LAB MANUAL

COMPUTER GRAPHICS AND MULTIMEDIA

CONTENTS

1. INTRODUCTION TO COMUTER GRAPHICS

What is computer graphics?

• ***Computer graphics*** **deals with all aspects of creating images with a computer**

Hardware

Software

Applications

Computer graphics are graphics created using computers and, more generally, the representation and manipulation of image data by a computer. The development of computer graphics, or simply referred to as CG, has made computers easier to interact with, and better for understanding and interpreting many types of data. Developments in computer graphics have had a profound impact on many types of media and have revolutionized the animation and video game industry. Typically, the term *computer graphics* refers to several different things: · the representation and manipulation of image data by a computer · the various technologies used to create and manipulate images · the images so produced and The subfield of computer science which studies methods for digitally synthesizing and manipulating visual content **Applications:** · Computer Aided Design · Computer simulation · Digital art · Education · Graphic design · Information visualization · Scientific visualization · Video Games · Virtual reality · Web design

Graphics mode Initialization

First of all we have to call the initgraph function that will intialize the graphics mode on the computer. initigraph have the following prototype. void initgraph(int far *graphdriver, int far *graphmode, char far *pathtodriver); Initgraph initializes the graphics system by loading a graphics driver from disk (or validating a registered driver) then putting the system into graphics mode. Initgraph also resets all graphics settings (color, palette, current position, viewport, etc.) to their defaults, then resets graphresult to 0.

***graphdriver** Integer that specifies the graphics driver to be used. You can give graphdriver a value using a constant of the graphics_drivers enumeration type.

***graphmode** Integer that specifies the initial graphics mode (unless *graphdriver = DETECT). If *graphdriver = DETECT, initgraph sets *graphmode to the highest resolution available for the detected driver. You can give *graphmode a value using a constant of the graphics_modes enumeration type.

***pathtodriver** Specifies the directory path where initgraph looks for graphics drivers (*.BGI) first.

1. If they‟re not there, initgraph looks in the current directory.
2. If pathtodriver is null, the driver files must be in the current directory.

*graphdriver and *graphmode must be set to valid graphics_drivers and graphics_mode values or you‟ll get unpredictable results. (The exception is graphdriver = DETECT.) After a call to initgraph, *graphdriver is set to the current graphics driver, and *graphmode is set to the current graphics mode. You can tell initgraph to use a particular graphics driver and mode, or to autodetect the attached video adapter at run time and pick the corresponding driver. If you tell initgraph to autodetect, it calls detectgraph to select a graphics driver and mode.

Basic Function : Cleardevice() Clears all previous graphical outputs generated by the previous programs.Its a good practice to include this method at the starting of each program. cleardevice()

gotoxy() This will initialize the graphics cusor to the specified co-ordiante.In C gotoxy function is used very frequently to locate the cursor at different locations whenever as necessary. Syntax : gotoxy(x,y)

putpixex() It will colour the pixel specified by the co-ordinates.

Syntax: putpixel(x,y,WHITE)

outtextxy() This method is used to display a text in any position on the screen.The numeric coordinates are substituted for x and y. Syntax: outtextxy(x,y,"HELLO") **rectangle()** Draws a rectangle according to the given parameter x and y are the top-left corner co-ordinates. Syntax : rectangle(int left, int top, int right, int bottom)

circle() Draws a circle with x,y as the center . Syntax: circle(x,y,radius)

line() Draws a line as per the given co-ordinates. Syntax : line(int startx, int starty, int endx, int endy)

moveto() Cursor is moved from the current location to the specified location dx,dy.These parameters can also be used as incremental values. Syntax : moveto(dx,dy)

lineto() Draws a line from its current location to the co-ordinate(x,y) Syntax : lineto(x,y)

ellipse() Draws the ellipse with the specified angles and coordinates. Syntax : ellipse(x-centre,y-center,starting_angle,ending_angle,x_radius,y_radius)

drawpoly() Draws a polygon with (num_of_points +1) edges.The array 'points' int points[ ]=(x1,y1,x2,y2,x3,y3...) Syntax : drawpoly(num_of_points + 1, points) **settextstyle()** The fonts available are :**TRIPLEX_FONT, SMALL_FONT** SANS_SERIE_FONT, GOTHIC_FONT The direction can be changed as HORIZ_DIR or VERT_DIR**,** The charecter size increases from **1** to **10** Syntax : settextstyle(DEFAULT_FONT,HORIZ_DIR,1)

setfillstyle() The fill styles avaliable are SOLID_FILL, LINE_FILL, HATCH_FILL, SLASH_FILL etc. Syntax : setfillstyle(SOLID_FILL,RED)

setcolor() Sets the color

Syntax : setcolor(color_name)

delay() Cause a pause in execution of the program 1000ms= 1 second Syntax : delay(100)

closegraph() Terminates

2. LAB REQUIREMENTS

Software requirements: Turbo C++ or Turbo C compiler, MAYA

Operating System: Windows XP (SP2)

Hardware requirements:

Windows and Linux: Intel 64/32 or AMD Athlon 64/32, or AMD Opteron processor

2 GB RAM

2 GB hard disk space

Qualified hardware-accelerated OpenGL graphics card

3. LIST OF EXPERIMENTS

(As prescribed by G.G.S.I.P.U)

COMPUTER GRAPHICS & MULTIMEDIA LAB

Paper Code: ETCS-257	**L**	**T**	**C**
Paper: Computer Graphics & Multimedia Lab	**0**	**2**	**1**

List of Experiments:

1. Study of Fundamental Graphics Functions.
2. Implementation of Line drawing algorithms: DDA Algorithm, Bresenham's Algorithm
3. Implementation of Circle drawing algorithms: Bresenham's Algorithm, Mid-Point Algorithm.
4. Programs on 2D and 3D transformations
5. Write a program to implement Cohen Sutherland line clipping algorithm
6. Write a program to draw Bezier curve.
7. Using Flash/Maya perform different operations (rotation, scaling move etc..) on objects
8. Create a Bouncing Ball using Key frame animation and Path animation.

NOTE: - At least 8 Experiments out of the list must be done in the semester.

4. LIST OF EXPERIMENTS

(Beyond the syllabus prescribed by G.G.S.I.P.U)

COMPUTER GRAPHICS & MULTIMEDIA LAB

List of Experiments

1. To Study various in build graphics functions in C library.
2. Write a program to draw a line using DDA algorithm.
3. Write a program to draw a line using Bresenham‘s algorithm.
4. Write a program to draw a circle using midpoint algorithm.
5. Write a program to draw a circle using Bresenham‘s algorithm.
6. Write a program to draw a rectangle using line drawing algorithm.
7. Write a program to perform 2D Transformation on a line.
8. Write a program to perform shear transformation on a rectangle.
9. Write a program to rotate a circle (alternatively inside and outside) around the circumference of another circle.
10. Write a program to draw a car using in build graphics function and translate it from bottom left corner to right bottom corner of screen.
11. Write a program to draw balloons using in build graphics function and translate it from bottom left corner to right top corner of screen.
12. Write a program to draw a cube using in build library function and perform 3D transformations

i) Translations in x, y, z directions

ii) Rotation by angle 45^0 about z axis, rotation by 60^0 about y-axis in succession.

iii) Scaling in x-direction by a factor of 2, scaling in y- direction by a factor of 3.

12. Write a program to implement line clipping (Cohen Sutherland algorithm).
13. Write a program for making Bezier curve.
14. Write a program to study various in build functions for 2D drawing in MAYA software.
15. Write a program to show animation of a ball moving in a helical path.
16. Write a program to show animation of solar system.

BASIC GRAPHICS FUNCTION

1) INITGRAPH

- Initializes the graphics system.

Declaration

- Void far initgraph(int far *graphdriver)

Remarks

- To start the graphic system, you must first call initgraph.
- Initgraph initializes the graphic system by loading a graphics driver from disk (or validating a registered driver) then putting the system into graphics mode.
- Initgraph also resets all graphics settings (color, palette, current position, viewport, etc) to their defaults then resets graph.

2) GETPIXEL, PUTPIXEL

- Getpixel gets the color of a specified pixel.
- Putpixel places a pixel at a specified point.

Decleration

- Unsigned far getpixel(int x, int y)
- Void far putpixel(int x, int y, int color)

Remarks

- Getpixel gets the color of the pixel located at (x,y);
- Putpixel plots a point in the color defined at (x, y).

Return value

- Getpixel returns the color of the given pixel.
- Putpixel does not return.

3) CLOSE GRAPH

- Shuts down the graphic system.

Decleration

- Void far closegraph(void);

Remarks

- Close graph deallocates all memory allocated by the graphic system.
- It then restores the screen to the mode it was in before you called initgraph.

Return value

- None.

4) ARC, CIRCLE, PIESLICE

- arc draws a circular arc.
- Circle draws a circle
- Pieslice draws and fills a circular pieslice

Decleration

- Void far arc(int x, int y, int stangle, int end_angle, int radius);
- Void far circle(int x, int y, int radius);
- Void far pieslice(int x, int y, int stangle, int end_angle, int radius);

Remarks

- Arc draws a circular arc in the current drawing color
- Circle draws a circle in the current drawing color
- Pieslice draws a pieslice in the current drawing color, then fills it using the current fill pattern and fill color.

5) ELLIPSE, FILLELIPSE, SECTOR

- Ellipse draws an elliptical arc.
- Fillellipse draws and fills ellipse.
- Sector draws and fills an elliptical pie slice.

Decleration

- Void far ellipse(int x, int y, int stangle, int end_angle, int xradius, int yradius)
- Void far fillellipse(int x, int y, int xradius, int yradius)
- Void farsectoe(int x, int y, int stangle, int end_angle, int xradius, int yradius)

Remarks

- Ellipse draws an elliptical arc in the current drawing color.
- Fillellipse draws an elliptical arc in the current drawing color and than fills it with fill color and fill pattern.

- Sector draws an elliptical pie slice in the current drawing color and than fills it using the pattern and color defined by setfillstyle or setfillpattern.

6) FLOODFILL

- Flood-fills a bounded region.

Decleration

- Void far floodfill(int x, int y, int border)

Remarks

- Floodfills an enclosed area on bitmap device.
- The area bounded by the color border is flooded with the current fill pattern and fill color.
- (x,y) is a ‐seed point
  - If the seed is within an enclosed area, the inside will be filled.
  - If the seed is outside the enclosed area, the exterior will be filled.
- Use fillpoly instead of floodfill wherever possible so you can maintain code compatibility with future versions.
- Floodfill doesnot work with the IBM-8514 driver.

Return value

- If an error occurs while flooding a region, graph result returns ‗1‘.

7) GETCOLOR, SETCOLOR

- Getcolor returns the current drawing color.
- Setcolor returns the current drawing color.

Decleration

- Int far getcolor(void);
- Void far setcolor(int color)

Remarks

- Getcolor returns the current drawing color.
- Setcolor sets the current drawing color to color, which can range from 0 to getmaxcolor.
- To set a drawing color with setcolor , you can pass either the color number or the equivalent color name.

8) LINE, LINEREL, LINETO

- Line draws a line between two specified pints.

- Onerel draws a line relative distance from current position(CP).
- Linrto draws a line from the current position (CP) to(x,y).

Decleration

- Void far lineto(int x, int y)

Remarks

- Line draws a line from (x1, y1) to (x2, y2) using the current color, line style and thickness. It does not update the current position (CP).
- Linerel draws a line from the CP to a point that is relative distance (dx, dy) from the CP, then advances the CP by (dx, dy).
- Lineto draws a line from the CP to (x, y), then moves the CP to (x,y).

Return value

- None

9) RECTANGLE

- Draws a rectangle in graphics mode.

Decleration

- Void far rectangle(int left, int top, int right, int bottom)

Remarks

- It draws a rectangle in the current line style, thickness and drawing color.
- (left, top) is the upper left corner of the rectangle, and (right, bottom) is its lower right corner.

DDA Algorithm for line drawing.

- Start.
- Declare variables x,y,x1,y1,x2,y2,k,dx,dy,s,xi,yi and also declare gdriver=DETECT,gmode.
- Initialize the graphic mode with the path location in TC folder.
- Input the two line end-points and store the left end-points in (x1,y1).
- Load (x1,y1) into the frame buffer; that is, plot the first point. Put x=x1,y=y1.
- Calculate dx=x2-x1 and dy=y2-y1.
- If abs(dx) > abs(dy), do s=abs(dx).
  Otherwise s= abs(dy).
- Then xi=dx/s and yi=dy/s.
- Start from k=0 and continuing till k<s,the points will be
  - x=x+xi.
  - y=y+yi.
- Place pixels using putpixel at points (x,y) in specified colour.
- Close Graph.
- Stop.

Bresenham's algorithm for line drawing.

- Start.
- Declare variables x,y,x1,y1,x2,y2,p,dx,dy and also declare gdriver=DETECT,gmode.
- Initialize the graphic mode with the path location in TC folder.
- Input the two line end-points and store the left end-points in (x1,y1).
- Load (x1,y1) into the frame buffer; that is, plot the first point put x=x1,y=y1.
- Calculate dx=x2-x1 and dy=y2-y1,and obtain the initial value of decision parameter p as:
- p=(2dy-dx).
- Starting from first point (x,y) perform the following test:
- Repeat step 9 while(x<=x2).
- If p<0,next point is (x+1,y) and p=(p+2dy).
- Otherwise, the next point to plot is (x+1,y+1) and p=(p+2dy-2dx).
- Place pixels using putpixel at points (x,y) in specified colour.
- Close Graph.
- Stop

Algorithm to Clip a Line.

- Start.
- Initialize the graphic system using initgraph function.
- Get the input of window coordinates from the user and draw a window.
- Get the input of line coordinates from user and draw the line.
- Calculate the region code of each end point of line using relation given in steps 6 to step
- Let (x,y) be the coordinates of end point of line and (xmin,ymin), (xmax,ymax) be coordinates of world window
- If y –ymax = +ve
- MSB region code = 1.
- Else MSB region code = 0.
- If ymin – y = +ve
- Region code = 1.
- Else Region code = 0.
- If x – xmax = +ve
- Region code = 1.
- Else Region code = 0.
- If xmin – x = +ve
- LSB Region code = 1.
- Else LSB Region code = 0.
- Calculate region code of both end points.
- Logically and both region code.
- If Logically anded result is = 0
- Line is not a clipping candidate.
- Else.
- Line is a clipping candidate.
- Calculate slope of line using formula slope=(y2-y1)/(x2-x1).
- If line is to be horizontally clipped.
- New y = ymin or ymax.
- New x = x1 + ((new y - y1)/slope).
- If line is to be vertically clipped.
- New x = xmin or xmax.
- New y = y1+slope*(new x –x1).
- Clip the lines from these intersection points.
- Display the new line.
- Close the graphic system. Stop.

- # Sample Code

A C++ program to draw a line using Bresenham's Line Algorithm (BLA) for lines with slopes negative and less than 1.

```
# include <iostream.h>
# include <graphics.h>
# include <conio.h>
# include <math.h>
void show_screen( );
void bresenham_line(const int,const int,const int,const int);
int main( )
  {
    int driver=VGA;
    int mode=VGAHI;
    int x_1=0;
    int y_1=0;
    int x_2=0;
    int y_2=0;
    do
       {
         show_screen( );
         gotoxy(8,10);
         cout<<"Coordinates of Point-I (x1,y1) :";
         gotoxy(8,11);
         cout<<"ÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ";
         gotoxy(12,13);
         cout<<"Enter the value of x1 = ";
         cin>>x_1;
         gotoxy(12,14);
         cout<<"Enter the value of y1 = ";
         cin>>y_1;
         gotoxy(8,18);
         cout<<"Coordinates of Point-II (x2,y2) :";
         gotoxy(8,19);
         cout<<"ÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ";
         gotoxy(12,21);
         cout<<"Enter the value of x2 = ";
         cin>>x_2;
         gotoxy(12,22);
         cout<<"Enter the value of y2 = ";
         cin>>y_2;
         initgraph(&driver,&mode,"..\\Bgi");
         setcolor(15);
         bresenham_line(x_1,y_1,x_2,y_2);
         setcolor(15);
```

```
            outtextxy(110,460,"Press <Enter> to continue or any other key to exit.");
            int key=int(getch( ));
            if(key!=13)
                break;
          }
    while(1);
    return 0;
  }
void bresenham_line(const int x_1,const int y_1,const int x_2,const int y_2)
  {
    int color=getcolor( );
    int x1=x_1;
    int  y1=y_1;
    int  x2=x_2;
    int  y2=y_2;
    if(x_1>x_2)
          {
            x1=x_2;
            y1=y_2;
            x2=x_1;
            y2=y_1;
          }
    int  dx=abs(x2-x1);
    int  dy=abs(y2-y1);
    int two_dy=(2*dy);
    int two_dy_dx=(2*(dy-dx));
    int p=((2*dy)-dx);
    int x=x1;
    int y=y1;
    putpixel(x,y,color);
   while(x<x2)
          {
            x++;
            if(p<0)
                p+=two_dy;
            else
                {
                  y++;
                  p+=two_dy_dx;
                }
            putpixel(x,y,color);
          }
  }
void show_screen( )
  {
    restorecrtmode( );
    textmode(C4350);
```

```
cprintf("\n*************************************************************");
   cprintf("***************************-                 -************************");
   cprintf("*..............................");
   textbackground(1);
   cprintf(" Bresenham's Line Algorithm ");
   textbackground(8);
   cprintf("---------------------- *");
   cprintf("*-************************- -**********************-*");
   cprintf("*-*************************************************************-*");
   for(int count=0;count<42;count++)
       cprintf("*-*                                             *-*");
   gotoxy(1,46);
   cprintf("*- ************************************************************-*");
   cprintf("*.........................................................................................*");
   cprintf("***************************************************");
   gotoxy(8,40);
   cout<<"Note :";
   gotoxy(8,41);
   cout<<"ÍÍÍÍÍÍ";
   gotoxy(12,43);
   cout<<"The slope of the line should be negative and less than 1.";
   gotoxy(1,2);
 }
```

A C++ program to draw a line using Bresenham's Line Algorithm (BLA) for lines with slopes negative and greater than 1.

```
# include <iostream.h>
# include <graphics.h>
# include    <conio.h>
# include     <math.h>
void show_screen( );
void bresenham_line(const int,const int,const int,const int);
int main( )
  {
   int driver=VGA;
   int mode=VGAHI;
   int x_1=0;
   int y_1=0;
   int x_2=0;
   int y_2=0;
   do
       {
        show_screen( );
        gotoxy(8,10);
        cout<<"Coordinates of Point-I (x1,y1) :";
        gotoxy(8,11);
        cout<<"ÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ";
```

```
            gotoxy(12,13);
            cout<<"Enter the value of x1 = ";
            cin>>x_1;
            gotoxy(12,14);
            cout<<"Enter the value of y1 = ";
            cin>>y_1;
            gotoxy(8,18);
            cout<<"Coordinates of Point-II (x2,y2) :";
            gotoxy(8,19);
            cout<<"ÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ";
            gotoxy(12,21);
            cout<<"Enter the value of x2 = ";
            cin>>x_2;
            gotoxy(12,22);
            cout<<"Enter the value of y2 = ";
            cin>>y_2;
            initgraph(&driver,&mode,"..\\Bgi");
            setcolor(15);
            bresenham_line(x_1,y_1,x_2,y_2);
            setcolor(15);
              outtextxy(110,460,"Press <Enter> to continue or any other key to exit.");
            int key=int(getch( ));
   if(key!=13)
                  break;
          }
     while(1);

     return 0;
  }
void bresenham_line(const int x_1,const int y_1,const int x_2,const int y_2)
  {
     int color=getcolor( );
     int x1=x_1;
     int  y1=y_1;
     int  x2=x_2;
     int  y2=y_2;
     if(x_1>x_2)
          {
            x1=x_2;
            y1=y_2;
            x2=x_1;
            y2=y_1;
          }
     int  dx=abs(x2-x1);
     int  dy=abs(y2-y1);
     int two_dx=(2*dx);
     int two_dx_dy=(2*(dx-dy));
```

```
      int p=((2*dx)-dy);
      int x=x1;
      int y=y1;
      putpixel(x,y,color);
      while(y<y2)
            {
              y++;
              if(p<0)
                    p+=two_dx;
              else
                    {
                      x++;
                      p+=two_dx_dy;
                    }
              putpixel(x,y,color);
            }
   }
 void show_screen( )
   {
     restorecrtmode( );
    textmode(C4350);
cprintf("\n*********************************************************************************
**********");
      cprintf("**************************-                    -************************");
      cprintf("*                                ");
      textbackground(1);
      cprintf(" Bresenham's Line Algorithm ");
      textbackground(8);
      cprintf("---------------------- *");
      cprintf("*-************************-                    -**********************-*");
      cprintf("*-*****************************************************************-*");
      for(int count=0;count<42;count++)
            cprintf("*-*                                                    *-*");
      gotoxy(1,46);
      cprintf("*-*****************************************************************-*");
      cprintf("*                                                                                *");
  cprintf("*******************************************************************");
      gotoxy(8,40);
      cout<<"Note :";
      gotoxy(8,41);
      cout<<"ÍÍÍÍÍÍ";
      gotoxy(12,43);
      cout<<"The slope of the line should be negative and greater than 1.";
      gotoxy(1,2);
   }
```

A C++ program to draw a line using Digital Differential Analyzer (DDA) Algorithm.

```
# include <iostream.h>
# include <graphics.h>
# include    <conio.h>
# include     <math.h>
void show_screen( );
void dda_line(const int,const int,const int,const int);
int main( )
  {
    int driver=VGA;
    int mode=VGAHI;
    int x_1=0;
    int y_1=0;
    int x_2=0;
    int y_2=0;
    do
        {
          show_screen( );
          gotoxy(8,10);
          cout<<"Coordinates of Point-I (x1,y1) :";
          gotoxy(8,11);
          cout<<"ÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ";
          gotoxy(12,13);
          cout<<"Enter the value of x1 = ";
          cin>>x_1;
          gotoxy(12,14);
          cout<<"Enter the value of y1 = ";
          cin>>y_1;
          gotoxy(8,18);
          cout<<"Coordinates of Point-II (x2,y2) :";
          gotoxy(8,19);
          cout<<"ÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ";
          gotoxy(12,21);
          cout<<"Enter the value of x2 = ";
          cin>>x_2;
          gotoxy(12,22);
          cout<<"Enter the value of y2 = ";
          cin>>y_2;
          initgraph(&driver,&mode,"..\\Bgi");
          setcolor(15);
          dda_line(x_1,y_1,x_2,y_2);
          setcolor(15);
            outtextxy(110,460,"Press <Enter> to continue or any other key to exit.");
          int key=int(getch( ));

          if(key!=13)
               break;
```

```
            }
       while(1);
        return 0;
    }
 void dda_line(const int x_1,const int y_1,const int x_2,const int y_2)
    {
       int color=getcolor( );
      int x1=x_1;
       int  y1=y_1;
       int  x2=x_2;
       int  y2=y_2;
       if(x_1>x_2)
             {
               x1=x_2;
               y1=y_2;
               x2=x_1;
               y2=y_1;
             }
       float dx=(x2-x1);
       float dy=(y2-y1);
       int steps=abs(dy);
       if(abs(dx)>abs(dy))
            steps=abs(dx);
       float x_inc=(dx/(float)steps);
       float y_inc=(dy/(float)steps);
       float x=x1;
       float y=y1;
      putpixel(x,y,color);
       for(int count=1;count<=steps;count++)
             {
               x+=x_inc;
               y+=y_inc;
               putpixel((int)(x+0.5),(int)(y+0.5),color);
             }
    }
 void show_screen( )
    {
       restorecrtmode( );
       textmode(C4350);
cprintf("\n*************************************************************************");
       cprintf("******************-                                -*****************");
       cprintf("*                                                                    ");
       textbackground(1);
       cprintf(" Digital Differential Analyzer Algorithm ");
       textbackground(8);
       cprintf("                                                                   *");
       cprintf("*-*****************-                                -***************-*");
```

```
    cprintf("*-************************************************************-*");
    for(int count=0;count<42;count++)
        cprintf("*-*                                                        *-*");
    gotoxy(1,46);
    cprintf("*-************************************************************-*");
    cprintf("*                                                              *");
cprintf("**************************************************************");
    gotoxy(1,2);
  }
```

A C++ program to draw a line using Bresenham's Line Algorithm (BLA) for lines with slopes positive and less than 1.

```
# include <iostream.h>
# include <graphics.h>
# include <conio.h>
# include <math.h>
void show_screen( );
void bresenham_line(const int,const int,const int,const int);
int main( )
  {
    int driver=VGA;
    int mode=VGAHI;\
    int x_1=0;
    int y_1=0;
    int x_2=0;
    int y_2=0;
    do
       {
         show_screen( );
         gotoxy(8,10);
         cout<<"Coordinates of Point-I (x1,y1) :";
         gotoxy(8,11);
         cout<<"ÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ";
         gotoxy(12,13);
         cout<<"Enter the value of x1 = ";
         cin>>x_1;
         gotoxy(12,14);
         cout<<"Enter the value of y1 = ";
         cin>>y_1;
         gotoxy(8,18);
         cout<<"Coordinates of Point-II (x2,y2) :";
         gotoxy(8,19);
         cout<<"ÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ";
         gotoxy(12,21);
         cout<<"Enter the value of x2 = ";
         cin>>x_2;
```

```
            gotoxy(12,22);
            cout<<"Enter the value of y2 = ";
            cin>>y_2;
            initgraph(&driver,&mode,"..\\Bgi");
            setcolor(15);
            bresenham_line(x_1,y_1,x_2,y_2);
            setcolor(15);
              outtextxy(110,460,"Press <Enter> to continue or any other key to exit.");
            int key=int(getch( ));
            if(key!=13)
                  break;
          }
     while(1);
     return 0;
   }
void bresenham_line(const int x_1,const int y_1,const int x_2,const int y_2)
   {
     int color=getcolor( );
     int x1=x_1;
     int  y1=y_1;
     int  x2=x_2;
     int  y2=y_2;
     if(x_1>x_2)
          {
            x1=x_2;
            y1=y_2;
            x2=x_1;
            y2=y_1;
          }
     int  dx=abs(x2-x1);
     int  dy=abs(y2-y1);
     int two_dy=(2*dy);
     int two_dy_dx=(2*(dy-dx));
     int p=((2*dy)-dx);
     int x=x1;
     int y=y1;
     putpixel(x,y,color);
     while(x<x2)
          {
            x++;
            if(p<0)
                  p+=two_dy;
            else
                  {
                    y--;
                    p+=two_dy_dx;
                  }
```

```
            putpixel(x,y,color);
          }
   }

 void show_screen( )
   {
     restorecrtmode( );
     textmode(C4350);
cprintf("\n*****************************************************************");
     cprintf("*************************-                    -************************");
     cprintf("*                                                                  ");
     textbackground(1);
     cprintf(" Bresenham's Line Algorithm ");
     textbackground(8);
     cprintf("---------------------- *");
     cprintf("*-***********************-                    -*********************-*");
     cprintf("*-*****************************************************************-*");

     for(int count=0;count<42;count++)
          cprintf("*-*                                                    *-*");
     gotoxy(1,46);
     cprintf("*-*****************************************************************-*");
     cprintf("*                                                                  *");
cprintf("*****************************************************************");
     gotoxy(8,40);
     cout<<"Note :";
     gotoxy(8,41);
     cout<<"ÍÍÍÍÍÍ";
     gotoxy(12,43);
     cout<<"The slope of the line should be positive and less than 1.";
     gotoxy(1,2);
   }
```

A C++ program to draw a circle using Bresenham's Circle Algorithm.

```
 # include <iostream.h>
 # include <graphics.h>
 # include    <conio.h>
 # include     <math.h>
 void show_screen( );
 void bresenham_circle(const int,const int,const int);
int main( )
   {
     int driver=VGA;
     int mode=VGAHI;
     int h=0;
     int k=0;
     int r=0;
```

```
   do
     {
      show_screen( );
      gotoxy(8,10);
      cout<<"Central Point of the Circle : (h,k) :";
      gotoxy(8,11);
      cout<<"ÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ";
      gotoxy(12,13);
      cout<<"Enter the value of h = ";
      cin>>h;
      gotoxy(12,14);
      cout<<"Enter the value of k = ";
      cin>>k;
      gotoxy(8,18);
      cout<<"Radius of the Circle : r :";
      gotoxy(8,19);
      cout<<"ÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ";
      gotoxy(12,21);
      cout<<"Enter the value of r = ";
      cin>>r;
      initgraph(&driver,&mode,"..\\Bgi");
      setcolor(15);
      bresenham_circle(h,k,r);
      setcolor(15);
        outtextxy(110,460,"Press <Enter> to continue or any other key to exit.");
      int key=int(getch( ));
      if(key!=13)
          break;
     }
   while(1);
   return 0;
  }
void bresenham_circle(const int h,const int k,const int r)
  {
   int color=getcolor( );
   int x=0;
   int y=r;
   int p=(3-(2*r));
   do
     {
      putpixel((h+x),(k+y),color);
      putpixel((h+y),(k+x),color);
      putpixel((h+y),(k-x),color);
      putpixel((h+x),(k-y),color);
      putpixel((h-x),(k-y),color);
      putpixel((h-y),(k-x),color);
      putpixel((h-y),(k+x),color);
```

```
            putpixel((h-x),(k+y),color);
            x++;
            if(p<0)
                  p+=((4*x)+6);
            else
                  {
                    y--;
                    p+=((4*(x-y))+10);
                  }
          }
      while(x<=y);
   }
 void show_screen( )
   {
     restorecrtmode( );
     textmode(C4350);
    cprintf("\n*************************************************************************");
     cprintf("************************-                          -************************");
     cprintf("*                                                                        ");
     textbackground(1);
     cprintf(" Bresenham's Circle Algorithm ");
     textbackground(8);
     cprintf("---------------------- *");
     cprintf("************************-                          -************************");
     cprintf("*-*********************************************************************-*");
     for(int count=0;count<42;count++)
          cprintf("*-*                                                          *-*");
     gotoxy(1,46);       gotoxy(1,2);    }
```

A C++ program to draw a circle using MidPoint Circle Algorithm.

```
# include <iostream.h>
# include <graphics.h>
# include    <conio.h>
# include     <math.h>
void show_screen( );
 void midpoint_circle(const int,const int,const int);
int main( )
   {
     int driver=VGA;
     int mode=VGAHI;
     int h=0;
     int k=0;
     int  r=0;
     do
          {
            show_screen( );
            gotoxy(8,10);
            cout<<"Central Point of the Circle : (h,k) :";
```

```
            gotoxy(8,11);
            cout<<"ÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ";
            gotoxy(12,13);
            cout<<"Enter the value of h = ";
            cin>>h;
            gotoxy(12,14);
            cout<<"Enter the value of k = ";
            cin>>k;
            gotoxy(8,18);
            cout<<"Radius of the Circle : r :";
            gotoxy(8,19);
            cout<<"ÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ";
            gotoxy(12,21);
            cout<<"Enter the value of r = ";
            cin>>r;
            initgraph(&driver,&mode,"..\\Bgi");
            setcolor(15);
              midpoint_circle(h,k,r);
            setcolor(15);
              outtextxy(110,460,"Press <Enter> to continue or any other key to exit.");
            int key=int(getch( ));
            if(key!=13)
                  break;
          }
      while(1);
      return 0;
    }
void midpoint_circle(const int h,const int k,const int r)
   {
      int color=getcolor( );
      int x=0;
      int y=r;
      int p=(1-r);
      do
          {
            putpixel((h+x),(k+y),color);
            putpixel((h+y),(k+x),color);
            putpixel((h+y),(k-x),color);
            putpixel((h+x),(k-y),color);
            putpixel((h-x),(k-y),color);
            putpixel((h-y),(k-x),color);
            putpixel((h-y),(k+x),color);
            putpixel((h-x),(k+y),color);
            x++;
            if(p<0)
                  p+=((2*x)+1);
            else
```

```
                {
                  y--;
                  p+=((2*(x-y))+1);
                }
            }
      while(x<=y);
   }
void show_screen( )
   {
      restorecrtmode( );
      textmode(C4350);
      cprintf("\n*************************************************************************");
      cprintf("*-************************-                    -************************-*");
      cprintf("*                                                                        ");
      textbackground(1);
      cprintf(" MidPoint Circle Algorithm ");
      textbackground(8);
      cprintf("------------------------ *");
      cprintf("*-*********************************************************************-*");
      for(int count=0;count<42;count++)
            cprintf("*-*                                                        *-*");
      gotoxy(1,46);
cprintf("*************************************************************************");
      gotoxy(1,2);    }
```

A C++ program to fill different types of geometric shapes using Boundary

```
# include <iostream.h>
# include <graphics.h>
# include    <conio.h>
# include     <math.h>
void show_screen( );
 void BoundaryFill(const int,const int,const int,const int);
 void Circle(const int,const int,const int);
 void Triangle(const int,const int,const int,const int,const int,const int);
 void Rectangle(const int,const int,const int,const int);
 void Polygon(const int,const int []);
 void Line(const int,const int,const int,const int);
int main( )
   {
      int driver=VGA;
      int mode=VGAHI;
      initgraph(&driver,&mode,"..\\Bgi");
      show_screen( );
      setcolor(15);
          Circle(175,175,50);
          BoundaryFill(175,175,10,15);
      setcolor(15);
      settextstyle(0,0,1);
```

```
        outtextxy(150,235,"Circle");
    setcolor(15);
        Rectangle(350,135,500,225);
        BoundaryFill(425,175,9,15);
    setcolor(15);
    settextstyle(0,0,1);
        outtextxy(390,235,"Rectangle");
    setcolor(15);
        Triangle(125,370,225,370,175,280);
        BoundaryFill(175,325,8,15);
    setcolor(15);
    settextstyle(0,0,1);
        outtextxy(145,380,"Triangle");
    int polygon_points[14]={ 340,330, 390,285, 460,285, 510,330,
                                                460,375, 390,375, 340,330 };
    setcolor(15);
        Polygon(7,polygon_points);
        BoundaryFill(425,325,12,15);
    setcolor(15);
    settextstyle(0,0,1);
        outtextxy(397,380,"Polygon");
    getch( );
    return 0;
  }
struct Pixel
  {
    int x;
    int y;
    Pixel *Next;
  };
 Pixel *Entry=NULL;
 Pixel *Start=NULL;
 Pixel *Last=NULL;
 void insert_pixel(const int x,const int y)
  {
    Entry=new Pixel;
    Entry->x=x;
    Entry->y=y;
    Entry->Next=NULL;
    Last->Next=Entry;
    Last=Entry;
  }
 void BoundaryFill(const int _x,const int _y,
                              const int fill_color,const int boundary_color)
  {
    if(getpixel(_x,_y)==boundary_color || getpixel(_x,_y)==fill_color)
         return;
```

```
    int x=_x;
    int y=_y;
    Start=new Pixel;
    Start->x=x;
    Start->y=y;
    Start->Next=NULL;
    Last=Start;
    while(Start!=NULL)
        {
          putpixel(x,y,fill_color);
          if(getpixel((x-1),y)!=boundary_color &&
                                getpixel((x-1),y)!=fill_color&& (x-1)>=0)
              {
                putpixel((x-1),y,fill_color);
                insert_pixel((x-1),y);
              }
          if(getpixel(x,(y-1))!=boundary_color &&
                                getpixel(x,(y-1))!=fill_color&& (y-1)>=0)
              {
                putpixel(x,(y-1),fill_color);
                insert_pixel(x,(y-1));
              }
          if(getpixel((x+1),y)!=boundary_color &&
                        getpixel((x+1),y)!=fill_color&& (x+1)<=getmaxx( ))
              {
                putpixel((x+1),y,fill_color);
                insert_pixel((x+1),y);
              }
          if(getpixel(x,(y+1))!=boundary_color &&
                        getpixel(x,(y+1))!=fill_color&& (y-1)<=getmaxy( ))
              {
                putpixel(x,(y+1),fill_color);
                insert_pixel(x,(y+1));
              }
          Entry=Start;
          Start=Start->Next;
          x=Start->x;
          y=Start->y;
          delete Entry;
        }
   }
void Circle(const int h,const int k,const int r)
   {
    int color=getcolor( );
    int x=0;
    int y=r;
    int p=(1-r);
```

```
    do
        {
          putpixel((h+x),(k+y),color);
          putpixel((h+y),(k+x),color);
          putpixel((h+y),(k-x),color);
          putpixel((h+x),(k-y),color);
          putpixel((h-x),(k-y),color);
          putpixel((h-y),(k-x),color);
          putpixel((h-y),(k+x),color);
          putpixel((h-x),(k+y),color);
          x++;
          if(p<0)
              p+=((2*x)+1);
          else
              {
                y--;
                p+=((2*(x-y))+1);
              }
        }
    while(x<=y);
  }
void Triangle(const int x_1,const int y_1,const int x_2,const int y_2,
                                                const int x_3,const int y_3)
  {
    Line(x_1,y_1,x_2,y_2);
    Line(x_2,y_2,x_3,y_3);
    Line(x_3,y_3,x_1,y_1);
  }
void Rectangle(const int x_1,const int y_1,const int x_2,const int y_2)
  {
    Line(x_1,y_1,x_2,y_1);
    Line(x_2,y_1,x_2,y_2);
    Line(x_2,y_2,x_1,y_2);
    Line(x_1,y_2,x_1,y_1);
  }
void Polygon(const int n,const int coordinates[])
  {
    if(n>=2)
        {
          Line(coordinates[0],coordinates[1],
                                    coordinates[2],coordinates[3]);
          for(int count=1;count<(n-1);count++)
              Line(coordinates[(count*2)],coordinates[((count*2)+1)],
                                    coordinates[((count+1)*2)],
                                    coordinates[(((count+1)*2)+1)]);
        }
  }
```

```
void Line(const int x_1,const int y_1,const int x_2,const int y_2)
   {
     int color=getcolor( );
     int x1=x_1;
     int  y1=y_1;
     int  x2=x_2;
     int  y2=y_2;
     if(x_1>x_2)
          {
            x1=x_2;
            y1=y_2;

            x2=x_1;
            y2=y_1;
          }
     int dx=abs(x2-x1);
     int dy=abs(y2-y1);
     int inc_dec=((y2>=y1)?1:-1);
     if(dx>dy)
          {
            int two_dy=(2*dy);
            int two_dy_dx=(2*(dy-dx));
            int p=((2*dy)-dx);
            int x=x1;
            int y=y1;
            putpixel(x,y,color);
            while(x<x2)
                 {
                   x++;
                   if(p<0)
                     p+=two_dy;
                   else
                     {
                          y+=inc_dec;
                          p+=two_dy_dx;
                     }
                   putpixel(x,y,color);
                 }
          }
     else
          {
            int two_dx=(2*dx);
            int two_dx_dy=(2*(dx-dy));
            int p=((2*dx)-dy);
            int x=x1;
            int y=y1;
            putpixel(x,y,color);
```

```
            while(y!=y2)
                {
                  y+=inc_dec;
                  if(p<0)
                    p+=two_dx;
                  else
                    {
                        x++;
                        p+=two_dx_dy;
                    }

                  putpixel(x,y,color);
                }
          }
   }
void show_screen( )
   {
     setfillstyle(1,1);
          bar(220,26,420,38);
     settextstyle(0,0,1);
          setcolor(15);
           outtextxy(5,5,"*************************************************************");
            outtextxy(5,17,"*-*************************************************-*");
            outtextxy(5,29,"*                                                   *");
            outtextxy(5,41,"*-*************************************************-*");
          outtextxy(5,53,"*-*************************************************-*");
          setcolor(11);
            outtextxy(228,29,"Boundary Fill Algorithm");
          setcolor(15);
            for(int count=0;count<=30;count++)
              outtextxy(5,(65+(count*12)),"*-*                                                 *-*");
            outtextxy(5,438,"*-*************************************************-*");
            outtextxy(5,450,"*                                                   *");
           outtextxy(5,462,"*************************************************");
          setcolor(12);
            outtextxy(229,450,"Press any Key to exit.");
   }
```

PROGRAM TO ROTATE A TRIANGLE ABOUT ORIGIN.

```
#include<iostream.h>
#include<conio.h>
#include<graphics.h>
#include<process.h>
#include<math.h>
void main()
{
```

```
clrscr();
int graphdriver=DETECT,graphmode;
initgraph(&graphdriver,&graphmode,"...\\bgi");

int x,y,x1,a[3][3];
double b[3][3],c[3][3];
printf"\n          Enter Ist coordinates of triangle:";
scanf(–%d%d|,&a[0][0],&a[1][0]);

printf"\n          Enter 2nd coordinates of triangle:";
scanf(–%d%d|,&a[0][1],&a[1][1]);

printf"\n          Enter 3rd coordinates of triangle:";
scanf(–%d%d|,&a[0][2],&a[1][2]);

line(a[0][0],a[1][0],a[0][1],a[1][1]);
line(a[0][1],a[1][1],a[0][2],a[1][2]);
line(a[0][0],a[1][0],a[0][2],a[1][2]);
getch();
cleardevice();
printf"\n Enter angle of rotation:\n";
scanf(–%d|,&x);

b[0][0]=b[1][1]=cos((x*3.14)/180);
b[0][1]=-sin((x*3.14)/180);
b[1][0]=sin((x*3.14)/180);
b[2][2]=1;
b[2][0]=b[2][1]=b[0][2]=b[1][2]= 0;
for(int i=0;i<3;i++)
{
for(int j=0;j<3;j++)
{                c[i][j]=0;
for (int k=0; k<3;k++)
{
c[i][j]+=a[i][k]*b[k][j];
}
x1=(c[i][j]+0.5);
a[i][j]=x1;
}
  }
printf"\n Triangle after rotation is:\n" ;

line(a[0][0],a[1][0],a[0][1],a[1][1]);
line(a[0][1],a[1][1],a[0][2],a[1][2]);
line(a[0][0],a[1][0],a[0][2],a[1][2]);

getch();
```

```
closegraph();
}
```

PROGRAM TO SCALE THE TRIANGLE

```
#include<iostream.h>
#include<conio.h>
#include<graphics.h>O
void main()
{
int gd=DETECT,gm;
initgraph(&gd, &gm,"");
cleardevice();
int x1,y1,x2,y2,x3,y3,x4,y4;
float sx,sy;
printf"Enter the first coordinates of triangle\n";
scanf(–%d%d|,&x1,&y1);
printf"Enter the second coordinates of triangle\n";
scanf(–%d%d|,&x2,&y2);
printf"Enter the third coordinates of triangle\n";
scanf(–%d%d|,&x3,&y3);
int poly[8]={x1,y1,x2,y2,x3,y3,x1,y1};
cleardevice();
drawpoly(4,poly);
getch();
printf"Enter the scaling factors\n";
scanf(–%d%d|,&sx,&sy);
x4=sx*x1-x1;
y4=sy*y1-y1;

x1=sx*x1-x4;
y1=sy*y1-y4;
x2=sx*x2-x4;
y2=sy*y2-y4;
x3=sx*x3-x4;
y3=sy*y3-y4;
poly[0]=x1;
poly[1]=y1;
poly[2]=x2;
poly[3]=y2;
poly[4]=x3;
poly[5]=y3;
poly[6]=x1;
poly[7]=y1;
getch();
cleardevice();
drawpoly(4,poly);
getch();
closegraph();
}
```

PROGRAM TO TRANSLATE A TRIANGLE

```
#include<iostream.h>
#include<conio.h>
#include<graphics.h>
#include<process.h>
#include<math.h>

void main()
{
clrscr();
int graphdriver=DETECT,graphmode;
initgraph(&graphdriver,&graphmode,"...\\bgi");

int x,y,x1,y1,x2,y2,x3,y3;
printf"\n        Enter Ist coordinates of triangle:";
scanf(–%d%d|,&x1,&y1);

printf"\n        Enter 2nd coordinates of triangle:";
scanf(–%d%d|,&x2,&y2);

printf"\n        Enter 3rd coordinates of triangle:";
scanf(–%d%d|,&x3,&y3);

cleardevice();
line(x1,y1,x2,y2);
line(x2,y2,x3,y3);
line(x1,y1,x3,y3);
getch();
cleardevice();

printf"\n Enter translatio factors :\n";
scanf(–%d%d|,&x,&y);
x1-=x;
y1-=y;
x2-=x;
y2-=y;
x3-=x;
y3-=y;

cleardevice();
line(x1,y1,x2,y2);
line(x2,y2,x3,y3);
line(x1,y1,x3,y3);
getch();
closegraph();
}
```

PROGRAM TO ROTATE A POINT ABOUT A POINT

```
#include<iostream.h>
#include<conio.h>
#include<graphics.h>
#include<math.h>
#include<dos.h>
void main()
{
clrscr();
int gm,gd=DETECT;
initgraph(&gd,&gm,"");
int h,k,x1,y1,x2,y2,x3,y3;
float t;
printf" OUTPUT");
printf"Enter the coordinates of point");
scanf(–%d%d|,&x2,&y2);
putpixel(x2,y2,2);

printf"Enter the coordinates of point around which rotation is done");
scanf(–%d%d|,&h,&k);
putpixel(h,k,2);

printf"Enter the angle for rotation");
scanf(–%d|,&t);
cleardevice();
x1=(h*cos(t))-(k*sin(t));
y1=(h*sin(t))+(k*cos(t));
x3=x1+x2-h;
y3=y1+y2-k;

printf"Point after rotation is:";
putpixel(x3,y3,2);

getch();
closegraph();
}
```

PROGRAM TO ROTATE A POINT ABOUT ORIGIN

```
#include<iostream.h>
#include<conio.h>
#include<graphics.h>
#include<math.h>
#include<dos.h>
void main()
{
clrscr();
int gm,gd=DETECT;
initgraph(&gd,&gm,"");
int h,k,x1,y1,x2,y2,x3,y3;
float t;
printf" OUTPUT");
printf"Enter the coordinates of point");
scanf(–%d%d|,&x2,&y2);
putpixel(x2,y2,2);

printf"Enter the angle for rotation");
scanf(–%d|,&t);
cleardevice();
x1=int(x2*cos(t*3.14/180))-(y2*sin(t*3.14/180));
y1=int(x2*sin(t*3.14/180))+(y2*cos(t*3.14/180));
printf"Point after rotation is:";
putpixel(x1,y1,2);

getch();
closegraph();
```

PROGRAM TO REFLECT A TRIANGLE

```
#include<iostream.h>
#include<conio.h>
#include<graphics.h>
#include<process.h>
#include<math.h>
void main()
{
clrscr();
int graphdriver=DETECT,graphmode;
initgraph(&graphdriver,&graphmode,"...\\bgi");

int x,y,x1,a[3][3];
double b[3][3],c[3][3];
printf"\n          Enter Ist coordinates of triangle:";
scanf(–%d%d|,&a[0][0],&a[1][0]);
```

```
printf"\n          Enter 2nd coordinates of triangle:";
scanf(–%d%d|,&a[0][1],&a[1][1]);

printf"\n          Enter 3rd coordinates of triangle:";
scanf(–%d%d|,&a[0][2],&a[1][2]);

printf"\n Enter 1. for reflection in x-axis:\n";
printf"\n Enter 2. for reflection in y-axis:\n";
printf"\n Enter 3. for reflection in both the axis:\n";
scanf(–%d|,&x);
cleardevice();
line(320,0,320,479);
line(0,240,639,240);

line(a[0][0],a[1][0],a[0][1],a[1][1]);
line(a[0][1],a[1][1],a[0][2],a[1][2]);
line(a[0][0],a[1][0],a[0][2],a[1][2]);
switch(x)
{
case 1:b[0][0]=640-a[0][0];
      b[0][1]=640-a[0][1];
      b[0][2]=640-a[0][2];
      b[1][0]=a[1][0];
      b[1][1]=a[1][1];
      b[1][2]=a[1][2];
                line(320,0,320,479);
                line(0,240,639,240);
                line(b[0][0],b[1][0],b[0][1],b[1][1]);
                line(b[0][1],b[1][1],b[0][2],b[1][2]);
                line(b[0][0],b[1][0],b[0][2],b[1][2]);
                getch();
                break;
case 2:b[1][0]=480-a[1][0];
      b[1][1]=480-a[1][1];
      b[1][2]=480-a[1][2];
      b[0][0]=a[0][0];
      b[0][1]=a[0][1];
      b[0][2]=a[0][2];
                line(320,0,320,479);
                line(0,240,639,240);
                line(b[0][0],b[1][0],b[0][1],b[1][1]);
                line(b[0][1],b[1][1],b[0][2],b[1][2]);
                line(b[0][0],b[1][0],b[0][2],b[1][2]);
                getch();
                break;
```

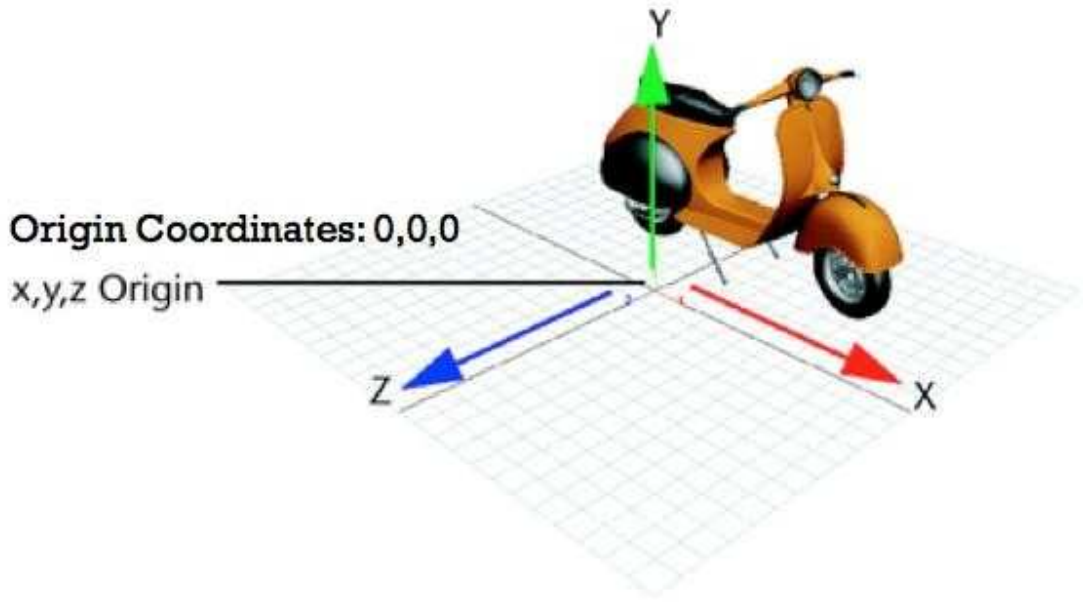
```
case 3: b[0][0]=640-a[0][0];
      b[0][1]=640-a[0][1];
      b[0][2]=640-a[0][2];
      b[1][0]=a[1][0];
      b[1][1]=a[1][1];
      b[1][2]=a[1][2];
                  line(320,0,320,479);
                  line(0,240,639,240);
                  line(b[0][0],b[1][0],b[0][1],b[1][1]);
                  line(b[0][1],b[1][1],b[0][2],b[1][2]);
                  line(b[0][0],b[1][0],b[0][2],b[1][2]);
                  b[1][0]=480-a[1][0];
      b[1][1]=480-a[1][1];
      b[1][2]=480-a[1][2];
      b[0][0]=a[0][0];
      b[0][1]=a[0][1];
      b[0][2]=a[0][2];
                  line(320,0,320,479);
                  line(0,240,639,240);
                  line(b[0][0],b[1][0],b[0][1],b[1][1]);
                  line(b[0][1],b[1][1],b[0][2],b[1][2]);
                  line(b[0][0],b[1][0],b[0][2],b[1][2]);
                  getch();
                  break;
}
getch();
closegraph();

}
```

MULTIMEDIA

Moving from 2D to 3D design can be challenging since most of us design using only two dimensions. However, you‘ve done this before but you just don‘t remember! Maybe you haven‘t realized that building objects in Maya 3D is somehow similar to building objects using Legos. The key to design in Maya 3D is to think spatially. In Maya, we design not only acknowledging width and height we also incorporate depth!
It takes some practice and lots of patience to develop 3D spatial ability, so below you‘ll find some basic Maya 101 concepts that will help you better understand the 3D environment.

---

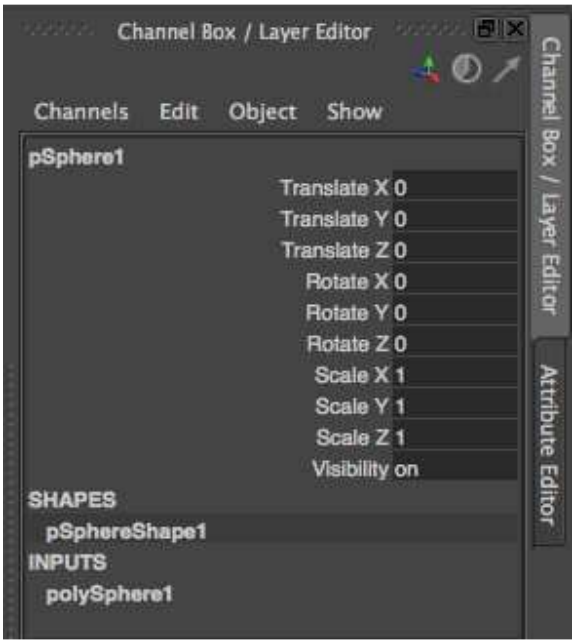
1. X,Y and Z



In Maya the X axis is the width (red), the y axis is the height (green) and the Z axis is the depth (blue). The y-axis is also referred to as Y-up. The center of the coordinate system is called the origin and the coordinates are 0, 0, 0 for x, y, z.

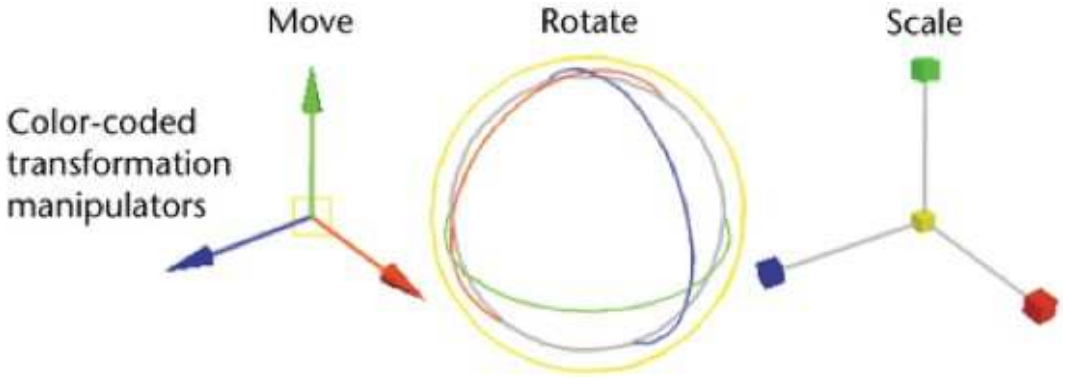
If you want to manipulate an object in Maya you can:

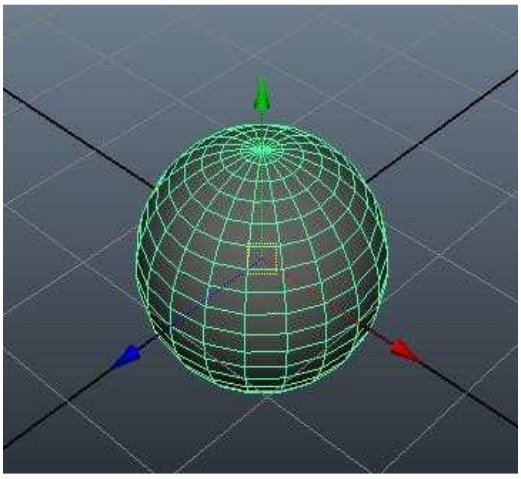
a) Enter x, y and z values for translate, rotate and scale in –Channel Box|



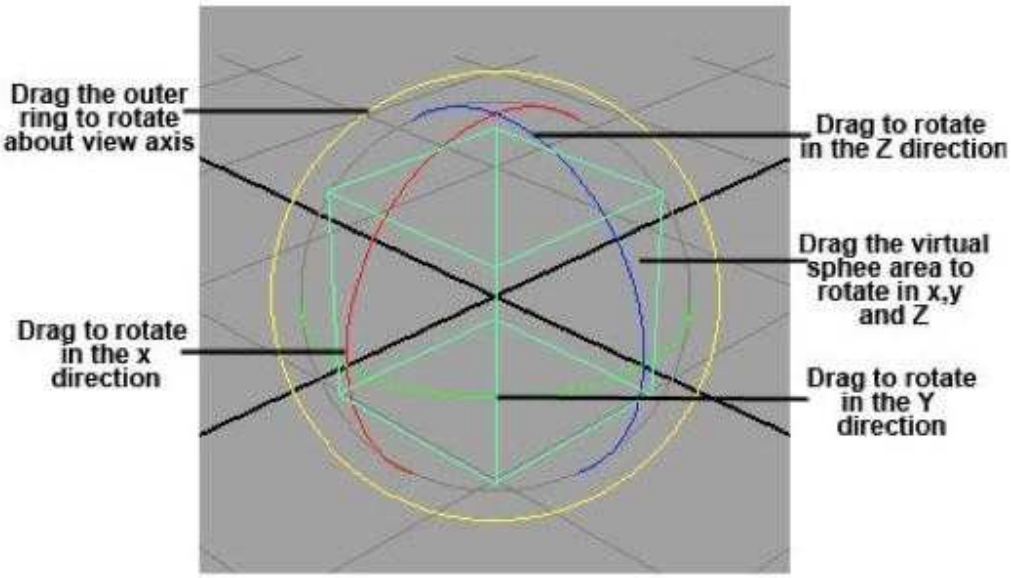
- Translate: Value –0‖ in x, y and z, position the object on the center or –origin‖ of the workspace.
- Rotate: If you by mistake rotate an object, set all the values to –0‖. This will take the object back to its default position.
- Scale: —1‖ is the default value and it means the object‘s size is 100%. If you want to make the object twice its size enter 2 in all the scale values.If you want to make an object half its size enter 0.5 in all the scale values

b) Use the manipulators handles of the move tool, scale tool and rotate tool.

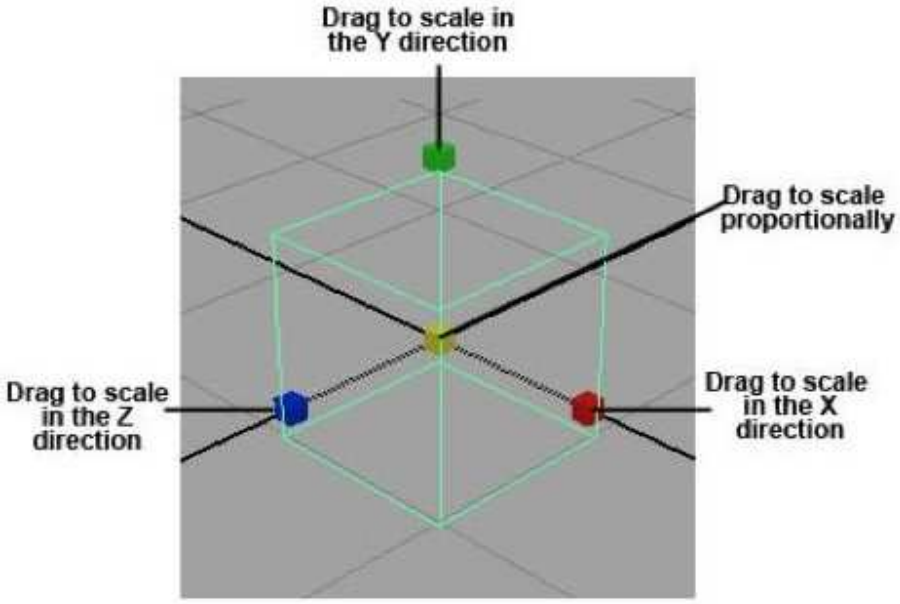




When moving & rotating objects in Maya, it‘s recommended that you move or rotate the object only along one axis, this can help you maintain your objects aligned.
You can do this by pulling the arrows of the move tool or by dragging the rings of the rotate tool. Moving the objects from the center handle will allow you to move the object freely across the view plane and it can be hard to control its translation and rotation.



When scaling object is Maya, if you want to maintain the object‘s proportions drag the center box to scale uniformly in all directions.

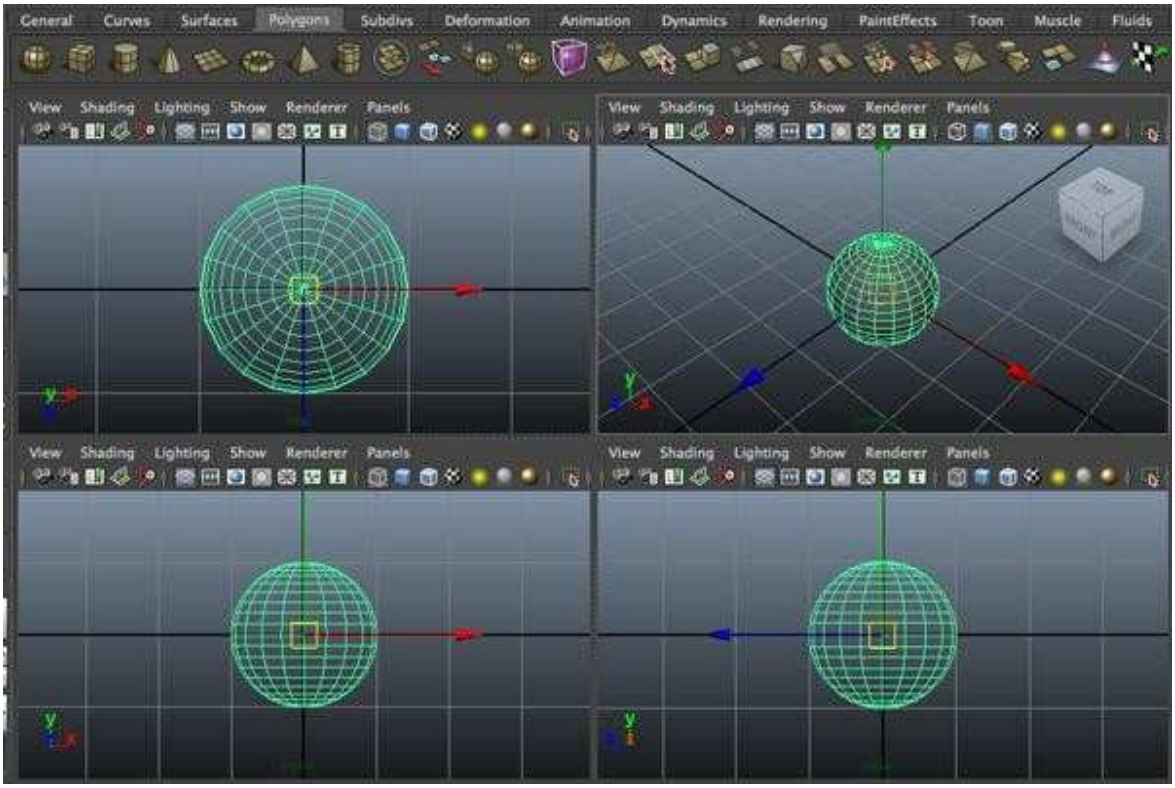


---

2. Views

When working in a 3D environment is very important to check that the object you are building looks good in all the views. Looking to just one view can be deceiving and you object might not be positioned correctly.

In order to ease the design process Maya provides up to 4 views, one perspective view and three orthographic views.



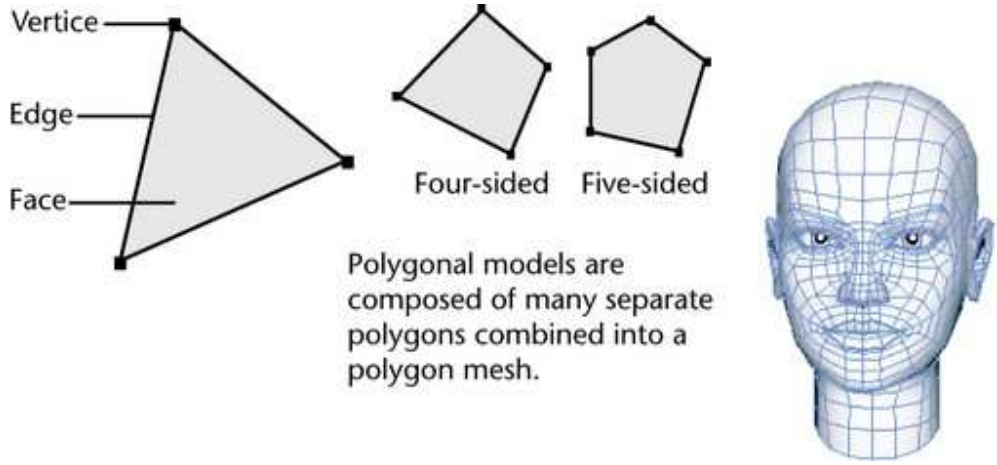
When you look at the object from the perspective view you can revolve the camera to freely tumble around the object. However, when you use orthographic views (top, side & front) they only focus in 2 axes at the time, so you can‘t tumble around (unless you use the tumble tool).

---

3. Polygonal Modeling vs NURBS Modeling

Polygons:

Polygons are the most basic geometry in Maya they are straight-sided shapes of 3 sides or more. Polygons have faces, edges & vertices, some examples of polygons or primitive shapes are spheres, cubes, cylinders, cones, and planes.

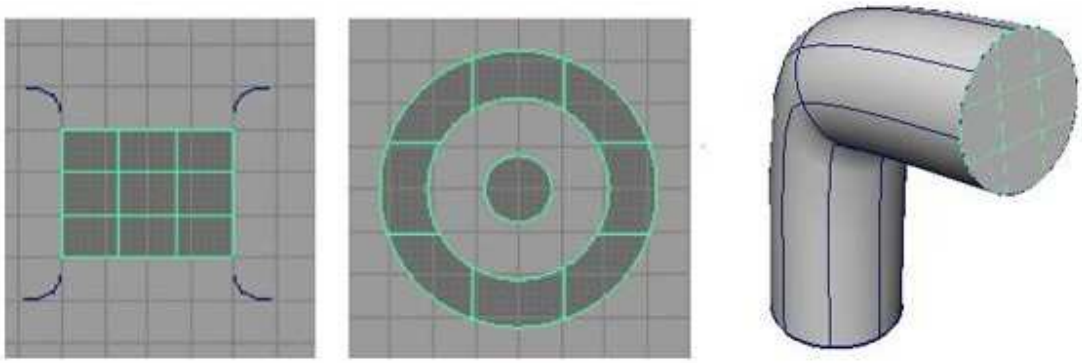


Because polygon‘s surfaces can be directly extruded, scaled and positioned designers prefer them to create characters in Maya.

NURBS Curves and NURBS Surfaces:

NURBS curves stand for Non-Uniform Rational B-Splines. NURBS curves are very handy since you can draw them in Maya or import them from a vector program like Adobe Illustrator.

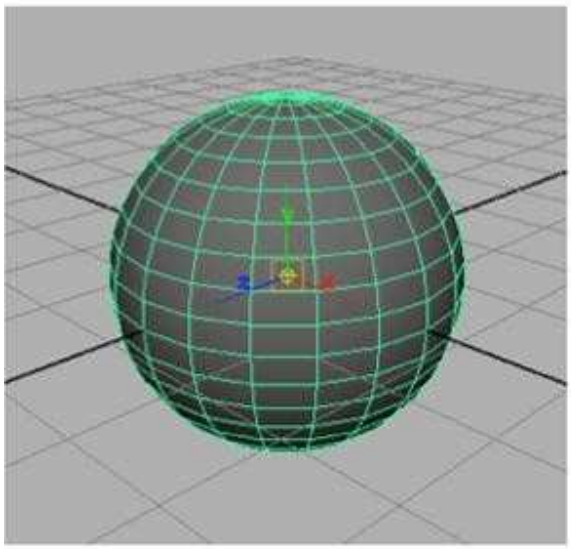
NURBS primitives or surfaces, use UV coordinate space and you can modify them by trimming away portions of their forms, beveling their edges, or by sculpting them into different shapes using the Maya Artisan sculpting tools

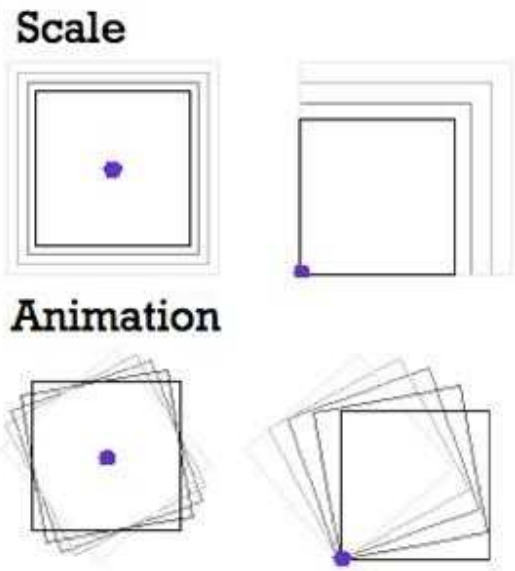


NURBS are preferred for constructing organic 3D forms because of their smooth & natural characteristics.

4. Pivot

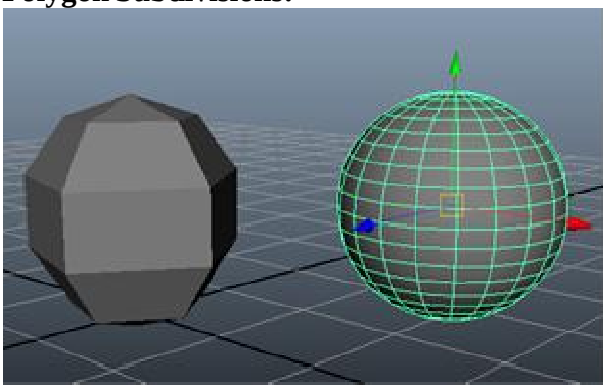
All transformations to an object are relative to the pivot point. The object‘s pivot affects scale since the object will scale out from or in toward the pivot point. The pivot also affects the object‘s rotation/animation because an object rotates around the pivot point.

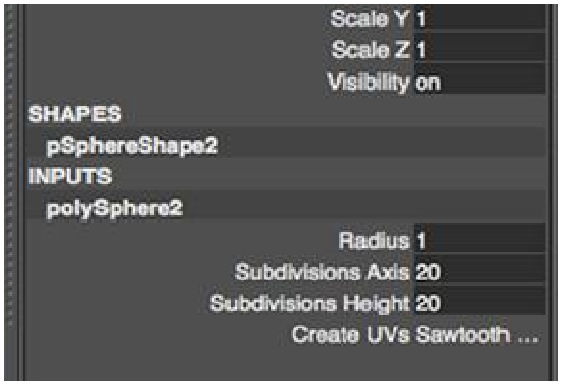




---

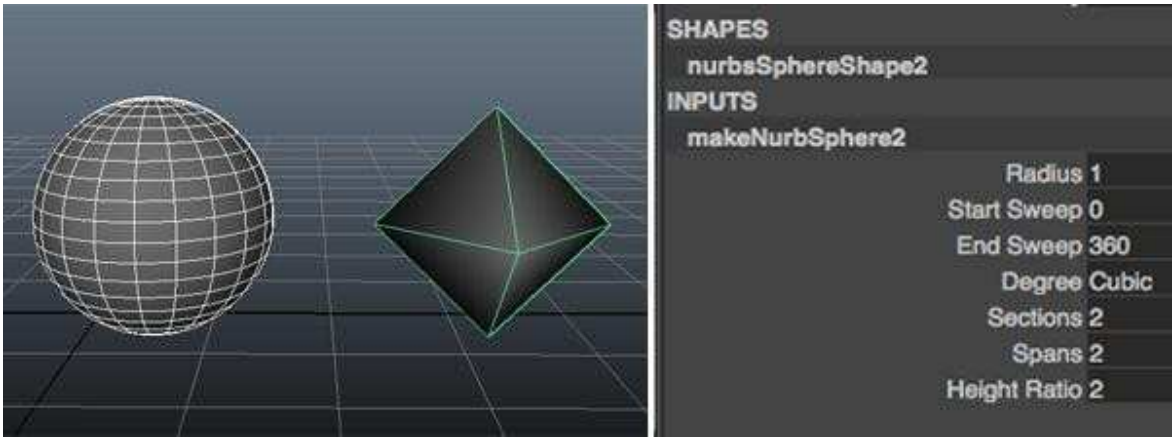
5. Subdivisions
Polygon Subdivisions:





If a polygon has less subdivisions, for example an value of –5|, it will have less detail and a rougher surface
If a polygon has more subdivisions, for example an value of –20|, it will have more detail & a smoother surface

NURBS Subdivisions



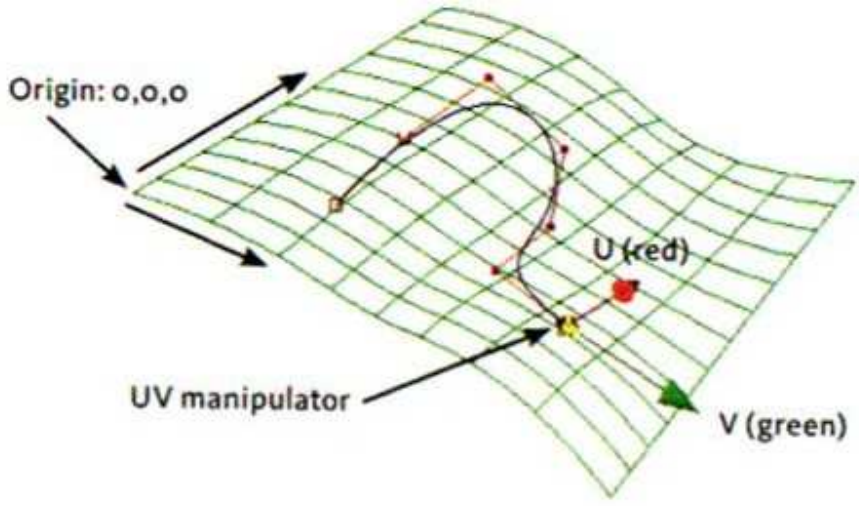
If a NURB has less sections & spans, for example an value of –2, it will have less detail and a rougher surface

If a NURB has more sections & spans, for example an value of –20, it will have more detail & a smoother surface

---

6. UV Coordinate Space:

Surfaces in Maya have their own coordinate space. UV Coordinate Space is useful when working with curve-on-surfaces and positioning textures.



UV coordinates on a live surface

---

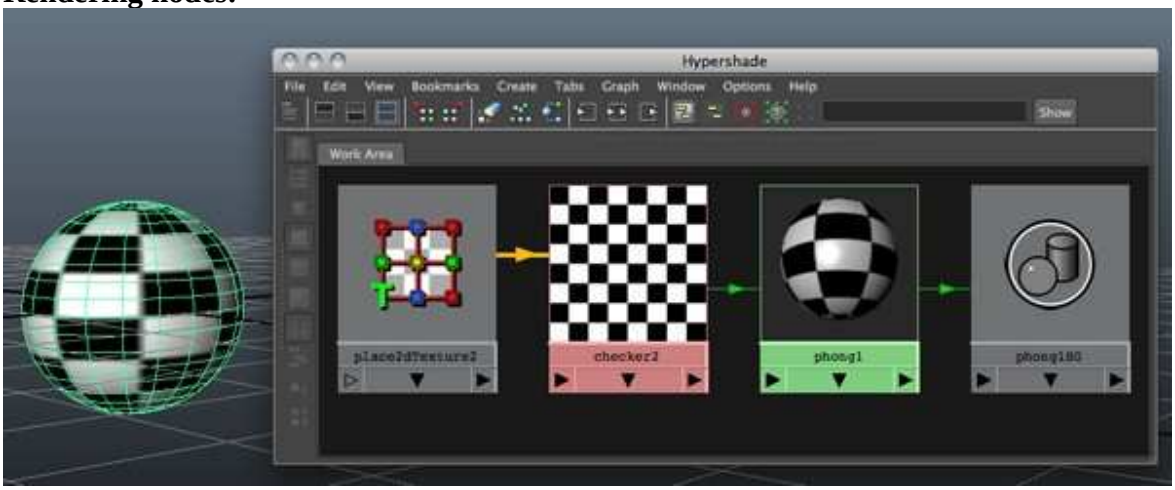
7. Nodes

Every element in Maya is built with a single or a series of nodes. Nodes define all attributes like lighting, shading and geometry.
For example a primitive polygon, such as a sphere, is built from several nodes: a creation node that records the options that created the sphere, a transform node that records how the object is moved, rotated, and scaled, and a shape node that stores the positions of the spheres control points.

Shape nodes:

Holds an object‘s geometry attributes or attributes other than the object‘s transform node attributes.

Rendering nodes:



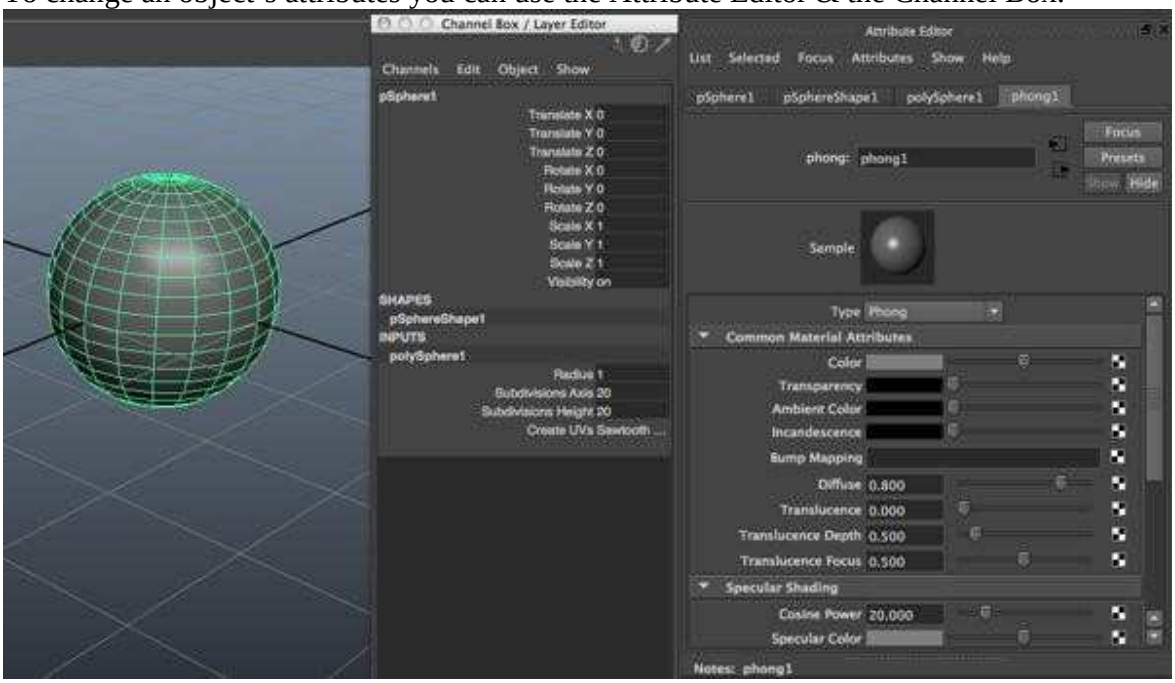
Materials and textures each have nodes containing attributes that control their look. Texture placement nodes have attributes that control how a texture is fitted onto a surface.

---

8. Attributes:

An attribute is a position associated with a node that can hold a value or a connection to another node. Attributes control how a node works.

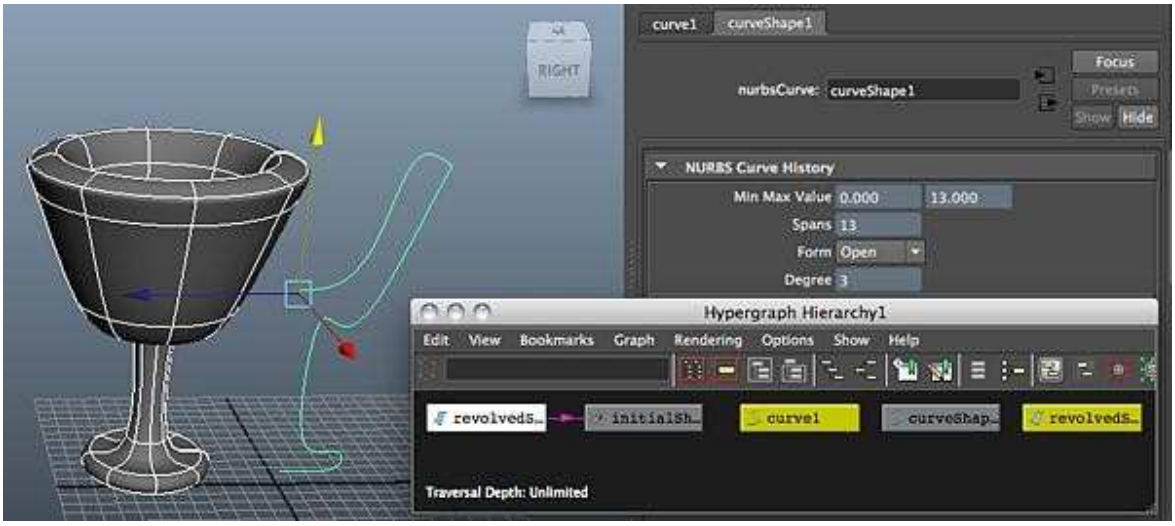
To change an object‘s attributes you can use the Attribute Editor & the Channel Box.



---

9. Construction History:

When you work in Maya, all the actions you make create nodes in the construction history of the objects you are working on. Construction history is very important because it allows you to change an object.

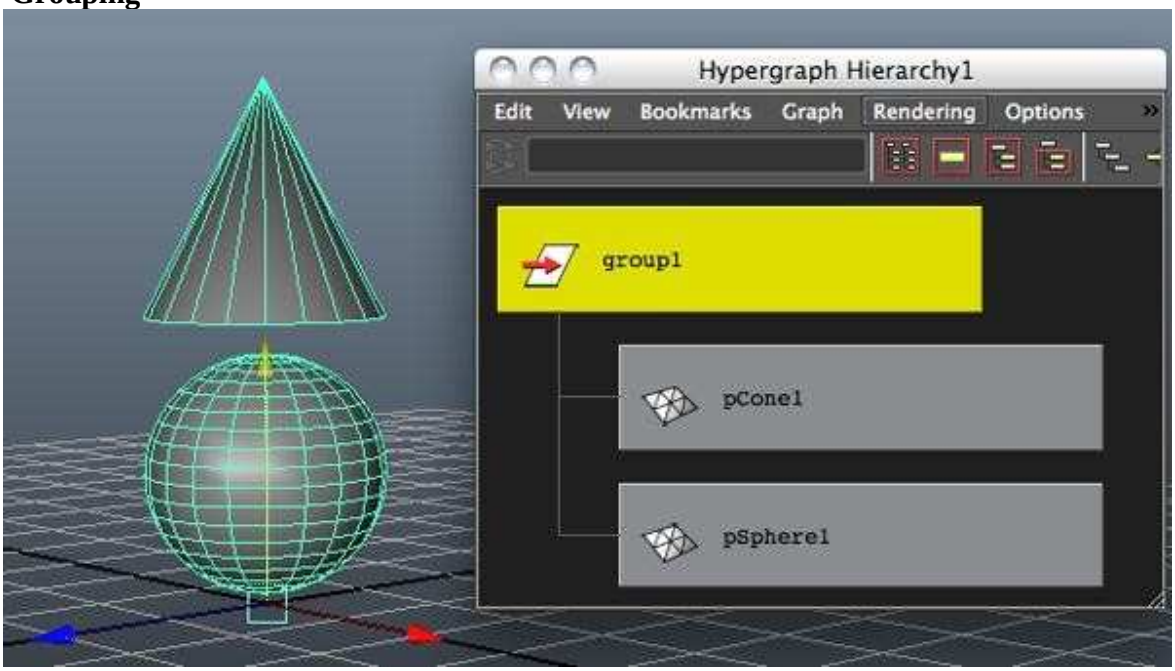


If you are only modeling it‘s recommended that you delete construction history once you are 100% sure you have the object you need. This prevents accidental changes and speeds up the shading & rendering process. However, if you are going to animate is not recommended that you don‘t delete construction history since some dependency nodes have attributes that can be animated.

---

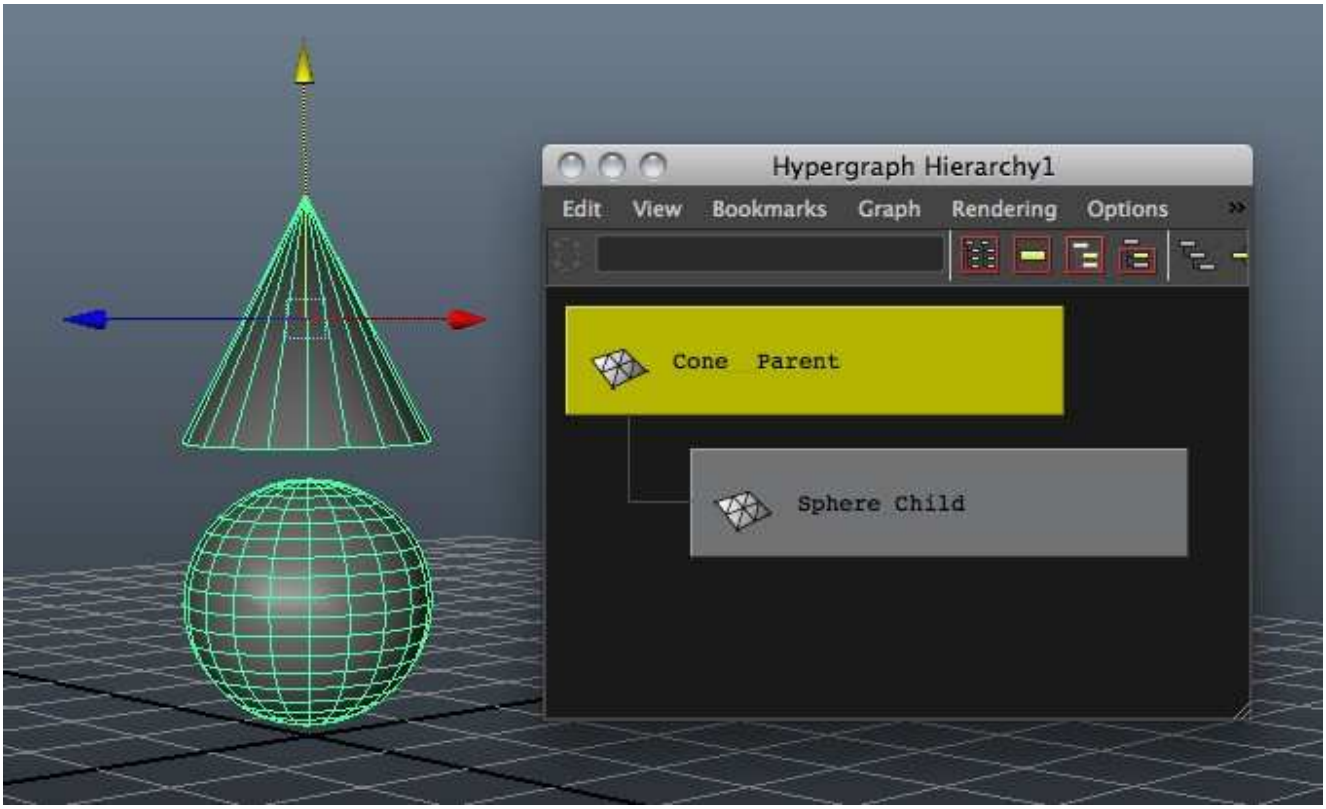
10. Hierarchy

Grouping



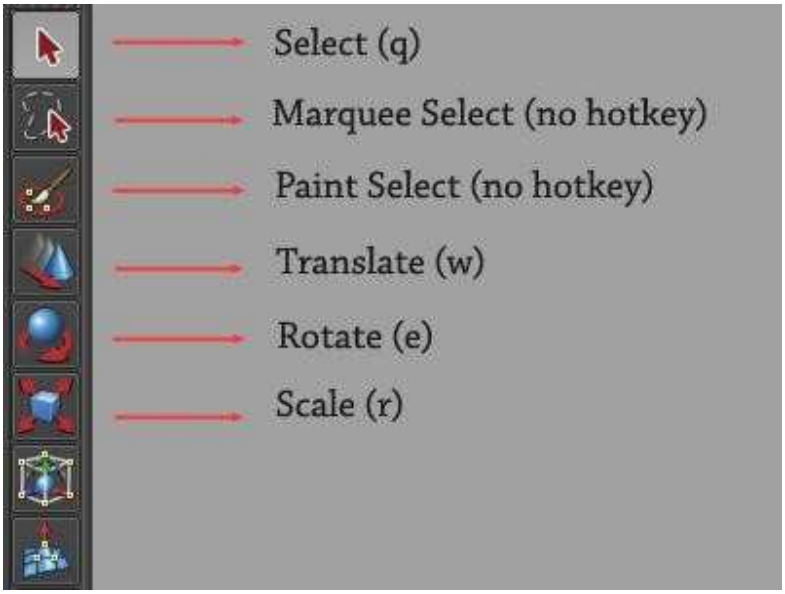
To control multiple objects with one node, you can group objects together under a new transform node. By grouping objects, you can move, shade, texture, and do many other actions to all the objects at the same time.

Parenting



The child objects have independent nodes, for example a child can rotate by itself without rotating the parent. However, if the parent is rotated or moved the child follows.

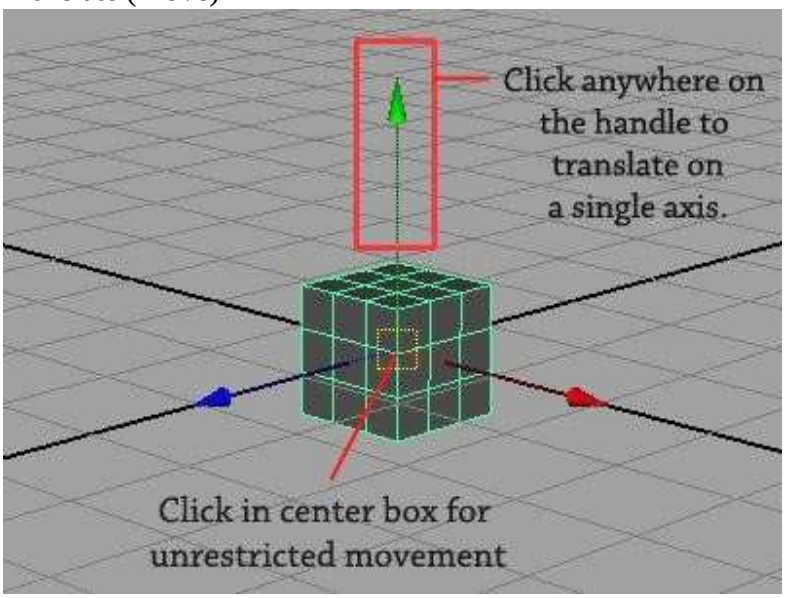
<u>Object Manipulation Tools</u>



Maya's tool selection icons on the left side of the user interface.
So now you know how to place an object in your scene and modify some of its basic attributes. Let's explore some of the ways we can change its position in space. There are three basic forms of object manipulation in any 3D application—translate (or move), scale, and rotate.
Obviously, these are all operations that sound relatively self-explanatory, but let's take a look at some of the technical considerations.

There are two different ways to bring up the translate, scale, and rotate tools:
First, they can be accessed from the toolbox panel (pictured above) on the left side of your viewport.
The second (preferred method) is to use keyboard hotkeys. During the modeling process, you'll be switching between tools constantly, so it's a good idea to learn the commands as quickly as possible.
With an object selected, use the following hotkeys to access Maya's translate, rotate, and scale tools:
Translate - **w**.
Rotate - **e**.
Scale - **r**.
To exit any tool, hit **q** to return to selection mode.

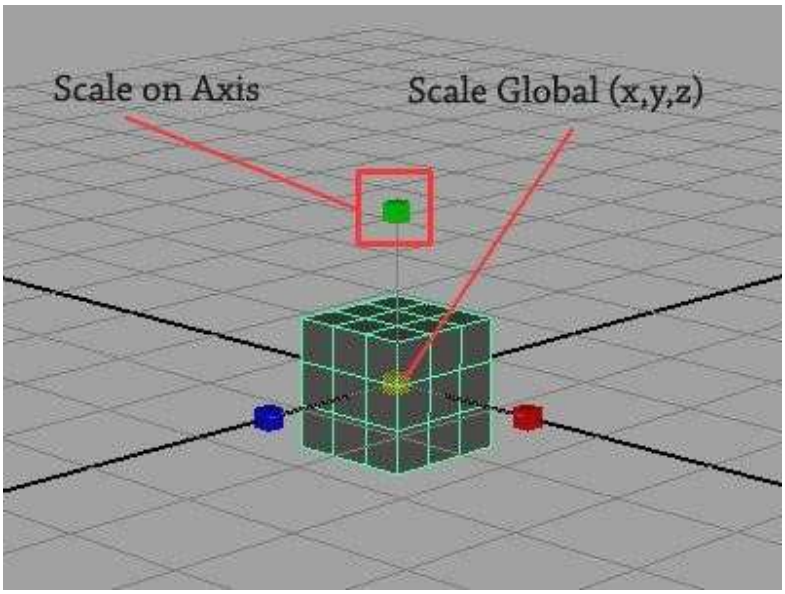
Translate (Move)



Select the object you created and strike the **w** key to bring up the translation tool.
When you access the tool, a control handle will appear at your object's central pivot point, with three arrows aimed along the X, Y, and Z axes.
To move your object away from the origin, click any one of the arrows and drag the object along that axis. Clicking anywhere on the arrow or shaft will constrain movement to the axis it represents, so if you only want to move your object vertically, simply click anywhere on the vertical arrow and your object will be constrained to vertical movement.

If you'd like to translate the object without constraining motion to a single axis, clicking in the yellow square at the center of the tool to allow free translation. When moving an object on multiple axes, it's often beneficial to switch into one of your orthographic cameras (by clicking **spacebar**, in case you'd forgotten) for more control.

Scale

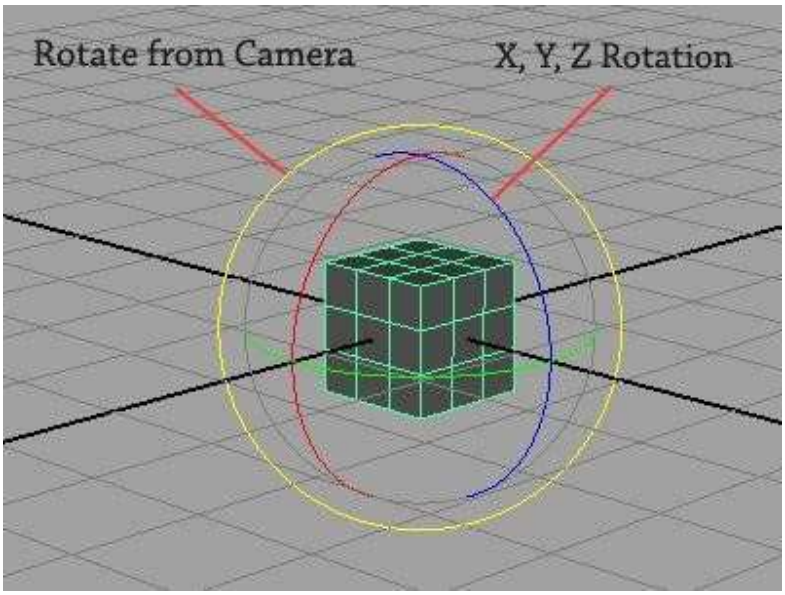


The scale tool functions almost exactly like the translate tool.

To scale along any axis, simply click and drag the (red, blue, or green) box that corresponds to the axis you'd like to manipulate.

To scale the object globally (simultaneously on all axes), click and drag the box located at the center of the tool. Simple as that!

Rotate



As you can see, the rotation tool appears and operates slightly different from the translate and scale tools.

Like translate and scale, you can constrain rotation to a single axis by clicking and dragging any of the three inner rings (red, green, blue) visible on the tool.
You can freely rotate the object along multiple axes, by simply clicking and dragging in the gaps between rings, however you're afforded a lot more control by rotating an object one axis at a time.
Finally, by clicking and dragging on the outer ring (yellow), you can rotate an object perpendicular to the camera.
With rotation, there are times when a bit more control is necessary—on the next page we'll look at how we can use the channel box for precise object manipulation.

In animation, there's more to animate than simple movement. Animation requires thinking about motion, timing, and smoothness of action. Almost anything in Maya with a number attached to it can be animated. Maya simplifies your work in creating the essence of animation---timing and motion. With maya , you can animate virtually anything you can imagine, no matter how surreal. When you set a keyframe (or *key*), you assign a value to an object‘s attribute (for example, translate, rotate, scale, color, etc.) at a specific time.

ANIMATION

Most animation systems use the *frame* as the basic unit of measurement because each frame is played back in rapid succession to provide the illusion of motion.
The frame rate (frames per second) that is used to play back an animation is based on the medium that the animation will be played back (for example, film, TV, video game, etc.)
When you set several keys at different times with different values, Maya generates the attribute values between those times as the scene plays back each frame. The

Animation Control and Interface

With Maya's animation controls, you choose how to key and play an animation. Two components of Maya's user interface are specific to animation: the Range Slider and the Time Slider. You can also quickly access and edit animation preferences from the animation controls area.

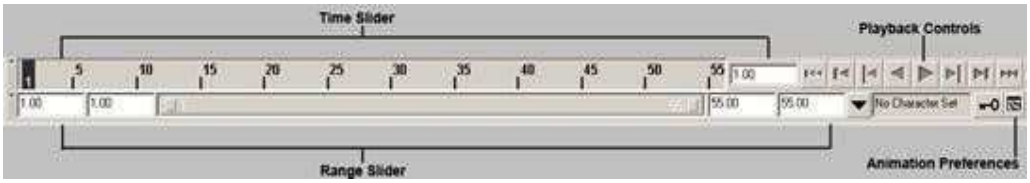


Fig 1 Animation controls

Between the Range slider and the Animation Preferences button are the current character control features and the Auto Key button.

Time Slider

The Time Slider is a vital part of the animation interface in Maya. The Time Slider Controls the Playback range, keys and breakdowns with in the playback range.

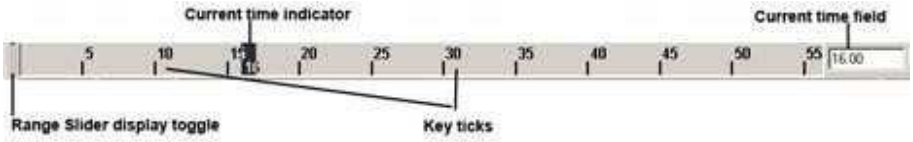


Fig 2 Time slider

Click in the Time Slider area and drag left and right to "scrub" the animation back and forward in time.

Key Ticks

Key Ticks are red marks in the Time Slider where you set a key for the selected object. Breakdowns are a special type of key .The visibility of Key Ticks can be turned off or on in the Preference window. The current Time indicator is a gray block on the Time Slider. You can drag it to move forward and backward in your animation.

Current Time Field
A black line indicates the current time field. When keys have been set for the currently selected object, thin vertical red lines appear in the Time Slider area to indicate the times for those keys.

Range Slider
The Range Slider controls the playback range reflected in the Time Slider. The Range Slider sets the total length of the animation in frames. You can also use the Range Slider to temporarily limit the range of playback and set the playback start and end frames.

Fig 3 Range slider

- You can toggle whether the Range Slider is Displayed or hidden by selecting Display>UI elements>Range Slider.
- Animation Start Time sets the start time of the animation.
- Animation End Time sets the end time of the animation.
- Playback Start Time This shows the current start time for the playback range. You can change it by entering a new start time.
- Playback End Time This shows the current end time for the playback range. You can change it by entering a new end frame.
- Range Slider Bar This lets you control the playback range of your animation up to the limits of the Animation start/end settings.

You use the Preferences dialog box to change values for the animation timeline and playback. You can also set the total time for your animation, the size of the timeline, and other related features.

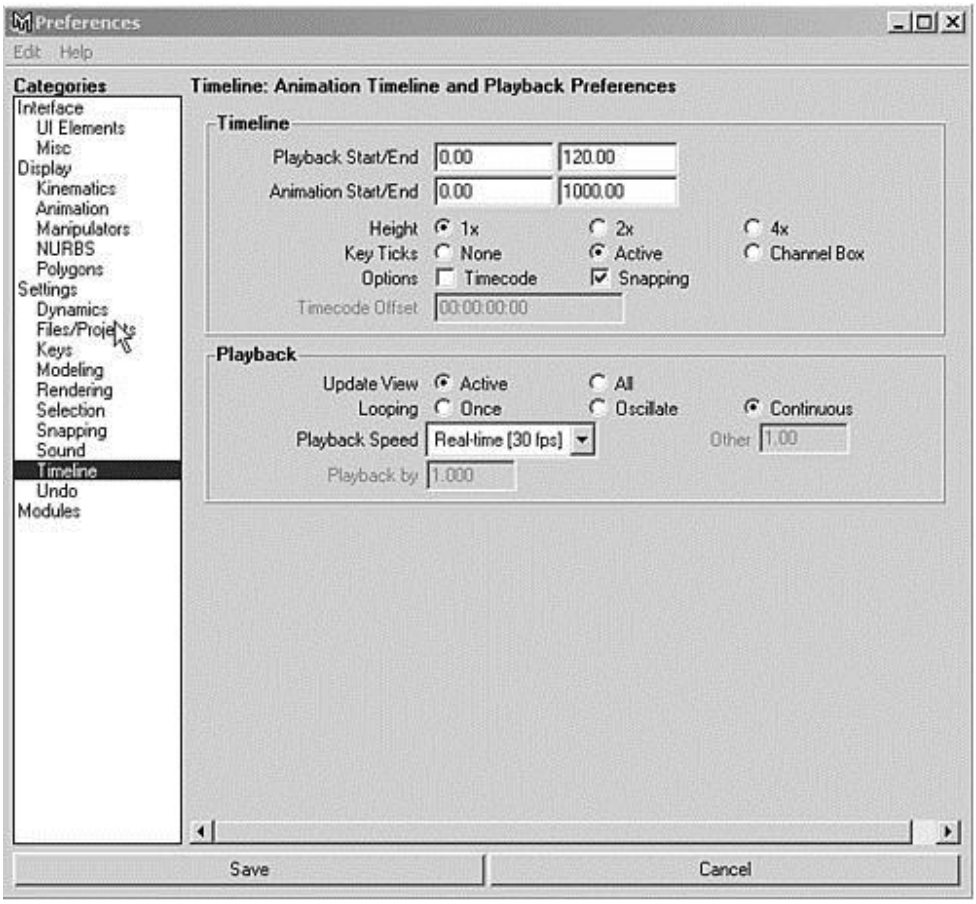


Fig 4 Preferences dialog box

Terms in Animation

Frame Rate
Frame rate is the first aspect of animation. By default, Maya sets your animation to Film, which plays at 24fps.You use 30fps in the United States and 25fps in other countries.

Range
The range of an animation determines the total length in frames. Then, multiply the animation's length in seconds by the frame rate .For e.g in this case you are using 24fps and animation length is of 2seconds.
24fps X 2secs =48 Frames

Setting Keys
You can set a Key by selecting Animate >Set Key. The attributes set by this menu item depend on the set Key option settings.

Animation Types
There are following types to animate your scene.

Path Animation

In this method, you create a NURBS-based curve and then attach an object to it in your scene. The object then follows the curved path to simulate motion. You can choose at which time the object is positioned at any point along the path, so the object can reverse itself, pause, or oscillate, if you want. The object automatically rotates from side to side as the curve changes directions. If the object is geometry, it can also be automatically deformed to follow the contours of the curve.

Select Animate>Motion paths>Attach to Motion path

To Animate an object along a surface

- Choose create >Nurbs Primitives > Plane to create a Nurbs plane.
- Select Modify>Transformation Tools>Proportional Modification Tool to introduce contours on the plane.
- Select Modify > Make Live ,then draw a curve on the plane.
- Create an object to animate along the path , and shift -click on the curve on surface to select it.
- Select Animate>Motion Paths >Attach to Motion Path option window. Ensure that follow is on ,and set the up direction to normal so that the object will stay normal to the surface.
- Click play to see your animation.

Flow Path Object Function

The Flow Path Object function creates a lattice around an object .

Select Animate>Motion paths>Flow Path Object

Keyframe Animation

Keyframe animation is the standard animation method. In this method, you set keys for an object's extreme positions and let the computer fill in the in-between motion. A key is an anchor point for a particular attribute at a designated time. When the animation reaches that specified time, the object's attribute will be at the value you set. As you set keys, you specify the time at which those changes in the attribute's value take place.

To set keys with the auto keyframe method, you click the Auto Keyframe button in the Range Slider (it turns red to indicate that it's enabled). With auto keyframing, you can animate quickly by simply dragging the Time Slider to a given frame and then changing an attribute.

Nonlinear Animation

Nonlinear animation is a more advanced method of animation. Unlike keyframing, nonlinear animation is completely independent of time. You blend and layer animation sequences—called clips—to set up the motion for objects. You can also use this method to explore variations in parts of the animation without losing your previous work or affecting other parts of the animation. For example-you can make the walking part of the animation a clip and then adjust the leg motion without affecting the way the rest of the character moves.

Graph Editor

The Graph Editor is a helpful tool for tweaking values for keys you have set. It gives you a visual representation—a curved line—of the attributes that are animated. The animation time goes from left to right, and any keyed variable appears as a line that ramps up or down to indicate its value over time. It can help you visualize how things are changing and how fast. You can pan and zoom this panel like any other.

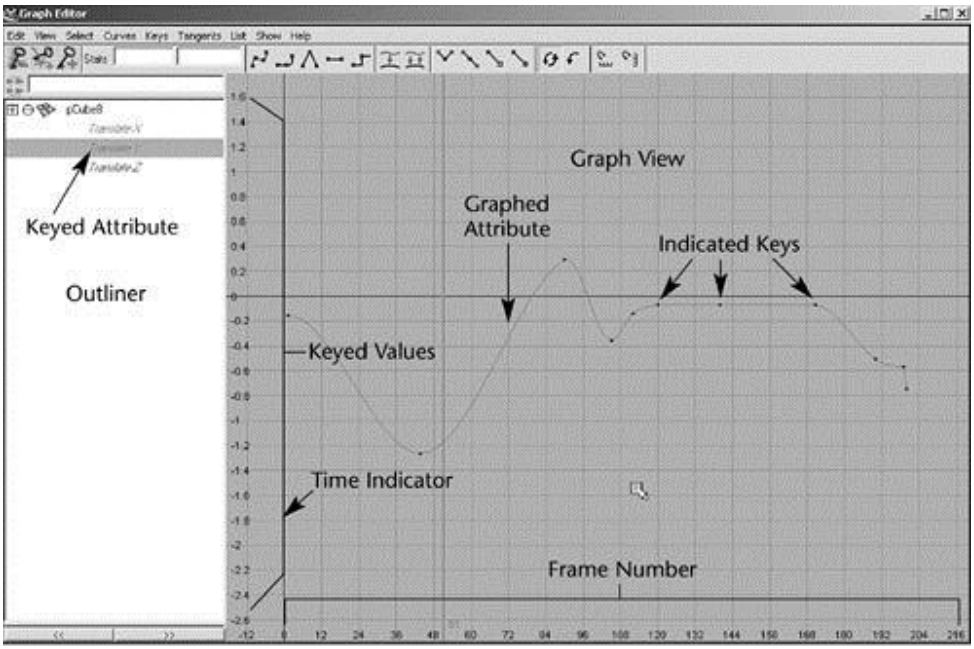


Fig 5 Graph Editor

To use it as a free-floating window, simply open it from the Hotbox (Window >Animation Editors> Graph Editor).

Graph Editor's Components

Menu bar

The Graph Editor menu bar contains tools and operations for manipulating animation curves and keys with in the graph view of the Graph editor. The Edit menu is similar to the one in text editors or word processors, except that you're working with keys instead of text.

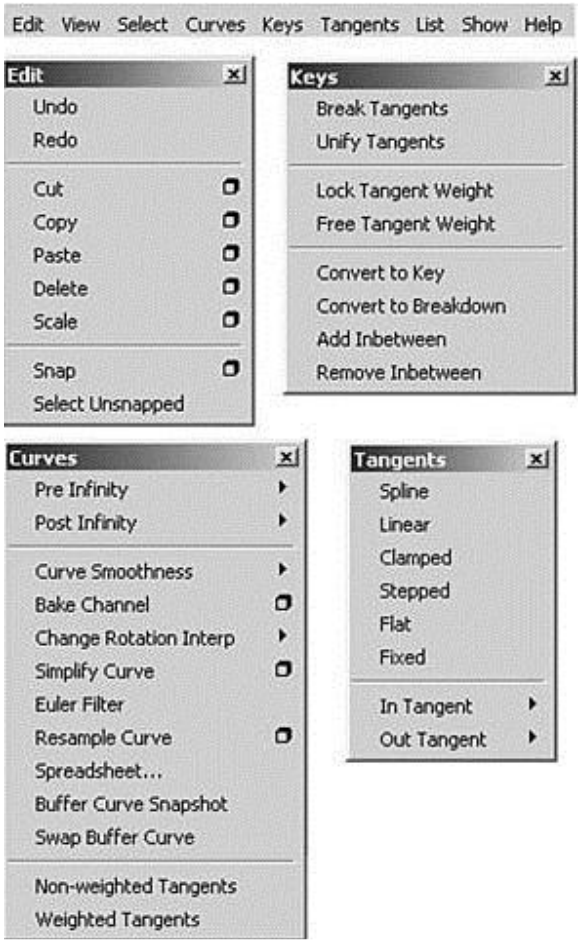


Fig 6 Menu Bar

The Edit Menu
The menu items appear under Edit menu behave in a similar fashion to the main Edit menu in the modeling view.

The View Menu
This menu controls which components are visible, and therefore editable, in the graph view of the Graph Editor.

The Select Menu
These options control which component of an animation curve are available for selection and editing.

The Curves Menu
The Curves menu gives you control over how the curves are set up with the keys in your scene.

The Keys menu
This menu includes Tangents which causes the manipulation of an in or out tangent handle.
The Tangents Menu

This describes the entry and exit of curve segments from a key.

The List Menu
This menu Loads the objects.

Toolbar
The toolbar gives you quick access to functions for modifying animation curves and keys.

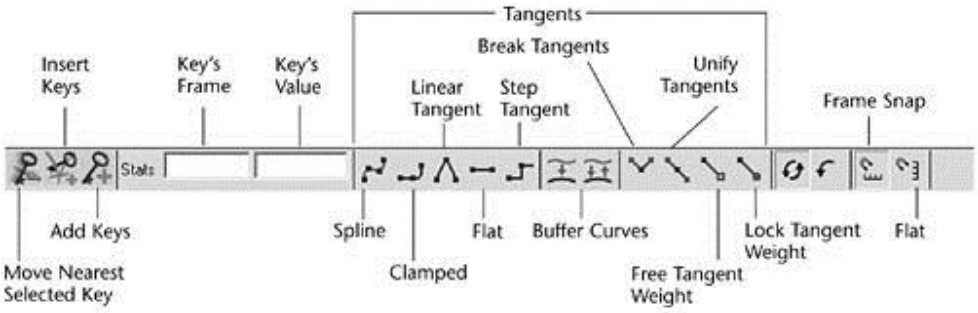


Fig 7 Toolbar

- Buffer curves Use Buffer Curve snapshot and swap Buffer curves to compare changes to the current animation curve with its previous shape.

- Break Tangents Allows manipulation of the in and out tangent handles individually so you can edit the curve segment entering the key without affecting its opposite handle.

- Unify Tangents causes the manipulation of an in or out tangent handle to affect its opposite handle equally. It retains the relative position of the tangent handles even after tangents are individually adjusted.

- Lock Tangent Weight specifies that when you move a tangent ,only its angle can be changed.

- Free Tangent Weight specifies that when you move a tangent ,only its angle can be changed. This allows the weight of a tangent to be adjusted as well as the angle.

- Clamped tangent creates an animation curve that has the characteristics of linear and spline curves. The key's tangents will be spline unless the value of two adjacent keys are very close.

- Step tangent creates an animation curve whose out tangent is a flat curve.
- Flat Sets the in and out tangents of the key to be horizontal.

The Dope Sheet
The Dope Sheet is another animation editor in Maya that is similar to the Graph Editor. Instead of displaying curves, the Dope Sheet displays key times as colored rectangles and lets you edit event timing in blocks of keyframes and synchronize motion to a sound file.

To open the Dope Sheet
Select Window > Animation Editors > Dope Sheet

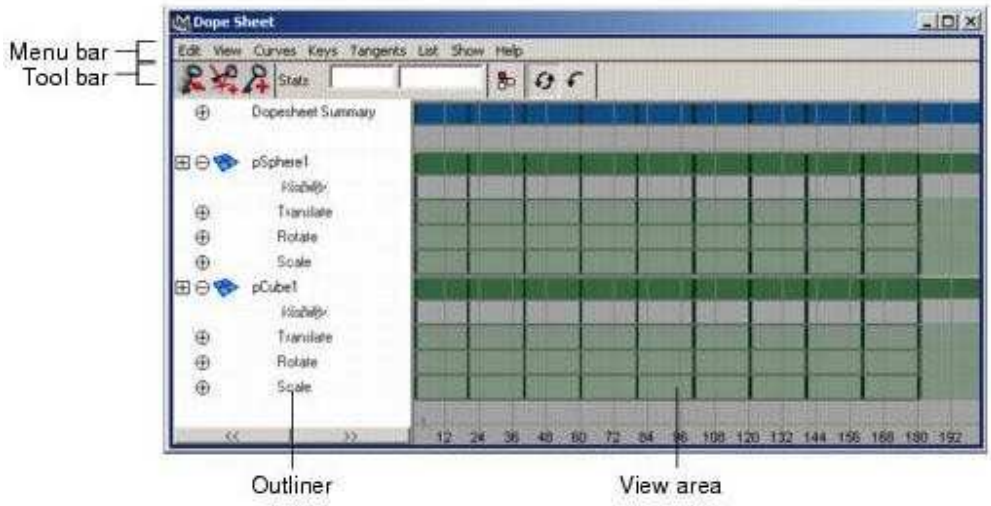


Fig 8 Dope sheet editor

To Place the Dope Sheet in a View
Select the view.
Select Panels > Panel > Dope Sheet.

EXPERIMENTS

Experiment-1

AIM:-To create a wine glass using EP curve tool.

STEPS:-

Step 1: Select _Modeling‘ from main menu bar.

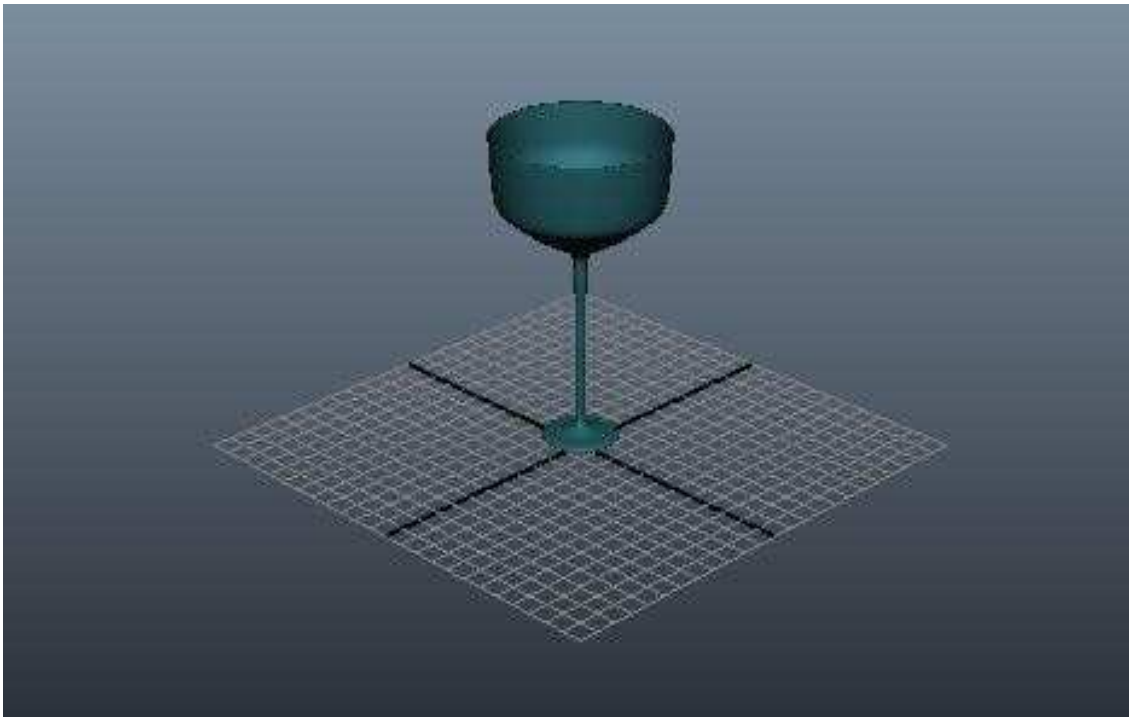
Step 2: Select _EP curve tool‘ from _Curves‘.

Step 3: Click each coordinate in plane to fix the points forming the curve in the shape of half wine glass.

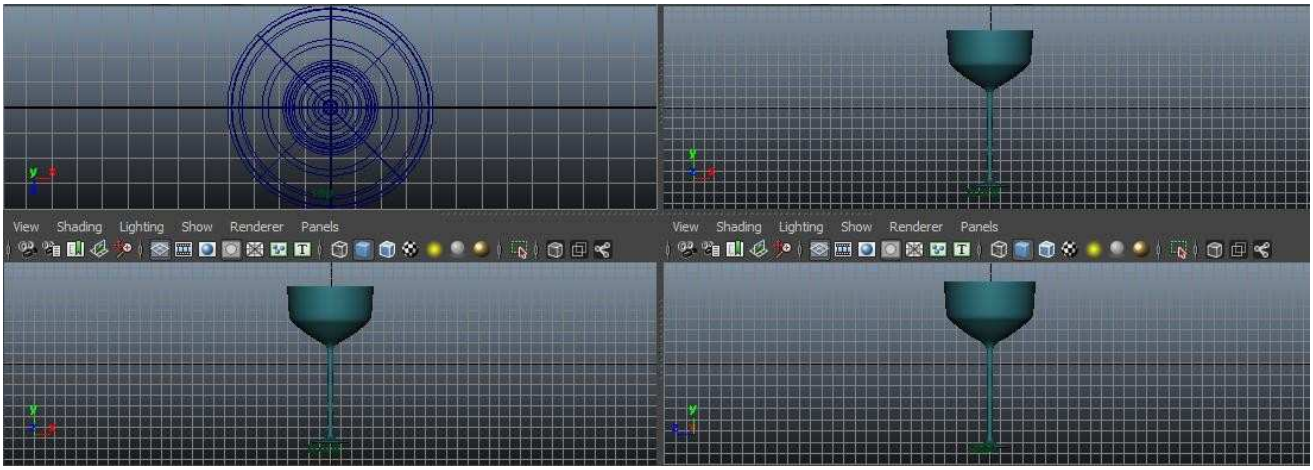
Step 4: Press 5 to solidify and click _Surfaces‘ from Maya title bar and select _Revolve‘ to obtain complete wine glass.

OUTPUT

Single Perspective View



Four View



Experiment-2

AIM:-To create a building with balcony and a door at the base of the building using polygon and NURBS primitives.

STEPS:-

Step 1: Select _Modeling‘ from main menu bar.

Step 2: Create the building by clicking on _Polygon‘s primitives‘ and then on _Cube‘.

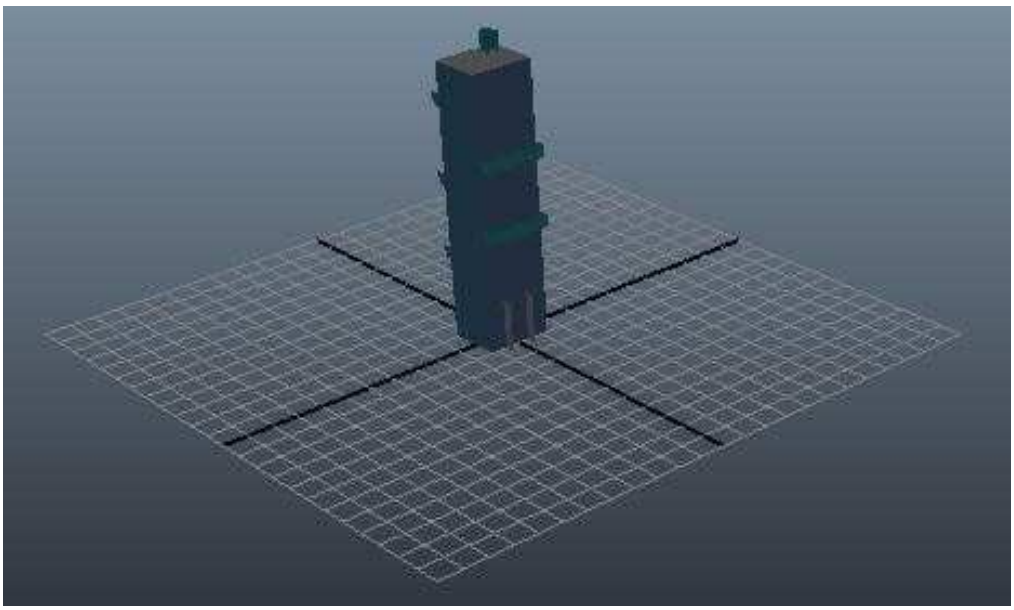
Step 3: Create a door using NURBS primitive cube.

Step 4: Create balconies using NURBS primitive cube and remove the top and backside.

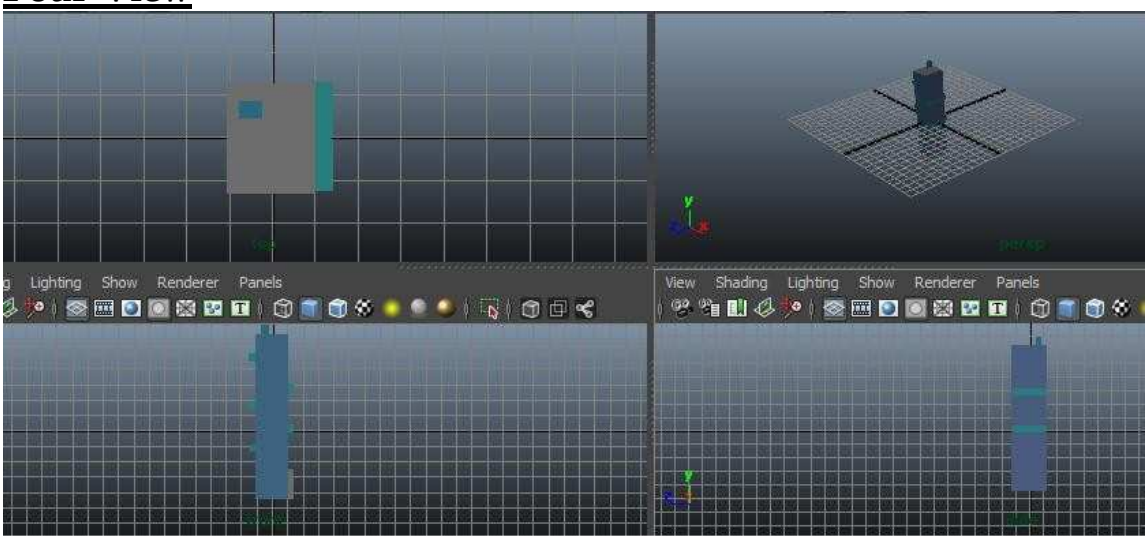
Step 5: Press 5 to solidify and render.

OUTPUT

Single Perspective View



Four View



Experiment-3

AIM:-To bounce a ball using Animation.

STEPS:-

Step 1: Select _Animation‘ from main menu bar.

Step 2: Click _Create‘ on Maya title bar and then click on _Polygon‘s primitives‘ and then click on _Sphere‘.

Step 3: Click _Move‘ tool to move the ball to position in a frame.

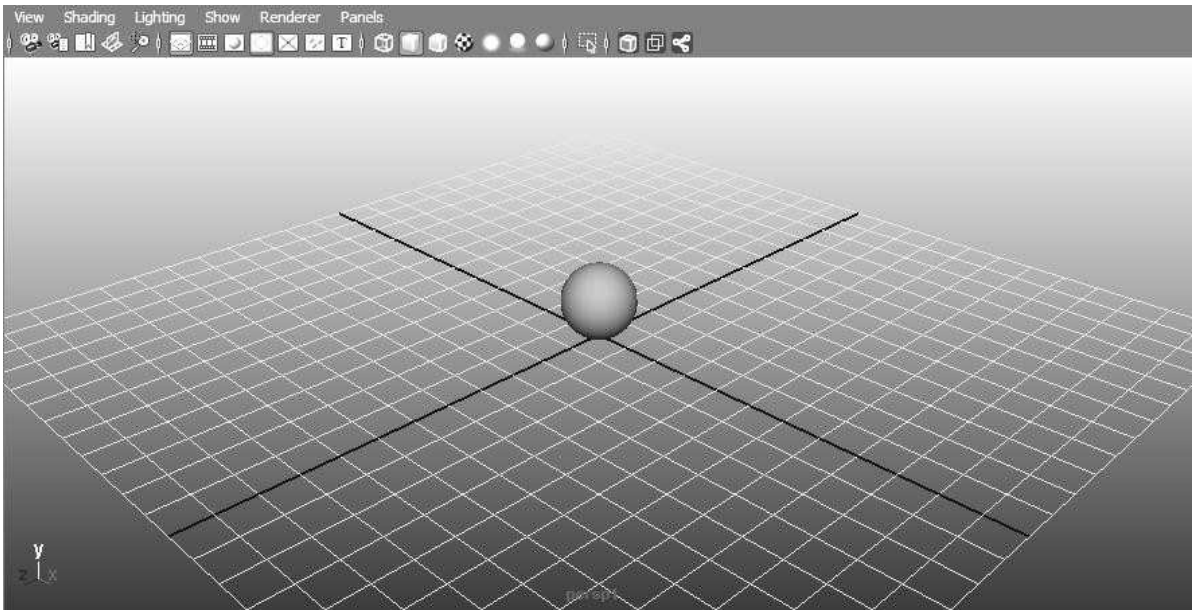
Step 4: Fix the position of the ball in a frame by pressing the _Set‘ key(Shortcut-S).

Step 5: Move to the next frame and set another position of the ball using Set key.

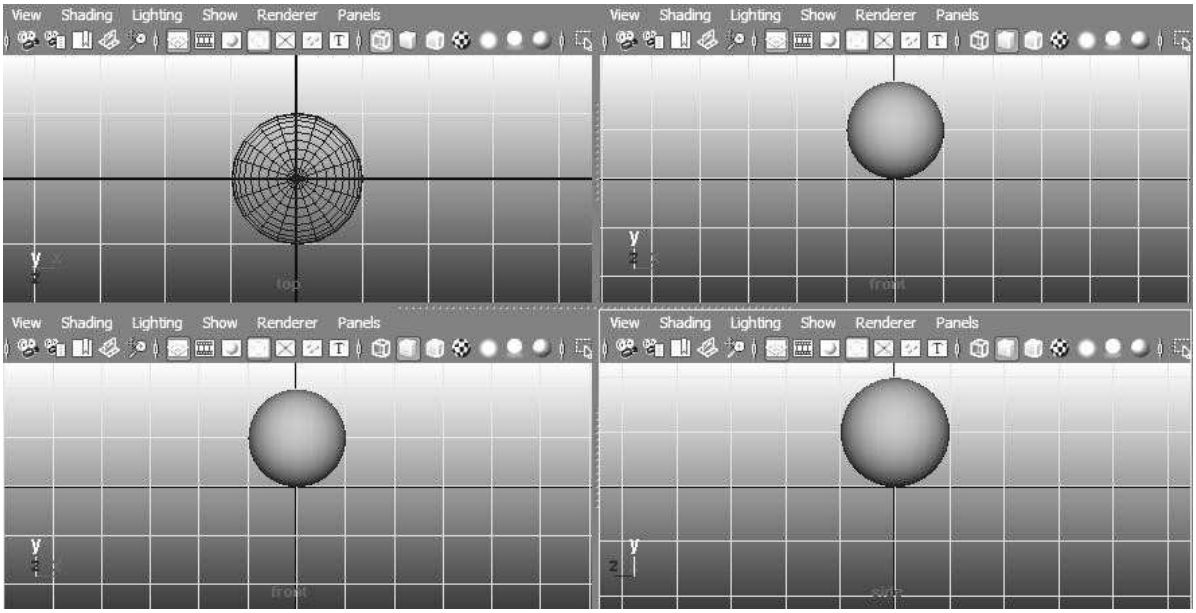
Step 6: Click _Play‘ to view the moving ball.

OUTPUT

Single Perspective View



Four View


View Shading Lighting Show Renderer Panels
View Shading Lighting Show Renderer Panels
View Shading Lighting Show Renderer Panels
View Shading Lighting Show Renderer Panels