7.3 Constructions from Mapping Text

Numeric data has values that can easily be interpreted visually by size, height, or intensity. Can the same be applied to text or any other nonnumeric data?

One method to handle such data is to substitute a symbol for a data element, this is called mapping.

A simple example is to map a alphabetic or numeric character to a image or drawing. In our examples we will create a series of drawings of individual letters and numbers.

Start a new drawing and set the units to architectural; turn off all snaps and dynamic input.

Start a new AutoLISP file in the editor: CH07C.LSP and add functions *dtr* and *rtd*.

One method to represent characters as drawings, is to create a series of BLOCKs, each representing an alphabetic character. We can built these to mimic an old fashion dot-matrix printer font. Each character is in an 7 x 7 matrix with circles as the dots in the font. Figure 7.33 displays the characters A through F that were made into BLOCKs. The block names for these are set to "blockA" through "blockF". The dimensions of the block are set to 1 inch by 1 inch so they can be scaled to any size when inserted. The insertion point is the bottom-left corner of the rectangular boundary. The boundary is not part of the block.

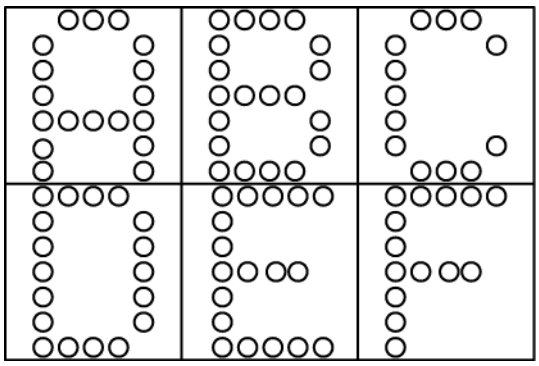

Figure 7.33: Dot-matrix characters

Create any number of these character blocks you wish, either consisting of circles or any other shape; square, rotated square, or an n-sided polygon.

For the first function, add function PROG01 to enter a character and have the corresponding BLOCK name simply identified.

The first step is to create a list to map a character to the its BLOCK name. In this case the list is:

```
(setq slist (list
   (list "A" "blockA")
   (list "B" "blockB")
   (list "C" "blockC")
   (list "D" "blockD")
   (list "E" "blockE")))
```

The list consists of five lists each containing a character and the BLOCK name associated with it.

The function PROG01 accepts a single character, coverts it to uppercase, and then checks to see if that character has an entry in the block list. The *assoc* function checks the list for a match, as follows:

```
(setq schar (strcase schar))
(setq mapto (assoc schar slist))
```

If found, the block name is extracted from the list with the *nth* function and it is displayed, otherwise *nil* is displayed.

Completed function PROG01:

```
;-----------------------------------
(defun prog01 ()
 (prompt "\nPROG01")
 (setq slist (list
   (list "A" "blockA")
   (list "B" "blockB")
   (list "C" "blockC")
   (list "D" "blockD")
   (list "E" "blockD")))
 (repeat 10
  (setq schar
    (getstring
    "\nEnter a single character:"))
  (setq schar (strcase schar))
  (setq mapto (assoc schar slist))
  (princ "\n")  (princ schar)
  (princ " = ") (princ mapto)
  (if (/= mapto nil) (progn
   (setq blockname (nth 1 mapto))
   (princ " = ") (princ blockname)
  ))
 )
(princ)
)
;-----------------------------------
```

Execute function PROG01 from the command line for characters that are in the list and that are not, for example the results could be:

```
Command: (prog01)
PROG01
Enter a single character:a
A = (A blockA) = blockA
Enter a single character:c
C = (C blockC) = blockC
Enter a single character:x
X = nil
Enter a single character:e
E = (E blockE) = blockE
```

Note that only uppercase characters were considered when the blocks were created.

Next, accept a string and display the block name for each character in the string.

Add function PROG01a. Using a copy of PROG01, change the character input to a string, change the repeat to consider the number of characters in the string, and set a pointer to select each character.

Each character is extracted from the string using:

```
(setq schar
 (substr instring inpt 1))
```

Where the variable *ipnt* is incremented from the first to the last character position in the string.

Completed function PROG01a:

```
;-----------------------------------
(defun prog01a ()
```

```
 (prompt "\nPROG01a")
 (setq slist (list
   (list "A" "blockA")
   (list "B" "blockB")
   (list "C" "blockC")
   (list "D" "blockD")
   (list "E" "blockE")
   (list "M" "blockM")
   (list "S" "blockS")))
 (setq instring
   (getstring
   "\nEnter a string: "))
 (setq inpt 1)
 (repeat (strlen instring)
  (setq schar
    (substr instring inpt 1))
  (setq schar (strcase schar))
  (setq mapto (assoc schar slist))
  (princ "\n")  (princ schar)
  (princ " = ") (princ mapto)
  (if (/= mapto nil) (progn
   (setq blockname (nth 1 mapto))
   (princ " = ") (princ blockname)
  ))
  (setq inpt (+ inpt 1))
 )
(princ)
)
;-----------------------------------
```

Review PROG01a structure.

Execute PROG01a from the command line, for example:

```
Command: (prog01a)
PROG01a
Enter a string: "less is more"
L = nil
E = (E blockE) = blockE
S = (S blockS) = blockS
S = (S blockS) = blockS
  = nil
I = nil
S = (S blockS) = blockS
  = nil
M = (M blockM) = blockM
O = nil
R = nil
E = (E blockE) = blockE
```

Note that a few more character blocks were defined, the string is entered with double quotes, and that the space character results in a *nil* block name, as do characters not defined in the list.

Execute PROG01a a few times with your own strings.

Now that we can parse a string into its block names, we can display the blocks the characters represent.

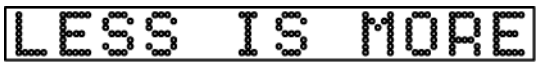

Figure 7.34: PROG01b, text mapped to dot-matrix characters

Add function PROG01b. Using a copy of PR0G01a, once we have the block name, use the INSERT command to display it in the drawing. Add input for the x and y character scale. Also add a rectangular boundary around the string.

The block is inserted and scaled using:

```
; insert block with scale
(command ".INSERT" blockname
  (list xpt ypt 0) xside yside "")
```

Execute function PROG02c for a number of different strings. Add other character blocks as needed.

Sample script file for PROG01b:

```
;-----------------------------------
(prog01b)
;Enter string:
"less is more"
;Enter X side:
10.0
;Enter Y side:
10.0
;-----------------------------------
```

Completed function PROG01b:

```
;-----------------------------------
(defun prog01b ()
 (prompt "\nPROG01b")
 (setq slist (list
   (list "A" "blockA")
   (list "B" "blockB")
   (list "C" "blockC")
   (list "D" "blockD")
   (list "E" "blockE")
   (list "I" "blockI")
   (list "L" "blockL")
   (list "M" "blockM")
   (list "O" "blockO")
   (list "R" "blockR")
   (list "S" "blockS")))
 (command ".ERASE" "all" "")
 (setq instring
   (getstring "\nEnter a string: "))
 (setq xside
   (getdist "\nEnter X side: "))
 (setq yside
   (getdist "\nEnter Y side: "))
 (setq pnt0 (list 0 0 0))
 (setq xpt (nth 0 pnt0))
 (setq ypt (nth 1 pnt0))
 (setq inpt 1)
 (repeat (strlen instring)
  (setq schar
    (substr instring inpt 1))
  (setq schar (strcase schar))
  (setq mapto (assoc schar slist))
  (if (/= mapto nil) (progn
   (setq blockname (nth 1 mapto))
   ; insert block with scale
   (command ".INSERT" blockname
     (list xpt ypt 0) xside yside "")
   (command ".ZOOM" "e")
  ))
  (setq inpt (+ inpt 1))
  (setq xpt (+ xpt xside))
 )
; draw edge
(command ".RECTANGLE"
  (list (- 0.0 (* 0.1 xside))
  (- ypt (* 0.1 yside)) 0)
  (list (+ xpt (* 0.1 xside))
  (+ ypt (* 1.1 yside)) 0))
(command ".ZOOM" "e")
(princ)
)
;-----------------------------------
```

Review PROG01b structure; how the insertion point for each block is computed and what happens when spaces or undefined character are in the string.

Note that the variable *pnt0* is the start point, it could be entered as input, and the entire drawing is erased before we start.

Also note that the boundary rectangle is set to be 10% larger than the string size.

The results from PROG01b could be used as a line drawing for etching or to cut perforations using a laser cutter.

Consider expanding PROG01b to handle multiple lines of text, entered or read from a file.

To highlight the dots in each character, add a solid fill to each one.

Figure 7.35: PROG01c, text mapped to dot-matrix fill characters

Add function PROG01c. Using a copy of PR0G01b, EXPLODE each of the blocks after they are inserted and then HATCH all the circles once the string is drawn.

Add the EXPLODE command after insertion:

```
; insert block with scale
(command ".INSERT" blockname
  (list xpt ypt 0) xside yside "")
(command ".ZOOM" "e")
(command ".EXPLODE" "last")
```

Before drawing the rectangular edge, hatch all the circles:

```
; fill the circles
(command ".FILL" "on")
(command ".HATCH" "solid" "all" "")
```

Use the same script file as for PROG01b.

Completed function PROG01c:

```
;-----------------------------------
(defun prog01c ()
 (prompt "\nPROG01c")
 (setq slist (list
   (list "A" "blockA")
   (list "B" "blockB")
   (list "C" "blockC")
   (list "D" "blockD")
   (list "E" "blockE")
   (list "I" "blockI")
   (list "L" "blockL")
   (list "M" "blockM")
   (list "O" "blockO")
   (list "R" "blockR")
   (list "S" "blockS") ))
 (command ".ERASE" "all" "")
 (setq instring
   (getstring
   "\nEnter a string: "))
 (setq xside
   (getdist "\nEnter X side: "))
 (setq yside
   (getdist "\nEnter Y side: "))
 (setq pt0 (list 0 0 0))
 (setq xpt (nth 0 pnt0)
 (setq ypt (nth 1 pnt0)
 (setq inpt 1)
 (repeat (strlen instring)
  (setq schar
    (substr instring inpt 1))
  (setq schar (strcase schar))
  (setq mapto (assoc schar slist))
  (if (/= mapto nil) (progn
   (setq blockname (nth 1 mapto))
   ; insert block with scale
   (command ".INSERT" blockname
     (list xpt ypt 0) xside yside "")
   (command ".ZOOM" "e")
   (command ".EXPLODE" "last")
  ))
  (setq inpt (+ inpt 1))
  (setq xpt (+ xpt xside))
 )
; fill the circles
(command ".FILL" "on")
(command ".HATCH" "solid" "all" "")
; draw edge
(command ".RECTANGLE"
  (list (- 0.0 (* 0.1 xside))
  (- ypt (* 0.1 yside)) 0)
  (list (+ xpt (* 0.1 xside))
  (+ ypt (* 1.1 yside)) 0))
(command ".ZOOM" "e")
(princ)
)
;-----------------------------------
```

A three-dimensional version of the characters can be constructed by extruding all the dots.

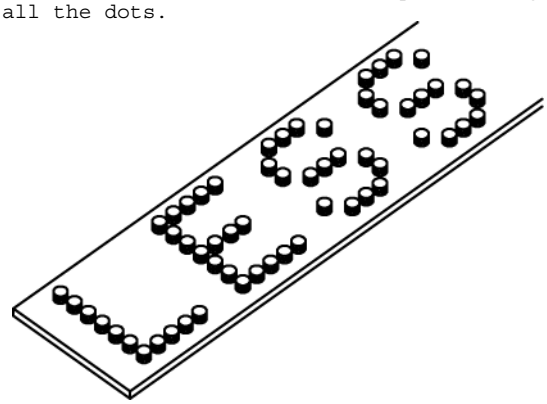

Figure 7.36: PROG01d, text mapped to dot-matrix 3D characters

Add function PROG01d. Using a copy of PR0G01c, extrude all of the circles to an entered height, also extrude the rectangular boundary. Union all the circles and boundary together.

Add input for the extrusion height:

```
(setq zheight
   (getdist "\nEnter Z height: "))
```

Replace the FILL and HATCH commands with the following extrusion command:

```
; extrude the circles
(command ".EXTRUDE"
  "all" "" zheight)
; versions prior to 2007 require
; the taper parameter
;(command ".EXTRUDE"
; "all" "" zheight "")
```

Add the same extrusion command for the rectangular boundary and then union everything together. The height of the boundary is set to 1/2 of the circle height.

Sample script file for PROG01d:

```
;-----------------------------------
(prog01d)
;Enter string:
"less is more"
;Enter X side:
10.0
;Enter Y side:
10.0
;Enter Z height
1.5
;-----------------------------------
```

Completed function PROG01d:

```
;-----------------------------------
(defun prog01d ()
 (prompt "\nPROG01d")
 (setq slist (list
   (list "A" "blockA")
   (list "B" "blockB")
   (list "C" "blockC")
   (list "D" "blockD")
   (list "E" "blockE")
   (list "I" "blockI")
   (list "L" "blockL")
   (list "M" "blockM")
   (list "O" "blockO")
   (list "R" "blockR")
   (list "S" "blockS") ))
 (command ".ERASE" "all" "")
 (setq instring
   (getstring
   "\nEnter a string: "))
 (setq xside
   (getdist "\nEnter X side: "))
 (setq yside
   (getdist "\nEnter Y side: "))
 (setq zheight
   (getdist "\nEnter Z height: "))
 (setq pt0 (list 0 0 0))
 (setq xpt (nth 0 pnt0)
 (setq ypt (nth 1 pnt0)
 (setq inpt 1)
 (repeat (strlen instring)
  (setq schar
    (substr instring inpt 1))
  (setq schar (strcase schar))
  (setq mapto (assoc schar slist))
  (if (/= mapto nil) (progn
   (setq blockname (nth 1 mapto))
   ; insert block with scale
   (command ".INSERT" blockname
     (list xpt ypt 0) xside yside "")
   (command ".ZOOM" "e")
   (command ".EXPLODE" "last")
  ))
  (setq inpt (+ inpt 1))
  (setq xpt (+ xpt xside))
 )
; extrude the circles
(command ".EXTRUDE"
  "all" "" zheight)
; versions prior to 2007 require
; the taper parameter
;(command ".EXTRUDE"
; "all" "" zheight "")
; draw edge
(command ".RECTANGLE"
  (list (- 0.0 (* 0.1 xside))
  (- ypt (* 0.1 yside)) 0)
  (list (+ xpt (* 0.1 xside))
  (+ ypt (* 1.1 yside)) 0))
(command ".ZOOM" "e")
; extrude the edge
(command ".EXTRUDE"
  "last" "" (/ zheight 2.0))
; versions prior to 2007 require
; the taper parameter
;(command ".EXTRUDE"
; "last" "" (/ zheight 2.0) "")
; union
(command ".UNION" "all" "")
(command ".ZOOM" "e")
(princ)
)
;-----------------------------------
```

The extrusion of the dots in each character can also be used to create perforations or indentations in the boundary surface.

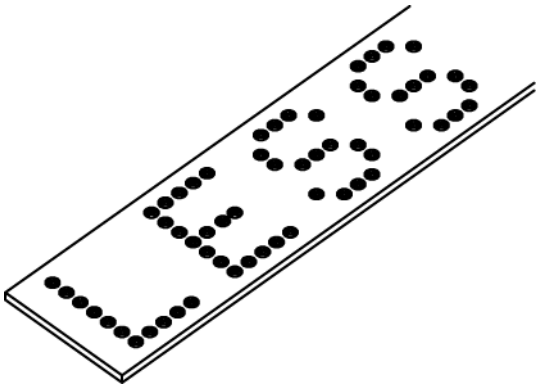

Figure 7.37: PROG01e, text mapped to dot-matrix 3D perforations

Add function PROG01e. Using a copy of PR0G01d, subtract the extruded circles from the boundary surface.

The dots are extruded, unioned and placed into a selection list. The boundary edge is also extruded and placed into a selection list. The selection lists are then subtracted from each other.

The relationship of heights between the dots and the boundary surface will determine if perforations or indentations are created.

Use the same script file as for PROG01d.

Completed function PROG01e:

```
;-----------------------------------
(defun prog01e ()
 (prompt "\nPROG01e")
 (setq slist (list
   (list "A" "blockA")
   (list "B" "blockB")
   (list "C" "blockC")
   (list "D" "blockD")
   (list "E" "blockE")
   (list "I" "blockI")
   (list "L" "blockL")
   (list "M" "blockM")
   (list "O" "blockO")
   (list "R" "blockR")
   (list "S" "blockS") ))
 (command ".ERASE" "all" "")
 (setq instring
   (getstring
   "\nEnter a string: "))
 (setq xside
   (getdist "\nEnter X side: "))
 (setq yside
   (getdist "\nEnter Y side: "))
 (setq zheight
   (getdist "\nEnter Z height: "))
 (setq pnt0 (list 0 0 0))
 (setq xpt (nth 0 pnt0))
 (setq ypt (nth 1 pnt0))
 (setq inpt 1)
 (repeat (strlen instring)
  (setq schar
    (substr instring inpt 1))
  (setq schar (strcase schar))
  (setq mapto (assoc schar slist))
  (if (/= mapto nil) (progn
   (setq blockname (nth 1 mapto))
   ; insert block with scale
   (command ".INSERT" blockname
     (list xpt ypt 0) xside yside "")
   (command ".ZOOM" "e")
   (command ".EXPLODE" "last")
  ))
  (setq inpt (+ inpt 1))
```

```
  (setq xpt (+ xpt xside))
 )
; extrude the circles
(command ".EXTRUDE"
  "all" "" zheight)
(command ".UNION" "all" "")
; versions prior to 2007 require
; the taper parameter
;(command ".EXTRUDE"
; "all" "" zheight "")
; select union
(setq obj1 (ssget "l"))
; draw edge
(command ".RECTANGLE"
  (list (- 0.0 (* 0.1 xside))
  (- ypt (* 0.1 yside)) 0)
  (list (+ xpt (* 0.1 xside))
  (+ ypt (* 1.1 yside)) 0))
(command ".ZOOM" "e")
; extrude the edge
(command ".EXTRUDE"
  "last" "" (/ zheight 2.0))
; versions prior to 2007 require
; the taper parameter
;(command ".EXTRUDE"
; "last" "" (/ zheight 2.0) "")
; select boundary
(setq obj2 (ssget "l"))
; subtract
(command ".SUBTRACT"
  obj2 "" obj1 "")
(command ".ZOOM" "e")
(princ)
)
;-----------------------------------
```

In the dot-matrix font example we defined the corresponding blocks to be an exact visual representation of each character. Mapping can involve visual representations that are not exactly the same as each character visually but are an encoded version. The Braille system is one such encoding, Morse Code is another.

The Braille alphabet is made up of a six element cell of raised dots. Figure 38 displays the cell and identifying numbers with a sample of the characters A through F. The Braille system supports uppercase characters A through Z, the numbers 0 through 9, and some special characters.

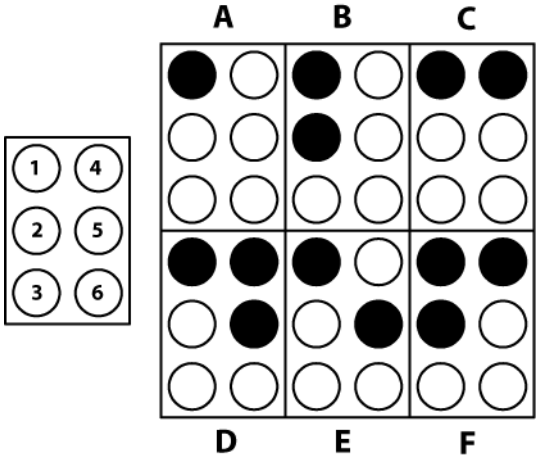

Figure 7.38: Braille system

Research the background of the Braille system and how the rest of the alphabet, numbers, and special characters are defined.

A series of blocks could be created for each character as we had previously. Since there are only six possible dots all in fixed locations and each one is numbered, we can easily create a mapping that includes which dots are raised.

For example, the letter A would be (1), B would be (1 2), C would be (1 4).

Add function PROG02. Using a copy of PROG01a, replace the block mapping list with a list of dot locations.

Completed function PROG02:

```
;-----------------------------------
(defun prog02 ()
 (prompt "\nPROG02")
 (setq slist (list
   (list "A" (list 1))
   (list "B" (list 1 2))
   (list "C" (list 1 4))
   (list "D" (list 1 4 5))
   (list "E" (list 1 5))
   (list "I" (list 2 4))
   (list "L" (list 1 2 3))
   (list "M" (list 1 3 4))
   (list "O" (list 1 3 5))
   (list "R" (list 1 2 3 5))
   (list "S" (list 2 3 4))))
 (setq instring
   (getstring "\nEnter a string: "))
 (setq inpt 1)
 (repeat (strlen instring)
  (setq schar
    (substr instring inpt 1))
  (setq schar (strcase schar))
  (setq mapto (assoc schar slist))
  (princ "\n")  (princ schar)
  (princ " = ") (princ mapto)
  (if (/= mapto nil) (progn
   (setq cell (nth 1 mapto))
   (princ " = ") (princ cell)
  ))
  (setq inpt (+ inpt 1))
 )
(princ)
)
;-----------------------------------
```

Execute function PROG02 from the command line, for example:

```
Command: (prog02)
PROG02
Enter a string: "less is more"
L = (L (1 2 3)) = (1 2 3)
E = (E (1 5)) = (1 5)
S = (S (2 3 4)) = (2 3 4)
S = (S (2 3 4)) = (2 3 4)
  = nil
I = (I (2 4)) = (2 4)
S = (S (2 3 4)) = (2 3 4)
  = nil
M = (M (1 3 4)) = (1 3 4)
O = (O (1 3 5)) = (1 3 5)
R = (R (1 2 3 5)) = (1 2 3 5)
E = (E (1 5)) = (1 5)
```

The Braille system has specifications for the diameter of the raised dots and their spacing. For our design purposes, we will specify a X dimension. Across this dimension we need to place two dots and three spacers. If we use the radius of a dot as our base measurement, we can compute the radius and spacing.

The X dimension would be:

1r + 2r + 1r + 2r + 1r or 7 radius

The radius would then be computed as:

X side / 7

After computing the radius, the Y dimension would be:

1r + 2r + 1r + 2r + 1r + 2r + 1r

Given this relationship, all of the dot offset locations can be computed from the bottom-left corner of the Braille cell.

Given the number, 1 through 6, location within the Braille cell, the XY offset location could be easily computed.

Add function PROG02a to compute the XY offset given an X dimension and the dot number.

Use a series of *if* functions to check what the offsets should be. This approach consists of checking which row and which column the dot is located.

Completed function PROG02b:

```
;-----------------------------------
(defun prog02a (xdim ndot
  / xoff yoff drad)
 ; offsets
 (setq xoff 0.0)
 (setq yoff 0.0)
 ; radius
 (setq drad (/ xdim 7.0))
 ; x offset
 (if (or (= ndot 1) (= ndot 2)
   (= ndot 3))
  (setq xoff (* drad 2.0))
 )
 (if (or (= ndot 4) (= ndot 5)
   (= ndot 6))
  (setq xoff (* drad 5.0))
 )
 ; y offset
 (if (or (= ndot 1) (= ndot 4))
  (setq yoff (* drad 8.0)))
 (if (or (= ndot 2) (= ndot 5))
  (setq yoff (* drad 5.0)))
 (if (or (= ndot 3) (= ndot 6))
  (setq yoff (* drad 2.0)))
; return list
(list xoff yoff)
)
;-----------------------------------
```

Execute function PROG02a from the command line, for example:

```
Command: (prog02a 14 4)
(10.0 16.0)

Command: (prog02a 14 1)
(4.0 16.0)

Command: (prog02a 14 3)
(4.0 4.0)

Command: (prog02a 14 5)
(10.0 10.0)

Command: (prog02a 14 0)
(0.0 0.0)
```

Another approach would be to continue the mapping concept to the computation of the dot locations.

For example, dot 1 is located at an XY offset:

(* radius 2.0) (* radius 8.0)

Dot 5 is located at XY offset:

(* radius 5.0) (* radius 5.0)

Add function PROG02b to compute the XY offset of each dot from a list of offset computations as strings.

Completed function PROG02b:

```
;-----------------------------------
(defun prog02b (drad ndot
  / xoff yoff clist)
 ; offsets
 (setq xoff 0.0)
 (setq yoff 0.0)
 ; cell locations
 (setq clist (list
  (list "0" "0")
  (list
    "(* drad 2.0)" "(* drad 8.0)")
  (list
    "(* drad 2.0)" "(* drad 5.0)")
  (list
    "(* drad 2.0)" "(* drad 2.0)")
  (list
    "(* drad 5.0)" "(* drad 8.0)")
  (list
    "(* drad 5.0)" "(* drad 5.0)")
  (list
    "(* drad 5.0)" "(* drad 2.0)")
 ))
 ; get offsets
 (setq xoff (eval (read
   (nth 0 (nth ndot clist)))))
 (setq yoff (eval (read
   (nth 1 (nth ndot clist)))))
; return list
(list xoff yoff)
)
;-----------------------------------
```

Review how the *read* and *eval* functions are used to convert a string into an expression that can be evaluated. Function PROG02b returns a list containing a computed x and y offset.

Execute function PROG02b from the command line for the same dot locations as for PROG02a:

```
Command: (prog02b 2 1)
(4.0 16.0)

Command: (prog02b 2 4)
(10.0 16.0)

Command: (prog02b 2 3)
(4.0 4.0)

Command: (prog02b 2 5)
(10.0 10.0)

Command: (prog02b 2 0)
(0 0)
```

Now that we have a mapping method for the dots, we can display a Braille cell for each character in a string.

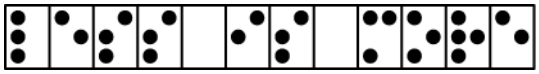

Figure 7.39: PROG02c, text mapped to Braille dots filled

Add function PROG02c. Using a copy of PROG02, following the same procedure as in function PROG01c, instead of inserting a block, us the PROG02b function to get the dot location. A second repeat is needed to

generate possible multiple dots based on the location list.

The computation of the dot locations is determined by the cell list as follows:

```
(setq cell (nth 1 mapto))
; get locations
(setq icell 0)
(repeat (length cell)
 (setq ndot (nth icell cell))
 (setq xyoff (prog02b drad ndot))
 (setq pnt1 (list
   (+ xpt (nth 0 xyoff))
   (+ ypt (nth 1 xyoff)) 0))
 (command ".CIRCLE" pnt1 drad)
 (command ".ZOOM" "e")
 (command ".FILL" "on")
 (command ".HATCH"
   "solid" "last" "")
 (setq icell (+ icell 1))
)
```

Once the all the dots are drawn, a cell boundary can be added:

```
; draw cell boundary
(command ".RECTANGLE"
  (list xpt ypt 0)
  (list (+ xpt xside)
  (+ ypt yside) 0))
(command ".ZOOM" "e")
```

Execute function PROG02c for a number of different strings. Add other character dot location lists as needed.

Sample script file for PROG02c:

```
;-----------------------------------
(prog02c)
;Enter string:
"less is more"
;Enter X side:
10.0
;-----------------------------------
```

Completed function PROG02c:

```
;-----------------------------------
(defun prog02c ()
 (prompt "\nPROG02c")
 (setq slist (list
   (list "A" (list 1))
   (list "B" (list 1 2))
   (list "C" (list 1 4))
   (list "D" (list 1 4 5))
   (list "E" (list 1 5))
   (list "I" (list 2 4))
   (list "L" (list 1 2 3))
   (list "M" (list 1 3 4))
   (list "O" (list 1 3 5))
   (list "R" (list 1 2 3 5))
   (list "S" (list 2 3 4))))
 (command ".ERASE" "all" "")
 (setq instring
   (getstring "\nEnter a string: "))
 (setq xside
   (getdist "\nEnter X side: "))
 ; compute radius and Y side
 (setq drad (/ xside 7.0))
 (setq yside (* drad 10.0))
 ; start point
 (setq pnt0 (list 0 0 0))
 (setq xpt (nth 0 pnt0))
 (setq ypt (nth 1 pnt0))
 (setq inpt 1)
 (repeat (strlen instring)
  (setq schar
    (substr instring inpt 1))
  (setq schar (strcase schar))
  (setq mapto (assoc schar slist))
  (if (/= mapto nil) (progn
   (setq cell (nth 1 mapto))
   ; get locations
   (setq icell 0)
   (repeat (length cell)
    (setq ndot (nth icell cell))
    (setq xyoff (prog02b drad ndot))
    (setq pnt1 (list
      (+ xpt (nth 0 xyoff))
      (+ ypt (nth 1 xyoff)) 0))
    (command ".CIRCLE" pnt1 drad)
    (command ".ZOOM" "e")
    (command ".FILL" "on")
    (command ".HATCH"
      "solid" "last" "")
    (setq icell (+ icell 1))
   )
  ))
  ; draw cell boundary
  (command ".RECTANGLE"
    (list xpt ypt 0)
    (list (+ xpt xside)
    (+ ypt yside) 0))
  (command ".ZOOM" "e")
  (setq inpt (+ inpt 1))
  (setq xpt (+ xpt xside))
 )
(command ".ZOOM" "e")
(princ)
)
;-----------------------------------
```

A three-dimensional version of the filled dots can be constructed as raised dots, modeled as a 1/2 portion of a sphere.

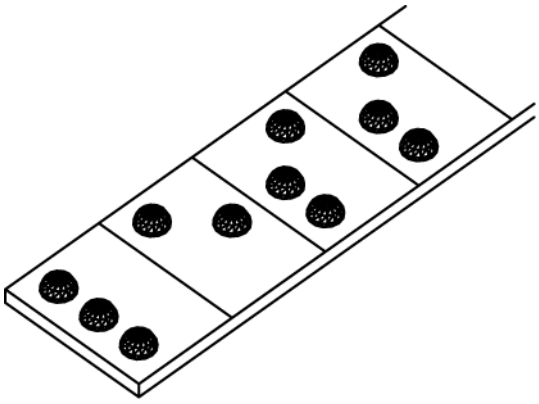

Figure 7.40: PROG02d, text mapped to Braille 3D raised dots

Add function PROG02d. Using a copy of PROG02c, replace the circle with a sphere and remove the FILL and HATCH commands. Once all the spheres are placed, created a base and UNION all the elements together.

The circle is replaced with:

```
(command ".SPHERE" pnt1 drad)
```

To create the base and union all the elements use:

```
; draw edge
(command ".RECTANGLE" pnt0
  (list (+ xpt xside)
  (+ ypt yside) 0))
(command ".ZOOM" "e")
; extrude the edge
(command ".EXTRUDE"
  "last" "" (* drad -1))
; versions prior to 2007 require
; the taper parameter
;(command ".EXTRUDE"
; "last" "" (* drad -1) "")
; union all
(command ".UNION" "all" "")
```

```
(command ".ZOOM" "e")
(princ)
```

Use the same script file as for PROG02c.

Completed function PROG02d:

```
;-----------------------------------
(defun prog02d ()
 (prompt "\nPROG02d")
 (setq slist (list
   (list "A" (list 1))
   (list "B" (list 1 2))
   (list "C" (list 1 4))
   (list "D" (list 1 4 5))
   (list "E" (list 1 5))
   (list "I" (list 2 4))
   (list "L" (list 1 2 3))
   (list "M" (list 1 3 4))
   (list "O" (list 1 3 5))
   (list "R" (list 1 2 3 5))
   (list "S" (list 2 3 4))))
 (command ".ERASE" "all" "")
 (setq instring
   (getstring "\nEnter a string: "))
 (setq xside
   (getdist "\nEnter X side: "))
 ; compute radius and Y side
 (setq drad (/ xside 7.0))
 (setq yside (* drad 10.0))
 ; start point
 (setq pnt0 (list 0 0 0))
 (setq xpt (nth 0 pnt0))
 (setq ypt (nth 1 pnt0))
 (setq inpt 1)
 (repeat (strlen instring)
  (setq schar
    (substr instring inpt 1))
  (setq schar (strcase schar))
  (setq mapto (assoc schar slist))
  (if (/= mapto nil) (progn
   (setq cell (nth 1 mapto))
   ; get locations
   (setq icell 0)
   (repeat (length cell)
    (setq ndot (nth icell cell))
    (setq xyoff (prog02b drad ndot))
    (setq pnt1 (list
      (+ xpt (nth 0 xyoff))
      (+ ypt (nth 1 xyoff)) 0))
    (command ".SPHERE" pnt1 drad)
    (command ".ZOOM" "e")
    (setq icell (+ icell 1))
   )
  ))
  ; draw cell boundary
  (command ".RECTANGLE"
    (list xpt ypt 0)
    (list (+ xpt xside)
    (+ ypt yside) 0))
  (command ".ZOOM" "e")
  (setq inpt (+ inpt 1))
  (setq xpt (+ xpt xside))
 )
; draw edge
(command ".RECTANGLE" pnt0
  (list (+ xpt xside)
  (+ ypt yside) 0))
(command ".ZOOM" "e")
; extrude the edge
(command ".EXTRUDE"
  "last" "" (* drad -1))
; versions prior to 2007 require
; the taper parameter
;(command ".EXTRUDE"
; "last" "" (* drad -1) "")
; union all
(command ".UNION" "all" "")
(command ".ZOOM" "e")
(princ)
)
;-----------------------------------
```

Note that the sphere is placed at its centroid, so the base needs to have a extrusion height that is negative to cover the bottom half of the sphere.

An alternate three-dimensional version would be to replace the raised dots with depressed dots.

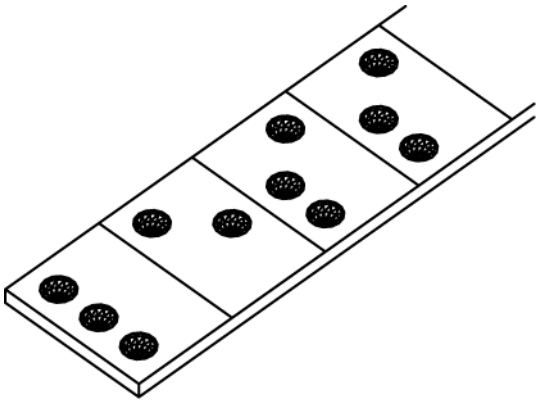

Figure 7.41: PROG02e, text mapped to Braille 3D depressed dots

Add function PROG02e. Using a copy of POG02d, subtract the spheres from the base.

The final section of the function should be change to UNION all of the spheres, place them in a selection list and then subtract them from the extruded base:

```
; union all dots
(command ".UNION" "all" "")
; select union
(setq obj1 (ssget "x"))
; draw edge
(command ".RECTANGLE" pnt0
  (list (+ xpt xside)
  (+ ypt yside) 0))
(command ".ZOOM" "e")
; extrude the edge
(command ".EXTRUDE"
  "last" "" (* drad -1))
; versions prior to 2007 require
; the taper parameter
;(command ".EXTRUDE"
; "last" "" (* drad -1) "")
; select boundary
(setq obj2 (ssget "l"))
; subtract
(command ".SUBTRACT"
  obj2 "" obj1 "")
(command ".ZOOM" "e")
(princ)
```

Use the same script file as for PROG02d.

Using the above examples the circle or the sphere can be replaced with any other shape extruded or placed to create a perforation; scale will determines possible uses.

Another encoding system for text is Morse Code.

The Morse Code alphabet is made up of a set of two states, a short and long pulse, a dot and dash. Figure 42 displays a sample of the characters A through F. The Morse Code supports uppercase characters A through Z and the numbers 0 through 9.

The formal specifications for the Morse Code are: a dot is one unit long, a dash is three units long, gap between dots and

dashes is one unit long, gap between letters is three units long, and the gap between words is seven units long.

The Morse Code differs from the dot-matrix font and the Braille system in that the element to define a single character is not fixed in length or size, it is variable. It is also a binary system, a series of on and off states. Two symbols are required to display this code.

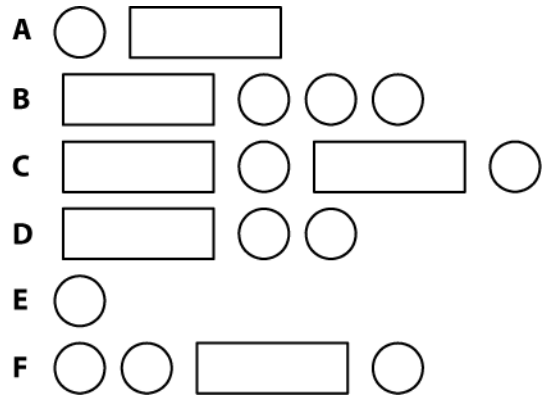

Figure 7.42: Morse Code

Research the background of the Morse Code and how the rest of the alphabet and numbers are defined.

Add function PROG03. Using a copy of PROG02, replace the dot mapping list with a list of dots and dashes. In this example we will use 0 for a dot and 1 for a dash.

Completed function PROG03:

```
;-------------------------------------
(defun prog03 ()
 (prompt "\nPROG03")
 (setq slist (list
   (list "A" (list 0 1))
   (list "B" (list 1 0 0))
   (list "C" (list 1 0 1 0))
   (list "D" (list 1 0 0))
   (list "E" (list 0))
   (list "I" (list 0 0))
   (list "L" (list 0 1 0 0))
   (list "M" (list 1 1))
   (list "O" (list 1 1 1))
   (list "R" (list 0 1 0))
   (list "S" (list 0 0 0))))
 (setq instring
   (getstring "\nEnter a string: "))
 (setq inpt 1)
 (repeat (strlen instring)
  (setq schar
    (substr instring inpt 1))
  (setq schar (strcase schar))
  (setq mapto (assoc schar slist))
  (princ "\n")  (princ schar)
  (princ " = ") (princ mapto)
  (if (/= mapto nil) (progn
   (setq cell (nth 1 mapto))
   (princ " = ") (princ cell)
  ))
  (setq inpt (+ inpt 1))
 )
(princ)
)
;-------------------------------------
```

Execute function PROG03 from the command line, for example:

```
PROG03
Enter a string: "less is more"
L = (L (0 1 0 0)) = (0 1 0 0)
E = (E (0)) = (0)
S = (S (0 0 0)) = (0 0 0)
S = (S (0 0 0)) = (0 0 0)
  = nil
I = (I (0 0)) = (0 0)
S = (S (0 0 0)) = (0 0 0)
  = nil
M = (M (1 1)) = (1 1)
O = (O (1 1 1)) = (1 1 1)
R = (R (0 1 0)) = (0 1 0)
E = (E (0)) = (0)
```

The simplest way to display the Morse Code is to use filled dots and dashes.

Add function PROG03a. Using a copy of PROG03, draw a circle for each dot and a rectangle for a dash. A dot radius will be entered to determine the dimensions for the dots and dashes.

Figure 7.43: PROG03a, text mapped to Morse Code, filled dots and dashes

All of the required dimensions and spacing is be based on a dot radius; a dash is 5 radius long, dot and dash spacing is 1 radius, letter spacing is 2 radius, and word spacing is 4 radius.

Sample script file for PROG03a:

```
;-------------------------------------
(prog03a)
;Enter string:
"less is more"
;Enter dot radius:
1.0
;-------------------------------------
```

Completed function PROG03a:

```
;-------------------------------------
(defun prog03a ()
 (prompt "\nPROG03a")
 (setq slist (list
   (list "A" (list 0 1))
   (list "B" (list 1 0 0))
   (list "C" (list 1 0 1 0))
   (list "D" (list 1 0 0))
   (list "E" (list 0))
   (list "I" (list 0 0))
   (list "L" (list 0 1 0 0))
   (list "M" (list 1 1))
   (list "O" (list 1 1 1))
   (list "R" (list 0 1 0))
   (list "S" (list 0 0 0))))
 (command ".ERASE" "all" "")
 (setq instring
   (getstring "\nEnter a string: "))
 (setq drad
   (getdist "\nEnter dot radius: "))
 ; set gaps
 (setq gapl drad)
 (setq gapl (* drad 2))
 (setq gapw (* drad 4))
 ; start point
 (setq pnt0 (list 0 0 0))
 (setq xpt (nth 0 pnt0))
 (setq ypt (nth 1 pnt0))
 (setq inpt 1)
 (repeat (strlen instring)
  (setq schar
    (substr instring inpt 1))
  (setq schar (strcase schar))
  (setq mapto (assoc schar slist))
  (if (/= mapto nil) (progn
   (setq cell (nth 1 mapto))
   ; get locations
   (setq icell 0)
```

```
   (repeat (length cell)
    (setq ndot (nth icell cell))
    ; dot
    (if (= ndot 0) (progn
     (setq pnt1
       (list (+ xpt drad)
       (+ ypt drad) 0))
     (command ".CIRCLE" pnt1 drad)
     (setq xpt
       (+ (+ xpt (* drad 2)) gapd))
    ))
    ; dash
    (if (= ndot 1) (progn
     (setq pnt1 (list xpt ypt 0))
     (command ".RECTANGLE"
       pnt1 (list (+ xpt (* drad 5))
       (+ ypt (* drad 2)) 0))
     (setq xpt (+ (+ xpt (* drad 5))
       gapd))
    ))
    (command ".ZOOM" "e")
    (command ".FILL" "on")
    (command ".HATCH"
      "solid" "last" "")
    (setq icell (+ icell 1))
   )
  ))
  ; add letter or word gap
  (if (= schar " ")
   (setq xpt (+ xpt (- gapw gapl))
   (setq xpt (+ xpt (- gapl gapd))
  )
  (setq inpt (+ inpt 1))
)
(command ".ZOOM" "e")
(princ)
)
;-----------------------------------
```

Review the structure of PROG03a, how the dot and dash are drawn and how the spacing between dots, dashes, letters, and words is handled.

We can substitute a more graphically compatible dash, one with rounded ends.

Figure 7.44: PROG03b, text mapped to Morse Code, filled dots and curved dashes

Add function PROG03b. Using a copy of PROG03a, replace the dash RECTANGLE with a PLINE that has arcs on each end.

The PLINE command sequence would be:

```
(command ".PLINE"
  (list (+ xpt drad) ypt 0)
  (list (+ xpt (* drad 3))
  ypt 0) "a"
  (list (+ xpt (* drad 3))
  (+ ypt (* drad 2)) 0) "l"
  (list (+ xpt drad)
  (+ ypt (* drad 2)) 0)
  "a" "cl")
(setq xpt (+ (+ xpt (* drad 4))
  gapd))
```

Use the same script file as for PROG03a.

One of the obvious visual issues with this particular mapping is that the results are very linear. For some applications a more compact representation would be desired. One alternative is to vertically place the dashes.

Figure 7.45: PROG03c, text mapped to Morse Code, vertical dash arrangement

Add function PROG03c. Using a copy of PROG03b, draw the dashes vertically.

The PLINE for the dash would be changed to:

```
; dash
(if (= ndot 1) (progn
 (command ".PLINE"
   (list xpt (+ ypt drad) 0)
   (list xpt (- ypt (* drad 3)) 0)
   "a" (list (+ xpt (* drad 2))
   (- ypt (* drad 3)) 0) "l"
   (list (+ xpt (* drad 2))
   (+ ypt drad) 0) "a" "cl")
 (setq xpt (+ (+ xpt (* drad 2))
  gapd))
```

Use the same script file as for PROG03b.

Another alternative would be to place all the characters vertically not horizontally.

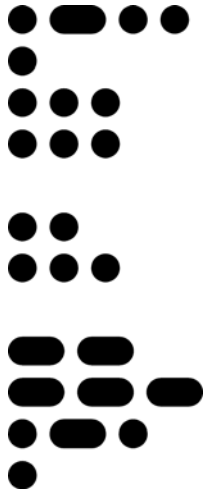

Figure 7.46: PROG03d, text mapped to Morse Code, vertical letter arrangement

Add function PROG03d. Using a copy of PROG03b, change the vertical spacing for every character and set the horizontal position back to the stating point.

The spacing after a letter is drawn is changed from:

```
 ; add letter or word gap
 (if (= schar " ")
  (setq xpt (+ xpt (- gapw gapl))
  (setq xpt (+ xpt (- gapl gapd))
 )
 (setq inpt (+ inpt 1))
)
```

to:

```
 ; vertical spacing for each letter
 (setq xpt (nth 0 pnt0))
 (setq ypt (- (- ypt (* drad 2)) gapd))
 (setq inpt (+ inpt 1))
)
```

Use the same script file as for PROG03c.

Completed function PROG03d:

```
;-----------------------------------
(defun prog03d ()
 (prompt "\nPROG03d")
```

```
 (setq slist (list
   (list "A" (list 0 1))
   (list "B" (list 1 0 0))
   (list "C" (list 1 0 1 0))
   (list "D" (list 1 0 0))
   (list "E" (list 0))
   (list "I" (list 0 0))
   (list "L" (list 0 1 0 0))
   (list "M" (list 1 1))
   (list "O" (list 1 1 1))
   (list "R" (list 0 1 0))
   (list "S" (list 0 0 0))))
 (command ".ERASE" "all" "")
 (setq instring
   (getstring "\nEnter a string: "))
 (setq drad
   (getdist "\nEnter dot radius: "))
 ; set gaps
 (setq gapd drad)
 (setq gapl (* drad 2))
 (setq gapw (* drad 4))
 ; start point
 (setq pnt0 (list 0 0 0))
 (setq xpt (nth 0 pnt0))
 (setq ypt (nth 1 pnt0))
 (setq inpt 1)
 (repeat (strlen instring)
  (setq schar
    (substr instring inpt 1))
  (setq schar (strcase schar))
  (setq mapto (assoc schar slist))
  (if (/= mapto nil) (progn
   (setq cell (nth 1 mapto))
   ; get locations
   (setq icell 0)
   (repeat (length cell)
    (setq ndot (nth icell cell))
    ; dot
    (if (= ndot 0) (progn
     (setq pnt1
       (list (+ xpt drad)
       (+ ypt drad) 0))
     (command ".CIRCLE" pnt1 drad)
     (setq xpt (+ (+ xpt (* drad 2))
       gapd))
    ))
    ; dash
    (if (= ndot 1) (progn
     (command ".PLINE"
       (list (+ xpt drad) ypt 0)
       (list (+ xpt (* drad 3))
      ypt 0) "a"
       (list (+ xpt (* drad 3))
       (+ ypt (* drad 2)) 0) "l"
       (list (+ xpt drad)
       (+ ypt (* drad 2)) 0)
       "a" "cl")
     (setq xpt (+ (+ xpt (* drad 4))
       gapd))
    ))
    (command ".ZOOM" "e")
    (command ".FILL" "on")
    (command ".HATCH"
      "solid" "last" "")
    (setq icell (+ icell 1))
   )
  ))
  ; vertical spacing for each letter
  (setq xpt (nth 0 pnt0))
  (setq ypt (- (- ypt (* drad 2))
    gapd))
  (setq inpt (+ inpt 1))
 )
(command ".ZOOM" "e")
(princ)
)
;-----------------------------------
```

These examples used a fairly short text string. What issues arise if we use a large block of text. For instance this quote from Chapter 1:

"My god is machinery, and the art of the future will be the expression of the individual artist through the thousand powers of the machine… the machine doing all those things that the individual workman cannot do. The creative artist is the man who controls all this and understands it."

Treating this as a single line of text would result in a very long linear series of dots and dashes. We could limit one dimension by specifying how many characters should be placed in a series. The number of lines will be determined by the total number of characters drawn.

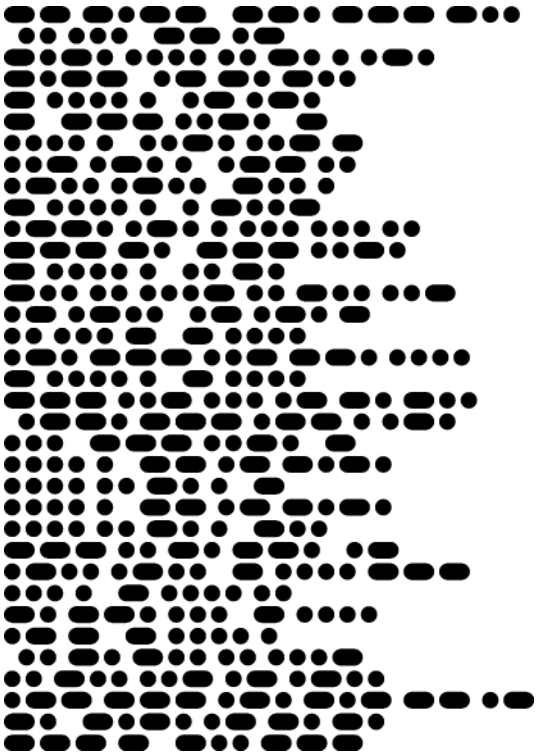

Figure 7.47: PROG03e, text mapped to Morse Code, limit on number of characters per line

Add function PROG03e. Using a copy of PROG03d, add the remaining characters for a complete alphabet, change entry of the text from a single string to the contents of a text file.

Additions include counting the characters as they are being drawn, and reading the text file for each string until it is empty.

After a character is drawn, check if it is equal to the maxiumum characters per line entered:

```
; check for characters per line
(if (= (rem tchars nchar) 0)
(progn
 (setq xpt (nth 0 pnt0))
 (setq ypt (- ypt (+ (* drad 2)
   gapd)))
))
```

Sample script file for PROG03e:

```
;-----------------------------------
(prog03e)
```

```
;Enter text filename:
c:\\text_data\\flw_text.txt
;Enter dot radius:
1.0
;Enter characters per line:
6
;-----------------------------------

Sample text file for PROG03e, file flw-
text.txt:

My god is machinery and the art of
the future will be the expression
of the individual artist through the
thousand powers of the machine
the machine doing all those things
that the individual workman
cannot do

The text file is prespared in Microsoft
NotePad.

Completed function PROG03e:

;-----------------------------------
(defun prog03e ()
 (prompt "\nPROG03e")
 (setq slist (list
   (list "A" (list 0 1))
   (list "B" (list 1 0 0))
   (list "C" (list 1 0 1 0))
   (list "D" (list 1 0 0))
   ; E through W
   (list "X" (list 1 0 0 1))
   (list "Y" (list 1 0 1 1))
   (list "Z" (list 1 1 0 0))
   ))
 (command ".ERASE" "all" "")
 (setq fname
   (getstring
   "\nEnter text filename: "))
 (setq fh1 (open fname "r"))
 (setq drad
   (getdist "\nEnter dot radius: "))
 (setq nchar
   (getint
   "\nEnter characters per line: "))
 ; set gaps
 (setq gapd drad)
 (setq gapl (* drad 2))
 (setq gapw (* drad 4))
 ; start point
 (setq pnt0 (list 0 0 0))
 (setq xpt (nth 0 pnt0))
 (setq ypt (nth 1 pnt0))
 ; total chars
 (setq tchars 0)
 ; read from file
 (while fh1
  (setq instring (read-line fh1))
  (if instring (progn
   (setq inpt 1)
   (repeat (strlen instring)
    (setq schar
      (substr instring inpt 1))
    (setq schar (strcase schar))
    (setq tchars (+ tchars 1))
    (setq mapto (assoc schar slist))
    ; draw character
    (if (/= mapto nil) (progn
     (setq cell (nth 1 mapto))
     ; get locations
     (setq icell 0)
     (repeat (length cell)
      (setq ndot (nth icell cell))
      ; dot
      (if (= ndot 0) (progn
       (setq pnt1
         (list (+ xpt drad)
         (+ ypt drad) 0))
       (command ".CIRCLE" pnt1 drad)
       (setq xpt (+ (+ xpt
         (* drad 2)) gapd))
      ))
      ; dash
      (if (= ndot 1) (progn
       (command ".PLINE"
         (list (+ xpt drad) ypt 0)
         (list (+ xpt (* drad 3))
        ypt 0) "a"
         (list (+ xpt (* drad 3))
         (+ ypt (* drad 2)) 0) "l"
         (list (+ xpt drad)
         (+ ypt (* drad 2)) 0)
         "a" "cl")
       (setq xpt (+ (+ xpt
         (* drad 4)) gapd))
      ))
      (command ".ZOOM" "e")
      (command ".FILL" "on")
      (command ".HATCH"
        "solid" "last" "")
      (setq icell (+ icell 1))
     )
    ))
    ; add letter and word gap
    (if (= schar " ")
     (setq xpt
       (+ xpt (- gapw gapl)))
     (setq xpt
       (+ xpt (- gapl gapd)))
    )
    ; check for characters per line
    (if (= (rem tchars nchar) 0)
     (progn
     (setq xpt (nth 0 pnt0))
     (setq ypt (- ypt (+ (* drad 2)
       gapd)))
    ))
    (setq inpt (+ inpt 1))
   )
  )
  (setq fh1 (close fh1))
 )
)
(command ".ZOOM" "e")
(princ)
)
;-----------------------------------
```

Since the encoding for each character is variable, using the number of characters does not give us a very compact arrangement.

One alternative is to standardize the character spacing, another is to break each line on a letter based on some maximum line dimension.

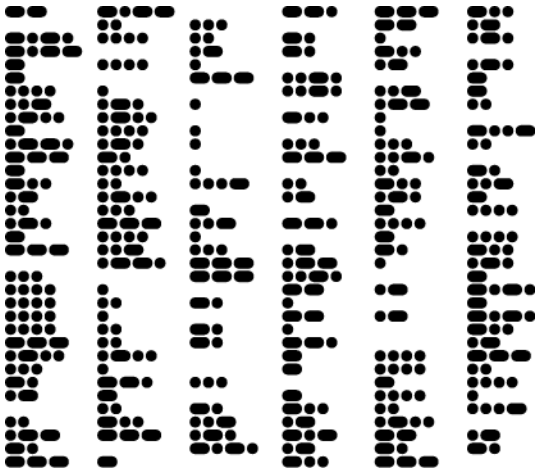

Figure 7.48: PROG03f, text mapped to Morse Code, standardize letter spacing

Add function PROG03f. Using a copy of PROG03e, set each character to a fixed spacing.
The maximum size for an alphabetic character is three dashes and one dot. The spacing can be computed as:

```
; max spacing 3 dashes 1 dot
(setq gapmax (+ (* (+ (* drad 5)
  gapd) 3) (+ (* drad 2) gapd)))
```

The letter and word spacing is changed from:

```
; add letter and word gap
(if (= schar " ")
 (setq xpt
   (+ xpt (- gapw gapl)))
 (setq xpt
   (+ xpt (- gapl gapd)))
)
```

to a fixed distance:

```
; set spacing
(setq sxpt (+ sxpt gapmax))
(setq xpt sxpt)
```

Note the use of the variable *sxpt* to keep track of the starting point for every character drawn and how *xpt* is reset for each character.

Use the same script file as for PROG03e.

Completed function PROG03f:

```
;-----------------------------------
(defun prog03f ()
 (prompt "\nPROG03f")
 (setq slist (list
   (list "A" (list 0 1))
   (list "B" (list 1 0 0))
   (list "C" (list 1 0 1 0))
   (list "D" (list 1 0 0))
   ; add E through W
   (list "X" (list 1 0 0 1))
   (list "Y" (list 1 0 1 1))
   (list "Z" (list 1 1 0 0))
   ))
 (command ".ERASE" "all" "")
 (setq fname
   (getstring
   "\nEnter text filename: "))
 (setq fh1 (open fname "r"))
 (setq drad
   (getdist "\nEnter dot radius: "))
 (setq nchar
   (getint
   "\nEnter characters per line: "))
 ; set gaps
 (setq gapd drad)
 (setq gapl (* drad 2))
 (setq gapw (* drad 4))
 ; max spacing 3 dashes 1 dot
 (setq gapmax (+ (* (+ (* drad 5)
   gapd) 3) (+ (* drad 2) gapd)))
 ; start point
 (setq pnt0 (list 0 0 0))
 (setq xpt (nth 0 pnt0))
 (setq ypt (nth 1 pnt0))
 (setq sxpt xpt)
 ; total chars
 (setq tchars 0)
 ; read from file
 (while fh1
  (setq instring (read-line fh1))
  (if instring (progn
   (setq inpt 1)
   (repeat (strlen instring)
    (setq schar
      (substr instring inpt 1))
    (setq schar (strcase schar))
    (setq tchars (+ tchars 1))
    (setq mapto (assoc schar slist))
    ; draw character
    (if (/= mapto nil) (progn
     (setq cell (nth 1 mapto))
     ; get locations
     (setq icell 0)
     (repeat (length cell)
      (setq ndot (nth icell cell))
      ; dot
      (if (= ndot 0) (progn
       (setq pnt1
         (list (+ xpt drad)
         (+ ypt drad) 0))
       (command ".CIRCLE" pnt1 drad)
       (setq xpt (+ (+ xpt
         (* drad 2)) gapd))
      ))
      ; dash
      (if (= ndot 1) (progn
       (command ".PLINE"
         (list (+ xpt drad) ypt 0)
         (list (+ xpt (* drad 3))
        ypt 0) "a"
         (list (+ xpt (* drad 3))
         (+ ypt (* drad 2)) 0) "l"
         (list (+ xpt drad)
         (+ ypt (* drad 2)) 0)
        "a" "cl")
       (setq xpt (+ (+ xpt
         (* drad 4)) gapd))
      ))
      (command ".ZOOM" "e")
      (command ".FILL" "on")
      (command ".HATCH"
        "solid" "last" "")
      (setq icell (+ icell 1))
     )
    ))
    ; set spacing
    (setq sxpt (+ sxpt gapmax))
    (setq xpt sxpt)
    ; check for characters per line
    (if (= (rem tchars nchar) 0)
     (progn
     (setq xpt (nth 0 pnt0))
     (setq sxpt xpt)
     (setq ypt (- ypt (+ (* drad 2)
       gapd)))
    ))
    (setq inpt (+ inpt 1))
   )
  )
  (setq fh1 (close fh1))
 )
)
(command ".ZOOM" "e")
(princ)
)
;-----------------------------------
```

We standardized the character spacing for the Morse Code, the Braille system already has a standard spacing based on its cell size.

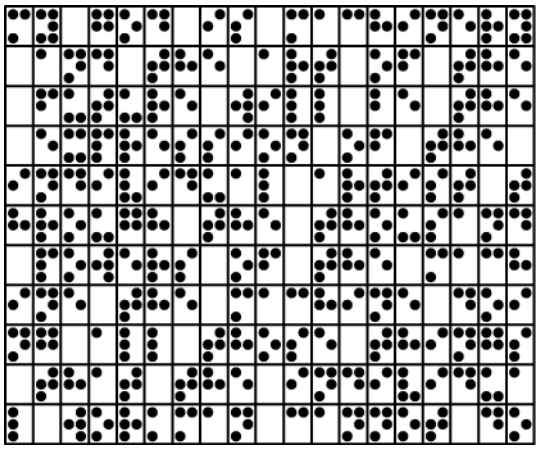

Figure 7.49: PROG02f, text mapped to Braille, standardize letter spacing

Add function PROG02f. Using a copy of PROG03f and PROG02c, use the Braille system to display a block of text.

Expand the Braille cell list to include the entire alphabet. Replace the drawing of the dot and dash with drawing a circle at the cell locations

Sample script file for PROG02f:

```
;-----------------------------------
(prog02f)
;Enter text filename:
c:\\text_data\\flw_text.txt
;Enter X side:
10.0
;Enter characters per line:
19
;-----------------------------------
```

Instead of the using standard spacing for the Morse Code, we can set a maximum line length and only draw the characters that fix on each line.

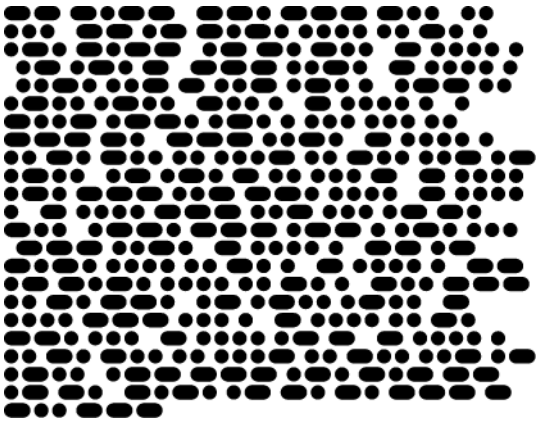

Figure 7.50: PROG03g, text mapped to Morse Code, limit characters by length of line

Add function PROG02g. Using a copy of PROG03e, only draw the characters that fit on each line.

Change the input from maximum characters to maximum line length:

```
(setq xmax
  (getdist
    "\nEnter maximum line length: "))
```

Remove the section that checks for the number of characters.

```
; check for characters per line
(if (= (rem tchars nchar) 0)
 (progn
 (setq xpt (nth 0 pnt0))
 (setq ypt (- ypt (+ (* drad 2)
   gapd)))
))
```

Add before the character is drawn a check to see if it will fit on the current line. If it does not increment the y coordinate.

The check for line length is as follows:

```
(repeat (strlen instring)
 (setq schar
   (substr instring inpt 1))
 (setq schar (strcase schar))
 (setq tchars (+ tchars 1))
 (setq mapto (assoc schar slist))
 ; assume space
 (setq clen gapw)
 (if (/= mapto nil) (progn
  ; check length
  (setq icell 0)
  (setq clen 0.0)
  (setq cell (nth 1 mapto))
  (repeat (length cell)
   (setq ndot (nth icell cell))
   (if (= ndot 0) (setq clen
     (+ clen (+ (* drad 2) gapd))))
   (if (= ndot 1) (setq clen
     (+ clen (+ (* drad 4) gapd))))
   (setq icell (+ icell 1))
  )
  (setq clen (+ clen (- gapl gapd)))
 ))
 ; mod y position
 (if (> (+ xpt clen) xptmax) (progn
  (setq xpt (nth 0 pnt0))
  (setq ypt (- ypt (+ (* drad 2) gapd)))
 ))
 ; draw character
 (if (/= mapto nil) (progn
```

Sample script file for PROG03g:

```
;-----------------------------------
(prog03g)
;Enter text filename:
c:\\Text_data\\flw_text.txt
;Enter dot radius:
1.0
;Enter maximum line length:
90.0
;-----------------------------------
```

The same line length criteria can be used for the vertical version of the dashes. With the vertical dash we can explore more of an abstract visual version of the mapping by modifying the spacing of the dots and dashes.

Figure 7.51: PROG03h, text mapped to Morse Code, limit characters by length of line, vertical dashes

Add function PROG02h. Using a copy of PROG03g, only draw the characters that fit on each line using vertical dashes.

Review the differences in spacing when the dashes are drawn vertically. Also note that the dot has been repositioned at the center of the dash.

Standardize the spacing to match the regular horizontal pattern of the dots and dashes:

```
; set gaps
(setq gapd (* drad 2))
(setq gapl (* drad 4))
(setq gapw (* drad 6))
```

Lower the dot to the center of the vertical dash:

```
; dot
(if (= ndot 0) (progn
 (setq pnt1 (list (+ xpt drad)
  (- ypt drad) 0))
 (command ".CIRCLE" pnt1 drad)
 (setq xpt (+ (+ xpt (* drad 2)) gapd))
))
```

Sample script file for PROG03h:

```
;-----------------------------------
(prog03h)
;Enter text filename:
c:\\Text_data\\flw_text.txt
;Enter dot radius:
1.0
;Enter maximum line length:
130.0
;-----------------------------------
```

Completed function PROG03h:

```
;-----------------------------------
(defun prog03h ()
 (prompt "\nPROG03h")
 (setq slist (list
   (list "A" (list 0 1))
   (list "B" (list 1 0 0))
   (list "C" (list 1 0 1 0))
   (list "D" (list 1 0 0))
   ; add E through W
   (list "X" (list 1 0 0 1))
   (list "Y" (list 1 0 1 1))
   (list "Z" (list 1 1 0 0))
  ))
(command ".ERASE" "all" "")
(setq fname
  (getstring
  "\nEnter text filename: "))
(setq fh1 (open fname "r"))
(setq drad
  (getdist "\nEnter dot radius: "))
(setq xmax
  (getdist
  "\nEnter maximum line length: "))
; set gaps
(setq gapd (* drad 2))
(setq gapl (* drad 4))
(setq gapw (* drad 6))
; start point
(setq pnt0 (list 0 0 0))
(setq xpt (nth 0 pnt0))
(setq ypt (nth 1 pnt0))
; max x
(setq xptmax (+ xpt xmax))
; total chars
(setq tchars 0)
; read from file
(while fh1
 (setq instring (read-line fh1))
 (if instring (progn
  (setq inpt 1)
  (repeat (strlen instring)
   (setq schar
     (substr instring inpt 1))
   (setq schar (strcase schar))
   (setq tchars (+ tchars 1))
   (setq mapto (assoc schar slist))
   ; assume space
   (setq clen gapw)
   (if (/= mapto nil) (progn
    ; check length
    (setq icell 0)
    (setq clen 0.0)
    (setq cell (nth 1 mapto))
    (repeat (length cell)
     (setq ndot (nth icell cell))
     (if (= ndot 0) (setq clen
       (+ clen (+ (* drad 2)
       gapd))))
     (if (= ndot 1) (setq clen
       (+ clen (+ (* drad 2)
       gapd))))
     (setq icell (+ icell 1))
    )
    (setq clen (+ clen
      (- gapl gapd)))
   ))
   ; mod y position
   (if (> (+ xpt clen) xptmax)
    (progn
    (setq xpt (nth 0 pnt0))
    (setq ypt (- ypt (+ (* drad 6)
      gapd)))
   ))
   (if (/= mapto nil) (progn
    (setq cell (nth 1 mapto))
    ; get locations
    (setq icell 0)
    (repeat (length cell)
     (setq ndot (nth icell cell))
     ; dot
     (if (= ndot 0) (progn
      (setq pnt1
        (list (+ xpt drad)
        (- ypt drad) 0))
      (command ".CIRCLE" pnt1 drad)
      (setq xpt (+ (+ xpt
        (* drad 2)) gapd))
     ))
     ; dash
     (if (= ndot 1) (progn
      (command ".PLINE"
        (list xpt (+ ypt drad) 0)
```

```
          (list xpt (- ypt
          (* drad 3)) 0) "a"
          (list (+ xpt (* drad 2))
          (- ypt (* drad 3)) 0) "l"
          (list (+ xpt (* drad 2))
          (+ ypt drad) 0) "a" "cl")
        (setq xpt (+ (+ xpt
          (* drad 2)) gapd))
       ))
       (command ".ZOOM" "e")
       (command ".FILL" "on")
       (command ".HATCH"
         "solid" "last" "")
       (setq icell (+ icell 1))
      )
     ))
     ; add letter and word gap
     (if (= schar " ")
      (setq xpt (+ xpt
        (- gapw gapl)))
      (setq xpt (+ xpt
        (- gapl gapd)))
     )
     (setq inpt (+ inpt 1))
    )
   )
   (setq fh1 (close fh1))
  )
)
(command ".ZOOM" "e")
(princ)
)
;-----------------------------------
```

The vertical dashes also offer the pontential to create larger patterns when they lineup with ones above and below them.

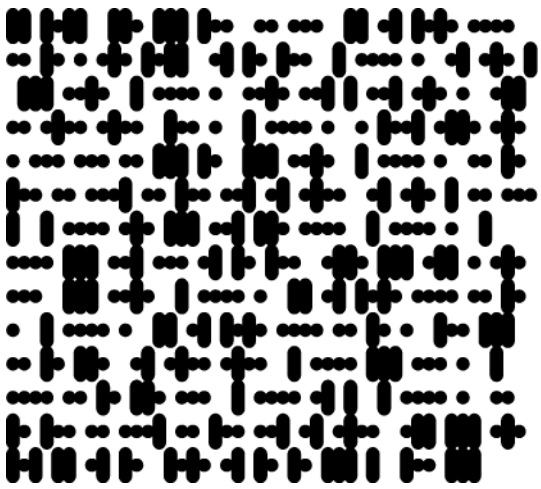

Figure 7.52: PROG03i, text mapped to Morse Code, limit characters by length of line, adjacent vertical dashes

Add function PROG02i. Using a copy of PROG03h, modify the vertical and horizontal spacing of the dots and dashes.

Set the spacing between the dots and dashes to zero, spacing between letters and words remains the same:

```
; set gaps
(setq gapd 0)
(setq gapl (* drad 2))
(setq gapw (* drad 4))
```

For figure 52, the vertical spacing is unchanged so the dashes are adjacent to each other since the spacing between dot and dashes was set to zero:

```
setq ypt (- ypt (+ (* drad 6) gapd)))
```

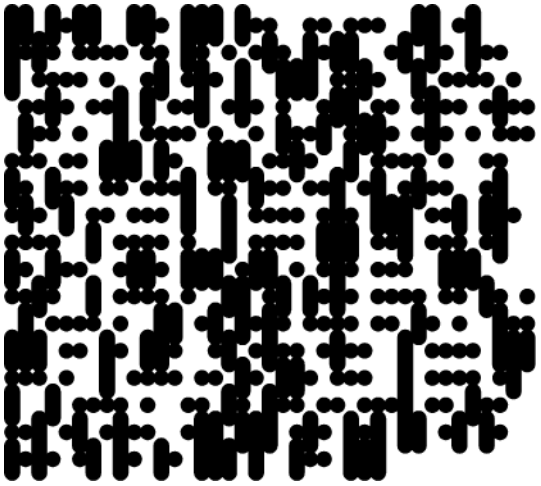

Figure 7.53: PROG03i, text mapped to Morse Code, limit characters by length of line, adjacent vertical dashes

For figure 53, set the vertical spacing so the dashes overlap each other, spacing is changed to:

```
(setq ypt (- ypt (+ (* drad 4) gapd)))
```

Sample script file for PROG03i:

```
;-----------------------------------
(prog03i)
;Enter text filename:
c:\\Text_data\\flw_text.txt
;Enter dot radius:
1.0
;Enter maximum line length:
80.0
;-----------------------------------
```

With this more abstract version of the Morse Code mapping, other dot and dash shapes, two and three-dimensional can be considered to create massing, reliefs, or perforations.

Text also can be represented as a numeric value. Each character that is printable and a number that are not, but are used to control computer processes are assigned an ASCII value, American Standard Code for Information Interchange.

ASCII codes have an integer value ranging from 0 to 255, a single byte of storage, or 8-bits. In binary the values are 00000000 to 11111111.

The numeric value of any character that appears on a keyboard can be determined by using the *ascii* function.

From the command line, enter

```
Command: (ascii "a")
97

Command: (ascii "A")
65

Command: (ascii "Z")
90

Command: (ascii " ")
32
```

```
Command: (ascii "0")
48

Command: (ascii "9")
57
```

Uppercase characters A-Z range 65-90, lowercase a-z range 97-122, numeric 0-9 range 48-57, a space is 32, special characters range 33-47, 58-64, 91-96, and 123-126.

Research the background of the ASCII coding system and how the rest of the values are defined.

Add function PROG04. Using a copy of PROG03, remove the dot and dash mapping list and use the *ascii* function to display the ASCII value of each uppercase character in an entered string.

Completed function PROG04:

```
;-----------------------------------
(defun prog04 ()
 (prompt "\nPROG04")
 (setq instring
   (getstring "\nEnter a string: "))
 (setq inpt 1)
 (repeat (strlen instring)
  (setq schar
    (substr instring inpt 1))
  (setq schar (strcase schar))
  (setq nascii (ascii schar))
  (princ "\n")  (princ schar)
  (princ " = ") (princ nascii)
  (setq inpt (+ inpt 1))
 )
(princ)
)
;-----------------------------------
```

Execute function PROG04 from the command line, for example:

```
Command: (prog04)
PROG04
Enter a string: "less is more"
L = 76
E = 69
S = 83
S = 83
  = 32
I = 73
S = 83
  = 32
M = 77
O = 79
R = 82
E = 69
```

Using the ASCII value of each character in a string to represent height, construct a series of rectangles.

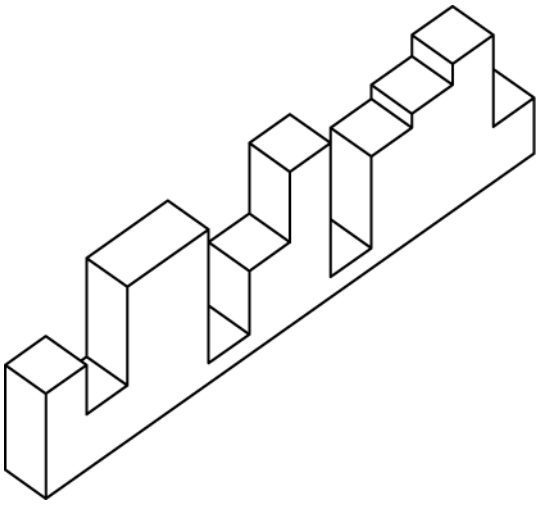

Figure 7.54: PROG04a, text mapped to ASCII values, representing height

Add function PROG04a. Using a copy of PROG03a, remove the dot and dash mapping list and use the *ascii* function to determine the height of each rectangle.

Enter the X dimension of the rectangle and minimum and maximum height. Assume that a space or any other non-alphabetic character is at the minimum height, and that each uppercase character from A through Z is proportional to the difference of the maximum and minimum height.

Extrude each rectangle to the computed height and after all the rectangles are placed, union them all together.

The height of each character is computed as:

```
; set height
(if (and (>= nascii 65)
  (<= nascii 90))
 (progn
  (setq zheight
    (* (/ (- zmax zmin) 26)
    (+ (abs (- 65 nascii)) 1)))
  (setq zheight
    (+ zheight zmin))
 )
 (progn
  (setq zheight zmin)
 )
)
```

Sample script file for PROG04a:

```
;-----------------------------------
(prog04a)
;Enter string:
"less is more"
;Enter X side:
2.0
;Enter Z minimum:
1.0
;Enter Z maximum:
10.0
;-----------------------------------
```

Completed function PROG04a:

```
;-----------------------------------
(defun prog04a ()
 (prompt "\nPROG04a")
 (command ".ERASE" "all" "")
 (setq instring
   (getstring "\nEnter a string: "))
```

```
 (setq xside
   (getdist "\nEnter X side: "))
 (setq zmin
   (getdist
   "\nEnter Z minimum height: "))
 (setq zmax
   (getdist
   "\nEnter Z maximum height: "))
 ; start point
 (setq pnt0 (list 0 0 0))
 (setq xpt (nth 0 pnt0))
 (setq ypt (nth 1 pnt0))
 (setq inpt 1)
 (repeat (strlen instring)
  (setq schar
    (substr instring inpt 1))
  (setq schar (strcase schar))
  (setq nascii (ascii schar))
  ; set height
  (if (and (>= nascii 65)
    (<= nascii 90))
   (progn
    (setq zheight
      (* (/ (- zmax zmin) 26)
      (+ (abs (- 65 nascii)) 1)))
    (setq zheight
      (+ zheight zmin))
   )
   (progn
    (setq zheight zmin)
   )
  )
  (setq pnt1 (list xpt ypt 0))
  (setq pnt2
    (list (+ xpt xside)
    (+ ypt xside) 0))
  (command ".RECTANGLE" pnt1 pnt2)
  (command ".ZOOM" "e")
  ; extrude
  (command ".EXTRUDE"
    "last" "" zheight)
  ; versions prior to 2007 require
  ; the taper parameter
  ;(command ".EXTRUDE"
  ; "last" "" zheight "")
  ; inc to next location
  (setq xpt (+ xpt xside))
  (setq inpt (+ inpt 1))
 )
; union
(command ".UNION" "all" "")
(command ".ZOOM" "e")
(princ)
)
;-----------------------------------
```

Using the ASCII values for height, instead of a single string of text, apply the concept to block of text.

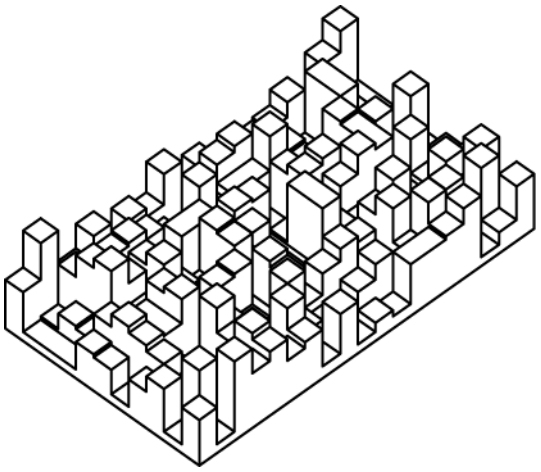

Figure 7.55: PROG04b, text mapped to ASCII values, rectangular volumes

Add function PROG04b. Using a copy of PROG03f and PROG04a, remove the dot and dash mapping list and use the *ascii* function to determine the height of each rectangle.

Compute the rectangle extrusion height in the same manner as in PROG04a.

Sample script file for PROG04b:

```
;-----------------------------------
(prog04b)
;Enter text filename:
c:\\Text_data\\flw_text.txt
;Enter X side:
2.0
;Enter Z minimum:
1.0
;Enter Z maximum:
10.0
;Enter characters per line:
19
;-----------------------------------
```

Completed function PROG04b:

```
;-----------------------------------
(defun prog04b ()
 (prompt "\nPROG04b")
 (command ".ERASE" "all" "")
 (setq fname
   (getstring
   "\nEnter text filename: "))
 (setq fh1 (open fname "r"))
 (setq xside
   (getdist "\nEnter X side: "))
 (setq zmin
   (getdist
   "\nEnter Z minimum height: "))
 (setq zmax
   (getdist
   "\nEnter Z maximum height: "))
 (setq nchar
   (getint
   "\nEnter characters per line: "))
 ; start point
 (setq pnt0 (list 0 0 0))
 (setq xpt (nth 0 pnt0))
 (setq ypt (nth 1 pnt0))
 ; total chars
 (setq tchars 0)
 ; read from file
 (while fh1
  (setq instring (read-line fh1))
  (if instring (progn
   (setq inpt 1)
   (repeat (strlen instring)
    (setq schar
      (substr instring inpt 1))
    (setq schar (strcase schar))
    (setq tchars (+ tchars 1))
    (setq nascii (ascii schar))
    ; set height
    (if (and (>= nascii 65)
      (<= nascii 90))
     (progn
      (setq zheight
        (* (/ (- zmax zmin) 26)
        (+ (abs (- 65 nascii)) 1)))
      (setq zheight
        (+ zheight zmin))
     )
     (progn
      (setq zheight zmin)
     )
    )
    (setq pnt1 (list xpt ypt 0))
    (setq pnt2 (list (+ xpt xside)
      (+ ypt xside) 0))
    (command ".RECTANGLE" pnt1 pnt2)
```

```
    (command ".ZOOM" "e")
    ; extrude
    (command ".EXTRUDE"
      "last" "" zheight)
    ; versions prior to 2007 require
    ; the taper parameter
    ;(command ".EXTRUDE"
    ;  "last" "" zheight "")
    ; inc to next location
    (setq xpt (+ xpt xside))
    ; check for characters per line
    (if (= (rem tchars nchar) 0)
     (progn
     (setq xpt (nth 0 pnt0))
     (setq ypt (- ypt xside))
    ))
    (setq inpt (+ inpt 1))
   )
  )
  (setq fh1 (close fh1))
 )
)
; union
(command ".UNION" "all" "")
(command ".ZOOM" "e")
(princ)
)
;-----------------------------------
```

A number of other shapes could be used to represent each characters. Replace the rectangular volume with a cylinder.

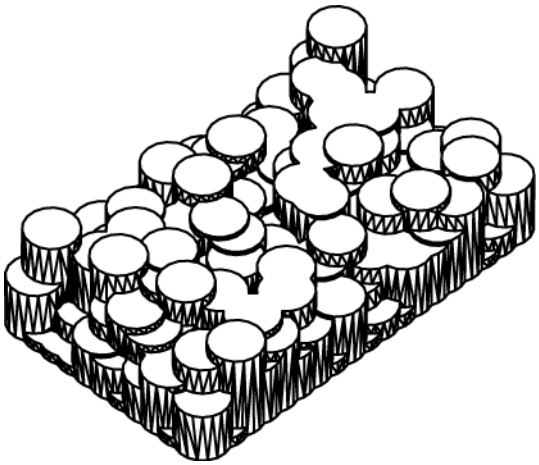

Figure 7.56: PROG04c, text mapped to ASCII values, cylinderical volumes

Add function PROG04c. Using a copy of PROG03b, add input for the cylinder radius and replace the rectangle with a circle.

Add input for cylinder radius:

```
(setq xrad
  (getdist
  "\nEnter cylinder radius: "))
```

Replace the rectangle with a circle, change:

```
(setq pnt1 (list xpt ypt 0))
(setq pnt2 (list (+ xpt xside)
  (+ ypt xside) 0))
(command ".RECTANGLE" pnt1 pnt2)
```

to:

```
(setq pnt1 (list xpt ypt 0))
(command ".CIRCLE" pnt1 xrad)
```

Sample script file for PROG04c:

```
;-----------------------------------
(prog04c)
;Enter text filename:
c:\\Text_data\\flw_text.txt
;Enter X side:
2.0
;Enter cylinder radius:
2.5
;Enter Z minimum:
1.0
;Enter Z maximum:
10.0
;Enter characters per line:
19
;-----------------------------------
```

Note that the cylinder radius is greater than the spacing between cylinders so that they are able to overlap each other to form larger contiguous areas.

To even form larger contiguous areas and further abstract this mapping, group the ASCII values.

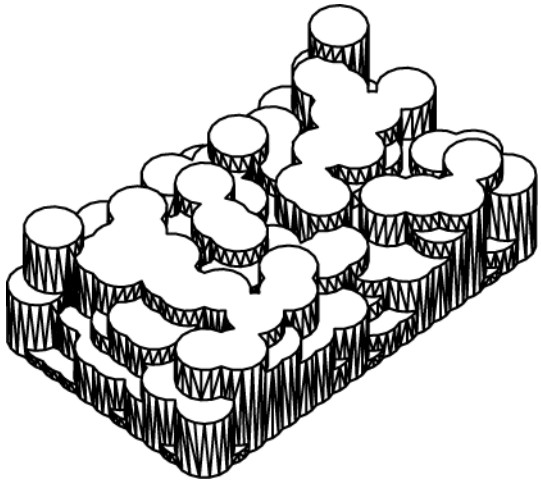

Figure 7.57: PROG04d, text mapped to ASCII values, cylindrical volumes at levels

Add function PROG04d. Using a copy of PROG03c, add input for the number of levels. Convert the ASCII value to a level value. Instead of the range of values to be from 1 to 26, the number of levels will group them; for example if the number of levels is 5, every 5 characters will be the same height.

Add input for number of levels:

```
(setq nlevels
  (getint
  "\nEnter number of levels: "))
```

Change the extrusion height computation from:

```
(setq zheight
  (* (/ (- zmax zmin) 26)
  (+ (abs (- 65 nascii)) 1)))
(setq zheight (+ zheight zmin))
```

to:

```
(setq levelchar (+ (fix
  (+ (abs (- 65 nascii)) 1)
  nlevels)) 1))
(setq zheight
  (* (/ (- zmax zmin) nlevels)
  levelchar))
(setq zheight
```

```
  (+ zheight zmin))
```

Sample script file for PROG04d:

```
;-----------------------------------
(prog04d)
;Enter text filename:
c:\\Text_data\\flw_text.txt
;Enter X side:
2.0
;Enter cylindder radius:
2.5
;Enter Z minimum:
1.0
;Enter Z maximum:
10.0
;Enter characters per line:
19
;Enter number of levels:
5
;-----------------------------------
```

Completed function PROG04d:

```
;-----------------------------------
(defun prog04d ()
 (prompt "\nPROG04d")
 (command ".ERASE" "all" "")
 (setq fname
   (getstring
   "\nEnter text filename: "))
 (setq fh1 (open fname "r"))
 (setq xside
   (getdist "\nEnter X side: "))
 (setq xrad
   (getdist
   "\nEnter cylinder radius: "))
 (setq zmin
   (getdist
   "\nEnter Z minimum height: "))
 (setq zmax
   (getdist
   "\nEnter Z maximum height: "))
 (setq nchar
   (getint
   "\nEnter characters per line: "))
 (setq nlevels
   (getint
   "\nEnter number of levels: "))
 ; start point
 (setq pnt0 (list 0 0 0))
 (setq xpt (nth 0 pnt0))
 (setq ypt (nth 1 pnt0))
 (setq sxpt xpt)
 ; total chars
 (setq tchars 0)
 ; read from file
 (while fh1
  (setq instring (read-line fh1))
  (if instring (progn
   (setq inpt 1)
   (repeat (strlen instring)
    (setq schar
      (substr instring inpt 1))
    (setq schar (strcase schar))
    (setq tchars (+ tchars 1))
    (setq nascii (ascii schar))
    ; set height
    (if (and (>= nascii 65)
      (<= nascii 90))
     (progn
      (setq levelchar (+ (fix (/
        (+ (abs (- 65 nascii)) 1)
       nlevels)) 1))
      (setq zheight
        (* (/ (- zmax zmin) nlevels)
        levelchar))
      (setq zheight
        (+ zheight zmin))
     )
     (progn
      (setq zheight zmin)
     )
    )
    (setq pnt1 (list xpt ypt 0))
    (command ".CIRCLE" pnt1 xrad)
    (command ".ZOOM" "e")
    ; extrude
    (command ".EXTRUDE"
      "last" "" zheight)
    ; versions prior to 2007
    ; require the taper parameter
    ;(command ".EXTRUDE"
    ;  "last" "" zheight "")
    ; inc to next location
    (setq xpt (+ xpt xside))
    ; check for characters per line
    (if (= (rem tchars nchar) 0)
     (progn
     (setq xpt (nth 0 pnt0))
     (setq ypt (- ypt xside))
    ))
    (setq inpt (+ inpt 1))
   )
  )
  (setq fh1 (close fh1))
 )
)
; union
(command ".UNION" "all" "")
(command ".ZOOM" "e")
(princ)
)
;-----------------------------------
```

The ASCII values can also be visually displayed as a eight bit string. The ASCII value of 0 is 00000000, the value of 255 is 11111111.

To get the 0 and 1 list for each ASCII value, the value needs to be converted to an 8-bit binary code. The bit values in decimal by position, left-to-right are:

```
128 + 64 + 32 + 16 + 8 + 4 + 2 + 1
```

For example, if the ASCII value is 65, an uppercase A:

```
65 / 128 = 0
65 / 64  = 1 remainder 1
1  / 32  = 0
1  / 16  = 0
1  / 8   = 0
1  / 4   = 0
1  / 2   = 0
1  / 1   = 1

ASCII 65 = 01000001
```

Add function PROG04e to convert an ASCII value to a 8-bit binary list.

Completed function PROG04e:

```
;-----------------------------------
(defun prog04e (num / pos1 pos2
  pos3 pos4 pos5 pos6 pos7 pos8)
 (setq pos8 (fix (/ num 128)))
 (setq num (- num (* pos8 128)))
 (setq pos7 (fix (/ num 64)))
 (setq num (- num (* pos7 64)))
 (setq pos6 (fix (/ num 32)))
 (setq num (- num (* pos6 32)))
 (setq pos5 (fix (/ num 16)))
 (setq num (- num (* pos5 16)))
 (setq pos4 (fix (/ num 8)))
 (setq num (- num (* pos4 8)))
 (setq pos3 (fix (/ num 4)))
 (setq num (- num (* pos3 4)))
 (setq pos2 (fix (/ num 2)))
 (setq num (- num (* pos2 2)))
 (setq pos1 num)
 (list pos8 pos7 pos6 pos5 pos4
   pos3 pos2 pos1)
)
;-----------------------------------
```

Execute PROG04e at the command for a few values, for example:

```
Command: (prog04e 65)
(0 1 0 0 0 0 0 1)

Command: (prog04e 32)
(0 0 1 0 0 0 0 0)

Command: (prog04e 0)
(0 0 0 0 0 0 0 0)

Command: (prog04e 255)
(1 1 1 1 1 1 1 1)

Command: (prog04e 90)
(0 1 0 1 1 0 1 0)
```

Use function PROG04e to parse the characters for a complete string.

Add function PROG04f. Using a copy of PROG04, add the conversion of the ASCII value to an 8-bit list.

Completed function PROG04f:

```
;-----------------------------------
(defun prog04f ()
 (prompt "\nPROG04f")
 (setq instring
   (getstring "\nEnter a string: "))
 (setq inpt 1)
 (repeat (strlen instring)
  (setq schar
    (substr instring inpt 1))
  (setq schar (strcase schar))
  (setq nascii (ascii schar))
  (setq nlist (prog04e nascii))
  (princ "\n")  (princ schar)
  (princ " = ") (princ nascii)
  (princ " = ") (princ nlist)

  (setq inpt (+ inpt 1))
 )
(princ)
)
;-----------------------------------
```

Execute PROG04f from the command line, for example:

```
Command: (prog04f)
PROG04f
Enter a string: "less is more"
L = 76 = (0 1 0 0 1 1 0 0)
E = 69 = (0 1 0 0 0 1 0 1)
S = 83 = (0 1 0 1 0 0 1 1)
S = 83 = (0 1 0 1 0 0 1 1)
  = 32 = (0 0 1 0 0 0 0 0)
I = 73 = (0 1 0 0 1 0 0 1)
S = 83 = (0 1 0 1 0 0 1 1)
  = 32 = (0 0 1 0 0 0 0 0)
M = 77 = (0 1 0 0 1 1 0 1)
O = 79 = (0 1 0 0 1 1 1 1)
R = 82 = (0 1 0 1 0 0 1 0)
E = 69 = (0 1 0 0 0 1 0 1)
```

This series of on and off states has some common features with the Braille system, it is also a fixed number of elements; and the Morse Code, it has only two states.

One approach to visualize the 8-bit pattern is to use the same concept as we did for the Braille system. Instead of a cell of six dots, we have a cell that is single row of eight dots.

Figure 7.58: PROG04g, text mapped to ASCII binary values

Add function PROG04g. Using a copy of PROG02c, remove the dot list, recompute the dot spacing based on the overall cell dimension, and place the filled dots at computed centers.

The dot radius is based on eight dots to a cell with one radius spacing:

```
; compute radius and Y side
(setq drad (/ xside 25.0))
(setq yside (* drad 4.0))
; spacing for dots
(setq gapd drad)
```

If the bit is on, the circle is filled:

```
(setq ndot (nth icell nlist))
(setq pnt1 (list cxpt
  (+ ypt (* drad 2)) 0))
(command ".CIRCLE" pnt1 drad)
(command ".ZOOM" "e")
(if (= ndot 1) (progn
 (command ".FILL" "on")
 (command ".HATCH"
   "solid" "last" "")
))
```

Sample script file for PROG04g:

```
;-----------------------------------
(prog04g)
;Enter string:
"less"
;Enter X side:
10.0
;-----------------------------------
```

Completed function PROG04g:

```
;-----------------------------------
(defun prog04g ()
 (prompt "\nPROG04g")
 (command ".ERASE" "all" "")
 (setq instring
   (getstring "\nEnter a string: "))
 (setq xside
   (getdist "\nEnter X side: "))
 ; compute radius and Y side
 (setq drad (/ xside 25.0))
 (setq yside (* drad 4.0))
 ; spacing for dots
 (setq gapd drad)
 ; start point
 (setq pnt0 (list 0 0 0))
 (setq xpt (nth 0 pnt0))
 (setq ypt (nth 1 pnt0))
 (setq inpt 1)
 (repeat (strlen instring)
  (setq schar
    (substr instring inpt 1))
  (setq schar (strcase schar))
  (setq nascii (ascii schar))
  (setq nlist (prog04e nascii))
  ; get locations
  (setq icell 0)
  (setq cxpt (+ xpt (+ drad gapd)))
  (repeat (length nlist)
   (setq ndot (nth icell nlist))
   (setq pnt1 (list
     cxpt (+ ypt (* drad 2)) 0))
   (command ".CIRCLE" pnt1 drad)
   (command ".ZOOM" "e")
   (if (= ndot 1) (progn
    (command ".FILL" "on")
    (command ".HATCH"
      "solid" "last" "")
   ))
   (setq icell (+ icell 1))
   (setq cxpt
     (+ cxpt (+ (* drad 2) gapd)))
  )
  ; draw cell boundary
```

```
  (command ".RECTANGLE"
    (list xpt ypt 0)
    (list (+ xpt xside)
    (+ ypt yside) 0))
  (command ".ZOOM" "e")
  (setq inpt (+ inpt 1))
  (setq xpt (+ xpt xside))
 )
(command ".ZOOM" "e")
(princ)
)
;-----------------------------------
```

Since every character now occupies eight dots, even a small collection of text will be a very long series of binary cells.

As we had completed for the Braille system, displaying a block of text based on the number of characters per line, we can do the same for this binary cell.

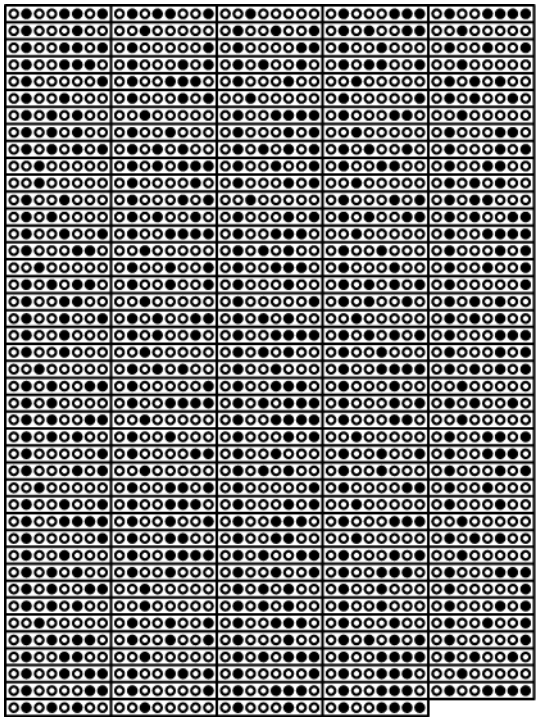

Figure 7.59: PROG04h, text mapped to ASCII binary values, fixed characters per line

Add function PROG04h. Using a copy of PROG04g and PROG02f, read the text from a file and place the dots in the same manner as PROG04g in an 8-bit arrangement.

Sample script file for PROG04h:

```
;-----------------------------------
(prog04h)
;Enter text filename:
c:\\Text_data\\flw_text.txt
;Enter X side:
10.0
;Enter characters per line:
5
;-----------------------------------
```

Completed function PROG04h:

```
;-----------------------------------
(defun prog04h ()
 (prompt "\nPROG04h")
 (command ".ERASE" "all" "")
 (setq fname
   (getstring
   "\nEnter text filename: "))
 (setq fh1 (open fname "r"))
 (setq xside
   (getdist "\nEnter X side: "))
 (setq nchar
   (getint
   "\nEnter characters per line: "))
 ; compute radius and Y side
 (setq drad (/ xside 25.0))
 (setq yside (* drad 4.0))
 ; spacing for dots
 (setq gapd drad)
 ; start point
 (setq pnt0 (list 0 0 0))
 (setq xpt (nth 0 pnt0))
 (setq ypt (nth 1 pnt0))
 ; total chars
 (setq tchars 0)
 ; read from file
 (while fh1
  (setq instring (read-line fh1))
  (if instring (progn
   (setq inpt 1)
   (repeat (strlen instring)
    (setq schar
      (substr instring inpt 1))
    (setq schar (strcase schar))
    (setq tchars (+ tchars 1))
    (setq nascii (ascii schar))
    (setq nlist (prog04e nascii))
    ; get locations
    (setq icell 0)
    (setq cxpt (+ xpt (+ drad gapd)))
    (repeat (length nlist)
     (setq ndot (nth icell nlist))
     (setq pnt1 (list cxpt
       (+ ypt (* drad 2)) 0))
     (command ".CIRCLE" pnt1 drad)
     (command ".ZOOM" "e")
     (if (= ndot 1) (progn
      (command ".FILL" "on")
      (command ".HATCH"
        "solid" "last" "")
     ))
     (setq icell (+ icell 1))
     (setq cxpt (+ cxpt (+
       (* drad 2) gapd)))
    )
    ; draw cell boundary
    (command ".RECTANGLE"
      (list xpt ypt 0)
      (list (+ xpt xside)
      (+ ypt yside) 0))
    (command ".ZOOM" "e")
    (setq xpt (+ xpt xside))
    ; check for characters per line
    (if (= (rem tchars nchar) 0)
     (progn
      (setq xpt (nth 0 pnt0))
      (setq ypt (- ypt yside))
     ))
     (setq inpt (+ inpt 1))
   )
  )
  (setq fh1 (close fh1))
 )
)
(command ".ZOOM" "e")
(princ)
)
;-----------------------------------
```

An abstract version of the ASCII codes can be created by modifying the vertical spacing of each line of bit codes and changing the shape of the on and off bits to read as 0 and 1.

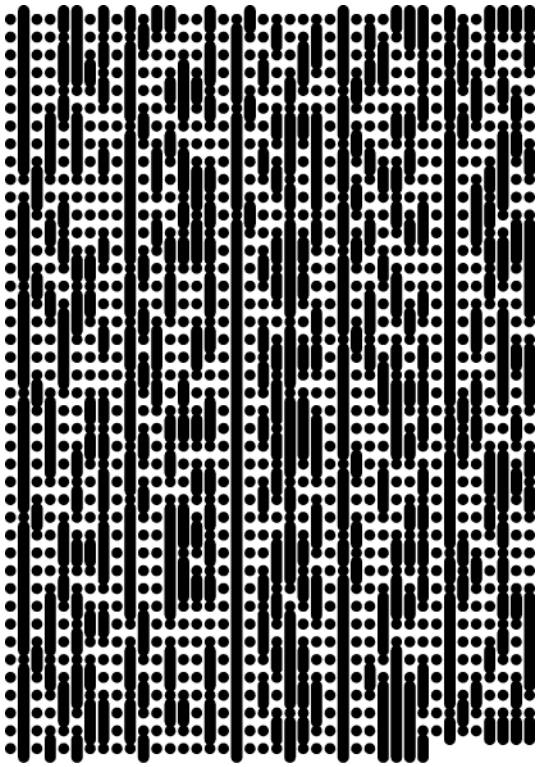

Figure 7.60: PROG04i, text mapped to ASCII binary values, fixed characters per line, modified spacing and dots

Add function PROG04i. Using a copy of PROG04h, change the off state to a filled dot, the on state to a vertical line, and remove the cell edge.

Sample script file for PROG04i:

```
;-----------------------------------
(prog04i)
;Enter text filename:
c:\\Text_data\\flw_text.txt
;Enter X side:
10.0
;Enter characters per line:
5
;-----------------------------------
```

Completed function PROG04i:

```
;-----------------------------------
(defun prog04i ()
 (prompt "\nPROG04i")
 (command ".ERASE" "all" "")
 (setq fname
   (getstring
   "\nEnter text filename: "))
 (setq fh1 (open fname "r"))
 (setq xside
   (getdist "\nEnter X side: "))
 (setq nchar
   (getint
   "\nEnter characters per line: "))
 ; compute radius and Y side
 (setq drad (/ xside 25.0))
 (setq yside (* drad 4.0))
 ; spacing for dots
 (setq gapd drad)
 ; start point
 (setq pnt0 (list 0 0 0))
 (setq xpt (nth 0 pnt0))
 (setq ypt (nth 1 pnt0))
 ; total chars
 (setq tchars 0)
 ; read from file
 (while fh1
  (setq instring (read-line fh1))
  (if instring (progn
   (setq inpt 1)
   (repeat (strlen instring)
    (setq schar
      (substr instring inpt 1))
    (setq schar (strcase schar))
    (setq tchars (+ tchars 1))
    (setq nascii (ascii schar))
    (setq nlist (prog04e nascii))
    ; get locations
    (setq icell 0)
    (setq cxpt
      (+ xpt (+ drad gapd)))
    (repeat (length nlist)
     (setq ndot (nth icell nlist))
     (setq pnt1 (list cxpt
       (- ypt (* drad 1)) 0))
     (if (= ndot 0) (progn
      (command ".CIRCLE" pnt1 drad)
      (command ".ZOOM" "e")
      (command ".FILL" "on")
      (command ".HATCH"
        "solid" "last" "")
     ))
     (if (= ndot 1) (progn
      (command ".PLINE"
        (list (- cxpt drad)
        (+ ypt drad) 0)
        (list (- cxpt drad)
        (- ypt (* drad 3)) 0) "a"
        (list (+ cxpt (* drad 1))
        (- ypt (* drad 3)) 0) "l"
        (list (+ cxpt (* drad 1))
        (+ ypt drad) 0) "a" "cl")
      (command ".ZOOM" "e")
      (command ".FILL" "on")
      (command ".HATCH"
        "solid" "last" "")
     ))
     (setq icell (+ icell 1))
     (setq cxpt
       (+ cxpt (+ (* drad 2) gapd)))
    )
    (setq xpt (- (- cxpt drad) gapd))
    ; check for characters per line
    (if (= (rem tchars nchar) 0)
     (progn
     (setq xpt (nth 0 pnt0))
     (setq ypt (- ypt yside))
    ))
    (setq inpt (+ inpt 1))
   )
  )
  (setq fh1 (close fh1))
 )
)
(command ".ZOOM" "e")
(princ)
)
;-----------------------------------
```

Review the PLINE for the vertical on bit and how the horizontal and vertical spacing has been changed.

Instead of setting the radius based on the 8-bit cell, a separate entry could be added specifying the radius of each bit, or a dimension per line could be entered, so a radius could be computed.

To further abstract the ASCII code and to create larger contiguous areas and patterns change the spacing so the 0s and 1s are adjacent to each other.

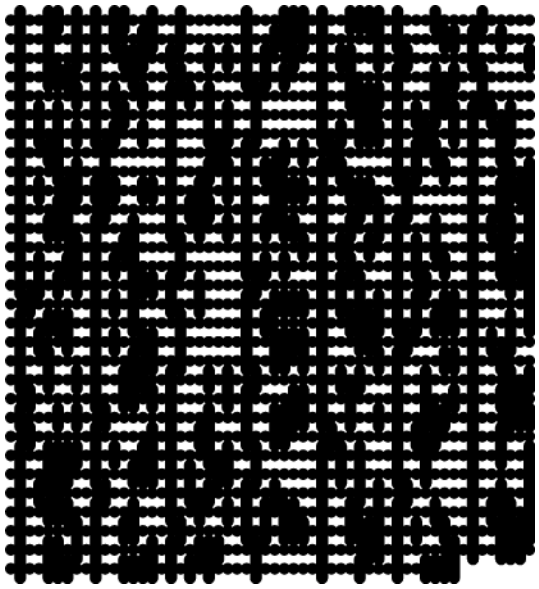

Figure 7.61: PROG04i, text mapped to ASCII binary values, fixed characters per line, zero spacing

In function PROG04i, change the spacing dimension from:

```
; spacing for dots
(setq gapd drad)
```

to:

```
; spacing for dots
(setq gapd 0)
```

Another consideration for this series would be to add padding at the end of the text with extra spaces when the last row is not complete. Also the overall size could be scaled to actual dimensions.

Most of these examples have remained two dimensional, many of then could also be have a three dimensions interpretation.

As we did in the last version, the on and off states could have the same symbol but be varied by height.

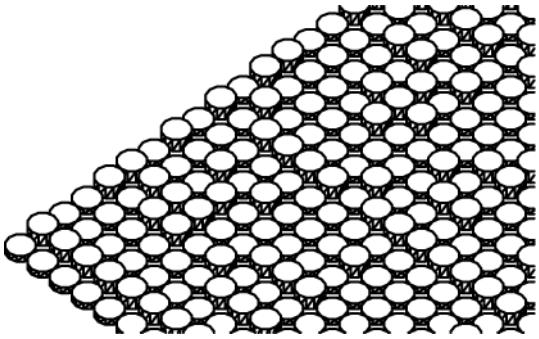

Figure 7.62: PROG04j, text mapped to ASCII binary values, vary heights

Add function PROG04j. Using a copy of PROG04i, set the on and off state as heights and use the x side dimension to determine the bit radius.

The x side dimension sets the horizontal and vertical spacing, and the radius for both the on and off bits:

```
; compute radius and Y side
(setq drad (/ xside 2.0))
(setq yside (* drad 2.0))
```

For example, the bit 0 state is set by:

```
(if (= ndot 0) (progn
 (command ".CIRCLE" pnt1 drad)
 (command ".ZOOM" "e")
 ; extrude
 (command ".EXTRUDE"
   "last" "" hstate0)
 ; versions prior to 2007
 ; require taper parameter
 ;(command ".EXTRUDE"
 ; "last" "" hstate0 "")
))
```

The bit 1 state would be set in the same way using a differenet height.

Sample script file for PROG04j:

```
;-------------------------------------
(prog04j)
;Enter text filename:
c:\\Text_data\\flw_text.txt
;Enter X side:
10.0
;Enter characters per line:
5
;Enter off state height:
2.5
;Enter on state height:
5.0
;-------------------------------------
```

Completed function PROG04j:

```
;-------------------------------------
(defun prog04j ()
 (prompt "\nPROG04j")
 (command ".ERASE" "all" "")
 (setq fname
   (getstring
   "\nEnter text filename: "))
 (setq fh1 (open fname "r"))
 (setq xside
   (getdist "\nEnter X side: "))
 (setq nchar
   (getint
   "\nEnter characters per line: "))
 (setq hstate0
   (getdist
   "\nEnter off state height: "))
 (setq hstate1
   (getdist
   "\nEnter on state height: "))
 ; compute radius and Y side
 (setq drad (/ xside 2.0))
 (setq yside (* drad 2.0))
 ; spacing for dots
 (setq gapd 0)
 ; start point
 (setq pnt0 (list 0 0 0))
 (setq xpt (nth 0 pnt0))
 (setq ypt (nth 1 pnt0))
 ; total chars
 (setq tchars 0)
 ; read from file
 (while fh1
  (setq instring (read-line fh1))
  (if instring (progn
   (setq inpt 1)
   (repeat (strlen instring)
    (setq schar
      (substr instring inpt 1))
    (setq schar (strcase schar))
    (setq tchars (+ tchars 1))
    (setq nascii (ascii schar))
```

```
    (setq nlist (prog04e nascii))
    ; get locations
    (setq icell 0)
    (repeat (length nlist)
     (setq ndot (nth icell nlist))
     (setq xpt
       (+ xpt (+ (* drad 2) gapd)))
     (setq pnt1 (list
       xpt (- ypt (* drad 1)) 0))
     (if (= ndot 0) (progn
      (command ".CIRCLE" pnt1 drad)
      (command ".ZOOM" "e")
      ; extrude
      (command ".EXTRUDE"
        "last" "" hstate0)
      ; versions prior to 2007
      ; require taper parameter
      ;(command ".EXTRUDE"
      ; "last" "" hstate0 "")
     ))
     (if (= ndot 1) (progn
      (command ".CIRCLE" pnt1 drad)
      (command ".ZOOM" "e")
      ; extrude
      (command ".EXTRUDE"
        "last" "" hstate1)
      ; versions prior to 2007
      ; require taper parameter
      ;(command ".EXTRUDE"
      ; "last" "" hstate1 "")
     ))
     (setq icell (+ icell 1))
    )
    ; check for characters per line
    (if (= (rem tchars nchar) 0)
     (progn
     (setq xpt (nth 0 pnt0))
     (setq ypt (- ypt yside))
    ))
    (setq inpt (+ inpt 1))
   )
  )
  (setq fh1 (close fh1))
 )
)
(command ".ZOOM" "e")
(princ)
)
;-----------------------------------
```

Also note that a simpler version of computing the horizontal and vertical spacing was added to this function based on a single bit value, not a set of 8-bits.

The on bit state could also be represented by a greater radius along with its height.

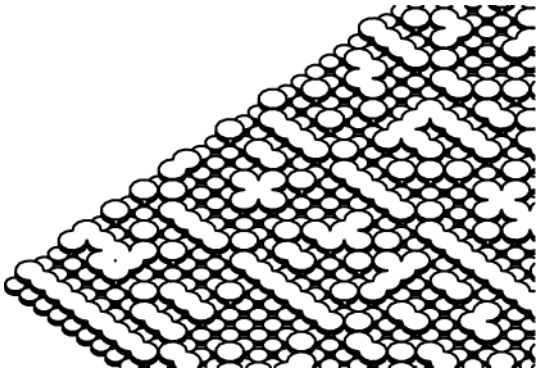

Figure 7.63: PROG04k, text mapped to ASCII binary values, vary heights and radius

Add function PROG04k. Using a copy of PROG04j, add input for the on bit radius and UNION all the bits together at the end of the function.

A greater radius would overlap adjacent bits forming larger contiguous patterns.

Sample script file for PROG04k:

```
;-----------------------------------
(prog04k)
;Enter text filename:
c:\\Text_data\\flw_text.txt
;Enter X side:
10.0
;Enter characters per line:
5
;Enter off state height:
2.5
;Enter on state height:
5.0
;Enter on state radius:
7.0
;-----------------------------------
```

Completed function PROG04k:

```
;-----------------------------------
(defun prog04k ()
 (prompt "\nPROG04k")
 (command ".ERASE" "all" "")
 (setq fname
   (getstring
   "\nEnter text filename: "))
 (setq fh1 (open fname "r"))
 (setq xside
   (getdist "\nEnter X side: "))
 (setq nchar
   (getint
   "\nEnter characters per line: "))
 (setq hstate0
   (getdist
   "\nEnter off state height: "))
 (setq hstate1
   (getdist
   "\nEnter on state height: "))
 (setq rstate1
   (getdist
   "\nEnter on state radius: "))
 ; compute radius and Y side
 (setq drad (/ xside 2.0))
 (setq yside (* drad 2.0))
 ; spacing for dots
 (setq gapd 0)
 ; start point
 (setq pnt0 (list 0 0 0))
 (setq xpt (nth 0 pnt0))
 (setq ypt (nth 1 pnt0))
 ; total chars
 (setq tchars 0)
 ; read from file
 (while fh1
  (setq instring (read-line fh1))
  (if instring (progn
   (setq inpt 1)
   (repeat (strlen instring)
    (setq schar
      (substr instring inpt 1))
    (setq schar (strcase schar))
    (setq tchars (+ tchars 1))
    (setq nascii (ascii schar))
    (setq nlist (prog04e nascii))
    ; get locations
    (setq icell 0)
    (repeat (length nlist)
     (setq ndot (nth icell nlist))
     (setq xpt
       (+ xpt (+ (* drad 2) gapd)))
     (setq pnt1 (list
       xpt (- ypt (* drad 1)) 0))
     (if (= ndot 0) (progn
      (command ".CIRCLE" pnt1 drad)
      (command ".ZOOM" "e")
      ; extrude
      (command ".EXTRUDE"
        "last" "" hstate0)
      ; versions prior to 2007
      ; require the taper parameter
```

```
      ;(command ".EXTRUDE"
      ;  "last" "" hstate0 "")
     ))
     (if (= ndot 1) (progn
      (command ".CIRCLE"
        pnt1 rstate1)
      (command ".ZOOM" "e")
      ; extrude
      (command ".EXTRUDE"
        "last" "" hstate1)
      ; versions prior to 2007
      ; require the taper parameter
      ;(command ".EXTRUDE"
      ;  "last" "" hstate1 "")
     ))
     (setq icell (+ icell 1))
    )
    ; check for characters per line
    (if (= (rem tchars nchar) 0)
     (progn
     (setq xpt (nth 0 pnt0))
     (setq ypt (- ypt yside))
    ))
    (setq inpt (+ inpt 1))
   )
  )
  (setq fh1 (close fh1))
 )
)
; union
(command ".BREP" "all" "")
(command ".UNION" "all" "")
(command ".ZOOM" "e")
(princ)
)
;-----------------------------------
```

Consider other possible shapes that could represent the on and off states of the ASCII code bits: circles, squares, rotated squares, and n-side polygons.

In all cases our goal is to retain the mapping of the characters and also begin to develop interesting overall patterns that could be used architecturally to express context and project meaning.