Introduction to Computer Graphics

(CS602)

Lecture 13

Drawing Example

Let us now learn some of the implementation techniques. So far we already have done with learning drawing primitives including output primitives as well as filling primitives. Also we have studied transformations. So we should be in position to make use of them in two-dimensional drawing. Though most of you will think that they can draw two-dimensional drawing very easily yet it may not be true due to lack of knowledge of some implementation techniques, which are very useful in drawing as well as in transformation. So we will cover this with some examples.

13.1 Drawing Table

First of all we are going to draw a very simple drawing that is a “table”. Yes, a simple rectangular table with four legs. So, in order to draw such table we have to draw “table top” plus four legs connecting four edges of the rectangle.

13.2 Design

Here we will first design the table like an ordinary student. So, what we will do we will see the location of the table. For example assume that our screen has dimensions 640*480 and initially we want to draw table right in the middle of the center. Also, another factor is important that is y axis travels from top to bottom. That is y=0 will be the top edge of the screen and 480 will be the lower edge of the screen. Another thing is the dimension of the table we want to draw. Therefore, we have table that has width 20, length 14 and height 10.

Therefore, we have to find out four vertices that make the corners of the table. So, first of all consider x coordinate. Left edge of the table will be 10 less from the center of the screen that is 320. Therefore, x1 and x4 values will be 310. Similarly right edge of the table will be 10 plus center of the screen. Therefore, x2 and x3 values will be 330. Similarly, top edge of the table will be 10 less from the center of the screen that is 240. Therefore, y1 and y2 values will be 233 and y3 and y4, which are lying on the lower edge, will be 247. Finally, last parameter is required to define the length of legs. Having length of legs we have to simply draw vertical lines of that length starting from each corner respectively. Therefore, following code will be required to draw such a table:

```
void translate(int tx, int ty)
{
      xc+=tx;
      yc+=ty;
      x1+=tx;
      x2+=tx;
      x3+=tx;
      x4+=tx;
      y1+=ty;
      y2+=ty;
      y3+=ty;
      y4+=ty;
}

void rotate (float angle)
{
      int tempx=x1;
      x1=xc+(tempx-xc)*cos(angle)-(y1-yc)*sin(angle);
      y1=yc+(tempx-xc)*sin(angle)+(y1-yc)*cos(angle);
      tempx=x2;
      x2=xc+(tempx-xc)*cos(angle)-(y2-yc)*sin(angle);
      y2=yc+(tempx-xc)*sin(angle)+(y2-yc)*cos(angle);
      tempx=x3;
      x3=xc+(tempx-xc)*cos(angle)-(y3-yc)*sin(angle);
      y3=yc+(tempx-xc)*sin(angle)+(y3-yc)*cos(angle);
      tempx=x4;
      x4=xc+(tempx-xc)*cos(angle)-(y4-yc)*sin(angle);
      y4=yc+(tempx-xc)*sin(angle)+(y4-yc)*cos(angle);
}

void scale(int sx, int sy)
{
      x1=xc+(x1-xc)*sx;
      x2=xc+(x2-xc)*sx;
      x3=xc+(x3-xc)*sx;
      x4=xc+(x4-xc)*sx;
      y1=yc+(y1-yc)*sy;
      y2=yc+(y2-yc)*sy;
      y3=yc+(y3-yc)*sy;
      y4=yc+(y4-yc)*sy;
      legLength*=sy;
}
```

```
      x1=310, x2=330, x3=330, x4=310;
      y1=233, y2=233, y3=247, y4=247;
      legLength=10;
```

`So, what I want that you should observe the issue. Now consider first of all translation. In translation you have to translate all the points one by one and redraw the picture. Instruction to translate the table will be of the fowm:`

Now in this design drawing is pretty simple. We have to draw 4 line between corner points. That is from (x1, y1) to (x2, y2), from (x2, y2) to (x3, y3), from (x3, y3) to (x4, y4) and from (x4, y4) to (x1, y1). That will suffice our table top. Next simply draw four lines each starting from one of the corner of the table in the vertical direction having length 10. Now let us see the simple code of drawing such table:

```
      //Table Top
      line (x1, y1, x2, y2);
      line (x2, y2, x3, y3);
      line (x3, y3, x4, y4);
      line (x4, y4, x1, y1);
      //Table Legs
      line (x1, y1, x1, y1+legLength);
      line (x2, y2, x2, y2+legLength);
      line (x3, y3, x3, y3+legLength);
      line (x4, y4, x4, y4+legLength);
```

Now is not that easy to draw table in the same manner? We will discuss the problem after take a bit look at basic transformations (translation, rotation, scaling). The code is:

```
      void translate(int tx, int ty)
      {
            xc+=tx;
            yc+=ty;
            x1+=tx;
            x2+=tx;
            x3+=tx;
            x4+=tx;
            y1+=ty;
            y2+=ty;
            y3+=ty;
            y4+=ty;
      }
```

Now seeing the code you can easily understand the idea. In translation we have to translate all points one by one and then redrawing the table at new calculated points. Later you will see in the other method that translation will only involve one line.

Anyhow next we will move to other transformation (rotation). Having pivot point at the center of the screen, we have to perform translation in three steps. That is translation then rotation and then translation. So take a look at the code:

```
void rotate (float angle)
{
      int tempx=x1;
      x1=xc+(tempx-xc)*cos(angle)-(y1-yc)*sin(angle);
      y1=yc+(tempx-xc)*sin(angle)+(y1-yc)*cos(angle);
      tempx=x2;
      x2=xc+(tempx-xc)*cos(angle)-(y2-yc)*sin(angle);
      y2=yc+(tempx-xc)*sin(angle)+(y2-yc)*cos(angle);
      tempx=x3;
      x3=xc+(tempx-xc)*cos(angle)-(y3-yc)*sin(angle);
      y3=yc+(tempx-xc)*sin(angle)+(y3-yc)*cos(angle);
      tempx=x4;
      x4=xc+(tempx-xc)*cos(angle)-(y4-yc)*sin(angle);
      y4=yc+(tempx-xc)*sin(angle)+(y4-yc)*cos(angle);
}
```

So, here calculations required each time and for each pixel; whereas; you will observe that we can make that rotation pretty simple. A similar problem is lying in the case of scaling that is to perform three steps; translation then scaling and then translation. So look at the code given below:

```
void scale(int sx, int sy)
{
      x1=xc+(x1-xc)*sx;
      x2=xc+(x2-xc)*sx;
      x3=xc+(x3-xc)*sx;
      x4=xc+(x4-xc)*sx;
      y1=yc+(y1-yc)*sy;
      y2=yc+(y2-yc)*sy;
      y3=yc+(y3-yc)*sy;
      y4=yc+(y4-yc)*sy;
      legLength*=sy;
}
```

Therefore, same heavy calculations involves in scaling. So, here we will conclude our first method and will start next method so that we can judge how calculations become simple. Now having all discussion on table drawing, let us now consider the complete implementation of class Table:

```
/*********************************************************************
Table is designed without considering pivot point simply taking points
according to the requirement.

Therefore, translation of table involve translation of all points.

Scaling and Rotation will be done after translation.
*********************************************************************/

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

float round(float x)
```

```
{
      return x+0.5;
}

class Table
{
      private:
            int xc, yc;//Center of the figure
            int xp, yp;//Pivot point for this figure
            int x1, x2, x3, x4;
            int y1, y2, y3, y4;
            int legLength;

      public:
            Table()
            {
                  xc=320, yc=240;//Center of the figure
                  xp=0; yp=0;//Pivot point for this figure
                  x1=310, x2=330, x3=330, x4=310;
                  y1=233, y2=233, y3=247, y4=247;
                  legLength=10;
            }

            void translate(int tx, int ty)
            {
                  xc+=tx;
                  yc+=ty;
                  x1+=tx;
                  x2+=tx;
                  x3+=tx;
                  x4+=tx;
                  y1+=ty;
                  y2+=ty;
                  y3+=ty;
                  y4+=ty;
            }

            void rotate (float angle)
            {
                  int tempx=x1;
                  x1=xc+(tempx-xc)*cos(angle)-(y1-yc)*sin(angle);
                  y1=yc+(tempx-xc)*sin(angle)+(y1-yc)*cos(angle);
                  tempx=x2;
                  x2=xc+(tempx-xc)*cos(angle)-(y2-yc)*sin(angle);
                  y2=yc+(tempx-xc)*sin(angle)+(y2-yc)*cos(angle);
                  tempx=x3;
                  x3=xc+(tempx-xc)*cos(angle)-(y3-yc)*sin(angle);
                  y3=yc+(tempx-xc)*sin(angle)+(y3-yc)*cos(angle);
                  tempx=x4;
                  x4=xc+(tempx-xc)*cos(angle)-(y4-yc)*sin(angle);
                  y4=yc+(tempx-xc)*sin(angle)+(y4-yc)*cos(angle);
            }

            void scale(int sx, int sy)
            {
                  x1=xc+(x1-xc)*sx;
                  x2=xc+(x2-xc)*sx;
                  x3=xc+(x3-xc)*sx;
                  x4=xc+(x4-xc)*sx;
                  y1=yc+(y1-yc)*sy;
                  y2=yc+(y2-yc)*sy;
                  y3=yc+(y3-yc)*sy;
                  y4=yc+(y4-yc)*sy;
                  legLength*=sy;
            }

            void draw()
            {
                  line (x1, y1, x2, y2);
                  line (x2, y2, x3, y3);
                  line (x3, y3, x4, y4);
                  line (x4, y4, x1, y1);
                  line (x1, y1, x1, y1+legLength);
                  line (x2, y2, x2, y2+legLength);
                  line (x3, y3, x3, y3+legLength);
                  line (x4, y4, x4, y4+legLength);
            }
};


void main()
{
      clrscr();
      int gdriver = DETECT, gmode, errorcode;
      initgraph(&gdriver, &gmode, "c:\\tc\\bgi");
      Table table;
      table.draw();
      setcolor(CYAN);
      table.translate(15, 25);
      table.draw();
      table.translate(50, 75);
      table.scale(3,2);
      table.draw();
      table.translate(-100, 75);
      table.rotate(3.14/4);
      table.draw();
      getch();
      closegraph();
}
```

13.3 Second Method

Well now we will design our table by considering pivot point. That is we will decide our pivot point and next all points will be taken according to that pivot point. Similarly you will see that this consideration will do a little effect on drawing portion of the code; otherwise all other things will become simpler.

13.4 Table Design

So let us start with designing the table, that is to calculate parameters of the table. That is 4 corners plus length of legs. First of all we assume that our pivot point is lying on the center of the screen and initially that is (0, 0). So having pivot point we will calculate other points with respect to that point.

So having length 20 units, left edge of the table will be ten digits away from the pivot point and away on the left side; therefore; x1 and x4 will be –10; and similarly right edge of the table will be ten digits away from the pivot point on the right side. Therefore, x2 and x3 will be 10 (yes, positive ten).

Now consider the top and lower edges of the table they will be 7 points away from the pivot point in each direction; therefore value of y on the upper edge will be –7 and on the lower edge it will be 7 (yes, positive seven). Now finally y1, y2 will be –7 and y3, y4 will be 7. Well, length of the leg will be simple ten. Therefore, now take a look at the parameters in this design:

```
xc=320, yc=240;//Center of the figure
xp=0; yp=0;//Pivot point for this figure
x1=-10, x2=10, x3=10, x4=-10;
y1=-7, y2=-7, y3=7, y4=7;
legLength=10;
```

13.5 Table Drawing

So, points x1, x2, x3, x4 are not having the value at which they will appear on the screen rather they are at the relative distance from the pivot point. Here, we are also using xc, yc that is center on the screen that will keep pivot point align. Now having vertices defined in this fashion our drawing method will be differ from the older one. That is while drawing lines we will add center of the screen and pivot point in each vertex. That will take us to the exact position of the screen. Let us look at the drawing code:

```
int xc=this->xc+xp;
int yc=this->yc+yp;
line (xc+x1, yc+y1, xc+x2, yc+y2);
line (xc+x2, yc+y2, xc+x3, yc+y3);
line (xc+x3, yc+y3, xc+x4, yc+y4);
line (xc+x4, yc+y4, xc+x1, yc+y1);
line (xc+x1, yc+y1, xc+x1, yc+y1+legLength);
line (xc+x2, yc+y2, xc+x2, yc+y2+legLength);
line (xc+x3, yc+y3, xc+x3, yc+y3+legLength);
line (xc+x4, yc+y4, xc+x4, yc+y4+legLength);
```

So, in the above code first we have added xp to xc in order to reduce some of the calculations required in each line drawing command. Next, we have added that calculated figure to all line drawing commands in order to draw them exactly at the position where it should be appear in the screen.

Now having a bit difficulty while drawing there are many more facilities that we will enjoy in especially transformation.

13.6 Table Transformation

So, first of all consider Translation. In this technique translation is quite simple that is simply add translation vector in the pivot point. All other points will be calculated accordingly. Now look at the very simple code of translation:

```
void translate(int tx, int ty)
{
      xp+=tx;
      yp+=ty;
}
```

So how simple add tx to xp and ty to yp. Similarly next consider rotation that is again very simple; no need of translation. Let us look at the code:

```
void rotate (float angle)
{
      int tempx=x1;
      x1=tempx*cos(angle)-y1*sin(angle);
      y1=tempx*sin(angle)+y1*cos(angle);
      tempx=x2;
      x2=tempx*cos(angle)-y2*sin(angle);
      y2=tempx*sin(angle)+y2*cos(angle);
      tempx=x3;
      x3=tempx*cos(angle)-y3*sin(angle);
      y3=tempx*sin(angle)+y3*cos(angle);
      tempx=x4;
      x4=tempx*cos(angle)-y4*sin(angle);
      y4=tempx*sin(angle)+y4*cos(angle);
}
```

So, here you can check that there is no extra calculation simply rotated points are calculated using formula that is used to rotate a point around the origin. Now similarly given below you can see calculations of scaling.

```
void scale(int sx, int sy)
{
      x1=x1*sx;
      x2=x2*sx;
      x3=x3*sx;
      x4=x4*sx;
      y1=y1*sy;
      y2=y2*sy;
      y3=y3*sy;
      y4=y4*sy;
      legLength=legLength*sy;
}
```

So very simple calculation is done again that is to multiply scaling factor with old vertices and new vertices will be obtained.

So, now we look at the class Table in which table is designed considering pivot point and taking all other points accordingly.

```
/*****************************************************************
Table is designed with considering pivot point and taking all other
points with respect to that pivot point.

Therefore, translation of table involves translation of only pivot
point, all other points will change respectively.

Scaling and Rotation will be done directly no translation or other
transformation is required.
*****************************************************************/


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

float round(float x)
{
      return x+0.5;
}

class Table
{
      private:
            int xc, yc;//Center of the figure
            int xp, yp;//Pivot point for this figure
            int x1, x2, x3, x4;
            int y1, y2, y3, y4;
            int legLength;
            int sfx, sfy;                 //Scaling factor
      public:
      Table()
      {
            xc=320, yc=240;//Center of the figure
            xp=0; yp=0;//Pivot point for this figure
            x1=-10, x2=10, x3=10, x4=-10;
            y1=-7, y2=-7, y3=7, y4=7;
            legLength=10;
            sfx=1, sfy=1;
      }

      void translate(int tx, int ty)
      {
            xp+=tx;
            yp+=ty;
      }

      void rotate (float angle)
      {
            int tempx=x1;
            x1=tempx*cos(angle)-y1*sin(angle);
            y1=tempx*sin(angle)+y1*cos(angle);
            tempx=x2;
            x2=tempx*cos(angle)-y2*sin(angle);
            y2=tempx*sin(angle)+y2*cos(angle);
            tempx=x3;
            x3=tempx*cos(angle)-y3*sin(angle);
            y3=tempx*sin(angle)+y3*cos(angle);
            tempx=x4;
            x4=tempx*cos(angle)-y4*sin(angle);
            y4=tempx*sin(angle)+y4*cos(angle);
      }

      void scale(int sx, int sy)
      {
            x1=x1*sx;
            x2=x2*sx;
            x3=x3*sx;
            x4=x4*sx;
            y1=y1*sy;
            y2=y2*sy;
            y3=y3*sy;
            y4=y4*sy;
            legLength=legLength*sy;
      }

      void draw()
      {
            int xc=this->xc+xp;
            int yc=this->yc+yp;
            line (xc+x1, yc+y1, xc+x2, yc+y2);
            line (xc+x2, yc+y2, xc+x3, yc+y3);
            line (xc+x3, yc+y3, xc+x4, yc+y4);
            line (xc+x4, yc+y4, xc+x1, yc+y1);
            line (xc+x1, yc+y1, xc+x1, yc+y1+legLength);
            line (xc+x2, yc+y2, xc+x2, yc+y2+legLength);
            line (xc+x3, yc+y3, xc+x3, yc+y3+legLength);
            line (xc+x4, yc+y4, xc+x4, yc+y4+legLength);
      }
};


void main()
{
      clrscr();
      int gdriver = DETECT, gmode, errorcode;
      initgraph(&gdriver, &gmode, "c:\\tc\\bgi");
      Table table;
      table.draw();
      setcolor(CYAN);
      table.translate(15, 25);
      table.draw();
      table.translate(50, 0);
      table.scale(3,2);
      table.draw();
      table.translate(-100, 0);
      table.rotate(3.14/4);
      table.draw();
      getch();
      closegraph();
}
```