**Aim:**  To implement Character Generation.

**Objective:**

Identify the different Methods for Character Generation and generate the character using Stroke

**Theory:**

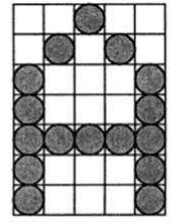
**Bit map method –**

Bitmap method is a called dot-matrix method as the name suggests this method use array of bits for generating a character. These dots are the points for array whose size is fixed.

∙ In bit matrix method when the dots are stored in the form of array the value 1 in array represent the characters i.e. where the dots appear we represent that position with numerical value 1 and the value where dots are not present is represented by 0 in array.

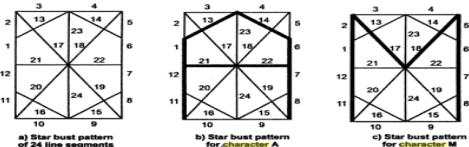
∙ It is also called dot matrix because in this method characters are represented by an array of dots in the matrix form. It is a two-dimensional array having columns and rows.

A 5x7 array is commonly used to represent characters. However, 7x9 and 9x13 arrays are also used. Higher resolution devices such as inkjet printer or laser printer may use character arrays that are over 100x100.



**Starburst method –**

In this method a fix pattern of line segments is used to generate characters. Out of these 24-line segments, segments required to display for particular character are highlighted. This method of character generation is called starburst method because of its characteristic appearance. The starburst patterns for characters A and M. the patterns for particular characters are stored in the form of 24 bit code, each bit representing one line segment. The bit is set to one to highlight the line segment; otherwise, it is set to zero. For example, 24-bit code for Character A is 0011 0000 0011 1100 1110 0001 and for character M is 0000 0011 0000 1100 1111 0011.



**Program:**

**#include<stdio.h>**

**#include<conio.h>**

**#include<graphics.h>**

**void main()**

**{**

**int gd=DETECT , gm;**

**int i, j, k;**

**int a[10][10] =**

**{**

**{1,1,1,1,1,1,1,1,1,1},**

**{1,1,1,1,1,1,1,1,1,1},**

**{0,0,0,0,1,1,0,0,0,0},**

**{0,0,0,0,1,1,0,0,0,0},**

**{0,0,0,0,1,1,0,0,0,0},**

**{0,1,0,0,1,1,0,0,0,0},**

**{0,0,0,0,1,1,0,0,0,0},**

**{0,0,0,0,1,1,0,0,0,0},**

**{1,1,1,1,1,1,0,0,0,0},**

**{1,1,1,1,1,1,0,0,0,0}**

**};**

**int b[10][10] = {**

**{0,0,0,1,1,1,1,0,0,0},**

**{0,0,1,1,1,1,1,1,0,0},**

**{0,1,1,0,0,0,0,1,1,0},**

**{0,1,1,0,0,0,0,1,1,0},**

**{0,1,1,0,0,0,0,1,1,0},**

**{0,1,1,1,1,1,1,1,1,0},**

**{0,1,1,1,1,1,1,1,1,0},**

**{0,1,1,0,0,0,0,1,1,0},**

**{0,1,1,0,0,0,0,1,1,0},**

**{1,1,1,1,0,0,1,1,1,1}**

**};**

**int c[10][10] = {**

**{1,1,1,1,1,1,1,1,1,1},**

**{1,1,1,1,1,1,1,1,1,1},**

**{0,0,0,0,1,1,0,0,0,0},**

**{0,0,0,0,1,1,0,0,0,0},**

**{0,0,0,0,1,1,0,0,0,0},**

**{0,0,0,0,1,1,0,0,0,0},**

**{0,0,0,0,1,1,0,0,0,0},**

**{0,0,0,0,1,1,0,0,0,0},**

**{1,1,1,1,1,1,1,1,1,1},**

**{1,1,1,1,1,1,1,1,1,1}**

**};**

**int d[10][10] = {**

**{1,1,0,0,0,0,0,0,1,1},**

**{1,1,1,0,0,0,0,0,1,1},**

**{1,1,1,1,0,0,0,0,1,1},**

**{1,1,0,1,1,0,0,0,1,1},**

**{1,1,0,0,1,1,0,0,1,1},**

**{1,1,0,0,0,1,1,0,1,1},**

**{1,1,0,0,0,0,1,1,1,1},**

**{1,1,0,0,0,0,0,1,1,1},**

**{1,1,0,0,0,0,0,1,1,1},**

**{1,1,0,0,0,0,0,0,1,1}**

**};**

**int e[10][10] = {**

**{0,0,0,1,1,1,1,0,0,0},**

**{0,0,1,1,1,1,1,1,0,0},**

**{0,1,1,0,0,0,0,1,1,0},**

**{0,1,1,0,0,0,0,1,1,0},**

**{0,1,1,0,0,0,0,1,1,0},**

**{0,1,1,1,1,1,1,1,1,0},**

**{0,1,1,1,1,1,1,1,1,0},**

**{0,1,1,0,0,0,0,1,1,0},**

**{0,1,1,0,0,0,0,1,1,0},**

**{1,1,1,1,0,0,1,1,1,1}**

**};**

**int f[10][10] = {**

**{1,1,0,0,0,0,0,0,1,1},**

**{1,1,1,0,0,0,0,0,1,1},**

**{1,1,1,0,0,0,0,1,1,1},**

**{1,1,0,1,0,0,1,1,1,1},**

**{1,1,0,1,1,1,1,0,1,1},**

**{1,1,0,0,1,1,0,0,1,1},**

**{1,1,0,0,0,0,0,0,1,1},**

**{1,1,0,0,0,0,0,0,1,1},**

**{1,1,0,0,0,0,0,0,1,1},**

**{1,1,0,0,0,0,0,0,1,1}**

**}**

**};**

**initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");**

**for(k=0;k<3;k++)**

**{**

**for(i=0;i<10;i++)**

**{**

**for(j=0;j<10;j++)**

**{**

**if(a[i][j]==1){**

**putpixel(200+j, 200+i,**

**15);**

**}**

**}**

**}**

**for(i=0;i<10;i++)**

**{**

**for(j=0;j<10;j++)**

**{**

**if(b[i][j]==1){**

**putpixel(220+j, 200+i,**

**15);**

**}**

**}**

**}**

**for(i=0;i<10;i++)**

**{**

**for(j=0;j<10;j++)**

**{**

**if(c[i][j]==1){**

**putpixel(240+j, 200+i,**

**15);**

**}**

**}**

**}**

**for(i=0;i<10;i++)**

**{**

**for(j=0;j<10;j++)**

**{**

**if(d[i][j]==1){**

**putpixel(260+j, 200+i,**

**15);**

**}**

**}**

**}**

**for(i=0;i<10;i++)**

**{**

**for(j=0;j<10;j++)**

**{**

**if(e[i][j]==1){**

**putpixel(280+j, 200+i,**

**15);**

**}**

**}**

**}**

**for(i=0;i<10;i++)**

**{**

**for(j=0;j<10;j++)**

**{**

**if(f[i][j]==1){**

**putpixel(300+j, 200+i,**

**15);**

**}**

**}**

**}**

**}**

**getch();**

**closegraph();**

**}**

**Output -**



**Conclusion:**  Comment on

1. different methods-Rasterization is common for real-time graphics. Ray tracing offers photorealism but demands computational power. Vector graphics suit 2D illustrations and logos. Voxel rendering handles detailed 3D scenes. Shader-based rendering manipulates object appearance in real-time. Procedural rendering generates assets algorithmically. The painter's algorithm sorts objects by distance. Cell shading creates a cartoon-like 3D look. Volume rendering visualizes 3D data sets. Path tracing provides precise global illumination. Choice depends on project needs and available resources.
2. advantage of stroke method-The stroke method in computer graphics offers advantages such as clarity, customizable stylization, scalability, editing flexibility, efficiency, consistency, versatility, layering, animation, and the ability to simulate realism. It's a valuable tool for emphasizing shapes, maintaining visual consistency, and achieving various artistic effects. However, its suitability depends on the specific project and goals.
3. one limitation-One limitation of the stroke method in computer graphics is that it may not be well-suited for rendering complex shading and lighting effects, making it less ideal for photorealistic 3D graphics.