Notes Regarding the Dimensions of an Image Grid

Daniel J. Sebald

May 18, 2004

The following notes describe an efficient method for determining the dimensions of an array of pixels when just the locations of individual pixels is know. The application is for the plotting program *gnuplot*. Individual data points are stored internal to *gnuplot*. Therefore, information about the dimensions, $K \times L$, of an image represented in such a way is lost. Hence the need for a method to verify that the points represent a valid image grid when the time comes to display an image. The method requires determining K and L and identifying those points representing all four corner pixels of the image.

Figure 1 shows a hypothetical image grid in its most general form. With the possibility of an image being projected from a 3D space into a 2D space, the grid can be one which is non-orthogonal in the 2D space. That is, the in-line scan direction need not necessarily be at a right angle with the line direction. However, it is assumed that location points $\{\mathbf{p}_i\} \in \mathbb{R}^2$ (a two dimensional vector) are ordered in a scanning fashion. This means that if $N = KL$ is the overall number of points, then $\mathbf{p}_1$ is the first corner, $\mathbf{p}_K$ is the second corner, $\mathbf{p}_{N-K+1}$ is the third corner and $\mathbf{p}_N$ is the fourth corner of the grid. However, this ordering says nothing about the orientations of the grid directions. They remain arbitrary.

The problem here then is summed up by determining K. Knowing K, one can do a sanity check such as ensuring that N/K is a whole number. A more refined check would be to verify that every point of the image grid falls in the proper place. However, this is not an efficient process, so in most instances one may be content with using the four corner pixels to generate the canonical grid when displaying the image. Once K is determined, the two image directions can be analyzed

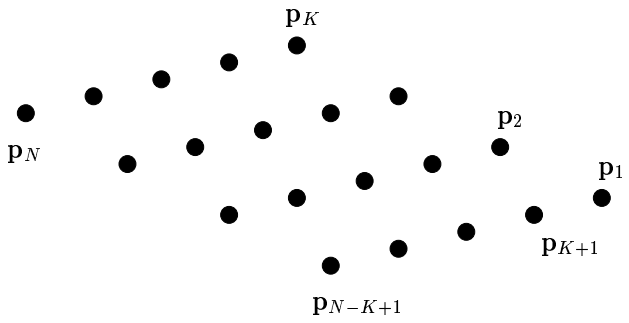

Figure 1: The most general form of a $K \times L$ image grid.

to determine the algorithm for structuring data in a form suitable for an image driver. That is, the image data must be put into an orientation suitable for a driver which expects image data in one specific scanning direction, e.g., the first corner is at the top, left of the image. That restructuring routine is not discussed here.

The problem of determining K, i.e., when the first scan line ends and second scan line begins, is a deceptively difficult problem when allowing for the possibility of some tolerance of numerical error in grid position values. Since points are stored with reference to some (x, y) coordinate system (Cartesian or polar), one approach might be to compute Δx and Δy for the first two points, i.e., $(\Delta x, \Delta y)_i = \Delta \mathbf{p}_i = \mathbf{p}_{i+1} - \mathbf{p}_i$ and continue along the first scan line until that pattern is violated. The first point to fall out of the pattern would be p_{K+1}. However, imagine doing this process of looking along one line only without view of the whole grid, given the fact that one direction could possibly have a scaling orders of magnitude different from the other direction. It is difficult to know what is an allowable tolerance for declaring when a pixel location falls out of the pattern set by $\Delta \mathbf{p}_1$. Part of the reason is that with the general orientation of the image grid, it is a possibility that the first point of the second line could deceptively fall seemingly within the pattern set by the first line if there happened to be enough skew in the grid such that scan lines do not lie "one above the other". Lastly, it can be slightly tedious to implement the above approach from a programming perspective.

So, the intuition is that in determining the end of the first scan line one must include some general information about the overall size of the grid. The question remains *how*? One alternative might be to first compute all of the $\Delta \mathbf{p}_i$ for $i = 1, \ldots, N-1$ and decipher an allowable tolerance from that. However, this is inefficient and again not a straightforward approach. These notes offer a more elegant solution utilizing a hyperplane strategy.

Hyperplane Incorporating Overall Grid Size

A hyperplane is simply a $D-1$ dimensional space which separates a D dimensional space into three parts. It is represented by $(\mathbf{w}, b) \in (\mathbb{R}^D, \mathbb{R})$. The set $\{\mathbf{p} : \mathbf{w}^T\mathbf{p} + b > 0\}$ is one side of the hyperplane, $\{\mathbf{p} : \mathbf{w}^T\mathbf{p} + b < 0\}$ is the other side of the hyperplane, and $\{\mathbf{p} : \mathbf{w}^T\mathbf{p} + b = 0\}$ is the hyperplane itself. In the case of a 2D space, the hyperplane is a line.

If one draws a line passing through the first and fourth corner of the grid, i.e., p_1 and p_N in Fig. 2, it should be apparent that the border points along any given edge of the grid—excluding p_1 and p_N—all lie on one side of the grid. Furthermore, the two edges that intersect either p_1 or p_N have all their points on opposite sides of the hyperplane. Therefore, a simple strategy is to determine the side of the hyperplane that p_2 lies on, then continue computing which side of the hyperplane that successive points lie on. The first point that lies on the side of the hyperplane opposite that on which p_2 lies is the first point of the second scan line, or p_{K+1}. This works for all orientations and arrangements of grids.

Hence, the process of computing the hyperplane is a method incorporating the overall dimen-

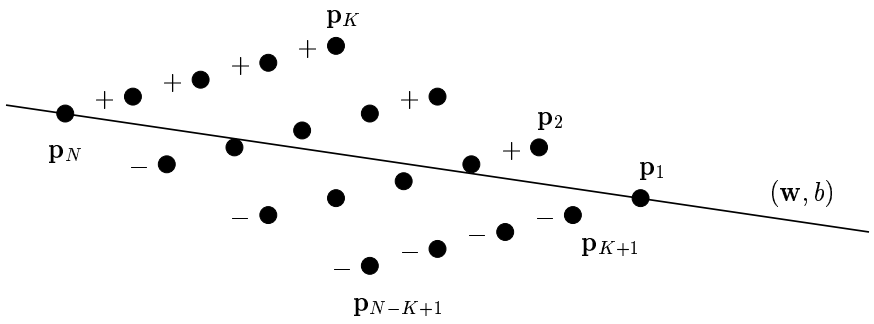

Figure 2: A hyperplane constructed through the first and last points in the image grid. The first line of points lies on one side of the hyperplane. The first point of the second line lies on the opposite side of the hyperplane.

sions of the grid, the intuitive notion referred to earlier. The strategy is very straightforward. In summary:

1. Compute the hyperplane $(\mathbf{w}, b)$ passing through points p_1 and p_N.
2. Determine the sign, σ, the second point generates:
$$\sigma = \text{sgn}(\mathbf{w}^T\mathbf{p}_2 + b).$$
3. Continue with $i = 3, 4, \ldots, N$ until
$$(\mathbf{w}^T\mathbf{p}_i + b)\sigma < 0,$$
at which point K = i - 1.

If a non-uniformly spaced grid not satisfying the hyperplane rule is supplied, some simple sanity checks will reveal that the value K is not valid.

The only slightly obscure detail is computing the hyperplane in the first step above. Since the vector representing a hyperplane is a vector normal to the hyperplane, the vector representing a hyperplane half way between two points, say p_1 and p_N, is simply a vector subtraction of the two points. Call this vector, $\mathbf{w}_\perp$. Then $\mathbf{w}_\perp = \mathbf{p}_N - \mathbf{p}_1$. The orientation of the desired vector in $\mathbb{R}^2$ is $\pm 90^\circ$ from $\mathbf{w}_\perp$. Furthermore, the 90° rotation matrix is

$$\mathbf{R} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}.$$

Hence,

$$\begin{aligned} \mathbf{w} &= \mathbf{R}\mathbf{w}_\perp \\ &= \begin{bmatrix} y_1 - y_N \\ x_N - x_1 \end{bmatrix} \end{aligned} \tag{1}$$

Since either of the two points lie on the hyperplane, choose

$$b = -\mathbf{w}^T\mathbf{p}_1. \tag{2}$$

Then, (1) and (2) constitute step 1 above.

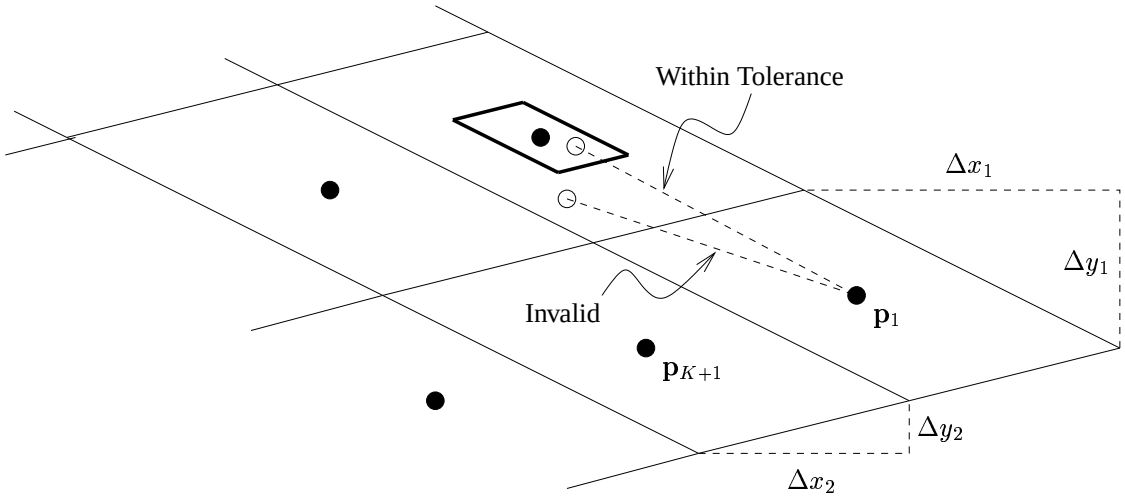

Figure 3: To verify that non-corner pixel locations are within tolerance, a parallelogram proportional to the pixel dimensions is constructed around each expected grid point. (Pixels are represented by faint lines.) If the pixel location given in the data falls outside this tolerance parallelogram, it is declared invalid.

Refined Total Grid Verification

In some instances, it may be desirable to check that every point in the image grid is valid, not just the corner points. To check this, the hyperplane approach may be used again to simplify matters. The basic approach, shown in Fig. 3 is to construct a tolerance parallelogram around the expected location of the point in question. If the position location in the image data for the pixel in question falls outside the tolerance parallelogram the location is declared invalid, i.e., an improper image grid.

If one considers the *difference* between the expected and actual pixel location, the tolerance parallelogram translates to being positioned about the origin. As illustrated in Fig. 4, each of the sides of the parallelogram can be treated as a hyperplane, and the intersection of all four regions on the origin (i.e., pixel center) side of the hyperplane constitutes the convex tolerance region. Say $(\mathbf{w}_{i,j}, b_{i,j})$, $i, j \in \{1, 2, 3, 4\}$, is the hyperplane associated with the line passing through corners i and j of the tolerance parallelogram. For convenience, the side of the hyperplanes on which the origin lies should yield a positive result in the hyperplane equation. This means that the vectors $\mathbf{w}_{i,j}$ point inward when $b_{i,j} > 0$. That is, the origin is at the center, hence $\mathbf{w}_{i,j}^T\mathbf{0} + b_{i,j} = b_{i,j} > 0$.

Make the further observation that opposite sides of the tolerance region are parallel, meaning

$$\mathbf{w}_{4,3} = -\mathbf{w}_{1,2}, \tag{3}$$

$$\mathbf{w}_{3,1} = -\mathbf{w}_{2,4}. \tag{4}$$

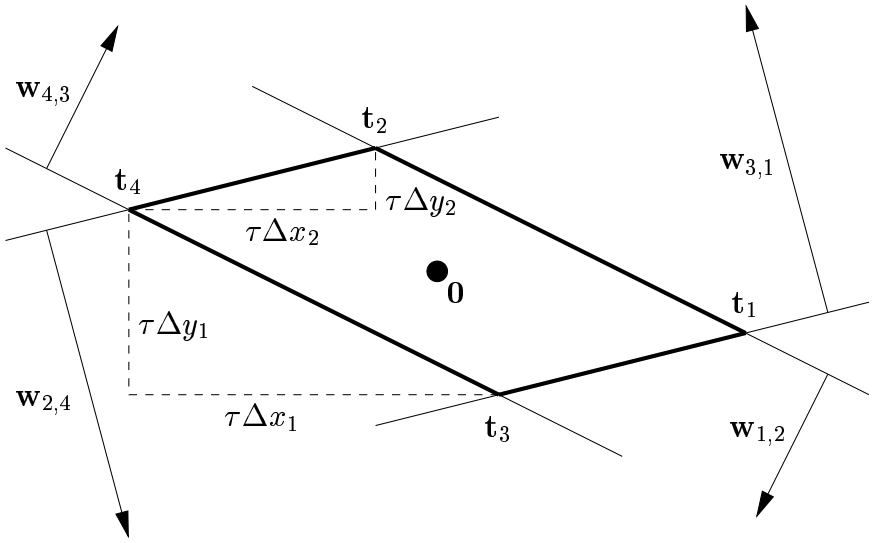

Figure 4: A tolerance parallelogram translated so that it is centered about the origin.

Furthermore, because the parallelogram is centered about the origin,

$$b_{4,3} = b_{1,2}, \tag{5}$$
$$b_{3,1} = b_{2,4}. \tag{6}$$

To derive formulas for testing tolerance, let $\tilde{\mathbf{p}}_i$ be the actual location associated with pixel i, i.e., the data supplied for testing. The expected value, $\mathbf{p}_i$, is that determined from assuming a uniformly sampled grid constructed from $\mathbf{p}_1$, $\mathbf{p}_K$ and $\mathbf{p}_{N-K+1}$. The requirements for a valid position are

$$\begin{aligned}
\mathbf{w}_{1,2}^T(\tilde{\mathbf{p}}_i - \mathbf{p}_i) + b_{1,2} > 0,\\
\mathbf{w}_{2,4}^T(\tilde{\mathbf{p}}_i - \mathbf{p}_i) + b_{2,4} > 0,\\
\mathbf{w}_{4,3}^T(\tilde{\mathbf{p}}_i - \mathbf{p}_i) + b_{4,3} > 0,\\
\mathbf{w}_{3,1}^T(\tilde{\mathbf{p}}_i - \mathbf{p}_i) + b_{3,1} > 0,
\end{aligned}$$

for $i = 1, \ldots, N$. Utilizing (3)–(6) in the above four formulas leads to

$$\mathbf{w}_{4,3}^T(\tilde{\mathbf{p}}_i - \mathbf{p}_i) + b_{4,3} > 0, \tag{7}$$
$$-\mathbf{w}_{2,4}^T(\tilde{\mathbf{p}}_i - \mathbf{p}_i) + b_{2,4} > 0, \tag{8}$$
$$-\mathbf{w}_{4,3}^T(\tilde{\mathbf{p}}_i - \mathbf{p}_i) + b_{4,3} > 0, \tag{9}$$
$$\mathbf{w}_{2,4}^T(\tilde{\mathbf{p}}_i - \mathbf{p}_i) + b_{2,4} > 0, \tag{10}$$

or, combining (7) and (9) and (8) and (10),

$$|\mathbf{w}_{4,3}^T(\tilde{\mathbf{p}}_i - \mathbf{p}_i)| < b_{4,3}, \tag{11}$$
$$|\mathbf{w}_{2,4}^T(\tilde{\mathbf{p}}_i - \mathbf{p}_i)| < b_{2,4}, \tag{12}$$

Hence, (11) and (12) are an easy test for each pixel location in the image grid.

Recall, at this stage, the hyperplane representations are assumed to be those for which constants $b_{4,3}$ and $b_{2,4}$ are positive. Even though there are two orientations for the corners $\mathbf{t}_1$ through $\mathbf{t}_4$ of the tolerance grid shown in Fig. 4 (clockwise and counter-clockwise), the absolute values in (11) and (12) and the fact that the arguments of the absolute values are linear with respect to the hyperplane vectors means that proper orientation of the hyperplanes can be disregarded while observing a minor adjustment in the equations. That is, simply generate hyperplanes through points $\mathbf{t}_4$ and $\mathbf{t}_3$ and points $\mathbf{t}_4$ and $\mathbf{t}_2$, and then test whether

$$
\begin{aligned}
|\mathbf{w}_{4,3}^T(\tilde{\mathbf{p}}_i - \mathbf{p}_i)| &< |b_{4,3}|, \\
|\mathbf{w}_{2,4}^T(\tilde{\mathbf{p}}_i - \mathbf{p}_i)| &< |b_{2,4}|.
\end{aligned}
$$

If not true for all i, the image grid is out of tolerance.

The only remaining detail is computing the hyperplanes. The same construction for a hyperplane between points described in (1) and (2) applies here. By sum or subtraction of vectors

$$
\begin{aligned}
\mathbf{t}_4 &= \begin{bmatrix} \tau(\Delta x_1 + \Delta x_2)/2 \\ \tau(\Delta y_1 + \Delta y_2)/2 \end{bmatrix}, \\
\mathbf{t}_2 &= \begin{bmatrix} \tau(\Delta x_1 - \Delta x_2)/2 \\ \tau(\Delta y_1 - \Delta y_2)/2 \end{bmatrix}, \\
\mathbf{t}_3 &= -\mathbf{t}_2,
\end{aligned}
$$

where τ is the scaling factor for tolerance, *e.g.*, say 0.001, or 0.1%.