

# 3D Object Perception Using Gradient Descent

Leemon Baird<sup>1</sup> and Patrick Wang  
College of Computer Science  
Northeastern University, Boston, MA 02115

## ABSTRACT

A new algorithm is presented for interpreting two-dimensional (2D) line drawings as three-dimensional (3D) objects without models. Even though no explicit models or additional heuristics are included, the algorithm tends to reach the same 3D interpretations of 2D line drawings that humans do. The algorithm explicitly calculates the partial derivatives of Marill's Minimum Standard Deviation of Angles (MSDA) with respect to all adjustable parameters, and follows this gradient to minimize SDA. For an image with lines meeting at  $m$  points forming  $n$  angles, the gradient descent algorithm requires  $O(n)$  time to adjust all the points, while Marill's method required  $O(mn)$  time to do so. Experimental results on various line drawing objects show that this gradient descent algorithm running on a Macintosh II is one to two orders of magnitude faster than the MSDA algorithm running on a Symbolics, while still giving comparable results.

Keyword: minimum standard deviation of angles, gradient descent, 3D object recognition and perception.

## 1. INTRODUCTION

Interpretation of 2D images as 3D objects is a difficult problem and has attracted wide attention among computer vision and pattern recognition researchers for many years[2,3,4]. This paper concentrates on a special kind of 2D images called line drawings, also known as "noble class" of pictures by Sugihara[11], in which the phenomenon of 3D interpretation holds for many interesting images in realistic world. For a more recent survey of line drawing image processing by computers, see Marill[7,8,9]. Generally speaking, a single line drawing shows only one view of a 3D object, and is therefore ambiguous. The ambiguity is often resolved through the use of a set of stored 3D models[5, 10]. Such approaches require prior knowledge of what objects are likely to appear in a drawing, and

---

<sup>1</sup>Current Address: WL/AAAT, Wright-Patterson AFB, OH 45433-6543

may not give good interpretations of novel drawings which do not correspond to any of the stored models.

Marill [7,8] has proposed a simple approach for interpreting line drawings. This approach requires no models, and it uses only one heuristic to generate a 3D wire-frame object from a 2D line drawing. A given 3D interpretation is considered less likely to be correct if some angles between the wires are much larger than others. Specifically, of all the 3D models consistent with a given 2D drawing, the preferred interpretation is the one with the least Standard Deviation of Angles (SDA).

The algorithm proposed in [7,8,9] uses this Minimum SDA (MSDA) principle to find 3D interpretations of drawings. The motivation behind this is that it is believed that MSDA tends to produce the most stable 3D object of all possible candidates that are derivable from 2D line drawing images. It can interpret a wide range of line drawings and seems to consistently generate the same interpretations that a human does, even without any explicit models. However, the computational complexity of the algorithm made it impractical for any but the simplest drawings.

The new algorithm presented here implements the MSDA principle through a more direct gradient descent algorithm. It is a faster algorithm, in that it reduces the time complexity from  $O(m.n)$  to  $O(m+n)$ , where  $m$  is the number of line segments and  $n$  is the number of angles in the image, yet gives the same results. A preliminary revision of this paper was presented in [1].

## **2. MSDA GRADIENT DESCENT ALGORITHM**

This algorithm transforms a line drawing into a wire-frame object. The line drawing is represented as a set of  $(x,y)$  points, and a set of lines connecting the points. The drawing is assumed to be one view of a 3D wire-frame object. Each of the points  $(x,y)$  in the 2D drawing corresponds to a point  $(x,y,z)$  in the 3D object generated. The  $x$  and  $y$  coordinates of the point on the object are constrained to be equal to the  $x$  and  $y$  coordinates of the point in the drawing. A line connecting two  $(x,y)$  points is assumed to correspond to a wire connecting the two corresponding  $(x,y,z)$  points. The problem then is to find the  $z$

coordinates of all the points in the drawing, thus generating a reasonable wire-frame interpretation of that drawing.

Consider the case where there is a line between two points  $a$  and  $b$ , and another line between points  $b$  and  $c$ , then the angle, points and vectors can be labeled as in the diagram below.

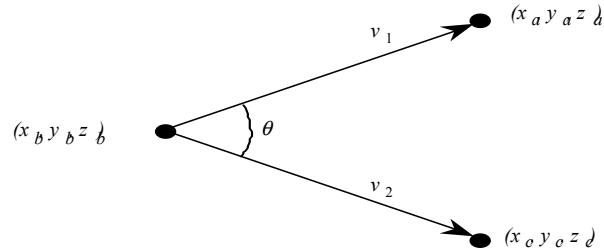


Figure 1. Angle formed by two lines

The angle between the two lines can be calculated from the coordinates of the three points as follows:

$$\theta = \cos^{-1} \left( \frac{v_1 \cdot v_2}{|v_1| |v_2|} \right) = \cos^{-1} \left( \frac{(x_a - x_b)(x_c - x_b) + (y_a - y_b)(y_c - y_b) + (z_a - z_b)(z_c - z_b)}{\sqrt{(x_a - x_b)^2 + (y_a - y_b)^2 + (z_a - z_b)^2} \sqrt{(x_c - x_b)^2 + (y_c - y_b)^2 + (z_c - z_b)^2}} \right) \quad (1)$$

The Standard Deviation of Angles (SDA) for all the angles in the entire picture is:

$$\text{SDA} = \frac{1}{n} \sqrt{n \sum_{\theta} \theta^2 - \left( \sum_{\theta} \theta \right)^2} \quad (2)$$

where the summations are over each angle where two lines in the picture meet. The partial derivative of the SDA with respect to a given  $z$  coordinate  $z_i$  is:

$$\frac{\partial \text{SDA}}{\partial z_i} = \frac{n \sum \theta \frac{\partial \theta}{\partial z_i} - \left( \sum \theta \right) \sum \frac{\partial \theta}{\partial z_i}}{n \sqrt{n \sum \theta^2 - \left( \sum \theta \right)^2}} \quad (3)$$

The partial derivative of each angle with respect to a given  $z_i$  is nonzero only if  $z_i$  is one of the three points forming that angle. If  $z_i$  is at point  $a$  in the diagram above, then the derivative is:

$$\frac{\partial q}{\partial z_a} = \frac{\mathbf{v}_1 \cdot \mathbf{v}_2 (z_a - z_b) - |\mathbf{v}_1|^2 (z_c - z_b)}{|\mathbf{v}_1|^2 \sqrt{|\mathbf{v}_1|^2 |\mathbf{v}_2|^2 - (\mathbf{v}_1 \cdot \mathbf{v}_2)^2}} \quad (4)$$

and if  $z_i$  is at point  $b$  in the diagram above, then the derivative is:

$$\frac{\partial q}{\partial z_b} = \frac{\mathbf{v}_1 \cdot \mathbf{v}_2 |\mathbf{v}_1|^2 (z_b - z_c) + \mathbf{v}_1 \cdot \mathbf{v}_2 |\mathbf{v}_2|^2 (z_b - z_a) + |\mathbf{v}_1|^2 |\mathbf{v}_2|^2 (z_a - z_b + z_c - z_b)}{|\mathbf{v}_1|^2 |\mathbf{v}_2|^2 \sqrt{|\mathbf{v}_1|^2 |\mathbf{v}_2|^2 - (\mathbf{v}_1 \cdot \mathbf{v}_2)^2}} \quad (5)$$

Each  $x$  and  $y$  coordinate is given by the 2D line drawing. The goal is to find  $z$  values for which the SDA is at a local minimum. To find these  $z$  values by gradient descent the  $z$ 's should be repeatedly updated according to:

$$z_i \leftarrow z_i - \alpha \frac{\partial \text{SDA}}{\partial z_i} \quad (6)$$

where  $\alpha$  is a small positive constant regulating the step size of the gradient descent. The partial derivative in (6) can be implemented by substituting (4) and (5) into (3).

Inspection of the above equations shows that if there are  $m$  points and  $n$  angles in the drawing, then the gradient descent algorithm takes  $O(n)$  operations for one iteration. Marill's algorithm performed a similar function on each iteration. It generated  $2m$  possible objects from the current object by adding or subtracting a small amount from each  $z$  coordinate. It calculated the SDA for each of them and picked the minimum SDA. The object with the minimum SDA then became the current object for the

next iteration. Calculating the SDA for  $m$  objects in a straightforward manner would take  $O(mn)$  operations. A more careful implementation would take  $O(n)$  operations. While the gradient algorithm can change all of the  $z$  coordinates in each iteration, the original algorithm requires at least  $m$  iterations in order for each  $z$  value to be changed at least once. It would take  $O(mn)$  operations to run through  $m$  iterations. Typically, large line drawings have a large number of lines, but relatively few lines meeting at each point. Therefore the number of angles  $n$  tends to be proportional to the number of points  $m$ . If this is the case, then the original algorithm takes  $O(n^2)$  operations to run through  $m$  iterations. It is difficult to predict exactly how one iteration of the gradient algorithm compares to  $m$  iterations of the original algorithm. Although it is hard to predict the rate of convergence for either algorithm, this analysis would tend to suggest that the gradient algorithm reduces the convergence time for large line drawings from  $O(n^2)$  to  $O(n)$ .

### 3. MSDA GRADIENT DESCENT RESULTS

To test the algorithm, a program in C was written and was implemented on Macintosh II, and executed on several line drawing patterns adapted from [3,4,5], including even stairs, uneven stairs, a cube, and a table, as shown in Figure 2.

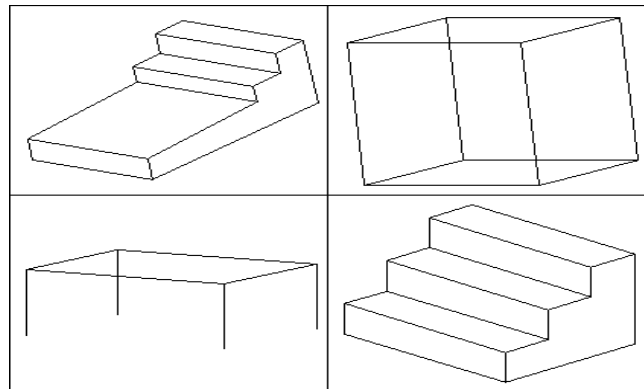


Figure 2. Four line drawings used as inputs

When the algorithm was presented with the drawing of the uneven stairs, it quickly converged as shown in Figure 3 with several rotations about a vertical axis.

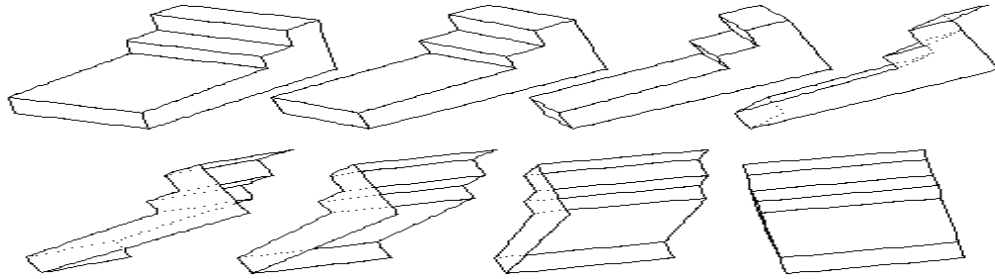


Figure 3. Object generated from uneven stairs (8 rotations)

The above figure shows the output from the algorithm, with some lines changed to dotted lines manually to aid in visualizing the result. Clearly the algorithm reached the same 3D object that humans "see" when looking at the original line drawing. Note that the original drawing had no right angles, yet the algorithm correctly interpreted them as right angles in 3D space. Consider a drawing consists only of a parallelogram. There are an infinite number of objects consistent with that drawing, but the one with the minimum SDA is a tilted rectangle. Similarly, if the drawing consists only of some triangle, then the MSDA object will be a tilted equilateral triangle.

For each of the objects in Figure 2, the gradient descent algorithm arrived at the same answer as Marill's method, but was much faster. The following table compares the results for the gradient algorithm on a Macintosh II with the published results of Marill's algorithm on a Symbolics. Notice that the experimental results are rather consistent with the theoretical conclusions drawn in the previous section. In implementation, however, it is also possible to obtain slightly different results, depending on hardware(e.g. computer) and software(e.g. languages), being used in the experiment.

Patterns	Numer of vertices (nodes)	MSDA Algorithm (Symbolics)	Gradient Descent (Macintosh II)
Even Stairs	15	320 sec (610)	9 sec (35)

Uneven Stairs	15	770 sec (1780)	7 sec (18)
Cube	8	5~57sec (13~120)	5 sec (112)
Table	8	5~57sec (13~120)	6 sec (140)

Table 1. Speed Comparisons and number of iterations (in parenthesis)

The times in table 1 compare the new algorithm with Marill's algorithm, which was implemented in LISP on a Symbolics, and the gradient descent algorithm was implemented in C on a Macintosh IIcx. The number of iterations of execution is shown in each pair of parenthesis. The former algorithm shows time for calculating the object (plus, for the uneven stairs, time to compare the objects, which was reported to be a small part of the total). The time for the gradient descent algorithm is the time to calculate the object, while also rotating, scaling, and drawing four views of the current object on the screen, 12 times a second. Without the animation, the gradient algorithm would have been only slightly faster.

Because of their different time complexities, the gradient algorithm is especially useful for large problems. Even in the fairly small examples shown here, it was better by one or two orders of magnitude. It also is a better approximation to true hill climbing. The earlier algorithm, with  $n$  points, only looks in  $n$  directions to see which is best. The gradient algorithm effectively looks in every direction to see which way is best, then takes a small step in that direction.

One good feature in Marill's algorithm which was not implemented here was a variable step size. It makes sense to try large changes at first, and then make smaller changes as the object approaches its final value. Future research might implement this in the gradient algorithm too, slowly decreasing the step size on each iteration. This is theoretically necessary in order to converge to a local optimum, and might be useful in practice.

Local minima can be a problem with any gradient algorithm. Both the algorithm presented here and the original algorithm were able to correctly recognize a line drawing of a cube. If the algorithm is run with different random initial values for the  $z$  coordinates, then it is possible for the algorithm to get stuck in a local minimum where the SDA is increased by any small changes, yet could be decreased by a large change to the  $z$  coordinates. The figures below show two runs of the gradient algorithm. The line drawing is the same in both cases, only the random number seed is different.

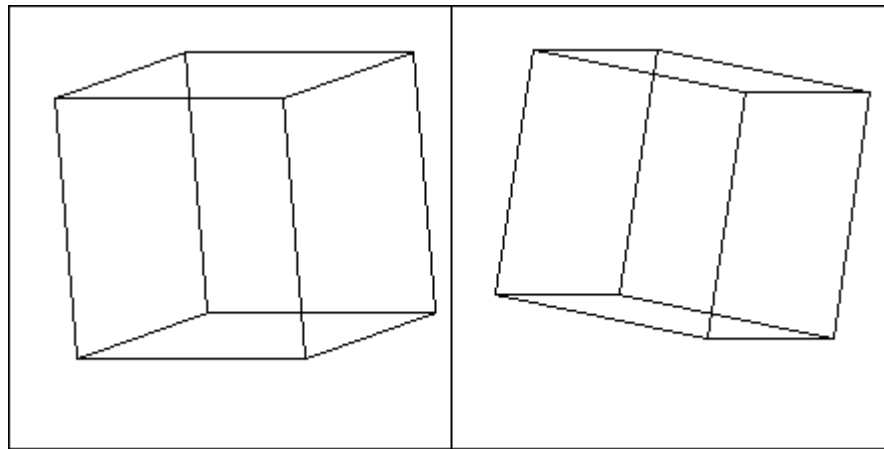


Figure 4. Front and side view of a good local minimum

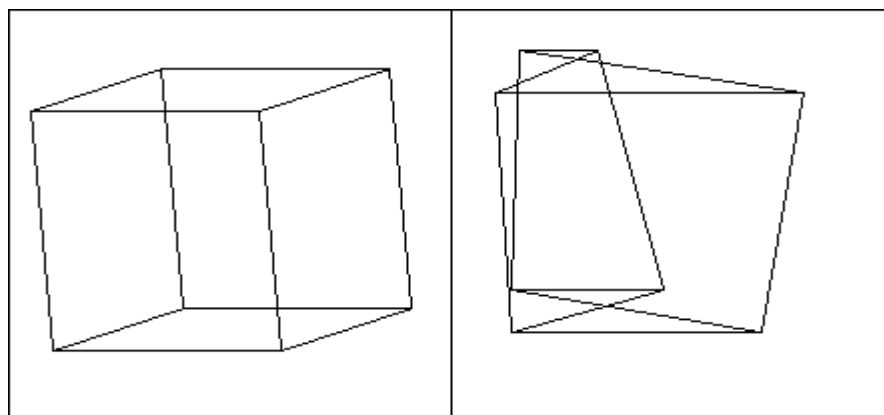


Figure 5. Front and side view of a poor local minimum

In both of the above cases, the figure on the left is the line drawing which was the input to the algorithm,. This is also the front view of the final object generated. The picture on the right is a side view of that object. The object starts almost flat, and then unfolds during the gradient descent process. As the above figure shows, it can become tangled while unfolding. This represents a local minimum, which cannot improve the SDA. Two pairs of points must be reversed in order to achieve the correct answer. In this case, the correct answer is a cube, which has an SDA of 0.

One possible solution to this problem would be to start with several different initial conditions, perform the gradient descent from each one, and then pick the final object with the minimum SDA. Another approach would be some kind of simulated annealing. In that case, the algorithm would include a random component so that sometimes the SDA would increase instead of decreasing. Over time the process would be cooled, making the SDA less likely to increase.

#### **4. DISCUSSION AND CONCLUSION**

In this paper, we have presented an algorithm that can interpret 2D line drawing images as 3D polyhedral objects in linear time. In the literature, Marill's minimum standard deviation of angles principle is very useful in interpreting line drawings but takes polynomial time [7,8,9], which make testing more complicated images extremely difficult if not impossible. An optimization-based approach proposed by Leclerc and Fischler [6] improved from Marill's can handle more general line images. but still takes polynomial time complexity. The gradient descent algorithm presented here takes linear time and is a much more efficient way than those in the literature, while the results maintain the same quality. It is also a more faithful implementation of hill climbing, while running much faster.

The examples illustrated in Section 3 are basically polynomials of  $90^\circ$  's. Figures 6-7 shows that actually our algorithm can also work for more general patterns of arbitrary angles except  $180^\circ$ 's, which normally result in bigger distortions due to MSDA.

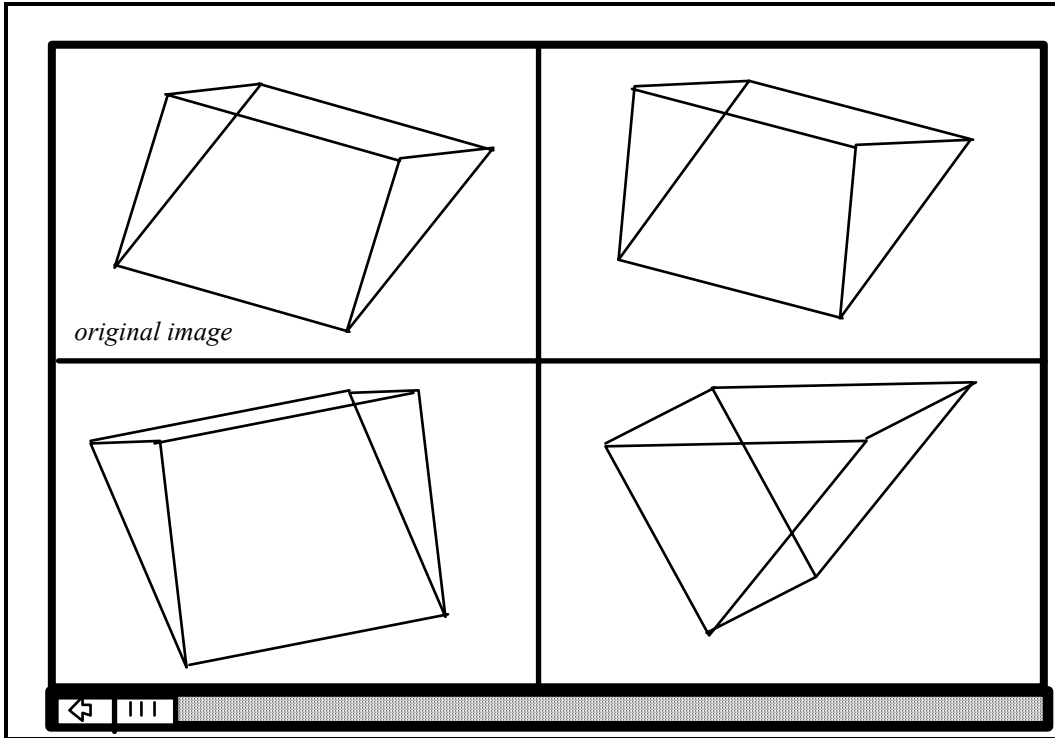


Figure 6. A prism and its rotations

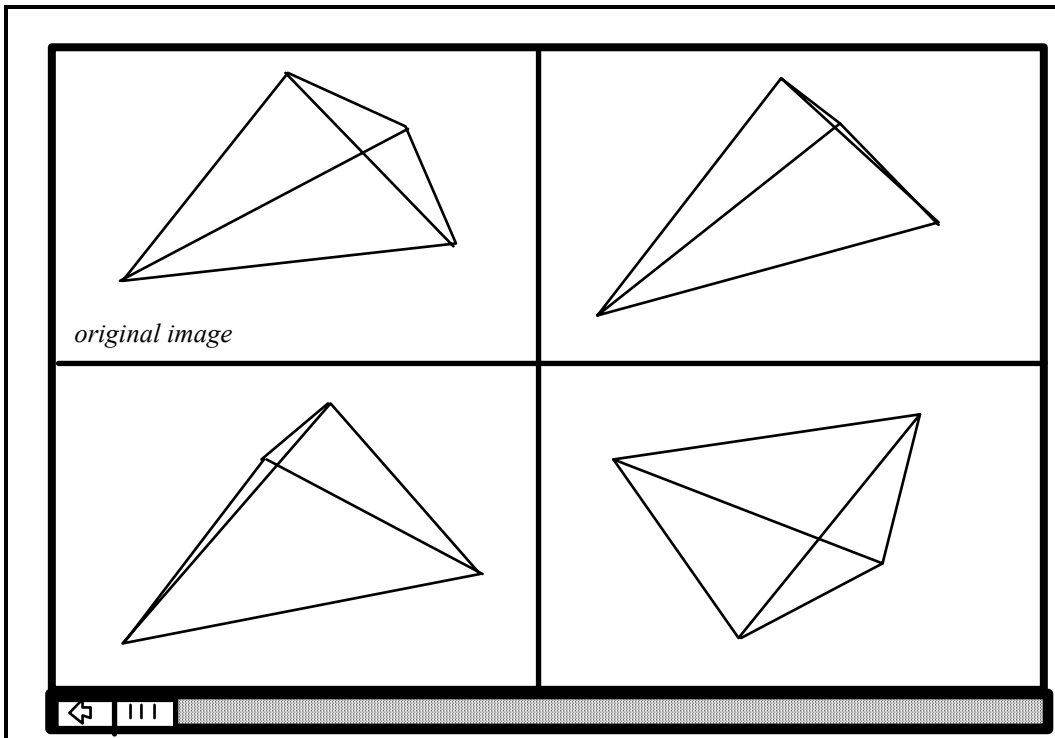


Figure 7. A pyramid and its rotations

The following figure shows a comparison of different methods in the literature.

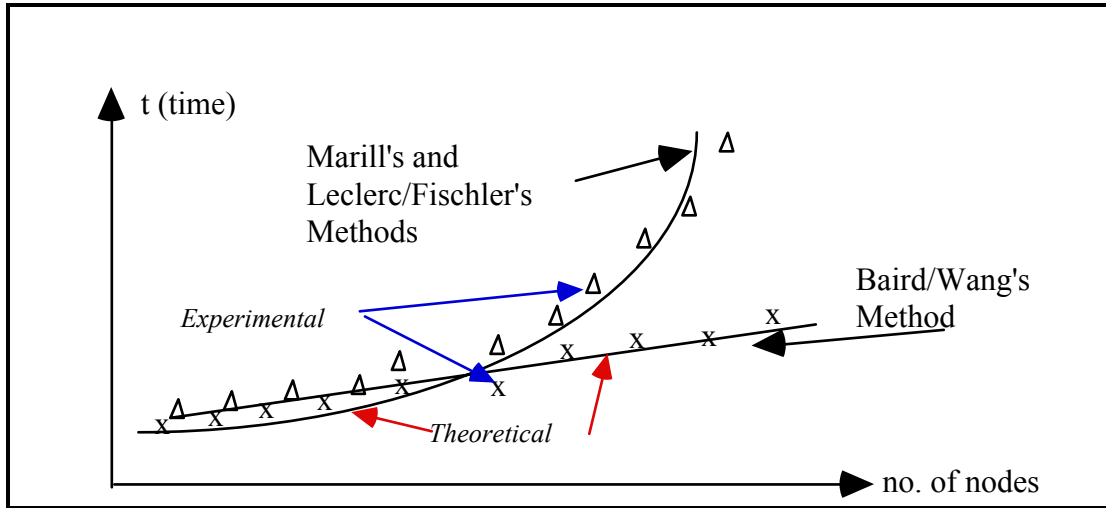


Figure 8, Comparison of methods

## 5. ALGORITHM PSEUDOCODE

The following is pseudocode for the two algorithms presented here.

### MSDA Gradient Descent Algorithm

Inputs: – The  $(x,y)$  coordinates of all the points in the line drawing where multiple line segments are connected.– A list of pairs of points are connected by line segments.

Outputs:– The  $z$  coordinate of each point, thus describing a 3D interpretation of the 2D line drawings.

Initialize  $z$  coordinates to small random values (corresponding to an almost flat wire-frame model).

#### **Loop**

Calculate the gradient: the partial derivative of the SDA with respect to each  $z$  coordinate.

Change each  $z$  coordinate by a small amount proportional to the negative of the gradient.

**Until** the gradient is zero

The coordinates for the line drawings interpreted in this paper and in [1] are given below. For each figure, the first array lists all of the points, and the second array gives the lines connecting the points. For example, in the cube, point number 0 is at location  $(-2.89,-1.62)$  and point number 1 is at location  $(.57,-1.62)$ , and there is a line connecting points 0 and 1.

```
/* Cube with 8 points and 12 lines */
```

```
double points[num_points][2] =  
    {-2.89,-1.62, 0.57,-1.62, 0.92,2.32, -2.54,2.32,  
     -0.92,-2.32, 2.54,-2.32, 2.89,1.62, -0.57,1.62};  
int lines[num_lines][2] =  
    {0,1, 1,2, 2,3, 0,3, 4,5, 5,6, 6,7, 4,7, 0,4, 1,5, 2,6, 3,7};
```

```
/* Stairs with 15 points and 21 lines */
```

```
double points[num_points][2] =  
    {-2.4,-1.45, .14,2.07, -.71,1.71, -.71,.9, -1.56,.54,  
     -1.56,-.28, -2.4,-.63, .78,-2.79, 3.32,-1.72, 3.32,.73,  
     2.47,.37, 2.47,-.45, 1.63,-.8, 1.63,-1.62, .78,-1.98};  
int lines[num_lines][2] =  
    {1,2, 2,3, 3,4, 4,5, 5,6, 6,0, 7,8, 8,9, 9,10, 10,11, 11,12,  
     12,13, 13,14, 14,7, 0,7, 1,9, 2,10, 3,11, 4,12, 5,13, 6,14};
```

```
/* Uneven stairs with 15 points and 21 lines */
```

```
double points[num_points][2] =  
    {-15.51,12.74, -13.76,14.99, -14.08,14.78, -14.02,14.45,  
     -14.34,14.24, -14.28,13.99, -15.57,13.07, -14.11,12.41,  
     -12.18,13.66, -12.35,14.67, -12.68,14.46, -12.62,14.13,  
     -12.94,13.92, -12.88,13.67, -14.17,12.75};  
int lines[num_lines][2] =  
    {1,2, 2,3, 3,4, 4,5, 5,6, 6,0, 7,8, 8,9, 9,10, 10,11, 11,12,  
     12,13, 13,14, 14,7, 0,7, 1,9, 2,10, 3,11, 4,12, 5,13, 6,14};
```

**/\* Table with 8 lines and 8 points \*/**

```
double points[num_points][2] =  
    {-0.67,1.5, 3.67,1.07, 1.67,.47, -2.67,.9,  
    -0.67,-.47, 3.67,-.9, 1.67,-1.5, -2.67,-1.07};  
int lines[num_lines][2] =  
    {0,1, 1,2, 2,3, 3,0, 0,4, 1,5, 2,6, 3,7};
```

**/\* Prism with 9 lines and 6 points \*/**

```
double points[num_points][2] =  
    {17,2, 9,0, 6,5, 14,7, 15,2, 7,0};
```

```
int lines[num_lines][2] =  
    {2,3, 3,4, 1,2, 0,3, 0,1,  
    0,4, 1,5, 2,5, 4,5};
```

**/\* Pyramid with 6 lines and 4 points \*/**

```
double points[num_points][2] =  
    {17,2, 14,7, 20,5, 19,3};
```

```
int lines[num_lines][2] =  
    {0,1, 0,2, 0,3, 1,2, 1,3, 2,3};
```

**/\* Pentagonal Prism with 15 lines and 10 points \*/**

```
double points[num_points][2] =  
    { 0.000,0.000, 3.460,0.000, 4.530,3.290,  
    1.730,5.320, -1.070,3.290, 1.970,0.700,  
    5.430,0.700, 6.500,3.990, 3.700,6.020,  
    0.900, 3.990 };
```

```
int lines[num_lines][2] =
{ 0, 1, 1, 2, 2, 3, 3, 4, 0, 4,
5, 6, 6, 7, 7, 8, 8, 9, 9, 5,
0, 5, 1, 6, 2, 7, 3, 8, 4, 9 };
```

```
/* Octahedron with 12 lines and 6 points */
```

```
double points[num_points][2] =
{ 0.920,2.310, -2.540,2.310, -0.570,1.610,
2.890,1.610, 0.180, -1.280, 0.180,5.200, };
int lines[num_lines][2] =
{ 0, 1, 1, 2, 2, 3, 0, 3, 0, 4,
1, 4, 2, 4, 3, 4, 0, 5, 1, 5,
2, 5, 3, 5 };
```

```
/* Pyramid2 with 8 lines and 5 points */
```

```
double points[num_points][2] =
{ 0.000,0.000, 1.000,0.000, 1.000,1.000,
0.000,1.000, 0.500,0.500 };
```

```
int lines[ num_points][2] =
{ 0, 1, 1, 2, 2, 3, 0, 3, 0, 4,
1, 4, 2, 4, 3, 4 };
```

## 6. ACKNOWLEDGEMENTS

This work was sponsored in part by a Draper Fellowship from The Charles Stark Draper Laboratory, Inc., and by the U.S. Air Force as well as the College of Computer Science, Northeastern University. The authors are thankful to their generosity and support. They would also like show their appreciations to reviewers' comments and suggestions, which lead to a better presentation of this paper.

## 7. BIBLIOGRAPHY

- 1.L.Baird and P.S.P.Wang, "3-D Object Recognition Using Gradient Descent and the Universal 3-D Array Grammer.,"*SPIE*, v1607, Intelligent Robotic and Computer Vision, (1992) 711-719.
- 2.R. Basri, "Viewer-centered representation in object recognition - a computational approach", in Handbook of Pattern recognition and Computer Vision (ed. by C.H. Chen, L.F. Paul and P.S.P. Wang), WSP(1993) pp 863-882
- 3.I.Chakravarty and H.Freeman, "Characteristic views as a base for 3D object recognition", *SPIE Robot Vision* (1982) 37-45.
- 4.R.T.Chin and C.R.Dyer, "Model-based recognition in robot vision", *ACM Computing Surveys*, v18, n1, (1986)67-108.
- 5.S.J.Dickinson, A.P.Pentland and A.Rosenfeld, "3D shape recovery using distributed aspect matching", *IEEE-PAMI*, v-14, n2(1992)174-198.
6. Y,G, Leclerc and M.A. Fischler, "An Optimization-Based Approach to the Interpretation of Single Line Drawings as 3D Wire Frames", *I.J. of Computer Vision*, 9:2, 113-136 (1992)
- 7.T Marill, "Emulating the Human Interpretation of Line-Drawings and 3-D Objects", *IJCV*, 6:2 , 147-161 (1991)
- 8.T.Marill, "Computer Perception of Three-Dimensional Objects", MIT A.I.Memo 1136, August 1989.
- 9.T. Marill, "Recognizing Three-Dimensional Objects Without the Use of Models," MIT A.I. Memo 1157, September, 1989.
10. S.Ullman and R.Basri, "Recognition by linear combination of models", *IEEE-PAMI*, v13, n10, (1991) 992-1006
11. K.Sugihara, *Machine Interpretation of Line Drawings*, MIT Press, Cambridge, 1986

[author's homepage](#) and [pwang@ccs.neu.edu](mailto:pwang@ccs.neu.edu)