

**STATE UNIVERSITY OF NEW YORK AT BUFFALO**  
*Mechanical and Aerospace Engineering Department.*

**MAE 573 GRAPHICS IN CAD**

Project 02: Simple 3D CAD Software

NAME: LENG-FENG LEE

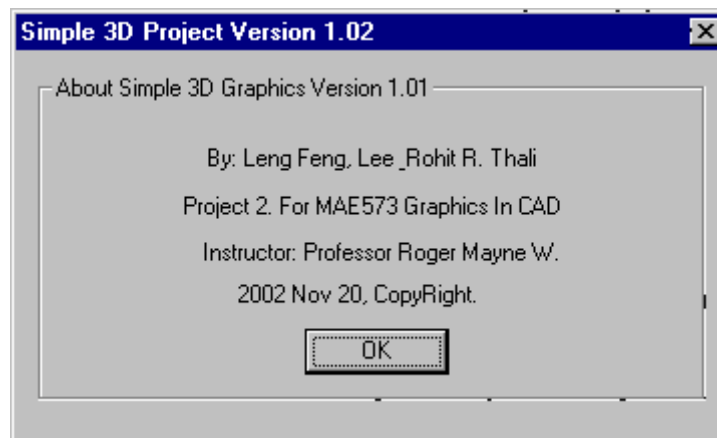
DATE: 22<sup>rd</sup> Nov 2002.

## INTRODUCTION:

This simple Three D Graphics project combines several features that we have learned in the class. It demonstrated the idea of hidden line removal for convex objects. In this project “dot product test” approach is used to eliminate those polygons that are not visible to the viewer. This program also allows the viewer to view the objects from different viewing axis. Beside this, the program also included a zooming feature such that allowed the viewer to specify the distance to view the object. Beyond this, a viewer is allowed to view the objects by orbiting the objects. Finally, a simple animation that allowed the objects to travel along a straight line or a Bezier curve is also presented.

## SOFTWARE CAPABILITIES AND FEATURES.

This is a simple 3 Dimension Display software that used to demonstrate some important ideas in 3 Dimensional Computer graphics.



We included some features in this software and each of them will be explained in the following paragraphs.

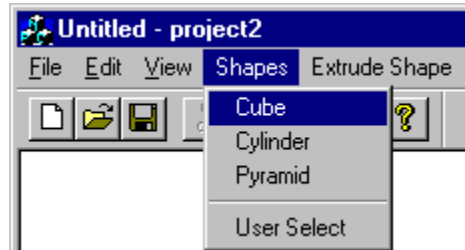
### 1. *Menu bar.*

The menu bar is shown below and this menu bar give us a brief ideas on what is the capabilities that contains in this software.

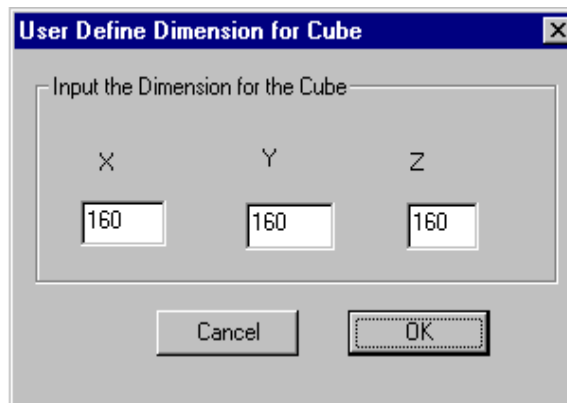


## 2. 3D Extrude Objects Display.

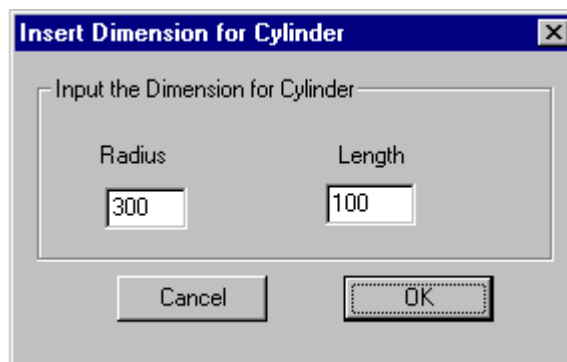
We have three basic shapes in this software package, they are Cube, Cylinder, and Pyramid. Each of the shape has its own default dimension. These shape can be select from a dialog box and the user can change the dimension of each shape.



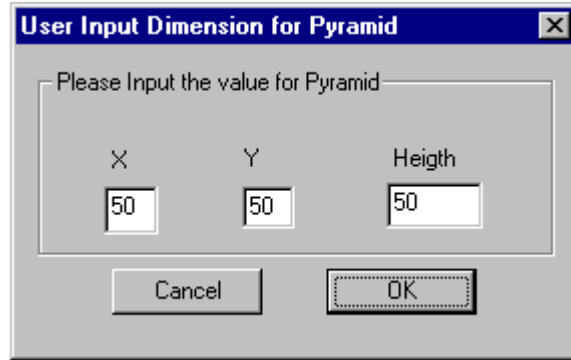
After selection, there will be dialog box for each object. These dialog boxes allowed the user to input a desired dimension for each shapes. Each of the Dialog Box is shown bellow:



*Dialog Box For Cube.*



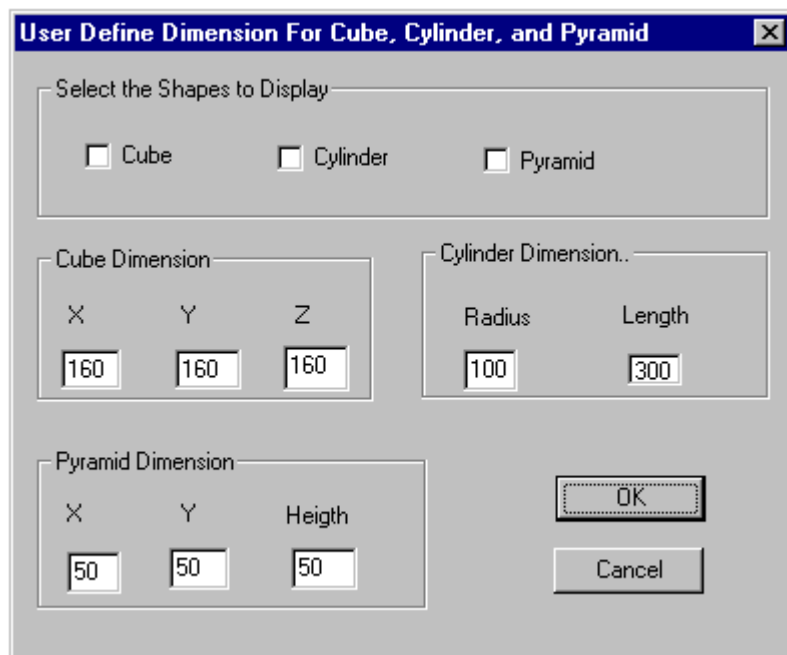
*Dialog Box for Cylinder*



*Dialog Box for Pyramid.*

## **2. Change of objects Dimensions.**

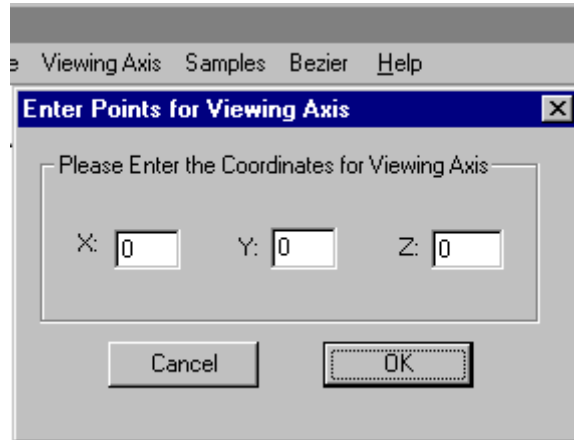
Change of the Dimension for the shapes is available to user by using dialog box. The user can choose to change the dimension of the objects individually (Cube, Cylinder, and Pyramid) using the dialog box of each shapes or change all of them together using a dialog box that available. The two dialog box are connected to each other, change of one individual dialog box will also change the value in the dialog box that can change the dimension of the shapes simultaneously.



*Dialog Box for Three Shapes to Display.*

### 3. Change of Viewing point through Dialog Box.

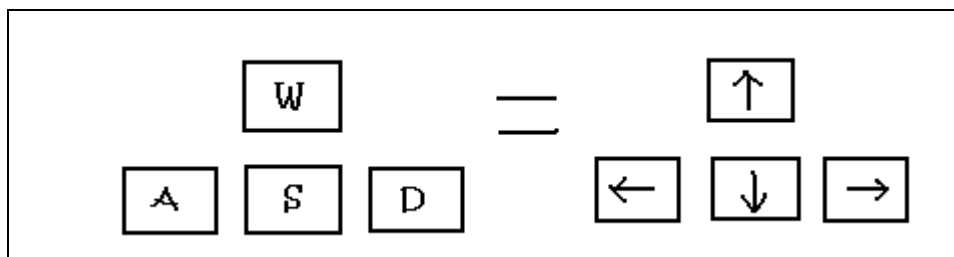
The user can change the viewing point of each objects so that can user can see the each face that of the objects that he desired. This is done by clicking on the “Viewing Point” on the menu bar.



*Dialog Box to input Value for Viewing Axis.*

### 3. Rotation and Zooming of Objects using keyboard buttons.

We implement the following key board button for rotation of each individual objects on its one axis. This is not a changing of viewing point nor the changing of the global viewing position. the corresponding key is adopted to represent the arrow key and each of them will cause rotation of the objects.



“W” key =Zoom out from the object.

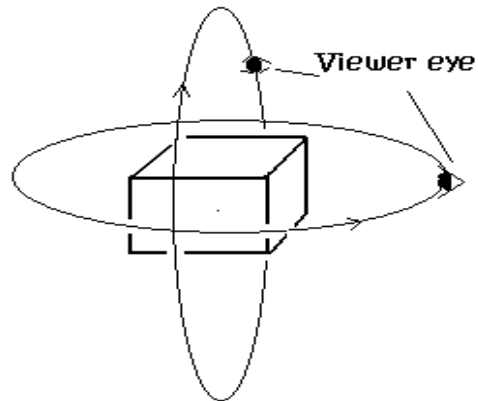
“A” key =Rotate about Z axis.

“S” key =Zoom in the object.

“D” key =Rotate about X axis.

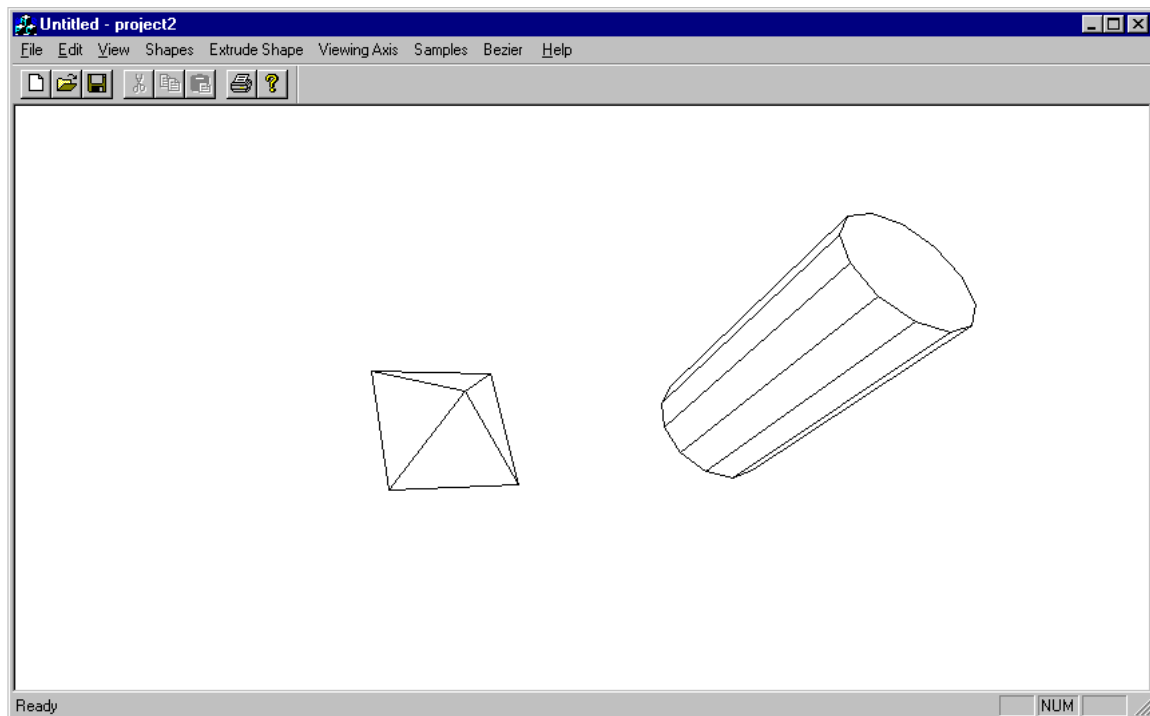
## 5. “Orbiting Viewing” using Arrow Key.

The arrow keys, on the other hand, were implement to view the objects in from a global perspective.



Using the left, right, up, and down arrow key on the keyboard, the user can view the object by rotating the viewing point around the object. Moving the “Left” key will move the viewing point to the left and moving the right required to use the “Right” arrow key. Here, however you will see that the viewing position is not moving in a straight line but instead, moving by orbiting the objects.

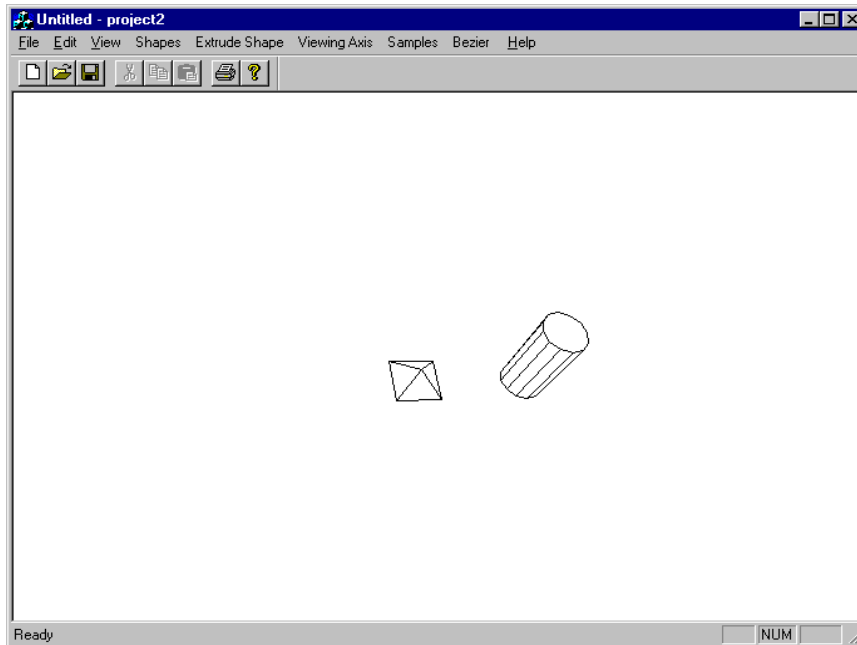
The “Up” and “Down” buttons use the same concepts to orbit the viewing point up and down of the objects.



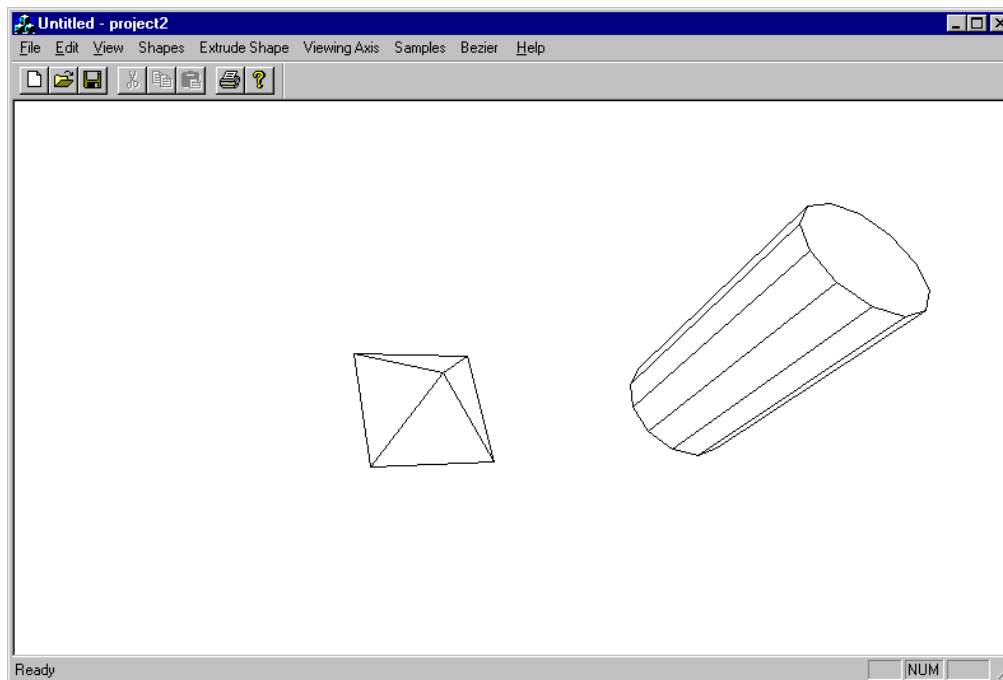
*Example of a “Global viewing” orbiting angel of two objects.*

## 6. Zooming of objects using key “i” and “o”.

We implement the “i” key and the “o” key as a zooming button such that when pressing “i” we are zooming into the object and by pressing “o” we are away from the objects. This function is similar to the camera zooming function so that you can have a better view of the objects.



*Example after using “o” key to zoom out the scene of two objects.*

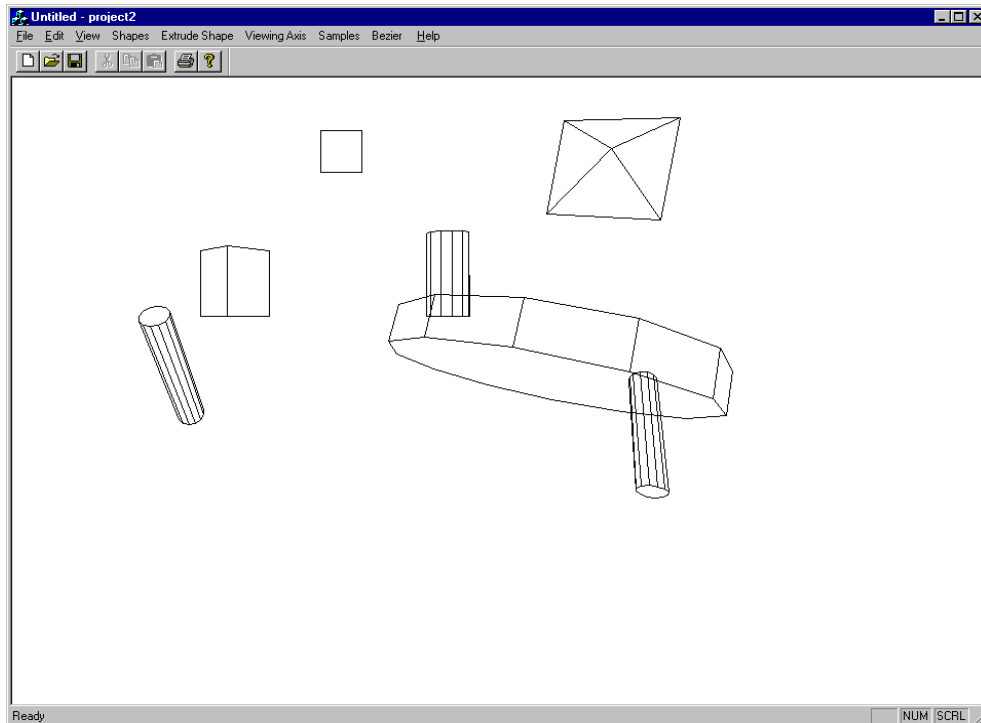
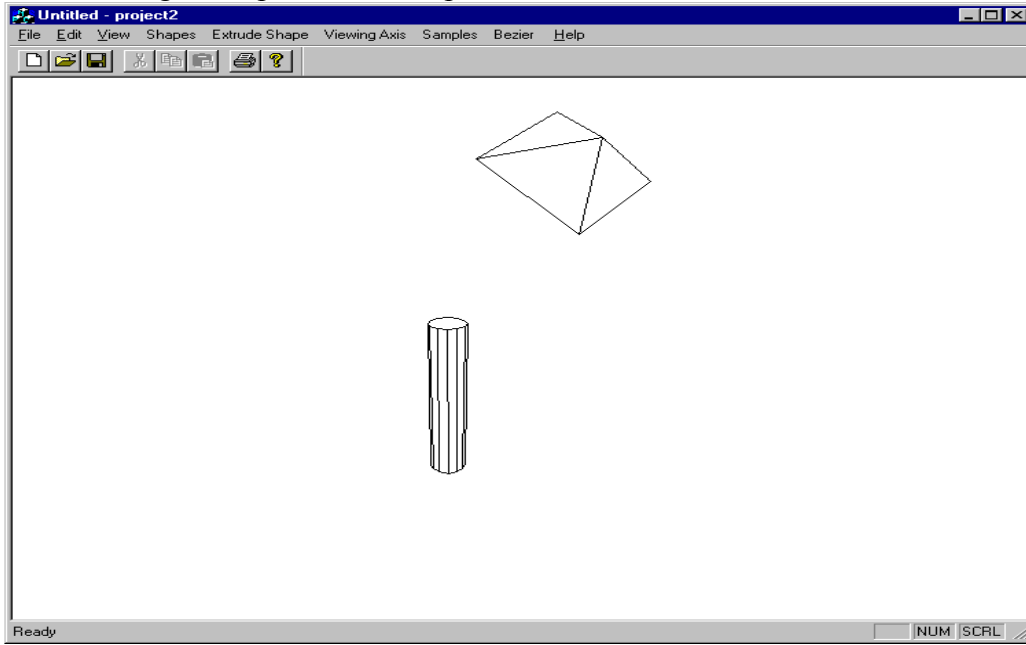


*Example for using “i” key to zoom in objects.*

## 7. *Simple Animation.*

The animation is done by pressing a button on the menu bar to start the animation. We have implemented some path for the objects to follow so that so that we can see the object moves along a specific path.

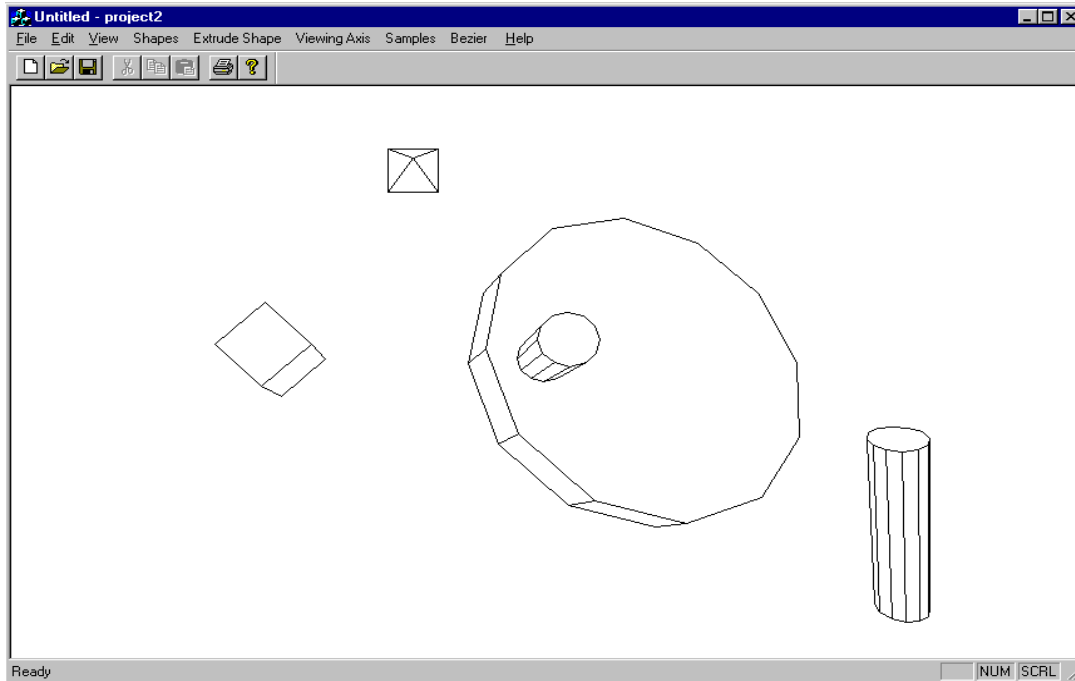
The following is the print screen capture of the animation.



*The objects are moving and rotating about a line.*

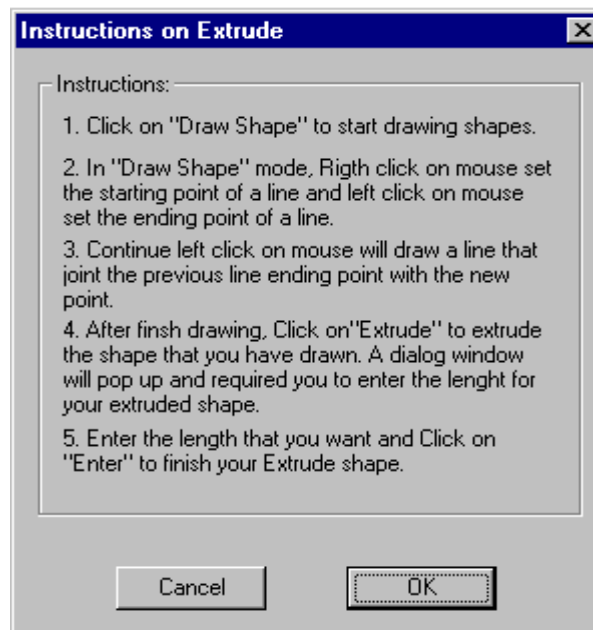
## 8. Scene display.

We have create a scene with different orientation of the objects so that the user can use some of the feature that we have mention above to see the objects.



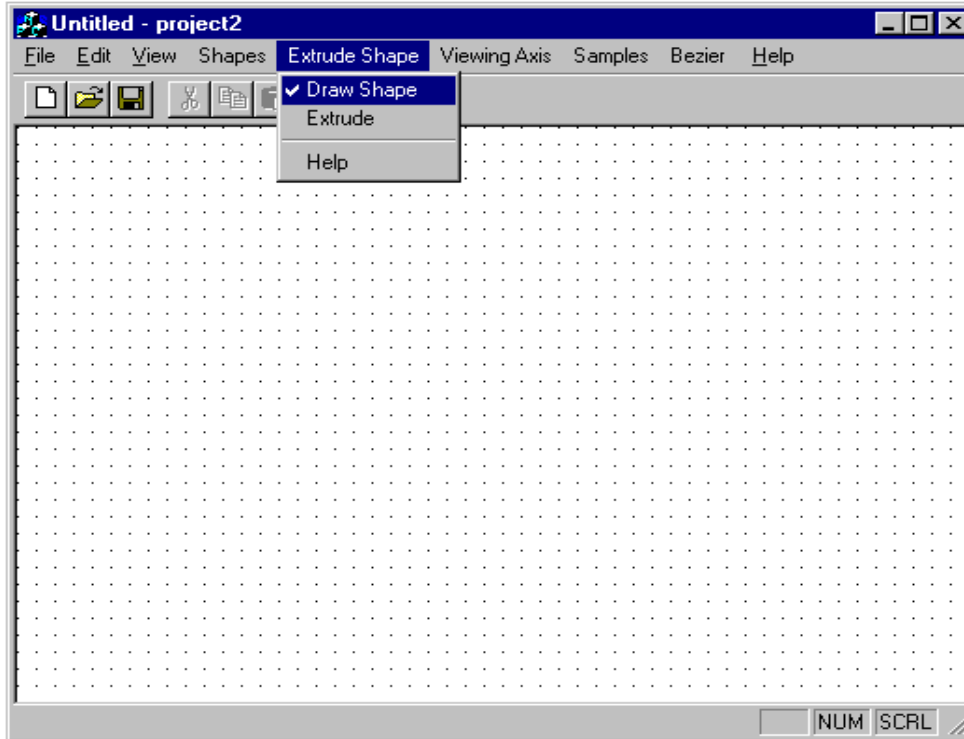
## 9. Extrude User define Shapes.

The user can also draw a shape and extrude it to a desired distance. However, the shape that the user draws must be convex shape. A help dialog box is provided to help the user on how to produce a extrude shape.

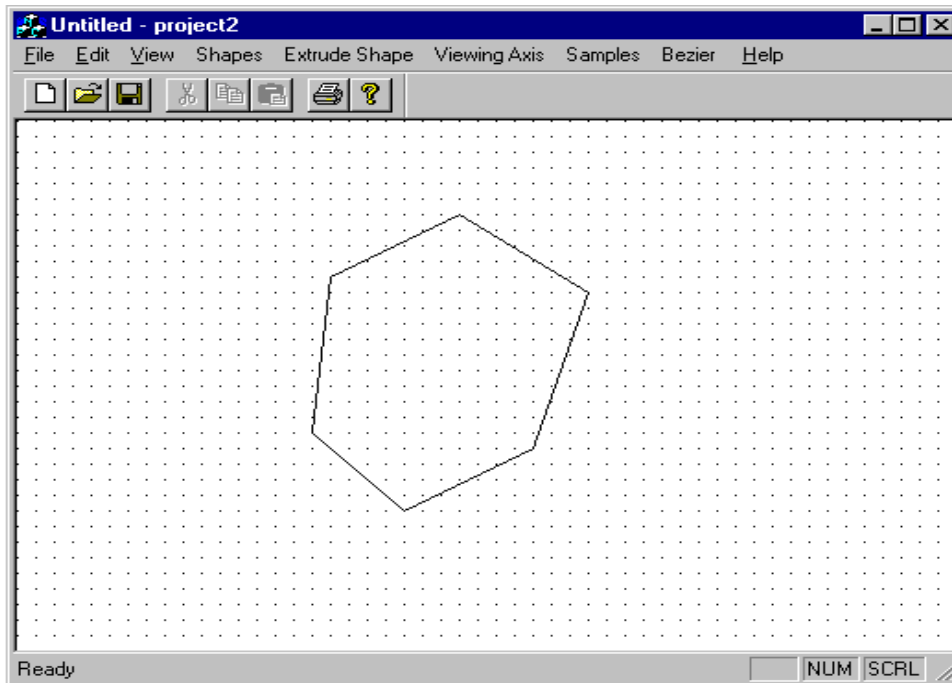


*Dialog Box on Helps for Extrude Shape.*

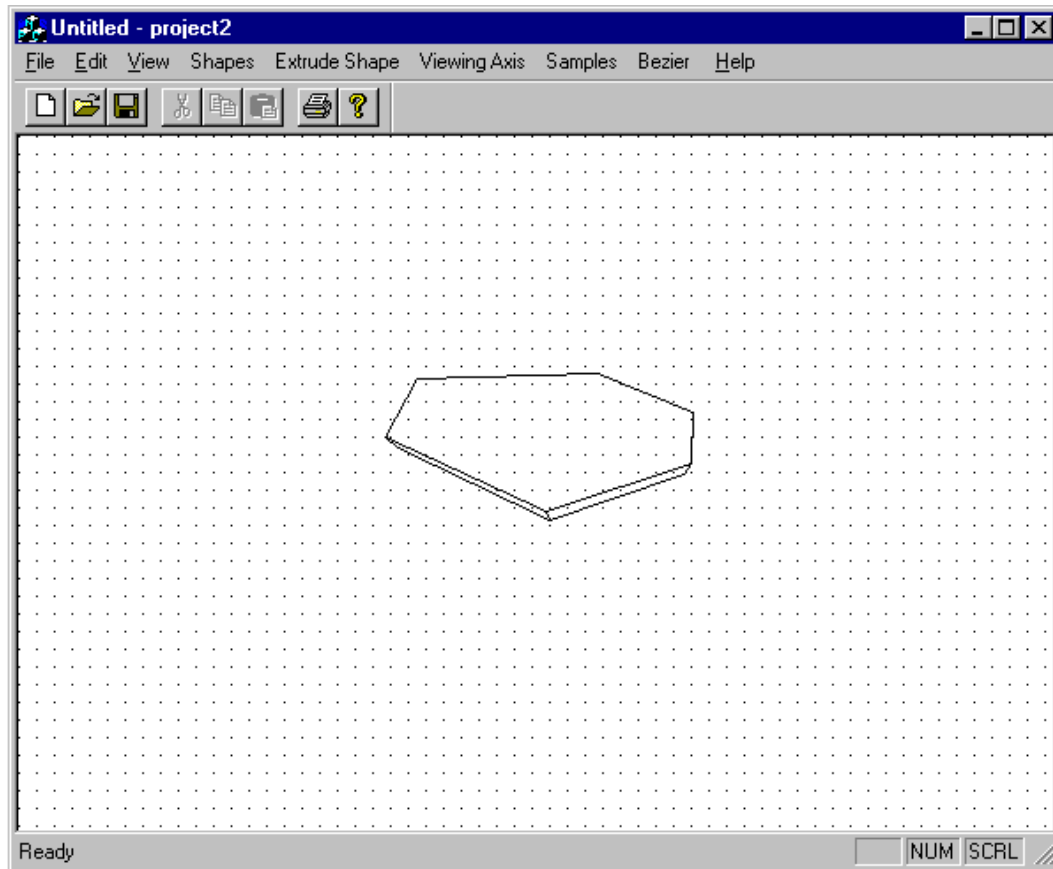
After the shape is drawn using mouse click, which is right button specifying the first point and left button specifying the second point. And consequently pressing left mouse button will draw a line that connects the last point drawn.



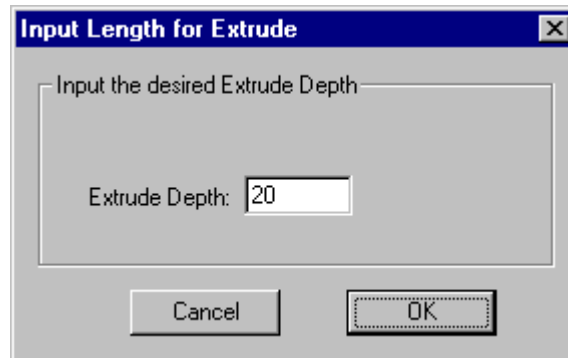
*When the "Draw Shape" is selected, grids will appears on the screen.*



*A shape is drawn with the "snap on grid" function also available.*



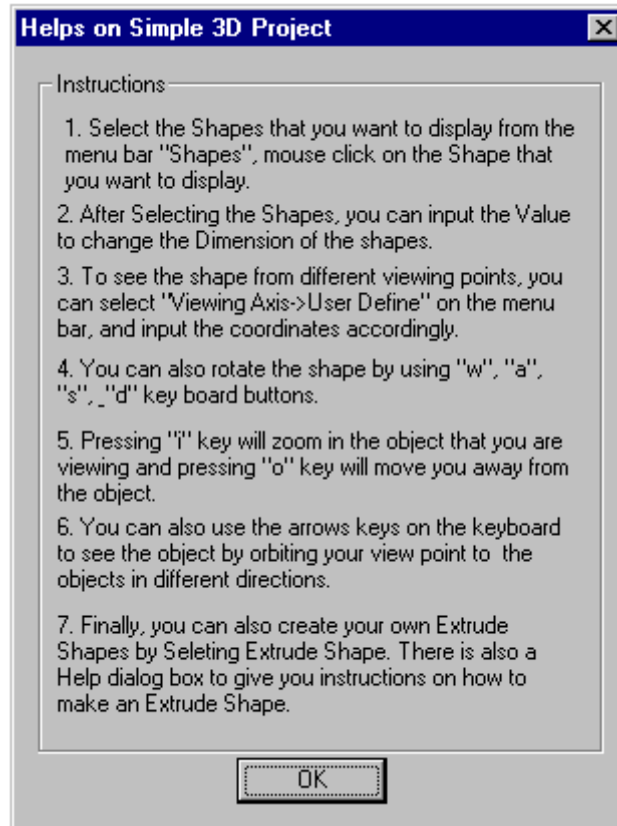
*a shape being produced with user can specify the depth of the extrude.*



*A dialog box pop up to ask user input the depth of the extrude.*

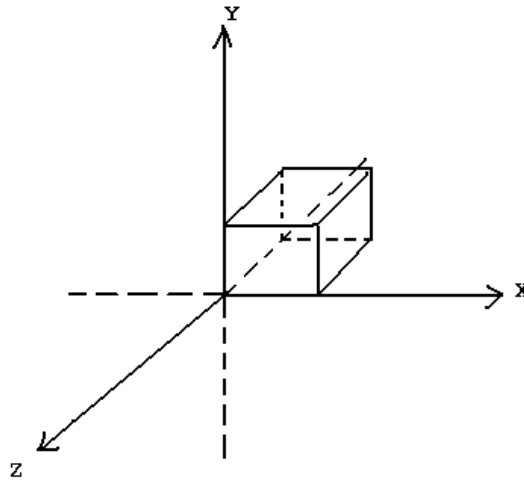
## 10. Help Dialog Box.

We have created a help dialog box that gives the user instruction on how to draw a shape and extrude it. This will help the user to understand how the program works.



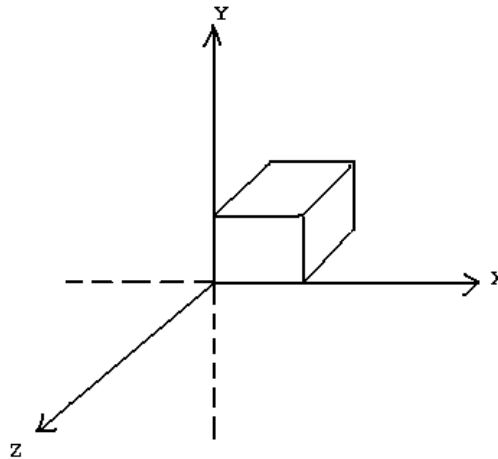
## PROGRAM FEATURES, CALCULATIONS, & EXPLANATIONS.

### 1. *Hidden line removal approach.*



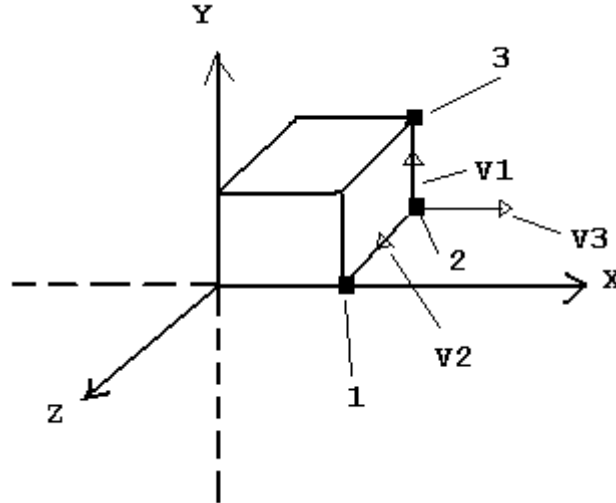
*Figure 1. A cube with Hidden Line not removed.*

Showing in Figure 1 a cube with hidden line dashed out. In this project, we need to removed these hidden line such that only visible line are shown, this is shown in figure 2.



*Figure 2. Cube with hidden line removed.*

In this case, we need to do the “dot product test”. In our modified Trace Function code, we first get the consequences three point of a polygon to construct two vector on the polygon. As shown in figure 3.



we now have point 1( $X_1, Y_1, Z_1$ ), 2( $X_2, Y_2, Z_2$ ), and point 3( $X_3, Y_3, Z_3$ ). using point 1 and point 2, we can create a surface vector of  $V_2$  and a surface vector of  $V_1$ , where,

$$V_1 = (X_3 - X_2)i + (Y_3 - Y_2)j + (Z_3 - Z_2)k$$

$$V_2 = (X_1 - X_2)i + (Y_1 - Y_2)j + (Z_1 - Z_2)k$$

Then we find the normal vector of the surface by taking cross product of  $V_1$  and  $V_2$ .

$$V_3 = V_1 \times V_2$$

$$V_3 = v_{3x}i + v_{3y}j + v_{3z}k$$

Our viewing point is define at ( $V_{px}, V_{py}, V_{pz}$ ) and thus our viewing vector  $V_{vec}$  with respect to the origin will be:

$$V_{vec} = V_{px}i + V_{py}j + V_{pz}k$$

Let our test vector to be  $TestVec$ , was then define as the vector between the viewing point and point 2 on figure 2.

$$TestVec = (V_{px} - X_2)i + (V_{py} - Y_2)j + (V_{pz} - Z_2)k$$

$$TestVec = (TV_{px})i + (TV_{py})j + (TV_{pz})k$$

The dot product test will be taken for the normal vector and the Test Vector. If the dot product returns a positive value, the surface is then visible and must be drawn. If the dot product returns a negative value, which means the surface is a hidden surface and should not be displayed on the screen.

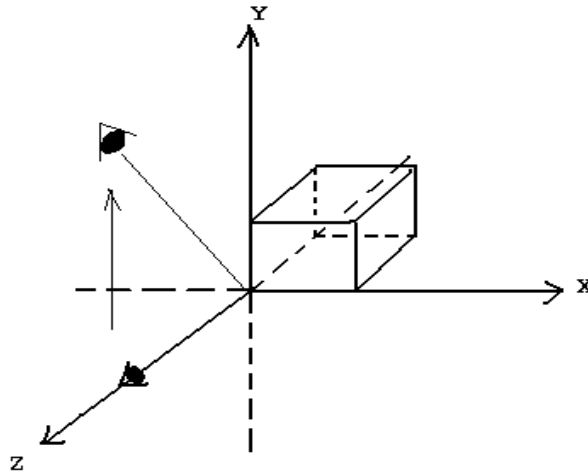
The dot product is:

$$TestVec = (v_{3x} \times V_{px}) + (v_{3y} \times V_{py}) + (v_{3z} \times V_{pz})$$

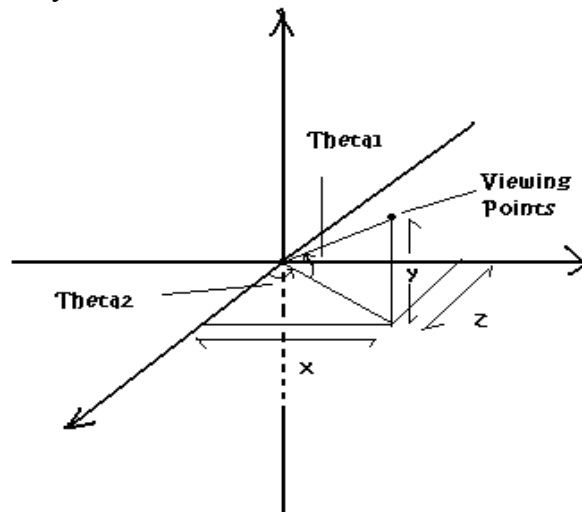
These steps are written in code to calculate whether a polygon face is to be drawn.

## 2. Rotation viewing approach.

Rotation viewing allowed the user to specify a viewing axis and passed through the origin and sees the object from different angle.



The rotation viewing approach is done by rotating the object in the opposite direction of the viewing point angle. This will give the same effect as we look from different view point to the object. This can be done in two steps, which is rotate around the x axis and rotating around the y axis.



With one viewing point, we are able to determine the angle to be rotated. As the user input the required viewing point, we can calculate theta1 and theta 2, then, by rotating respect to y-axis for theta 2 and rotating with respect to x-axis for theta 2 can give us the desired viewing angle.

### ***3. Orbiting viewing approach.***

Orbiting is the feature that allows the user to move around the objects, as if covered by an imaginary sphere. The user can view the objects through any point on the sphere. To enable this feature, we have used a ‘Global Viewing Strategy’. Once the user has placed all the required shapes at their respective orientations, the global viewing can be used. The basic idea is not to rotate the objects about their individual axes but to rotate them as one whole object about its center point.

The center point here is the center point that we use to position objects, that is (320,240). Hence, the objects on the screen are rotated about the center, and not about their base points. For global viewing we have two rotations, one about the Y-Axis and the other about the X-Axis. Both of these movements combined together with a constant viewing distance, give the user a “Global Viewing Experience”, whereon the user can view the object from any desired angle.

### ***4. User define Extruded shape.***

Extrude is the feature which allows the user to draw an object using the Line Drawing capability of our software. After drawing the required shape, the user is prompted to input the depth of extrusion, upon inputting this the Extrude function projects the drawn face by the depth to produce an Extruded Solid of the required shape and depth.

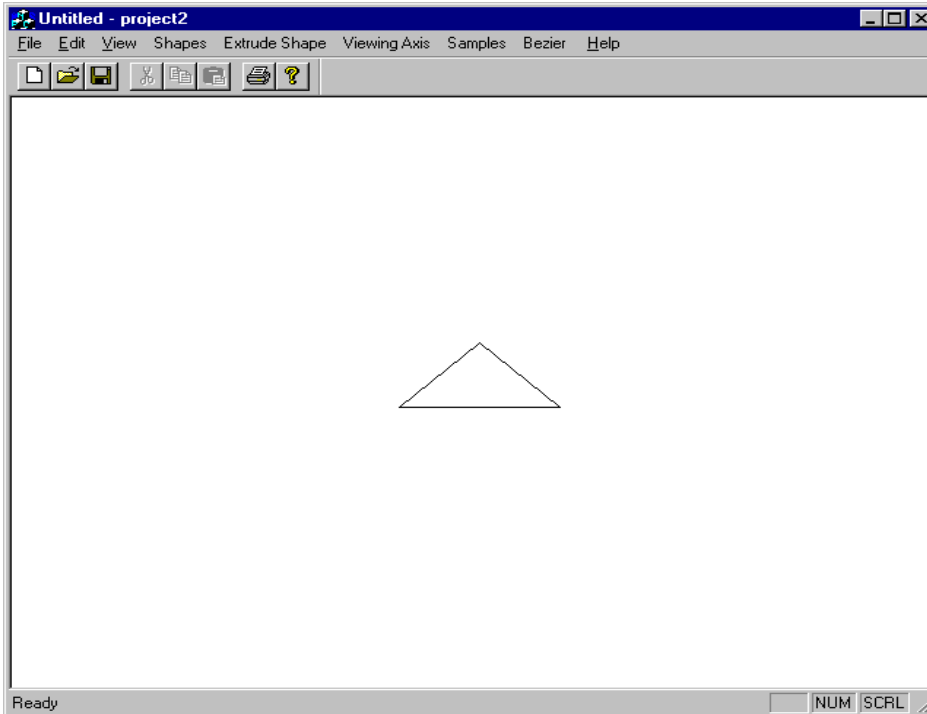
To achieve this functionality, we created a new class known as the CExtrude. This class is modeled on the CylShapes class. The CylShapes class uses the radius and the incremental angles to calculate the coordinates of the face (top or bottom) of the cylinder. The coordinates for the back face are then just the same but start from the last point on the front face and the numbering is opposite, to account for visibility problems.

The same concept is used to generate extruded shapes, except that we now use the coordinates that are obtained when drawing the required shape, using lines. These coordinates are now the coordinates of the front face of the object. The coordinates of the back face are calculated as before; the ordering of the polygons too, is calculated as before, we then go ahead with calling the Trace function to draw the Extruded Object.

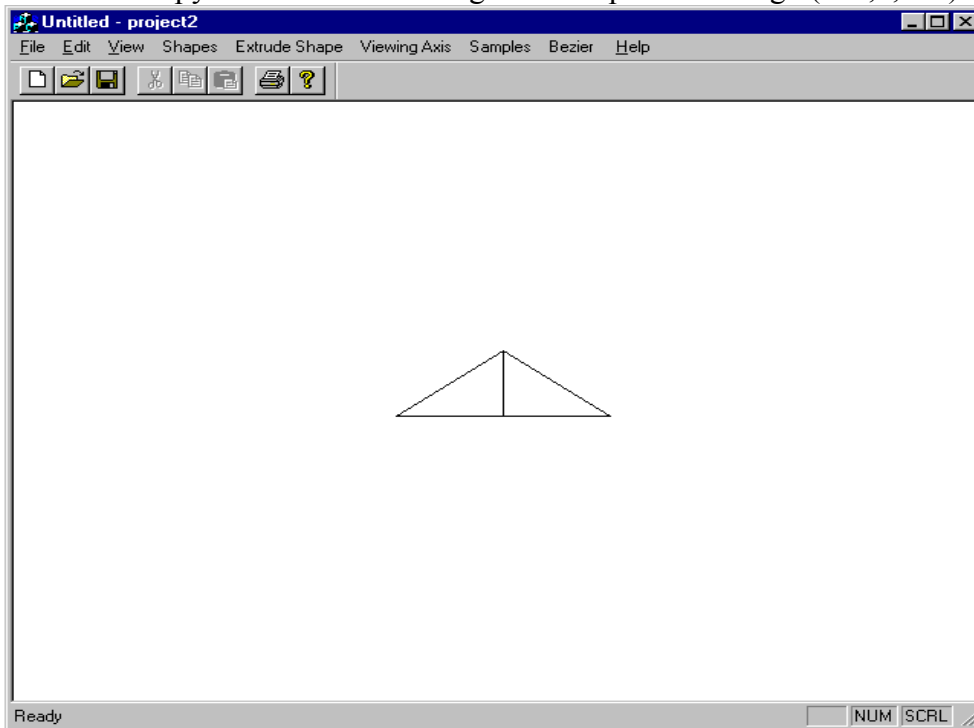
The Code for Extrude Class is listed in the Code Listing Section.

***PRINT SCREEN OF PYRAMID SHAPE FOR TESTING THE PROGRAM.***

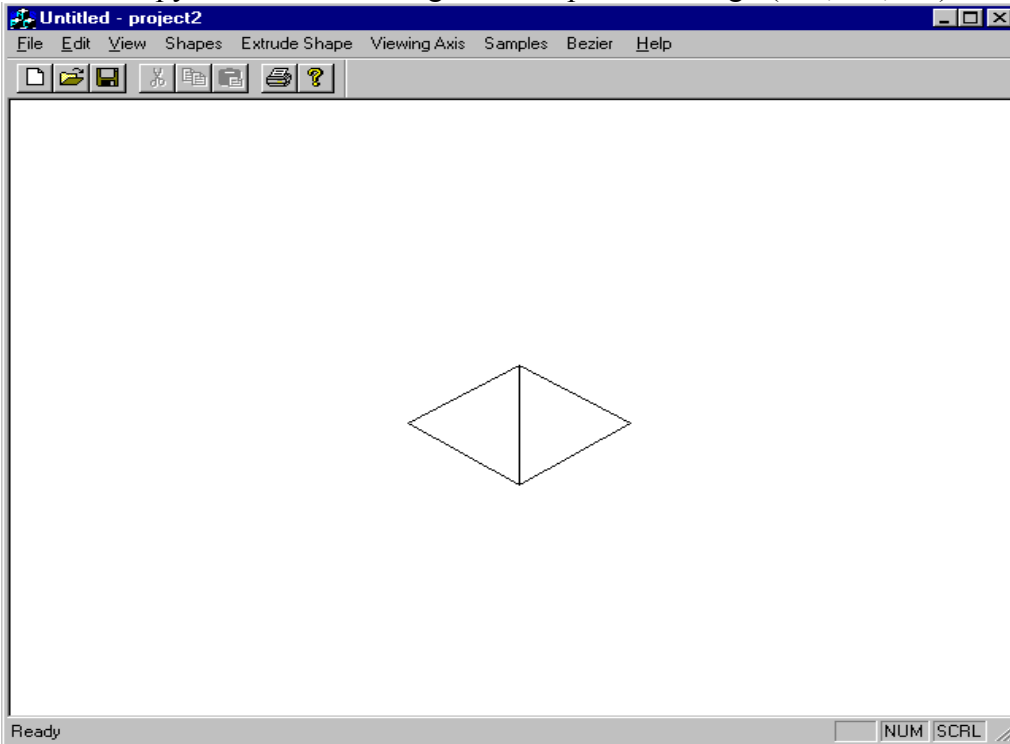
1. View the pyramid directly from point (0,0,500).



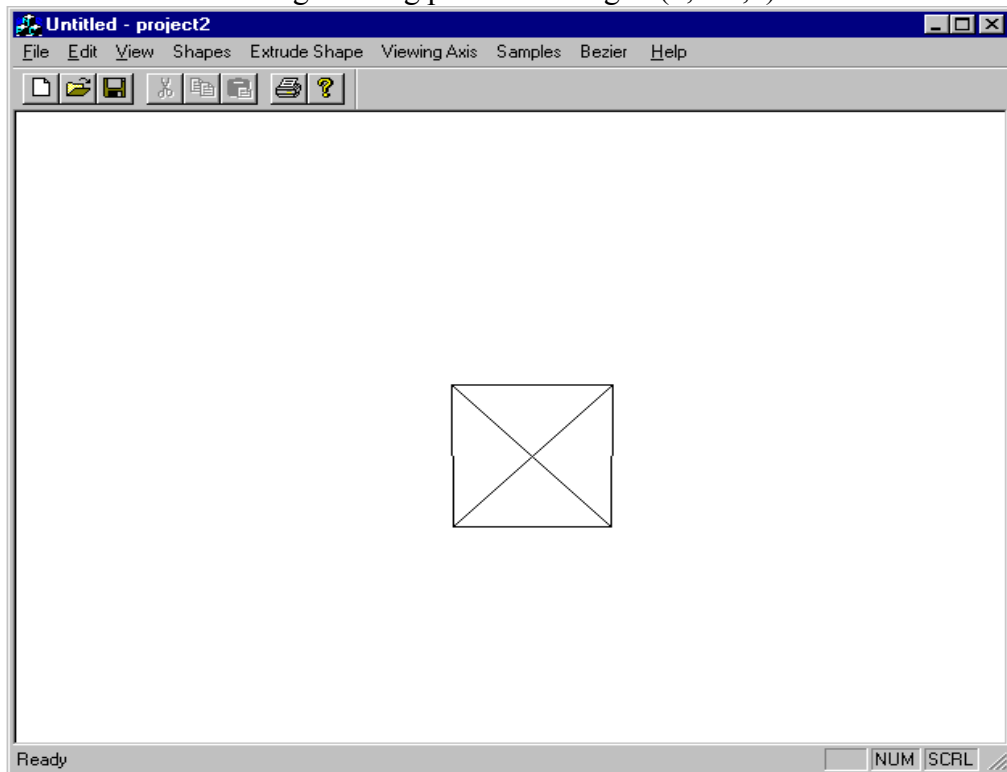
2. View the pyramid from a viewing axis that passed through (100,0,100).



3. View the pyramid from viewing axis that passed through (100,100,100).

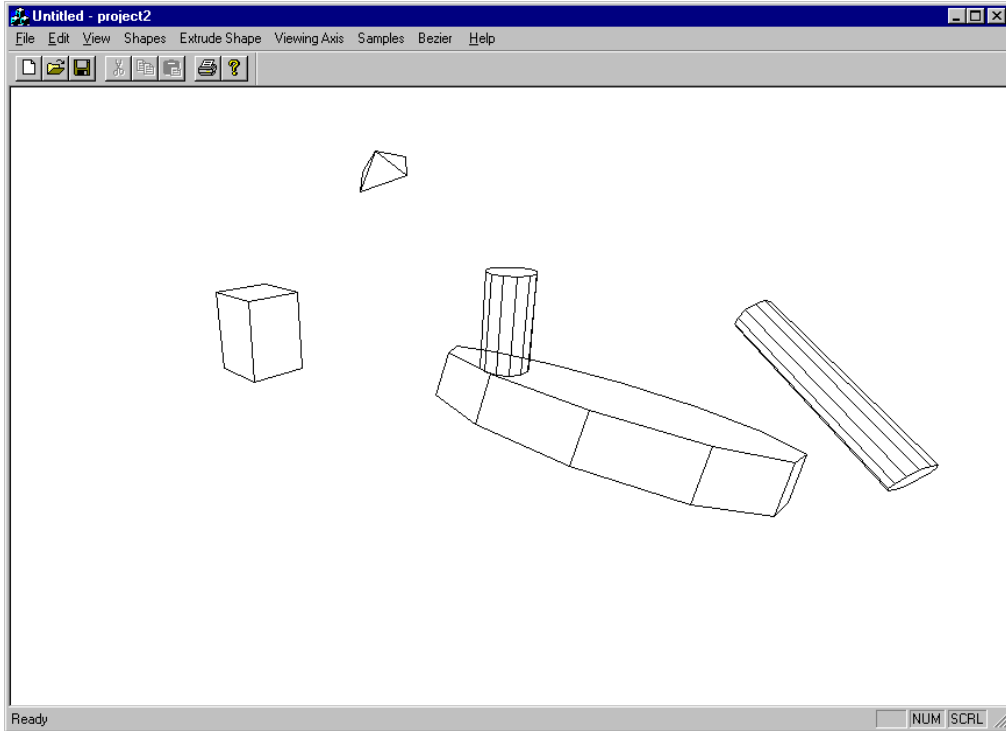


4. Additional interesting viewing point. Viewing at (0,200,0)

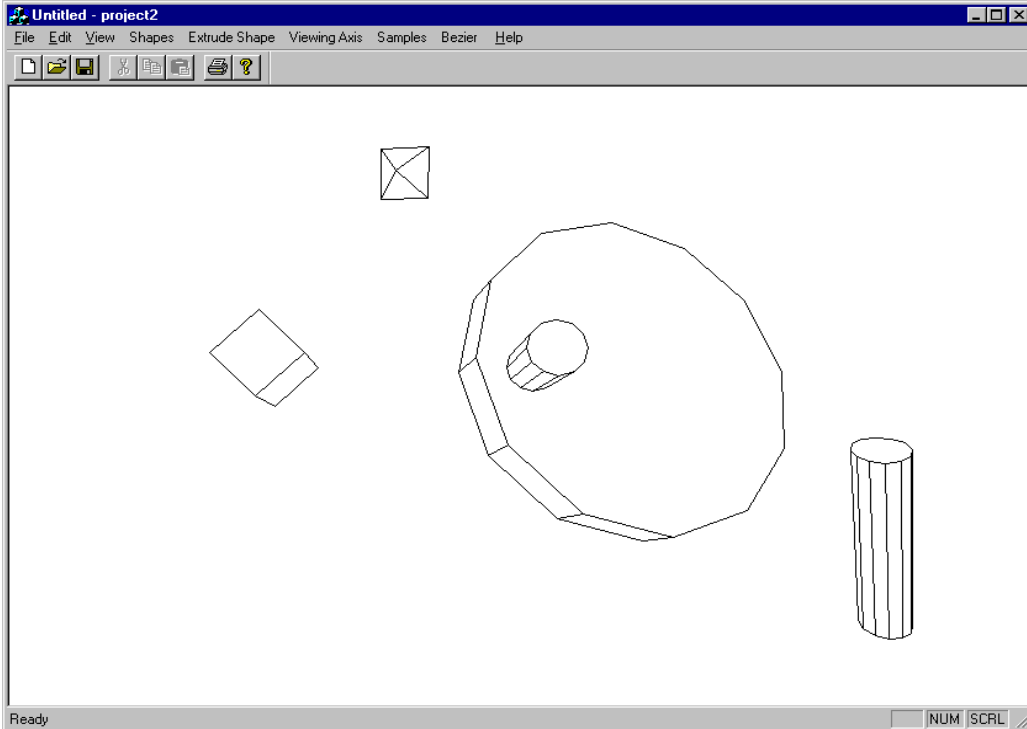


PRINT SCREEN OF SHAPES VIEW FROM DIFFERENT VIEWING POSITION.

1. Three shape being view from point ( ).



2. Three shapes viewing from a viewing angle using the orbiting viewing approach.



## CODE LISTING.

### *A. Listing of Modified Trace Function.*

```
class Polygns : public Points {
protected:
    int npolys;
    int npnts[100];
    int ptvec[100][100];

public:
    Polygns(){};
    void Polygns::Trace(CDC* aDC){
        int zvp = 500;
        //variables added to complete the visibility test
        double normalvec[3]={0,0,0}; //Initialize Normal vector
        double survec1[3]={0,0,0}; //Initialize surface vector1
        double survec2[3]={0,0,0}; //Initialize surface vector1
        double testvec[3]={0,0,0}; //Initialize Test vector
        double viewvec[3]={0,0,500}; //Viewing Vector
        double visicheck[13]={0}; //Array for storing data
        double dotproduct=0; //Variable to store dotproduct value

        //***** Start visibility test*****
        //For loop to get the first three points from a polygon face.
        for (int icnta = 0; icnta < npolys; icnta++){
            int jcnt=0;
            int jmax = npnts[icnta];
            ptvec[icnta][jmax] = ptvec[icnta][0];
            //get point 1
            double x = xb + xp[ptvec[icnta][jcnt]];
            double y = yb + yp[ptvec[icnta][jcnt]];
            double z = zb + zp[ptvec[icnta][jcnt]];

            //get point 2
            double xp1 = xb + xp[ptvec[icnta][jcnt+1]];
            double yp1 = yb + yp[ptvec[icnta][jcnt+1]];
            double zp1 = zb + zp[ptvec[icnta][jcnt+1]];

            //get point 2
            double xp2 = xb + xp[ptvec[icnta][jcnt+2]];
            double yp2 = yb + yp[ptvec[icnta][jcnt+2]];
            double zp2 = zb + zp[ptvec[icnta][jcnt+2]];

            //Define vector for visibility test.
            survec1[0]=xp2-xp1;
            survec1[1]=yp2-yp1;
            survec1[2]=zp2-zp1;

            survec2[0]=x-xp1;
            survec2[1]=y-yp1;
            survec2[2]=z-zp1;

            //calculating the normal vector of each polygon
            normalvec[0]=survec1[1]*survec2[2]-survec2[1]*survec1[2];
```



## B. Extrude Function Class.

-Extrude Function Class are create for drawing a user define Extrude shape.

```

//*****EXTRUDE CLASS BEGINS HERE*****
class CExtrude : Polygns
{
    public:

        CExtrude(){};

        void CExtrude::RotateX(double thetaX){
            Points::PRotateX(thetaX);
        };

        void CExtrude::RotateY(double thetaY){
            Points::PRotateY(thetaY);
        };

        void CExtrude::RotateZ(double thetaZ){
            Points::PRotateZ(thetaZ);
        };

        void CExtrude::Dmove(double dx, double dy, double dz){
            Basepoint::DmoveBP(dx,dy,dz);
        };

        void CExtrude::Move(double nx, double ny, double nz){
            Basepoint::MoveBP(nx,ny,nz);
        };

        void CExtrude::Show(CDC* aDC){
            Polygns::Trace(aDC);
        };

        void CExtrude::Drawshapes (double ibx, double iby, double ibz, int
nseg,float Length,float* p1,float* p2)
        {
            xb = ibx;
            yb = iby;
            zb = ibz;

            for (int i = 0; i<nseg; i++)
            {
                xp[i] = *(p1+i);
                yp[i] = *(p2+i)-30;
                zp[i] = 0;
                xp[nseg+i] = *(p1+i);
                yp[nseg+i] = *(p2+i);
                zp[nseg+i] = -Length;
            }

            tnpts = 2*nseg;

            npolys = nseg+2;//there should have 12 polygon +2 faces
            npnts[0] = nseg;
            npnts[1] = nseg;

            for (i=0; i<nseg; i++)//numbering for the front face
            {ptvec[0][i]=i;}

            for (i=0; i<nseg; i++)//numbering for the back face
            {ptvec[1][i]=nseg+(nseg-1)-i;}//add one less than nseg

```

```

        for (i=0; i<nseg-1; i++)
        { //numbering for the 12 polygons      for (i=0; i<nseg-1; i++)

        npnts[i+2]=4;
        ptvec[i+2][0]=i+1;//1;
        ptvec[i+2][1]=i;//0;
        ptvec[i+2][2]=nseg+i;//3;
        ptvec[i+2][3]=nseg+i+1;//4;

        }
        npnts[nseg+1]=4;
        ptvec[nseg+1][0]=0;
        ptvec[nseg+1][1]=nseg-1;
        ptvec[nseg+1][2]=2*nseg-1;
        ptvec[nseg+1][3]=nseg;

    };
};

```

### ***C. Bezier Function Class.***

Bezier function class is create for drawing a Bezier curve on the screen.

```

//*****BEZIER CLASS STARTS HERE*****
class CBezier : Polygns
{
    public:

        CBezier(){};

        void CBezier::RotateX(double thetaX){
        Points::PRotateX(thetaX);
        };

        void CBezier::RotateY(double thetaY){
        Points::PRotateY(thetaY);
        };

        void CBezier::RotateZ(double thetaZ){
        Points::PRotateZ(thetaZ);
        };

        void CBezier::Dmove(double dx, double dy, double dz){
        Basepoint::DmoveBP(dx,dy,dz);
        };

        void CBezier::Move(double nx, double ny, double nz){
        Basepoint::MoveBP(nx,ny,nz);
        };

        void CBezier::Show(CDC* aDC){
        Polygns::Btrace(aDC);
        };

        void CBezier::Drawshapes1(double ibx, double iby, double ibz)
        {
            xb = ibx;
            yb = iby;
            zb = ibz;
            double t=0;
            float bx[4]={0};
            float by[4]={0};
            float bz[4]={0};

```

```

        bx[0]=-300;
        by[0]=100;
        bz[0]=0;

        bx[1]=200;
        by[1]=200;
        bz[1]=0;

        bx[2]=-100;
        by[2]=-100;
        bz[2]=0;

        bx[3]=400;
        by[3]=-100;
        bz[3]=0;

    for (int i = 0; i<61; i++)
    {
        t=i/60.0;
        xp[i]          =bx[0]*pow((1-t),3)+bx[1]*3*t*pow((1-
t),2)+bx[2]*3*t*t*(1-t)+bx[3]*pow(t,3);
        yp[i]          =by[0]*pow((1-t),3)+by[1]*3*t*pow((1-
t),2)+by[2]*3*t*t*(1-t)+by[3]*pow(t,3);
        zp[i]          =bz[0]*pow((1-t),3)+bz[1]*3*t*pow((1-
t),2)+bz[2]*3*t*t*(1-t)+bz[3]*pow(t,3);
    }
};

```

## USER INSTRUCTIONS:

### *General:*

For brief user instruction, you can click on Help-> User Help on the Menu Bar. There, you will find simple instruction on how to use this 3D CAD software.

### (1) Display Of Shapes:

We have created 3 basic shapes, which are Cube, Cylinder, and the Pyramid. In addition to this there is the Extruded Shapes, using which we can generate any kind of shape. To display one of the basic shapes there are two options,

- 1) Individual display using the down menu and clicking on the required basic shape, this will pop up a dialog box thru which the dimensions are input. Clicking OK after this will display the shape.
- 2) The second option is to use the User Select option available under shapes. Clicking on this will pop up a comprehensive dialog box, where the user can select one or all of the basic shapes to display and input their dimensions.

## (2) Extrude Shapes:

To extrude shapes, click on Extrude from the menu bar, this will bring down two steps. One is to draw the required shape. Clicking on this will turn the grid and snap to grid on, the user can then draw the required shape using the mouse. A right click on the screen will be the starting point for the drawing, a left click ends a line, successive left clicks ending at the starting point, complete the drawing.

The second step is to click on Extrude Shape, this will pop up a dialog box asking for the extrude depth; once that is input, the extruded shape is drawn.

## (3)Viewing Axis Manipulation:

To view the object thru an axis passing thru any desired point, the user has to click on this option, this again pops up a dialog box and the coordinates for the desired point are entered. Once the user clicks OK, the view displayed on the screen is the one that would be seen if the user were to actually move to the specified point and look at the object.

## (4)Bezier Curve Display:

Clicking on this button displays a sample Bezier curve on the screen. We could have made this option user interactive, where the user manually inputs the number of control points and their coordinates for drawing the curve. Then we could generate curves to model any required surface.

## (5)Samples:

Clicking on Samples gives the user two options:

- 1) A simple animation
- 2) An animation following a Bezier curve.

The other two options are to stop the animation.

## (6)Functional Keys:

- 1) Arrow up, down, left, right: Global Rotation
- 2) W, S, D, A: Zoom in , Zoom out, Rotate X, Rotate Z.
- 3) I,O: Zoom in and out when viewing from any axis.

## CONCLUSION:

We complete the requirements for this project and we have added some dialog boxes to increase the interaction between the user and the program. We have tried out the many possibilities in this project but some of the alternatives requirements we are not able to accomplish. For the part that we have finish, we found that it was working well and if we have more time one it, we could probably added more interesting features to increase its usability and this should be a very interesting software. We enjoy doing this project and we have learn a lot from this project.

