## WOJSKOWA AKADEMIA TECHNICZNA

# *LABORATORIUM*

# *GRAFIKA KOMPUTEROWA*

Stopień, imię i nazwisko słuchacza

Grupa szkoleniowa

Stopień, imię i nazwisko prowadzącego

***Grzegorz Pol***

***I7X3S1***

***dr inż. Marek Salamon***

Data wykonania ćwiczenia

***26.11.2008 r.***

***SPRAWOZDANIE***

***Z***

***PRACY LABORATORYJNEJ***

***NR 2***

|  |  |
| --- | --- |
| **Temat:** | Modelowanie prostych obiektów przy pomocy prymitywów. |

* 1. **Zadania**

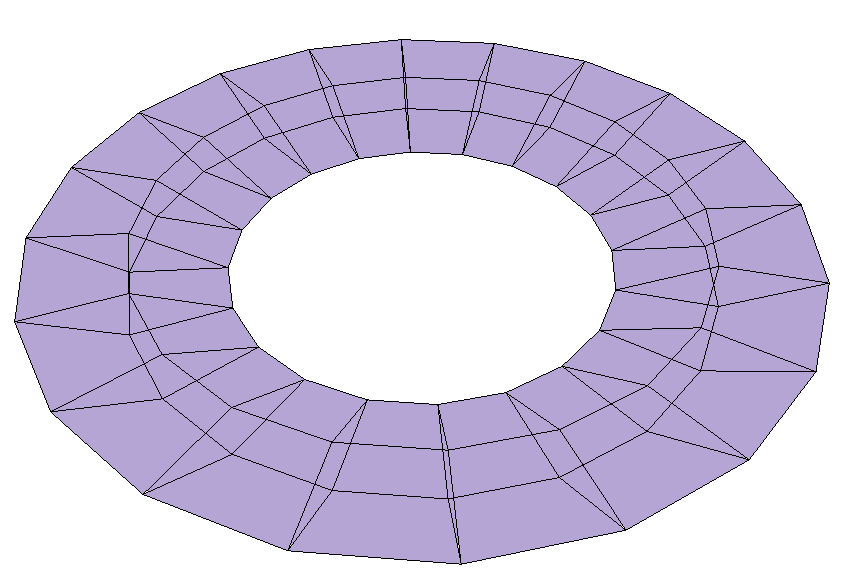
Podczas zajęć laboratoryjnych należało wykonać zadanie o poniższej treści:

1. Stworzyć pierścień o przekroju w kształcie rombu o boku 5, gdzie kąt =30, a promień wewnętrzny wynosi 10 za pomocą trybu modelowania GL\_QUADS
2. Program ponadto powinien mieć możliwość zmiany liczby podziałów bryły, odległości obserwatora oraz jego orientacji.
   1. **Tryb modelowania GL\_QUADS**

Tryb modelowania GL\_QUADS oczekuje od nas podania 4 wierzchołków, które następnie połączy w czworokątny prymityw.

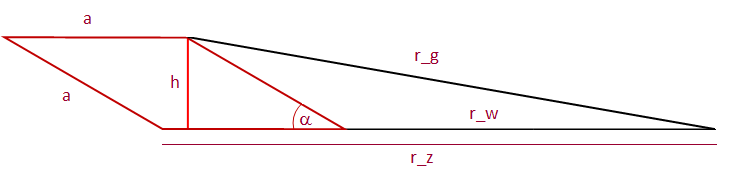
* 1. **Cel zadania**

Celem zajęć laboratoryjnych było zbudowanie obiektu widocznego na poniższym rysunku



* 1. **Obliczenia**

Kluczem do zrealizowania zadania było rozwiązanie zadania geometrycznego gdzie a=5, =30 a r\_w=10.



Po wykonanych przeze mnie obliczeń geometrycznych otrzymałem następujące wyniki:

r\_g = 14.546, r\_z = 15.000, h = 2.500

Powyższe dane zapisałem jako stałe na początku programu. W ten sposób nie występują one bezpośrednio w funkcjach, co ułatwia nam w przyszłości bardzo szybką możliwość dostosowania rombu do naszych potrzeb.

* 1. **Etapy realizacji *pierścienia***

Modelowanie obiektu w moim programie oparte jest na czterech pętlach *for*, których iteracja zależy od liczby podziałów jaką sobie życzy obserwator. Każda z tych pętli odpowiedzialna jest za stworzenie pojedynczej ściany a więc pętla:

for(i=0;i\*dAlpha<=360;i++){

glBegin(GL\_QUADS);

glVertex3f(r\_w\*cos(DEG2RAD(i\*dAlpha)), 0, r\_w\*sin(DEG2RAD(i\*dAlpha)));

glVertex3f(r\_z\*cos(DEG2RAD(i\*dAlpha)), 0, r\_z\*sin(DEG2RAD(i\*dAlpha)));

i++;

glVertex3f(r\_z\*cos(DEG2RAD(i\*dAlpha)), 0, r\_z\*sin(DEG2RAD(i\*dAlpha)));

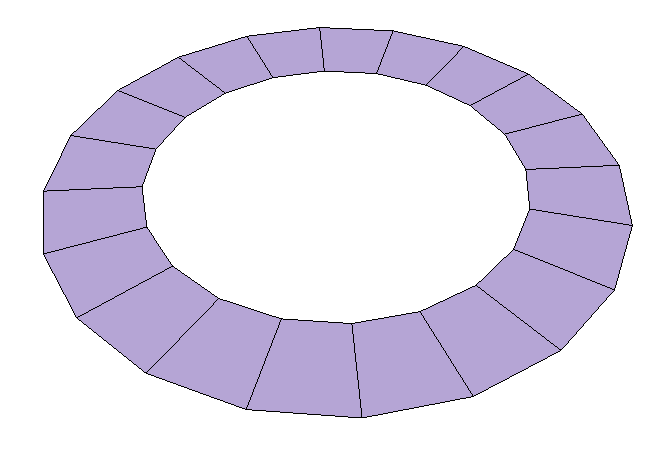
glVertex3f(r\_w\*cos(DEG2RAD(i\*dAlpha)), 0, r\_w\*sin(DEG2RAD(i\*dAlpha)));

i--;

glEnd();

}

odpowiedzialna jest za utworzenie podstawy danego rombu. Warto w tym miejscu nadmienić zasadę oraz cel działania funkcji *DEG2RAD*. Wspomniana funkcja jak sama nazwa wskazuje odpowiedzialna jest za zamianę stopni na radiany. W ten sposób bezproblemowe staje się, użycie funkcji trygonometrycznych w pętlach. Wizualizacje modelu po wykonaniu pierwszej pętli przedstawiam poniżej:



Drugi etap tworzenia naszego modelu polega na stworzeniu wewnętrznej ściany bocznej *pierścienia*. Warto wspomnieć, że rysujemy *pierścień* za pomocą czworokątów, a jego ściany będą tylko imitowały kształty okręgu. Im więcej operator zażyczy sobie boków, tym uzyskiwany efekt będzie coraz lepszy. Druga pętla *for* wygląda następująco:

for(i=0;i\*dAlpha<=360;i++){

glBegin(GL\_QUADS);

glVertex3f(r\_w\*cos(DEG2RAD(i\*dAlpha)), 0, r\_w\*sin(DEG2RAD(i\*dAlpha)));

glVertex3f(r\_g\*cos(DEG2RAD(i\*dAlpha)), h, r\_g\*sin(DEG2RAD(i\*dAlpha)));

i++;

glVertex3f(r\_g\*cos(DEG2RAD(i\*dAlpha)), h, r\_g\*sin(DEG2RAD(i\*dAlpha)));

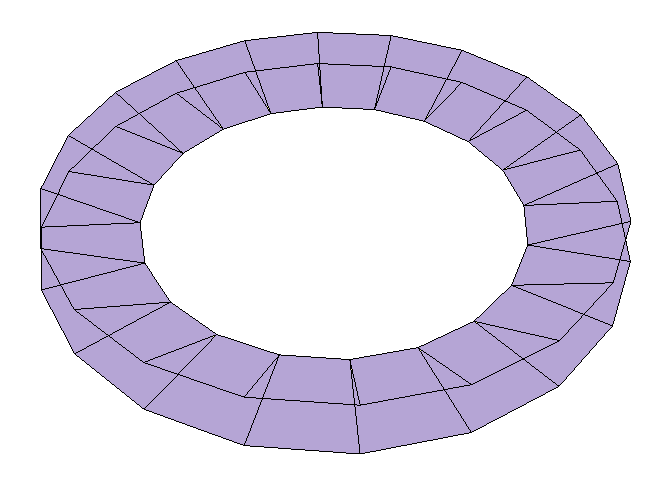
glVertex3f(r\_w\*cos(DEG2RAD(i\*dAlpha)), 0, r\_w\*sin(DEG2RAD(i\*dAlpha)));

i--;

glEnd();

}

Jak widać w tej funkcji użyliśmy obliczoną przez nas wcześniej wartość *r\_g.* Warto zauważyć, że dwa punkty w drugiej pętli *for* już wystąpiły przy tworzeniu podstawy. Wizualizacje już dwóch połączonych ze sobą ścian zamieszczam poniżej:



Trzecia pętla *for* będzie odpowiadała za drugą ścianę boczną, tym razem zewnętrzną. Jest ona bliźniaczo podobna do poprzedniej, gdyż jest przesunięta o długość ściany rombu *a*. Kod wygląda następująco:

for(i=0;i\*dAlpha<=360;i++){

glBegin(GL\_QUADS);

glVertex3f((r\_w+a)\*cos(DEG2RAD(i\*dAlpha)), 0,(r\_w+a)\*sin(DEG2RAD(i\*dAlpha)));

glVertex3f((r\_g+a)\*cos(DEG2RAD(i\*dAlpha)), h, (r\_g+a)\*sin(DEG2RAD(i\*dAlpha)));

i++;

glVertex3f((r\_g+a)\*cos(DEG2RAD(i\*dAlpha)), h, (r\_g+a)\*sin(DEG2RAD(i\*dAlpha)));

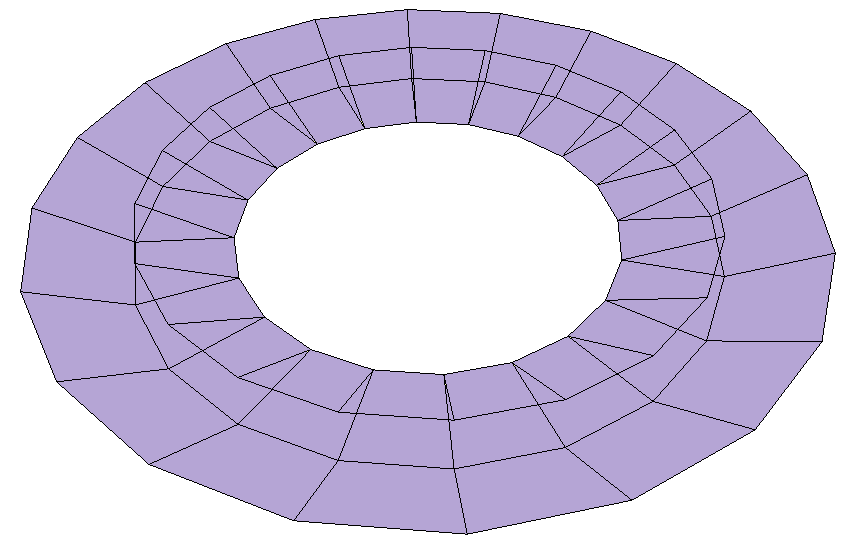
glVertex3f((r\_w+a)\*cos(DEG2RAD(i\*dAlpha)), 0, (r\_w+a)\*sin(DEG2RAD(i\*dAlpha)));

i--;

glEnd();

}

W tym wypadku także widzimy, że dwa z czterech wierzchołków zostały już użyte w podstawie. Efekt połączenia trzech ścian zamieszczam poniżej:



Ostatni etap polega na połączeniu wszystkich czterech wierzchołków, które nie wystąpiły dwukrotnie czyli są połączone tylko raz. W ten sposób za pomocą ostatniej pętli *for* tworzymy górną podstawę naszego rombu i całego pierścienia:

for(i=0;i\*dAlpha<=360;i++){

glBegin(GL\_QUADS);

glVertex3f(r\_g\*cos(DEG2RAD(i\*dAlpha)), h, r\_g\*sin(DEG2RAD(i\*dAlpha)));

glVertex3f((r\_g+a)\*cos(DEG2RAD(i\*dAlpha)), h, (r\_g+a)\*sin(DEG2RAD(i\*dAlpha)));

i++;

glVertex3f((r\_g+a)\*cos(DEG2RAD(i\*dAlpha)), h, (r\_g+a)\*sin(DEG2RAD(i\*dAlpha)));

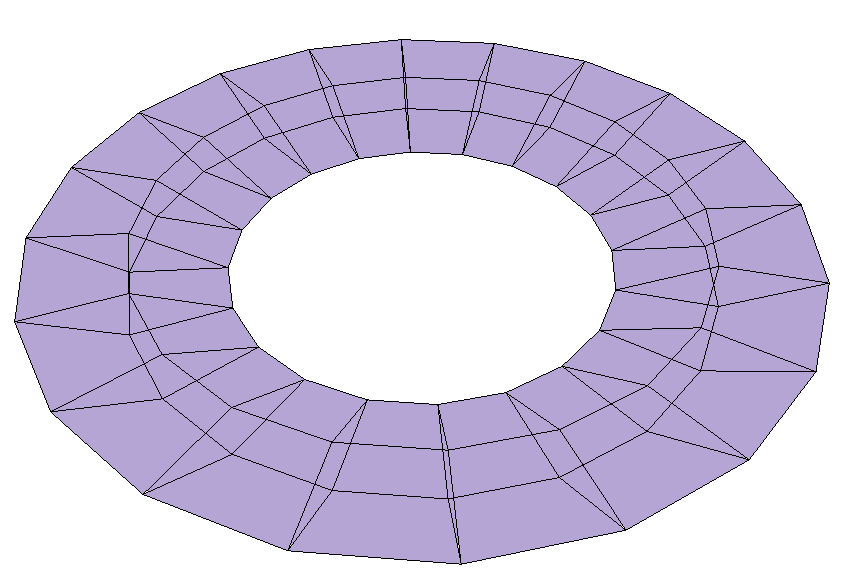
glVertex3f(r\_g\*cos(DEG2RAD(i\*dAlpha)), h, r\_g\*sin(DEG2RAD(i\*dAlpha)));

i--;

glEnd();

}

Efekt wizualny jak widać poniżej jest zgodny z tym co zamierzaliśmy osiągnąć. Cel został zrealizowany:



* 1. **Obsługa modelu przez użytkownika**

Program zgodnie z zadaniem powinien mieć możliwość zmiany liczby podziałów, obrotu bryły w trzech kierunkach oraz oddalania i przybliżania. Zrealizowałem to przy pomocy funkcji *ObsługaKlawiatury*, która to pobiera wciśnięty klawisz, a następnie sprawdza czy została mu przypisana konkretna akcja.

void ObslugaKlawiatury(unsigned char klawisz, int x, int y){

if (klawisz == '+')

if(n<=60)

n++;

if (klawisz == '-')

if(n>3)

n--;

if (klawisz == 'r')

odleglosc++;

if (klawisz == 'f')

odleglosc--;

if (klawisz == 'e')

os\_Y--;

if (klawisz == 'd')

os\_Y++;

if (klawisz == 'w')

os\_X++;

if (klawisz == 's')

os\_X--;

if (klawisz == 'q')

os\_Z++;

if (klawisz == 'a')

os\_Z--;

if (klawisz == 27)

exit(0);

}

Każda z tych pętli wygląda podobnie. Jedynym wyjątkiem są dwa *if*-y, odpowiedzialne za ilość podziałów. Użyłem tam zagnieżdżonych pętli w celu zabezpieczanie się przed tym aby użytkownik przypadkiem nie chciał zmodyfikować ilość ścian bocznych poniżej trzech (byłby to bezsens – gdyż dopiero trzy ściany dają nam pierwszą figurę – trójkąt), a także aby nie obciążał zbytnio komputera zmuszając go do rysowania zbyt wielu prymitywów.

* 1. **Wnioski**

Postawione zadania z pkt. 1 zostały tutaj przeze mnie poprawnie zrealizowane. Uniknąłem tutaj błędu, który popełniłem podczas zajęć laboratoryjnych, a mianowicie górną podstawę zrealizowałem jako połączenie punktów wcześniej narysowanych, nie sugerując się już wyliczeniami matematycznymi. W ten sposób wszystkie ściany są ze sobą idealnie złączone.

Warto zauważyć, że w OpenGL-u każdy model składa się z prostych prymitywów, które jednak przy większej ilości potrafią w bardzo dobry sposób imitować nawet bardzo skomplikowane figury. Ponadto ciekawym pomysłem na ożywienie modelu jest możliwość ingerencji w niego przez użytkownika.