Uniwersytet Mikołaja Kopernika
Wydział Fizyki, Astronomii i Informatyki Stosowanej

Jakub Przewłocki
nr albumu: 169422

Praca magisterska na kierunku fizyka techniczna

Wizualizacja danych z użyciem grafiki trójwymiarowej i mechanizmy interakcji z użytkownikiem w OpenGL

Opiekun pracy dyplomowej
dr Jacek Matulewski
Zakład Mechaniki Kwantowej

Toruń 2008

Pracę przyjmuję i akceptuję

.....

data i podpis opiekuna pracy

Potwierdzam złożenie pracy dyplomowe

.....

data i podpis pracownika dziekanatu

Dziękuję Panu doktorowi
Jackowi Matulewskiemu
za cierpliwość, pomoc i opiekę
nad projektem

Spis treści

1. Wstęp

Moim zadaniem było przygotowanie programu wykorzystującego bibliotekę OpenGL do wizualizacji funkcji dwóch zmiennych $z = f(x,y)$ zadanej przez odczytany z pliku zbiór punktów (x,y,z). Charakterystyczną cechą programu ma być możliwość interakcji z użytkownikiem obejmująca dowolne obracania wykresu za pomocą myszki, zaznaczanie i przeciąganie źródeł światła (również myszką) i podokna pozwalające na wygodne ustalanie własności wykresu oraz tworzenia i edycji źródeł światła. Wymagać to będzie m. in. użycia trybu selekcji OpenGL umożliwiającego na interakcję użytkownika z zawartością sceny oraz wprowadzenia do obsługi kamery mechanizmu rotacji sferycznej. Program zawiera też moduł importu pozwalający na wczytywanie plików z danymi w postaci trójek współrzędnych x, y i z oraz plików zawierających jedynie część rzeczywistą i urojoną funkcji zespolonej. W tym drugim przypadku parametry sieci muszą być pobierane z osobnego pliku lub wpisane przez użytkownika w udostępnionym przez program formularzu.

W pierwszych rozdziałach pracy opisuję najciekawsze, a zarazem najtrudniejsze do zaimplementowania, techniki użyte w programie, a więc rotację sferyczną, mechanizm selekcji obiektów w OpenGL oraz poruszanie źródeł światła za pomocą myszki. Kolejne rozdziały poświęcone są zaimplementowanym w programie narzędziom takim, jak moduł do importu danych, narzędzie do definiowania i edycji własności źródeł światła oraz narzędzia pozwalające na modyfikowanie wyglądu sceny oraz dobierania parametrów wykresu.

2. Metody poruszania obiektami

2.1. Rotacja sferyczna

Każdy zaawansowany program wyświetlający grafikę trójwymiarową daje możliwość swobodnego obracania modelu za pomocą myszy. Na tym właśnie polega rotacja sferyczna. W tym rozdziale przedstawię implementacje tej techniki na przykładzie mojego programu (Rys. 1.) oraz powody dla których watro jej używać. Całość oparta jest na standardowej klasie *ArcBall*, która została opisana w „Perełkach programowania gier"[13], a także w witrynie NeHe (lekcja 48)[7].

Klasa *ArcBall* oparta jest na kwaternionach. Są to obiekty matematyczne, które służą programistą grafiki i gier do reprezentowania rotacji i orientacji w przestrzeni 3D [17]. Ogólnie kwaterniony to struktura algebraiczna będąca rozszerzeniem ciała liczb zespolonych. Jedna z jego notacji reprezentowana jest w postaci pary, w której pierwsza część to rzeczywista wielkość skalarna, a druga to trójwymiarowy wektor współczynników części urojonych:

```
[w (x y z)]
```

W grafice 3D kwaternion tworzy kąt obrotu *w*, oraz oś obrotu (*x y z*) wokół której wykonywany jest obrót [19]. Jeżeli przyjmiemy, że mamy kąt obrotu α i *n* jako wektor jednostkowy opisujący oś, to kwaternion reprezentujący określony obrót będzie miał postać (1):

$$w = \cos(\frac{\theta}{2})$$

$$\vec{v} = \sin(\frac{\theta}{2})\vec{n} \qquad \text{gdzie} \quad \vec{v} = (x,y,z) \qquad (1)$$

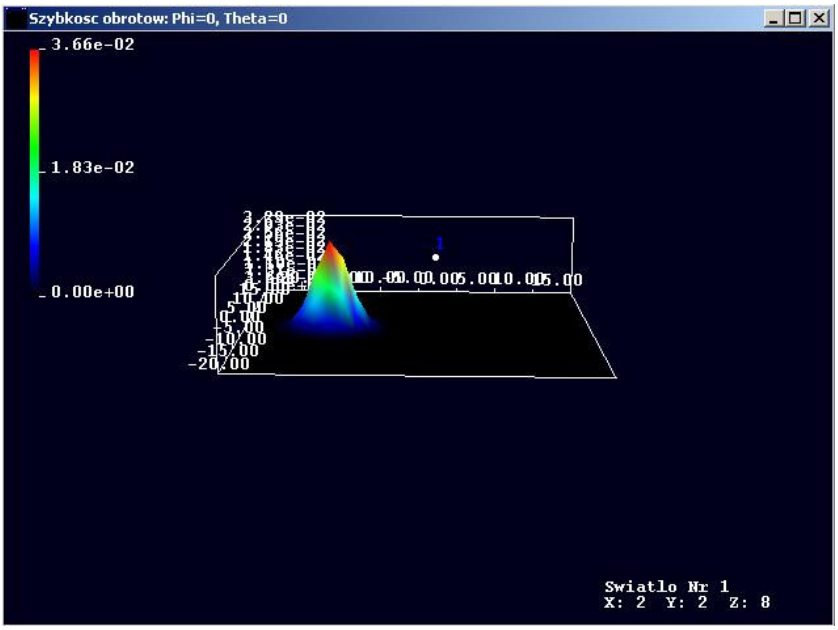
$$[\cos(\frac{\theta}{2}) \quad \sin(\frac{\theta}{2})\vec{n}\,]$$

Mając te wszystkie dane możemy stworzyć macierz (2):

$$\begin{matrix} w^2 + x^2 - y^2 - z^2 & 2xy + 2wz & 2xz - 2wy & 0 \\ 2xy - 2wz & w^2 - x^2 + y^2 - z^2 & 2yz + 2wx & 0 \\ 2xz + 2wy & 2yz - 2wx & w^2 - x^2 - y^2 + z^2 & 0 \\ 0 & 0 & 0 & w^2 + x^2 + y^2 + z^2 \end{matrix} \quad (2)$$

Mając punkty i wektory możemy przekształcić je, poprzez przemnożenie ich przez powyższą macierz (2) reprezentującą obrót [17].

Rotacja sferyczna wymaga mapowania dwuwymiarowych współrzędnych miejsca, w którym znajduje się mysz w oknie w momencie kliknięcia do trójwymiarowych koordynatów wirtualnej kuli[7]. Kiedy klikniemy myszą w okno tworzony jest wektor początkowy(`ArcBall.click(&MousePt))`(Listing 3.), a następnie podczas przeciągania kursora wektor końcowy, który modyfikowany jest przez funkcje `ArcBall.drag(&MousePt, &ThisQuat)`. Mając oba wektory możemy obliczyć wektor prostopadły do wektora początkowego i końcowego wokół, którego wykonywany jest obrót. Te informacje wystarczą nam do stworzenia macierzy obrotu.



Rys. 1. Program GrafDyn

Obiekt implementujący system rotacji sferycznej inicjowany jest przy pomocy następującego konstruktora.

```
ArcBall_t::ArcBall_t(GLfloat NewWidth, GLfloat NewHeight)
```

gdzie `NewWidth` i `NewHeight` odpowiadają szerokości i wysokości obszaru roboczego okna, a dokładniej tej części okna, która zarezerwowana jest na potrzeby OpenGL (*viewport*). W programie inicjuję go wprowadzając następujące wartości:

```
ArcBall(640.0f, 480.0f);
```

Nie są to prawdziwe rozmiary okna (te nie są jeszcze w momencie tworzenie instancji `ArcBall` znane), dlatego po jego utworzeniu konieczne jest przesłanie prawdziwych danych. To samo należy robić za każdym razem, kiedy jego rozmiar się zmienia. Uaktualnienia danych dokonujemy za pomocą funkcji:

```
void ArcBall_t::setBounds(GLfloat NewWidth, GLfloat NewHeight)
```

Listing 1. Funkcja ReSizeGLScene – wywoływana po zmianie rozmiarów formy

```
void __fastcall TGLForm::ReSizeGLScene(GLsizei width, GLsizei height)
{
        if (height==0)         // zapobiegnij dzieleniu przez zero...
        {
                height=1;         // ...ustawiając liczbę 1
        }
        glViewport(0, 0, width, height);        // zresetuj pole widzenia
        glMatrixMode(GL_PROJECTION) ;
        glLoadIdentity();         // zresetuj ją
        // oblicz perspektywę dla okna
        gluPerspective(45.0f,(GLfloat)width/(GLfloat)height,0.1f,100.0f);
        glMatrixMode(GL_MODELVIEW);         // wybierz macierz modeli
        glLoadIdentity();         // zresetuj ją
        ArcBall.setBounds((GLfloat)width, (GLfloat)height);
}
```

Metoda `ArcBall::setBounds` wywoływana jest w funkcji `ReSizeGLScene` (Listing 1.), gdzie jako argumenty podawane są nowe rozmiary okna. Jednocześnie w funkcji tej ustawiamy perspektywę do rozmiarów okna w momencie rozciągnięcia okna (perspektywa będzie omówiona szerzej w dalszej części pracy). Znaczy to, że to co jest dalej, jest mniejsze i jednocześnie stworzy to realistyczną scenę [7] (lekcja 1).

Aby program reagował na każdą zmianę rozmiaru okna, musimy skojarzyć określone zdarzenie z odpowiednim komunikatem Windows. Służy do tego procedura obsługi komunikatów nazywana zwykle `WndProc`. To w niej umieszczamy zbiór instrukcji wykonywanych w reakcji na odebranie komunikatu `WM_SIZING` (Listing 2.), który informuje o zmianie rozmiaru okna.

Listing 2. Sekcja instrukcji switch w WndProc, która odpowiedzialna jest za obsługę komunikatu WM_SIZING

```
switch(Message.Msg)
{
      case WM_SIZING:
                        ReSizeGLScene(ClientWidth,ClientHeight);
      …
}
```

Listing 3. Trzy macierze odpowiedzialne za obrót oraz funkcja UpdateTransform

```
Matrix4fT   Transform   = {  1.0f,  0.0f,  0.0f,  0.0f,
                               0.0f,  1.0f,  0.0f,  0.0f,
                               0.0f,  0.0f,  1.0f,  0.0f,
                               0.0f,  0.0f,  0.0f,  1.0f };

Matrix3fT   LastRot     = {  1.0f,  0.0f,  0.0f,
                               0.0f,  1.0f,  0.0f,
                               0.0f,  0.0f,  1.0f };

Matrix3fT   ThisRot     = {  1.0f,  0.0f,  0.0f,
                               0.0f,  1.0f,  0.0f,
                               0.0f,  0.0f,  1.0f };
ArcBallT    ArcBall(640.0f, 480.0f);
Point2fT    MousePt;
```

```
bool        isClicked  = false;
bool        isDragging = false;


void UpdateTransform()
{

   if (!isDragging)
    {
        if (isClicked)
        {
            isDragging = true;
            LastRot = ThisRot;
            ArcBall.click(&MousePt);
        }
    }
    else
    {
        if (isClicked)
        {
            Quat4fT     ThisQuat;

            ArcBall.drag(&MousePt, &ThisQuat);
            Matrix3fSetRotationFromQuat4f(&ThisRot, &ThisQuat);
            Matrix3fMulMatrix3f(&ThisRot, &LastRot);
            Matrix4fSetRotationFromMatrix3f(&Transform, &ThisRot);
        }
        else
            isDragging = false;
    }
}
```

W programie zainicjowane są trzy macierze odpowiedzialne za nasz obrót. `Transform` to macierz 4x4 przechowująca informacje o obrocie układu odniesienia sceny i wszelkich translacjach jakich dokonamy myszą. Druga macierz 3x3 `LastRot` zapamiętuje ostatni obrót, jaki uzyskaliśmy po ostatnim przeciąganiu myszką. Ostatnią macierzą jest `ThisRot` przechowująca obrót podczas bieżącego przeciągania. Wszystkie te macierze są inicjowane jako macierze jednostkowe.

Współrzędne punktu wskazywanego przez mysz przechowuje zmienna `MousePt` typu `Point2fT`. `MousePt` jest uaktualniane za każdym razem, gdy poruszymy myszą przy jednoczesnym przytrzymaniu jej przycisku. Do obsługi myszki użyjemy jeszcze dwóch dodatkowych zmiennych `isClicked` i `isDragging`. Obie inicjalizujemy wartościami `false`. Pierwsza odpowiada za to czy wcisnęliśmy przycisk myszki, natomiast druga sprawdza czy aktywne jest przeciąganie.

Kliknięcie okna lewym przyciskiem rozpoczyna cały proces. W tym momencie macierz obrotu ustawiana jest na macierz jednostkową. W trakcie zmiany pozycji kursora na bieżąco obliczany jest obrót od początkowego punktu kliknięcia do aktualnych współrzędnych myszy.

W metodzie `ArcBall::UpdateTransform` (Listing 3.), zaimplementowany jest algorytm obrotów naszego obiektu. Podczas kliknięcia myszką, w `LastRot` zapamiętywany jest wynik wszystkich obrotów, natomiast w `ThisRot` tylko aktualny. Podczas przeciągania kursora myszki (`isDragging = true`), `ThisRot` modyfikowany jest najpierw przez oryginalny obrót, a następnie przez `LastRot`. Proces ten ilustrują poniższe kroki:

- `ArcBall.drag(&MousePt, &ThisQuat)` aktualizowany jest wektor końcowy i pobierany wynikowy obrót w postaci kwaternionu,
- `Matrix3fSetRotationFromQuat4f(&ThisRot, &ThisQuat)` konwertujemy kwaternion do Matrix3fT
- `Matrix3fMulMatrix3f(&ThisRot, &LastRot)` dodajemy wynik ostatniego obrotu
- `Matrix4fSetRotationFromMatrix3f(&Transform, &ThisRot)` ustawiamy nasz ostateczny obrót

Kiedy kończymy przeciąganie myszą (`isDragging = false`), do macierzy `LastRot` zostaje skopiowana wartość `ThisRot`.

Bardzo ważne w całym algorytmie rotacji sferycznej jest to, że zapamiętujemy ostatni obrót, jaki otrzymaliśmy pod koniec przeciągania (`LastRot`). Gdybyśmy tego nie robili, po każdym kliknięciu model zaczynałby obrót od wyjściowej pozycji. Przykładowo, jeśli obrócimy model wokół osi X o 50 stopni, a potem o 25 stopni w tym samym kierunku, chcielibyśmy otrzymać obrót o 75 stopni, a nie o ostatnie 25.

Kiedy już mam gotowy algorytm rotacji sferycznej, nadszedł czas umiejscowić odpowiednie funkcje w kodzie programu, abyśmy mogli swobodnie obracać nasz model. Na początku zajmiemy się zmienną `isClicked`. Chcemy aby program za każdym razem podczas poruszania myszką, sprawdzał czy został wciśnięty lewy przycisk myszki i do naszej zmiennej przypisywał odpowiednią wartość. W tym celu musimy skojarzyć określone zdarzenie (w naszym przypadku poruszanie myszką) z odpowiednim komunikatem Windows. Zatem w `WndProc` umieszczamy zbiór instrukcji wykonywanych w reakcji na odebranie komunikatu `WM_MOUSEMOVE` (Listing 4.), który informuje o ruchu myszy w obrębie okna.

Listing 4. Sekcja instrukcji switch w WndProc, która odpowiedzialna jest za obsługę komunikatu WM_MOUSEMOVE

```
case WM_MOUSEMOVE
      isClicked   = (LOWORD(Message.WParam) & MK_LBUTTON) ? true : false;
```

Aby wszystko działało poprawnie należy użyć nowej macierz transformacji (przed renderowaniem wykresu). Tylko wówczas transformacja będzie wpływać na jego położenie. W klasie `GLForm` implementującej główne okno aplikacji zdefiniowana jest funkcja `GL_RysujScene` (Listing 5.), która odpowiedzialna jest za generowanie sceny. To w niej właśnie rysujemy nasz aktualny wykres, światła i legendę. Chcemy aby rotacja obejmowało tylko wykres i światła, więc tuż przed nimi wywołujemy metodę `ArcBall::UpdateTransform` i `glMultMatrixf(Transform.M)`. Zwróćmy uwagę, że w metodzie `GL_RysujScene` druga funkcja umieszcza bieżącą macierz model-widok na stosie macierzy (zapamiętuje w ten sposób jej wartość). Po zakończeniu renderowania modelu przywracamy zapamiętany stan ściągając go ze stosu.

Listing 5. Kod programu odpowiedzialny za transformacje w funkcji GL_RysujScene

```
void __fastcall TGLForm::GL_RysujScene()
{
      .....
      UpdateTransform()
```

```
        glPushMatrix();
        glMultMatrixf(Transform.M);
        ……//   renderujemy model podlegający rotacji
        glPopMatrix();
}
```

W ten sposób zakończyliśmy implementację rotacji sferycznej w programie. Postępując zgodnie z opisanymi powyżej krokami możemy użyć jej także w innych projektach korzystających z OpenGL.

2.2 Przeciąganie obiektów na scenie za pomocą myszy

Biblioteka OpenGL zawiera funkcje, które pozwalają stworzyć odpowiedni algorytm do przesuwania obiektów za pomocą myszy. W tym rozdziale, na przykładzie źródeł światła jako obiektów, postaram się go dokładnie opisać.

W programie GrafDyn, każde nowo dodane źródło światła posiada swoją określoną pozycję (*x,y,z*), które początkowo ustawione jest na (0,0,0). Mamy możliwość zmiany ustawienia położenia światła za pomocą formularza umieszczonego w oknie `Oświetlenie` (opis w rozdziale 5.4). Wygodniejszą metodą zmiany jego pozycji jest jednak przeciąganie go za pomocą myszy w przestrzeni 3D. Jednak zanim przystąpię do opisania algorytmu, który zaimplementowałem w tym celu w programie GrafDyn, opiszę dwie użyte do tego funkcje udostępniane przez bibliotekę GLU dołączaną do OpenGL. Pierwsza z nich to:

```
gluProject(x,y,z,model_view, projection, viewport,
                &winX, &winY, &winZ);
```

Służy do przeliczenia współrzędnych w przestrzeni widoku (*x, y, z*) na współrzędne w przestrzeni okna (*winX,winY,winZ*)[14]. Wzory na podstawie którego następuje przeliczenie to (3):

```
v=(x,y,z,1.0)
v' = P x M x v
winX = viewport(0) + viewport(2) * (v'(0) + 1) / 2      (3)
winY = viewport(1) + viewport(3) * (v'(1) + 1) / 2
```

```
winZ = (v'(2)  + 1) / 2
```

gdzie M to macierz widoku, a P to macierz rzutowania.
Przeciwieństwem tej funkcji jest:

```
gluUnProject(winX, winY, winZ, model_view, projection,
        viewport, &pos3D_x, &pos3D_y, &pos3D_z);
```

Służy do przeliczenia współrzędnych w przestrzeni okna (*winX,winY,winZ*) na współrzędne w przestrzeni widoku (*pos3D_x, pos3D_y, pos3D_z*)[14]. Przeliczanie to odbywa się na podstawie wzoru (4):

$$\begin{pmatrix} pos3d\_x \\ pos3d\_y \\ pos3d\_z \\ W \end{pmatrix} = INV(PM) \begin{pmatrix} \frac{2(winX - viewport[0])}{viewport[2]} - 1 \\ \frac{2(winY - viewport[1])}{viewport[3]} - 1 \\ 2(winZ) - 1 \\ 1 \end{pmatrix} \quad (4)$$

Obie funkcje do przeliczenia współrzędnych wykorzystują macierz modelowania (parametr `model_view`), macierz rzutowania (parametr `projection`) oraz współrzędne okna renderingu (parametr `viewport`). Macierz modelowania mówi nam jak zmienić układ odniesienia sceny na układ odniesienia kamery. Druga macierz (macierz rzutowania) wprowadza do obrazu perspektywę [16].

Algorytm przesuwania świateł zaszyty jest w metodzie `MoveLight` (Listing 6.), który jako argumenty przyjmuje bieżącą pozycję myszy na ekranie. Funkcja ta wywoływana jest w `GL_RysujScene()`, czyli za każdym razem kiedy jest generowana scena.

Listing 6. Funkcja MoveLight

```
void __fastcall TGLGraf::MoveLight(int xPos, int yPos)
{

      int nrLight = NrLightMove-1;
      if(HitsMove == 1 || HitsMove == 2)
      {
            if(nrLight >= 0 && nrLight <= 6)
```

```
{
  if(clickMouseMove)
  {
      float x = tabPropertiesLight[nrLight].GetPosLight(0);
      float y = tabPropertiesLight[nrLight].GetPosLight(1);
      float z = tabPropertiesLight[nrLight].GetPosLight(2);
      GLdouble pos3D_x, pos3D_y, pos3D_z;
      GLdouble winX, winY, winZ;
      GLdouble model_view[16];

      glGetDoublev(GL_MODELVIEW_MATRIX, model_view);

       GLdouble projection[16];
       glGetDoublev(GL_PROJECTION_MATRIX, projection);

       GLint viewport[4];
       glGetIntegerv(GL_VIEWPORT, viewport);

       //szerokość i wysokść obszaru renderingu
        int width = viewport[2];
        int height = viewport[3];

        gluProject(x,y,z,model_view, projection, viewport,
        &winX, &winY, &winZ);

        winX=xPos;
        winY = (float)viewport[3] - (float)yPos;


      gluUnProject(winX, winY, winZ,model_view, projection,
      viewport, &pos3D_x, &pos3D_y, &pos3D_z);


      tabPropertiesLight[nrLight].SetPosLight(0,(float)pos3D_
      x);

      tabPropertiesLight[nrLight].SetPosLight(1,(float)pos3D_
      y);

      tabPropertiesLight[nrLight].SetPosLight(2,(float)pos3D_
      z);
```

```
                SetPositionLight((float)pos3D_x, (float)pos3D_y,
                (float)pos3D_z);
              }
            }
            else
                clickMouseMove = false;
          }
          else
             clickMouseMove = false;

    }
```

W `MoveLight` początkowo sprawdzamy czy zostało wybrane któreś ze zdefiniowanych źródeł światła (wybór czyli selekcja źródła światła jest opisana w rozdziale 3). Gdy zaznaczyliśmy jedno ze źródeł światła myszą, zmienna `clickMouseMove` przyjmuje wartość true. Jednocześnie zmienna `isClicked` zmienia się na false co nie pozwala na rotację modelu (wykresu, osi itd.). Zapamiętujemy bieżące położenie źródła światła (*x,y,z*). Następnie za pomocą funkcji `glGetDoublev` pobieramy macierz widoku modelu (używana do rozmieszczenia elementów na scenie) oraz macierz rzutowania (używanego do definiowania bryły obcinania). Metoda `glGetIntegerv` pomoże nam pobrać aktualne współrzędne okna. Gdy wszystkie parametry są już gotowe, przeliczamy współrzędne (*x,y,z*) bieżącego światła na współrzędne w przestrzeni okna (*winX,winY,winZ*). Następnie przypisujemy do zmiennych *winX* i *winY* bieżącą pozycje myszki. Pamiętać należy że współrzędne okna w Windows mają punkt (0,0) w górnym lewym rogu, natomiast w OpenGL ma początek układu współrzędnych jest w dolnym lewym rogu. Dlatego należy wykonać następującą operację:

```
winY = (float)viewport[3] - (float)yPos;
```

Kolejnym krokiem jest przeliczenie współrzędnych przestrzeni okna (*winX,winY,winZ*) na współrzędne w przestrzeni widoku (*pos3D_x, pos3D_y, pos3D_z*) i zapamiętanie aktualnej pozycji światła. W efekcie możemy swobodnie ustalać pozycję źródeł światła w przestrzeni 3D klikając na wybrane źródło i przeciągając je.

3. Selekcja jako sposób zaznaczania obiektów

3.1. Selekcja

OpenGL pozwala programiście tworzyć grafikę trójwymiarową. Uwzględnia również możliwość interakcji użytkownika z zawartością sceny włączając w to zaznaczanie obiektów. Selekcja, bardzo użyteczny element OpenGL, umożliwia kliknięcie myszką w pewne miejsce okna i wyznaczanie obiektu, w obrębie którego nastąpiło to kliknięcie. Zaznaczenie konkretnego obiektu nazywamy wybraniem. Tworzona jest wtedy specjalna matryca, oparta wyłącznie na współrzędnych ekranu i rozmiarach w pikselach. Za pomocą tej matrycy można stworzyć bryłę widzenia z wierzchołkiem w położeniu kursora myszy. Następnie można użyć selekcji do sprawdzenia, jakie obiekty należą do owej bryły widzenia.

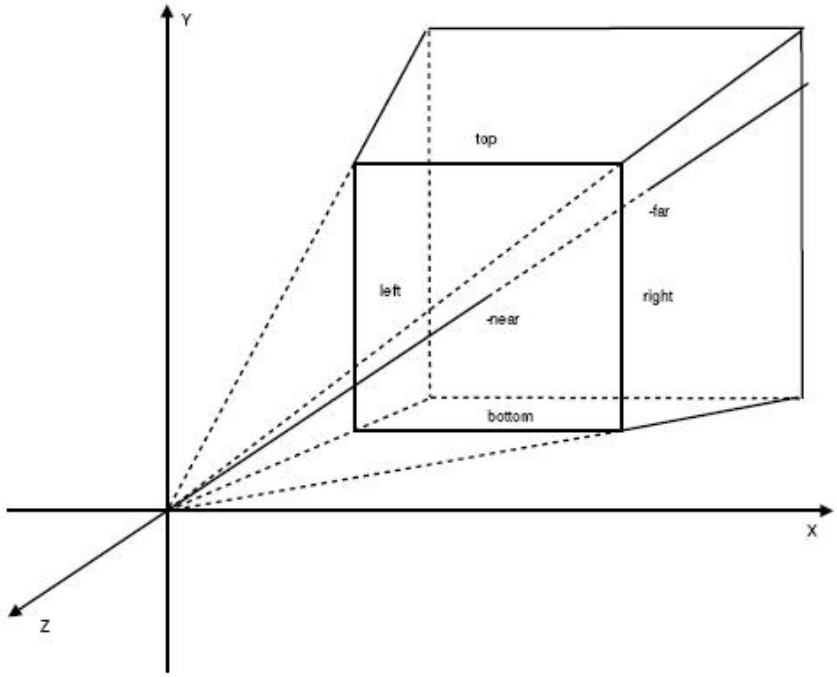
3.2. Tryby renderowania

Tryb selekcji jest jednym z trzech trybów renderingu dostępnych w bibliotece OpenGL. W trybie tym żadne piksele nie są kopiowane do bufora ramki. Zamiast tego tworzona jest lista prymitywów [Dodatek A] których wierzchołki przecinają bryłę widzenia [15].

Bryła widzenia stanowi ograniczenie sceny 3D, w związku z tym obiekty znajdujące się poza obszarem widzenia nie są rysowane, a te które ja przecinają są rysowane tylko częściowo. Z obszarem widzenia ściśle związane jest rzutowanie. Określamy go jako odwzorowanie zawartości trójwymiarowej sceny graficznej na płaskim ekranie monitora [15]. Biblioteka OpenGL oferuje standardowo dwie metody rzutowania: rzutowanie prostokątne i rzutowanie perspektywiczne. W programie GrafDyn zostało użyte rzutowanie perspektywiczne, które pozwala na ocenę odległości przedmiotów od kamery, a tym samym daje bardziej realistyczny efekt. Bryła odcinania w tym przypadku ma postać ściętego ostrosłupa (Rys. 2.). Jej parametry ustawiamy za pomocą funkcji:

```
void glFrustum (GLdouble left, GLdouble right, GLdouble
bottom, GLdouble top, GLdouble near, GLdouble far)
```

Parametry `left`, `right`, `bottom` i `top` wyznaczają rozmiary górnej podstawy bryły odcinania, a `near` i `far` wyznaczają położenie odpowiednio górnej i dolnej ściany ostrosłupa. Funkcja ta mnoży jednostkową macierz rzutowania przez macierz perspektywy wygenerowaną na podstawie argumentów funkcji, generując w efekcie rzut perspektywiczny[16].



Rys. 2. Bryła widzenia w rzutowaniu perspektywicznym

W programie GrafDyn wykorzystuję tryb selekcji do wybrania określonych obiektów (źródła światła) rysowanych w bryle widzenia. W trybie tym wykonywane są te przekształcenia sceny, które wykonuje się w trybie renderowania, aby klikając w dany obiekt trafić w to samo miejsce, w którym został on narysowany. W buforze selekcji tworzone są rekordy trafień. Aby móc z niego korzystać należy wcześniej go przygotować i ponadawać nazwy prymitywom lub grupie prymitywów (obiektom) w celu ich identyfikacji w buforze. Częstą praktyką jest określanie bryły widzenia odpowiadającą wskaźnikowi myszy, a następnie sprawdza się, który z nazwanych obiektów jest przez nią wskazywany.

Kolejnym i jednocześnie domyślnym trybem w bibliotece OpenGL jest tryb renderowania. W odróżnieniu od selekcji, wykonywane tu operacje graficzne pojawiają się

na ekranie. Ostatnim trybem jest tryb sprzężenia zwrotnego. Różni się on od selekcji tym, że w buforze sprzężenia zwrotnego tworzone są dane o współrzędnych wierzchołków prymitywów, składowych kolorów oraz współrzędnych tekstur. Można użyć tych informacji do przesyłania rezultatów renderowania przez sieć, rysowania sceny za pomocą plotera czy też łączenia obiektów OpenGL z grafika GDI[1].

Aby móc korzystać z selekcji obiektów w bibliotece OpenGL trzeba zmienić tryb renderingu. Do zmiany trybu służy funkcja:

```
GLint glRenderMode (GLenum mode)
```

której parametr `mode` przyjmować może jedną z następujących wartości:

- GL_RENDER – tryb renderowania,
- GL_SELECT – tryb selekcji,
- GL_FEEDBACK – tryb sprzężenia zwrotnego

Wartość zwracana przez funkcję `glRenderMode` zależy od tryby renderingu. W przypadku trybu renderowania funkcja zwraca wartość 0, a w trybie selekcji zwraca ilość rekordów trafień znajdujących się w buforze selekcji.

3.3. Stos nazw obiektów

Nazwy obiektów, które mogą zostać wybrane w trakcie selekcji przechowywane są na stosie jako cyfrowe identyfikatory w postaci liczb całkowitych bez znaku. Po zainicjowaniu stosu można odkładać na niego nazwy. Gdy podczas selekcji nastąpi trafienie, wszystkie nazwy ze stosu są kopiowane do bufora selekcji. W związku z tym pojedyncze trafienie może zwrócić więcej niż jedną nazwę.
Funkcja:

```
glInitNames()
```

jest odpowiedzialna za inicjalizacje pustego stosu nazw obiektów. Nazwy poszczególnym obiektom lub grupie obiektów, które mają być narysowane, nadaje się wywołując funkcję:

```
glLoadName(GLuint name),
```

gdzie parametr `name` jest unikatowym identyfikatorem obiektu. Jednocześnie nazwy w postaci liczb całkowitych bez znaku, nie są odkładane na stos nazw, lecz zastępują bieżącą nazwę na szczycie stosu. Funkcję tę należy wywołać tuż przed narysowaniem prymitywu, któremu właśnie nadajemy nazwę (Listing 7.).

Czasem jednak w programie chcemy mieć informacje o tym że trafiony został obiekt związany z innym obiektem. W takim przypadku ich nazwy należy umieścić na bieżącym stosie przy użyciu funkcji:

```
void glPushName(GLuint name),
```

gdzie parametr `name` jest unikatowym identyfikatorem obiektu. W ten sposób możemy wykrywać selekcje obiektów o hierarchicznej strukturze, ponieważ w na stosie mamy kilka nazw, które po trafieniu zostaną umieszczone w buforze (W moim kodzie nie używam tej funkcji gdyż źródło światła składa się z jednego elementu (kula)). Na sam koniec trzeba zdjąć nazwę obiektu ze stosu, co wymaga użycia funkcji:

```
void glPopName();
```

Wielkość stosu nazw nie może być mniejsza niż 64, a niedomiar lub przepełnienie stosu powoduje zgłoszenie błędów `GL_STACK_UNDERFLOW` i `GL_STACK_OVERFLOW` [15].

Listing 7. Funkcja RenderLights

```
void __fastcall TForm1::RenderLights(void)
{
        bool enabled;

       // inicjalizacja stosu nazw obiektów
        glInitNames();

      // odkłada na stos identyfikator 0 w celu wypełnienia stosu
      przynajmniej jedną nazwą
```

```
glPushName(0);

  for(int nrLight = 0; nrLight<lengthTabLight; nrLight++)
  {
          if(existLight[nrLight])
          {
                  enabled = glIsEnabled(tabLight[nrLight]);
                  glPushMatrix();
                  if(enabled)
                          glColor3f(1.0f,1.0f,1.0f);
                  else
                          glColor3f(1.0f,0.0f,0.0f);
                  glLoadName(nrLight+1);
                  RysujLampe(nrLight);

                  glPopMatrix();
          }
  }


}
```

W funkcji `RenderLights(void)` (Listing 7.) nadawane są nazwy tylko tym źródłom światła które są zdefiniowane . Dodatkowo włączone światła otrzymują kolor biały, a wyłączone kolor czerwony.

3.4. Bufor selekcji – przetwarzanie rekordów trafień

Podczas renderowania bufor selekcji jest wypełniany rekordami trafień. Rekord trafienia jest generowany za każdym razem, gdy renderowany prymityw przecina bryłę widzenia[1]. Aby móc korzystać z selekcji obiektów należy ustawić odpowiednio bufor selekcji. Wymaga to wywołania funkcji:

```
void glSelectBuffer(GLsize size, GLuint *buffer)
```

gdzie parametr `size` określa wielkość bufora, do którego wskaźnik zawiera parametr `buffer`. Bufor selekcji to tablica liczb całkowitych bez znaku. Powinien być on na tyle duży, aby zmieścić informacje o wszystkich obiektach wybranych w wyniku selekcji [15]. Bardzo ważne jest aby ustawić bufor selekcji przed przełączeniem biblioteki OpenGL w

tryb selekcji. Każdy rekord selekcji zajmuje przynajmniej cztery elementy tablicy (Listing 8.). Pierwsza pozycja tablicy zawiera ilość nazw na stosie w momencie wystąpienia trafienia. Kolejne pozycje zawierają minimalną i maksymalną wartość współrzędnych *z* prymitywów graficznych wchodzących w skład wybranego obiektu. Te wartości z zakresu <0 ,1>, są pomnożone prze $2^{32} - 1$ i zaokrąglone do najbliższej liczby całkowitej.

Listing 8. Bufor selekcji

```
Bufor selekcji [0] – ilość nazw na stosie nazw w momencie trafienia
Bufor selekcji [1] – minimalna wartość współrzędnych Z
Bufor selekcji [2] – maksymalna wartości współrzędnych Z
Bufor selekcji [3] – najniższy element stosu nazw obiektów
.
.
.
Bufor selekcji [n] – najwyższy element sotsu nazw obiektów
```

Trzeba pamiętać, że z bufora selekcji nie można wyciągnąć informacji o ilości rekordów trafień. Jest tak ponieważ bufor selekcji nie jest wypełniany, aż do momentu powrotu do trybu `GL_RENDER`. Ilość wybranych obiektów zwraca funkcja `glRenderMode` w momencie powrotu do domyślnego trybu renderowania, czyli po zakończeniu pracy w trybie selekcji.

3.5. Wybieranie obiektów

W programie możemy stworzyć siedem niezależnych źródeł światła, ale tylko aktywne poddawane są selekcji. Wybieranie następuje wtedy, gdy podczas selekcji wykorzystuje się położnie myszy do utworzenia bryły obcinania. Gdy utworzę mniejszą bryłę widzenia znajdującą się na scenie w miejscu wskaźnika myszy, generowane będą trafienia dla tych obiektów, które przecinają bryłę widzenia. Następnie sprawdzam zawartość bufora selekcji i mogę prześledzić na których obiektach nastąpiło kliknięcie myszą. Aby ograniczyć bryłę widzenia (Listing 9.), tak aby obejmowała bliską okolicę punktu, w którym został naciśnięty przycisk myszki najlepiej wykorzystać funkcje z biblioteki GLU:

```
void gluPickMatrix( GLdouble x, GLdouble y, GLdouble width,
GLdouble height, const GLint viewport[4])
```

Parametry x i y określają współrzędne środka nowej bryły obcinania (współrzędne okienkowe), w których wyrażona jest pozycja myszy. Kolejne parametry `width` i `height` określają szerokość i wysokość bryły widzenia w pikselach okna [1]. Jeżeli chcemy klikać w pobliżu obiektu trzeba użyć większych wartości, jednak w moim przypadku kliknięcia będą bezpośrednio na obiektach, dlatego też użyłem mniejszych wartości. Tablica `viewport` zawiera dane opisujące aktualne rozmiary okna. Dane te najlepiej jest pobrać wywołując funkcję `glGetIntegerv` z parametrem `GL_VIEWPORT` (`glGetIntegerv(GL_VIEWPORT, viewport)`).

.

Listing 9. Funkcja ProcessSelection

```
void __fastcall TGLForm::ProcessSelection(int xPos, int yPos)
{

        int hits;

        // buffor selekcji
        GLuint selectBuff[BUFFER_LENGTH];

        //przygotowanie buffora selekcji
        glSelectBuffer(BUFFER_LENGTH, selectBuff);

        // pobranie obszaru renderingu
       // int viewport[4];
       // glGetIntegerv(GL_VIEWPORT, viewport);

        //szerokość i wysokść obszaru renderingu
        int width = viewport[2];
        int height = viewport[3];

        // wybor macierzy rzutowania
        glMatrixMode(GL_PROJECTION);
```

```
    // odlozenie macierzy rzutowania na stos
    glPushMatrix();

    // wlaczenie trybu selekcji
    glRenderMode(GL_SELECT);

    // marierz rzutowania = macierz jednostkowa
    glLoadIdentity();

    // parametry bryly obcinania (jednostkowa kostka)
    gluPickMatrix(xPos,height - yPos, 2, 2, viewport);

    //parametyr bryly obcinania
    float wsp=(float)height/(float)width ;
    glFrustum(-0.1, 0.1, wsp*-0.1, wsp*0.1, 0.3, 1000.0);

    // generowanie sceny 3D
    GL_RysujScene();


  // zliczanie ilości rekordow trafien i powrot do domyslnego trybu
  renderowania
    hits = glRenderMode(GL_RENDER);

    // wybor macierzy rzutowania
    glMatrixMode(GL_PROJECTION);

    // zdjecie macierzy rzutowania ze stosu
    glPopMatrix();


    glMatrixMode(GL_MODELVIEW);
    if(hits)
            MakeSelection(selectBuff, hits);


}
```

Zanim użyjemy jednak funkcji `gluPickMatrix`, musimy przechować bieżący stan macierzy rzutowania na stosie. Następnie trzeba załadować do macierzy rzutowania

macierz jednostkową poprzez wywołanie funkcji `glLoadIdentity()`. Dopiero teraz możemy zmodyfikować nową bryłę widzenia tak aby obejmowała jedynie obszar pod wskaźnikiem myszki. Bardzo ważne jest aby ustawić macierz rzutowania perspektywicznego `glFrustum` takie jakie zostało użyte w oryginalnej scenie gdyż w przeciwnym razie nie otrzymamy właściwego odwzorowania[1]. Następnie wywołujemy `GL_RysujScene()`, gdzie wykonywany jest etap renderingu sceny. Po odrysowaniu sceny ponownie wywołujemy funkcje `glRenderMode(GL_RENDER)` w celu przełączenia OpenGL do trybu renderowania. Po powrocie do tego trybu otrzymujemy ilość wygenerowanych rekordów trafień i wypełniony bufor selekcji odpowiednimi danymi.

```
hits = glRenderMode(GL_RENDER)
```

Po ustaleniu ilości rekordów trafień trzeba dokonać analizy zawartości bufora selekcji. W tym celu do naszej funkcji `MakeSelection` (Listing 10.) przekazujemy tablice `selectBuff` oraz zmienną `hits` przechowującą ilość rekordów trafień.

Listing 10. Funkcja MakeSelection

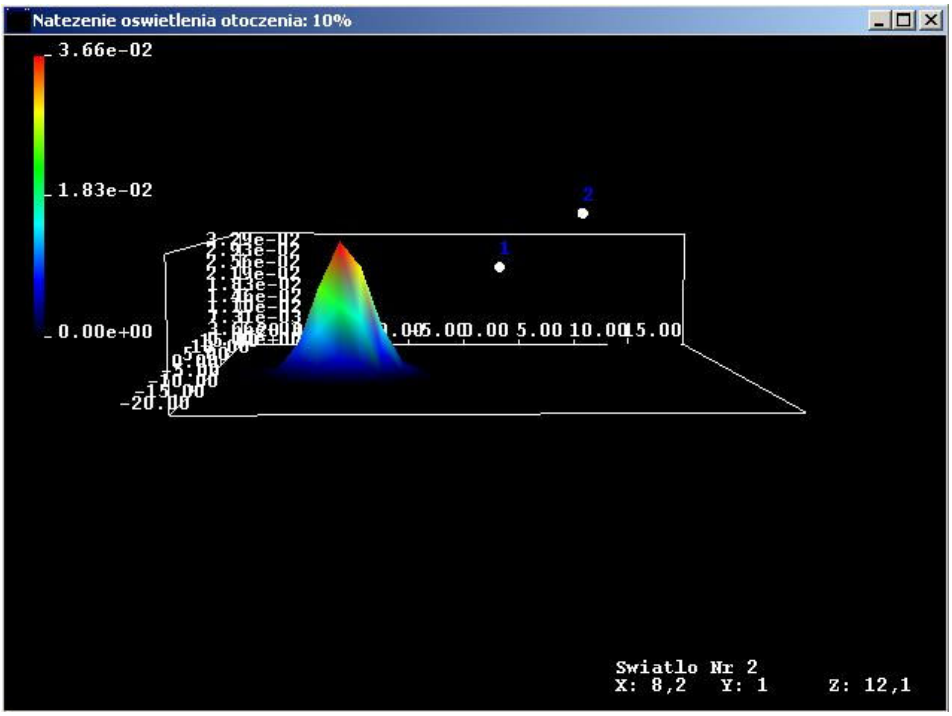
```
void __fastcall TGLGraf::MakeSelection(GLuint select, int hits)
{

  NrLightMove = select;
  HitsMove = hits;
  AnsiString NrLight = IntToStr(select);
  if(hits == 1)
  {
        if(select >= 1 && select <= 7)
        {
                lightStr = "Swiatlo Nr "+ NrLight;
                GLOswietlenieForm->ChangeLight(select-1);
                clickMouseMove = true;
        }
  }
  if(hits == 2)
  {
        if(select >= 1 && select <= 7)
```

```
        {
                lightStr = "Swiatlo Nr "+ NrLight;
                GLOswietlenieForm->ChangeLight(select-1);
                clickMouseMove = true;
        }
    }
    GL_RysujScene();

}
```

Funkcja `MakeSelection` wyświetla w prawym dolnym rogu numer światła, na które nastąpiło kliknięcie, jego aktualną pozycję oraz przekazuje do programu wszystkie właściwości związane z tym światłem(Rys.3.).



Rys. 3. Wykres z zaznaczonym źródłem światła numer 2

W momencie kliknięcia lewym przyciskiem myszki wywoływany jest komunikat `WM_LBUTTONDOWN`, który odpowiedzialny jest za przekazanie odpowiednich współrzędnych myszy (przechowywanych w parametru LParam) do aplikacji. Są one przekazywane dalej do funkcji `ProcessSelection` (Listing 11.). W programie następuje wówczas przetwarzanie kliknięcia myszki.

Listing 11. Odbiór komunikatu WM_LBUTTONDOWN

```
case WM_LBUTTONDOWN:
                  X0=LOWORD(Message.LParam); //biezaca poz. x kursora
                  Y0=HIWORD(Message.LParam); //biezaca poz. y kursora
                   ProcessSelection(X0,Y0);
```

4. Sprzężenie zwrotne

W tym rozdziale omówię sprzężenie zwrotne, które podobnie jak selekcja jest trybem renderowania, w którym nic faktycznie nie jest rysowane na ekranie. Sprzężenie zwrotne nie jest zaimplementowane w naszym programie, ale warto o nim wspomnieć aby lepiej zrozumieć sposób wybierania obiektów dostępnych i dopełnić informacje o trybach renderowania. Swoją budową bufor sprzężenia zwrotnego bardzo przypomina bufor selekcji, z tą jednak różnicą, że zamiast tablicy liczb całkowitych mamy do czynienia z tablicą liczb zmiennoprzecinkowych typu `GLfloat` [15]. Wpisywane są tam informacje na temat sposobu renderowania sceny.

4.1. Tryb renderowania

Aby móc skorzystać ze sprzężenia zwrotnego należy zmienić tryb renderowania. Podobnie jak w trybie selekcji wykonuję się to poprzez wywołanie funkcji `glRenderMode`. Tym razem z parametrem GL_FEEDBACK. Aby wypełnić bufor sprzężenia zwrotnego i powrócić do domyślnego trybu renderowania należy wywołać funkcję `glReneder Mode(GL_RENDER)`.

4.2. Buffor sprzeżenia zwrotnego

Za utworzenie bufora sprzężenia zwrotnego odpowiedzialna jest funkcja:

```
Void glFeedbackBuffer(GLsizei size, GLenum type, GLfloat
*Buffet)
```

gdzie parametr `size` określa wielkość tablicy wskazywanej w parametrze `buffer`, a rodzaj danych przekazywanych do bufora zwrotnego określa parametr `type` [15]. Dostępne wartości dla typu danych zostały określone w tabeli (Tab.1.).

Typ	Współrzędne wierzchołka	Składowe koloru	Współrzędne tekstury
GL_2D	(x,y)	-	-
GL_3D	(x,y,z)	-	-
GL_3D_COLOR	(x,y,z)	(R,G,B,A)	-
GL_3D_COLOR_TEXTURE	(x,y,z)	(R,G,B,A)	(s,t,r,q)
GL_4D_COLOR_TEXTURE	(x,y,z,w)	(R,G,B,A)	(s,t,r,q)

Tab.1. Typy danych zwracanych w buforze sprzężenia zwrotnego.

W OpenGL dane związane z kolorem reprezentowane są przez cztery składowe RGBA albo pojedyncza wartość – numer indeksu koloru. Współrzędne x, y, z położenia wierzchołków podawane są w odniesieniu do współrzędnych renderingu.

4.3. Dane bufora sprzężenia zwrotnego

W buforze sprzężenia zwrotnego mieszczą się specjalne znaczniki, które oddzielają dane. Znaczniki identyfikowane są przez następujące stałe[1]:

- `GL_POINT_TOKEN` – punkty,
- `GL_LINE_TOKEN` – linie,
- `GL_LINE_RESET_TOKEN` – segment linii po wyzerowaniu wzorca linii
- `GL_POLYGON_TOKEN` – wielokąt,
- `GL_BITMAP_TOKEN` – bitmapa,
- `GL_DRAW_PIXEL_TOKEN` – mapa pikselowa
- `GL_COPY_PIXEL_TOKEN` – kopiowanie mapy pikselowej
- `GL_PASS_THROUGH_TOKEN` – znacznik zdefiniowany przez użytkownika.

Po znaczniku w buforze występują dane pojedynczego wierzchołka oraz ewentualnie dane koloru i tekstury [1]. Wszystko zależy od parametru `type` funkcji `glFeedbackBuffer`. W przypadku `GL_LINE_TOKEN`, `GL_LINE_RESET_TOKEN` zwracane są dwa zestawy danych wierzchołków, a po `GL_POLYGON_TOKEN` występuje wartość określająca ilość wierzchołków wielokąta. Bezpośrednio po `GL_POINT_TOKEN`, `GL_BITMAP_TOKEN`, `GL_DRAW_PIXEL_TOKEN`, `GL_COPY_PIXEL_TOKEN` będą to

dane pojedynczego wierzchołka. Poniżej przedstawiam przykładową zawartości bufora sprzężenia zwrotnego:

```
Bufor sprzężenia zwrotnego:
[0] – GL_POINT_TOKEN
[1] – Współrzędna x
[2] – Współrzędna y
[3] – Współrzędna z
[4] – GL_PASS_THROUGH_TOKEN
[5] – Dowolna wartość zdefiniowana przez użytkownika
[6] – GL_POLYGON_TOKEN
[7] – Ilość wierzchołków wielkąta
[8] – Współrzędna x pierwszego wierzchołka
[9] – Współrzędna y pierwszego wierzchołka
[10] – Współrzędna z pierwszego wierzchołka
[11] - Współrzędna x drugiego wierzchołka
…
```

W trybie sprzężenia zwrotnego można utworzyć znacznik zdefiniowany przez użytkownika. Aby to zrobić należy wywołać funkcję:

```
void glPassThrough(GLfloat token)
```

Ta funkcja umieszcza element GL_PASS_THROUGH_TOKEN w buforze selekcji, a bezpośrednio po nim wartość podana przy wywołaniu funkcji [1].

4.4. Podsumowanie

Sprzężenie zwrotne, tak jak selekcja, jest bardzo przydatnym elementem biblioteki OpenGL, służącym do identyfikacji obiektów. Zwraca informacje dotyczące położenia rysowanych prymitywów we współrzędnych okna.

5. GrafDyn – program do wizualizacji danych

Zadaniem programu GrafDyn jest wyświetlanie funkcji dwóch zmiennych zapisanych w pliku tekstowym (*raw data*). Domyślnie wczytuje on dwa rodzaje plików, choć dzięki modułowi importu możliwe jest w zasadzie wczytywanie danych z każdego rodzaju pliku tekstowego. Domyślne formaty rekordów rozpoznawane są przez program po następujących szablonach nazw plików: **.plt*, *plt*.dat* i *psi*.dat* (Listing 12.).

Listing 12. Plik typu plt*.dat

```
#Funkcja falowa w t=0 (0)
#Kwadrat modulu |Psi(x,y)|^2
#Parametry sieci x: 256 (-100,100), y: 256 (-100,100), t: 10
(0,0.0628319)
#
#x y norm(Psi(x,y))
-100  -100  1.03775e-28
-99.2157  -100  1.30337e-28
-98.4314  -100  1.63553e-28
-97.6471  -100  2.05049e-28
-96.8627  -100  2.56839e-28
```

Pierwszy i drugi zawiera wartości funkcji wraz z odpowiadającymi jej współrzędnymi x i y. W nagłówku tego pliku zawarte są parametry sieci. Nie są one jednak wykorzystywane przez program. Parametry te mogą być odczytane z osobnego pliku lub wpisane w trakcie do importu danych. Drugi typ plików (*psi*.dat*) zawiera tylko dwie kolumny danych z częścią rzeczywistą i urojoną funkcji (bez współrzędnych przestrzennych).

W programie GrafDyn został zaimplementowany moduł importu plików, który odczytuje funkcję z plików tekstowych i zapisuje do odpowiedniej tablicy. Moduł ten wczytuje również parametry sieci przestrzennej z pliku z rozszerzeniem *.par*. Struktura pliku parametrów opisana jest w listingu 13. Jego pierwsza linia zawiera wersję pliku. Z punktu widzenia naszego programu najważniejsze są linie druga i trzecia, które zawierają informacje o wielkości i zakresie sieci, a mianowicie: ilość węzłów oraz minimalną i maksymalną wartość współrzędnych dla danej osi.

Listing 13. Struktura pliku z rozszerzeniem typu par

```
Schemat:                Przykład:

wersja                  1.00000000000000e+00
N_x, xmin, xmax         256 -1.00000000000000e+02 1.00000000000000e+02
N_y, ymin, ymax         256 -1.00000000000000e+02 1.00000000000000e+02
N_t, t_min, dt          1000 0.00000000000000e+00 6.28318530717957e-03
```

Po wczytaniu pliku parametrów i pliku danych w programie pojawi sie wykres, który w zależności od naszych potrzeb możemy obracać, skalowywać, czy dobierać sposób kolorowania. Możemy także definiować źródła światła (w OpenGL jest ich maksimum osiem).

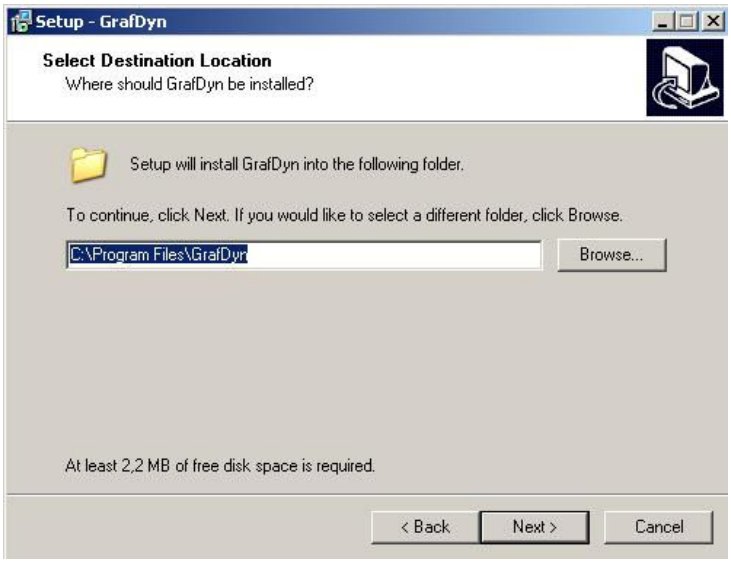
5.1. Instalacja

Aby móc korzystać z programu GrafDyn, musimy go zainstalować na naszym komputerze. Instalacja przebiega według standardowego kreatora, który krok po korku prowadzi nas przez kolejne etapy pozwalając na wybór odpowiedniej konfiguracji.

W pierwszym okienku zostanie wyświetlona informacja o tym że przystępujemy do instalacji programu GrafDyn ver. 1.0 (Rys. 4.).

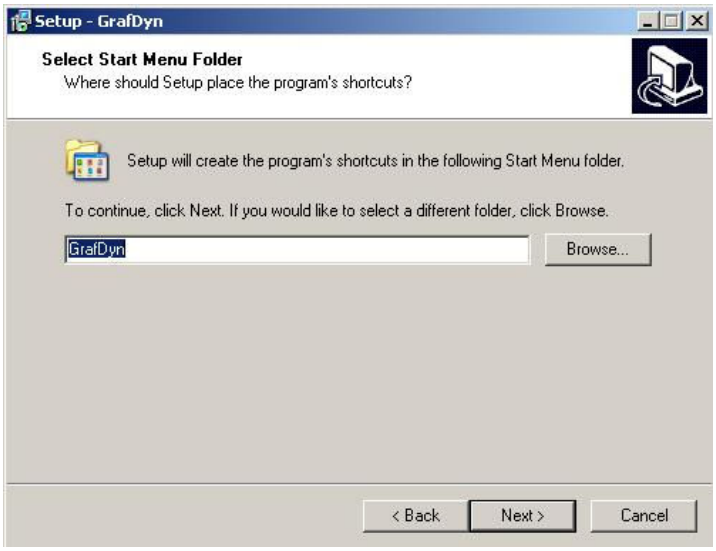
Rys. 4. Instalacja - powitanie

Po kliknięciu klawisza *Next* mamy możliwość wyboru katalogu w którym zainstaluje się program. Standardowo program instalujący proponuje *C:\Program Files\GrafDyn* (Rys. 5.).



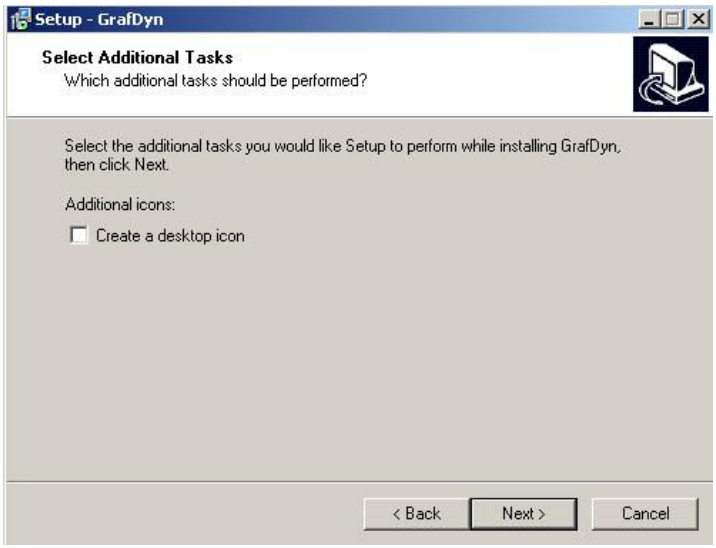
Rys. 5. Instalacja – wybór katalogu

W kolejnym okienku instalator stworzy nam nowy folder o nazwie GrafDyn w *Start Menu* (Rys. 6.).



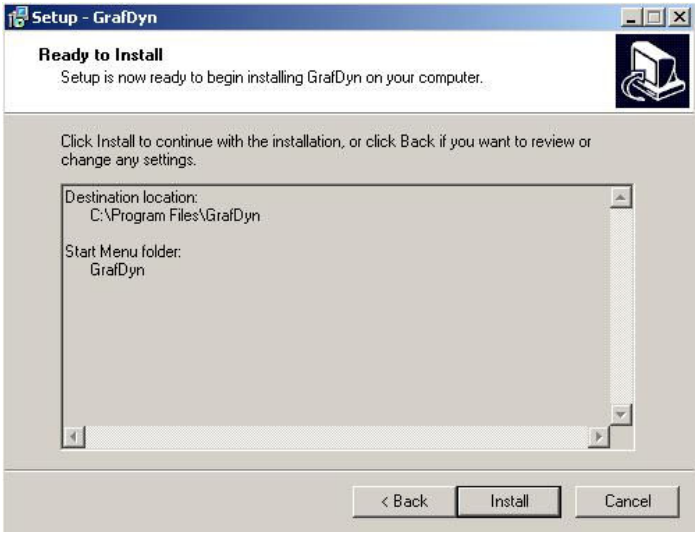
Rys. 6. Instalacja – skrót start menu

Po wyborze program instalacyjny zapyta nas o to, czy chcemy umieścić skrót do programu GrafDyn na pulpicie (Rys. 7.).



Rys. 7. Instalacja – skrót pulpit

Przed samą instalacją pojawi się jeszcze krótkie podsumowanie naszych ustawień i jeżeli wszystko się zgadza możemy przystąpić do instalacji programu GrafDyn (Rys. 8.) klikając przycisk *Install*.



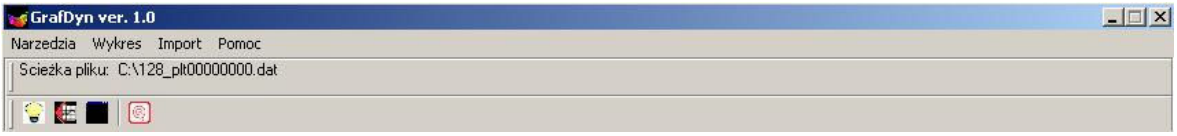
Rys. 8. Instalacja - podsumowanie

Jeżeli wszystko przebiegło bez żadnych problemów zobaczymy okienko z informacją o poprawnym zainstalowaniu programu (Rys.9.). Po zakończonej instalacji możemy przystąpić do pierwszego uruchomienia programu.

Rys. 9. Instalacja -koniec

5.2. Interfejs aplikacji GrafDyn

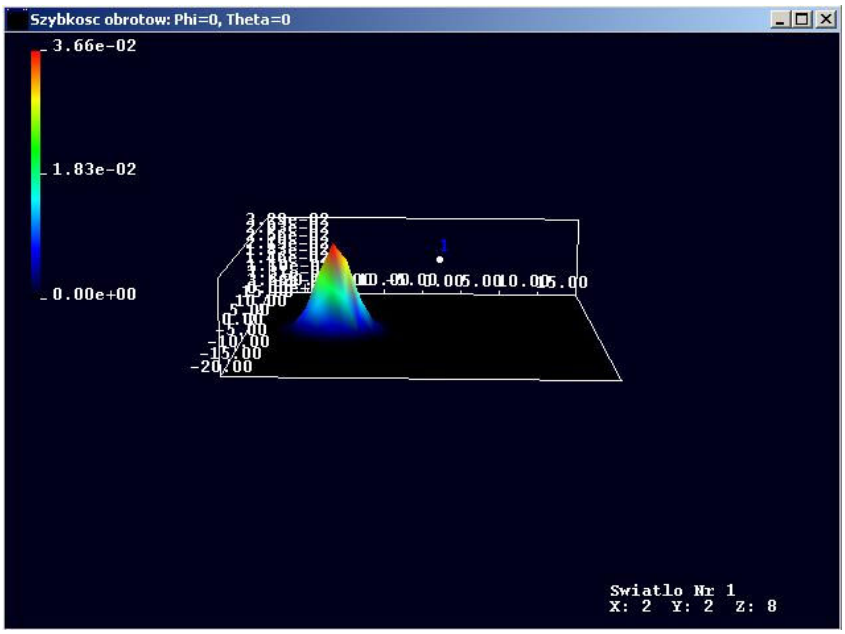
Po uruchomieniu programu GrafDyn, ukaże się nam główny interfejs aplikacji (Rys. 10.), na którym znajdują się wszystkie ważne funkcje potrzebne do zaimportowania, obracania oraz przekształcania naszego wykresu.



Rys. 10. Program GrafDyn

Interfejs składa się z głównego menu, informacji o pliku z danymi, który aktualnie jest wczytany oraz paska narzędzi. W tym pierwszym mamy pola takie jak:

- Narzędzia – służy do wybrania oświetlenia oraz własności wykresu,
- Wykres – włącza okno, w którym wyświetlony jest wykres (Rys.11.),
- Import - uruchamia moduł importu danych z plików,
- Pomoc – wyświetla informacje o wersji OpenGL.



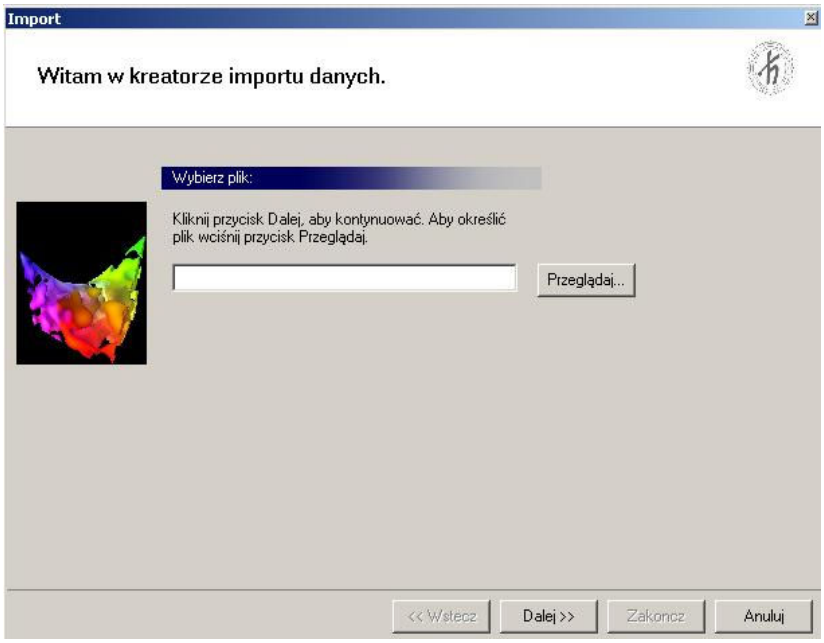
Rys. 11. Program GrafDyn

W pasku narzędzi znajdują się ikonki służące do szybkiego uruchamiania odpowiednich narzędzi (oświetlenie, własności wykresu, wykres i import). Po najechaniu kursorem myszki nad odpowiednią ikonę, system wyświetla pełną jego nazwę.

5.3. Moduł importu

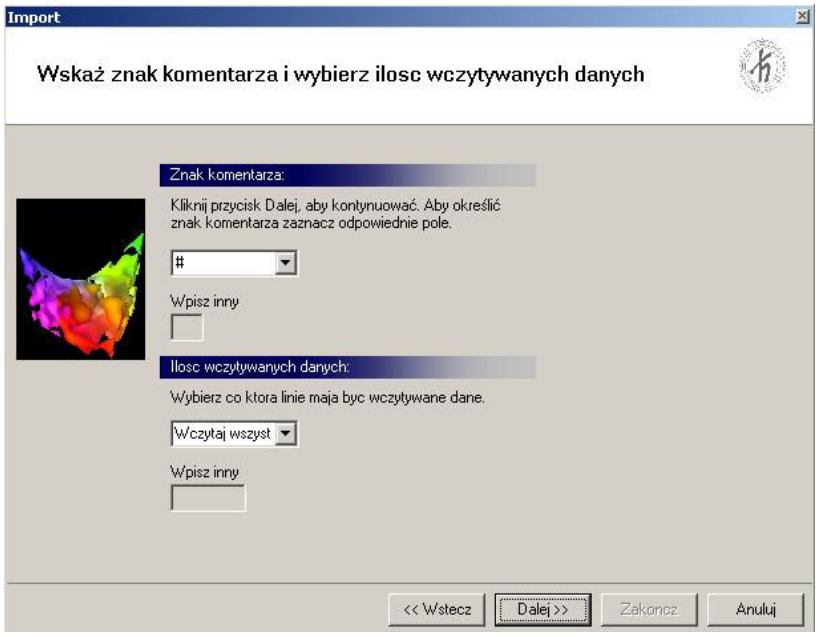
Moduł importu danych jest zaprojektowany na wzór analogicznego kreatora w Excelu, choć bardziej rozbudowany. Służy on do tego, aby nawet niezaawansowany użytkownik aplikacji mógł wybierając krok po kroku odpowiednie ustawienia wczytać dane z dowolnego pliku.

Po uruchomieniu funkcji Importu wyświetli się pierwsze okno kreatora (Rys 12.), w którym mamy możliwość wskazania pliku wpisując jego nazwę w polu tekstowym lub wybierając go za pomocą standardowego okna dialogowego Windows.



Rys. 12. Moduł importu – wybieranie pliku

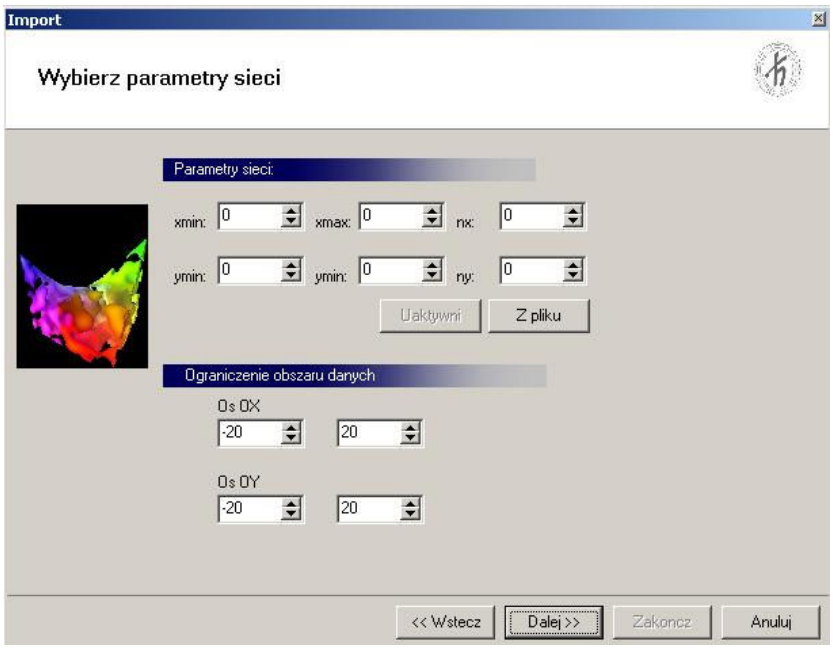
Kliknięcie przycisku *Dalej*, powoduje uruchomienie kolejnego kroku kreatora (Rys. 13.), w którym mamy możliwość wyboru znaku używanego do oznaczenia linii komentarza. Podczas importu program będzie omijał te linie w pliku, które rozpoczynają się od tego znaku. Mamy możliwość wyboru znaku z listy lub zdefiniowania własnego.



Rys. 13. Moduł importu – znak komentarza, ilość wczytywanych danych

Możemy też wybrać krok, z którym program będzie odczytywał linie z pliku. Dzięki temu można wizualizować nawet bardzo duże pliki narzucając uwzględnianie np. co drugiej, co trzeciej lub co dziesiątej linii.

W kolejnym oknie zostaniemy poproszeni o wybór parametrów sieci oraz zakres wyświetlanych danych (Rys. 14.). Jak wspomniałem wcześniej, parametry sieci możemy wczytać z pliku `.par`. Możemy też wpisywać je do odpowiednich pól edycyjnych. Oczywiście musimy uważać, aby były one poprawne. Są one bowiem konieczne do poprawnego odczytania danych z pliku. W tym samym oknie możliwe jest również ograniczenie obszaru, dla którego prezentujemy wykres (*clipping*).

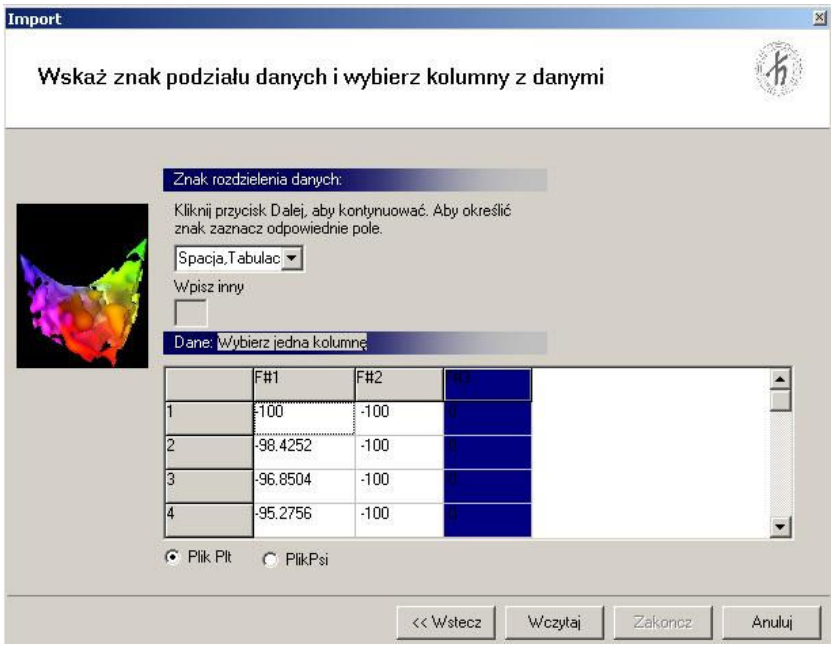


Rys. 14. Moduł importu – parametry sieci

Przed ostatecznym wczytaniem pliku musimy wybrać znak separacji pól z danymi w tej samej linii (rekordzie). Do wyboru mamy: spacja, tabulator, średnik, przecinek dwukropek. Możemy również zdefiniować własny znak podziału. Na wzór modułu importu danych Excela w oknie kreatora (Rys. 15.) wyświetlanych jest kilkanaście pierwszych linii pliku, które pomagają wybrać znak separacji. W tym samym oknie znajduje się też pole wyboru pozwalające wybrać rodzaj pliku, jaki wczytujemy W przypadku plików `*.plt` lub `plt*.dat` musimy wybrać jedną kolumnę zawierającą wartości funkcji (współrzędna z), zaś w przypadku plików `*.dat` wybieramy dwie kolumny odpowiadające części rzeczywistej i urojonej zespolonej wartości funkcji. W tym przypadku na wykresie pokazywany będzie kwadrat modułu funkcji. W obu przypadkach współrzędne x i y

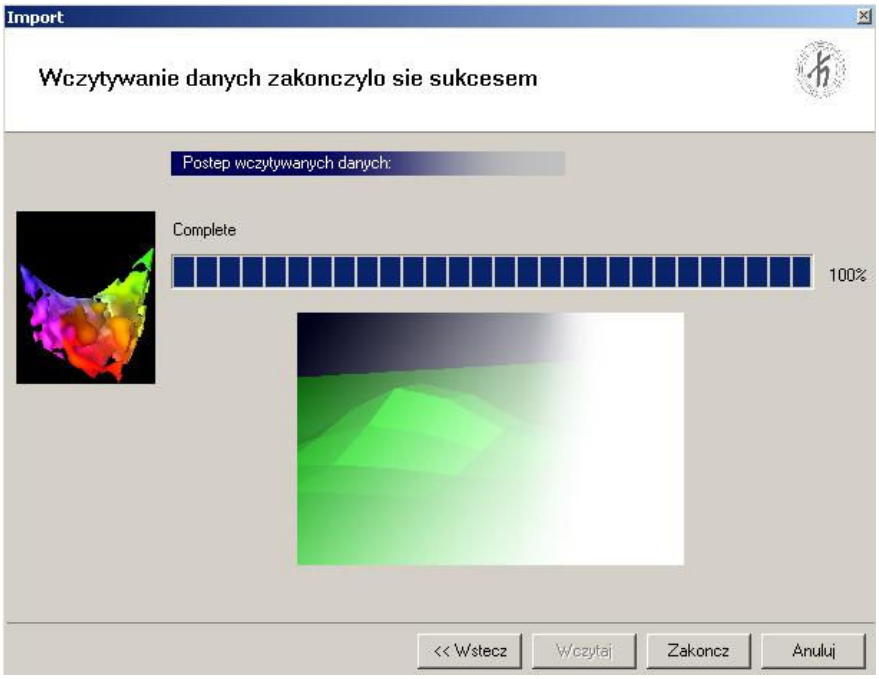
ustalane będą na podstawie parametrów sieci (zakresy i ilości węzłów) – zakładamy przy tym że sieć jest jednorodna.

Przed wczytaniem pliku, w każdym momencie możemy cofnąć się do dowolnego okna kreatora i zmienić każde z ustawień. Jeżeli jednak cofniemy się aż do pierwszego kroku i zmienimy plik danych, to we wszystkich polach zostaną przywrócone standardowe wartości.



Rys. 15. Moduł importu – znak podziału, kolumna z danymi

Po kliknięciu przycisku `Wczytaj` w ostatnim oknie kreatora nastąpi wczytanie danych z pliku do programu. Postęp wczytywania widoczny jest na pasku postępu (Rys. 16.). Jeżeli w trakcie wczytywania nie wystąpił błąd, zostaniemy poinformowani, że proces importu zakończył się sukcesem. W przeciwnym przypadku pojawi się komunikat o błędzie.



Rys. 16. Moduł importu – wczytywanie danych

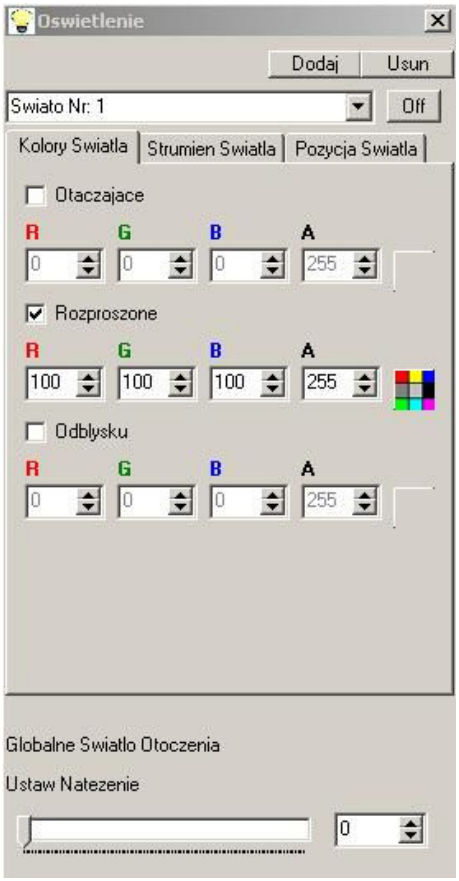
Wreszcie, po zakończeniu importu wyświetli się wykres funkcji. Możemy nim swobodnie obracać i modyfikować wygląd wykresu (zob. opis w kolejnych podrozdziałach).

5.4. Oświetlenie

Światło ma ogromny wpływ na postrzeganie obiektów na scenie, gdyż w rzeczywistym świecie nie liczy się tylko kolor obiektu. Każdy przedmiot może się wydawać lśniący lub matowy, a jego kolor może być inny w jasnym lub ciemnym oświetleniu. OpenGL pozwala symulować różne źródła światła i opisywać sposoby jego emisji. Biblioteka ta obsługuje do ośmiu niezależnych źródeł światła. W programie jednak mamy możliwość zdefiniowania tylko siedmiu, gdyż jedno użyte jest do określenia globalnego światła otaczającego. Pozwala ono oświetlić wszystkie obiekty równomiernie ze wszystkich stron.

Aby w programie GrafDyn dodać światło wystarczy wcisnąć przycisk `Dodaj` (Rys. 17.).Wówczas na ekranie pojawi się nowe źródło światła w postaci kuli z odpowiednim numerem. Zaznaczone światło można również usunąć ze sceny wciskając przycisk `Usuń`. Warto pamiętać, że po tej operacji bezpowrotnie stracimy ustawione właściwości usuniętego źródła światła. Jeśli z jakiś powodów chcemy, żeby część elementów nie była oświetlona, możemy w każdej chwili wyłączyć określone oświetlenie (przycisk `Wyłącz` (Rys. 17.)) zachowując jego właściwości. Nieaktywne źródło światła

przyjmuje wówczas kolor czerwony. Możemy oczywiście włączyć go ponownie wciskając przycisk `Włącz`. Aktywne światło ma kolor biały i bierze udział w oświetlaniu sceny.



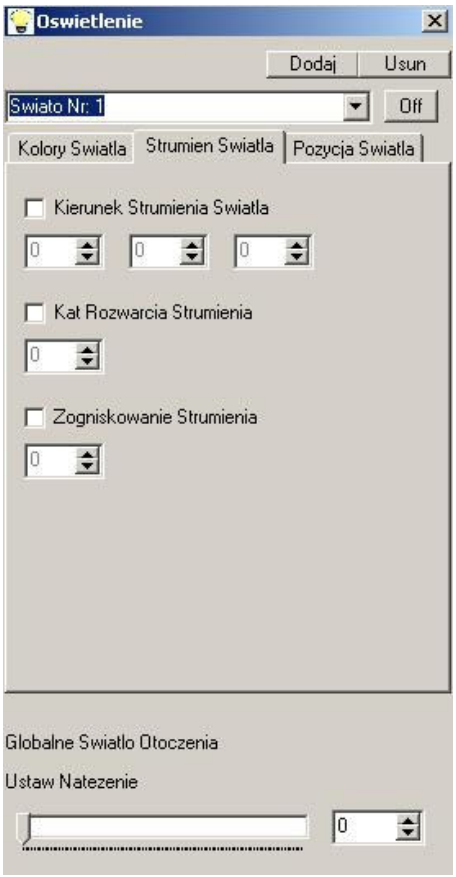
Rys. 17. Oświetlenie – kolory światła

OpenGL wykorzystuje lambertowski model światła, co oznacza, że źródła światła określane są za pomocą trzech składowych R (czerwony), G (zielony) i B (niebieski), definiujących barwę promieni [20]. Ponadto każde źródło światło możemy opisać za pomocą trzech różnych rodzajów oświetlenia (Rys. 17.):

- Światło otaczające (*Ambient Light*) – oświetla jednakowo każdą powierzchnię wykresu (źródło nie ma określonej pozycji),
- Światło rozproszone (*Diffuse Light*) - charakteryzuje się tym, że jego promienie biegną z konkretnego miejsca dzięki czemu jasność powierzchni zależy od ustawienia względem źródła,
- Światło odbłysku (*Specular Light*), podobnie jak rozproszone ma ustaloną pozycję, a ponadto także kierunek. Odbija się od oświetlanej powierzchni

tylko w jednym kierunku tworząc rozbłysk gdy na drodze promieni odbitych znajduje się kamera.

W programie możemy ustalić parametry oświetlenia w formularzu lub włączyć paletę kolorów i wybrać odpowiedni kolor. W podoknie `Oświetlenie` mamy także możliwość zmiany parametrów światła otaczającego dla całej sceny, które jest niezależne od pozostałych źródeł. Dzięki temu nie musimy włączać całego modułu oświetleniowego, aby obiekty na scenie były widoczne. Jego natężenie ustawiamy w zakresie od 1 do 10 za pomocą suwaka.

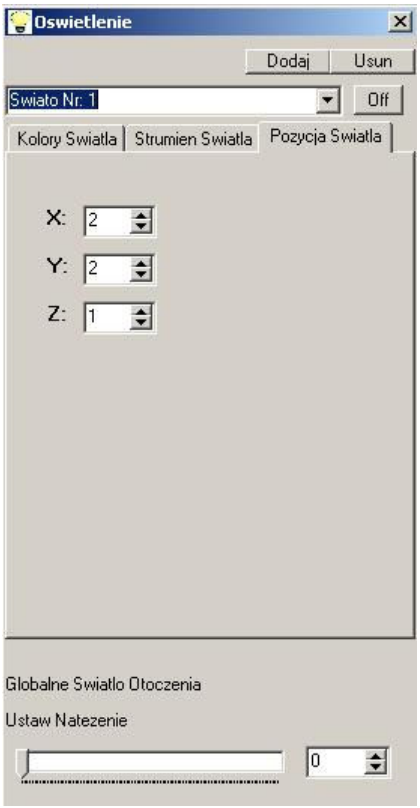


Rys. 18. Oświetlenie – strumień światła

W zakładce `Strumień Światła` (Rys. 18.) mamy możliwość zdefiniowania innych własności źródła światła takich jak:

- `GL_SPOT_DIRECTION` - określa kierunek strumienia światła, czyli kierunek, w którym zostanie skierowany "reflektor",
- `GL_SPOT_EXPONENT` - określa intensywność światła. Przyjmuje wartości z przedziału [0-128],

- `GL_SPOT_CUTOFF` - określa kąt stożka światła kierunkowego.



Rys. 19. Oświetlenie – pozycja światła

W ostatniej zakładce (Rys. 19.) mamy możliwość zmiany pozycji źródła światła, podając jego współrzędne *x*, *y* i *z*.

W górnym okienku narzędzia `Oświetlenie` mamy wyświetloną listę wszystkich zdefiniowanych źródeł światła. Przechodząc po kolei po każdym źródle możemy obejrzeć jego własności. Wybór źródła światła możemy również dokonać poprzez jego kliknięcie w oknie wykresu (postać kuli z numerem światła). Implementację tej możliwości opisuje rozdział 2 dotyczący trybu selekcji w OpenGL.

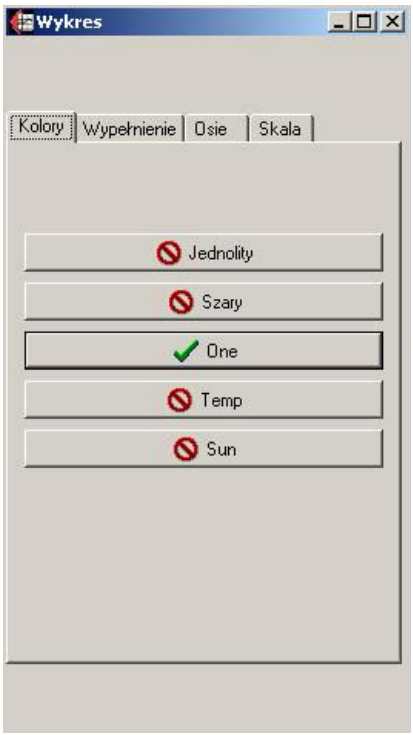
Należy pamiętać, że każdy zdefiniowany w OpenGL obiekt ma własne kolory rozumiane jako własność materiału. Kolor widziany na ekranie powstaje przez uwzględnienie tego koloru oraz koloru źródła światła. Dobranie światła tak, aby uzyskać pożądany efekt wymaga od użytkownika odrobiny praktyki.

5.5. Własności wykresu

Program GrafDyn daje użytkownikowi spore możliwości manipulowania wykresem. Do tego celu służy podokno `Wykres`, które jest dostępne z paska narzędzi lub menu głównego aplikacji. Narzędzie to pozwala ustawić określone własności wyświetlanej funkcji, jak również ustawić pewne własności dotyczące okna wykresu.

W pierwszej zakładce o nazwie `Kolory` (Rys. 20.) mamy możliwość wyboru palety barw, która zmienia kolory użyte do rysowania funkcji. Algorytm zaimplementowany w programie wyznacza składowe koloru odpowiadające wartościom z wyznaczonego zakresu danych.

Pierwszy określa jednolity kolor – jednakowy kolor dla wszystkich naszych danych (oczywiście przy włączonych źródłach światła jasność wykresu nadal będzie różna w różnych punktach ze względu na cieniowanie).



Rys. 20. Własności wykresu – kolory

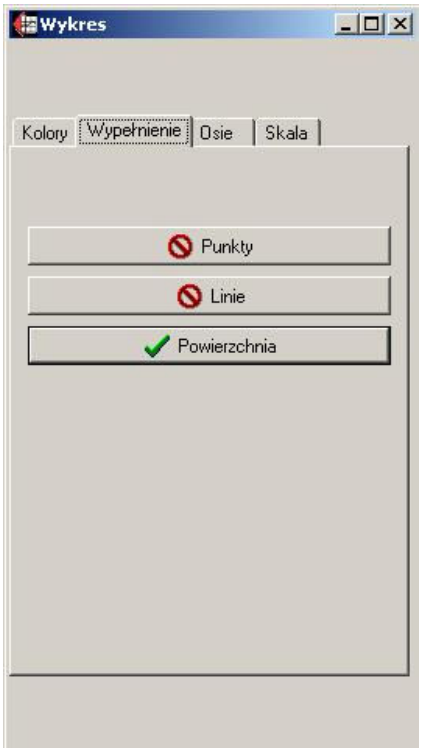
Barwa `Szara` to paleta w której mamy do czynienia z jednym przedziałem. Wartości wszystkich składowych są sobie równe i wyznaczone z prostej proporcji (Listing 14.).

Listing 14. Barwa Szara

```
rgb_triplet Color;
    Color.r = 0;
    Color.g = 0;
    Color.b = 0;
    double delta = maxVal - minVal,
    dp  =data_Val - minVal;
    if (!delta) return Color;
      Color.r = (255*dp/delta);
      Color.g = (255*dp/delta);
      Color.b = (255*dp/delta);
    return Color;
```

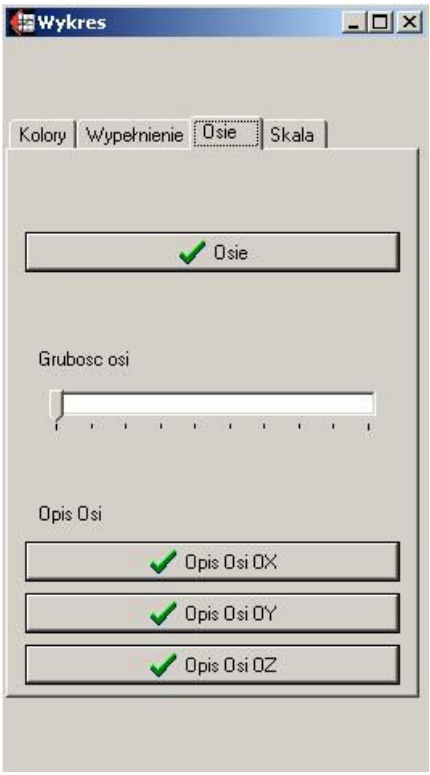
Kolejna paleta `One` jest oparta o algorytm, w którym wartości dzielimy na przedziały. Każdy z nich odpowiada zmianie jednej ze składowych koloru. Następnie po wczytaniu danych przypisujemy każdej wartości odpowiedni przedział i wyliczamy odpowiednie wartości składowych koloru. Ostatnie dwie palety `Temp` i *`Sun`* wykorzystują bibliotekę funkcji graficznych z pliku *colormaps.cpp*, której autorem jest dr Jacek Matalewski. Każda zmiana palety barw wymaga ponownego wyznaczenia kolorów dla każdej wartości wyświetlanych danych.

W kolejnej zakładce o nazwie `Wypełnienie` (Rys. 21.) mamy możliwość ustawienia kolejnego stylu wyświetlania naszych danych. Po wybraniu przycisku `Punkty`, dane na wykresie będą przedstawione w formie punktów. Podobnie po wybraniu własności `Linie`, dane wyświetla się nam w oknie w postaci linii. Najciekawszą i najbardziej naturalną a zarazem najtrudniejszą do zaimplementowania jest własność `Powierzchnia`. Aby wykres ładnie się prezentował wymagało to wyznaczenia normalnych do powierzchni. Wykres złożony jest z połączonych pasków czworokątów za pomocą parametru `GL_QUAD_STRIP`. Paski będą tworzone ze wszystkich werteksów wywoływanych w trakcie wykonywania pętli. Powierzchnia taka ma ładny i jednolity kształt.



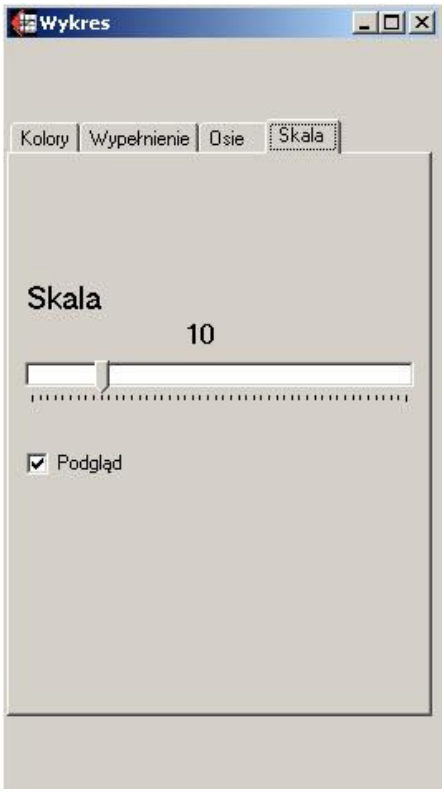
Rys. 21. Własności wykresu – wypełnienie

W zakładce `Osie` (Rys. 22.) mamy możliwość manipulowania osiami. Możemy wyłączyć wszystkie osie jak również opisy poszczególnych z nich. Dostępny jest też parametr pozwalający na ustawienie grubości osi. Osie *X* i *Y* przyjmują wartości z zakresu wyświetlania, który podaliśmy podczas importu danych. Oś *Z* posiada zakres wyznaczony przez minimalną i maksymalną wartość wczytanej z pliku funkcji.



Rys. 22. Własności wykresu – osie

W ostatniej zakładce (Rys. 23.) możemy przeskalować nasz wykres. Gdybyśmy tego nie zrobili, to po wczytaniu danych z niektórych plików, wyglądałoby one jakby miały tą samą wartość w każdym punkcie. Dzieje się tak, gdyż dane mają zbyt małą wartość w porównaniu z rozmiarami sieci i nie widzimy żadnej interesującej nas struktury. Aby móc zobaczyć wykres, musimy przeskalować dane z pliku do większej wartości. Dlatego też w programie została zaimplementowana własność `skala`, która pozwala użytkownikowi samemu wybrać do jakiej liczby chce skalować wykres.



Rys. 23. Własności wykresu – skala

6. Podsumowanie

Program GrafDyn został napisany z myślą o wyświetlaniu funkcji dwóch zmiennych używając do tego biblioteki OpenGL. Dzięki tej bibliotece działa znacznie szybciej niż analogiczny moduł w popularnym programie gnuplot. Dane są wczytywane za pomocą specjalnego modułu importu, który pozwala użytkownikowi na ustawienie własności wczytywanych plików bez ingerencji w kod programu. Dodatkowym atutem GrafDyn jest możliwość personalizacji sposobu, w jaki funkcja jest wyświetlania (zmiana barwy, opis osi, legenda). Mamy możliwość własnego definiowania źródeł światła i ustawienia dowolnego oświetlenia na scenie. Możliwość interakcji użytkownika ze sceną, powoduje, że obsługa programu jest intuicyjna.. Program został napisany tak, aby można go było rozszerzać o kolejne funkcjonalności bez większej ingerencji w dotychczasowy kod programu.

Uważam, że napisany przeze mnie program GrafDyn nie tylko stanie się użytecznym narzędziem do wyświetlana funkcji dwóch zmiennych ale również będzie niezawodnie służył swoim nowym użytkownikom w przejrzystym obrazowaniu wykresów.

7. Literatura

[1] R.S. Wright, M. Sweet, *Księga eksperta*, Helion, Gliwice (1999).

[2] K. Hawkins, D. Stale, *OpenGL Programowanie Gier*, Helion, Gliwice(2003).

[3] B. Eckel, C. Allison, *Thinking In C++ Tom 2*, Helion, Gliwice(2004).

[4] J. Gręboszl, *Symfonia C++*, Edition 2000, Kraków (2000).

[5] R.Eisberg, R.Resnick, *Fizyka kwantowa*, PWN, Warszawa (1983).

[6] R.L.Liboff, *Wstęp do mechaniki kwantowej*, PWN, Warszawa (1987).

[7] http://nehe.gamedev.net - Strona poświecona grafice 3D wykorzystująca bibliotekę OpenGL.

[8] http://www.dimmension3.spine.pl/index.php - strona poświęcona programowaniu grafiki 3D.

[9] http://www.3d.pl – strona poświęcona grafice 3D.

[10] http://www.talisman.org/opengl-1.1/OpenGL.html - strona poświecona OpenGL.

[11] http://ddt.pl/kursy/?TutorialId=10 – strona zawierająca kurs OpenGL.

[12] http://www.borland.pl/tech/opengl.shtml - strona opisująca funkcje OpenGL.

[13] M. DeLoura, *Perełki programowania gier*, Helion, Gliwice(2002).

[14] http://www.potu.com/man/pyopengl.sourceforge.net - strona poświęcona OpenGL.

[15] http://januszg.hg.pl/opengl/index.html - strona zawiera informacje na temat programowania z zastosowaniem biblioteki OpenGL i GLUT.

[16] J.Matulewski, *C++ Builder 2006. 222 gotowe rozwiązania*, Helion, Gliwice(2006).

[17] http://www.gamedev.pl/articles.php?x=view&id=269 – strona poświęcona kwaternionom w grafice 3D.

[18] http://rainwarrior.thenoos.net/dragon/arcball.html - strona poświęcona klasie ArcBall i użytych w niej kwaternionu.

[19] http://www.gamedev.pl/files/articles/kwaterniony/Streszczenie.doc - strona poświęcona kwaternionom oraz ich zastosowaniu w grafice komputerowej.

[20] http://www.pcworld.pl/artykuly/42219/Swiatla.na.scene.html - strona poświęcona definiowaniem oświetlenia w OpenGL.

Dodatek A – Prymitywy

Biblioteka OpenGL oferuje kilka rodzajów prymitywów graficznych, które służą do modelowania dowolnych obiektów. Definiowanie współrzędnych wierzchołków musi zawierać się pomiędzy wywołaniami funkcji:

```
void glBegin (GLenum mode)
void glEnd (void),
```

gdzie parametr *mode* określa rodzaj prymitywu, którego wierzchołki będą definiowane. Parametr ten przyjmuje określone wartości i najczęściej używane to:

- GL POINTS - punkty
- GL LINES - odcinki
- GL LINE STRIP - łamana
- GL LINE LOOP - łamana zamknięta
- GL TRIANGLES - trójkąty
- GL TRIANGLE STRIP - wstęga trójkątów
- GL QUADS - czworokąty,
- GL QUAD STRIP - wstęga czworokątów,
- GL POLYGON - wielokąt.