UNIGRAFIX 2.0

User's Manual and Tutorial

Carlo H. Séquin, Mark Segal, Paul Wensley

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

ABSTRACT

UNIGRAFIX, as the name implies, is a graphics system that runs under the UNIX operating system. It consists of a descriptive language and various programs that allow a user to create, modify, and display scenes consisting of polyhedral objects.

The UNIGRAFIX language is in a terse, human-readable format that allows *scene files* of complex objects to be created with little effort. These scene files may be created manually with use of a text editor, or may be output by special-purpose generator programs (for more complex scenes). Once created, scenes can be used as input to the UNIGRAFIX rendering programs. These programs can be run either as separate functional units, or from within the interactive UNIGRAFIX environment. Programs are also available to transform and display the scenes according to user specifications.

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

1. OVERVIEW

UNIGRAFIX, as the name implies, is a graphics system that runs under the UNIX* operating system. It consists of a descriptive language and several programs that allow a user to create, modify, and display scenes consisting of polyhedral objects.

The UNIGRAFIX language is in a terse, human-readable format that allows *scene files* of complex objects to be created with little effort. These scene files may be created manually with use of an editor, or may be output by a special-purpose generator program (for more complex scenes). Once created, scenes can be used as input to the UNIGRAFIX programs. These programs can be run either as separate functional units, or from within the interactive UNIGRAFIX environment. Programs are available to transform and display the scenes according to user specifications.

Typical use of UNIGRAFIX starts with creation of a scene file, which can be transformed with the **ugxform** command until satisfactory. Then, the **uglook** command can be used to determine the optimum view for the scene interactively on a display terminal. Several different display devices are supported. When displaying to a CRT terminal, it is advisable to have one with graphics capability, such as an HP 2648A or AED 512. On "dumb" (i.e. non-graphics) terminals, display output will be coarser and more difficult to interpret. Finally, **ugplot** or **ugshow** may be used to produce the desired output in hardcopy.

UNIGRAFIX allows a scene to be displayed in different ways. Display of a scene may be in *wire-frame* format, where all of the edges of objects appear as lines, or faces may be shown as shaded areas, with hidden surfaces removed. It is also possible to combine these two types, and display only edges, but with hidden surfaces removed.

2. UNIGRAFIX SCENE FILES

Input for UNIGRAFIX is in the form of *scene files,* which contain descriptions about the objects in the scene. These files are in readable ASCII format, so that it is possible to create and modify them with any text editor.

Scenes may contain any number of *objects,* each of which is some sort of polyhedral body. This means that all objects can consist only of straight lines and planar polygonal surfaces. These lines and surfaces are defined in the UNIGRAFIX language as *wires* and *faces,* both of which pass through a series of *vertices.* Vertices are defined in terms of a three-dimensional coordinate system. In addition, it is possible to group vertices, wires, and faces together as a *definition,* which can then be used in one or more *instances* or *arrays.* These constructs permit easy repetition of objects within scenes. Finally, it is possible to specify sources of illumination to obtain shaded output.

2.1. The Coordinate System

UNIGRAFIX uses a *left-handed* coordinate system, with the positive x-axis directed to the right, the positive y-axis going upwards, and the positive z-axis pointed away from the viewer. Units in the coordinate system are arbitrary, as the size of the coordinate system is unbounded. It is possible to have one scene in which all objects lie between -1 and +1 in all dimensions, and another in which the coordinates range into the millions; both scenes will be scaled appropriately However, relative sizes of objects within the same scene are preserved.

By default, the origin of the coordinate system is used as the *view center,* which means that the axis of a perspective projection will go through this point. The default *view direction* is along the z-axis, looking towards positive z (away from the viewer). These parameters may be changed when the scene is displayed, but it is often convenient to structure a scene so it can be viewed with the default set-up.

*UNIX is a trademark of Bell Laboratories.

2.2. UNIGRAFIX Language Description

The UNIGRAFIX language consists of statements for specifying vertices, edges, faces, etc..

Each of these statements allows a unique *identifier* (id) to be associated with an object. For certain objects, an identifier is *required*; for others it is *optional*. Identifiers are string names of any length, which begin with a letter or sharp sign ("#"), and contain only letters, digits, sharp signs, underscores, colons, and periods. (The reasons why several of these characters are included will be explained later on). Identifiers for objects of each type (e.g. vertices) must be different, although the same id may be used for objects of different types.

All UNIGRAFIX statements follow these general conventions:

Statements begin with a keyword and end with a semicolon.

White space (blanks, tabs, and newlines) is ignored, except as a delimiter between items.

Statement keywords, string names, and other character items are terminated by white space.

Numbers (integers and real numbers) are terminated when they stop looking like numbers. For example, if an integer is needed, and **"123a"** is the next item, the number **123** will be used; **"a"** will be used as the next item.

All characters with special meaning to UNIGRAFIX ("{", "}", ";", etc.) do not need to be separated by white space.

In the specifications below, the following additional conventions are used:

Any item listed in **boldface** type is literal. Items listed in *italics* denote values to be specified by the user.

Item listed within square brackets ("[" and "]") are optional.

2.2.1. Vertices

Vertices are the building blocks of UNIGRAFIX scenes; everything else is defined in terms of them. The **vertex** statements denote the points through which edges and faces pass:

v *id x y z* **;**

The *id* is an identifier (described previously) that is different for all vertices in one definition or scene. The *x*, *y*, and *z* values represent the coordinates of the vertex in three dimensions. These values may be either integers or floating-point numbers.

Examples of valid **vertex** statements are:

```
v origin 0 0 0;
v v12 1.2 30 5e-2;
```

2.2.2. Wires

Wires are line segments or sequences of segments that pass through two or more vertices. The simple **wire** statement looks like this:

w [*id*] **(** v1 v2 ... vn **)** [*ColorId*] **;**

The *v1 v2 ... vn* arguments are the *id*'s of the vertices, in order, that the wire is to pass through, starting at the first vertex listed end ending on the last one. If the wire is to form a closed polygon, the first vertex should be repeated at the end of the list. If the *ColorId* argument (see below) is given and has been previoulsy defined, it will be used when the wire is displayed on a device that makes use of color.

All vertices referenced by the **wire** statement must have been specified previously. No forward references are allowed.

Examples of valid **wire** statements are:

```
w  W12  ( first_vertex second_vertex );
w  closed_loop  ( v1 v2 v3 v4 v5 v6 v7 v8 v9 v1 );
w  ( origin v12 );
```

Multiple trains of wire, all associated with the same id and color, can be created with multiple sequences of vertex id's. For example,

```
w  tetra  ( left right top back )  ( right back left top ) red;
```

creates the 'red' wire frame of a tetrahedron in a single statement.

2.2.3. Faces

A face is a closed polygon whose boundary is defined by three or more vertices. The simple **face** statement is:

f [*id*] (v1 v2 ... vn) [*ColorID*] [*lightness*] [**<** *A B C D* **>**] **;**

The arguments correspond to those used in the **wire** statement. One difference is that the list of vertices is assumed to be closed, so that there is no need to re-specify the first vertex at the end of the list. For faces, there must be at least three vertices. The *ColorID* argument is interpreted as in the case of wires. A *lightness* value is specified as a number between 0 and 1, and if present will be used as the lightness of the face, regardless of it's orientation or light sources.

The remaining arguments are auxiliary fields that need not be specified. They can be used to achieve special effects and are sometimes helpful in debuggung. Lightness is a light value forced onto this face. The angle-bracketed arguments denote the coefficients of the plane equation of the face: $Ax + By + Cz + D = 0$. These may also be left out and the equation will be computed by UNIGRAFIX.

Some valid **face** statements are:

```
f  triangle  ( lower_left top lower_right );
f  unusual  ( v1 v2 v3 v4 v5 v6 )  green  < 0 0 1 2 >;
f  ( A B C )  0.75 ;
```

In the second example, the color of the face is *green*, and the plane equation is defined to be $z + 2 = 0$, regardless of what the coordinates of its vertices are.

In some cases, faces with "holes" in them are useful. As in the case of wires, multiple sequences of vertex id's can be used. Each sequence in parentheses is interpreted as a separate closed polygonal edge. Thus, one can define an "inside edge" of a face with a hole by reversing its orientation. For example,

```
f  square_ with_ triangular_ hole  ( out1 out2 out3 out4 ) ( in1 in2 in3 );
```

It is possible to put other edges inside a hole, creating new pieces of the face, and so on. Even slits with only two vertices and point holes to pass wires through are allowed:

```
f  face_with_slit_and_pinhole  ( o1 o2 o3 o4 ) ( sA sB ) ( p_h );
```

Some care must be taken when defining faces. First of all, the order in which a face's vertices are listed defines which side of the face is the "inside," or visible side. The rule for determining the "inside" of a face is that any contour traced in the clockwise direction encloses a visible portion of the face, while any material not enclosed by such a contour is an "outside", or invisible portion of the face (either the area outside the face, or a hole).

Secondly, the face's vertices should all lie in one plane. This is necessary so that the face can be described with one plane equation, with no unexpected consequences. UNIGRAFIX may complain a great deal if faces are not planar. Newell's algorithm is used to determine face normals; it averages face normals across the vertices defining the outside contour of the face. Therefore, the first set of vertices should define an outside contour, insuring correct determination of the normal's sign.

In accordance with the rule that defines insides and outsides of faces, the vertices on the inside contour of a hole (e.g *in1* - *in9* in the above face) must be listed in counterclockwise order.

2.2.4. Light sources

If scenes are to be shown as shaded faces, there should be some illumination present in the scene. The **light-source** statement creates sources of light to be used for this purpose:

l [*id*] *intensity* [*x y z* [*h*]] **;**

The values of *intensity* and *x, y, z,* and *h,* are integers or floating-point numbers. There are three types of light sources: *ambient* sources (those with no illumination vector specified), *directional* sources (those with *x, y* and *z* values specified), and (in the future) point light sources (those with the homogeneous coordinate value *h* specified and not equal to zero.

Ambient light sources cause equal effects on all faces, regardless of their orientations. Each face will receive the full intensity of the source. Since the lightness value for a face can only range from 0 to 1, the most meaningful values for *intensity* lie in this range. Values outside this range will be clipped to either 0 or 1. For *directional* sources, the *x, y,* and *z* arguments define a vector which points from the origin towards the source of the illumination. This vector need not be normalized. The effect of a directional illumination source upon a face is computed by finding the angle between the illumination vector and the normal vector of the face (derived from the plane equation). The face will receive the full intensity of the source if this angle is zero (i.e. the source is perpendicular to the face), and will receive no illumination from the source if the angle is 90 degrees or greater.

The format makes future extension to *point* light sources possible. *Point* light sources are assumed to lie at coordinates (*x/h, y/h, z/h*). Their intensity is characterized by the intensity received by a point at distance one from them. They are not yet implemented in UNIGRAFIX.

When there are multiple light sources for the same scene, the effects on a particular face are cumulative. For each light source, the effect upon each face is calculated, and added to the current illumination value for that face.

Examples of valid **light-source** statements are:

```
l sun .7 10 30 -100;
l glow .2;
l 1;
```

2.2.5. Color

Color can be introduced into scenes by using the **color** statement. Color images can then be produced on devices which have color capabitilies. The **color** statement creates color identifiers which can be applied to faces, edges and higher level objects. The format of the statement is:

c *id lightness* [*hue* [*saturation* [*translucency*]]] **;**

The statement associates the given color specifications with the identifier; the identifier can then be used to specify color for various objects. The color system used is the same as Smith's HSV model. The first three parameters define a point in color space. Color space is defined to be a cone with saturated hues lying on the perimeter of the circle at the base of the cone. Moving into the circle decreases the saturation and adds more gray to the colors. Colors become dimmer (less intense) as one moves towards the tip of the cone. The tip of the cone is black.

Lightness should fall in the range zero to one. Zero represents black; a value of one places the color somewhere in the circle lying at the base of the color cone (as defined by hue and saturation). Values outside this range will be clipped to this range. *Hue* should be a number between 0 and 360, signifying the position on the color circle of the hue. 0 (and 360) is defined to be red; the circle progresses from red through yellow to green to cyan to blue to magenta and back to red. If *hue* is not in the specified range, its value mod 360 will be used. *Saturation* should also fall in the range zero to one; zero is unsaturated (gray) and one is complete saturation. *Translucency* refers to the property of a color when an already colored object is recolored in an instance statement. A *translucency* of 0 means the new color covers the underlying one completely. The other extreme is the value 1 where the underlying color shines through unchanged. Currently, *translucency* is accepted but ignored; the mixing process has not yet been implemented.

Not all values need be specified. If only one value is specified, it is interpreted as a gray value on a colorless face. If *lightness* and *hue* are present, the color will be assumed to have a saturation of 1. The default *translucency* is 0.

When a colored object is displayed on a color output device, color and illumination are combined to produce a shaded object. Therefore many shades (more correctly, lightness values) will usually appear in a displayed object even if only one color is defined.

Although an infinite palate of colors is specifiable, real devices can only display a fixed number of colors. UNIGRAFIX takes whatever color has been specified, and after combining it with illumination, rounds it to the nearest color in the device's color map. The rounding method is device dependent and is currently not accessible to the user. On devices incapable of color display, the only color field which has any meaning is lightness; hue and saturation are ignored. In later versions, this should be modified, since some hues (yellow) appear lighter than others (blue).

2.2.6. Definitions

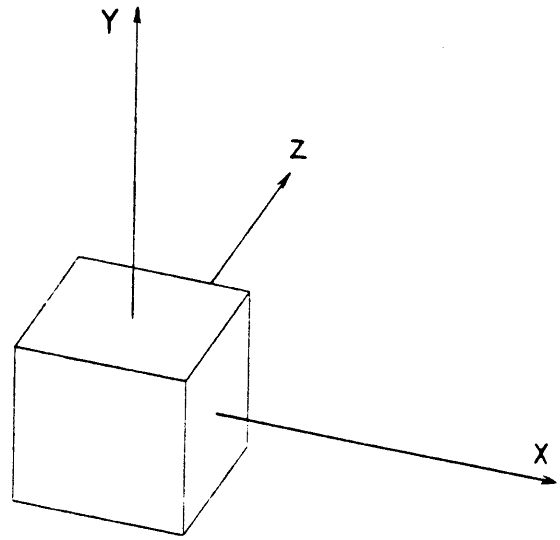
Much of the power of the UNIGRAFIX language comes from the use of the **definition** statement. It allows objects to be grouped together and given a common name. After definition, the **instance** and **array** statements can be used to place copies of the definition in any location in the scene. The statement itself is very simple:

> **def** *name* ;
> *statements* ...
> **end**;

The name of the definition may be anything that would be a valid identifier. The *statements* may be anything except **definition** statements (so nesting of definitions is not allowed), and **light-source** statements ("local" illumination will be ignored and a warning message printed). All the objects created by the *statements* become part of the definition; they do not actually become part of the scene until an **instance** or **array** statement calls for that definition. In a sense, the **definition** statement creates a "macro" that can be used as many times as desired.

The following is an example of a **definition** statement that defines a simple cube. Each face of the cube is assigned a letter name (**f** for "front," **b** for "back," **l** for "left," **r** for "right," **u** for "up," and **d** for "down"). The names of the vertices are derived from the three faces adjacent to it.

```
def cube;
  v lfd -1 -1 -1;
  v lfu -1 1 -1;
  v rfu 1 1 -1;
  v rfd 1 -1 -1;
  v rbd 1 -1 1;
  v rbu 1 1 1;
  v lbu -1 1 1;
  v lbd -1 -1 1;
  f f ( lfd lfu rfu rfd );
  f d ( lbd lfd rfd rbd );
  f b ( rbd rbu lbu lbd );
  f u ( lfu lbu rbu rfu );
  f l ( lbd lbu lfu lfd );
  f r ( rfd rfu rbu rbd );
end;
```



2.2.7. Instances and Arrays

Once a definition has been created, the **instance** and **array** statements can be used to place copies of the defined object into the scene. An *instance* is a single copy of the defined object, possibly with some transformations applied to it. An *array* is a multiple copy of the object, also with optional transformations, and specifications of how to separate the copies. The statements look like this:

i [*id*] (*defname* [*ColorID*] [*transformations*]) ;

a [*id*] (*defname* [*ColorID*] [*transforms*]) *size* [*transforms*] ;

The *id* is an identifier for the instance or array, while the *defname* refers to the name used for the object in the **definition** statement for that object. The two may be identical.

The *color* field, if specified, forces every face and wire in the object to be of that color, mixing in the object's original coloring if the specified color's *translucency* is greater than 0.

2.2.8. Transformations

Transformations allow an object to be *scaled* (changed in size in any or all dimensions), *translated* (moved in any direction), *rotated*, or *mirrored*. Rotation is by degrees about a given axis of the coordinate system. When *rotation* of objects is specified, the following cyclic conventions are used (assuming a left-handed coordinate system): Positive rotation about the x-axis is from the positive y-axis to the positive z-axis. Rotation about the y-axis is from the z-axis to the x-axis and rotation about the z-axis is from the x-axis to the y-axis. An easier way of remembering this is that positive rotation about any axis appears clockwise when viewed from the positive side of the axis, looking towards the origin. Mirroring is always performed about a specified axis. The formats to be used for the *transformations* field are:

-s?	*scale_ factor*	for scaling
-t?	*translation_ amount*	for translation
-r?	*rotation_ angle*	for rotation
-m?		for mirroring
-M3	*3x3 Matrix*	for linear 3-dimensional transformation
-M4	*4x4 Matrix*	for homogeneous 3-D transformation

The **?** should be replaced with **x, y,** or **z** to denote in which dimension to scale, mirror, or translate, or about which axis to rotate. As a shorthand way of specifying scaling or mirroring in *all* dimensions, the **?** can be replaced with **a,** for "all." This construct is not valid for translation or rotation.

When specifying a transformation in matrix form, from 1 to 9 numbers (for **-M3**) or from 1 to 16 numbers (for **-M4**) may be specified. The numbers specified replace the entries by rows, in a unity matrix of degree 3 or 4, respectively.

Transformations are applied to the defined object in the order given. Arguments may be integer or floating-point numbers. Care should be taken when mirroring or scaling by a negative number, since the orientation of vertices around a face may change as well. For mirroring, besides negating the appropriate coordinate value, the order of vertices in all affected faces is reversed. This is not true for negative scaling: the original order is maintained. Thus, a body might be turned "inside-out" if you do an odd number of negative scaling operations.

For the **array** statement, the *size* and second set of *transformations* must also be specified. The *size* is an integer that says how many copies of the object are desired. The second set of *transformations* is applied incrementally between individual copies. The first set of transformations is applied before the copies are made.

Examples of the use of instances and arrays are:

```
i bigrotcube ( cube -mz -sa 10 -rx 20 -ry -50 ) ;
a rowof3cubes ( cube -tx -10 ) 3 -tx 10 ;
a ( cube ) 10 -sa 0.5 ;
```

The first example creates an instance of the previously defined object *cube,* mirrors it in the z dimension (mirrors about the xy-plane), makes it larger by a factor of 10 in all three dimensions, rotates it around the x-axis by 20 degrees, and then around the y-axis by -50 degrees, and then adds the resulting object to the scene. The second example uses the same *cube,* but makes three copies of it. Each of the copies is separated by 10 coordinate units in the x dimension. The initial translation by -10 has the effect of centering the new object *(rowof3cubes)* at the same place where the original definition was centered. This is often useful, in order to keep a scene centered about a fixed point, usually the origin. The last example creates 10 concentric, scaled down copies of a cube.

2.2.9. Hierarchies of Instances, and Arrays

It is possible to create a hierarchy of more than one level by placing **instance** and **array** statements within definitions. For example, to put together three of the *rowof3cubes* defined in the previous section, the following can be done:

```
def rowof3cubes;
   a rowof3cubes ( cube -tx -10 ) 3 -tx 10 ;
end;
a squareof9cubes ( rowof3cubes -ty -10 ) 3 -ty 10 ;
```

This stacks the rows in the y direction to form a square.

As a rule, definitions must be self-contained. This means that all faces and edges within a definition may reference *only* those vertices within that definition.

2.2.10. Including Files

The **include** statement is a way of allowing modularity in UNIGRAFIX scene files. The format is:

include *filename* **;**

When this statement is encountered, the contents of the file *filename* are used as input. When end-of-file is reached, input continues at the line following the **include** statement. Included files may be nested, to a maximum of 16 levels. Recursive file loops are not checked for, and should be avoided. UNIGRAFIX recognizes the file metacharacter '~'. If a file is not found in the current directory, then the UNIGRAFIX library is searched. If a file is not in either place, a syntax error results.

2.2.11. Comments

Comments may be placed in UNIGRAFIX scene files anywhere that white space may occur. This includes almost every possible place except within identifiers or other names. Comments are surrounded by curly braces ("{" and "}"), and may contain any characters. Comments follow nesting rules, so that if two left-braces are used, two right-braces are necessary to end the comment. Examples:

```
{ This is a simple comment }
{ This is also a { more complicated } legal { using nested {}'s } comment }
```

3. UNIGRAFIX BATCH PROGRAMS

Unigrafix includes a series of programs that allow scene files to be displayed and modified until satisfactory output is achieved. These programs are *batch* programs, in that there is no user interaction after the program has been started with the proper parameters. Input to the programs comes from standard input, which in most cases will be redirected from a UNIGRAFIX scene file.

3.1. Ugxform and ugexpand

Both these programs make global transformations on the whole scene file read. **Ugxform** is a fast stream editor which carries out all specified transformations on the top-level commands and passes all other commands unaltered. **Ugexpand** instanciates all instance and array commands and outputs a hierarchically flat description of the scene with unique identifiers for all elements.

Both programs take all of the following options:

-tx	*amount*	
-ty	*amount*	Translate scene by *amount* in the specified direction.
-tz	*amount*	
-rx	*angle*	Rotate scene around specified axis by *angle* (in degrees).
-ry	*angle*	Direction of rotation is described in section 2.1.
-rz	*angle*	
-sx	*factor*	
-sy	*factor*	Scale the scene by *factor* in the appropriate dimension.
-sz	*factor*	
-sa	*factor*	Scale the scene by *factor* in all three dimensions.
-mx		Mirror x-coordinates. (Mirroring about yz-plane).
-my		Mirror y-coordinates. (Mirroring about zx-plane).
-mz		Mirror z-coordinates. (Mirroring about xy-plane).
-ma		Mirror all coordinates. (Mirroring about origin).
-M3	*3x3 matrix*	From one to nine numbers as transformation matrix.
-M4	*4x4 matrix*	From one to sixteen numbers as transformation matrix. Matrix elements specified replace entires in an identity matrix row by row.
-xl		Transform coordinates of light sources as well.
-px		Print (to stderr) the list of specified transformations.
-pm		Print (to stderr) the total transformation matrix.
-pl		Print (to stderr) the list of all light sources.
-fc	*filename*	Use file *filename* to find command-line options.
-fi	*filename*	Use file *filename* as standard input.
-fo	*filename*	Write standard output into file *filename*.

-oi Omit *all* include files and print transformation matrix after include filename. The default is to do this only for the cases where top-level include files do not exist; the programs then also print a warning and continue with the processing of the calling file.

The transformed scene files go to standard output, so it is probably best to redirect it to a file. Output from the **-px** or from the **-pm** options concerns all transformations in effect at the time when one of these options is read. Both of these options output to standard error.

Vertices and illumination sources are the only top level objects that have coordinate values and thus can be transformed explicitly. Vertices will always be transformed; illumination sources will only be transformed if the **-xl** option is given.

Arguments to transformations may be integers or floating-point numbers. Again, care should be taken when scaling by a negative number or mirroring.

3.1.1. Use of ugxform

These examples are intended to show some possible uses of the **ugxform** command and its various options.

ugxform -sx 10 -ry 23 -mz < scene > newscene

This takes the description in file *scene,* scales the x dimension of all vertices by 10, rotates around the y-axis by 23 degrees, mirrors the z-coordinates, and puts the resuling scene into *newscene.*

ugxform -fc commandfile -xl -fi scenefile | ugplot -dv

This uses the options specified in *commandfile* to transform the description in *scenefile.* Because **-xl** is specified, illumination sources are transformed as well. the result is then piped to the rendering program **ugplot.**

ugxform -fc optfile1 -px -fc optfile2 -px -fi infile -fo outfile.

In this example, two transformation option files are used to transform the scene in *infile.* The result is written into *outfile.* The two calls of the **-px** option will print to standard error (presumably the terminal) the current list of transformations at the times when they are encountered.

Instance and array commands are transformed in the same way that given definitions are transformed in instance or array calls. The specified transformations are appended in the specified order to the transformations already present in each instance command.

For arrays there is an additional difficulty. Since there is no field that permits the specification of a global transformation, GT, that works on an array as a whole, the original transformation, OT, of the first instance as well as the incremental transformation, IT, between instances need to be modified according to the following formula:

a array (defname OT GT) size GTinv IT GT ;

If the inverse, GTinv, of the transformation GT does not exist, top-level array commands must be broken into their components, and the specified transformation GT is then appended to each individual instance call.

In addition to the options already listed, **ugxform** also takes the following options:

-cm Coalesce transformations to one matrix on instances and arrays.

-oc Omit comments in output file.

The **-cm** option helps to clean up instance statements and prevents the accumulation of long lists of transformation steps.

3.1.2. Use of ugexpand

Ugexpand expands instances and arrays into vertices, wires and faces but leaves the described scene basically unchanged. It copies all top-level elements to the output and converts all **instance** or **array** statement into their constituent parts. Include files are expanded and placed into the output. This program can also calculate the illumination on each surface, and record it there for subsequent use by a display program.

In addition to the general options already listed, **ugexpand** also takes the following options:

-nl		New labels. The program creates new, short, sequential names for all items.
-me	[*epsilon*]	Merge edges. All edges within *epsilon* of each other are cut to be a single contour. If *epsilon* is not specified, it defaults to 1e-6.
-ae		Attach plane equation to each face statement.
-al		Attach computed light value on each face.
-fw	*x y z d1 d2*	Fade against white background in interval d1-d2.
-fb	*x y z d1 d2*	Fade against black background in interval d1-d2. *x, y* and *z* specify the eye-point; *d1* and *d2* are distances from the eyepoint.

3.2. Ugisect

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.

Ugisect will normally have to create many new vertices. These vertices are given suquential numerical names of the form v#*n*, where *n* is a number. Therefore, UNIGRAFIX files should not contain vertices with names of this form.

Upon termination, *ugisect* will print out some statistics concerning the number of intersecting elements. Eventually **ugisect** will allow some method for specifying the removal of collections of object intersections; in this way a file describing the results of geometric operations such as *union* or *intersection* of various objects can be created.

3.3. Ugshow and ugplot

Ugshow is the original display program of the UNIGRAFIX system. **Ugplot** is a newer program created for the UNIGRAFIX2 system. Both can render a stored scene on various devices. The default is to output to the terminal from where the commands originate. This works even from a "dumb" terminal, but the output will be coarse and difficult to understand.

The difference between the two programs lies primarily in the algorithms used for hidden surface elimination and in the internal data structures. There are a few enhancements in UNIGRAFIX2 for which the older **ugshow** has not yet been upgraded.

With both programs, there are quite a few options available, and they will be discussed in three groups of commands that belong together connceptually. However these options can occur in any order on the command line. The *viewing parameters* are geometrical specifications that affect the direction and manner in which the scene is viewed. The *display modes* determine what aspects of the scene (e.g. edges, faces) will be shown. The rest of the options are those that do not fit into either of the first two groups.

3.3.1. Viewing Geometry

These options specify whether or not perspective is desired, and from what direction and under what angle the scene is to be viewed.

-ep	*x y z*	Eye point location; implies perspective view from this point.
-ed	*x y z*	Eye direction; *x*, *y*, and *z* define a vector (of arbitrary length) from the origin to the viewer's eye. The view of the scene is an orthogonal (non-perspective) projection from this direction. Orthogonal view is a default, and without this option the eye direction will be (0, 0, -1), which means the view is a parallel projection from the negative z-axis.
-vc	*x y z*	View center; specifies the point in the scene that will become the center of the display. Works for either perspective or orthogonal views. Defaults to the origin.
-vr	*angle*	View rotation; specifies what direction will be "up" in the displayed scene. By default, the y-axis points up, but specifying an *angle* (in degrees) causes the displayed scene to be rotated counterclockwise.
-vz	*factor*	Zoom. The default *zoom factor* is one, which fits the picture in the specified display size. The **-vz** option allows respecifying this constant. Zoom factors greater than one will blow the picture up, causing objects expanded off the screen to be clipped, while zoom factors less than one will cause the picture to shrink. The picture's centering on the screen remains constant regardless of the zoom factor, except Varian and Versatec which are positioned to the upper left corner of the page to save paper.
-va	*angle*	View angle. This option is only valid with a perspective view. It defines the maximum angle of a square-based viewing pyramid in which the scene is viewed, anchored at the eye point. Everything outside this viewing pyramid is clipped. The *angle* should be between 0 and 180, exclusively. Default is to find the smallest angle within which the entire scene will fit. However, if the computed angle for the whole scene exceeds 90 degrees, the scene will be clipped to 90 degrees.
-ft	*epsilon*	Face Tolerance. (**Ugplot** only) Change the tolerance for rejecting faces that are close to being back faces. Useful when faces are slightly warped and viewing them edge-on results in a self-intersecting face which will be plotted with varying results. Epsilon defaults to 1e-2 and should be between 0 and 1.

It is an error to specify both perspective and orthogonal views (i.e. use both the **-ep** and **-ed** options) at the same time. Furthermore, for a perspective view, the center of view and the eye point may not be the same point. For an orthogonal view, the eye direction vector may not be of zero length.

The viewing specifications determine the set of transformations applied to the scene in world coordinates; the end result of these transformations is a scene on an output device. First, a translation is applied to the scene which moves the viewcenter to the middle of the viewing space, and thus to the center of the final viewport. The view direction is determined from the eye direction or from the vector from the eye point to the view center. Then the view rotation is taken into consideration. Finally, the view angles or zoom factor are used to determine the extent of the displayed part of the scene, and the scene is mapped to device coordinates.

The viewcenter typically maps to the center of the viewport. If it is not explicitly specified, the origin is used for the case of perspective projections and, in the case of parallel projections, the average of the minimum and maximum coordinates of the picture.

3.3.2. Display Mode Options

These options set the rendering style and specify what features of the scene are to be displayed:

-se Default. Show edges only, i.e. wires and borders of displayed faces.
-sf Show only faces without edges. (Implies -ho).
-sa Show all features (faces and edges). (Implies -ho).

-ab Add backfaces

-hn Default. Hide nothing, make no visibility checks.
-hb Hide back-faces, i.e. faces with face normal pointing away from eye.
-ho Hide ovelaps; remove all features hidden by overlaps.
-hd Hidden lines dashed (currently only for Gremlin files).

-lv Label vertices (currently only for HP, AED, Vectrix, Imagen and dumb terminals).
-lf Label faces (currently only for HP, AED, Vectrix and dumb terminals).
-lw Label wires (currently only for HP, AED, Vectrix and dumb terminals).
-la Label all (currently only for HP, AED, Vectrix and dumb terminals).

-fw x y z d1 d2 Fade against white background in interval d1-d2.
-fb x y z d1 d2 Fade against black background in interval d1-d2.
x, y and *z* specify the eye-point;
d1 and *d2* are distances from the eyepoint.

Default is output in *wire-frame* format. This is usually very fast, since the costly hidden-surface elimination process need not be performed.

If contradictory **-s** options are specified (e.g. **-se** and **-sf, -sa**), the last one overrides any previous ones.

When faces are shown (with **-sf** or **-sa** options), the **-hb** and **-ho** options are implied.

Rendered faces will be shaded (or colored, for some devices) according to that face's ligthness value. Edges appear in their given color (if on a color device), with maximum illumination in all cases.

When labels are specified, the names of labeled objects are output at the appropriate locations. The name that is used for each obeject is the object's identifier, which may be quite long after expansion of instances or arrays. Because of this, it is possible to use only part of the id as a label. If a sharp sign ("#") is included within an object's id, only the part of the id that *follows* the sharp sign will be output as a label. For example, if a vertex in a definition has an id *#vert1,* and that definition is used in *inst234,* the label displayed for the corrsponding vertex will be *vert1,* even though the expanded id is *inst234.#vert1.* This feature also permits identical labels to be printed for several different objects. For example, three vertices with id's *A#top, B#top,* and *C#top* will all be labeled with *top.*

3.3.3. Input / Output Options

Other options available for the programs **ugplot** and **ugshow** are:

-fc	*cmdfile*	Read options from file *cmdfile.*
-fi	*inputfile*	Read input from file *inputfile.*
-wg	*gremlinfile*	Write an output file in Gremlin format (implies -gi -se), (currently only line drawings can be produced); the number and type of edges shown depends on the -h? option chosen.
-dv		Output device is Varian plotter.
-dw		Output device is Versatec plotter.
-dm		Output device is Imagen printer.
-da		Output device is AED 512 color display (set GRTERM).

-dx		Output device is Vectrix color display (set GRTERM).
-dr		Output device is IRIS graphics terminal (set GRTERM).
-dl		Output is for Ikonas frame buffer. A raster file called "rast.iv" will be created. This can be sent to the Ikonas with the **iv** program.
-sx	*number*	Plot is sized so that x-dimension fits in *number* inches. Default is to make the picture as large as the width of the display device.
-sy	*number*	Plot is sized so that y-dimension fits in *number* inches. Default is to make picture as large as the height of the display device. On the Varian and Versatec plotters, the default is 8 and 36 inches, respectively. For these plotters only, the specified y-dimension may be as much as twice the default.
-kf		Keep raster files. Valid only with **-dv** or **-dw**. Useful if you want to run off several copies of something. The raster file is of the form "/usr/tmp/ugXXXXX." The name of the created raster file will be printed to standard error.

Any options to **ugplot**, including viewing parameters and display modes, may be put in a file and then called with the **-fc** option. Input and output files can also be specified with the **-fi** and **-fo** options rather than by redirection.

If no **-d?** option is given, output goes to the terminal from which the program was invoked. For display terminals which are generally used for display only where commands are typed on a different terminal (i.e. our set up for the AED and Vectrix), the shell environment variable 'GRTERM' should be set using the 'setenv' command. The value of this variable should be the output display's device file name, usually something like '/dev/tty??'.

The options **-sx** and **-sy** specify the size of the viewport on a display device or the size of the total frame of a hardcopy output on one of the plotters. The command

ugplot -sx 4 -sy 2 < scene

will produce output that fits into the frame of 4 inches by 2 inches. The scale of the picture will be chosen to meet the tighter constraint. By default this viewport will be centered on the screen of a display device. On hardcopy devices the plot will only be centered in the x direction but will be started immediately with the first scan line in the y direction in order to conserve paper. Exactly what appears in the viewport will depend on the viewing parameters.

3.3.4. Tutorial on Use of the Options

Viewing Parameters

These examples demonstrate how different views of scenes may be obtained by setting the viewing parameters. A default rendering on the user's terminal is obtained with:

ugplot < easy

This displays the scene in *easy* with an orthogonal projection, viewing from along the negative z-axis towards the origin. The entire scene will be scaled evenly to fit the display. A specific view can be specified with:

ugplot -ed 1 1 1 -vc 0 50 0 -vr 90 < special

Again, parallel projection is used, but this time the view direction is along the vector (-1, -1, -1). The entire scene will be displayed. The point (0, 50, 0) is forced to become the center of the displayed scene. Note that this does not change the view direction; it just forces an offset of the displayed scene in the y direction. And finally the y-axis is no longer pointing upwards, but has been turned 90 degrees counter-clockwise.

To display only part of the scene, one may use the zoom option:

ugplot -vc 10 20 0 -vz 4 < close_look

Now the scene is clipped to only include what lies around the specified viewcenter.

An easy way to obtain a perspective view is with the command:

ugplot -ep 0 0 -100 < perspect

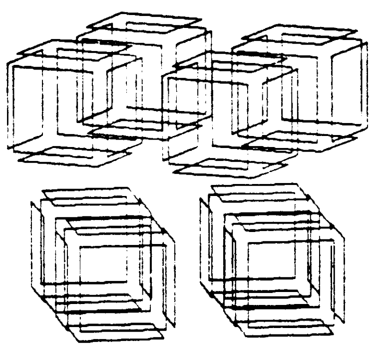
This places the eye at point (0, 0, -100). The viewing pyramid will be scaled until it just surrounds the scene, while the axis of the viewing pyramid is kept going through the origin. If the coordinates in the scene file are much smaller than the distance to the eye point (in this case, 100), then the view approaches an orthogonal projection. If the viewing pyramid exceeds 90 degrees, the scene is clipped. The command:

ugplot -ep 0 0 -100 -vc 100 0 0 -va 5 < narrowview

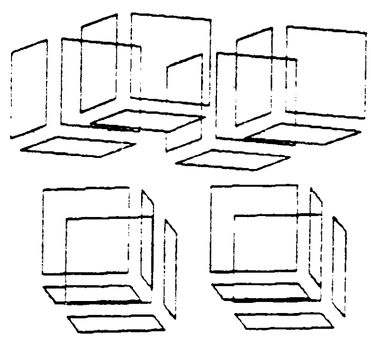
produces a view from the same point, but centered on (100, 0, 0). This is equivalent to looking 45 degrees to the right. In addition, the view angle is explicitly stated to be only 5 degrees. Whatever is visible within a viewing pyramid of 5 degrees will be displayed.

Display Mode Options

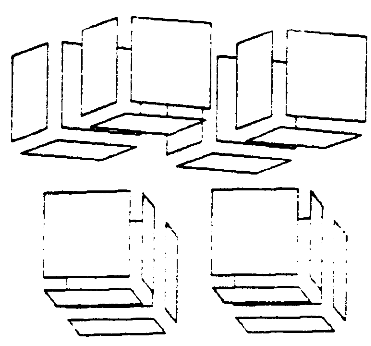
The six figures A through H show the same object with the same view but with different display mode parameters:



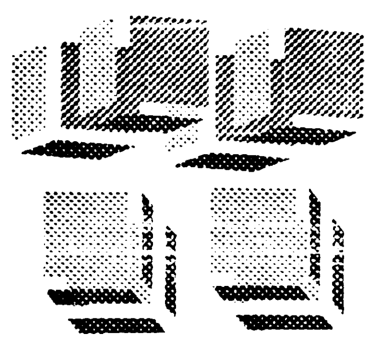
A) ugplot < cubes
Default wire-frame



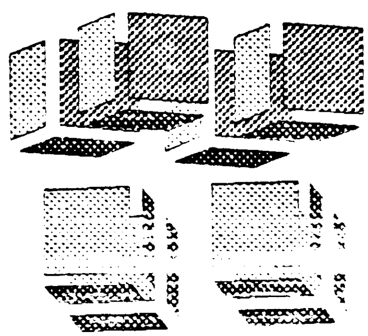
B) ugplot -hb < cubes
Back-faces eliminated



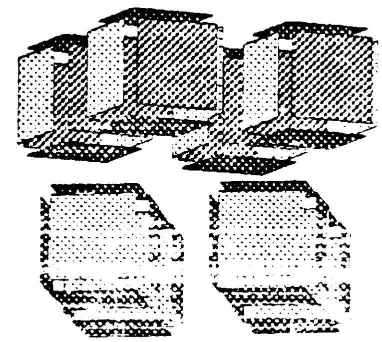
C) ugplot -ho < cubes
Overlapping features removed



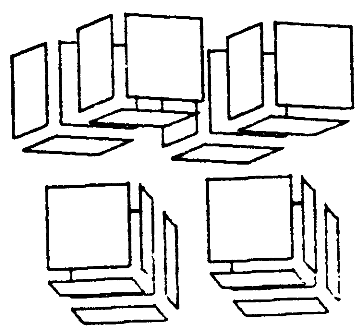
D) ugplot -sf < cubes
Shaded front faces



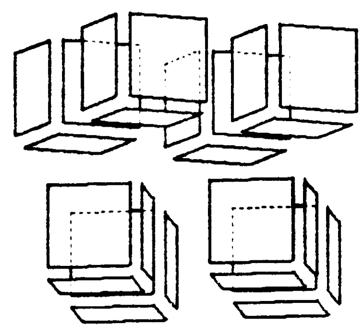
E) ugplot -sa -ho < cubes
Faces & edges, overlaps removed



F) ugplot -sa -ab -ho < cubes
Two-sided faces, overlap removed



G) ugplot -ho -wg gremfile < cubes
Gremlin format



H) ugplot -hd -wg gremfile < cubes
Dotted hidden lines

Input / Output Options

For the case of parallel projection, the 2-D image of the scene will normally be scaled and translated so that it touches two opposite sides in the dimension that is more constraining; it is symmetrically centered in the other dimension. However, if the **-vc** option is used, the projection of view center is forced to appear in the middle of the viewport, and the extent of the 2-D image is checked in all four directions to determine the most stringent constraint and to derive the necessary scale factor.

For the case of a perspective view, the transform that moves the eye into the origin of the viewing coordinate system is carried out first. In this state the clipping is carried out against the viewing cone with a 90 degree opening. If nothing is clipped by this cone, then it is narrowed to the maximum angle under which any one of the vertices is seen from the eye-point. If the view angle was specified, the corresponding scaling in x and y is performed before the clipping operation, and no further scaling is performed. The perspective view has no self centering option as exists for the parallel projection. If the view center is badly chosen, e.g. drastically outside the scene, there will be nothing on the display. Care must also be taken with wrap around: if the eye-point moves too close to the scene, so that the viewing angle exceeds 180 degrees, points from behind the viewers eye may get wrapped around into their counter points on the other side of the origin, which may lead to very strange displays.

4. INTERACTIVE UNIGRAFIX

More often than not, several attempts must be made before a satisfactory display of a scene is achieved. Since each one of these attempts involves running at least one Unigrafix batch program, much processing time will be spent reading in and possibly writing out scene files. The overhead incurred with this approach is often restrictive, especially for large scenes, or ones with multi-level hierarchies.

In order to speed up this iterative modify/display process, an interactive UNIGRAFIX shell has been created. A scene is read into this shell as one or more files, the internally stored scene can then be transformed, illuminated, and displayed repeatedly from different viewpoints. At the end, the scene in its final state can either be written back into a file in a flat format or a script of all the modifications made to the scene can be obtained. With this script and the original starting scene it is then possible to directly recreate the final view.

NOTE: This interactive program has not yet been converted to the UNIGRAFIX2 system. It is still changing, and the section here is more a preview of what we are aiming for. Consult later editions of this manual to get the most up-to-date information.

4.1. Interactive Commands

When you enter the interactive Unigrafix environment with the command

ugi

you will see a prompt that looks like this:

ug>

You may now enter a command. Command names are single words, which may be abbreviated by any amount, down to a single letter. For example, **read, rea, re,** and **r** are all valid forms of the **read** command.

The following sections discuss each of the available commands.

4.1.1. Read

The **read** command is used to read in scenes from files. The syntax of the command is:

read *filename1* *filename2* ...

All of the specified filenames are read in and added to the *current scene*. You should be careful when reading in multiple files to avoid naming conflicts between files. Since everything becomes part of the same scene, an object in one file should not have the same id as an object of the same type in another file. We are contemplating creating an additional implicit level in the hierarchy that corresponds to the individual files read in.

4.1.2. Xform

The **xform** command can be used to transform the current scene. Its syntax is:

xform *options*

The available options and their effects are identical to those of the batch **ugxform** program.

4.1.3. Illum

This command modifies the illumination for the current scene, or illuminates faces in the current scene. Additionally, it can be used to tell you what illumination is present in the scene. The command looks like:

illum *options*

This command was implemented in Unigrafix 1, but because of changes in how illumination is performed, it has yet to be implemented in Unigrafix 2.0. The exact number and form of the options has not yet been defined either.

4.1.4. View

This command has the form

view *options*

It is used to set the *viewing parameters* for the current scene. Once set, the parameters remain in effect for the rest of the session, unless reset by another **view** command. They can also be temporarily overridden by using options on the **plot** or **show** commands, as explained in the next sections.

The options for this command are all those available to the batch ugplot command that affect viewing parameters. These are **-ep, -ed, -vc, -vr, -va, -vz, -fw** and **-fb.** Additionally, if

view -p

is entered, a list of the current viewing parameters will be printed on the terminal.

Viewing parameters not changed by the command remain the same, with one exception. If you switch from a perspective view to an orthogonal view (or vice-versa), all parameters that are no longer valid for the new view are ignored. For example, if the current viewing parameters are

-ep 0 100 -300 -va 85

and you enter

viewparams -ed 1 1 1

the view is no longer a perspective projection, and the view angle specification (-va 85) no longer has any effect.

4.1.5. Display

This command, like **ugplot** or **ugshow,** displays the current scene. The syntax is:

display *options*

Options may include any of those available for the two rendering programs **ugshow** and **ugplot**.

If no viewing parameters are changed on the command line, the ones specified with the most recent **viewparams** command are used. If any are changed, the effect is only for this one display, and the parameters are reset to their previous values when the display is finished.

All other (non-viewing parameter) options are in effect only for one display. If you want to see only shaded faces, for example, you must specify

display -sf

each time.

4.1.6. Write

If you want to save the current scene in a file, you can use the **write** command. Its syntax is:

write *options filename*

The two possible options are **-f** for "flattened" output or **-s** for "symbolic" (default) output. If both are specified in the same command, only the last one will have effect. (To write out the scene both ways, use the command twice).

When a **write** is done, the current scene will be written to the named file. If the **-f** option is specified, the flattened version of the scene will contain current illumination values on all faces, if illumination was calculated, either by an **illum** command, or a **display** with faces visible.

If the **-s** option is used, a file describing the scene will not be output. Instead, a shell script, consisting of UNIGRAFIX commands that produce the current scene from the original, is written out to the named file. This file can then be run by the shell to produce the "saved" scene on standard output, which can then be sent to a rendering program.

4.1.7. Help

The **help** command prints a brief list of available commands.

4.1.8. Run

Another planned extension that makes it possible to run one of the generator programs[1] from within the interactive UNIGRAFIX shell. Thus it will be possible to use simple scenes as a base to create more complicated objects such as worms, trees, and stairs without leaving the **ugi** program.

5. DISPLAY OUTPUT

Although UNIGRAFIX is designed to be as device-independent as possible, there are some device-dependent details that do exist. Because of the varying natures of the different output devices, total uniformity is not possible. Therefore, this section describes how each type of device is treated. Currently, UNIGRAFIX supports these devices:

- Non-graphics ("dumb") terminals
- Hewlett-Packard 2648A Graphics Terminal
- SUN Workstation
- IRIS graphics terminal
- AED 512 Color Graphics/Imaging Terminal
- Vectrix Color Graphics Terminal
- Imagen Laser Printer
- Varian 11-inch plotter
- Versatec 3-foot wide-body plotter
- Ikonas Frame Buffer

The unique properties of display on each type of device are discussed in the remainder of this section. Some of the material is subject to change.

The following is a table of the supported devices and some of their salient characteristics. The defaults referred to are the sizes that UNIGRAFIX will normally fit a picture into; these can be overriden by using the sizing options with **ugplot** or **ugshow** (see Section 3).

Device	Horizontally			Vertically		
	pixels/ inch	maximum pixels	default pixels	lines/ inch	maximum lines	default lines
Dumb	80/ 8	80	80	24/ 5.5	24	24
2648A	720/ 12	720	720	360/ 6	360	360
SUN	1024/ 12	1024	1024	780/ 10	780	780
IRIS	1024/ 14	1024	1024	767/ 12	767	767
AED	512/ 14	512	512	512/ 14	512	512
Vectrix	672/ 10	672	672	480/ 7.5	480	480
Varian	200	2200	2200	200	3400	1600
Versatec	200	7200	7200	200	14400	7200
Imagen	120	1020	1020	120	1320	1320
Ikonas	512/ 12	512	512	512/ 12	512	512

5.1. Non-graphics ("Dumb") Terminals

Display may be sent to a terminal, even if it is not recognized as a graphics terminal, In such a case, the **curses** package [2] is used. In order for **curses** to know the type of terminal you are using, the **TERM** variable should be set in your shell.

Output to a dumb terminal is rather crude. Eight levels of shading are used, simulated by varying-intensity characters (the characeters used are: N X Y < ! : , . and (space)). Edges are displayed with the edge character "*".

5.2. Hewlett-Packard 2648A Graphics Terminal

UNIGRAFIX makes full use of the graphics capabilities of the HP 2648A. Wire-frame display uses the line-drawing facilities. Shaded faces (and edges with hidden surfaces removed) are drawn with lines in the appropriate area fill pattern.

Shading is approximated by using a range of sixty-five cluster-pattern stipples. A lighter face will appear as being whiter on the screen, and vice-versa. Maximum illumination on a white face will appear solid white on the screen. Any face with zero illumination, or a face of pure black, will not be visible unless edges are shown. Edges are always displayed as pure white.

When using an HP 2648A for display, be sure your terminal type is set to "hp2648a" so that UNIGRAFIX can recognize your terminal.

5.3. SUN Workstation

Display on the SUN worksations is much like that on the HP. Shaded faces and/or lines are possible. The stipple patterns used are the same as those for the HP.

5.4. IRIS graphics terminal

The IRIS is capable of displaying objects in color, much like the AED 512. UNIGRAFIX is capable of sending the IRIS visible polygons which the IRIS fills locally; UNIGRAFIX need not output explicit scanlines, resulting is faster display. The color map is set up like that of the AED 512.

5.5. AED 512 Color Graphics/Imaging Terminal

The AED is one of the supported devices capable of color graphics. Therefore, stipple patterns are not used; objects are colored by controlling the intensities of the red, green, and blue guns. Edges are always displayed in their given color (actually, the nearest color available); edges are not shaded. The AED is capable of 256 colors (drawn from a much larger palate); currently these colors are divided up into 16 intensities of 16 fully saturated hues. The user has no choice in how the AED's color palate is selected; hopefully this will change in later versions.

5.6. Vectrix Color Graphics Terminal

Vectrix support is similar to that of the AED. The color scheme is currently the same as that of the AED.

5.7. Varian and Versatec Plotters

Hard-copy output on these devices is achieved through creation of a temporary raster file, which contains the image of the display. This file will be sent to be printed on the plotter, much like any other file.

5.8. Imagen Printer

Output on the Imagen is very similar to that on the Varian and Versatec. However, polygons, lines and labels are sent as primitives to the imagen graphics routines. Stipple patterns are used to simulate shading.

The shading scheme used for the plotters is the same as that used for the HP 2648A. The same stipple patterns (with finer resolution) are used to create shading. Edges are output as solid black lines.

5.9. Ikonas Frame Buffer

Current implementation of display on the Ikonas uses low-resolution (512 X 512 pixels) mode, with 256 available colors. The color map used is the same as that for the AED. The Ikonas is capable of far more colors, so hopefully this will change.

The output for the Ikonas is in the form of a raster file that is suitable for use with the Iv (Ikonas view) program. The raster file that is created is called "rast.iv" and is placed in the current directory.

6. ERRORS AND ERROR-HANDLING

There are four types of errors that may occur when you are using Unigrafix. These are *command-line* errors, *input* errors, *scene* errors, and *system* errors. Each if these types is handled differently, but all result in a message being sent to standard error (presumably the terminal).

6.1. Command-line Errors

Since the Unigrafix programs allow specification of many options on the command line, there is much room for error. Some of these include:

- Invalid option. Either the wrong letter, or a missing hyphen.
- Bad or missing arguments to an option.
- Conflicting arguments (especially for the **ugshow** and **ugplot** programs).

Most of the time these errors will be detected before any other processing occurs, but this is not always the case. For batch programs, execution ends as soon as a command-line error is found. For the interactive version, you will be prompted for the next command.

6.2. Input Errors

Input errors may occur when scene files are read in. They result from problems with the Unigrafix language statements that specify the scene. Some common input errors are:

- Statements starting with an unknown keyword.
- Unexpected items in a statement, often because of a wrong number of arguments, or a missing semicolon.
- Badly-formed numbers or transformation specifications.
- Unterminated comments.
- Garbage characters in the scene file.
- Duplicate identifiers for the same type of object.
- Referencing an as yet undefined vertex id or definition name.
- Less than three vertices in the first contour of a face, or less than two in a wire segment.
- Improper statements within definitions.
- Recursive definitions.

- Unterminated definition in a scene file.

Input errors do not cause immediate termination of execution. If an error occurs in the middle of a Unigrafix language statement, the statement is flushed up to the next semicolon. Read-in continues, checking the remainder of the scene file is checked for more errors.

If any errors have been found, the program (whether batch or interactive) is terminated, after the whole scene file has been read.

6.3. Scene Errors

Scene errors are all user errors discovered during the further processing of the read-in file. Some of these are:

- Perspective view so far away that view angle is zero (due to rounding effects).

Scene errors always result in immediate termination of the program.

6.4. System Errors

System errors are problems originating from the Unigrafix programs. The various algorithms are more or less sensitive to inconsistent information about the scene. Such inconsistencies, e.g., non-planar faces, may appear as the result of arithmetic inaccuracies.[3]

References

1. C.H. Séquin, "Creative Geometric Modeling with UNIGRAFIX," Tech. Report (UCB/CSD 83/162), U.C. Berkeley (Dec. 1983).
2. K. Arnold, "Screen Updating and Cursor Movement Optimization: A Library Package," CS Tech. Report, U.C. Berkeley (198?).
3. C.H. Séquin and P.R. Wensley, "Fast Visible Polygon Return from an Extended Cross Algorithm," *Submitted to SIGGRAPH'84*, (Jan. 1984).

Appendix A - Manual Pages

This section contains the most up-to-date manual pages for the programs discussed in the main part of this report.

NAME

ugxform – make fast global transformations of a scene

SYNOPSIS

ugxform [arguments, options] < oldscene > newscene

DESCRIPTION

Ugxform transforms the read scene as a whole according to the transformation arguments given. Only top-level statemants are transformed; all other statements are copied to the output. Valid arguments and options are:

–tx, -ty, -tz *amount*	Translate scene by *amount* in the specified direction.
–rx, -ry, -rz *angle*	Rotate scene around specified axis by *angle*. (degrees CCW when viewed in direction of positive axis).
–sx, -sy, -sz *factor*	Scale the scene by *factor* in the appropriate dimension.
–sa *factor*	Scale the scene by *factor* in all three dimensions.
–mx, -my, -mz	Mirror specified coordinates.
–ma	Mirror all coordinates about origin.
–M3 *3x3 matrix*	Use *one to nine numbers* as transformation matrix.
–M4 *4x4 matrix*	Use *one to sixteen numbers* as transformation matrix.
–cm	Coalesce transformations into single matrix on **l**, **a**.
–xl	Transform coordinates of light sources as well.
–px	Print (to stderr) the list of specified transformations.
–pm	Print (to stderr) the total transformation matrix.
–pl	Print (to stderr) the list of all light sources.
–fc *filename*	Use file *filename* to find command-line options.
–fi *filename*	Use file *filename* as input file.
–fo *filename*	Write output into file *filename*.
–oi	Omit *all* include files and print transformation matrix after include filename. The default is to do this only for the cases where top-level include file do not exist; the programs then also print a warning and continue with the processing of the calling file.
–oc	Omit comments.

EXAMPLE

ugxform -sa 3 -tx -10 -oc < inputfile > shifted_big_scene

FILES

~ug/bin/ugxform
~ug/src/ugc

SEE ALSO

ugexpand (UG), ugplot (UG)

BUGS

Does not handle arrays correctly

AUTHOR

Looking for a volunteer to fix it

NAME

ugexpand – flatten the hierarchy of a scene description

SYNOPSIS

ugexpand [arguments, options] < oldscene > newscene

DESCRIPTION

Ugexpand expands all instances and array statements so that the resulting scene description has only top-level statements and no more comments. A global transformation can be carried out at the same time as with *ugxform*. The following arguments can be used:

–tx, -ty, -tz *amount*	Translate scene by *amount* in the specified direction.
–rx, -ry, -rz *angle*	Rotate scene around specified axis by *angle*. (degrees CCW when viewed in direction of pos. axis).
–sx, -sy, -sz *factor*	Scale the scene by *factor* in the appropriate dimension.
–sa *factor*	Scale the scene by *factor* in all three dimensions.
–mx, -my, -mz	Mirror specifies coordinates.
–ma	Mirror all coordinates about origin.
–M3 *3x3 matrix*	Use *one to nine numbers* as transformation matrix.
–M4 *4x4 matrix*	Use *one to sixteen numbers* as transformation matrix.
–xl	Transform coordinates of light sources as well.
–px	Print (to stderr) the list of specified transformations.
–pm	Print (to stderr) the total transformation matrix.
–pl	Print (to stderr) the list of all light sources.
–nl	New labels. The program creates new, short, sequential names for all items.
–mv	Merge vertices. All coinciding vertices are combined into a single one which is given the name of the first one encountered at that position.
–ae	Attach plane equation to each face statement.
–al	Attach computed light value on each face.
–fw x y z d1 d2	Fade against white background in interval d1-d2.
–fb x y z d1 d2	Fade against black background in interval d1-d2. *x, y* and *z* specify the eye-point; *d1* and *d2* are distances from the eyepoint.
–fc *filename*	Use file *filename* to find command-line options.
–fi *filename*	Use file *filename* as input file.
–fo *filename*	Write output into file *filename*.
–ol	Omit *all* include files and print transformation matrix after include filename. The default is to do this only for the cases where top-level include file do not exist; the programs then also print a warning and continue with the processing of the calling file.

EXAMPLE

ugexpand -ma -ae < inputfile > mirrored_scene_with_planeeqn

FILES

~ug/bin/ugexpand
~ug/src/ugc

SEE ALSO

ugxform (UG), ugplot (UG)

DIAGNOSICS

Checks input file for syntax errors and duplicate names.

BUGS

Not all options are implemented yet.

AUTHOR

Looking for a volunteer to fix it

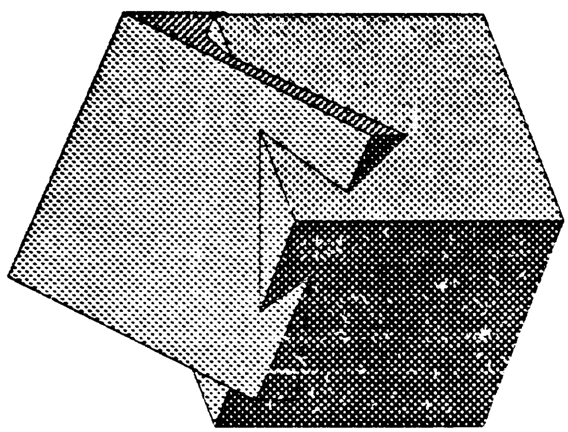
NAME
ugisect – convert intersecting faces and wires into non-intersecting objects

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



FILES
~ug/bin/ugisect
~ug/src/ugc

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

ugshow - rendering of a scene on screen or plotter

SYNOPSIS

ugshow [arguments, options] < scene

DESCRIPTION

Ugshow produces a rendering of a scene on many possible output devices. Viewing geometry and display style are specified with the following arguments:

-ep	*x y z*	Eye point for perspective view from this point.
-ed	*x y z*	Eye direction for parallel projection. Default is ed=(0, 0, -1), i.e., an orthogonal projection from the negative z-axis.
-vc	*x y z*	View center; i.e., the point in the scene that will become the center of the display. Defaults to the origin.
-vr	*angle*	View rotation. By default the y-axis points up; displayed scene is rotated CCW by *angle* degrees.
-vz	*factor*	Zoom. The default *zoom factor* is one which fits the picture in the specified display size. The **-vz** option allows respecifying this constant. Zoom factors greater than one will blow the picture up, causing objects expanded off the screen to be clipped, while zoom factors less than one will cause the picture to shrink. The picture's centering on the screen remains constant regardless of the zoom factor.
-va	*angle*	View angle for a perspective view; must be between 0 and 180, exclusively. It defines the maximum angle of a square-based viewing pyramid, anchored at the eye point. By default the scene will be clipped to 90 degrees.

-se	Default. Show edges and wires only.
-sf	Show only faces without edges. (Implies -hb).
-sa	Show all faces and edges. (Implies -hb).
-ab	Add backfaces
-hn	Default. Hide nothing, make no visibility checks.
-hb	Hide back-faces, i.e. faces with face normal pointing away from eye.
-ho	Hide ovelaps; remove back-faces and all features hidden by overlap.
-hd	Hidden lines dashed (currently only for Gremlin files).
-lv	Label vertices.

-fc	*cmdfile*	Read options from file *cmdfile*.
-fi	*inputfile*	Read input from file *inputfile*.
-wg	*gremlinfile*	Write an output file in Gremlin format (implies -gi -se), (currently only line drawings can be produced); the number and type of edges shown depends on the -h? option chosen.
-dv		Output device is Varian plotter.
-dw		Output device is Versatec plotter.
-dm		Output device is Imagen printer.
-da		Output device is AED 512 color display (set GRTERM).
-dx		Output device is Vectrix color display (set GRTERM).
-dr		Output device is IRIS graphics terminal (set GRTERM).
-di		Output is for Ikonas frame buffer. A raster file called "rast.iv" will be created. This can be sent to the Ikonas with the **iv** program.

-sx	*number*	x-size of plot is adjusted to fit into *number* inches. Default: width of the display device.
-sy	*number*	y-size of plot is adjusted to fit into *number* inches. Default: height of the display device. On the Varian and Versatec plotters, the default is 8 and 36 inches, respectively. Specified y-size can be up to twice the default.
-kf		Keep raster file. Valid only with **-dv** or **-dw**. Useful if you want to run off several copies of something. The raster file is of the form "/usr/tmp/ugXXXXX." The name of the created raster file will be printed to standard error.

EXAMPLE

ugshow -sa -ep -1 2 -10 -va 30 -dv -sy 4 < scenefile

FILES

~ug/bin/ugshow
~ug/src/UG1

SEE ALSO

ugexpand (UG), **ugisect** (UG), **ugplot** (UG)

BUGS

Trouble with horizontal border lines.
Sometimes wrongly orderes faces at points such as concave corners.

AUTHOR

Looking for a volunteer to take charge of it.

NAME

ugplot – rendering of a scene on a screen or plotter

SYNOPSIS

ugplot [arguments, options] < scene

DESCRIPTION

Ugplot can render a scene on many possible output devices. The viewing geometry and the display style are specified with the following arguments:

-ep	*x y z*	Eye point for perspective view from this point.
-ed	*x y z*	Eye direction for parallel projection. Default is ed=(0, 0, -1), i.e., an orthogonal projection from the negative z-axis.
-vc	*x y z*	View center; i.e., the point in the scene that will become the center of the display. Defaults to the origin.
-vr	*angle*	View rotation. By default the y-axis points up; displayed scene is rotated CCW by *angle* degrees.
-vz	*factor*	Zoom. The default *zoom factor* is one which fits the picture in the specified display size. The **-vz** option allows respecifying this constant. Zoom factors greater than one will blow the picture up, causing objects expanded off the screen to be clipped, while zoom factors less than one will cause the picture to shrink. The picture's centering on the screen remains constant regardless of the zoom factor.
-va	*angle*	View angle for a perspective view; must be between 0 and 180, exclusively. It defines the maximum angle of a square-based viewing pyramid, anchored at the eye point. By default the scene will be clipped to 90 degrees.
-ft	*epsilon*	Face Tolerance. Change the tolerance for rejecting faces that are close to being back faces. Useful when faces are slightly warped and viewing them edge-on results in a self-intersecting face which will be plotted with varying results. Epsilon defaults to 1e-2 and should be between 0 and 1.

-se	Default. Show edges and wires only.
-sf	Show only faces without edges. (Implies -hb).
-sa	Show all faces and edges. (Implies -hb).
-ab	Add backfaces
-hn	Default. Hide nothing, make no visibility checks.
-hb	Hide back-faces, i.e. faces with face normal pointing away from eye.
-ho	Hide ovelaps; remove back-faces and all features hidden by overlap.
-hd	Hidden lines dashed (currently only for Gremlin files).
-lv	Label vertices.
-lf	Label faces.
-lw	Label wires.
-la	Label all.

-fc	*cmdfile*	Read options from file *cmdfile*.
-fi	*inputfile*	Read input from file *inputfile*.
-wg	*gremlinfile*	Write an output file in Gremlin format (implies -gi -se), (currently only line drawings can be produced); the number and type of edges shown depends on the -h? option chosen.
-dv		Output device is Varian plotter.
-dw		Output device is Versatec plotter.
-dm		Output device is Imagen printer.

-da		Output device is AED 512 color display (set GRTERM).
-dx		Output device is Vectrix color display (set GRTERM).
-dr		Output device is IRIS graphics terminal (set GRTERM).
-dl		Output is for Ikonas frame buffer. A raster file called "rast.iv" will be created. This can be sent to the Ikonas with the **iv** program.
-sx	*number*	x-size of plot is adjusted to fit into *number* inches. Default: width of the display device.
-sy	*number*	y-size of plot is adjusted to fit into *number* inches. Default: height of the display device. On the Varian and Versatec plotters, the default is 8 and 36 inches, respectively. Specified y-size can be up to twice the default.
-kf		Keep raster files. Valid only with **-dv** or **-dw**. Useful if you want to run off several copies of something. The raster file is of the form "/usr/tmp/ugXXXXXX." The name of the created raster file will be printed to standard error.

EXAMPLE

Ugplot -sa -ep -1 2 -10 -va 30 -dv -sy 4 < scenefile

FILES

~ug/bin/ugplot
~ug/src/ugc

SEE ALSO

ugexpand (UG), ugisect (UG), ugshow (UG)

BUGS

Sometimes faces at points such as pyramid tops get wrongly sorted.
Gets confused by non-planar faces and by accidental coincidences of vertices in 3D space.

AUTHOR

Paul Wensley

Appendix B - Unigrafix Language Summary

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 basic statement types are:

vertices:	**v**	*ID x y z* **;**
color:	**c**	*colorID intensity* [*hue* [*saturation* [*translucency*]]] **;**
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* [*colorID*] [*transformations*] **)** **;**
arrays:	**a**	[*ID*] **(** *defID* [*colorID*] [*transforms*] **)** *size* [*transforms*] **;**
lights:	**l**	[*ID*] *intensity* [*x y z* [*h*]] **;**
include files:	**include**	*filename* [*transformations*] **;**
comments:	**{**	[*anything {nesting is OK} but unmatched* { *or* }] **}**

Appendix C - Formal Syntax Definition

This is an explanatory definition of the Unigrafix language. It is not an exact description of what the language parser will and will not accept.

ugFile = { command }
command = primCommand semi | definitionCommand semi
primCommand = vertexCommand | wireCommand | faceCommand | lightCommand | colorCommand | instanceCommand | arrayCommand | includeCommand | comment | emptyCommand
vertexCommand = **v** id 3dVector
wireCommand = **w** [id] (id id {id}) { (id id {id}) } [colorId]
faceCommand = **f** [id] (id id id {id}) { (id {id}) } [colorId] [illum] [< homogVector >]
lightCommand = **l** [id] intensity [3dVector [number]]
colorCommand = **c** id number [number [number [number]]]
definitionCommand = defStartCommand semi {primCommand} defEndCommand
defStartCommand = **def** defname
defEndCommand = **end**
instanceCommand = **i** [id] (defname [transformations])
arrayCommand = **a** [id] (defname [transformations]) integer [transformations]
includeCommand = **include** filename
comment = { {commentChar} [comment] {commentChar} }
emptyCommand =
semi = **;**
id = stringvar
ColorId = stringvar
defname = stringvar
stringvar = *letter* | *sharpsign* { *letter* | *digit* | *underscore* | *sharpsign* | *period* | *colon* }
illum = number
homogVector = 3dVector number
3dVector = number number number
transformations = **-sx** number | **-sy** number | **-sz** number | **-sa** number | **-rx** number | **-ry** number | **-rz** number | **-tx** number | **-ty** number | **-tz** number | **-mx** | **-my** | **-mz** | **-ma** | **-M3** 1 to 9 numbers | **-M4** 1 to 16 numbers
number = *integer* | *real*
commentChar = *any character* except { and }