TRANSPOSED TWILLS: A PROGRAMMING APPROACH

B. Pourdeyhimi

Introduction.

Transposing, alternatively known as reversing, is primarily used in the construction of broken twills. These represent an interesting class of weaves as to their aesthetic features. Watson's Design and Color reserves the term 'transposed' as applied to twills [1]. However, the method may be equally utilized in the construction of fancy variations of other standard or non-standard weaves. In this paper, the techniques discussed use twills as examples although they may be used to transpose any weave.

Weaves may be rearranged to provide transposed warp or weft effects. This paper considers the warp first for demonstrating the technique. Later, the transposed weft effects will be contrasted to transposed warp effects. The procedures for constructing a transposed warp effect are:

1. a base weave is selected;
2. draft and peg-plans are determined;
3. the draft will be rearranged in a transposed order;
4. the transposed weave is determined using the new draft and the original peg-plan.

There are, of course, other considerations to be taken into account, depending on the method of transposing employed and the number of ends transposed as a group.

Regardless of the number of ends to be transposed, the final result would depend on the technique employed. Watson's Design and Color gives three variations for transposing twills. Figure 1 illustrates these techniques when transposed in groups of 2 ends. The techniques shown in Figures 1.b to 1.d are known as a regular,

ARS TEXTRINA 9 (1988), pp. 43-62

regular and straight and 4-end sateen transpose of the draft given in Figure 1.a, respectively.

It sometimes may be necessary to extend the design. In the first variety, if the number of warp ends is not a multiple of the number of ends to be transposed as a group, the number of ends has to be extended such that it will equal the lowest common denominator (LCD) of the number of ends in the repeat unit of the base weave (E_{rep}) and the number of ends to be transposed as a group (E_{tr}). For example, an eight end twill transposed in groups of 3 ends will have to be extended over 24 ends, that is, LCD of 8 and 3. Similarly, in the second and third variety, if E_{rep} is not a multiple of $2E_{tr}$ and $4E_{tr}$ respectively, the ends have to be extended accordingly.

In what follows, a simple computer program will be developed for transposing weaves. The benefit of such a computerized technique lies in its efficiency. Manual transposition of weaves is a somewhat tedious and laborious task.

Method.

Representation of weave point diagrams in the form of a binary matrix of integers of 0 and 1, where a 1 corresponds to a black square and a 0 to a white one, is not a new concept [2]. The last few years have witnessed development of techniques for representation and analyses of various structures based on this concept [3 - 6]. The work reported here represents one part of developing an interactive program for instruction of weave design and uses the same binary matrix as the starting point. A system designed for instructional purposes must be capable of generating various weaves by means of a simple protocol. That is, very little effort should be required for the operation of the system. Rather than the operator deciding the size of the weave, and manually transposing the draft or the peg-plan and recreating the weave, the computer will generate the required end result. Computer transposition poses a problem of terminology and language to be used; this, however, is not the concern of this paper. A simplified flow chart is given in Figure 2 for creating transposed weaves.

One method that may be employed for transposing weaves involves developing algorithms for swapping the required columns of the draft or the weave. For example, regular transposition of the draft in groups of 2 ends for a 2/2 twill with a binary matrix representation of

Draft		Weave	
	0 0 0 1		0 0 1 1
	0 0 1 0		0 1 1 0
	0 1 0 0		1 1 0 0
	1 0 0 0		1 0 0 1

may be easily accomplished by swapping columns 1 and 2, and columns 3 and 4, as seen below.

Draft		Weave	
	0 0 1 0		0 0 1 1
	0 0 0 1		1 0 0 1
	1 0 0 0		1 1 0 0
	0 1 0 0		0 1 1 0

This method is most effective for regular transposition of weaves, but will get a little complicated for regular, straight and 4-end sateen transposition of weaves, and some unnecessary repetition would be involved in transforming the matrix. Closer inspection of the draft matrix reveals that transposing of each cell (E_{tr}, in groups of 2 ends in this case) would best be explained by a 90 degree clockwise rotation of the appropriate cells. That is a matrix of

0 1
1 0

is transformed into

1 0
0 1

The third variety (4-end sateen base) on the other hand, involves a two step rotation process. The first step involves a 90 degree clockwise rotation of every block or cell (that is, the same as the first variety). The second step requires a 90 degree counter-clockwise rotation of the second half of the structure. A 2/2 twill weave transposed in groups of 2 ends is shown below:

Weave	0 0 1 1 0 0 1 1
	0 1 1 0 0 1 1 0
	1 1 0 0 1 1 0 0
	1 0 0 1 1 0 0 1
Step 1	0 0 1 1 0 0 1 1
	1 0 0 1 1 0 0 1
	1 1 0 0 1 1 0 0
	0 1 1 0 0 1 1 0
Step 2	0 0 1 1 0 0 1 1
	1 0 0 1 0 1 1 0
	1 1 0 0 1 1 0 0
	0 1 1 0 1 0 0 1

This means that a single rotation algorithm can be used both for rotating and transposing the weave matrix. Obviously, the computer program to accomplish this would need to first copy the contents of the array into a temporary array before any transformation is attempted. The programs are written in 'C' due to code size and speed considerations. The program 'Rotate' given in Appendix A, can be used to rotate (clockwise or counter-clockwise) an array in its entirety, or only portions thereof, by 90, 180 or 270 degrees. For the purposes of transposing a weave, however, the weave is rotated by 90 degrees only.

The algorithms developed in Appendix B, demonstrate the use of the 'Rotate' routine for the transposition of the weaves. The routine called 'Transpose' determines the method to be employed and passes the control to the required routines. It will be noted that we have categorized the transposition techniques into regular and irregular with a further sub classification of the regular where every cell or alternate cells may be transposed.

The 'Rotate' routine may be used to transpose the draft, peg-plan or the weave (for warp effects only), depending on the way in which the rest of the system would operate. If the weave were to be transposed in the weft direction, the peg-plan would need to be transposed while still using the old draft. Alternatively, the 'for' loops in 'RegTranspose' and 'IrregTranspose' routines may be modified to handle the weave directly. The same 'Rotate' routine may be utilized to achieve that.

Summary and conclusions.

A program developed in 'C' for IBM Personal Computers allows for simple transposition of weaves. The program was developed for rotating an array or portions thereof, and may be used for transposing weave arrays. This technique allows for the generation of transposed weaves using various methods described in Watson's Design and Color. Figures 3.b to 3.d demonstrate how the routines *transpose the twill given in Figure 3.a in the warp direction in* groups of 2 ends.

The routine is equally applicable to other weaves. That is, any other weave may also be transposed using this technique. The structure of the program allows for this, as it handles the contents of a two dimensional binary array independent of the type of weave. Another interesting application of the 'Rotate' routine may be for converting right-handed twills into left-handed ones or vice versa, by rotating the entire array.

Acknowledgements.

This work has been supported by the Instructional Computing Program, Fulcrum Project, Computer Science Center, University of Maryland. Thanks are also due to David Cohen and Jim Mowbray for their assistance. I also would like to thank Professor Steven Spivak for reading the manuscript and for his valuable suggestions.

References

[1] Z.J. Grosicki, "Watson's Design and Color, 4th ed.", London: Newnes-Butterworths, 1977

[2] A. Newton and B.P. Sarkar, *An analysis of compound weaves,* J.Textile Inst. **10** (1979), p. 427.

[3] J.A. Hoskins, *Multi-layered cloths: A structured approach,* Ars Textrina **1** (1983), p. 137.

[4] J.A. Hoskins, *Factoring binary matrices: A weaver's approach,* Comb.Math. IX, Lecture Notes in Math. **952** (1982), p. 300.

[5] J.A. Hoskins, R.G. Stanton and A.P. Street, *Enumerating the compound twillins,* Congressus Numeratium **38** (1983), p. 3.

[6] J.A. Hoskins, R.G. Stanton and P. Street, *The compound twillins: Reflection at an element,* Ars Combinatoria, **17** (1984), p. 177.

Department of Textiles & Consumer Economics
University of Maryland
College Park, Maryland 20742
U.S.A.

APPENDIX A

Program for rotating the weave array

'C' stores multi-dimensional arrays as single-dimensional arrays and accesses them using the following formula:

$$\text{addr}(A[X1][X2]\ldots[Xn]) = A + * \sum_{(i=1)}^{n} \left(Xi * \sum_{(j=i+1)}^{n} (Mj * Sj) \right)$$

where:

A = base address of the array;
n = number of dimensions in the array;
Xi = index of the i^{th} level;
Mj = limit of the index at the j^{th} level; and
Sj = size multiplier at the j^{th} level.

If the array is homogeneous, then:
Si = 1 for all i < n, and
Sn is the size of the base elements of the array.

Although it would have been possible to use a single-dimensional array for storing the weave information, the programs employ a two-dimensional array. A two-dimensional array may be more appropriate since it relates more directly to the way in which weaves are generated.

The following program can be used to rotate an array in its entirety, or only portions thereof, clockwise or counter-clockwise in 90 degree increments. Since the weaves can be rotated only by 90 degree increments, rather than using the degree by which the weave is to be rotated, the program uses the number of quarter turns necessary for rotating the weave. The content of the weave array is copied into a temporary array before it is rearranged.

```
static char tmp [ 100 ] [ 100 ];        /*
                                         Temporary array for
                                          storing the information
                                        */
```

```
Rotate (Array, A, B, M, N, turn )
char Array [ 100 ] [ 100 ];
int A, B, *M, *N, turn;
/*
Arguments:
                Array  = data to rotate ( int Array [ M ] [ N ]; )
                A,B    = start point in Array to rotate
                M,N    = dimensions (rows, cols ) of Array to
Rotation:
                turn = number of quarter-turns
                     < 0 = counter-clockwise
                     > 0 = clockwise
Note: If turn is odd, then the dimensions of Array will be reversed upon exit.
       Array will be defined as:
       'int Array[ N ] [ M ];'.
        Note that the transposed routines should only use 1 or -1 for the turns.
*/
{
  /* Local variables */
   int x, y, s, f, hy, hx;
/*
make turn in range -3 . . 3
Although any integer number may be used, the result would be a rotation by 1, 2 or
3 quarter turns only.
*/
      if ( turn < 0 )
      {
        turn = - turn;
        turn %= 4;
        turn = 4 - turn;
      }
      else turn % = 4;

/* If turn is zero, then return (no operation necessary) */
if (turn == 0) return;

/* Set hx and hy equal to number of ends and picks */
     hx = weave.h_size;
```

```
        hy = weave.v_size;

/*  Copy to temporary array  */
for ( x = 0; x < hx; ++ x )
    for ( y = 0; y < hy; ++ y )
        tmp[ x ] [ y ] = Array [ x ] [ y ];

/*  Rotate data  */
for ( x = 0; x < *M; ++ x )
    for ( y = 0 ); y < *N; ++ y )
    {
        int  x1, y1, t;

/*  Compute rotation coefficients  */
      x1 = *M - x - 1;
      y1 = *N - y - 1;

/*  Get element to move  */
      t = Array [ A + x ] [ B + y ];

/*  Move to space determined by turn  */
      switch (turn)
      {
    case 1 :  /*  cw 1; ccw 3  */
        tmp [ A + y ] [ B + x1 ] = t;
        break;
    case 2 :  /*  cw 2; ccw 2  */
        tmp [ A + x1 ] [ B + y1 ] = t;
        break;
    case 3 :  /*  cw 3; ccw 1  */
        tmp [ A + y1 ] [ B + x ] = t;
       }
   }
if  (turn == 1  ||  turn == 3)
{
  x  = *M;
*M  = *N;
*N  =  x;
```

```
/* Reset the weave size */
hx = weave.v_size;
hy = weave.h_size;
}
/* Copy back to original array */
for ( x = 0; x < hx; ++ x )
  for ( y = 0; y < hy; ++ y )
    Array [ x ] [ y ] = tmp [ x ] [ y ];
}
```

APPENDIX B

Programs for transposing a weave array

The programs given below demonstrate how the Rotate routine is used to transpose the weave array. The program follows closely the flow chart given in Figure 2. What the flow chart does not indicate, however, is that only after a weave has been generated, can the operator select other functions such as 'Transpose'. If a weave is not present, the main program will not make available those functions that are intended for the manipulation of the weave.

The first routine given below, 'Transpose', simply checks the input and passes control to the routines that are to be used for the transposition of the weave. The Transpose function passes control to 'RegTranspose' for Methods 1 and 2 and to 'IrregTranspose' for Method 3.

```
Transpose()
{
    /* Local variables */
     char *t1[1], *t2[1];
     int cnt, width;
     int status, response;

    /* Set up question box */
    t1[0] = "Regular or Irregular Transpose (R/I)";
    cnt = 1;
    width = 4;
    t2[0] = malloc ( width+1);
    t2[0][1] = '\0';
    t2[0][0] = 'R';
    /* Get answers and parse them */
    /* Question displays a window for input */
    Status = Question ( t1, t2, cnt, width );
    if (!status)
      {
      free ( t2[0] );
```

```
    return False;
    }
    response = toupper ( t2[0][0] );
  /* Check the input parameters */
    if (response != 'R' || response != 'I')
        ShowError(R_I);
  /* ShowError displays the appropriate error message */

    if (response == 'R') RegTranspose();
     else if ( response == 'I') IrregTranspose();
  free ( t2[0] );
  return(status);
}
```

'RegTranspose' given below, will transpose the weave grid rather than the draft for the first and second variety. The program first determines whether every cell or only alternate cells (Method 1 or 2) have to be transposed. Thereafter, the number of ends to be transposed needs to be determined. Following input, 'CheckInput' routine checks the request for validity. If the request is valid, the weave grid will be transposed. If, however, the input is incorrect, an error message will be displayed warning the operator of the error. Subsequently, the operator can either change the input or abort the routine.

```
RegTranspose()
{
    /* Local variables */
    char *t1[2], *t2[2];
    int cnt, width;
    int status, response, response1;
    int param1 = 0, param2 = 0;

    /* Set up question box */
    t1[0]   = "Transpose Every Block or Alternate Blocks (E/A)";
    t1[1]   = "# of Cells to Transpose";
    cnt     = 2;
    width   = 4;
```

```
        t2[0]   = malloc ( width+1);
        t2[1]   = malloc ( width=1);
        t2[0] [1] = '\0';
        t2[1] [1] = '\0';
        t2[0] [0] = 'E';
        t2[1] [0] = '2';

        /* Get answers and parse them */
        status = Question ( t1, t2, cnt, width );
        if (!status)
          {
          free ( t2[0] );
          free ( t2[1] );
          return False;
          }
           response = toupper ( t2[0] [0] );
           response1 = atoi ( t2[1] );

        /* CheckInput checks the input.
             If the input is valid, the program will continue.
        */
             CheckInput(R, response, response1);

             param1 = param2 = response1;
if (response == 'E')
  {
    for (loop = 0; loop < weave.h_size; loop += response1)
    {
    for (loop1 = 0; loop1 < weave.v_size; loop1 += response1)
     Rotate(weave.grid, loop, loop1, &param1, &param2, 1);
    }
    }
}
 else if (response == 'A')
    {
    for (loop = 0; loop < weave.h_size; loop += 2*response1)
    {
    for (loop1 = 0; loop1 < weave.v_size; loop1 += response1)
```

```
    {
      Rotate(weave.grid, loop, loop1, &param1, &param2, 1);
    }
    }
      free ( t2[0] );
      free ( t2[1] );
      return(status);
}
```

'IrregTranspose' given below, will transpose the weave grid using the third method (4-end sateen). The operation of the routine is very similar to that of 'RegTranspose'. The program checks the input for validity and will transpose the weave grid only if the request is valid.

```
IrregTranspose()
{
     /* Local variables */
     char *t1[1], *t2[1];
     int cnt, width;
     int status, response;
     int dir = 1;
     int param1 = 0, param2 = 0;

    /* Set up question box */
    t1[0]  = "# of Cells to Transpose";
    cnt    = 1;
    width  = 4;
    t2[0]  = malloc ( width+1 );
    t2[0] [1] = '\0';
    t2[0] [0] = '2';

    /* Get answers and parse them */
    status = Question ( t1, t2, cnt, width );
    if (!status)
      {
      free ( t2[0] );
      return False;
      }
```

```
        response = atoi (t2[0]);

      /* Check the input  */
        CheckInput (I, E, response);

param1 = param2 = response;

      for (loop = 0; loop < weave.h_size; loop += response)
        for (loop1 = 0; loop1 < weave.v size; loop1 += response)
        {
          if (loop >= weave.h_size/2) dir = -1;
           Rotate( weave.grid, loop, loop1, & param1, & param2, dir );
        }
param1 = param2 = 2*response;
Rotate( weave.grid, 2*response, 2*response, &param1, &param2, 1);
      Rotate(weave.grid, 2*response, 0, &param1, &param2, 1 );
        free ( t2[0] );
        return(status);
}
```

The routines that check the input and display error messages are not included since they are not directly related to the transposition algorithms.

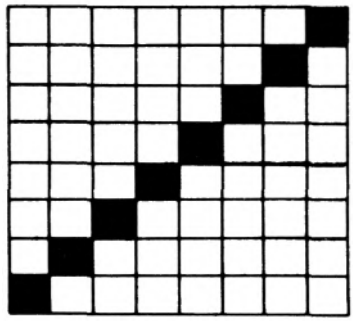


Figure 1.a
A straight draft

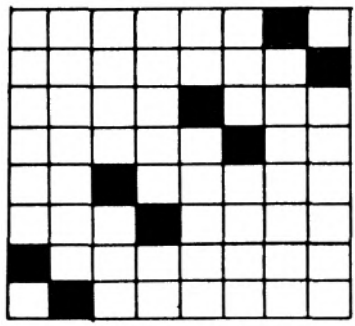


Figure 1.b
Regular transposition in groups of 2 ends (Method 1)

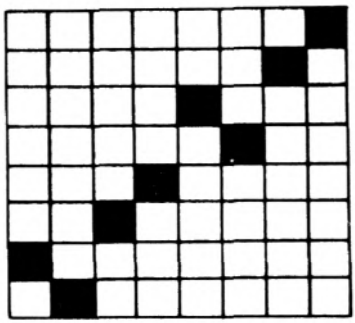


Figure 1.c
Regular and straight transposition in groups of 2 ends (Method 2)

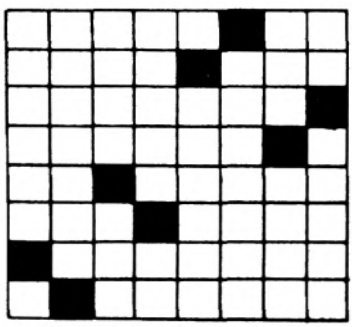


Figure 1.d
4-end sateen transposition in groups of 2 ends (Method 3)

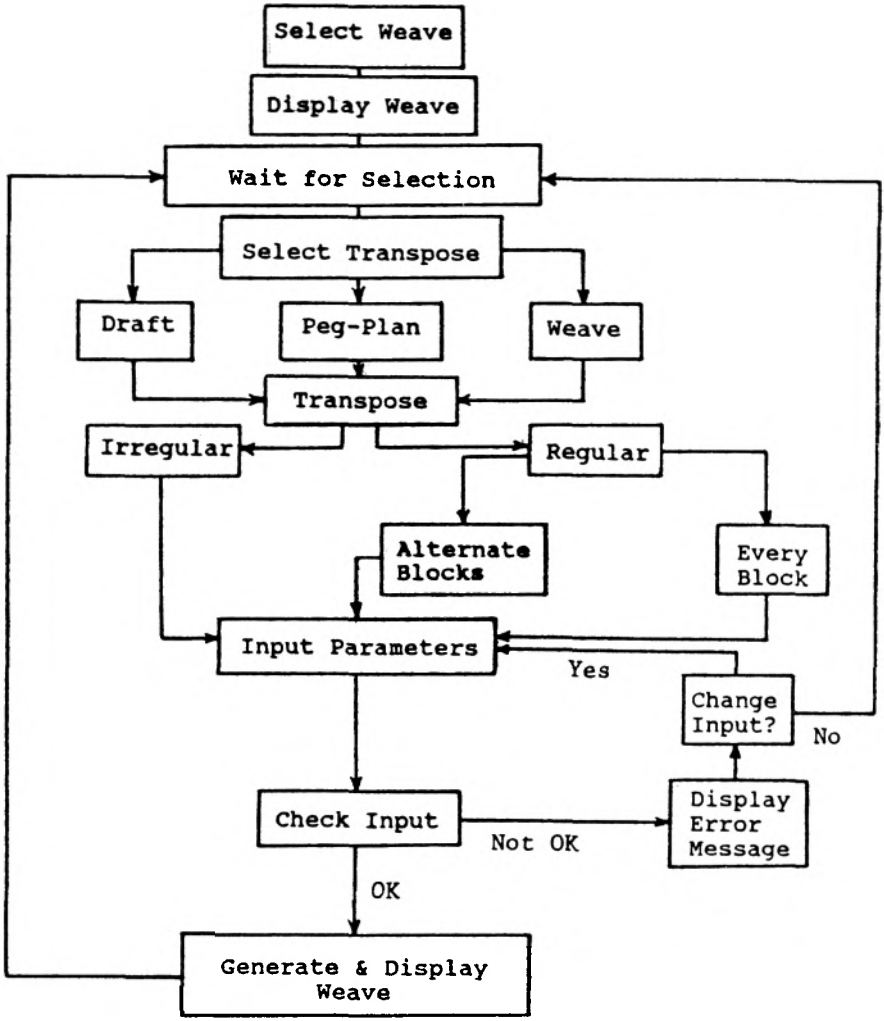


Figure 2
A simplified flow chart for computerized transposition of weaves

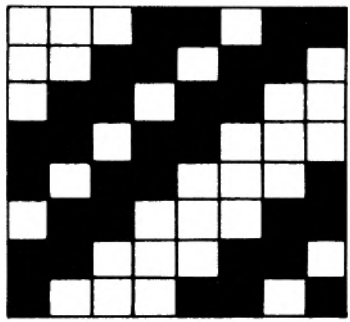
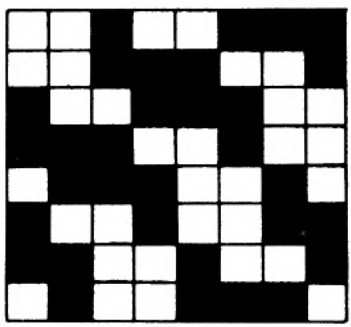


Figure 3.a
A regular twill weave



'Figure 3.b
Transposed twill in groups of 2 ends (Method 1)

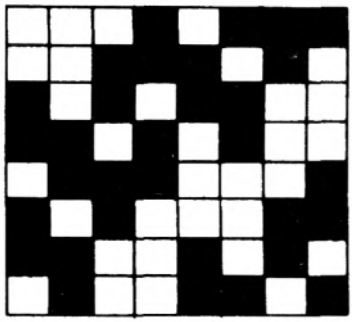


Figure 3.c
Transposed twill in groups of 2 ends (Method 2)

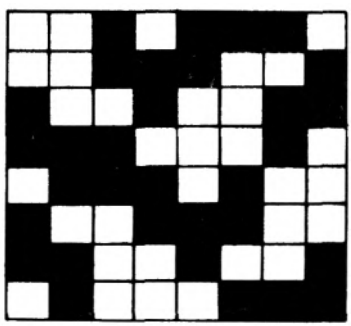


Figure 3.d
Transposed twill in groups of 2 ends (Method 3)