

Why We See Three-Dimensional Objects: Another Approach

Eric W. Brown, M.S.
E-mail: *feneric@ccs.neu.edu*
Copyright © 1994

Abstract

This paper considers the work done by Thomas Marill in a series of papers on the recognition of two-dimensional wire-frame figures as three-dimensional objects without the use of models. Marill discovered that if one minimizes the standard deviation of the angles found at each vertex of the figure, the likelihood of the computer interpretation of the figure matching the human interpretation is much higher than might be expected a priori. Here it is observed that the human mind's tendency to simplify inputs and find patterns even where there are none might be at least partly responsible for the observed phenomenon. It is conjectured that if this is indeed the case, it should be possible to get similar behavior from minimizing the standard deviation of other features. In particular, segment length presents itself as being an excellent choice of a test feature - it is very different from angles and is less computationally intensive. Thus another approach is considered: minimum standard deviation of segment magnitudes is explored in lieu of minimum standard deviation of angles.

Marill's original experiment is then carefully repeated with several additional figures that were deliberately chosen not to have all equal angles. The experiment is described in detail, and all the failures of Marill's algorithm are carefully studied and explained. The problem of straight angles is touched upon and the difficulties of solving it as a special case are briefly discussed.

A new program is then written to minimize the standard deviation of segment magnitudes instead of minimizing the standard deviation of angles. This program is run on the same test figures as the original algorithm. Its successes and failures are noted and explained, and its behaviors are studied.

The results of the two algorithms are then compared and the differences noted. It is of particular interest that the two algorithms have different areas of failure, suggesting that a combined algorithm should be able to produce better results than either one alone. This and some other avenues of future work are suggested.

Finally, some comments about the basic behaviors of both algorithms are made.

Keywords

object recognition and interpretation, 3D pattern recovery from 2D images, Minimum Standard Deviation of Angles (MSDA), Minimum Standard Deviation of Segment Magnitudes (MSDSM)

Introduction

The interpretation of two-dimensional wire-frame diagrams as three-dimensional objects in a human manner is a topic that has received much attention and study. Restricting the domain of the problem to wire-frame pictures simplifies the matter enough to be explored in a solid manner while still maintaining enough generality to provide insight into the greater problem of interpreting the full set of two-dimensional pictures as three-dimensional objects. Many methods of interpretation utilized models and thus were restricted to only identifying a fixed number of different objects. Thomas Marill pioneered a novel approach that does not require models¹; rather it simply tries to return the three-dimensional object that minimizes the standard deviation of its angles (*M.S.D.A.*). He experimented with this method in several papers^{1, 2, 3} and other authors followed suit^{4, 5, 6}. He hypothesized³ that this algorithm worked because the human mind has an in-built compression system of sorts; it will always store the data in a minimal fashion.

Theory

It seems reasonable to assume that while this is possible, it is not the only explanation. Another possibility is that the human mind attempts to simplify all of its inputs, building patterns even where there are none. If Marill's minimization of standard deviation of angles approach works because of this tendency, there is a likelihood that other similar approaches would also work. In particular the minimization of standard deviation of segment magnitudes (*M.S.D.S.M.*) suggests itself as a less computationally intensive alternative.

M.S.D.A. Experimental Setup

In principle it is not too hard to test this theory. In practice, a few real-world problems (such as differences between independent computer platforms) presented themselves. The program written by Baird & Wang for their paper⁵ seemed nearly ideal as a cornerstone upon which to build a test program, but it was written for the Macintosh platform. With some effort, the program was converted to compile and build using standard X-Windows and GNU's gcc and could then be built and run on virtually any UNIX box. The program was additionally tested after being built with Sun's acc compiler, and it was carefully linted and made clean. The program was enhanced to take advantage of some of the features of its new environment and to provide more user feedback and current result information, and to read its shape information at run-time so that new shapes could be added without requiring a rebuild. This exercise unearthed a heretofore unmentioned behavior of the Marill algorithm: it is very sensitive to initial conditions. It is somewhat chaotic: tiny differences in its initial input (such as the ones resulting from the use of two different random seeds or the random number generators on two different platforms) can mean the difference between a successful run and an unsuccessful one. The sensitivity of the assorted figures to these tiny differences of initial conditions seemed to vary from figure to figure; no detailed analysis was done on this behavior but it may be an interesting avenue for future work. Certainly it is advisable that future Marill-type algorithms make hidden initial conditions (such as the ones resulting from different random seeds) explicit.

Failures of M.S.D.A.

In the cases where Marill's algorithm failed, the results were still easily explained. There were two primary causes of failure: the gradient descent bottomed out in a local minima rather than the global one, or the M.S.D.A. for the figure actually produced false results.

A failure in the gradient descent resulted in a tangled figure that often contained obfuscated, non-planar faces (see Figure 1, "Examples of Tangles"). This was by far the most common problem with the test set of figures, and this was what was most sensitive to initial conditions. The difference of a random seed or an internal step size or method of gradient descent had large impact on the results; a particular implementation on a particular machine could very easily be tweaked to give excellent results for a specific test set while still having only mediocre results on a more general test set. While tangles were responsible for many failures, they are really not a serious problem. Expectations are that the additional psychological assumptions suggested by Leclerc & Fischler⁴ will virtually eliminate tangles as these obfuscations seem to universally result from solutions involving non-planar faces and the like.

While it is true that the set of gradient descent failures is potentially much larger than the set of tangles, no such occurrences were ever seen during the full course of this experiment. If this sort of failure ever does become more troublesome, undoubtedly standard methods for minimizing local minima difficulties could be employed with typical amounts of success. Certainly the creation of an algorithm that combines the ideas of Marill, Leclerc & Fischler, and this paper will help clarify the extent of the problem posed by general gradient descent failure and even perhaps pave the way for the application of other techniques (included but not limited to the family of simulated annealing algorithms).

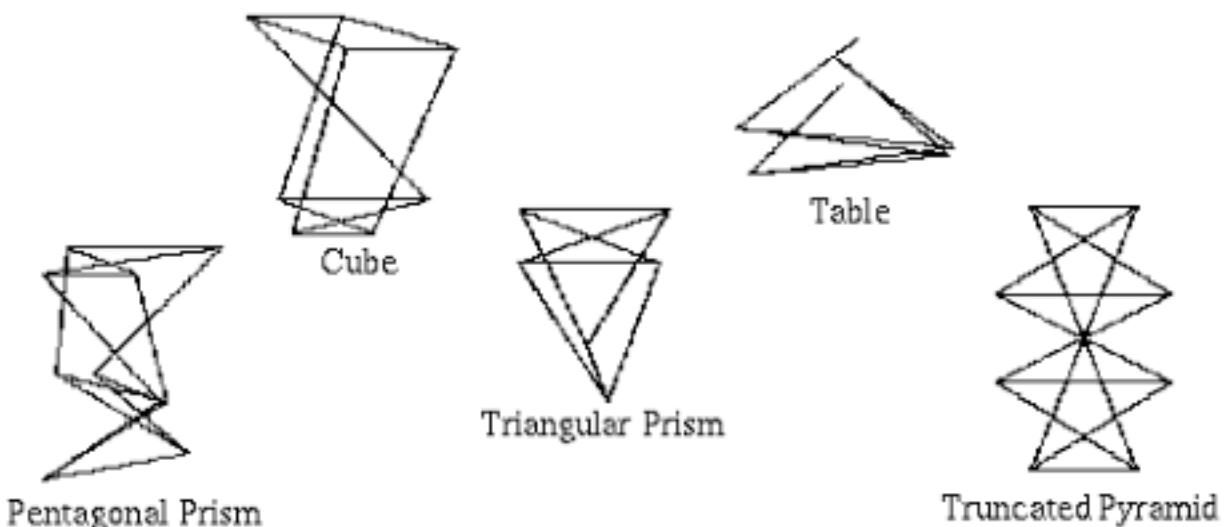


Figure 1: Examples of Tangles

The second cause of failure is more fundamental and thus more problematical. It is the case where using the correct M.S.D.A. still produces incorrect results. This type of error is due to a real limitation with the algorithm; it cannot be fully eliminated with Leclerc & Fischler's additions. There were not too many examples in Marill's original test data set that had this problem. Most simple wire-frame images do not, and it requires a little deliberate thought to create such examples. Shapes such as prisms and pyramids often have multiple dominant angles and thus were potential troublemakers for the straight Marill algorithm. Irregular shapes often have no real dominant angle and can wreak equal havoc. A few such shapes were added to the test set. Although these shapes did seem to cause difficulties for the algorithm, it was typically tough to see the errors caused by inappropriate M.S.D.A.'s through the tangles.

Figure 2, "Examples of Bad M.S.D.A.'s" shows a couple of cases where the fact that the calculated M.S.D.A. really did not reflect the real shape was a little more apparent than average.



Figure 2: Examples of Bad M.S.D.A.'s

A special case of inappropriate M.S.D.A.'s occurs in wire-frames that contain 180° angles. These straight angles seem to universally confuse the pure form of Marill's algorithm.

Figure 3, "Examples of Straight Angle Confusions," shows two of these cases. While at first blush it might seem to be a relatively simple fix to make the algorithm recognize this special case, it turns out not to be so. Simply removing all straight angles from the M.S.D.A. calculations does not work. In addition to introducing a new threshold in the definition of a straight angle (is a 177° angle straight or not?) it is important to remember that a given point is usually involved in other angles besides the straight one of interest. The consequences of the other modifications affecting that point can still nullify any advantages gained by eliminating the straight angle from the M.S.D.A. calculation. Straight angle elimination was incorporated into a version of Marill's algorithm anyway; no improvements were noted. While it is possible that careful tweaks to the algorithm could have some positive impact, it is not clear that this path would be fruitful enough to justify any effort; at best it could only improve the performance for one small special case of a much larger general problem.

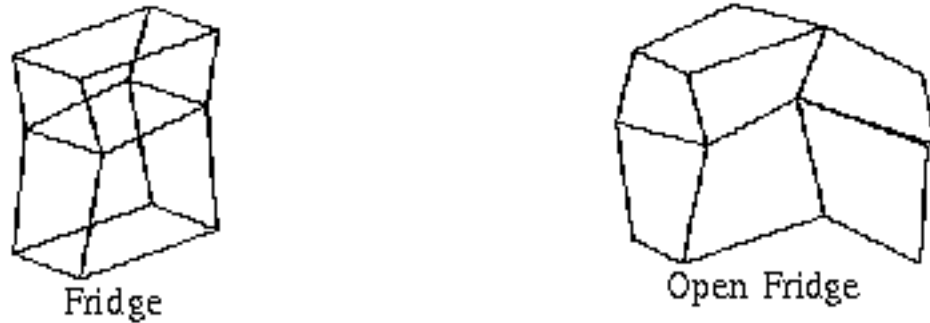


Figure 3: Examples of Straight Angle Confusions

M.S.D.S.M. Experimental Setup

Finally, after this converted program had been tested and verified, it was used as the basis for a program that worked with the standard deviation of segment lengths rather than the standard deviation of angles. Such a program was written. It took a straightforward approach very similar to Marill's original one¹: new objects were generated by adding or subtracting a small amount from each z-coordinate in the original figure. The standard deviation of segment lengths was calculated for each and the minimum picked. The cycle was then repeated endlessly until the image settled.

The gradient descent method used by Baird & Wang⁵ is clearly superior to this method, but for initial testing of the theory this method was far more direct. Additionally, since the calculation of segment lengths is simpler than the calculation of angles, it turned out that the overall speed of even the simplistic version of this algorithm was comparable to the speed of the M.S.D.A. algorithm with Baird & Wang's improvements. It is expected that further speed improvements could be made to this algorithm using an approach analogous to that used by Baird & Wang.

M.S.D.S.M. Algorithm Behavior

Expectations were that if the initial assumption of the human mind naturally trying to build patterns was the underlying reason the M.S.D.A. approach worked was in fact correct, then the minimum standard deviation of segment magnitudes approach should also work for certain cases with similar (albeit hopefully different) strengths and weaknesses. Initial experimentation showed that this was indeed the case. Just as with M.S.D.A., the biggest problem M.S.D.S.M. faced (in terms of number of failures) was wire-frames getting tangled. Not only was the percentage of tangled cases similar, the actual resulting tangled images themselves were indistinguishable from the tangled images produced by the M.S.D.A. algorithm. Once again, please refer to Figure 1, "Examples of Tangles," to see some examples of tangled images. As with M.S.D.A., it is expected that the addition of the conditions suggested by Leclerc & Fischler could virtually eliminate this particular type of failure from the algorithm.

Also, just as it was possible for the M.S.D.A. algorithm to fail in cases where the M.S.D.A. simply was not adequate to represent the intended image, it was expected that the M.S.D.S.M.

algorithm would fail in some cases where the M.S.D.S.M. was not adequate. The obvious place to look for such failures was in wire-frame objects built from segments of varying lengths, and upon searching such failures were indeed found (see Figure 4, "Examples of Bad M.S.D.S.M.'s"). There is really no natural analog to M.S.D.A.'s special case problem of straight angles in the M.S.D.S.M. domain; the closest is perhaps segments of zero length. This seemed to be such an odd case that it was ignored.

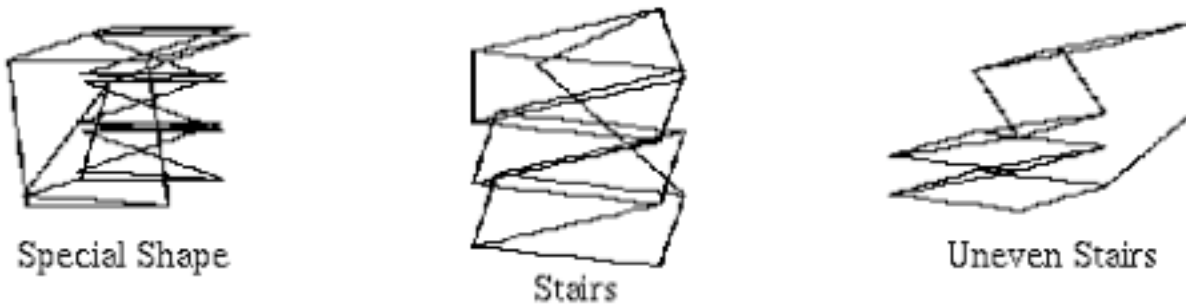


Figure 3: Examples of Bad M.S.D.S.M.'s

One somewhat unexpected difference between the two algorithms was their sensitivity to small changes in initial conditions. While slightly different inputs often resulted in vastly different outputs for the M.S.D.A. algorithm, equivalent differences input to the M.S.D.S.M. algorithm rarely resulted in noticeable outputs at all. It is hypothesized that this is somehow related to the nature of the calculations required for determining angles, but it could also be somehow related to the different types of gradient descent algorithms employed.

Result Comparison

The two algorithms were then run side by side for five trials on each object in the test data set (see "Appendix I: Shape Line Drawings"). A random seed of 1.0 was used in every case. Neither program was restarted after individual trials, and the settling time given for each trial was three minutes. While it is of course true that some randomness in running time will inevitably be present in every multi-tasking system, all attempts were made to minimize such effects (only one user was allowed on the system while the test program was running, no additional user programs were running, etc.). Table 1, "Test Results" below summarizes these results.

Shape Name	Results of M.S.D.A. Algorithm	Results of M.S.D.S.M. Algorithm
Special Shape	0%	0%
Cube	40%	0%
Stairs	80%	0%
Uneven Stairs	0%	0%
Table	20%	0%
Symphony Hall	0%	0%
Triangular Prism	60%	100%
Tetrahedron	100%	100%
Simple Fridge (HL)	40%	0%
Fridge	0%	0%
Symphony Hall (HL)	0%	0%
Open Fridge (HL)	0%	0%
Pyramid	0%	100%
Octahedron	0%	100%
Pentagonal Prism	0%	80%
Tetrahedron 2	100%	100%
Pyramid 2	0%	100%
Truncated Pyramid	0%	0%

Table 1: Test Results

Note that the abbreviation HL (both here and elsewhere) indicates that the wire-frame diagram contained hidden lines. The percentage numbers indicate the percentage of successful trials out of the five. Thus, a 10% indicates that in only one trial of five did the algorithm produce correct results. Note also that the rigor used in allowing a certain amount of settling time might not be fully necessary. On no occasion was it observed that an image started to obvolve and then correct itself. Every such occurrence ended in a tangled image.

Discussion

Several observations may be made from these results. The first is that both algorithms were able to correctly interpret the same number of images (seven) correctly some of the time. The other images they completely failed to interpret properly every time. The second is that while the M.S.D.S.M. seemed to score better on this data set, the difference was not a huge one particularly when the first observation is taken into account and it is remembered that some cases that were expected to be hard for M.S.D.A. were deliberately added to the data set. The third is

that there were some images that were easy for both algorithms (such as the tetrahedrons), some images that were hard for both algorithms (such as irregular shapes like the special shape and mixed shapes like the truncated pyramid), some images that were hard for M.S.D.A. but easy for M.S.D.S.M. (such as the pyramid), and some images that were easy for M.S.D.A. and hard for M.S.D.S.M. (such as the stairs). Other observations about algorithm behaviors (such as sensitivity to initial conditions) have already been mentioned and need not be repeated here.

Future Work

These results suggest many avenues for future work. Since the sets of cases that each algorithm is good at are not identical, it seems extremely reasonable to assume that improved overall results could be obtained by combining the two. If an algorithm were devised to implement both the M.S.D.A. and M.S.D.S.M. approaches along with the conditions suggested by Leclerc & Fischler, expectations are that results would be dramatically improved. Irregular shapes would probably still be problematical, but possibly the forced balance between the different algorithms would help in this regard, or at least provide more insight into new directions for even further research.

It is also clear that there may be other features that could be exploited in much the way that angles and segment magnitudes are to produce even further dramatic improvements.

Conclusion

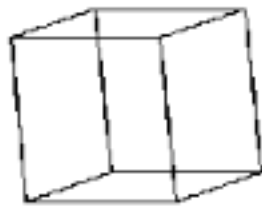
The results clearly indicate that the M.S.D.S.M. approach has some merit. They would also argue against the use of either pure M.S.D.S.M. or pure M.S.D.A. algorithms as the only means of object reconstruction in a serious project. While both provide some ability to recognize objects, neither provides enough alone to be generally reliable. The results also reinforce the initial theory of this paper -- that Marill's approach works because of the nature of the human mind.

On a final note, it may appear that this paper is treating Marill's M.S.D.A. algorithm somewhat harshly. It should be clearly stated that this is not the intent. While it is true that the M.S.D.A. algorithm by itself cannot completely deal with the problem of interpreting wire-frame diagrams as three-dimensional objects in a human-like manner, it is true that Marill's experiment has introduced a wonderful new technique to the world that is capable of much expansion and enhancement. The advantages of a non-model based approach to the problem should be so obvious as to not require comment.

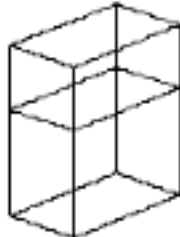
References

- 1 Thomas Marill, "Recognizing Three-Dimensional Objects Without the Use of Models", A. I. Memo #1157 of the MIT Artificial Intelligence Laboratory, 1-13, September 1989
- 2 Thomas Marill, "Emulating the Human Interpretation of Line-Drawings as Three-Dimensional Objects", *International Journal of Computer Vision*, Iss. 6:2, 147-161, 1991, Kluwer Academic Publishers, The Netherlands
- 3 Thomas Marill, "Why Do We See Three-Dimensional Objects?", A. I. Memo #1366 of the MIT Artificial Intelligence Laboratory, 1-22, June 1992
- 4 Yvan G. Leclerc & Martin A. Fischler, "An Optimization-Based Approach to the Interpretation of Single Line Drawings as 3-D Wire Frames", *International Journal of Computer Vision*, 1-14, 1992, The Netherlands
- 5 Leemon Baird & Patrick Wang, "3D Object Recognition Using Gradient Descent and the Universal 3-D Array Grammar", *SPIE*, Vol. 1607, 711-716, 1991
- 6 Leemon Baird & Patrick Wang, "3D Object Perception Using Gradient Descent", *Int. Journal of Mathematical Imaging and Vision (IJMIV)*, 5, 111-117, 1995

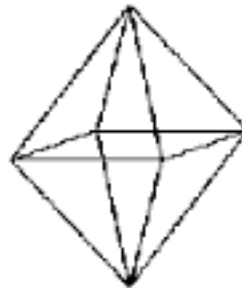
Appendix I: Shape Line Drawings



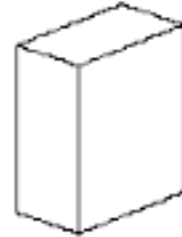
Cube



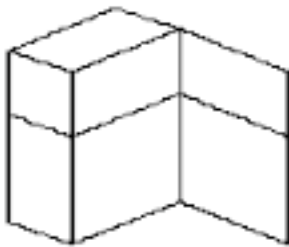
Fridge



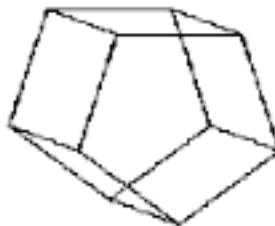
Octahedron



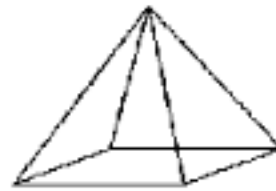
Simple Fridge (HL)



Open Fridge (HL)



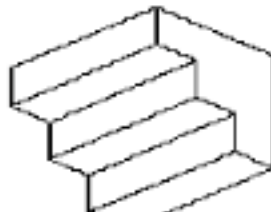
Pentagonal Prism



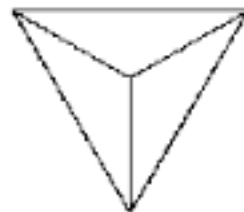
Pyramid



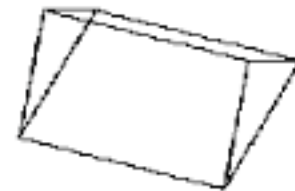
Pyramid 2



Stairs



Tetrahedron 2



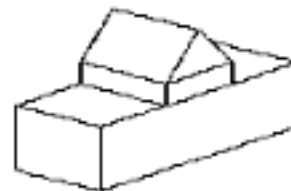
Triangular Prism



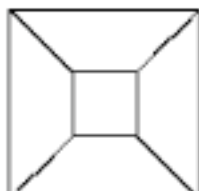
Symphony Hall



Special Shape



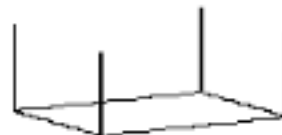
Symphony Hall (HL)



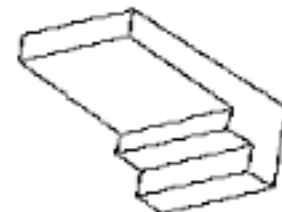
Truncated Pyramid



Tetrahedron



Table



Uneven Stairs

Appendix II: Shape Definitions

```
# The wire-frame data file for the xmar program
#
# Format Description:
#
# Recognized fields include: Name, Random Seed, Random Range
# Step Size, Step Decrement, Scale, Points List, and Lines List.
# The octothorpe '#' character is used at the beginning of a line to
# indicate a comment.
# All the fields (except for the two list types) should be followed by
# a colon ':' and two spaces before the argument is given. All of
# these fields expect a floating point argument with the exception
# of Name. Name expects a string.
# The two list types should sit on a line by themselves followed by
# a colon. The following lines contain the actual point or line data.
# Point data is contained in ordered triples of floating point
# numbers delineated by parentheses and separated by commas.
# Line data is contained in ordered pairs of floating point numbers
# delineated by braces and separated by commas.
#
# Each defined shape should have a Name and Point & Lines lists.
# Other fields are optional.
#
```

Name: Special Shape
Random Seed: 1.000

Points List:
(0.000, 0.000,0.0), (3.460, 0.000,0.0), (5.430, -0.700,0.0),
(16.290, -2.100,0.0), (19.750, -2.100,0.0), (21.720, -2.800,0.0),
(21.370, -6.740,0.0), (17.910, -6.740,0.0), (17.560,-10.680,0.0),
(14.100,-10.680,0.0), (13.750,-14.620,0.0), (6.830,-14.620,0.0),
(-1.050,-11.820,0.0), (19.400, -6.040,0.0), (15.940, -6.040,0.0),
(15.590, -9.980,0.0), (12.130, -9.980,0.0), (11.780,-13.920,0.0),
(8.320,-13.920,0.0), (2.410,-11.820,0.0), (8.670, -9.980,0.0)

Lines List:
{ 0, 1}, { 0,12}, { 1,19}, {12,19}, { 2,20},
{20, 3}, { 3, 2}, { 1, 2}, {20,18}, { 3, 4},
{17,10}, {16, 9}, {15, 8}, {14, 7}, {13, 6},
{ 4, 5}, {12,11}, {11,10}, {10, 9}, { 9, 8},
{ 8, 7}, { 7, 6}, { 6, 5}, { 4,13}, {13,14},
{14,15}, {16,17}, {17,18}, {18,19}, {15,16}

Name: Cube
Random Seed: 1.000

Points List:
(-2.890, -1.620,0.0), (0.570, -1.620,0.0), (0.920, 2.320,0.0),
(-2.540, 2.320,0.0), (-0.920, -2.320,0.0), (2.540, -2.320,0.0),
(2.890, 1.620,0.0), (-0.570, 1.620,0.0)

Lines List:
{ 0, 1}, { 1, 2}, { 2, 3}, { 0, 3}, { 4, 5},
{ 5, 6}, { 6, 7}, { 4, 7}, { 0, 4}, { 1, 5},
{ 2, 6}, { 3, 7}

Name: Stairs
Random Seed: 1.000

Points List:
(-2.400, -1.450,0.0), (0.140, 2.070,0.0), (-0.710, 1.710,0.0),
(-0.710, 0.900,0.0), (-1.560, 0.540,0.0), (-1.560, -0.280,0.0),
(-2.400, -0.630,0.0), (0.780, -2.790,0.0), (3.320, -1.720,0.0),
(3.320, 0.730,0.0), (2.470, 0.370,0.0), (2.470, -0.450,0.0),
(1.630, -0.800,0.0), (1.630, -1.620,0.0), (0.780, -1.980,0.0)

Lines List:

{ 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}

Name: Uneven Stairs
Random Seed: 1.000

Points List:
(-15.510, 12.740,0.0), (-13.760, 14.990,0.0), (-14.080, 14.780,0.0),
(-14.020, 14.450,0.0), (-14.340, 14.240,0.0), (-14.280, 13.990,0.0),
(-15.570, 13.070,0.0), (-14.110, 12.410,0.0), (-12.180, 13.660,0.0),
(-12.350, 14.670,0.0), (-12.680, 14.460,0.0), (-12.620, 14.130,0.0),
(-12.940, 13.920,0.0), (-12.880, 13.670,0.0), (-14.170, 12.750,0.0)

Lines List:
{ 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}

Name: Table
Random Seed: 1.000

Points List:
(-0.670, 1.500,0.0), (3.670, 1.070,0.0), (1.670, 0.470,0.0),
(-2.670, 0.900,0.0), (-0.670, -0.470,0.0), (3.670, -0.900,0.0),
(1.670, -1.500,0.0), (-2.670, -1.070,0.0)

Lines List:
{ 0, 1}, { 1, 2}, { 2, 3}, { 3, 0}, { 0, 4},
{ 1, 5}, { 2, 6}, { 3, 7}

Name: Syphony Hall
Random Seed: 1.000

Points List:
(0.000, 15.000,0.0), (8.000, 17.000,0.0), (26.000, 11.000,0.0),
(26.000, 5.000,0.0), (18.000, 3.000,0.0), (17.000, 2.000,0.0),
(9.000, 0.000,0.0), (6.000, 5.000,0.0), (6.000, 7.000,0.0),
(0.000, 9.000,0.0), (8.000, 11.000,0.0), (14.000, 9.000,0.0),
(14.000, 7.000,0.0), (20.000, 5.000,0.0), (20.000, 7.000,0.0),
(18.000, 9.000,0.0), (12.000, 3.000,0.0), (12.000, 5.000,0.0)

Lines List:
{ 0, 1}, { 0, 9}, { 9,10}, { 1,10}, { 9, 8},
{10,11}, { 1, 2}, { 2, 3}, { 3, 4}, { 3,14},
{ 8,11}, {11,14}, { 7, 8}, {11,12}, {13,14},
{ 7,12}, {12,13}, { 6, 7}, { 5,12}, { 5, 6},
{ 5,13}, { 0,15}, { 2,15}, { 4,15}, { 8,17},
{14,17}, {16,17}, { 4,17}, { 6,16}, { 7,16},
{13,16}

Name: Triangular Prism
Random Seed: 1.000

Points List:
(17.000, 2.000,0.0), (9.000, 0.000,0.0), (6.000, 5.000,0.0),
(14.000, 7.000,0.0), (15.000, 2.000,0.0), (7.000, 0.000,0.0)

Lines List:
{ 2, 3}, { 3, 4}, { 1, 2}, { 0, 3}, { 0, 1},
{ 0, 4}, { 1, 5}, { 2, 5}, { 4, 5}

Name: Tetrahedron
Random Seed: 1.000

Points List:
(17.000, 2.000,0.0), (14.000, 7.000,0.0), (20.000, 5.000,0.0),
(19.000, 3.000,0.0)

Lines List:

{ 0, 1}, { 0, 2}, { 0, 3}, { 1, 2}, { 1, 3},
{ 2, 3}

Name: Simple Fridge (HL)

Random Seed: 1.000

Points List:

(13.000, 11.000,0.0), (13.000, 6.000,0.0), (13.000, 6.000,0.0),
(13.000, 3.000,0.0), (8.000, 1.000,0.0), (5.000, 0.000,0.0),
(0.000, 2.000,0.0), (0.000, 10.000,0.0), (3.000, 11.000,0.0),
(8.000, 9.000,0.0), (8.000, 4.000,0.0), (3.000, 3.000,0.0)

Lines List:

{ 4, 5}, { 4, 9}, { 4,11}, { 5, 6}, { 6, 7},
{ 6,11}, { 7, 8}, { 8, 9}, { 8,11}

Name: Fridge

Random Seed: 1.000

Points List:

(13.000, 11.000,0.0), (13.000, 6.000,0.0), (13.000, 6.000,0.0),
(13.000, 3.000,0.0), (8.000, 1.000,0.0), (5.000, 0.000,0.0),
(0.000, 2.000,0.0), (0.000, 5.000,0.0), (0.000, 10.000,0.0),
(3.000, 11.000,0.0), (8.000, 9.000,0.0), (8.000, 4.000,0.0),
(3.000, 3.000,0.0), (3.000, 6.000,0.0), (5.000, 3.000,0.0),
(5.000, 8.000,0.0)

Lines List:

{ 4, 5}, { 4,11}, { 4,12}, { 5, 6}, { 6, 7},
{ 6,12}, { 7, 8}, { 7,13}, { 8, 9}, { 9,10},
{ 9,13}, {10,11}, {11,13}, {12,13}, { 5,14},
{ 7,14}, { 8,15}, {10,15}, {14,15}, {11,14}

Name: Symphony Hall (HL)

Random Seed: 1.000

Points List:

(0.000, 15.000,0.0), (8.000, 17.000,0.0), (26.000, 11.000,0.0),
(26.000, 5.000,0.0), (18.000, 3.000,0.0), (17.000, 2.000,0.0),
(9.000, 0.000,0.0), (6.000, 5.000,0.0), (6.000, 7.000,0.0),
(0.000, 9.000,0.0), (8.000, 11.000,0.0), (14.000, 9.000,0.0),
(14.000, 7.000,0.0), (20.000, 5.000,0.0), (20.000, 7.000,0.0)

Lines List:

{ 0, 1}, { 0, 9}, { 9,10}, { 1,10}, { 9, 8},
{10,11}, { 1, 2}, { 2, 3}, { 3, 4}, { 3,14},
{ 8,11}, {11,14}, { 7, 8}, {11,12}, {13,14},
{ 7,12}, {12,13}, { 6, 7}, { 5,12}, { 5, 6},
{ 5,13}

Name: Open Fridge (HL)

Random Seed: 1.000

Points List:

(13.000, 11.000,0.0), (13.000, 6.000,0.0), (13.000, 6.000,0.0),
(13.000, 3.000,0.0), (8.000, 1.000,0.0), (5.000, 0.000,0.0),
(0.000, 2.000,0.0), (0.000, 5.000,0.0), (0.000, 10.000,0.0),
(3.000, 11.000,0.0), (8.000, 9.000,0.0), (8.000, 4.000,0.0),
(3.000, 3.000,0.0), (3.000, 6.000,0.0)

Lines List:

{ 0, 1}, { 0,10}, { 1,11}, { 2, 3}, { 2,11},
{ 3, 4}, { 4, 5}, { 4,11}, { 4,12}, { 5, 6},
{ 6, 7}, { 6,12}, { 7, 8}, { 7,13}, { 8, 9},
{ 9,10}, { 9,13}, {10,11}, {11,13}, {12,13}

Name: Pyramid

Random Seed: 1.000

Points List:

(0.920, 2.320,0.0), (-2.540, 2.320,0.0), (-0.570, 1.620,0.0),
(2.890, 1.620,0.0), (0.180, -1.270,0.0)

Lines List:

{ 0, 1}, { 1, 2}, { 2, 3}, { 0, 3}, { 0, 4},
{ 1, 4}, { 2, 4}, { 3, 4}

Name: Octahedron

Random Seed: 1.000

Points List:

(0.920, 2.310,0.0), (-2.540, 2.310,0.0), (-0.570, 1.610,0.0),
(2.890, 1.610,0.0), (0.180, -1.280,0.0), (0.180, 5.200,0.0)

Lines List:

{ 0, 1}, { 1, 2}, { 2, 3}, { 0, 3}, { 0, 4},
{ 1, 4}, { 2, 4}, { 3, 4}, { 0, 5}, { 1, 5},
{ 2, 5}, { 3, 5}

Name: Pentagonal Prism

Random Seed: 1.000

Points List:

(0.000, 0.000,0.0), (3.460, 0.000,0.0), (4.530, 3.290,0.0),
(1.730, 5.320,0.0), (-1.070, 3.290,0.0), (1.970, 0.700,0.0),
(5.430, 0.700,0.0), (6.500, 3.990,0.0), (3.700, 6.020,0.0),
(0.900, 3.990,0.0)

Lines List:

{ 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}

Name: Tetrahedron 2

Random Seed: 1.000

Points List:

(0.000, 0.000,0.0), (1.000, 0.000,0.0), (0.500, 0.866,0.0),
(0.500, 0.288,0.0)

Lines List:

{ 0, 1}, { 1, 2}, { 2, 3}, { 0, 2}, { 0, 3},
{ 1, 3}

Name: Pyramid 2

Random Seed: 1.000

Points List:

(0.000, 0.000,0.0), (1.000, 0.000,0.0), (1.000, 1.000,0.0),
(0.000, 1.000,0.0), (0.500, 0.500,0.0)

Lines List:

{ 0, 1}, { 1, 2}, { 2, 3}, { 0, 3}, { 0, 4},
{ 1, 4}, { 2, 4}, { 3, 4}

Name: Truncated Pyramid

Random Seed: 1.000

Points List:

(0.000, 0.000,0.0), (3.000, 0.000,0.0), (3.000, 3.000,0.0),
(0.000, 3.000,0.0), (1.000, 1.000,0.0), (2.000, 1.000,0.0),
(2.000, 2.000,0.0), (1.000, 2.000,0.0)

Lines List:

{ 0, 1}, { 1, 2}, { 2, 3}, { 0, 3}, { 4, 5},
{ 5, 6}, { 6, 7}, { 4, 7}, { 0, 4}, { 1, 5},
{ 2, 6}, { 3, 7}

Copyright © 1994 Eric W. Brown, Northeastern University