[8] = 0;

      y = atof (szTmcz);
      strncpy (szTmcz, &bufor[46], 8);
      szTmcz[8] = 0;
      z = atof (szTmcz);

      /* --- Indeksy atomów zaczynają się od 1 --- */
      nAtomy++;

      /* --- Wypełniamy strukturę danych --- */
      atom = &listaatomow[nAtomy];
      atom->x = x;
      atom->y = y;
      atom->z = z;
      atom->szNazwa = strdup (szNazwa);
      atom->listaWiazan = NULL;
      sortindeks[nAtomy-1] = nAtomy;

   /* --- Czy linia opisuje wiązanie? --- */
   } else if (strncmp (bufor, "CONECT", 6) == 0) {
```

```
            /* --- Pobieramy pierwszy atom wiązania --- */
            sTmcz = PobierzNastepnaWartosc (&bufor[6], &nIndeks1);

            /* --- Pobieramy następne atomy wiązania --- */
            while (sTmcz = PobierzNastepnaWartosc (sTmcz, &nIndeks2)) {

               /* --- Wiązanie jest od nIndeks1 do nIndeks2 --- */
               listawiazan[nWiazania].atom1 = &listaatomow[nIndeks1];
               listawiazan[nWiazania].atom2 = &listaatomow[nIndeks2];

               /* --- Oczywiście atom musi wiedzieć,
                *     jakie tworzy wiązania...
                */
               listaatomow[nIndeks1].listaWiazan =
                  g_slist_append(listaatomow[nIndeks1].listaWiazan,
                  &listawiazan[nWiazania]);

               /* --- ...i drugi atom również --- */
               listaatomow[nIndeks2].listaWiazan =
                  g_slist_append (listaatomow[nIndeks2].listaWiazan,
                  &listawiazan[nWiazania]);

               /* --- Zwiększamy liczbę wiązań --- */
               nWiazania++;
            }
         }
      }

      /* --- Znajdujemy prostopadłościan ograniczający --- */
      ZnajdzPO ();

      /* --- Sortujemy atomy --- */
      SortujAtomy (listaatomow, sortindeks);
      OdswiezCzasteczke ();
   }
```

ZnajdzPO

Prostopadłościan ograniczający jest to najmniejszy prostopadłościan, który mógłby pomieścić wyświetlaną cząsteczkę. Procedura ZnajdzPO oblicza wymiary prostopadłościanu ograniczającego i pozwala przeskalować obiekt tak, aby zmieścił się na ekranie.

```
/*
 * ZnajdzPO
 *
 * Znajduje najmniejszy prostopadłościan, który mógłby
 * pomieścić wszystkie atomy w cz±steczce.
 */
void ZnajdzPO ()
{
   int       i;
   typAtom    *atom;

   /* --- Najpierw prostopadłościan zawiera pojedynczy
          atom. Atomy zaczynają się od 1       --- */
   atom = &listaatomow[1];

   xmin = atom->x;
   xmax = atom->x;

   ymin = atom->y;
   ymax = atom->y;

   zmin = atom->z;
   zmax = atom->z;

   /* --- Teraz dodajemy całą resztę --- */
   for (i = 2; i <= nAtomy; i++) {

      atom = &listaatomow[i];
      if (atom->x < xmin) xmin = atom->x;
      if (atom->x > xmax) xmax = atom->x;
      if (atom->y < ymin) ymin = atom->y;
      if (atom->y > ymax) ymax = atom->y;
      if (atom->z < zmin) zmin = atom->z;
      if (atom->z > zmax) zmax = atom->z;
   }

   /* --- ...i mamy nasz prostopadłościan --- */
}
```

Sortowanie atomów

Aby uzyskać poprawny, trójwymiarowy obraz, trzeba narysować cząsteczkę zaczynając od najdalszego atomu, a kończąc na najbliższym. Musimy więc posortować atomy według ich przekształconej współrzędnej z. Nie możemy wykorzystać rzeczywistej współrzędnej z, ponieważ użytkownik mógł obrócić cząsteczkę, i ogląda ją w takiej właśnie obróconej postaci. Po przeprowadzeniu sortowania najbliższe atomy zostaną narysowane na wierzchu, tworząc poprawny (choć iluzoryczny) trójwymiarowy obraz. Zamiast sortować same atomy, sortujemy ich indeksy, ponieważ przesuwanie liczb całkowitych w obrębie tablicy jest dużo szybsze, niż przemieszczanie całych struktur.

```
/*
 * SortujAtomy
 *
 * Wywołuje funkcje sortującą
 */
void SortujAtomy (typAtom *listaatomow, int *sortindeks)
{
   QSortujAtomy (listaatomow, sortindeks, 0, nAtomy-1);
}

/*
 * QSortujAtomy
 *
 * Sortowanie typu quicksort wszystkich atomów w cząsteczce.
 *
 * Uwaga: Zamiast sortować samą listę atomów, sortujemy
 * tablicę indeksów (sortindeks). Operowanie na liczbach
 * całkowitych jest szybsze, niż przemieszczanie struktur -
 * zwłaszcza, że musimy robić to w czasie rzeczywistym i
 * bardzo często.
 *
 * listaatomow - lista atomów do posortowania
 * sortlist - tablica indeksów
 */
void QSortujAtomy (typAtom *listaatomow, int *sortindeks,
                   int dol0, int gora0)
{
   int    nTmcz;
   int    dol = dol0;
   int    gora = gora0;
```

```
    int    srodek;

    if (gora0 > dol0) {

       srodek = listaatomow[sortindeks[(dol0 + gora0) / 2]].tz;

       while (dol <= gora) {

          while (dol < gora0 &&
                listaatomow[sortindeks[dol]].tz < srodek) {
             dol++;
          }

          while (gora > dol0 &&
                listaatomow[sortindeks[gora]].tz > srodek) {
             gora--;
          }

          if (dol <= gora) {

             nTmcz = sortindeks[dol];
             sortindeks[dol] = sortindeks[gora];
             sortindeks[gora] = nTmcz;
             dol++;
             gora--;
          }
       }
       if (dol0 < gora) QSortujAtomy (listaatomow, sortindeks,
                                dol0, gora);
       if (dol < gora0) QSortujAtomy (listaatomow, sortindeks,
                                dol, gora0);
    }
}
```

Transformuj Punkty

Procedura TransformujPunkty przeprowadza obliczenia macierzowe na wszystkich atomach cząsteczki. W rezultacie otrzymujemy cząsteczkę obróconą i przeskalowaną do rozmiarów ekranu. Właściwe obliczenia odbywają się w innej funkcji.

```
/*
 * TransformujPunkty
```

```
 *
 * Obraca atomy zgodnie z ich aktualną pozycją na
 * ekranie, aby można je było narysować.
 *
 * macierz - Macierz wykorzystywana do obliczenia
 *           nowych pozycji atomów.
 */
void TransformujPunkty (typMacierz3D *macierz)
{
   int    i;

   for (i = 1; i <= nAtomy; i++) {

      Transformuj (macierz, &listaatomow [i]);
   }
}
```

rysuj

Funkcja rysuj w rzeczywistości wcale nie wykonuje rysowania, ale jest wywoływana przed każdym przerysowaniem cząsteczki. Właściwe rysowanie wykonuje funkcja RysujCzasteczke, ale zanim będzie można ją wywołać, należy przypisać atomy do macierzy, przy pomocy której będzie można obliczyć obrót cząsteczki. Dopiero wtedy można narysować cząsteczkę.

```
/*
 * rysuj
 *
 * rysuje cząsteczkę na ekranie, a właściwie w drugoplanowym
 * buforze. Większość kodu w procedurze zajmuje się obliczeniami
 * na macierzach, dopiero potem wywoływana jest funkcja, która
 * dokonuje rzeczywistego rysowania.
 */
void rysuj (typGrafika *g)
{
   double xczyn;
   double f1;
   double f2;

   /* --- Jaki zakres? (delta x, delta y, delta z) --- */
   double xw = xmax - xmin;
```

```
    double yw = ymax - ymin;
    double zw = zmax - zmin;

    /* --- Upewniamy się, że zasoby są przydzielone --- */
    Inicjuj3d ();

    /* --- Obliczamy czynnik skalowania cząsteczki --- */
    if (yw > xw) xw = yw;
    if (zw > xw) xw = zw;
    f1 = nSzerEkranu / xw;
    f2 = nWysokEkranu / xw;
    xczyn = .7 * (f1 < f2 ? f1 : f2);

    /* --- Najpierw czynimy macierz macierzą jednostkową --- */
    jednostka (macierz);

    /* --- Przestawiamy macierz wokół środka translacji.
     *     Dzięki temu cząsteczka będzie wyśrodkowana wokół osi,
     *     biegnących przez środek prostopadłościanu
     *     ograniczającego cząsteczkę.
     */
    przestaw (macierz, -(xmin + xmax) / 2,
                       -(ymin + ymax) / 2,
                       -(zmin + zmax) / 2);

    /* --- Obracamy obraz wokół osi. amacierz określa,
           w jakim stopniu należy obrócić cząsteczkę
     */
    mnoz (macierz, amacierz);

    /* --- Skalujemy cząsteczkę na podstawie szerokości ekranu --- */
    skaluj3 (macierz, xczyn, -xczyn, 16 * xczyn / nSzerEkranu);

    /* --- Przesuwamy cząsteczkę na podstawie szerokości
           i wysokości ekranu  --- */
    przestaw (macierz, nSzerEkranu / 2, nWysokEkranu / 2, 10);

    /* --- Obliczamy nowe położenia wszystkich punktów --- */
    TransformujPunkty (macierz);

    /* --- Rysujemy cząsteczkę na podstawie przekształconych
           współrzędnych  --- */
    RysujCzasteczke (g);
}
```

Rysowanie wiązań

Można narysować wiązania podczas rysowania każdego atomu, ponieważ wiemy, które wiązanie należy do którego atomu. Metoda taka spowodowałaby jednak, że każde wiązanie byłoby rysowane dwa razy, co jest niepożądane. Można także rysować wiązania albo przed, albo po narysowaniu atomów, ale w obu przypadkach zrujnowalibyśmy iluzję trójwymiarowości, którą usiłujemy nadać wyświetlanej cząsteczce. Najlepszym rozwiązaniem jest narysowanie wiązania tylko podczas rysowania odleglejszego atomu. Dzięki temu wiązanie zostanie narysowane tylko raz i nie będzie przykrywać atomów znajdujących się bliżej oglądającego.

Cząsteczka jest rysowana w funkcji RysujCzasteczke. Atomy są rysowane w kolejności określonej przez tablicę sortindeks, która przechowuje indeksy posortowanych atomów. Dzięki użyciu tej tablicy kolejność rysowania atomów odpowiada ich odległości od oglądającego. W znaczniku bNarysowane zapamiętujemy, czy wiązanie zostało już narysowane; jeśli tak, nie będzie rysowane ponownie (nie ma sensu tracić czasu procesora). Wiązania są rysowane wraz z odpowiednimi atomami, aby zapewnić dobry trójwymiarowy efekt.

```
/*
 * RysujCzasteczke
 *
 * Rysuje cząsteczkę
 */
void RysujCzasteczke (typGrafika *g)
{
   int nIndeks;
   int nSrednica;
   typWiazanie *wiazanie;
   GSList *lista;
   typAtom *atom;
   int i;
   GdkGC *pioro;

   /* --- Upewniamy się, że wszystko jest gotowe --- */
   Inicjuj3d ();

   /* --- Sortujemy atomy --- */
   SortujAtomy (listaatomow, sortindeks);

   /* --- Pobieramy średnicę cząsteczki --- */
   nSrednica = nPromienCzasteczki + nPromienCzasteczki;
```

```
/* --- Jeśli pokazujemy wiązania... --- */
if (PokazLinie()) {

   /* --- Czyścimy znacznik, który wskazuje, że
    *     to wiązanie zostało już narysowane
    */
   for (i = 0; i < nWiazania; i++) {

      listawiazan[i].bNarysowane = FALSE;
   }
}

/* --- Wyświetlamy wszystkie atomy na liście --- */
for (i = 0; i < nAtomy; i++) {

   /* --- Używamy listy posortowanej - rysujemy od
    *     najdalszego do najbliższego atomu.
    */
   nIndeks = sortindeks[i];

   /* --- Pobieramy atom, wskazywany przez indeks --- */
   atom = &listaatomow[nIndeks];

   /* --- Czy ten atom ma swój kolor? --- */
   pioro = PobierzKolorAtomu (atom);

   /* --- Rysujemy koło w kolorze atomu --- */
   gdk_draw_arc (g->piksmapa, pioro, TRUE,
            atom->tx - 3, atom->ty - 3, 7, 7, 0, 360 * 64);

   /* --- Jeśli włączono pokazywanie nazw... --- */
   if (PokazEtykiety ()) {

      /* --- Wyświetlamy nazwę atomu --- */
      if (atom->szNazwa) {
         gdk_draw_string (g->piksmapa, czcionka,
                  pioro, atom->tx + 5, atom->ty,
                  atom->szNazwa);
      }
   }

   /* --- Jeśli włączono pokazywanie wiązań... --- */
   if (PokazLinie()) {
```

```
            /* --- Rysujemy wszystkie wiązania atomu --- */
            for (lista = atom->listaWiazan; lista;
                lista = lista->next) {

                /* --- Pobieramy wiązanie z listy --- */
                wiazanie = (typWiazanie *) lista->data;

                /* --- Jeśli jeszcze nie było rysowane... --- */
                if (wiazanie->bNarysowane == FALSE) {

                    /* --- Rysujemy wiązanie (linię) --- */
                    RysujWiazanie (g, wiazanie->atom1,
                            wiazanie->atom2);

                    /* --- Zaznaczamy wiązanie jako narysowane --- */
                    wiazanie->bNarysowane = TRUE;
                }
            }
        }
    }

}
```

Kolory atomów

Atomy są wyświetlane w kolorach, jeśli wciśnięty jest odpowiedni przycisk na pasku narzędziowym. Jeśli przycisk nie jest wciśnięty, wszystkie atomy są rysowane na czarno. W przeciwnym przypadku atomy są rysowane w kolorze określonym na podstawie nazwy atomu w pliku. Jeśli na przykład nazwa atomu zaczyna się od N, jest to prawdopodobnie atom azotu i zostanie narysowany na niebiesko (procedura zwróci kontekst GdkGC z niebieskim kolorem).

```
/*
 * PobierzKolorAtomu
 *
 * Zwraca kolor atomu na podstawie jego nazwy.
 *
 * Cząsteczki są rysowane albo na czarno, albo w
 * kolorze, zależnie od ustawienia przycisku na
 * pasku narzędziowym.
 *
 * Jeśli przycisk jest włączony...
```

```
 * Jeśli nazwa zaczyna się od C (carbonium, węgiel),
 * rysujemy atom na czarno. Jeśli zaczyna się od
 * N (nitrogenium, azot), rysujemy go na niebiesko.
 * Jeśli zaczyna się od O (oxygenium, tlen), rysujemy
 * go na czerwono itd. Wszystkie inne atomy są
 * rysowane w ciemnoszarym kolorze.
 *
 * atom - atom, którego kolor należy określić
 */
GdkGC *PobierzKolorAtomu (typAtom *atom)
{
   char szNazwa[10];

   /* --- Nie ma nazwy --- */
   if (atom->szNazwa == NULL) {
      return (pioroCzarne);
   }

   /* --- Nie pokazujemy kolorów --- */
   if (!PokazRGB ()) {
      return (pioroCzarne);
   }

   PobierzNazweAtomu (szNazwa, atom);

   if (!strcmp (szNazwa, "CL")) {
       return (pioroZielone);
   } else if (!strcmp (szNazwa, "C")) {
       return (pioroCzarne);
   } else if (!strcmp (szNazwa, "S")) {
       return (pioroZolte);
   } else if (!strcmp (szNazwa, "P")) {
       return (pioroPomaranczowe);
   } else if (!strcmp (szNazwa, "N")) {
       return (pioroNiebieskie);
   } else if (!strcmp (szNazwa, "O")) {
       return (pioroCzerwone);
   } else if (!strcmp (szNazwa, "H")) {
       return (pioroBiale);
   } else {
       return (pioroCiemnoszare);
   }
```

```
}
```

configure_event

Funkcja configure_event inicjuje struktury. Ponieważ wywoływana jest w razie tworzenia albo zmiany rozmiarów kontrolki obszaru rysunkowego, ustawia ona rozmiar drugoplanowej piksmapy na rozmiar kontrolki.

```
/*
 * Tworzymy nową, drugoplanową piksmapę o właściwych rozmiarach
 */
static gint configure_event (GtkWidget *kontrolka,
                             GdkEventConfigure *zdarzenie)
{
   /* --- Struktura nie istnieje? --- */
   if (g == NULL) {

      /* --- Tworzymy ją --- */
      g = NowaGrafika ();
   }

   /* --- Piksmapa istnieje?  --- */
   if (g->piksmapa) {

      /* --- Zwalniamy ją --- */
      gdk_pixmap_unref (g->piksmapa);
   }

   /* --- Tworzymy piksmapę --- */
   g->piksmapa = gdk_pixmap_new (kontrolka->window,
                                 kontrolka->allocation.width,
                                 kontrolka->allocation.height,
                                 -1);

   /* --- Pobieramy szerokość i wysokość, żeby wyczyścić ekran --- */
   nSzerEkranu = kontrolka->allocation.width;
   nWysokEkranu = kontrolka->allocation.height;

   /* --- Czyścimy ekran --- */
   gdk_draw_rectangle (g->piksmapa,
                       kontrolka->style->white_gc,
                       TRUE,
                       0, 0,
```

```
                kontrolka->allocation.width,
                kontrolka->allocation.height);

    /* --- Przerysowujemy cząsteczkę --- */
    OdswiezCzasteczke ();

    return TRUE;
}
```

expose_event

Zdarzenie expose_event zachodzi wtedy, kiedy obszar ekranu wymaga uaktualnienia. W przypadku naszej aplikacji funkcja expose_event po prostu kopiuje drugoplanową piksmapę do kontrolki obszaru rysunkowego. Dlatego jest bardzo szybka.

```
/*
 * expose_event
 *
 * Wywoływana wtedy, kiedy obszar ekranu został odsłonięty
 * i musimy go przerysować. Obszar jest kopiowany z
 * drugoplanowego bufora.
 */
static gint expose_event (GtkWidget *kontrolka,
                GdkEventExpose *zdarzenie)
{

  gdk_draw_pixmap(kontrolka->window,
        kontrolka->style->fg_gc[GTK_WIDGET_STATE (kontrolka)],
        g->piksmapa,
        zdarzenie->area.x -1, zdarzenie->area.y -1 ,
        zdarzenie->area.x -1, zdarzenie->area.y -1 ,
        zdarzenie->area.width -1, zdarzenie->area.height - 1);

    return FALSE;
}
```

Odśwież Cząsteczkę

Funkcja ta wywoływana jest wtedy, kiedy z jakichś powodów należy przerysować cząsteczkę. Czyścimy drugoplanową piksmapę, rysujemy

na niej cząsteczkę, a następnie wywołujemy funkcję expose_event, która kopiuje drugoplanową piksmapę na ekran.

```
/*
 * OdswiezCzasteczke
 *
 * Wywoływana wtedy, kiedy użytkownik przesuwa cząsteczkę
 * przy pomocy myszy. Powoduje to przerysowanie
 * cząsteczki.
 */
void OdswiezCzasteczke ()
{
   GdkRectangle  uakt_prostokat;

   Inicjuj3d ();

   /* --- czyścimy piksmapę --- */
   gdk_draw_rectangle (g->piksmapa,
                       pioroSzare,
               TRUE,
               0, 0,
               obszar_rys->allocation.width,
               obszar_rys->allocation.height);

   /* --- Rysujemy cząsteczkę na drugim planie --- */
   rysuj (g);

   /* --- Odświeżamy cały ekran --- */
   uakt_prostokat.x = 0;
   uakt_prostokat.y = 0;
   uakt_prostokat.width = obszar_rys->allocation.width;
   uakt_prostokat.height = obszar_rys->allocation.height;

   /* --- Wywołujemy funkcję expose_event, która
    *     kopiuje drugoplanowy bufor do kontrolki
    */
   gtk_widget_draw (obszar_rys, &uakt_prostokat);
}
```

Cząsteczka jest rysowana na drugoplanowej piksmapie za każdym razem, kiedy użytkownik przesunie ją przy pomocy myszy. Istnieje niewielka różnica pomiędzy techniką używaną tutaj, a tą z aplikacji zegara (patrz rozdział 10, „GDK"): tam przerysowywanie zegara odbywało się

co sekundę i było powodowane przez czasomierz; tutaj cząsteczka jest rysowana na drugoplanowej piksmapie wtedy, kiedy użytkownik przeciągnie ją myszą.

Tworzenie obszaru rysunkowego

Funkcja UtworzObszarRysunkowy jest wywoływana z pliku interfejs.c, który tworzy główne okno aplikacji. Pojedynczy blok kodu obejmuje wszystkie funkcje, które są niezbędne do stworzenia obszaru, na którym będą wyświetlane cząsteczki i nie ma żadnego wpływu na resztę aplikacji. Można wstawić tę funkcję do dowolnego programu bez żadnych zmian. Obszar rysunkowy odbiera sygnały expose_event, aby można było przerysować kontrolkę, oraz sygnały configure_event, dzięki czemu obszar rysunkowy będzie odpowiednio skonfigurowany i gotowy do rysowania. Musimy także sprawdzać sygnał motion_notify_event, aby wiedzieć, że użytkownik przeciąga cząsteczkę przy pomocy myszy. Nie możemy jednak bezpośrednio wykorzystać sygnału motion_notify_event w funkcji gtk_signal_connect. W GTK+ niskopoziomowe zdarzenia GDK nie są wysyłane do aplikacji, o ile jawnie tego nie zażądamy. Posłużymy się funkcją gtk_widget_set_events, której można przekazać jedną z poniższych wartości:

- GDK_EXPOSURE_MASK
- GDK_POINTER_MOTION_MASK
- GDK_POINTER_MOTION_HINT_MASK
- GDK_BUTTON_MOTION_MASK
- GDK_BUTTON1_MOTION_MASK
- GDK_BUTTON2_MOTION_MASK
- GDK_BUTTON3_MOTION_MASK
- GDK_BUTTON_PRESS_MASK
- GDK_BUTTON_RELEASE_MASK
- GDK_KEY_PRESS_MASK
- GDK_KEY_RELEASE_MASK
- GDK_ENTER_NOTIFY_MASK
- GDK_LEAVE_NOTIFY_MASK
- GDK_FOCUS_CHANGE_MASK
- GDK_STRUCTURE_MASK

- GDK_PROPERTY_CHANGE_MASK
- GDK_PROXIMITY_IN_MASK
- GDK_PROXIMITY_OUT_MASK

W naszej aplikacji będziemy potrzebować GDK_POINTER_MOTION_MASK, który informuje, że przesunął się wskaźnik myszy. Istnieje jednak pewien problem: zdarzenia związane z ruchem myszy występują bardzo często, kiedy mysz jest przesuwana. Jeśli komputer nie jest wystarczająco szybki, aby na nie odpowiedzieć, albo przetwarzanie zajmuje zbyt wiele czasu, wówczas zdarzenia zaczynają się gromadzić. Sytuacja taka mogłaby mieć miejsce, kiedy wyświetlalibyśmy bardzo dużą cząsteczkę, wymagającą złożonych obliczeń i długiego rysowania. W takim przypadku przetwarzanie zdarzeń i rysowanie cząsteczki trwałoby nadal już po zaprzestaniu ruchu myszy (aby oczyścić kolejkę zdarzeń), co nie wygląda najlepiej (aby zapoznać się z tym problemem, możemy wykomentować GDK_POINTER_MOTION_HINT_MASK i przesunąć jedną ze złożonych cząsteczek, wyświetlaną wraz z kolorami, wiązaniami i nazwami atomów. Przesuwajmy ją przez kilka sekund i zatrzymajmy mysz). Lepiej jest skorzystać ze wskazówek o ruchu myszy.

Zazwyczaj każdy ruch myszy generuje zdarzenie. Mogą one się nawarstwić i spowodować ociężałość aplikacji, która będzie próbowała oczyścić kolejkę z wszystkich nagromadzonych zdarzeń. Lepiej jest skorzystać z GDK_POINTER_MOTION_HINT_MASK i pozwolić aplikacji na przetwarzanie zdarzeń związanych z ruchem myszy w jej własnym tempie. Zdarzenia nie nawarstwiają się w trakcie ruchu myszy; generowane jest pojedyncze zdarzenie, które wskazuje, że wskaźnik myszy zmienił położenie. Ponieważ ustawiono znacznik wskazówki, w zdarzeniu nie ma żadnych danych na temat aktualnej pozycji myszy. Aplikacja musi sprawdzić, gdzie znajduje się wskaźnik, przy pomocy funkcji gdk_window_ get_pointer.

Prosty przykład unaoczni różnicę pomiędzy używaniem zwykłych zdarzeń związanych z ruchem myszy a korzystaniem ze wskazówek. Załóżmy, że napisaliśmy aplikację (na przykład przeglądarkę cząsteczek), która rysuje znacznie bardziej skomplikowane modele, z trójwymiarowym cieniowaniem atomów. Działa doskonale na naszym najnowocześniejszym, 600-megahercowym, dwuprocesorowym komputerze, więc decydujemy się udostępnić ją reszcie świata na naszej osobistej stronie WWW. Użytkownik komputera 386SX 16MHz ściąga przeglądarkę, chcąc zanalizować kilka skomplikowanych cząsteczek. Przypuśćmy teraz, że aplikacja używa standardowych zdarzeń związanych z ruchem myszy. Nasz supernowoczesny komputer potrzebuje jednej setnej sekundy, aby

narysować model, kiedy obracamy go przy użyciu myszy. Wszystko działa świetnie, kiedy obracamy cząsteczkę w różne strony. Niestety, na 386-ce rysowanie modelu zajmuje 20 sekund, a kiedy komputer ze wszystkich sił stara się wyświetlić cząsteczkę, użytkownik przeciąga ją myszą, zastanawiając się, czemu rysowanie trwa tak długo. Po 20 sekundach cząsteczka wreszcie pojawia się na ekranie-ale w kolejce znajdują się setki zdarzeń, związanych z ruchem myszy. Nawet, jeśli użytkownik nie zrobi nic innego, każde z tych zdarzeń spowoduje przerysowanie cząsteczki, blokując komputer (przynajmniej w teorii) na długie minuty. Jeśli jednak program będzie używał wskazówek o ruchu myszy, wówczas po przerysowaniu modelu w kolejce będzie oczekiwało tylko jedno zdarzenie, informujące o zmianie położenia wskaźnika. Można sprawdzić, gdzie użytkownik przesunął mysz i odpowiednio zareagować. Oczywiście, nasz przykład jest mocno przesadzony, a osoby zajmujące się trójwymiarową grafiką na 386SX mają nierówno pod sufitem.

Aby wykryć naciśnięcie lub zwolnienie klawisza myszy, musimy ustawić także znaczniki GDK_BUTTON_PRESS_MASK i GDK_BUTTON_RELEASE_MASK. Zmiany masek zdarzeń, dzięki którym program będzie otrzymywać niskopoziomowe zdarzenia, muszą być przeprowadzone przed realizacją kontrolki. Najlepszym momentem do ustawienia masek zdarzeń dla dowolnej kontrolki jest chwila tuż po jej utworzeniu. Poniższy przykład pokazuje, w jaki sposób można skonfigurować obszar rysunkowy tak, aby odbierał zdarzenia związane z ruchem myszy.

```
GtkWidget *UtworzObszarRysunkowy ()
{
   GtkWidget *okno;

   /* --- Tworzymy okno najwyższego poziomu --- */
   okno = gtk_window_new (GTK_WINDOW_TOPLEVEL);

   /* --- Tworzymy obszar rysunkowy  --- */
   obszar_rys = gtk_drawing_area_new ();

   /* --- Ustawiamy rozmiary --- */
   gtk_drawing_area_size (GTK_DRAWING_AREA (obszar_rys), 300, 300);

   /* --- Uwidaczniamy obszar rysunkowy --- */
   gtk_widget_show (obszar_rys);

   /* - Sygnały wykorzystywane do obsługi drugoplanowej piksmapy - */
   gtk_signal_connect (GTK_OBJECT (obszar_rys), "expose_event",
                       (GtkSignalFunc) expose_event, NULL);
```

```
    gtk_signal_connect (GTK_OBJECT(obszar_rys), "configure_event",
                        (GtkSignalFunc) configure_event, NULL);

    /* --- Musimy być informowani o ruchach myszy --- */
    gtk_signal_connect (GTK_OBJECT (obszar_rys),"motion_notify_event",
                        (GtkSignalFunc) motion_notify_event, NULL);

    /* --- Zdarzenia, które nas interesują --- */
    gtk_widget_set_events (obszar_rys, GDK_EXPOSURE_MASK
                         | GDK_LEAVE_NOTIFY_MASK
                         | GDK_BUTTON_PRESS_MASK
                         | GDK_POINTER_MOTION_MASK
                         | GDK_POINTER_MOTION_HINT_MASK);

    return (obszar_rys);
}
```

Aby zobaczyć efekt, jaki dają wskazówki o ruchu myszy, możemy usunąć znacznik GDK_POINTER_MOTION_HINT_MASK z wywołania funkcji gtk_widget_set_events. Oczywiście musimy wyświetlić skomplikowaną cząsteczkę, albo uruchomić komputer na powolnym komputerze, aby zaobserwować nawarstwianie się zdarzeń.

motion_notify_event

Funkcja motion_notify_event wykrywa ruchy myszy i określa, czy należy przerysować cząsteczkę. W celu obrócenia cząsteczki wywoływana jest funkcja ruchMyszy. Zmienne poprzx i poprzy przechowują poprzednie położenie wskaźnika myszy; służą do obliczenia kierunku, w którym została przesunięta mysz. Znając kierunek ruchu myszy możemy zbudować właściwą macierz i obrócić cząsteczkę. Funkcja ta obsługuje zarówno wskazówki, jak i zwykłe zdarzenia związane z ruchem myszy. Jeśli obszar rysunkowy skonfigurowany jest tak, aby korzystał ze wskazówek o ruchu myszy, wówczas zdarzenie->is_hint ma wartość TRUE i funkcja będzie w każdym wywołaniu sprawdzać bieżącą pozycję wskaźnika myszy.

```
/*
 * motion_notify_event
 *
 * Wywoływana podczas ruchu myszy w obrębie okna
 */
gint motion_notify_event (GtkWidget *kontrolka, GdkEventMotion *zdarzenie)
```

```
{
   int x, y;
   GdkModifierType stan;

   /* --- Jeśli to wskazówka (kombinacja kilku zdarzeń) --- */
   if (zdarzenie->is_hint) {

      /* --- Pobieramy nową pozycję --- */
      gdk_window_get_pointer (zdarzenie->window, &x, &y, &stan);
   } else {

      /* --- Pobieramy nową pozycję --- */
      x = zdarzenie->x;
      y = zdarzenie->y;
      stan = zdarzenie->state;
   }

   /* --- Jeśli wciśnięty jest przycisk myszy --- */
   if (stan & GDK_BUTTON1_MASK && g->piksmapa != NULL) {

      /* --- Obliczamy wpływ ruchu myszy na
       *     wyświetlaną cząsteczkę
       */
      ruchMyszy (x, y);
   }

   /* --- Zapamiętujemy położenie myszy --- */
   poprzx = x;
   poprzy = y;

   return TRUE;
}
```

ruch Myszy

Funkcja ruchMyszy przelicza macierz, służącą do obracania cząsteczki. Przypomnijmy sobie, że każdy atom ma położenie pierwotne (x, y, z), odczytane z pliku .pdb, oraz położenie przekształcone, używane podczas rysowania cząsteczki. Oba położenia są powiązane w następujący sposób: cząsteczka jest obracana przez mnożenie pierwotnego położenia atomu przez macierz atomu (amacierz), w wyniku czego otrzymujemy przekształcone położenie. Macierz atomu jest modyfikowana tak, aby uwzględnić ruch myszy. Tworzymy nową macierz (tmacierz) i zmieniamy

ją tak, aby odzwierciedlała ruch cząsteczki wokół osi x i y. Nowa macierz jest następnie mnożona przez macierz atomu, dzięki czemu macierz atomu również odzwierciedla zmiany spowodowane ruchem myszy. Pierwotne współrzędne są więc przekształcane przez macierz atomu, aby uzyskać przekształcone współrzędne atomu. Następnie wywoływana jest funkcja OdswiezCzasteczke, aby wyświetlić cząsteczkę na ekranie pod nowym kątem.

```
/*
 * ruchMyszy
 *
 * Oblicza obrót cząsteczki na podstawie sposobu,
 * w jaki użytkownik przeciągnął mysz nad cząsteczką.
 *
 * x - położenie x myszy
 * x - położenie y myszy
 */
int ruchMyszy (int x, int y)
{

   /* --- Obliczamy różnicę x --- */
   double xtheta = (poprzy - y) * (360.0f / nSzerEkranu);

   /* --- Obliczamy różnicę y --- */
   double ytheta = (x - poprzx) * (360.0f / nWysokEkranu);

   /* --- Macierz jednostkowa --- */
   jednostka (tmacierz);

   /* --- Obracamy o różnicę -x- ruchu myszy --- */
   xobrot (tmacierz, xtheta);

   /* --- Obracamy o różnicę -y- ruchu myszy --- */
   yobrot (tmacierz, ytheta);

   /* --- Łączymy z bieżącym obrotem, aby uzyskać nowy --- */
   mnoz (amacierz, tmacierz);

   /* --- Przerysowujemy z nowym obrotem --- */
   OdswiezCzasteczke ();

   return TRUE;
}
```

interfejs.c

Plik interfejs.c zawiera funkcje służące do utworzenia głównego okna, ustawienia paska narzędziowego i menu oraz obsługi głównych zdarzeń (pochodzących od przycisków paska, menu itd.). Kod jest prawie taki sam, jak w poprzednich przykładach; dzięki kilku zmianom przeglądarka cząsteczek ma swoje własne menu i pasek narzędziowy.

```
/*
 * Plik: interfejs.c
 * Autor: Eric Harlow
 *
 * Interfejs GUI dla przeglądarki cząsteczek
 *
 */

#include <sys/stat.h>
#include <unistd.h>
#include <errno.h>
#include <gtk/gtk.h>

/*
 * --- Prototypy funkcji
 */
void PobierzNazwePliku (char *sTitle, void (*callback) (char *));
void OdswiezCzasteczke ();
void CzasteczkaPokazLinie (int bWartosc);
void WczytajCzasteczke (char *);
GtkWidget *UtworzObszarRysunkowy ();
static void UtworzGlowneOkno ();
void UtworzPasek (GtkWidget *ypole);
void UstawPasek (char *szPrzycisk, int nStan);
void ZaznaczMenu (GtkWidget *kontrolka, gpointer dane);
void OdznaczMenu (GtkWidget *kontrolka, gpointer dane);
void UstawMenu (char *szPrzycisk, int nStan) ;
GtkWidget *UtworzKontrolkeZXpm (GtkWidget *okno, gchar **xpm_dane);
GtkWidget *UtworzElementMenu (GtkWidget *menu,
                    char *szNazwa,
                    char *szSkrot,
                    char *szPodp,
                    GtkSignalFunc funkcja,
                    gpointer dane);
GtkWidget *UtworzZaznaczalnyElement (GtkWidget *menu,
```

```
                    char *szNazwa,
                    GtkSignalFunc funkcja,
                    gpointer dane);
GtkWidget *UtworzPodmenu (GtkWidget *pasekmenu, char *szNazwa);
GtkWidget *UtworzPodmenuPaska (GtkWidget *menu, char *szNazwa);

/*
 * --- Zmienne globalne
 */
GtkWidget       *glowne_okno;
GtkTooltips     *podpowiedzi;
GtkAccelGroup   *grupa_skrotow;
GtkWidget       *pasek;
GtkWidget       *pasek_linie;
GtkWidget       *pasek_rgb;
GtkWidget       *pasek_etykiety;

/*
 * --- Bitmapa dla przycisku "otwórz"
 */
static const gchar *xpm_otworz[] = {
"16 16 4 1",
"  c None",
"B c #000000000000",
"Y c #FFFFFFFF0000",
"y c #999999990000",
"              ",
"         BBB  ",
"  BBBBB  B   BB ",
"  BYYYB      BB ",
" BYYYYYBBBBB    ",
" BYYYYYYYYYB    ",
" BYYYYYYYYYB    ",
" BYYYYYYYYYB    ",
" BYYBBBBBBBBBB ",
" BYYByyyyyyyyyB ",
" BYByyyyyyyyyB  ",
" BYByyyyyyyyyB  ",
" BByyyyyyyyyB   ",
" BByyyyyyyyyB   ",
" BBBBBBBBBBB    ",
```

```
"                ",
};

/*
 * --- Bitmapa dla przycisku "linie"
 */
static const char *xpm_linie[] = {
"16 16 2 1",
"  c None",
"B c #000000000000",
"                ",
"                ",
"  BB       BB   ",
"  BB       BB   ",
"   B       B    ",
"   B      B     ",
"    B    B      ",
"    B   B       ",
"     B B        ",
"     BB         ",
"     BBBB        ",
"         BB     ",
"           BBB  ",
"            BB  ",
"                ",
"                ",
};

/*
 * --- Bitmapa dla przycisku "kolor"
 */
static const char *xpm_rgb[] = {
"16 16 4 1",
"  c None",
"R c #FF0000",
"G c #00FF00",
"B c #0000FF",
"                ",
"     BBBRRR     ",
"   BBBBBRRRRR   ",
"  BBBBBBRRRRRR  ",
```

```
"  BBBBBBRRRRRR  ",
"  BBBBBBRRRRRR  ",
" BBBBBBBRRRRRRR ",
" BBBBBBBRRRRRRR ",
" BBBBBGGGGRRRRR ",
" BBBGGGGGGGGRRR ",
"  GGGGGGGGGGGG  ",
"  GGGGGGGGGGGG  ",
"  GGGGGGGGGGGG  ",
"   GGGGGGGGGG   ",
"     GGGGGGG    ",
"                ",
};

/*
 * --- Bitmapa dla przycisku "etykiety"
 */
static const char *xpm_etykiety[] = {
"16 16 4 1",
"  c None",
"R c #FF0000",
"G c #00FF00",
"B c #000000",
"                ",
"       BB       ",
"     BBBBBB     ",
"    BBB  BBB    ",
"   BB      BB   ",
"   BB      BB   ",
"  BB        BB  ",
"  BB        BB  ",
"  BBBBBBBBBBBB  ",
"  BBBBBBBBBBBB  ",
"  BB        BB  ",
"  BB        BB  ",
"  BB        BB  ",
"  BB        BB  ",
"  BB        BB  ",
"                ",
};
```

```
/*
 * KoniecProgramu
 *
 * Wyjście z programu
 */
void KoniecProgramu ()
{
   gtk_main_quit ();
}

/*
 * main
 *
 * --- Program zaczyna się tutaj
 */
int main(int argc, char *argv[])
{
   /* --- Inicjacja GTK --- */
   gtk_init (&argc, &argv);

   /* --- Inicjacja podpowiedzi --- */
   podpowiedzi = gtk_tooltips_new ();

   /* --- Tworzymy okno --- */
   UtworzGlowneOkno ();

   /* --- Wczytujemy domyślną cząsteczkę --- */
   WczytajCzasteczke ("molecule.pdb");

   /* --- Główna pętla obsługi zdarzeń --- */
   gtk_main();

   return 0;
}

/*
 * PokazEtykiety
 *
 * Czy jest wciśnięty przycisk "pokaż etykiety" na
 * pasku narzędziowym, który wskazuje, że użytkownik
 * chce wyświetlić nazwy atomów?
 */
int PokazEtykiety ()
```

```
{
    return (GTK_TOGGLE_BUTTON (pasek_etykiety)->active);
}

/*
 * PokazRGB
 *
 * Czy jest wciśnięty przycisk "pokaż kolory" na
 * pasku narzędziowym, który wskazuje, że użytkownik
 * chce wyświetlić atomy w kolorze?
 */
int PokazRGB ()
{
    return (GTK_TOGGLE_BUTTON (pasek_rgb)->active);
}

/*
 * PokazLinie
 *
 * Wskazuje, czy pomiędzy atomami należy rysować linie,
 * które przedstawiają wiązania w cząsteczce.
 */
int PokazLinie ()
{
    return (GTK_TOGGLE_BUTTON (pasek_linie)->active);
}

/*
 * OtworzPlik
 *
 * Otwiera plik .pdb
 */
void OtworzPlik (GtkWidget *kontrolka, gpointer dane)
{
    PobierzNazwePliku ("Otwórz cząsteczkę", WczytajCzasteczke);
}

/*
 * UtworzGlowneOkno
 *
```

```
 * Tworzy główne okno i związane z nim menu/paski narzędziowe.
 */
static void UtworzGlowneOkno ()
{
   GtkWidget *kontrolka;
   GtkWidget *ypole;
   GtkWidget *pasekmenu;
   GtkWidget *menu;
   GtkWidget *elmenu;

   /* --- Tworzymy główne okno i ustawiamy jego rozmiary --- */
   glowne_okno = gtk_window_new(GTK_WINDOW_TOPLEVEL);
   gtk_widget_set_usize(glowne_okno, 360, 260);
   gtk_window_set_title (GTK_WINDOW (glowne_okno),
                         "Przeglądarka cząsteczek");
   /* gtk_container_border_width (GTK_CONTAINER (glowne_okno), 0); */

   /* --- Tworzymy skróty klawiszowe --- */
   grupa_skrotow = gtk_accel_group_new();
   gtk_accel_group_attach (grupa_skrotow, GTK_OBJECT (glowne_okno));

   /* -- Okno najwyższego poziomu musi czekać na sygnał destroy -- */
   gtk_signal_connect (GTK_OBJECT (glowne_okno), "destroy",
           GTK_SIGNAL_FUNC(KoniecProgramu), NULL);

   /* --- Tworzymy pole pakujące na menu i pasek narzędziowy --- */
   ypole = gtk_vbox_new (FALSE, 0);

   /* --- Pokazujemy pole pakujące --- */
   gtk_container_add (GTK_CONTAINER (glowne_okno), ypole);

   gtk_widget_show (ypole);
   gtk_widget_show (glowne_okno);

   /* --- Pasek menu --- */
   pasekmenu = gtk_menu_bar_new ();
   gtk_box_pack_start (GTK_BOX (ypole), pasekmenu, FALSE, TRUE, 0);
   gtk_widget_show (pasekmenu);

   /* ------------------
     --- Menu Plik ---
     ------------------ */
   menu = UtworzPodmenuPaska (pasekmenu, "Plik");
```

```
    elmenu = UtworzElementMenu (menu, "Otwórz", "^O",
                "Otwiera cząsteczkę",
                GTK_SIGNAL_FUNC (OtworzPlik), "otwórz");

    elmenu = UtworzElementMenu (menu, NULL, NULL,
                NULL, NULL, NULL);

    elmenu = UtworzElementMenu (menu, "Zakończ", "",
                "Czy jest bardziej wymowna opcja?",
                GTK_SIGNAL_FUNC (KoniecProgramu), "zakończ");

    /* --- Tworzymy pasek narzędziowy --- */
    UtworzPasek (ypole);

    kontrolka = UtworzObszarRysunkowy ();
    gtk_box_pack_start (GTK_BOX (ypole), kontrolka, TRUE, TRUE, 0);
}

/*
 * UtworzPasek
 *
 * Tworzy pasek narzędziowy
 */
void UtworzPasek (GtkWidget *ypole)
{

    /* --- Tworzymy pasek i dodajemy go do okna --- */
    pasek = gtk_toolbar_new (GTK_ORIENTATION_HORIZONTAL,
                        GTK_TOOLBAR_ICONS);
    gtk_box_pack_start (GTK_BOX (ypole), pasek, FALSE, TRUE, 0);
    gtk_widget_show (pasek);

    /* --- Tworzymy przycisk "otwórz" --- */
    gtk_toolbar_append_item (GTK_TOOLBAR (pasek),
             "Okno dialogowe Otwórz", "Okno dialogowe Otwórz", "",
             UtworzKontrolkeZXpm (ypole, (gchar **) xpm_otworz),
             (GtkSignalFunc) OtworzPlik,
             NULL);

    /* --- Niewielki odstęp --- */
    gtk_toolbar_append_space (GTK_TOOLBAR (pasek));

    pasek_linie = gtk_toolbar_append_element (GTK_TOOLBAR (pasek),
             GTK_TOOLBAR_CHILD_TOGGLEBUTTON,
```

```
                NULL,
                "Wiązania", "Wiązania", "Wiązania",
                UtworzKontrolkeZXpm (ypole, (gchar **) xpm_linie),
                (GtkSignalFunc) OdswiezCzasteczke,
                NULL);

        pasek_rgb = gtk_toolbar_append_element (GTK_TOOLBAR (pasek),
                GTK_TOOLBAR_CHILD_TOGGLEBUTTON,
                NULL,
                "Kolory", "Kolory", "Kolory",
                UtworzKontrolkeZXpm (ypole, (gchar **) xpm_rgb),
                (GtkSignalFunc) OdswiezCzasteczke,
                NULL);

        pasek_etykiety = gtk_toolbar_append_element (GTK_TOOLBAR (pasek),
                GTK_TOOLBAR_CHILD_TOGGLEBUTTON,
                NULL,
                "Nazwy atomów", "Nazwy atomów", "Nazwy atomów",
                UtworzKontrolkeZXpm (ypole, (gchar **) xpm_etykiety),
                (GtkSignalFunc) OdswiezCzasteczke,
                NULL);

    }
```

macierz 3d.c

W pliku macierz3d.c znajdują się funkcje służące do mnożenia macierzy. Jeśli czytelnik byłby zainteresowany bliższym poznaniem arytmetyki trójwymiarowej, zachęcamy do sięgnięcia po odpowiednią książkę (mnożenie macierzy powinna wyjaśnić książka poświęcona algebrze liniowej, a także wiele książek traktujących o grafice 3D). Załączamy tutaj kod dla pełnego obrazu.

```
/*
 * Plik: macierz3d.c
 * Autor: Eric Harlow
 *
 * Konwersja z pliku klasy Javy Matrix 3D
 */

#include "atom.h"
#include "macierz3d.h"
#include <math.h>
```

```
static double pi = 3.14159265;

/*
 * NowaMacierz3D
 *
 * Tworzy nową macierz
 */
typMacierz3D *NowaMacierz3D ()
{
   typMacierz3D *macierz;

   macierz = (typMacierz3D *) g_malloc (sizeof (typMacierz3D));

   jednostka (macierz);

   return (macierz);
}

/*
 * skaluj
 *
 * Skaluje obiekt
 */
void sksluj (typMacierz3D *macierz, double f)
{
   macierz->xx *= f;
   macierz->xy *= f;
   macierz->xz *= f;
   macierz->xo *= f;
   macierz->yx *= f;
   macierz->yy *= f;
   macierz->yz *= f;
   macierz->yo *= f;
   macierz->zx *= f;
   macierz->zy *= f;
   macierz->zz *= f;
   macierz->zo *= f;
}

/*
 * skaluj3
 *
 * Skaluje każdy kierunek o inny czynnik
```

```
 */
void skaluj3 (typMacierz3D *macierz, double xf, double yf, double zf)
{
   macierz->xx *= xf;
   macierz->xy *= xf;
   macierz->xz *= xf;
   macierz->xo *= xf;
   macierz->yx *= yf;
   macierz->yy *= yf;
   macierz->yz *= yf;
   macierz->yo *= yf;
   macierz->zx *= zf;
   macierz->zy *= zf;
   macierz->zz *= zf;
   macierz->zo *= zf;
}

/*
 * przestaw
 *
 * Przesuwa punkt reprezentowany przez macierz o (x, y, z)
 */
void przestaw (typMacierz3D *macierz, double x, double y, double z)
{
   macierz->xo += x;
   macierz->yo += y;
   macierz->zo += z;
}

/*
 * yobrot
 *
 * Dodaje do macierzy obrót wokół osi y o kąt (theta).
 */
void yobrot (typMacierz3D *macierz, double theta)
{
   double ct;
   double st;
   double Nxx;
   double Nxy;
   double Nxz;
```

```
    double Nxo;
    double Nzx;
    double Nzy;
    double Nzz;
    double Nzo;

    theta *= (pi / 180);
    ct = cos (theta);
    st = sin (theta);

    Nxx = (double) (macierz->xx * ct + macierz->zx * st);
    Nxy = (double) (macierz->xy * ct + macierz->zy * st);
    Nxz = (double) (macierz->xz * ct + macierz->zz * st);
    Nxo = (double) (macierz->xo * ct + macierz->zo * st);

    Nzx = (double) (macierz->zx * ct - macierz->xx * st);
    Nzy = (double) (macierz->zy * ct - macierz->xy * st);
    Nzz = (double) (macierz->zz * ct - macierz->xz * st);
    Nzo = (double) (macierz->zo * ct - macierz->xo * st);

    macierz->xo = Nxo;
    macierz->xx = Nxx;
    macierz->xy = Nxy;
    macierz->xz = Nxz;
    macierz->zo = Nzo;
    macierz->zx = Nzx;
    macierz->zy = Nzy;
    macierz->zz = Nzz;
}

/*
 * xobrot
 *
 * Dodaje do macierzy obrót wokół osi x o kąt (theta).
 */
void xobrot (typMacierz3D *macierz, double theta)
{
    double ct;
    double st;
    double Nyx;
    double Nyy;
    double Nyz;
```

```
    double Nyo;
    double Nzx;
    double Nzy;
    double Nzz;
    double Nzo;

    theta *= (pi / 180);
    ct = cos (theta);
    st = sin (theta);

    Nyx = (double) (macierz->yx * ct + macierz->zx * st);
    Nyy = (double) (macierz->yy * ct + macierz->zy * st);
    Nyz = (double) (macierz->yz * ct + macierz->zz * st);
    Nyo = (double) (macierz->yo * ct + macierz->zo * st);

    Nzx = (double) (macierz->zx * ct - macierz->yx * st);
    Nzy = (double) (macierz->zy * ct - macierz->yy * st);
    Nzz = (double) (macierz->zz * ct - macierz->yz * st);
    Nzo = (double) (macierz->zo * ct - macierz->yo * st);

    macierz->yo = Nyo;
    macierz->yx = Nyx;
    macierz->yy = Nyy;
    macierz->yz = Nyz;
    macierz->zo = Nzo;
    macierz->zx = Nzx;
    macierz->zy = Nzy;
    macierz->zz = Nzz;
}

/*
 * zobrot
 *
 * Dodaje do macierzy obrót wokół osi z o kąt (theta).
 */
void zobrot (typMacierz3D *macierz, double theta)
{
    double ct;
    double st;
    double Nyx;
    double Nyy;
    double Nyz;
```

```
    double Nyo;
    double Nxx;
    double Nxy;
    double Nxz;
    double Nxo;

    theta *= (pi / 180);
    ct = cos(theta);
    st = sin(theta);

    Nyx = (double) (macierz->yx * ct + macierz->xx * st);
    Nyy = (double) (macierz->yy * ct + macierz->xy * st);
    Nyz = (double) (macierz->yz * ct + macierz->xz * st);
    Nyo = (double) (macierz->yo * ct + macierz->xo * st);

    Nxx = (double) (macierz->xx * ct - macierz->yx * st);
    Nxy = (double) (macierz->xy * ct - macierz->yy * st);
    Nxz = (double) (macierz->xz * ct - macierz->yz * st);
    Nxo = (double) (macierz->xo * ct - macierz->yo * st);

    macierz->yo = Nyo;
    macierz->yx = Nyx;
    macierz->yy = Nyy;
    macierz->yz = Nyz;
    macierz->xo = Nxo;
    macierz->xx = Nxx;
    macierz->xy = Nxy;
    macierz->xz = Nxz;
}

/*
 * mnoz
 *
 * Mnoży pierwszą macierz przez drugą.
 * Nowa wartość jest zachowywana w pierwszej macierzy.
 */
void mnoz (typMacierz3D *macierz, typMacierz3D *rhs)
{
    double lxx = macierz->xx * rhs->xx +
                 macierz->yx * rhs->xy +
                 macierz->zx * rhs->xz;
    double lxy = macierz->xy * rhs->xx +
```

```
                macierz->yy * rhs->xy +
                macierz->zy * rhs->xz;
double lxz = macierz->xz * rhs->xx +
                macierz->yz * rhs->xy +
                macierz->zz * rhs->xz;
double lxo = macierz->xo * rhs->xx +
                macierz->yo * rhs->xy +
                macierz->zo * rhs->xz + rhs->xo;

double lyx = macierz->xx * rhs->yx +
                macierz->yx * rhs->yy +
                macierz->zx * rhs->yz;
double lyy = macierz->xy * rhs->yx +
                macierz->yy * rhs->yy +
                macierz->zy * rhs->yz;
double lyz = macierz->xz * rhs->yx +
                macierz->yz * rhs->yy +
                macierz->zz * rhs->yz;
double lyo = macierz->xo * rhs->yx +
                macierz->yo * rhs->yy +
                macierz->zo * rhs->yz + rhs->yo;

double lzx = macierz->xx * rhs->zx +
                macierz->yx * rhs->zy +
                macierz->zx * rhs->zz;
double lzy = macierz->xy * rhs->zx +
                macierz->yy * rhs->zy +
                macierz->zy * rhs->zz;
double lzz = macierz->xz * rhs->zx +
                macierz->yz * rhs->zy +
                macierz->zz * rhs->zz;
double lzo = macierz->xo * rhs->zx +
                macierz->yo * rhs->zy +
                macierz->zo * rhs->zz + rhs->zo;

macierz->xx = lxx;
macierz->xy = lxy;
macierz->xz = lxz;
macierz->xo = lxo;

macierz->yx = lyx;
macierz->yy = lyy;
```

```
    macierz->yz = lyz;
    macierz->yo = lyo;

    macierz->zx = lzx;
    macierz->zy = lzy;
    macierz->zz = lzz;
    macierz->zo = lzo;
}

/*
 * jednostka
 *
 * Czyni macierz macierzą jednostkową
 */
void jednostka (typMacierz3D *macierz)
{
    macierz->xo = 0;
    macierz->xx = 1;
    macierz->xy = 0;
    macierz->xz = 0;
    macierz->yo = 0;
    macierz->yx = 0;
    macierz->yy = 1;
    macierz->yz = 0;
    macierz->zo = 0;
    macierz->zx = 0;
    macierz->zy = 0;
    macierz->zz = 1;
}

/*
 * Transformuj
 *
 * Poddaje translacji współrzędne atomu
 */
void Transformuj (typMacierz3D *macierz, typAtom *atom)
{
    double lxx = macierz->xx, lxy = macierz->xy, lxz = macierz->xz, lxo = ma-
cierz->xo;
    double lyx = macierz->yx, lyy = macierz->yy, lyz = macierz->yz, lyo = ma-
cierz->yo;
```

```
    double lzx = macierz->zx, lzy = macierz->zy, lzz = macierz->zz, lzo = ma-
cierz->zo;

    double x = atom->x;
    double y = atom->y;
    double z = atom->z;

    atom->tx = (x * lxx + y * lxy + z * lxz + lxo);
    atom->ty = (x * lyx + y * lyy + z * lyz + lyo);
    atom->tz = (x * lzx + y * lzy + z * lzz + lzo);
}
```

Pozostała część kodu

Pozostała część kodu została wzięta z innych rozdziałów. Plik rozne.c pochodzi z rozdziału 5, „Menu, paski narzędziowe i podpowiedzi" i pomaga w tworzeniu menu i paska narzędziowego. Kod wybpliku.c pochodzi z rozdziału 6, „Dalsze kontrolki: ramki, tekst, okna dialogowe, okno wyboru pliku i pasek postępów". Moduły te nie wymagały wielu zmian (jeśli w ogóle) w celu dostosowania ich do pracy w przeglądarce cząsteczek. Zmieniono w niewielkim stopniu wybpliku.c, aby uczynić go bardziej uniwersalnym; teraz można wykorzystywać go w dowolnym programie (co, jak wiadomo, jest oznaką modularności kodu).

Podsumowanie

Przeglądarka cząsteczek jest bardziej skomplikowanym przykładem aplikacji stworzonej przy użyciu GDK. Ilustruje ona sposób interakcji użytkownika z kontrolką obszaru rysunkowego GDK. Do obracania cząsteczki wykorzystaliśmy zdarzenia związane z ruchem myszy. Rozdział omówił niektóre spośród niskopoziomowych zdarzeń, których obsługa jest konieczna, aby możliwe było obracanie cząsteczek przy pomocy myszy.