CREATIVE GEOMETRIC MODELING
WITH
UNIGRAFIX

Edited by Carlo H. Séquin

With contributions by
Z. Gigus, E. Hunter,
M. Liebman, J. Mock,
G. Sanborn, M. Segal,
P. Ts'o, and P. Wensley.

Computer Science Division
Electrical Engineering and Computer Sciences
University of California, Berkeley, CA 94720

ABSTRACT

This is a report on a new graduate course in Computer Graphics and Solids Modeling, first offered in Fall 1983. The goal of the course was to improve the spatial perception of the participants by modeling three- and four-dimensional objects.

As part of the course, some utility programs for the UNIGRAFIX system where developed that aid in the construction of polyhedral geometrical objects. These programs were then used to create artistic displays. Examples of both activities are also presented in this report.

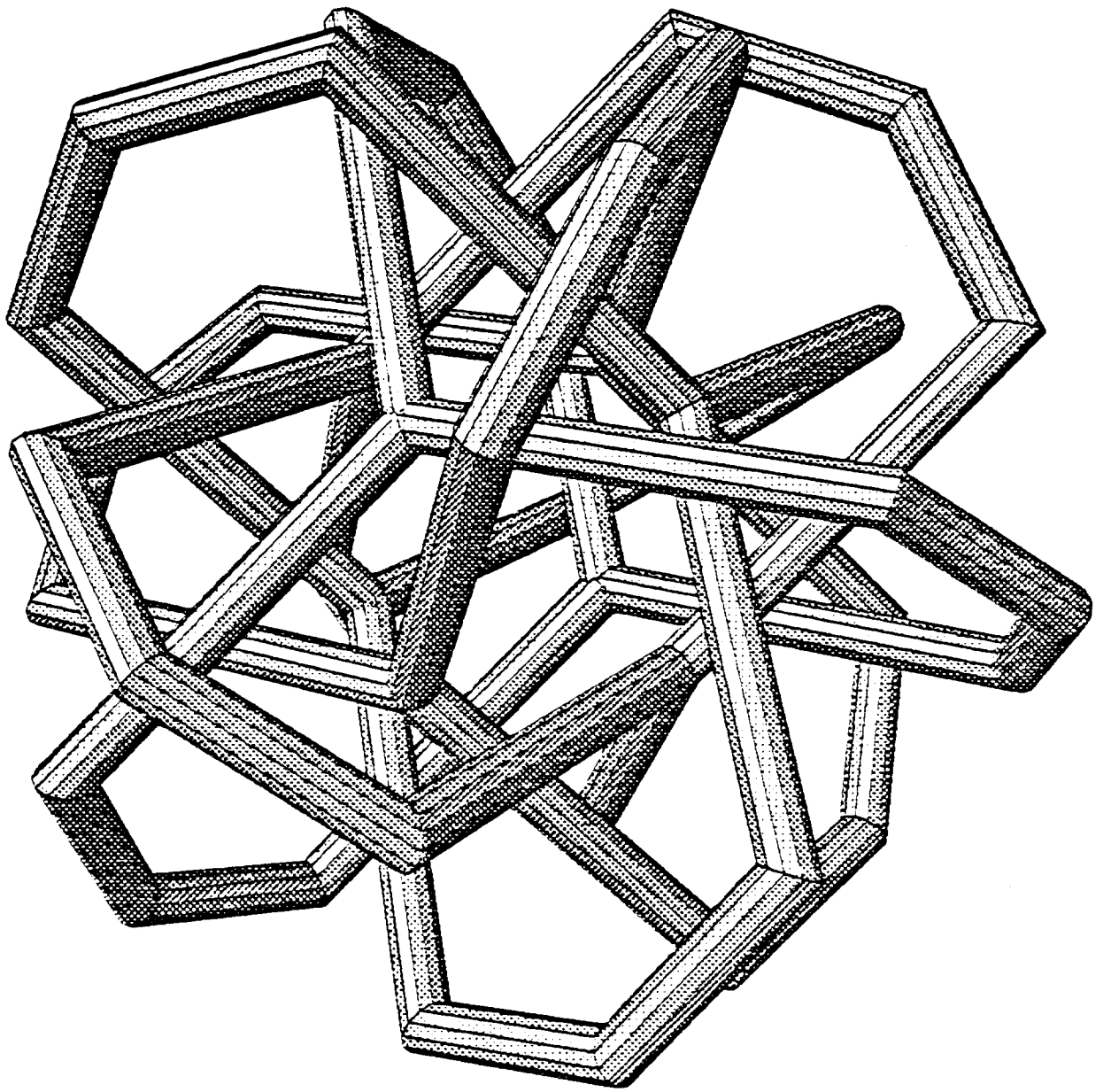
The development of the UNIGRAFIX system is supported by the Semiconductor Research Cooperative under grant number SRC-82-11-008.

NEW IN FALL 1983:

Creative Geometric Modeling

CS 292 ; 1.5 lect.hrs./week; 2 units; S/U or letter grade.

Algorithms and techniques for the creation of interesting and artistic objects in two, three, or four dimensions. Creation of a library of generator routines. Rendering of such objects in various styles on different output devices. Introduction to, use of, and extension of UNIGRAFIX.



1. THE COURSE CS 292A

Studies at Rensselear Polytechnic Institute have shown evidence that there is a strong correlation between the performance of engineering students and the strength of their spatial perception. The skills to visualize complex data as a functional entity permits them to find more easily a good way to optimize the solution to a multi-faceted problem. As a consequence, R.P.I. has introduced computer graphics as a standard tool in many of the courses in the main-stream of the engineering curriculum. Simulation results, for instance, are presented as a group of curves on an interactive computer terminal rather than simply as a heap of numbers in a print-out.

Computer graphics in Berkeley has been revived in 1981 with the development of the Berkeley Computer Graphics Laboratory under the direction of Prof. Brian Barsky. In 1983 a new graduate course, "Creative Geometric Modeling" was added to the catalog of our offerings in the area of computer graphics. The main goals of this course were:

1) To give the recently developed UNIGRAFIX system a hard work-out to identify bugs in the algorithms and in the user interface.
2) To enhance the UNIGRAFIX environment by adding facilities that make it easier to create complex geometrical objects.
3) To develop the spatial perception of the participants and to train their skills in geometry.
4) To satisfy the increasing demand from students to learn more about the fast-moving and ever more pervasive field of computer graphics.

The course had eight formally enrolled students and three regular auditors. In order to satisfy the course requirements, each participants had to complete the following assignments:

1) Invent or extract from the literature an algorithm useful for the creation of interesting geometrical objects and present this approach in class.
2) Implement the algorithm in C or in Pascal and document it with manual pages and tutorial examples.
3) Use a combination of the evolving utility programs to create an artistic display.

In addition, the course encompassed two take-home tests that contained a variety of think-puzzles exercising spatial perception in 3 and 4 dimensions.

Syllabus

The course extended over 15 weeks with 1½ lecture hours per week. This is a rough outline of the lecture and discussion topics on a week by week basis:

1) Importance of spatial perception and geometric skills.
Course goals, class format, deliverables, background questionnaire.
Introduction to the UNIGRAFIX system, the language,
rendering with *ugshow*.
Assignment: Get familiar with UNIGRAFIX; do your initials in 3D.

2) Concept of generator programs: example *mkworm*.
Axfile specifications; worm parameters: radius, n-sides, azimuth, twist.
Ax-analysis files: axspec, axdist; use of "wordy" mode.
Mitred prismatic segments; relation of ax-turn, azimuth, twist.
Implementation of *mkworm*, incremental transformation of local coordinates.
Need for other generators for linear sweep and rotational sweep.
Derivation of specifications for a universal sweep program: *ugsweep*.

3) Discuss experience with *mkworm*.
Problem of determining end-to-end axturn from local information.
Joining multiple cylindrical pipes, algorithm for intersection.
Constraints that make mitring of prismatic joints possible.
Knots: square knot - Granny knot, clover-leaf knot, torus knots.
Constraints for tight knots, use of symmetry to reduce degrees of freedom.

4) More on problem with ax-turn: 360 degree ambiguity.
Presentation by Paul Wensley:
"Data structures and algorithms in UNIGRAFIX2".
Discussion: What makes a good project presentation.

5) New features in *mkworm*: options -L, -N, new distance algorithm.
Presentation by Pauline Ts'o:
"Growth algorithm for trees, corals, shells".
Mitring of cylinders of unequal diameter and of prismatic cones.
Discussion: What are good course projects.

6) Introduction to the Platonic solids. Construction and visualization of Platonic solids in n dimensions.
Simplex, N-cube (measure polytope) and its dual (cross polytope).
Problems involving regular solids: e.g., how does a cube float.

7) Construction of icosahedron and dodecahedron, duality.
Derivation of Archimedean solids.
Presentation by Mark Segal:
"Intersecting faces".
Take-home test number 1.

8) Discussion of exam problems.
Hints for the use of AED and SUN terminals, the SUN file system.

9) Icosahedral symmetry generator and its application.
Presentation by Jeff Mock:
"Geodesic domes"
Exchange of first drafts of manual pages for software projects.
Presentation by Mark Liebman
"Projection from 4D space to 3D space".

10) More on projecting from 4D to 3D space.
The more complicated 4-dimensional regular polytopes.

11) Presentation by Ed Hunter:
"*Ugstar* - putting pyramids on faces".
Presentation by Ziv Gigus:
"*Ugsweep* - universal sweep generator".

12) Presentation by Greg Sanborn:
"*Mkstairs* and *Mkpath*".
Presentation by Eric Bier:
"Rendering half-spaces and infinite planes"
Take-home test number 2.

13) Discussion of exam problems.
Status check on programming projects.

14) Test-run for talk in front of CAD-Committee:
"TOOL-BUILDING: Computer Graphics and Solids Modeling at UCB".
Guide-lines for final manual pages, tutorial examples, and artwork.

15) Moebius band with a circular boundary.
Transformations between Platonic and Archimedean solids.
Truncation and dual operation for irregular solids.
The translation-rotation operation ("snub" operation).
What is "ART" ?? Hints for artistic projects.

2. UNIGRAFIX PROGRAMS

This section gives a brief introduction to the programs developed or completed during this course offering. To put these programs in context, a brief overview over the UNIGRAFIX system is first presented.

2.1. Object Description

UNIGRAFIX uses a terse ASCII format to represent polyhedral objects. A UNIGRAFIX file consists of statements, starting with a keyword and ending with a semicolon. Statements consist of lexical tokens, separated by commas, blanks, tabs, or newlines. The language is simple and has only a few different types of statements:

vertices:	**v**	*ID x y z* ;
wires:	**w**	[*ID*] (*v1 v2 ... vn*) (...) [*colorID*] ;
faces:	**f**	[*ID*] (*v1 v2 ... vn*) (...) [*colorID*] ;
definitions:	**def**	*defID* ; *non-def-commands* **end;**
instances:	**i**	[*ID*] (*defID* [*transformations*]) ;
arrays:	**a**	[*ID*] (*defID* [*transforms*]) *size* [*transforms*] ;
lights:	**l**	[*ID*] *intensity* [*x y z*] ;
color:	**c**	*colorID intensity* [*hue* [*saturation*]] ;
include files:	**include**	*filename* [*transformations*] ;
comments:	{	[*anything* {*nesting is OK*} *but unmatched* { *or* }] }

These descriptions are currently created with a text editor or with the generator or modifier programs discussed in a later section. An interactive editor is still in the planning stage.

2.2. Rendering

Once an object description has been created, the object can be rendered on a variety of output devices. Currently there are drivers for the following terminals: Dumb terminals, HP2648A, AED 512, Vectrix, and Ikonas frame buffer. Hardcopy can be produced on an 11-inch-wide Varian printer or on a 36-inch Versatec printer, both with a resolution of 200 dots per inch.

Programs that take a UNIGRAFIX description and produce a display or hardcopy output are:

ugshow (Strauss),

ugplot (Wensley, Segal).

These are described in detail in the "UNIGRAFIX 2 User's Manual and Tutorial," CS-Technical Report, UCB/CSD 83/161, December 1983.

The collection of all objects in the scene can be globally transformed in the world coordinate system with the program

ugxform (Wensley).

This program reads a scene from standard input and returns the transformed scene to standard output. It permits scaling along each one of the three coordinate axes, as well as mirroring of, rotation around, or translation along each axes. In addition, the program accepts an arbitrary 4×4 matrix and performs the corresponding transformation on the scene.

Some of the modifier programs can only handle hierarchically flat UNIGRAFIX descriptions. Such a flat description of the scene can be generated with the filter program

ugexpand (Wensley).

This program also permits the specification of a transformation as the previously discussed *ugxform*.

Other programs that modify a UNIGRAFIX description to prepare it for rendering are:

ugisect (Segal)

removes all intersections which cannot be handled by the rendering algorithms in *ugshow* and *ugplot*.

These programs are also described in detail in the "UNIGRAFIX 2 User's Manual and Tutorial."

2.3. Generators

There are several programs that create an object description in UNIGRAFIX format from scratch:

mkstairs (Sanborn)

creates spiral staircases or ramps according to a set of parameters.

mkpath (Sanborn)

creates a random orthogonal walk through 3D space and implements this path with streets and stairs.

mktree (Ts'o)

grows trees, shells, or corals in conjunction with *mkworm*.

mkworm (Séquin)

creates properly mitred prismatic tube sections around piece-wise linear paths through 3D space.

2.4. Modifiers

These programs change a UNIGRAFIX description to produce a new object.

Some programs only modify the individual faces:

ughole (Mock)

cuts a hole into each face of the given polyhedron.

ug4hole (Mock, Liebman)

same as *ughole* but for 4-dimensional objects.

ugshrink (Séquin)

separates the faces of a polyhedron and shrinks them; can also be used to cut holes or to produce concentric rings.

ugsweep (Gigus)

sweeps a polygon through space with an arbitrary transform and produces the surface of the swept out volume.

ugfreq (Mock)

subdivides triangular faces into smaller similar facets.

Other programs change the actual shape of the object:

ugtrunc (Séquin)

truncates the corners of a polyhedron.

ugstar (Hunter)

constructs inside or outside pyramids on all faces of a polyhedron.

ugdual (Séquin)

creates the dual of a regular solid and tries to do something reasonable for irregular solids.

ugsphere (Mock)

projects all vertices radially from the origin onto a sphere.

Yet other programs preserve the underlying shape of the original object, but render it by drawing sticks or worms along the original edges, or project it from higher space into 3D space:

ugwire (Séquin)

creates a wire segment for every physical edge in the original polyhedron.

ugstick (Mock)

replaces each edge with an (irregular) prismatic stick.

ug4to3 (Liebman)

projects 4-dimensional vertex coordinates into 3D space.

2.5. Program Documentation

The following section presents the manual pages for the generator and modifier programs. In some cases, these are followed by a few tutorial examples to show the variety of effects that can be produced with these programs.

NAME

mkpath – make UNIGRAFIX description of a random three-dimensional path.

SYNOPSIS

mkpath [**–n** int] [**–s** x y z] [**–R**] > pathfile

DESCRIPTION

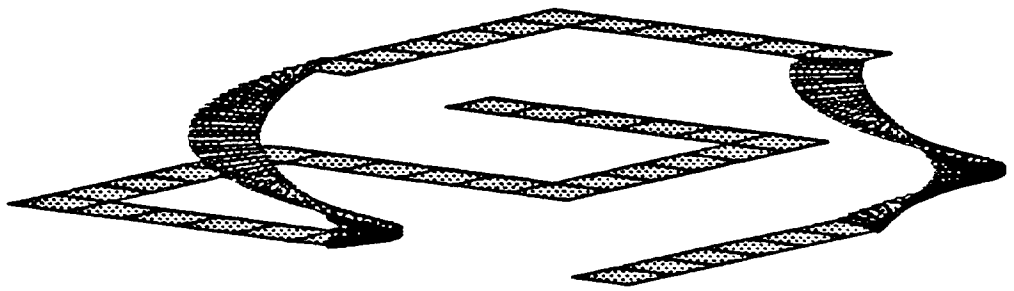
Mkpath outputs a UNIGRAFIX description of a random three-dimensional orthogonal path. It uses a three-dimensional array to construct the path and avoid self-intersections. It uses a tree-growing algorithm starting at a random place near the center of the space array. This path is then followed with level streets and with stairs or ramps created with the *mkstairs* algorithm, which has been incorporated into *mkpath*.

Mkpath creates a description containing six main items. There are five definitions: a simple segment for level straight paths, one for a single step, a definition for a staircase one level high, a landing to join levels of staircases and two landings that join a straight path to a staircase. The rest of the file consists of instances and arrays that form the path. The options are:

–n *number*	build a path with *number* segments,
–s *x y z*	set space limits of path for the *x*, *y*, and *z* directions.
–R	build stairways as a helical ramp, made with triangles, instead of the default stairs.

EXAMPLE

```
mkpath -s 3 2 3 >pathfile;
cat light pathfile | ugplot -sa -ed -9 2 -7 -dv -sy 7
```



FILES

~ug/bin/mkpath
~ug/src/MKPATH/*

SEE ALSO

mkstairs(UG), ugshow(UG), ugplot(UG)

BUGS

Stairs and streets float freely in space. Options for the number of steps per level, supporting pillars and other embellishments may be added later.

AUTHOR

Greg Sanborn

NAME

mkstairs – make UNIGRAFIX description of helical stairways

SYNOPSIS

mkstairs [parameters] > stairsfile

DESCRIPTION

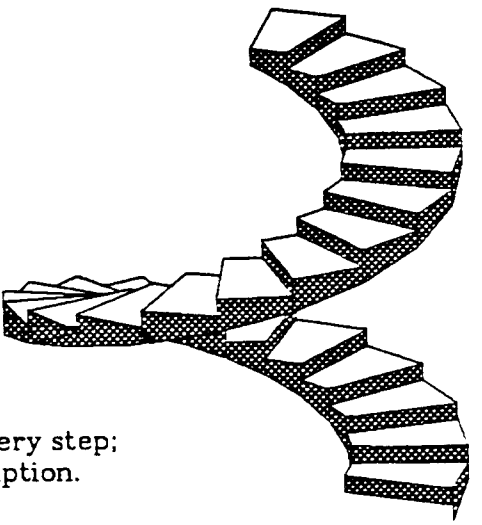
Mkstairs outputs a UNIGRAFIX description of a helical staircase or ramp. Each step is an instance of a definition which describes a single step or ramp segment with the proper geometry for a smooth fit. Parameters for the stairs are:

–n *number*	make *number* steps. The default is 24.
–R	build a smooth ramp with triangles instead of the default stairs.
–P	attach hexagonal pads to the top and bottom of the staircase.
–i *radius*	set inner *radius* of the staircase. The default is 2.
–o *radius*	set outer *radius* of the staircase. The default is 4.
–a *angle*	set the *angle* of each step. The default is 15 degrees.
–A *Angle*	set the total *Angle* of the staircase, the sum of the angles of all the steps. The default is a full circle, 360 degrees.
–r *run*	set the *run* of each step, the distance from the front of the stair to the riser of the next step measured at the center of the step.
–h *height*	set the *height* of the risers. The default is 0.5.
–H *Height*	set the total absolute *Height* of the staircase, from the bottom level to the upper level.

The program may adjust the **–a**, **–r** and the **–h** parameters so that an integer number of steps will result. If both angle options are set as well as the number option, then the total angle value is used to calculate a new step angle value. The same is true for the height options. If the number option is not set, then the number of steps is calculated from the angle or height values.

EXAMPLE

mkstairs -a 18



FILES

~ug/bin/mkstairs
~ug/src/MKSTAIRS/*

SEE ALSO

mkpath(UG), ugshow(UG), ugplot(UG)

BUGS

Right now, instances are produced for every step; arrays would give a more compact description.

AUTHOR

Greg Sanborn

NAME

mktree – generate joint-coordinates for tree-like objects

SYNOPSIS

mktree [**–a**? s l u] [**–t**? s l u] [**–h**? s l u]

where "?" maybe either **f** (fixed), **o** (oscillating), or **r** (random).

DESCRIPTION

The program *mktree* outputs, in *mkworm*-format, the joint-coordinates of a tree-like object based on the growth algorithm of Yoishiro Kawagushi. The joint-coordinates are written into a file called "**ax**" in the user's current working directory. The shape of the tree can be altered by a set of command-line options:

–a? Default: ? = o. < s=20, l=5, u=45 >.
Specify bending angle for each joint of the tree. The values are in degrees and must not lie outside the interval 5 .. 45 degrees.

–t? Default: ? = r. < s=999, l=30, u=90 >.
Specify twist angle (in degrees) for each joint of the tree.

–h? Default: ? = f, <size = 1.0>.
Specify size (length) of trunc (branch) segment in units of its diameter.

–?f <value>
Specifies a fixed <value> for one of the above three parameters.

–?o <step> <lower> <upper>
Specifies an oscillating value for one of the above three parameters. The parameter changes by <step> from one joint to the next, sweeping back and forth between the <lower> and <upper> bounds.

–?r <seed> <lower> <upper>
Specifies a random value for one of the above three parameters. Starting from the given <seed>, a pseudorandom generator choses the values for the joints in the interval between <lower> and <upper> bounds.

–rb <beginning radius>
Specifies the radius of the beginning trunc segment.

–re <ending radius>
Specifies the radius below which the creation of branches is suppressed.

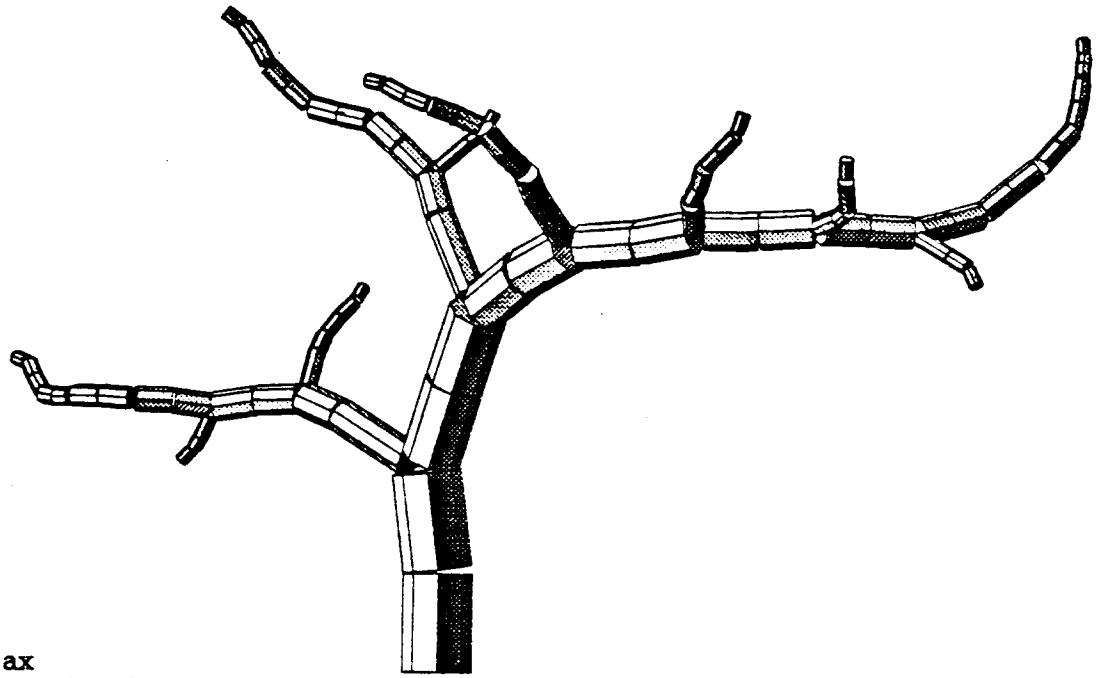
–g <max generation>
Specifies the maximum depth of the tree in the number of generations.

–N Suppresses the generation of all side-chains; results in a single "spiral".

Default values of all options are as follows: -oa 20 5 45 -tr 999 30 90 -hf 1.0

EXAMPLE

```
mktree -oa 20 5 45 -tr 999 30 90 -hf 1.0
mkworm -n6
cat ~ug/lib/illum worm | ugplot -sa -ed -2 1 -5 -dv -sy 6
```



FILES

ax
~ug/bin/mktree
~ug/src/mktree

SEE ALSO

mkworm (UG), ugshow (UG)

Yoishiro Kawagushi, *SIGGRAPH 1982 Conference Proceedings*, " A Morphological Study of the Form of Nature "

BUGS

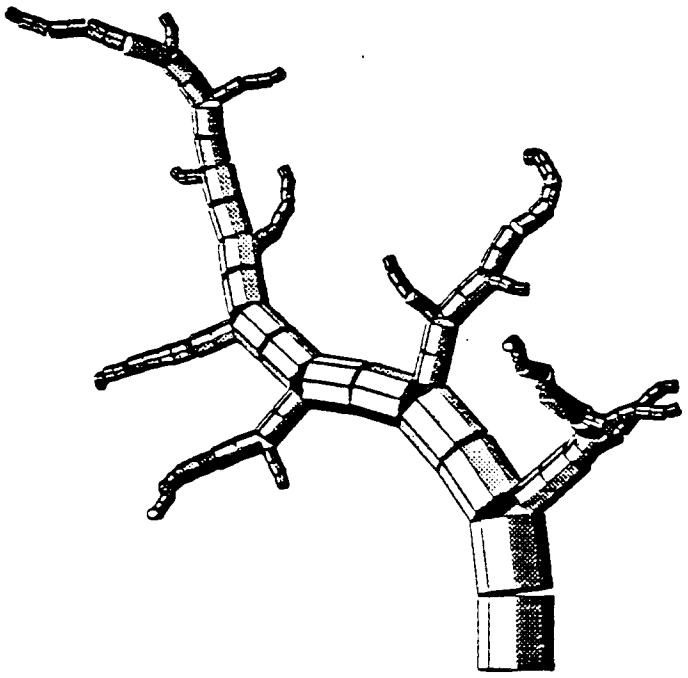
Yet to be reported.

AUTHOR

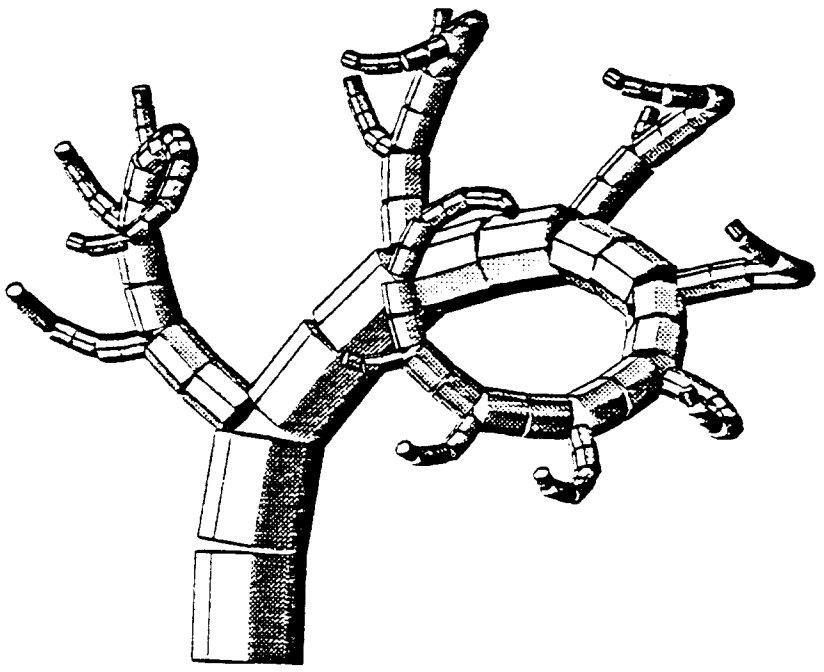
Pauline Ts'o

Tutorial examples for mktree

mktree; mkworm -n8; cat light worm | ugshow -sa -dv
(default tree, twist is random; angle is oscillating; height is 1.)

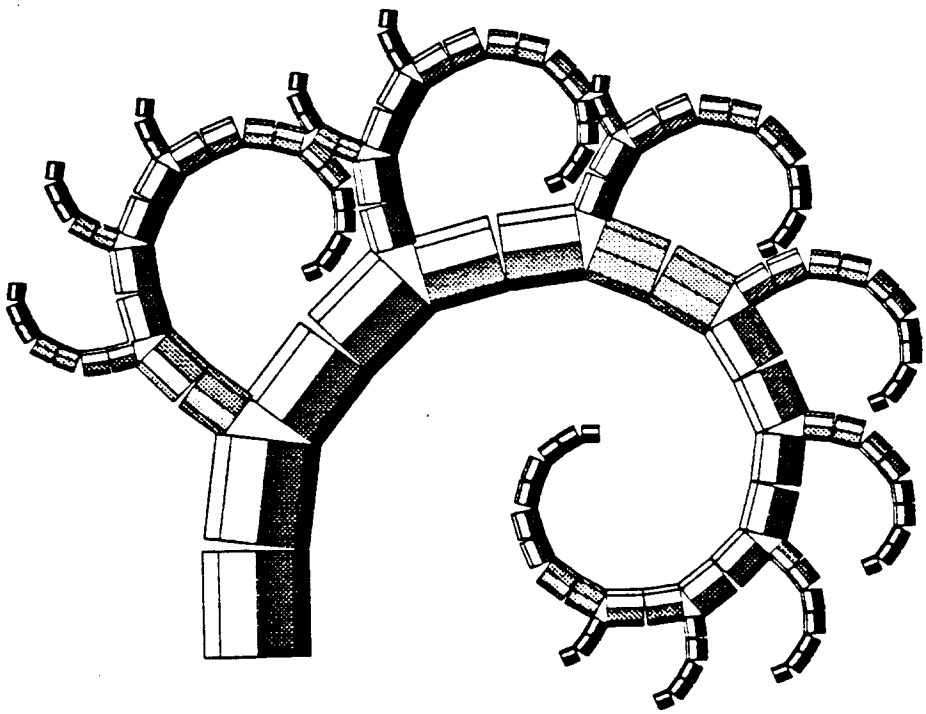


mktree -tf10; mkworm -n8; cat light worm | ugshow -sa -dv
(twist has been changed to a constant ten degrees; all other parameters remain as above)

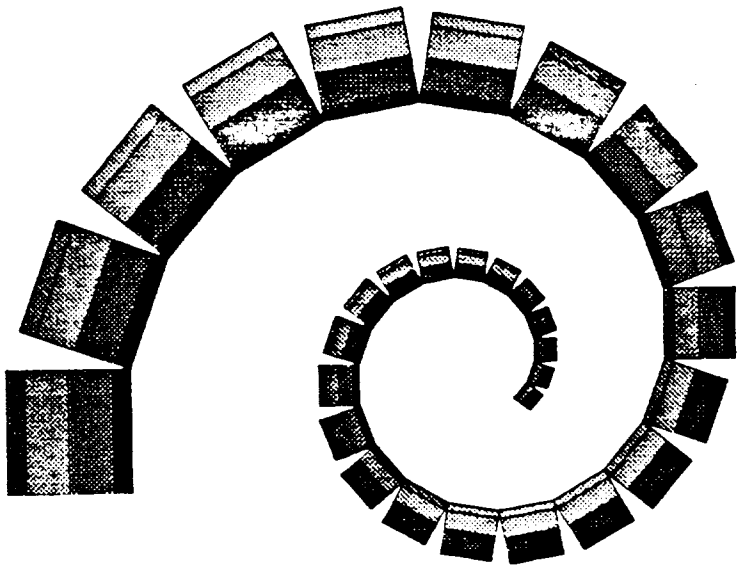


Tutorial examples for mktree (cont.)

mktree -tf 0; mkworm -n8; cat light worm | ugshow -sa -dv
(twist has been removed, all other parameters remain as before)



mktree -tf 0 -af 20 -N; mkworm -n8; cat light worm | ugshow -sa -dv
(angle fixed at 20 degrees, side-branches suppressed)



NAME

mkworm – generate piecewise prismatic tubes in UNIGRAFIX format

SYNOPSIS

mkworm [options]

DESCRIPTION

Mkworm is a generator program for a UNIGRAFIX description of worm-like bodies consisting of prismatic sections. *Mkworm* reads the file "**ax**" in the current directory and produces as output the file "**worm**" in UNIGRAFIX format.

The file "**ax**" contains the description of the axis of the worm embedded in 3-dimensional space. The statements specifying each joint are of the form:

j x-coord y-coord z-coord radius ;

One such statement is needed for every joint. The number of joints is currently limited to 513. These statements can be grouped together into individual open ended sections or into loops with the command pairs:

B(egin) – E(nd) : for open section with orthogonally terminated ends;

L(oop) – R(eturn) : for creating a closed loop.

Mkworm takes a number of parameters either on the command line or in an interactive question and answer session. If any one parameter is specified on the command line, the interactive session is suppressed. The parameters are:

-n <integer> Default: n= 4.
This specifies the number of sides that each prism section should have. There is a built-in limit of 52. If n is specified to be less than 2, the output in worm will be the UNIGRAFIX description of a wire along the ax of the worm.

-r <real> Default: r = 0.
This specifies a global radius for the whole worm. If the value is omitted or explicitly set to 0, the data that appears with each joint specification in the ax-file is used. This makes it possible to generate worms of varying thickness.

-t <real> Default: t = 0.
This defines the amount of twist in degrees that each prismatic section should have. It allows one to smoothly close non planar loops that have a net twist from beginning to end. If the total twist is evenly distributed among all the sections of the loop, the edges of the beginning and the end sections can be made to merge.

-a <real> Default: a = 0.
This defines the starting azimuth in degrees, so that the position of the edges of the prisms can be properly positioned in space.

-T This option specifies that the surface of the worm should be tessellated with triangles rather than with trapezoids. Planar ax-loops should have an even number of joints so that the edges of the end sections will properly meet.

-L This option will produce labels in the generated face statements.

-d This also suppresses the questions and uses default values.

-W This "Wordy" mode reports various parameters about the ax-segment orientation as it constructs the worm. This and the following options help in the analysis of complicated axes.

-S This prints Specifications for joints and segments.

-D This prints Distances between ax segments; it is useful when constructing tight knots.

-N This option suppresses the worm output; it is useful when one is interested in the ax-parameters only.

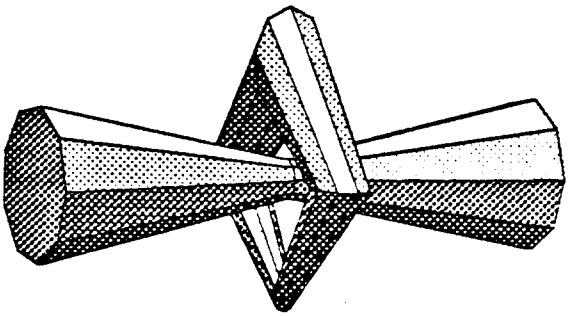
As the main product, mkworm will create an elliptical rib around each joint with the minor half-axis equal to the radius specified for that joint. By connecting subsequent ribs with prismatic sections, properly mitred corners are generated.

EXAMPLE

```
BEGIN
j 7 0 0 2;
j 0 0 0 0.5;
j -7 0 0 2;
END
LOOP
j 0 3 0 1;
j 0 0 3 0.7;
j 0 -3 0 0.3;
j 0 0 -3 0.7;
RET

mkworm -n9;

cat ~ug/lib/illum worm | ugplot -ep -50 10 -100 -sa -dv -sy 3
```



FILES

ax, worm,
~ug/bin/mkworm
~ug/src/PASCAL/mkworm.p

SEE ALSO

ugworm (UG), sweep (UG), ugplot (UG)

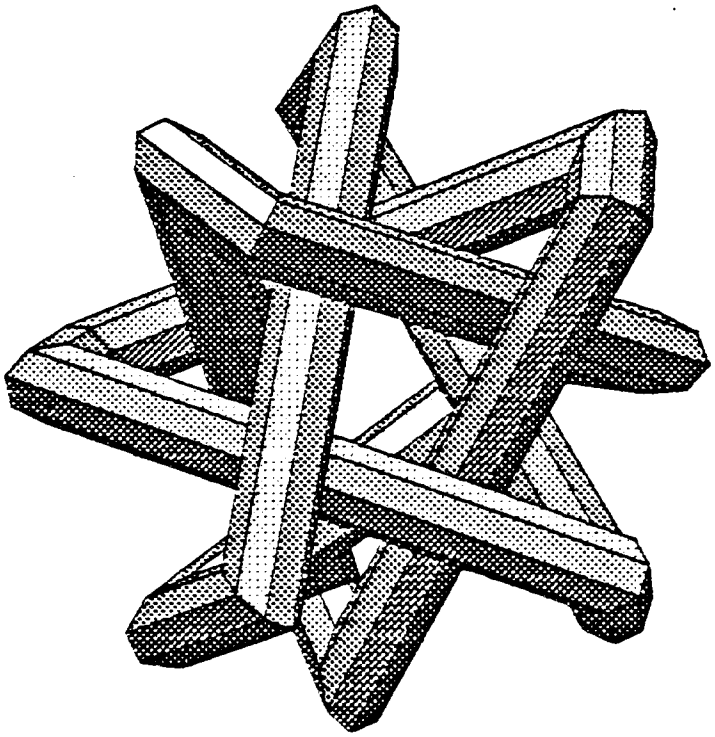
BUGS

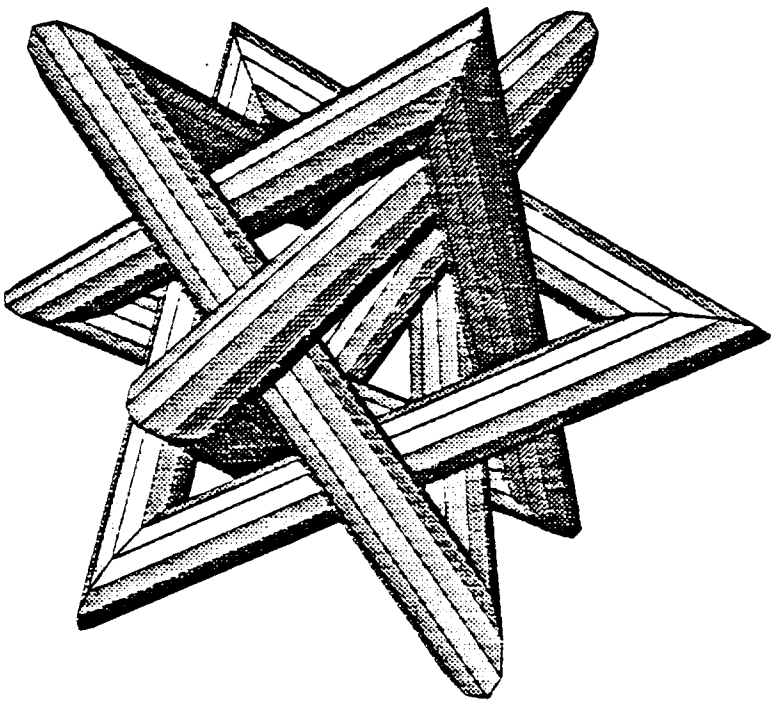
Cannot handle branches. May yield non-planar faces when there is twist or radius changes. Very acute angles will create rather large protrusions. If the specified axis doubles up on itself, the prismatic edges can really get mixed up.

AUTHOR

C.H. Séquin

Examples generated with mkworm





NAME

ug4to3 – project a 4-dimensional object to 3 space.

SYNOPSIS

ug4to3 [**-ep** x y z w] [**-ed** x y z w] [**-D**] < 4D-object > 3D-object

DESCRIPTION

Ug4to3 applies to each vertex the transformation specified by the **-ep** or **-ed** options. The default transformation is a parallel projection along the w-axis, i.e., simply a removal of the w-component. In any case, the input vertex statments must have four components, and the output vertices have three. All other lines are passed unaltered to the output. Available options:

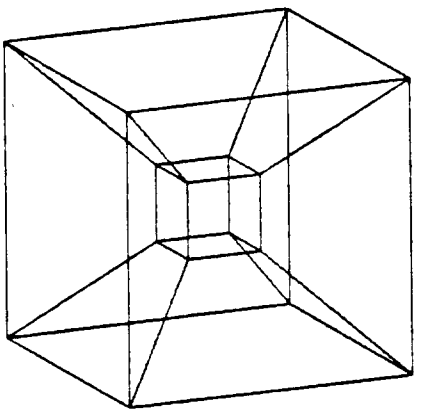
-ep <x y z w>
Sets the eyepoint at the specified location in 4D-space.

-ed <x y z w>
Sets the eyepoint at infinity in the specified direction in 4D-space.

-D Turns on debug mode and prints the transformation matrix to standard output.

EXAMPLE

cat ~ug/lib/Hcube | ug4to3 -ep 1.7 0 0 0 | ugshow -dv -se -ed -3 2 -7 -sy 2.5



FILES

~ug/bin/ug4to3
~ug/src/PASCAL/ug4to3.p

SEE ALSO

ughole (UG), ugxform (UG), ugexpand (UG), ugplot (UG)

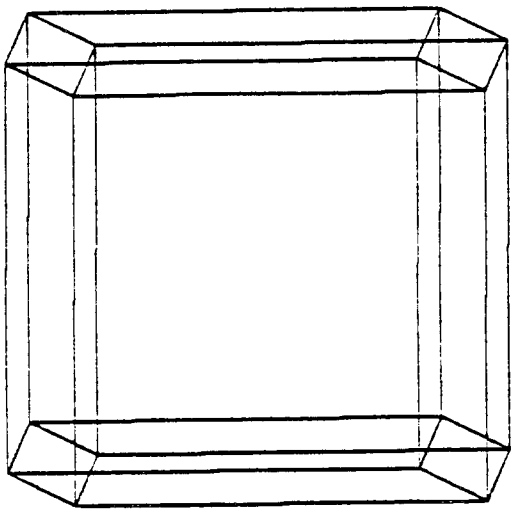
BUGS

Yet to be reported.

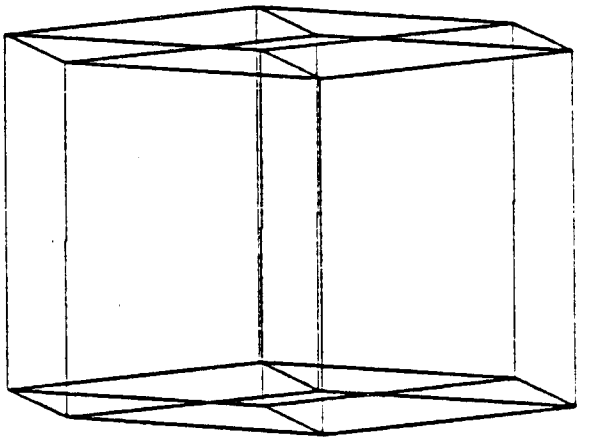
AUTHOR

Mark Liebman

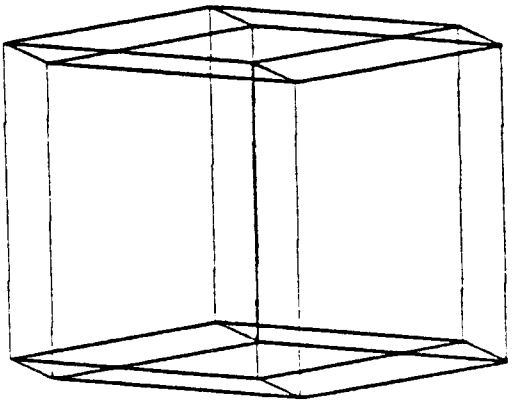
Tutorial Examples for Ug4to3



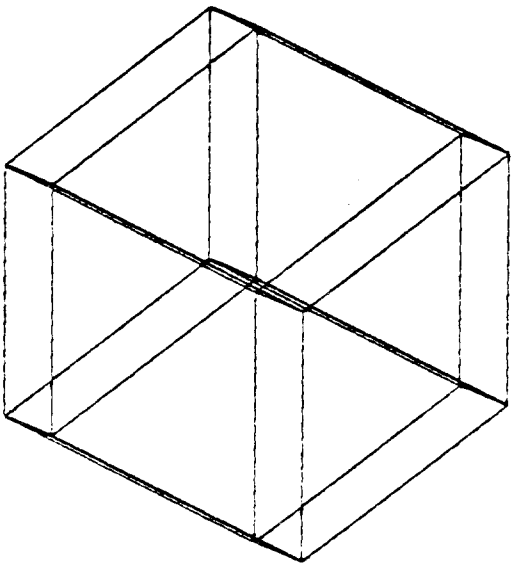
{near cell-cell coincidence}
cat ~lib/D4cube |
ug4to3 -ed 5.3 0.4 0.6 0.7 |
ugplot -ed -2 1 -12



{near face-face coincidence}
cat ~lib/D4cube |
ug4to3 -ed 5.3 5.4 0.6 0.7 |
ugplot -ed -2 1 -12



{near edge-edge coincidence}
cat ~lib/D4cube |
ug4to3 -ed 5.3 5.4 0.6 5.7 |
ugplot -ed -2 1 -12



{near vertex-vertex coincidence}
cat ~lib/D4cube |
ug4to3 -ed 5.3 5.4 5.6 5.7 |
ugplot -ed -2 1 -12

NAME

ugdual – create dual of a polyhedron described in UNIGRAFIX format

SYNOPSIS

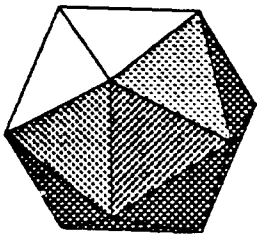
ugdual [options ??] < oldobject > newobject

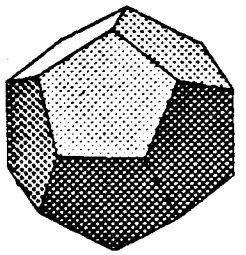
DESCRIPTION

Ugdual forms, on the standard output, the UNIGRAFIX description of the polyhedron dual to the one read in. It uses a very simple parser for the input which understands only vertex and face commands and ignores all others. In face commands it only reads the first contour group.

EXAMPLE

cat ~ug/lib/illum ~ug/lib/icosa | ugdual | ugplot -ed -2 1 -5 -sa -dv -sy 3





FILES

~ug/bin/ugdual
~ug/src/PASCAL/dual.p ~ug/src/PASCAL/readug.i

SEE ALSO

ugtrunc (UG), ugstar (UG), ugplot (UG)

BUGS

If the original object is not a well-behaved, fairly regular solid, you may get really weird results.

AUTHOR

C.H. Séquin

NAME

ugfreq – divide triangular faces into a number of triangles at a given frequency

SYNOPSIS

ugfreq [–f*freq*] < oldobject > newobject

DESCRIPTION

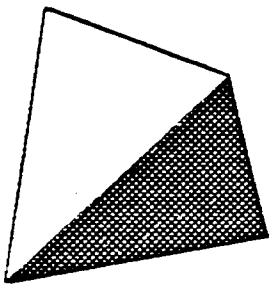
Ugfreq reads a flat UNIGRAFIX description from standard input, divides each triangular face into a number of triangles, and sends the resulting description to standard output. The default is a tessalation into 2 rows, i.e. four triangles.

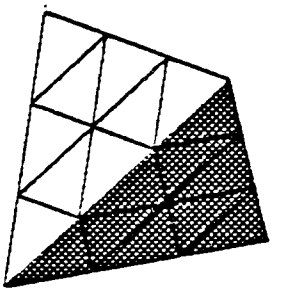
–f<*number*> Default: f = 2.

Specifies the ***number*** of rows of triangles to be created in each original triangle.

EXAMPLE

cat ~ug/lib/illum ~ug/lib/tetra | ugfreq -f3 | ugplot -ep 4 1 5 -sa -dv -sy 3





FILES

~ug/bin/ugfreq
~ug/src/PASCAL/freq.p ~ug/src/PASCAL/parse.h

SEE ALSO

ughole (UG), ugtrunc (UG), ugplot (UG)

BUGS

Works only for hierarchically flat descriptions.

AUTHOR

Jeff Mock

NAME

ughole – cuts holes in polygonal faces

SYNOPSIS

ughole [*–rratio*] [**–N**] [**–S**] < oldobject > newobject

DESCRIPTION

Ughole reads a flat UNIGRAFIX description from standard input, cuts holes into each face and turns each face into a thin, two-sided frame, and then outputs the result to standard output. The shape of the holes is similar to the shape of the original face but scaled down around the "center" of each face. The center is the mean of all corner vertices of the face.

–r*<rim-ratio>* Default: r = 0.2.

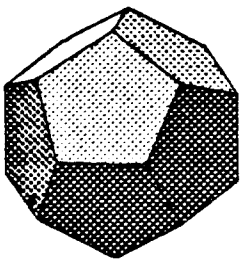
Defines the size of the resulting hole by specifying the width of the remaining *rim* as a *ratio* of the distance from the center of the face to the vertices.

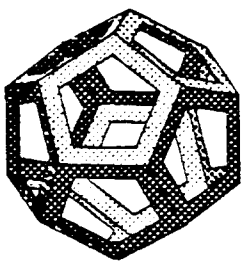
–S Causes each original edge of the face to be converted into a separate, convex polygon; the polygons together then make up the new face with a hole in the middle. The default is to generate a single face with an extra contour for the hole.

–N No backfaces. Eliminates the default creation of the faces with opposite orientation.

EXAMPLE

cat ~ug/lib/illum ~ug/lib/icosa | ughole | ugplot -ep 4 1 5 -sa -dv -sy 3





FILES

~ug/bin/ughole
~ug/src/PASCAL/hole.p ~ug/src/PASCAL/parse.h

SEE ALSO

ugshrink (UG), ugstick (UG), ugplot (UG)

BUGS

Faces with concave corners can cause problems, as the hole may intersect with the outer contour.

AUTHOR

Jeff Mock

NAME

ugisect – convert intersecting faces and wires into non-intersecting objects

SYNOPSIS

ugisect [options ??] < inputfile > outputfile

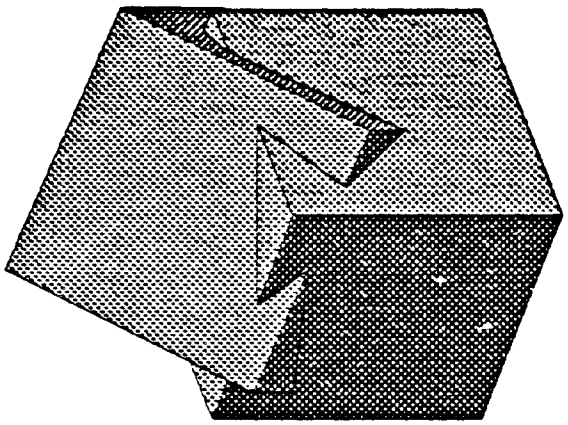
DESCRIPTION

Ugisect reads a UNIGRAFIX file and cuts up any intersecting faces and wires to produce a scene description with no intersecting elements. Each existing intersecting element is partitioned into several pieces. The default is to keep all these pieces together in a single statement with multiple contour groups.

Instances of definitions that are intersecting are expanded to the next lower hierarchical level, where all components are again checked for intersection.

EXAMPLE

cat ~ug/lib/illum two_cubes | ugisect | ugplot -ed -2 1 -5 -sa -dv -sy 3



FILES

~ug/bin/ugisect
~ug/src/UGISECT/*

SEE ALSO

ugexpand (UG), ugxform (UG), ugshow (UG), ugplot (UG)

DIAGNOSTICS

Upon termination *ugisect* will print out some statistics concerning the number of intersecting elements.

BUGS

So far, works only for flat UNIGRAFIX files.

AUTHOR

Mark Segal

NAME

ugshrink – shrinks all faces of a polyhedron

SYNOPSIS

ugshrink [**–f** 0.x] [**–H**] [**–B**] < oldobject > newobject

DESCRIPTION

Ugshrink reads a flat UNIGRAFIX description and forms, on the standard output, the UNIGRAFIX description of a polyhedron in which all faces have been separated and individually shrunk by the factor f.

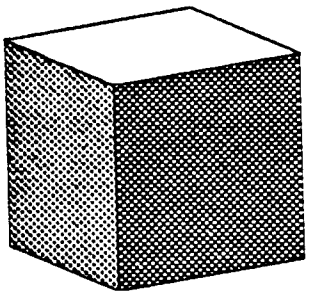
–f <real>
This option specifies the amount by which each face is scaled down in the range 0 to 1. Default is 0.999 which provides an invisible shrinkage, but is sufficient to circumvent certain problems with coinciding vertices that arise in UNIGRAFIX 2.

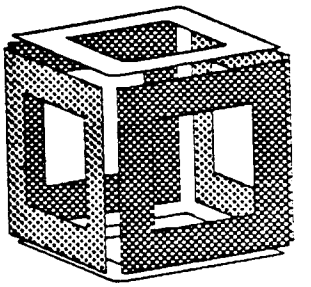
–H This option produces a frame with a hole rather than a shrunk face. It is similar to ughole. It can be applied multiple time to the same object and will then lead to concentric rings if the scale factors are suitably chosen.

–B This option also generates the backface to each face.

EXAMPLE

cat ~ug/lib/illum ~ug/lib/cube | ugshrink -f 0.9 | ugshrink -H -B -f 0.6 | ugplot -ed -2 1 -5 -sa -dv -sy 3





FILES

~ug/bin/ugshrink
~ug/src/PASCAL/shrink.p ~ug/src/PASCAL/readug.i

SEE ALSO

ughole (UG), ugtrunc (UG), ugplot (UG)

BUGS

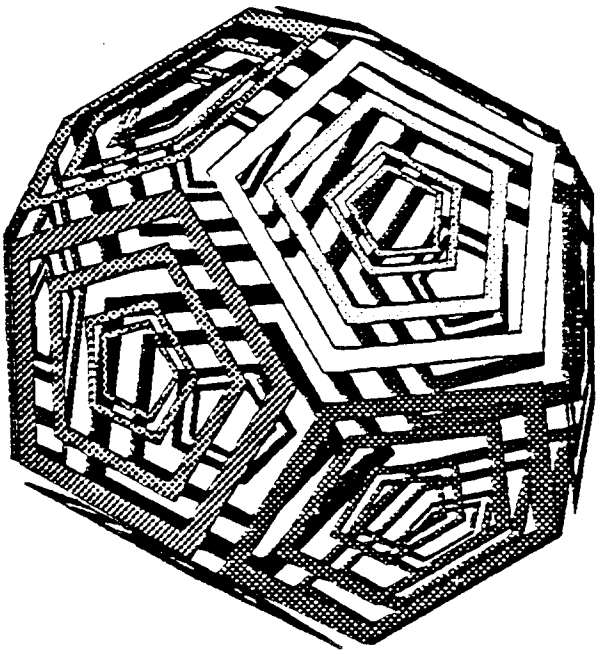
Yet to be reported.

AUTHOR

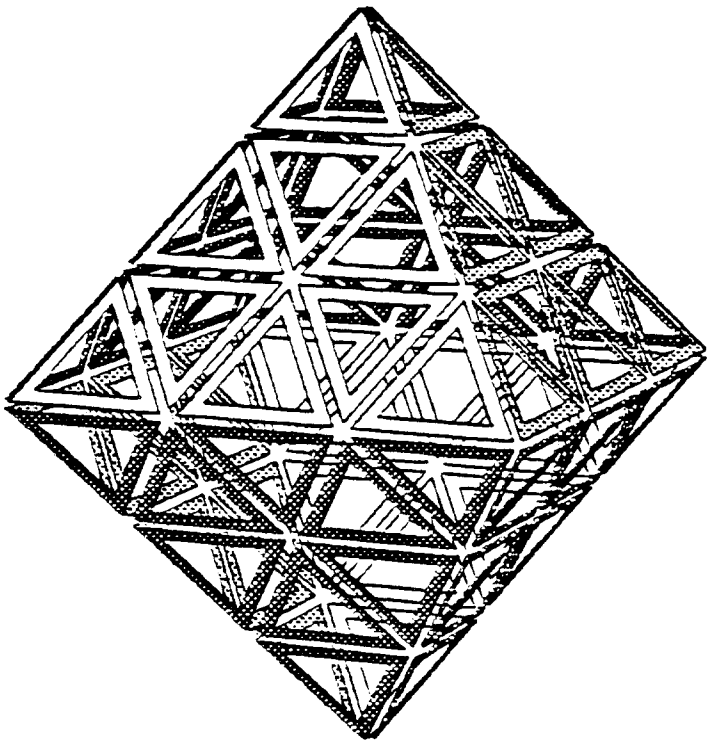
C.H. Séquin

Tutorial Examples for ugshrink

cat illum ../lib/dodeca | ugshrink -f0.9 | ugshrink -f0.4 -H | ugshrink -f0.7 -H | ugshrink -f0.85 -H -B | ugshow -ep -4 2 -10 -sa -dv -sy 4 -mi -4 2 -10 10 12



cat illum ../lib/octa | ugfreq -f3 | ugshrink -f 0.9 | ugshrink -f 0.7 -H -B | ugshow -ep -4 2 -10 -sa -dv -sy 4



NAME

ugsphere – project a polyhedron onto a sphere

SYNOPSIS

ugsphere [*–rradius*] [*–xxcenter*] [*–yycenter*] [*–zzcenter*] < oldobject > newobject

DESCRIPTION

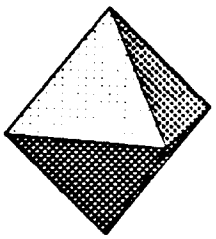
Ugsphere reads a flat UNIGRAFIX description from standard input, projects the vertices onto a sphere of a given radius around a specified center point, and sends the altered image to standard output. The parameters are specified as follows:

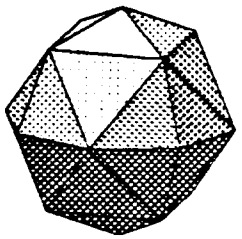
–r<radius> Default: r = 1.0.
Defines the radius of the sphere.

–x<xcenter>
–y<ycenter> Default: (x, y, z) = (0, 0, 0).
–z<zcenter>
Defines the center coordinates of the sphere.

EXAMPLE

cat ~ug/lib/illum ~ug/lib/octa | ugfreq -f2 | ugsphere | ugplot -ep 4 1 5 -sa -dv -sy 3





FILES

~ug/bin/ugsphere
~ug/src/PASCAL/sphere.p ~ug/src/PASCAL/parse.h

SEE ALSO

ugdual (UG), ugtrunc (UG), ugplot (UG)

BUGS

The generated faces might not be planar.

AUTHOR

Jeff Mock

NAME

ugstar – construct pyramids on all faces of a polyhedron

SYNOPSIS

ugstar **–h** height [**–N**] [**–C**] [**–D**] [**–f** filename] < oldobject > newobject

DESCRIPTION

Ugstar takes the UNIGRAFIX description of polyhedron and creates a pyramid on each face of the object. The tip of the pyramid lies on the face-normal through the face-center, where the face-center is defined by the mean of all vertices of the face, and the face-normal is computed from the vectors formed by the first three vertices of the face. The height of the pryamid is determined by the **–h** argument. If the height given is negative, or if the first three vertices of the face form a concave corner, then the direction of the pryamid is *into* the body.

–N This causes the height of the pyramids to be normalized by the average length of the line segments on the perimeter of the face. If all sides of the face are of unit length, the height of the pyramid will be equal to the value specified in the **–h** argument.

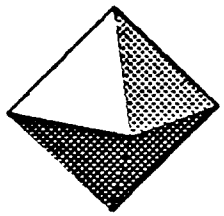
–C This causes *ugstar* to use the centers of the contour line segments of each face as the base points for the pyramid.

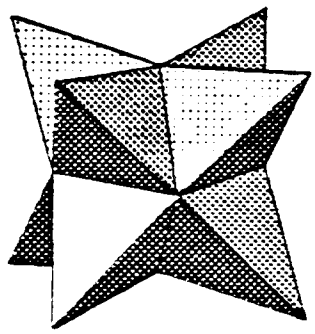
–D Debugging mode which will cause verbose output to **stderr**.

–f <filename>
This option will redirect the output into the specified file.

EXAMPLE

cat ~ug/lib/illum ~ug/lib/octa | ugstar -h 2.5 | ugplot -ed -2 1 -5 -sa -dv -sy 3





FILES

~ug/bin/ugstar
~ug/src/UGSTAR/*

SEE ALSO

ugdual (UG), ugtrunc (UG), mkworm (UG), ugplot (UG)

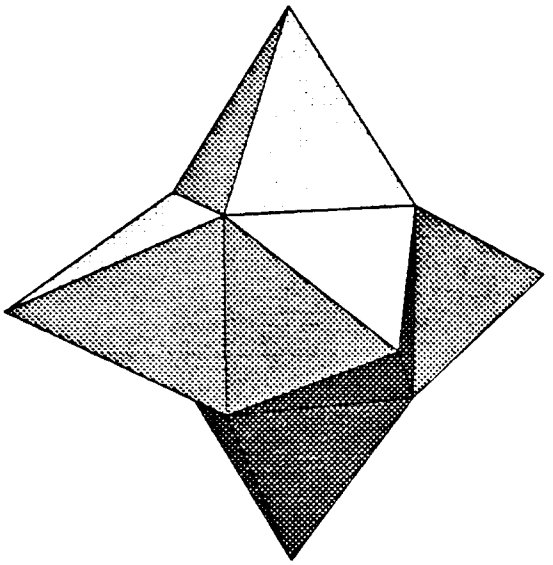
BUGS

For irregular faces with concave corners, the definition of what constitutes the "center" of the face is somewhat arbitrary.

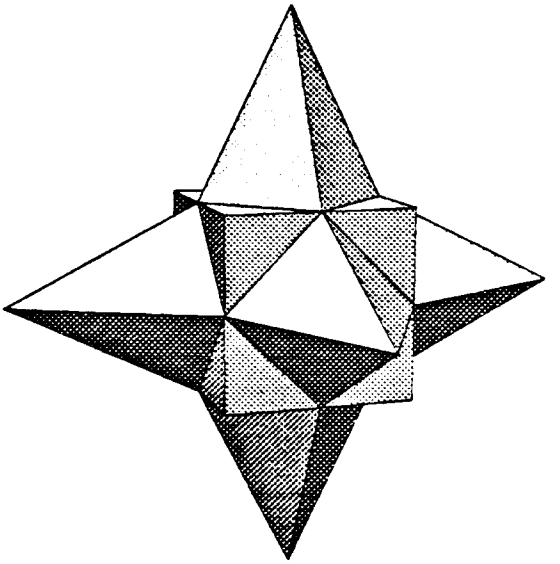
AUTHOR

Edward Hunter

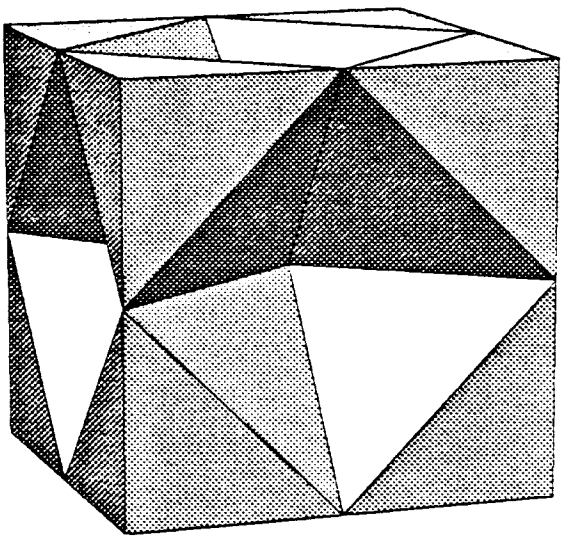
Tutorial Examples for Ugstar



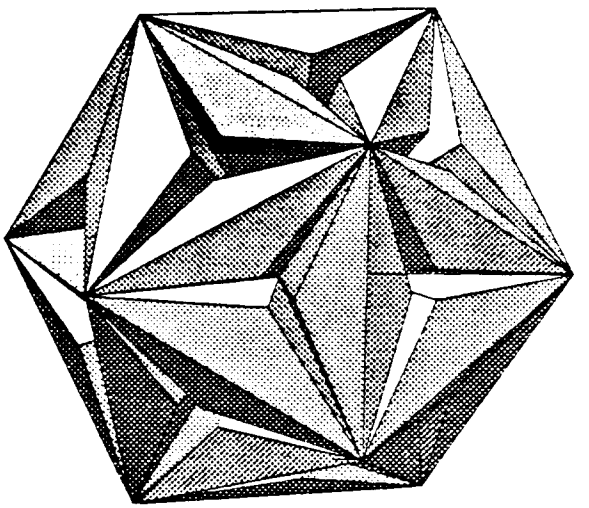
Example 1.
cat cube illum |
ugstar -h 2 |
ugshow -sa -ep -5 3 -15



Example 2.
cat cube illum |
ugstar -h 2 -C
ugshow -sa -ep -5 3 -15



Example 3.
cat cube illum |
ugstar -h -.75 -C |
ugshow -sa -ep -5 3 -15



Example 4.
cat icosa illum |
ugstar -h -.5 -C | ugstar -h .25 -N |
ugshow -sa -ep -5 3 -15

NAME

ugstick – produces a stick model of the input polyhedron

SYNOPSIS

ugstick [*–rratio*] [*–tthickness*] < oldobject > newobject

DESCRIPTION

Ugstick reads a flat UNIGRAFIX description from standard input, cuts holes into each face and turns each created segment into a three-dimensional member by projecting the remaining rim of the face towards the origin.

–r*<rim-ratio>* Default: r = 0.2.

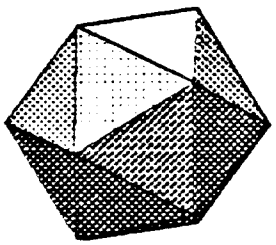
Defines the size of the resulting hole by specifying the width of the remaining *rim* as a *ratio* of the distance from the center of the face to the vertices.

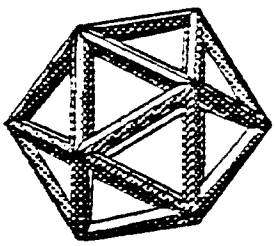
–t*<thickness>* Default: t = 0.2.

Specifies the *thickness* of the members as a fraction of the distance of the vertices from the origin.

EXAMPLE

cat ~ug/lib/illum ~ug/lib/icosa | ugstick | ugplot -ep 4 1 5 -sa -dv -sy 3





FILES

~ug/bin/ugstick
~ug/src/PASCAL/stick.p ~ug/src/PASCAL/parse.h

SEE ALSO

ughole (UG), ugtrunc (UG), ugplot (UG)

BUGS

This program works well only for objects centered about the origin.

AUTHOR

Jeff Mock

NAME

Ugsweep - a sweep generator for UNIGRAFIX

SYNOPSIS

ugsweep [[**–n** *number*] *transformations*] [*options*] < input > ouput

DESCRIPTION

Ugsweep takes faces and wires in a flat UNIGRAFIX format and produces a UNIGRAFIX description of the bodies (for faces) and faces (for wires) that result from sweeping those faces and wires through space. The sweeping is done according to the specified transformations. It treats one face or wire at a time to produce its swept envelope. Transformations are specified as follows:

–n *<number of repetitions>* Default: n = 1.
Repeat the subsequent set of transformations up to the next –n flag the specified number of times. Note that when a new transformation starts the coordinate system that aplies for the new transformation is the last coordinate system transformed by the previous transformation.

–sx, –sy, –sz, –sa *<factor>*
Scaling of the corresponding coordinate components.

–tx, –ty, –tz *<amount>*
Traslation along corresponding coordinate axis.

–rx,–ry,–rz *<angle>*
Rotation around corresponding axis

–M3 *<3x3 Matrix>*
Takes up to 9 numbers as a transformation matrix

–F Omit the initial face. Does not affect wires.

–L Do not built closing face. Does not affect wires.

–T Swept surfaces are to be tessellated with triangles rather then with possibly non-planar quadrilaterals.

–B Produce *both* sides of all faces. Always true for wires.

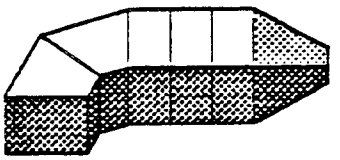
–fc *<filename>*
Use *filename* for command-line arguments.

–fi *<filename>*
Use *filename* as input. Default is standard input.

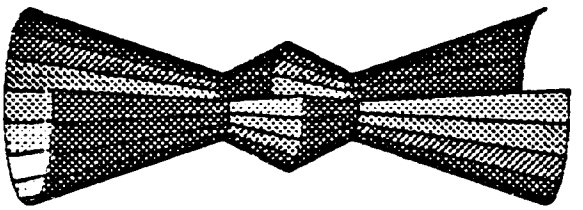
–fo *<filename>*
Use *filename* as output. Default is standard output.

EXAMPLES

```
ugsweep -fl square -n 2 -tx -2 -ry 45 -tx 2 -n 3 -tz 1 -n 1 -tz 6 -sa 0.3 > left
cat sun left | ugshow -ed 0 1 -1 -sa -dv -sx 3
```



```
ugsweep -fl wireW -D -n 14 -rx 20 > right
cat sun right | ugshow -sa -ed -1 0 -4 -dv -sx 3
```



FILES

~ug/bin/ugsweep
~ug/src/UGSWEEP/*

SEE ALSO

ugstick (UG), ugrot (UG), ugplot (UG).

BUGS

Will be found soon.

AUTHOR

Ziv Gigus

NAME

ugtrunc – truncate a polyhedron

SYNOPSIS

ugtrunc [**–t** 0.x] [**–T**] [**–C**] < oldobject > newobject

DESCRIPTION

Ugtrunc reads a flat UNIGRAFIX description and forms, on the standard output, the UNIGRAFIX description of a polyhedron that results by truncating the one read in.

Without the **-t** option, a new vertex is formed in the middle of every edge, and these new vertices are linked in a circular manner around every old vertex to form the new faces. It may result in nonplanar faces.

–t <real> Default: t = 0.999

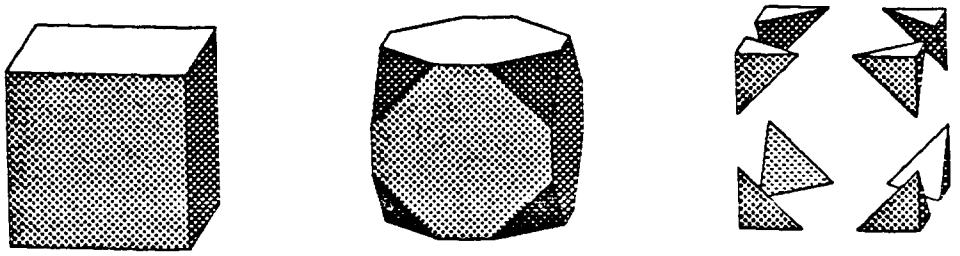
This option guarantees planar truncation faces. The numeric value following the **-t** specifies the amount of truncation in the range 0 to 1, where the value 1 corresponds to the middle of each emerging edge, and for smaller values the truncation is proportionally reduced. First, an approximate plane is placed through all the vertices determined in this manner on the edges emerging from a particular vertex; then the truncation plane is moved through the new vertex that minimizes the distance of the plane from the old vertex (to guarantee that the truncation on every edge is less than 1). Vertices with emerging edges that occupy more than a half-space (saddle points) will not get truncated.

–T This option returns the truncated tips instead of the remaining body.

–C This option adds the special truncation color CT to each truncation face.

EXAMPLES

```
cat ~ug/lib/illum ~ug/lib/cube | ugtrunc -t0.7 -C | ugplot -ed -2 1 -5 -sa -dv -sy 3
cat ~ug/lib/illum ~ug/lib/cube | ugtrunc -t0.7 -T | ugplot -ed -2 1 -5 -sa -dv -sy 3
```



FILES

~ug/bin/ugtrunc
~ug/src/PASCAL/trunc.p, ~ug/src/PASCAL/readug.i

SEE ALSO

ugdual (UG), ugshrink (UG), ugplot (UG)

BUGS

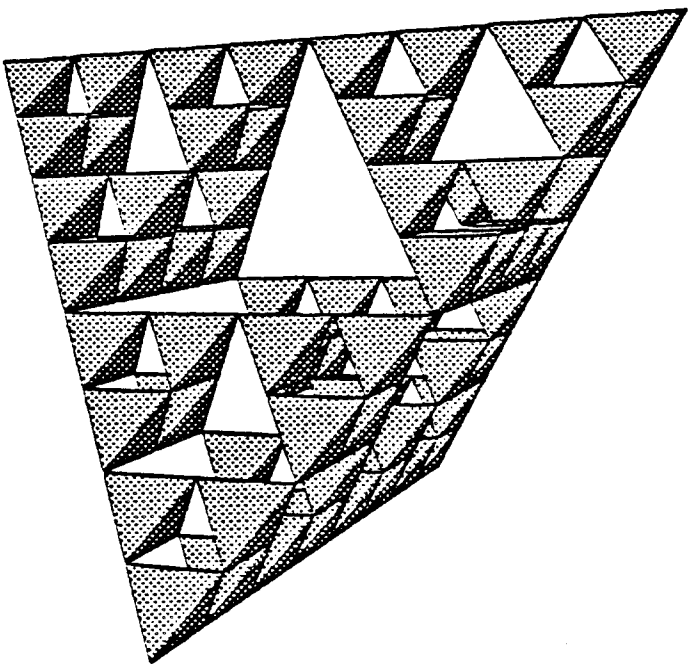
Yet to be reported.

AUTHOR

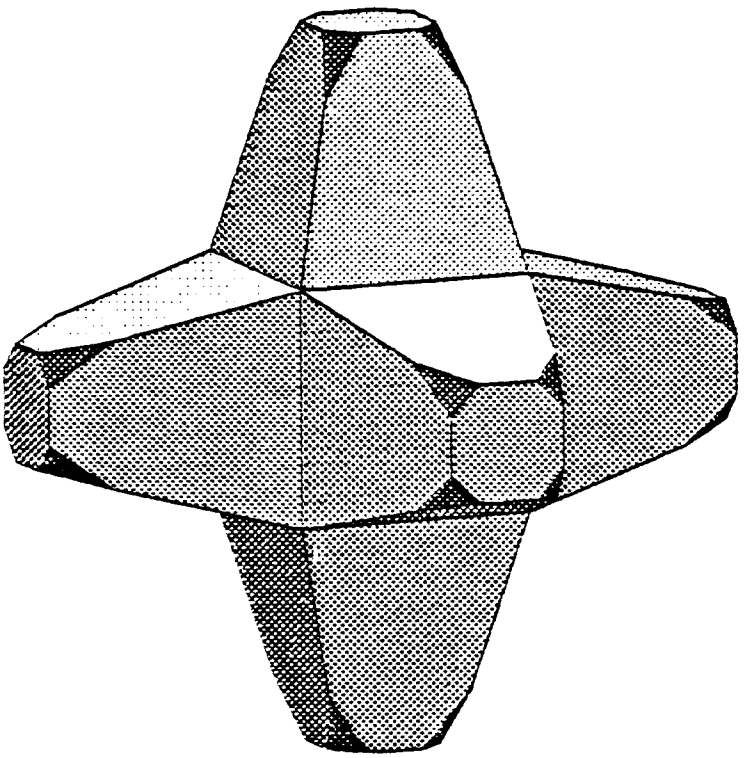
C.H. Séquin

Tutorial Examples for ugtrunc

cat illum ../lib/tetra | ugtrunc -t0.999 -T | ugtrunc -t0.998 -T | ugtrunc -t0.996 -T | ugshow -sa -dv -ep -2 0.5 -5 -sy 4 -mi -20 5 -50 55 58



cat illum ../lib/cube | ugstar -h 4 | ugtrunc -t 0.99 | ugtrunc -C -t0.5 | ugshow -sa -ed -2 1 -5 -dv -sy 4



NAME

ugwire – create the wire-frame of a polyhedron

SYNOPSIS

ugwire [**–t** 0.x] < oldobject > newobject

DESCRIPTION

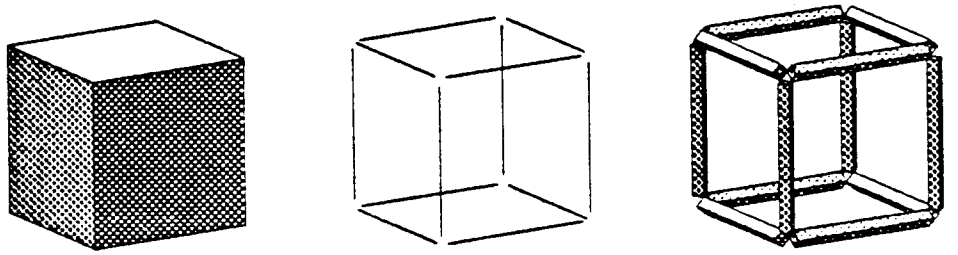
Ugwire forms, on the standard output, the UNIGRAFIX description of the wire-frame of the polyhedron read in. The wires sections are disassembled at the corners and can be shortened by a specified amount. They can be used as ax-input to the mkworm program. It understands only flat UNIGRAFIX descriptions. It deals with vertex and face commands, passes through color definition and light sources and ignores all other commands.

–A This option will output the wires as ax-segments and joints suitable for the mkworm program. This can be used to create a stick-approximations of a polyhedron, but with un-mitred corners.

–t <real> Default: t = 0.5.
This option specifies the amount of truncation applied to each wire segment. It pulls the wires apart, so that mkworm will not produce intersecting faces, when run with a small enough radius. Default is 0.5, which means that every wire gets shortened at both ends by 0.5 unit lengths.

EXAMPLE

```
cat ~ug/lib/cube | ugwire -A -t0.1
mkworm -r0.1 -n4;  cat ~ug/lib/illum  worm  | ugplot -ed -2 1 -5 -sa -dv -sy 3
```



FILES

~ug/bin/ugwire
~ug/src/PASCAL/wire.p ~ug/src/PASCAL/readug.i

SEE ALSO

mkworm (UG), ugtrunc (UG), ugplot (UG)

BUGS

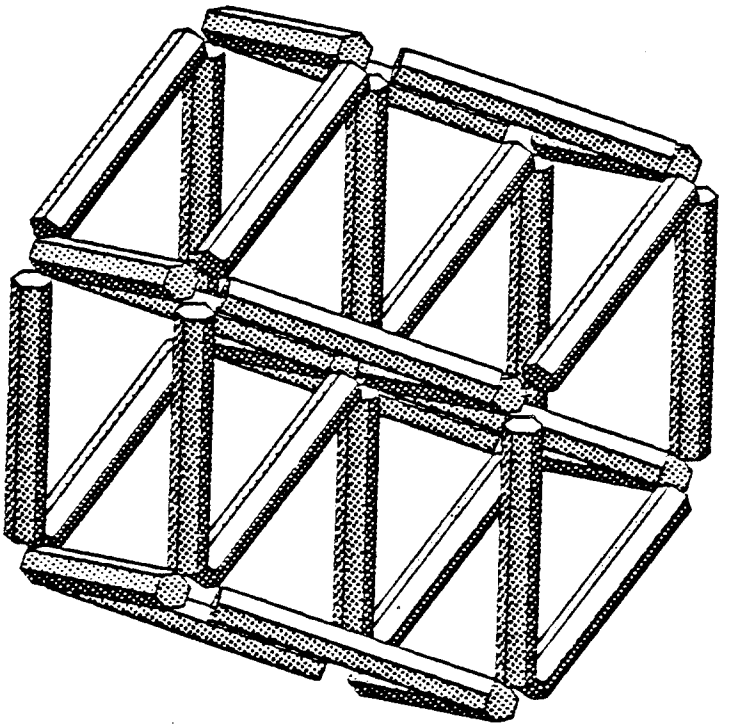
Yet to be reported.

AUTHOR

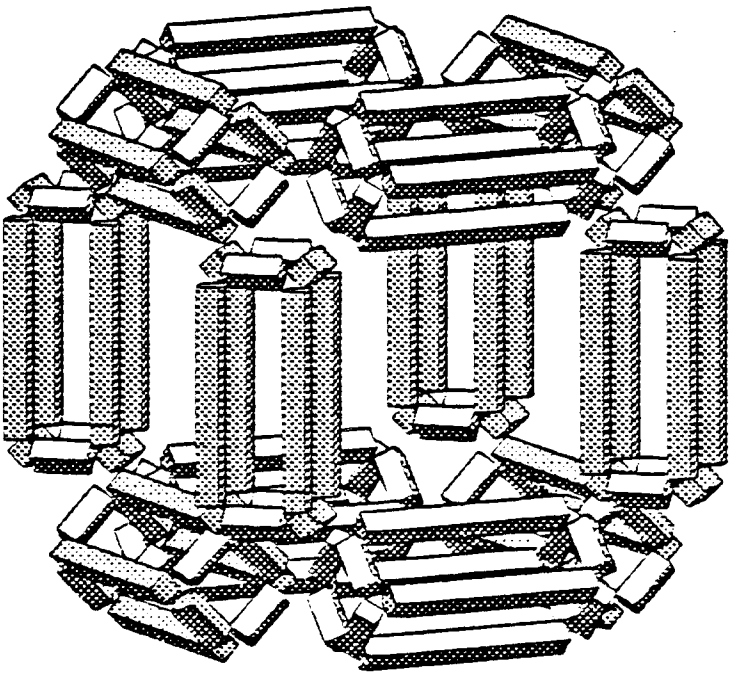
C.H. Séquin

Tutorial Examples for ugwire

```
cat D4cube | ug4to3 -ed 2 2 2 2 | ugwire -t 0.15 -A:   mkworm -n 6 -r 0.1
cat illum  worm | ugshow  -ed -5 2 -10 -dv -sa -sy 4
```



```
cat cube |ugwire -t 0.4 -A:   mkworm -n 4 -r 0.3
cat worm | ugwire -t 0.075 -A:   mkworm -n 4 -r 0.08
cat illum  worm | ugshow  -ed -5 2 -10 -dv -sa -sy 4
```



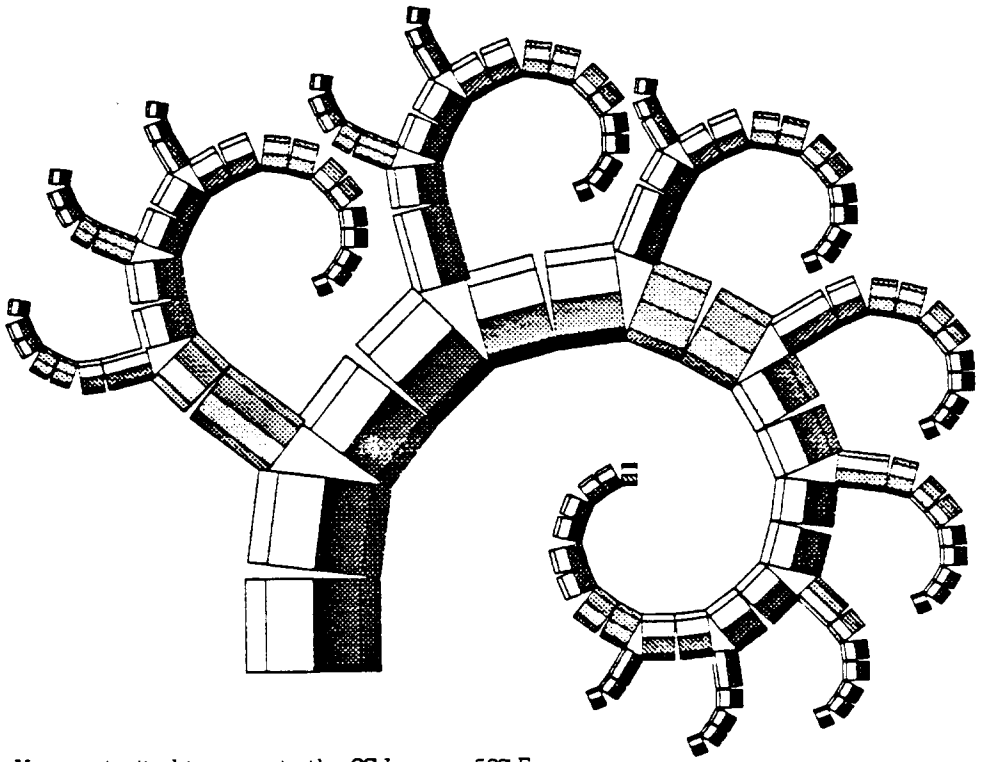
3. ARTWORK

Using a combination of the new programs, some crafty hand-editing of the UNIGRAFIX description at some stage of the transformation process, as well as clever arrangement of objects generated with different runs, the students managed to create artistic displays – some of which are reproduced here. In each case a short explanatory paragraph gives some information about the concept and the necessary steps that were used to produce the artwork.

Somewhat more elaborate versions, often ranging in size up to three feet, were exhibited at an UNIGRAFIX Art-show in the CS Lounge on December 16, 1983. (see poster below). We hope that the presentation of these examples inspires the reader to make creative use of UNIGRAFIX.

UNIGRAFIX

ART-SHOW & RECEPTION



You are invited to come to the CS Lounge, 597 Evans,
on Friday afternoon of Finals Week (Dec. 16, 1983).

The 292A Class, "Creative Geometric Modeling",
will present its results ranging from computer-grown trees
through geodesic domes to the skeleton of a Klein-bottle.
You will also find out what happens to all the cardboard
tubes that carry the paper for the versatec printer.

The "Art Gallery" will be open from 1pm to 4pm.

Design by Pauline Ts'o

CHS 83/12/9

" SKELETON OF A KLEIN-BOTTLE "

Carlo H. Séquin

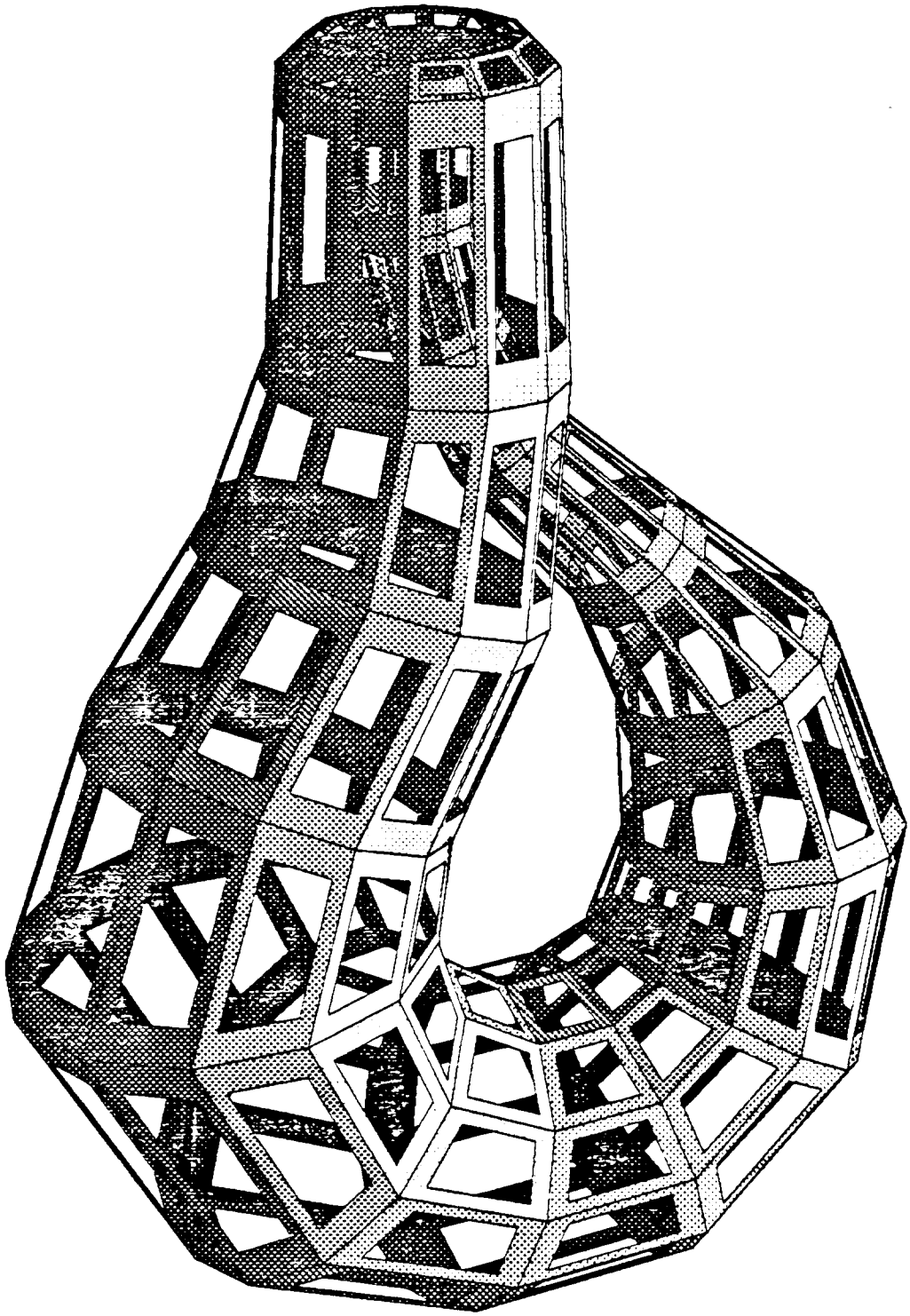
Computer Science Division
Electrical Engineering and Computer Sciences
University of California, Berkeley, CA 94720

A Klein-bottle is an object with a single surface – no inside and outside – and without a rim. This particular Klein-bottle was defined by specifying the joint coordinates of the axis of the main tube and the radius of the tube at each joint. These ax-specifications were then fed to the program *mkworm* that formed the main body. The specifications were changed by trial and error until a pleasing shape resulted.

A special filter was then used to remove the intersections of the faces where the thinner part of the tube penetrates the side-wall of the thicker tube to create a consistent UNIGRAFIX description. Finally the whole structure was sent through *ugshrink* *–H* to cut holes into each surface facet so that one can gain some insight into the non-existing inside. The *–mu* -option of *ugshow* darkens the more distant wall.

" SKELETON OF A KLEIN-BOTTLE "

Carlo H. Séquin



" icosaèdre arabique "

Carlo H. Séquin

Computer Science Division
Electrical Engineering and Computer Sciences
University of California, Berkeley, CA 94720

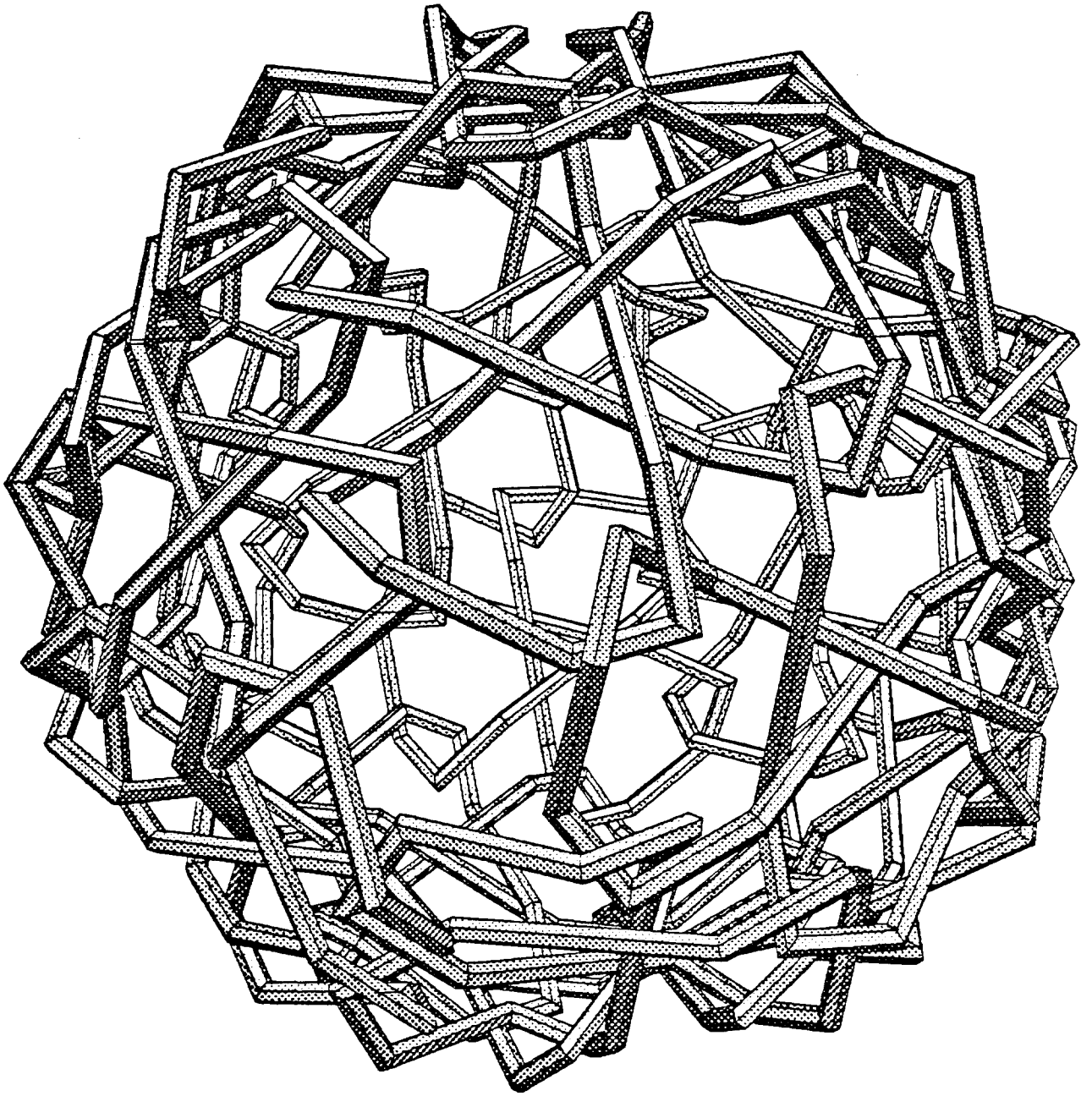
It may be far from obvious, but this ball of worms was created conceptually by wrapping the carved pattern from the window pane of an arabic mosque around an icosahedron.

To realize this concept, the ax of a worm in the shape of a clover-leaf knot was first defined. Two such worms were then interlocked with two pairs of their loops and with the proper angle between their respective planes to match the dihedral angle of the icosahedron. The waviness and the diameter of the worms were then carefully balanced so that the worms do not intersect.

With the individual clover-leaf thus defined, twenty instances were called with the proper angle of rotation around the center of the icosahedron to form a symmetrical object with twenty interlocking clover-leafs. The *-mi* -option was used in *ugshow* to reduce the contrast on the branches on the other side of this icosaèdre arabique.

" icosaèdre arabique "

Carlo H. Séquin



" NESTED OCTAHEDRONS "

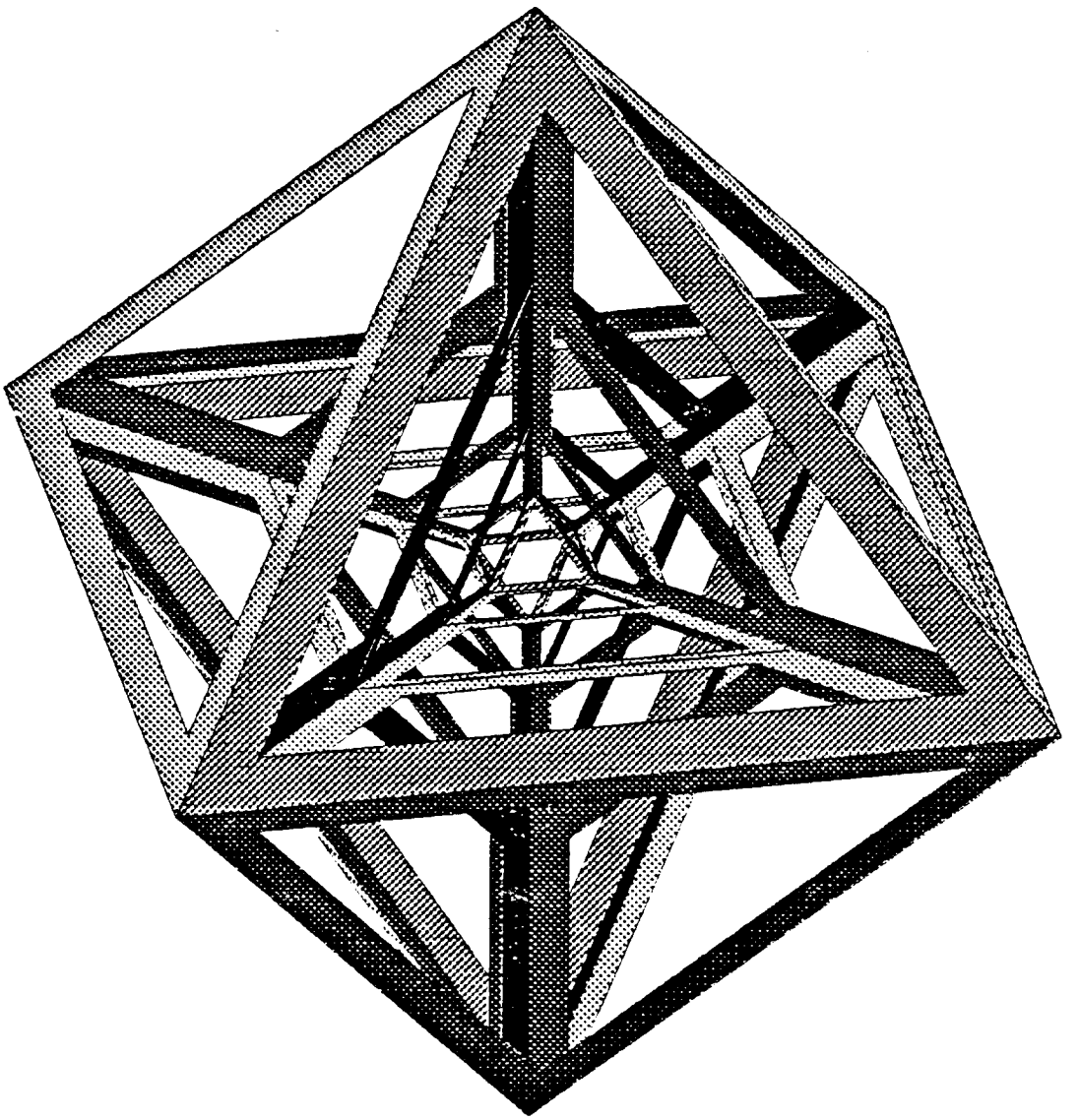
Ziv Gigus

Computer Science Division
Electrical Engineering and Computer Sciences
University of California, Berkeley, CA. 94704.

The basis for the figure shown is the simple octahedron. Using *ugsweep* the faces of the octahydron were swept towards its center so that each face becomes part of a triangular prism consisted of three joints. Next holes were created in each of the faces using *ugshrink* and back faces were added to get the result presented here.

" NESTED OCTAHEDRONS "

Ziv Gigus



" STAR PATTERNS OF AN DODECAHEDRON "

Edward Hunter

Computer Science Division
Electrical Engineering and Computer Sciences
University of California, Berkeley, CA 94720

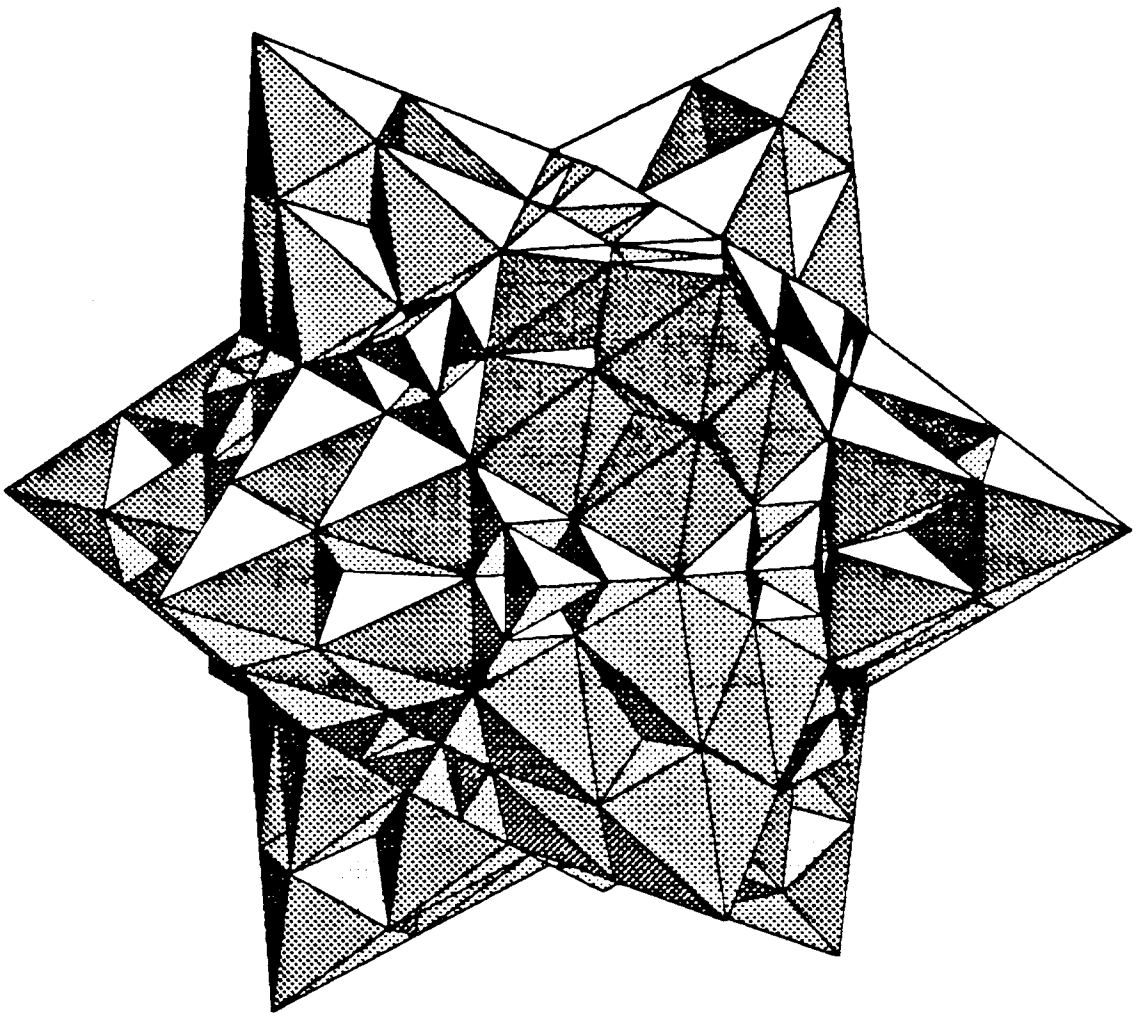
ABSTRACT

The figure shown was formed by taking a dodecahedron and growing points on the faces using *ugstar*. Next dimples were put on the sides of the points such that the depth of the dimples is proportional to the perimeters of the faces on which they reside.

Multiple light sources were used in order to give some faces highlights while allowing others to be shaded.

" STAR PATTERNS OF AN DODECAHEDRON "

Edward Hunter



" ICOSAHEDRON STARFISH "

Edward Hunter

Computer Science Division
Electrical Engineering and Computer Sciences
University of California, Berkeley, CA 94720

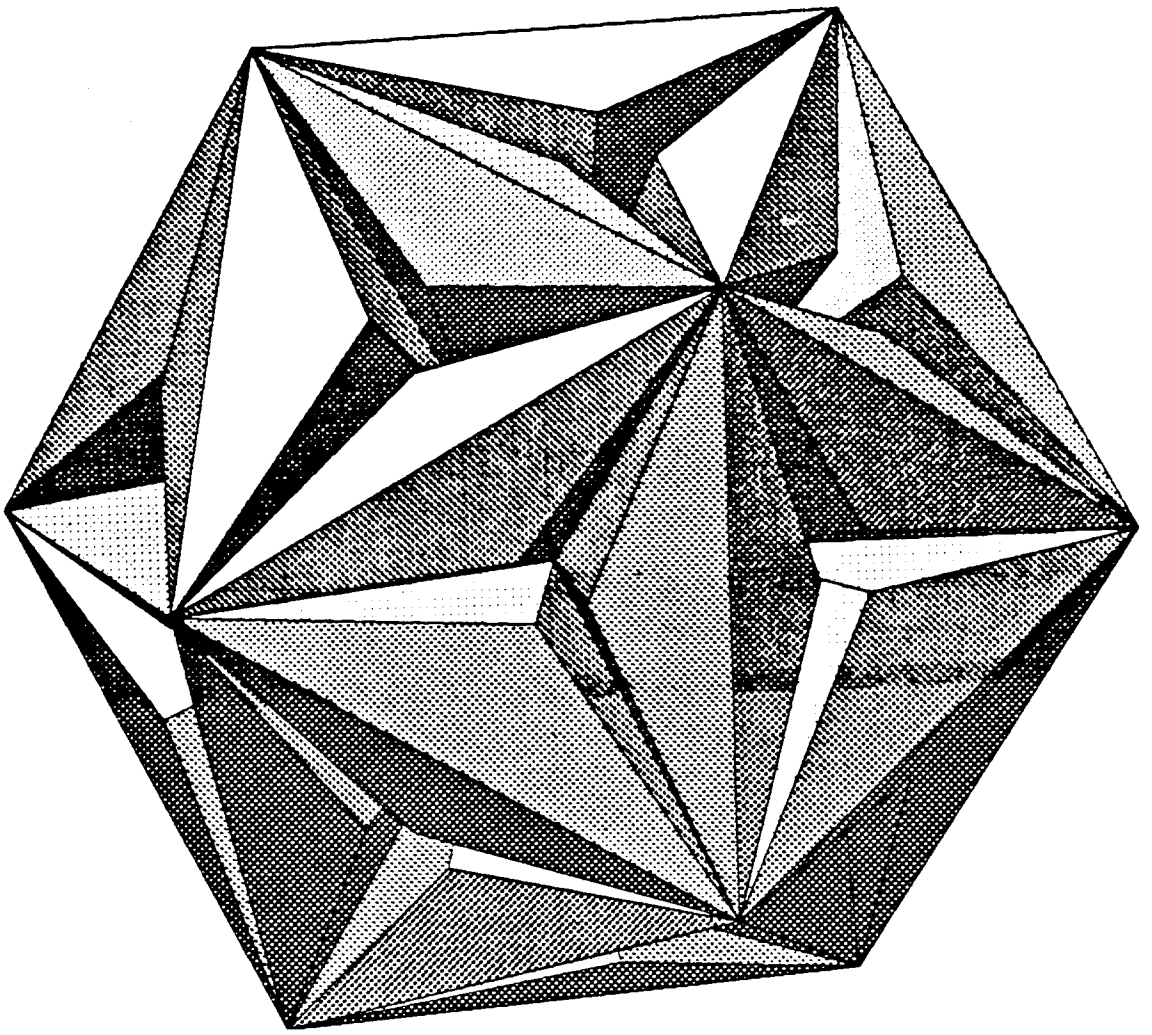
ABSTRACT

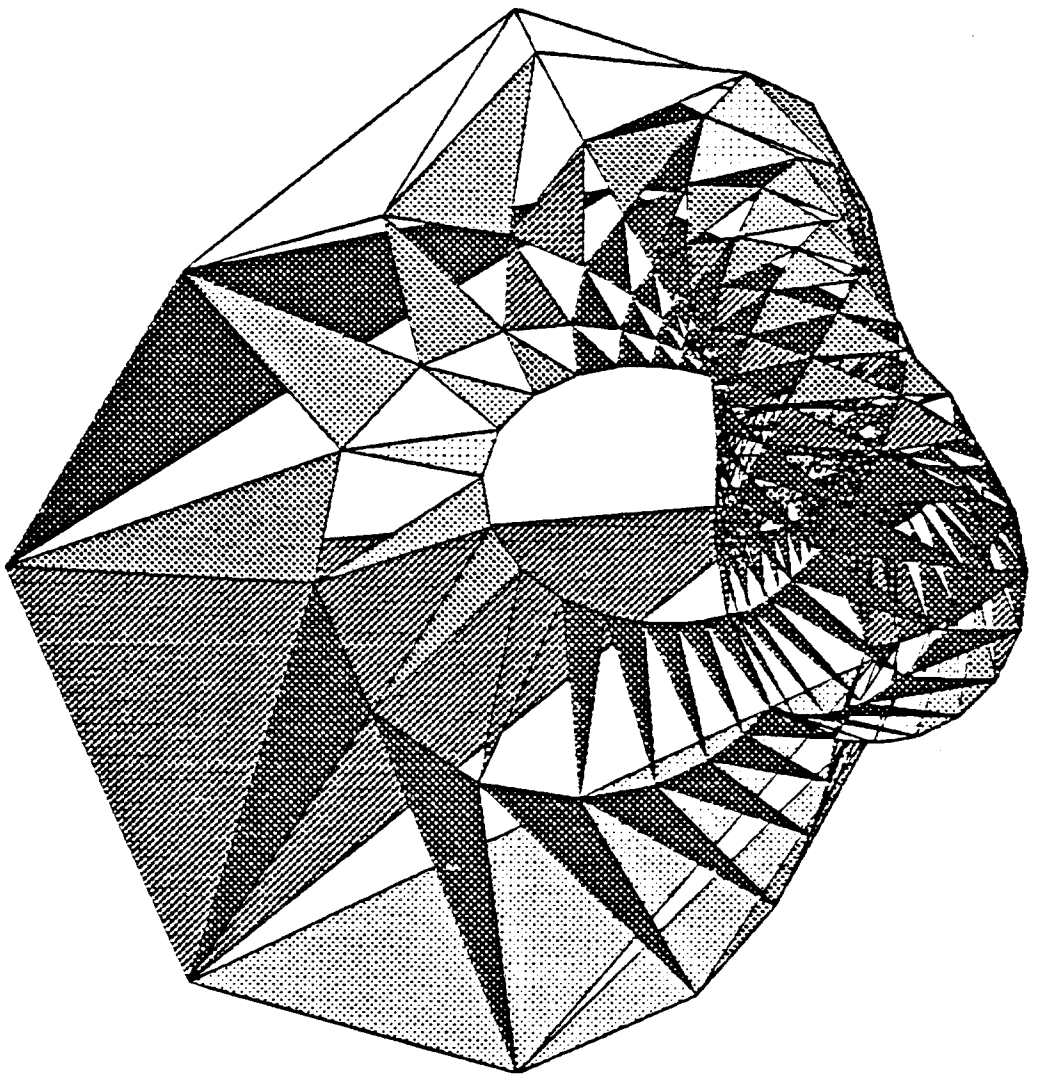
The figure shown was formed by taking an icosahedron and dimpling the faces using *ugstar*. Next peaks were grown on the sides of the dimples such that the height of the peaks is proportional to the perimeters of the faces.

The lighting was adjusted to give all of the faces a slightly shaded look.

" ICOSAHEDRON STARFISH "

Edward Hunter





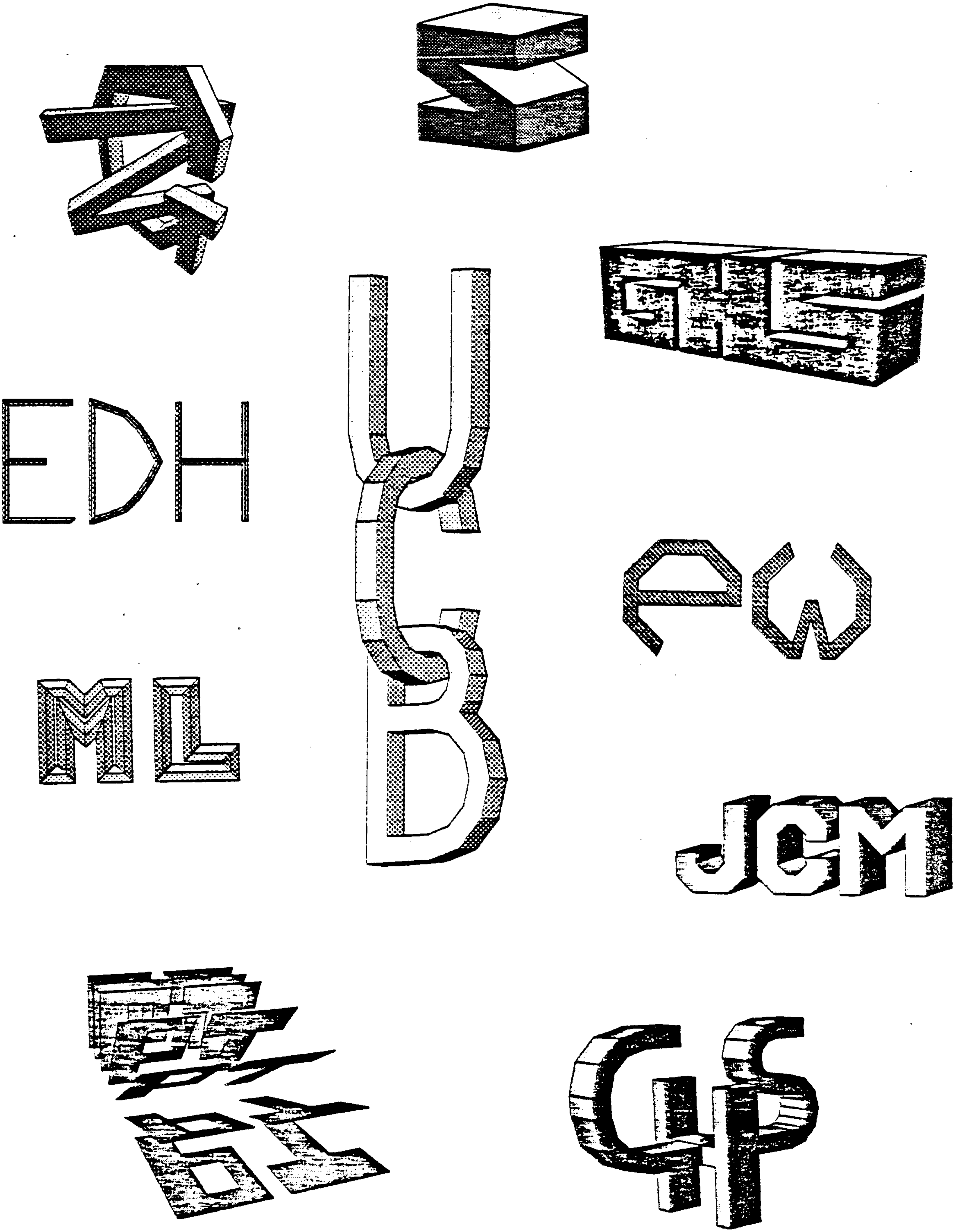
--- and now it is your turn !

BIBLIOGRAPHY

This is the beginning of a rather informal and incomplete collection of book titles which I found to be all very interesting and helpful in my research in the area of creative geometric modeling. Contributions of additional titles and suggestions for improving the synoptic description of these works are welcome.

References

1. M.J. Wenninger, *Polyhedron Models*, Cambridge University Press, Cambridge, England, 1971. Evans Hall Library: QA491 W391 – Contains 119 photos of paper models of all the Platonic and Archimedean Solids and of many of their stellations. Contains explicit instructions and nets for building the models.
2. M.J. Wenninger, *Sperical Models*, Cambridge University Press, Cambridge, England, 1979. Evans Hall Library: QA 491 W43 – Contains 47 photos of paper models of all the sperical frame projections of the Platonic and Archimedean Solids and of many of their stellations. Contains explicit instructions and nets for building the models.
3. H.S.M. Coxeter, *Regular Polytopes*, MacMillan Co., New York, 2nd edition, 1963. Evans Hall Library: QA691 C68 – Detailed discussion of regular and semiregular polyhedrons in all dimensions.
4. H.S.M. Coxeter, *Regular Complex Polytopes*, Cambridge University Press, Cambridge, Great Britain, 1974. Evans Hall Library: QA691 C661 – Profound geometric, algebraic, and group theoretic treatment of regular solids and lattices in all dimensions.
5. L. Fejes Tóth, *Regultäre Figuren*, Ungarische Akademie der Wissenschaften, Budapest, 1965. Evans Hall Library: QA601 F3815 – Good construction for the 4D regular polytopes. Interesting 3D stereograms of 3D semiregular solids.
6. A. Pugh, *Polyhedra, A Visual Approach*, University of California Press, Berkeley, CA, 1976. Evans Hall Library: QA491 P831 – A concise, simple and very attractive introduction to the regular and semi-regular solids and their relationships.
7. D.W. Brisson, *Hypergraphics*, AAAS Selected Symposia Series, Westview Press, Boulder, Colo., 1978. Doe Library: QA491 H97 – Visualization of the commplex relationships in art, science and technology.
8. R. Williams, *The Geometrical Foundation of Natural Structure*, Dover Publications, Inc., New York, N.Y., 1979. Found at Cody's. – This work is subtitled: "A source book of design". It indeed contains a rich set of ideas about the interrelationships of the various Platonic and Archimedean solids and transformations leading from one to the other. Particularly strong in the issues of subdivision of 3-D space with these solids.
9. A.L. Loeb, *Space Structures*, Addison-Wesley Co., Reading, Mass., 1976. Evans Hall Library: QA491 L631 – Nice coverage of 2-D tesselations and space-filling polyhedra. Intuitive treatment of vertex and edge truncations.
10. B.M. Stewart, *Adventure Among the TOROIDS*, B.M. Stewart, Okemos, Mich., 1970. Evans Hall Library: QA491 S75 – A mind-boggling book, full of crazy constructions of toroidal solids bounded by equilateral polygons.


EDH
UCB
ML
JCM