Spatial transformation techniques

Dear Editor:

In a recent paper Fant\textsuperscript{1} presented a technique for spatial transformations. The paper claims that "the technique was developed intuitively" and "formal characterization has proved difficult." We claim that there is a simple formal characterization which we state below and we also provide a fragment of C code implementing Fant's method. We focus on the one-dimensional transformation, since its use for two-dimensional transformation is based on Catmull's\textsuperscript{3} results, namely the expression of two-dimensional transformation as products of two scan-line transforms. The related expressions for the case of linear transformations such as rotation are particularly simple based on the fact that any two-by-two matrix can be expressed as

\[
\begin{bmatrix}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{bmatrix} = \begin{bmatrix} a_{11} - (a_{12}/a_{22})a_{21} & a_{12}/a_{22} \\
a_{21} & a_{22}
\end{bmatrix} \begin{bmatrix} 1 & 0 \\
0 & 1
\end{bmatrix}
\]

Then the application of, say, the rightmost matrix amounts to leaving \(x\) alone, scaling \(y\) by \(a_{22}\) and then shifting by \(a_{21}x\).

What is claimed to be new in the paper is a nonaliasing transformation from the pixel array \(V\)

\[v_1, v_2, \ldots, v_n\]

to a pixel array \(U\) consisting of \(s\cdot n\) elements. For a scaling factor \(s\) less than one, the basic formula used in Fant\textsuperscript{1} is

\[u_j = s \cdot \left( \sum_i a_i v_{(j+m)_n} \right)\]

where

\[\sum_i a_i = \frac{1}{s}\]

and \(m\) is an offset. When the scaling factor exceeds one the formula without "expansion smoothing" is

\[u_j = v_i\text{ when }m \leq i < n\]

\[u_j = a_1 v_i + a_2 v_{i+1}\text{ otherwise}\]

where \(m\) and \(n\) are offsets and

\[a_1 + a_2 = 1\]

It is possible to derive formulas for the offsets in terms of \(s\), but is simpler to state in a single sentence what this operation (and Fant's\textsuperscript{1} method) amounts to:

(Form the input array \(U\) construct a continuous "staircase" function such that \(v_i(t) = v_{i+1} \leq t < v_i\), where \(v_i\) are the locations of the input samples. Then set

\[u_j = \int_{y-j}^{y} v_i(t) dt\]

where \(y\) are the locations of the output samples.

This mapping into a continuous space provides the simple and concise description that is missing from Fant.\textsuperscript{1} It may also be used to test any claims about the originality of the proposed method. The expansion smoothing amounts only to linear interpolation of the output samples. Of course, the continuous formalism is not needed for the implementation, as the following C code implementing the one-dimensional transformation of Fant\textsuperscript{*} demonstrates.

```c
if(factor < 1) {
    ns = (1/factor) - 1;
    a = 0;
    do {
        b = 1. - a;
        c = b * (*ip++);
        if(c < SMALL & ip >= iend) break;
        while (b < ns && ip < iend) {
            c += *ip++;
            b += 1;
        }
    } while(ip < iend);
    a = invf - b;
    c += a * (*ip);
}
```
\[
\begin{align*}
\cdot & \text{op}++ = \text{factor} \cdot c + 0.5; \\
\} \text{ while (ip < iend); }
\end{align*}
\]

else {
  \[
\begin{align*}
  c &= 0; \\
  a &= 0; \\
  \text{do} \\
  \text{if(a) } & | \\
  c &+= a \cdot (\text{ip}); \\
  \cdot & \text{op}++ = c + 0.5; \\
  \} \\
  \text{ns} &= \text{factor} - a; \\
  \text{for (j=0; j < ns; j++) } & | \\
  b &= \text{factor} - \text{ns} - a; \\
  c &= b \cdot (\cdot \text{ip}++); \\
  a &= 1. - b; \\
  \} \text{ while (ip < iend); }
\end{align*}
\]

ip is a pointer to the input array, op is a pointer to the output array, a, b, c, area auxiliary variables, and ns the number of whole pixels mapped to the output.

T.D.S. Duff
T. Pavlidis
AT&T Bell Labs
Murray Hill, NJ


The author replies:

Thank you for your response. The algorithm was indeed developed intuitively and experimentally, and the minds that have worked on the algorithm since its development have been stuck in the realms of algebraic point transforms and projective geometry. The integral calculus didn't occur to us.

It seems to be a simple, straightforward, and elegant characterization of the algorithm, but if I understand the integral correctly, it should be scaled by s to achieve the correct output value. The C code appears to be a correct implementation but it's much more complex than necessary.

The strength and novelty of the algorithm stems from the fact that the interloop can be implemented without expansion smoothing with two adders, one multiply, and one accumulate, and that it still generates nonalised pictures.

Karl M. Fant