B1. Object Files (.obj)

Object files define the geometry and other properties for objects in Wavefront's Advanced Visualizer. Object files can also be used to transfer geometric data back and forth between the Advanced Visualizer and other applications.

Object files can be in ASCII format (.obj) or binary format (.mod). This appendix describes the ASCII format for object files. These files must have the extension *.obj*.

In this release, the .obj file format supports both *polygonal* objects and *free-form* objects. Polygonal geometry uses points, lines, and faces to define objects while free-form geometry uses curves and surfaces.

About this section

The .obj appendix is for those who want to use the .obj format to translate geometric data from other software applications to Wavefront products. It also provides information for Advanced Visualizer users who want detailed information on the Wavefront .obj file format.

If you are a 2.11 user and want to understand the significance of the 3.0 release and how it affects your existing files, you may be especially interested in the section called “Superseded statements” at the end of the appendix. The section, “Patches and free-form surfaces,” gives examples of how 2.11 patches look in 3.0.

How this section is organized

Most of this appendix describes the different parts of an .obj file and how those parts are arranged in the file. The three sections at the end of the appendix provide background information on the 3.0 release of the .obj format.

The .obj appendix includes the following sections:

- File structure
- General statement
- Vertex data
- Specifying free-form curves/surfaces
- Free-form curve/surface attributes
- Elements
- Free-form curve/surface body statements
- Connectivity between free-form surfaces
- Grouping
- Display/render attributes
- Comments
- Mathematics for free-form curves/surfaces
- Superseded statements
- Patches and free-form surfaces

The curve and surface extensions to the .obj file format were developed in conjunction with mental images GmbH&Co.KG, Berlin, Germany, as part of a joint development project to incorporate free-form surfaces into Wavefront’s Advanced Visualizer.

File structure

The following types of data may be included in an .obj file. In this list, the keyword (in parentheses) follows the data type.

Vertex data

- geometric vertices (*v*)
- texture vertices (*vt*)
- vertex normals (*vn*)
- parameter space vertices (*vp*)

Free-form curve/surface attributes

- rational or non-rational forms of curve or surface type: basis matrix, Bezier, B-spline, Cardinal, Taylor (*cstype*)
- degree (*deg*)
- basis matrix (*bmat*)
- step size (*step*)

Elements

- point (*p*)
- line (*l*)
- face (*f*)
- curve (*curv*)
- 2D curve (*curv*2)
- surface (*surf*)

Free-form curve/surface body statements

- parameter values (*parm*)
- outer trimming loop (*trim*)
- inner trimming loop (*hole*)
- special curve (*scrv*)
- special point (*sp*)
- end statement (*end*)

Connectivity between free-form surfaces

- connect (*con*)

Grouping

- group name (*g*)
- smoothing group (*s*)
- merging group (*mg*)
- object name (*o*)

Display/render attributes

- bevel interpolation (*bevel*)
- color interpolation (*c_interp*)
- dissolve interpolation (*d_interp*)
- level of detail (*lod*)
- material name (*usemtl*)
- material library (*mtllib*)
- shadow casting (*shadow_obj*)
- ray tracing (*trace_obj*)
- curve approximation technique (*ctech*)
- surface approximation technique (*stech*)

The following diagram shows how these parts fit together in a typical .obj file.

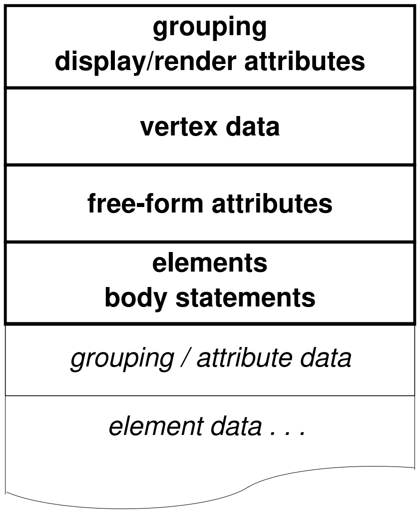


Figure B1-1. Typical .obj file structure

General statement

call ***filename.ext*** *arg1 arg2 . . .*

Reads the contents of the specified .obj or .mod file at this location. The call statement can be inserted into .obj files using a text editor.

filename.ext is the name of the .obj or .mod file to be read. You must include the extension with the filename.

arg1 arg2 . . . specifies a series of optional integer arguments that are passed to the called file. There is no limit to the number of nested calls that can be made.

Arguments passed to the called file are substituted in the same way as in UNIX scripts; for example, *$1* in the called file is replaced by *arg1*, *$2* in the called file is replaced by *arg2*, and so on.

If the frame number is needed in the called file for variable substitution, "$1" must be used as the first argument in the call statement. For example:

```
call filename.obj $1
```

Then the statement in the called file,

```
scmp filename.pv $1
```

will work as expected. For more information on the scmp statement, see appendix C, Variable Substitution for more information.

Another method to do the same thing is:

```
scmp filename.pv $1
call filename.obj
```

Using this method, the *scmp* statement provides the .pv file for all subsequently called .obj or .mod files.

csh ***command***
csh -*command*

Executes the requested UNIX command. If the UNIX command returns an error, the parser flags an error during parsing.

If a dash (-) precedes the UNIX command, the error is ignored.

command is the UNIX command.

Vertex data

Vertex data provides coordinates for:

- geometric vertices
- texture vertices
- vertex normals

For free-form objects, the vertex data also provides:

- parameter space vertices

The vertex data is represented by four vertex lists; one for each type of vertex coordinate. A right-hand coordinate system is used to specify the coordinate locations.

The following sample is a portion of an .obj file that contains the four types of vertex information.

```
v    -5.000000  5.000000  0.000000
v    -5.000000 -5.000000  0.000000
v     5.000000 -5.000000  0.000000
v     5.000000  5.000000  0.000000
vt   -5.000000  5.000000  0.000000
vt   -5.000000 -5.000000  0.000000
vt    5.000000 -5.000000  0.000000
vt    5.000000  5.000000  0.000000
vn    0.000000  0.000000  1.000000
vn    0.000000  0.000000  1.000000
vn    0.000000  0.000000  1.000000
vn    0.000000  0.000000  1.000000
vp    0.210000  3.590000
vp    0.000000  0.000000
vp    1.000000  0.000000
vp    0.500000  0.500000
```

When vertices are loaded into the Advanced Visualizer, they are sequentially numbered, starting with 1. These reference numbers are used in element statements.

Syntax

The following syntax statements are listed in order of complexity.

v *x y z* w

Polygonal and free-form geometry statement.
Specifies a geometric vertex and its x y z coordinates. Rational curves and surfaces require a fourth homogeneous coordinate, also called the *weight*.

x y z are the x, y, and z coordinates for the vertex. These are floating point numbers that define the position of the vertex in three dimensions.

w is the weight required for rational curves and surfaces. It is not required for non-rational curves and surfaces. If you do not specify a value for *w*, the default is 1.0.

NOTE: A positive weight value is recommended. Using zero or negative values may result in an undefined point in a curve or surface.

vp *u* *v w*

Free-form geometry statement.
Specifies a point in the parameter space of a curve or surface.

The usage determines how many coordinates are required. Special points for curves require a 1D control point (u only) in the parameter space of the curve. Special points for surfaces require a 2D point (u and v) in the parameter space of the surface. Control points for non-rational trimming curves require u and v coordinates. Control points for rational trimming curves require u, v, and w (weight) coordinates.

u is the point in the parameter space of a curve or the first coordinate in the parameter space of a surface.

v is the second coordinate in the parameter space of a surface.

w is the weight required for rational trimming curves. If you do not specify a value for *w*, it defaults to 1.0.

NOTE: For additional information on parameter vertices, see the *curv2* and *sp* statements

vn *i j k*

Polygonal and free-form geometry statement.
Specifies a normal vector with components i, j, and k.

Vertex normals affect the smooth-shading and rendering of geometry. For polygons, vertex normals are used in place of the actual facet normals. For surfaces, vertex normals are interpolated over the entire surface and replace the actual analytic surface normal.

When vertex normals are present, they supersede smoothing groups.

i j k are the i, j, and k coordinates for the vertex normal. They are floating point numbers.

vt *u* *v w*

Vertex statement for both polygonal and free-form geometry. Specifies a texture vertex and its coordinates. A 1D texture requires only *u* texture coordinates, a 2D texture requires both *u* and *v* texture coordinates, and a 3D texture requires all three coordinates.

u is the value for the horizontal direction of the texture.

v is an optional argument.
v is the value for the vertical direction of the texture. The default is 0.

w is an optional argument.
w is a value for the depth of the texture. The default is 0.

Specifying free-form curves/surfaces

There are three steps involved in specifying a free-form curve or surface element.

- Specify the type of curve or surface (basis matrix, Bezier, B-spline, Cardinal, or Taylor) using free-form curve/surface attributes.
- Describe the curve or surface with element statements.
- Supply additional information, using free-form curve/surface body statements

The next three sections of this appendix provide detailed information on each of these steps.

Data requirements for curves and surfaces

All curves and surfaces require a certain set of data. This consists of the following:

Free-form curve/surface attributes

- All curves and surfaces require type data, which is given with the *cstype* statement.
- All curves and surfaces require degree data, which is given with the *deg* statement.
- Basis matrix curves or surfaces require a *bmat* statement.
- Basis matrix curves or surfaces also require a step size, which is given with the *step* statement.

Elements

- All curves and surfaces require control points, which are referenced in the c*urv*, c*urv2*, or s*urf* statements.
- 3D curves and surfaces require a parameter range, which is given in the c*urv* and s*urf* statements, respectively.

Free-form curve/surface body statements

- All curves and surfaces require a set of global parameters or a knot vector, both of which are given with the *parm* statement.
- All curves and surfaces body statements require an explicit *end* statement.

Error checks

The above set of data starts out empty with no default values when reading of an .obj file begins. While the file is being read, statements are encountered, information is accumulated, and some errors may be reported.

When the end statement is encountered, the following error checks, which involve consistency between various statements, are performed:

- All required information is present.
- The number of control points, number of parameter values (knots), and degree are consistent with the curve or surface type. If the type is *bmatrix*, the step size is also consistent. (For more information, refer to the parameter vector equations in the section, "Mathematics of free-form curves/surfaces" at the end of appendix B1.)
- If the type is *bmatrix* and the degree is n, the size of the basis matrix is $(n + 1) \times (n + 1)$.

Note that any information given by the state-setting statements remains in effect from one curve or surface to the next. Information given within a curve or surface body is only effective for the curve or surface it is given with.

Free-form curve/surface attributes

Five types of free-form geometry are available in the .obj file format:

- Bezier
- basis matrix
- B-spline
- Cardinal
- Taylor

You can apply these types only to curves and surfaces. Each of these five types can be rational or non-rational.

In addition to specifying the type, you must define the degree for the curve or surface. For basis matrix curve and surface elements, you must also specify the basis matrix and step size.

All free-form curve and surface attribute statements are state-setting. This means that once an attribute statement is set, it applies to all elements that follow until it is reset to a different value.

Syntax

The following syntax statements are listed in order of use.

cstype rat ***type***

Free-form geometry statement.
Specifies the type of curve or surface and indicates a rational or non-rational form.

rat is an optional argument.
rat specifies a rational form for the curve or surface type. If *rat* is not included, the curve or surface is non-rational

type specifies the curve or surface type. Allowed types are:

bmatrix	basis matrix
bezier	Bezier
bspline	B-spline
cardinal	Cardinal
taylor	Taylor

There is no default. A value must be supplied.

deg *degu* *degv*

Free-form geometry statement.
Sets the polynomial degree for curves and surfaces.

degu is the degree in the u direction. It is required for both curves and surfaces.

degv is the degree in the v direction. It is required only for surfaces. For Bezier, B-spline, Taylor, and basis matrix, there is no default; a value must be supplied. For Cardinal, the degree is always 3. If some other value is given for Cardinal, it will be ignored.

bmat *u matrix*
bmat *v matrix*

Free-form geometry statement.
Sets the basis matrices used for basis matrix curves and surfaces. The u and v values must be specified in separate *bmat* statements.

NOTE: The *deg* statement must be given before the *bmat* statements and the size of the matrix must be appropriate for the degree.

---

u specifies that the basis matrix is applied in the u direction.

v specifies that the basis matrix is applied in the v direction.

matrix lists the contents of the basis matrix with column subscript *j* varying the fastest. If *n* is the degree in the given u or v direction, the matrix (*i,j*) should be of size *(n + 1)* x *(n + 1)*.

There is no default. A value must be supplied.

NOTE: The arrangement of the matrix is different from that commonly found in other references. For more information, see the examples at the end of this section and also the section, "Mathematics for free-form curves and surfaces."

---

step *stepu* *stepv*

Free-form geometry statement.
Sets the step size for curves and surfaces that use a basis matrix.

stepu is the step size in the *u* direction. It is required for both curves and surfaces that use a basis matrix.

stepv is the step size in the *v* direction. It is required only for surfaces that use a basis matrix. There is no default. A value must be supplied.

When a curve or surface is being evaluated and a transition from one segment or patch to the next occurs, the set of control points used is incremented by the step size. The appropriate step size depends on the representation type, which is expressed through the basis matrix, and on the degree.

That is, suppose we are given a curve with k control points:

$$\{\mathbf{v}_1, \ldots, \mathbf{v}_k\}$$

If the curve is of degree n, then $n + 1$ control points are needed for each polynomial segment. If the step size is given as s, then the ith polynomial segment, where $i = 0$ is the first segment, will use the control points:

$$\{\mathbf{v}_{is+1}, \ldots, \mathbf{v}_{is+n+1}\}$$

For example, for Bezier curves, $s = n$.

For surfaces, the above description applies independently to each parametric direction.

When you create a file which uses the basis matrix type, be sure to specify a step size appropriate for the current curve or surface representation.

Examples

1. Cubic Bezier surface made with a basis matrix

To create a cubic Bezier surface:

```
cstype bmatrix
deg 3 3
step 3 3
bmat u  1 -3  3 -1 \
        0  3 -6  3 \
        0  0  3 -3 \
        0  0  0  1
bmat v  1 -3  3 -1 \
        0  3 -6  3 \
        0  0  3 -3 \
        0  0  0  1
```

2. Hermite curve made with a basis matrix

To create a Hermite curve:

```
cstype bmatrix
deg 3
step 2
bmat u  1  0 -3  2  0  0  3 -2  \
        0  1 -2  1  0  0 -1  1
```

3. Bezier in u direction with B-spline in v direction; made with a basis matrix

To create a surface with a cubic Bezier in the u direction and cubic uniform B-spline in the v direction:

```
cstype bmatrix
deg 3 3
step 3 1
bmat u  1 -3  3 -1 \
        0  3 -6  3 \
        0  0  3 -3 \
        0  0  0  1
bmat v  0.16666 -0.50000  0.50000 -0.16666 \
        0.66666  0.00000 -1.00000  0.50000 \
        0.16666  0.50000  0.50000 -0.50000 \
        0.00000  0.00000  0.00000  0.16666
```

Elements

For polygonal geometry, the element types available in the .obj file are:

- points
- lines
- faces

For free-form geometry, the element types available in the .obj file are:

- curve
- 2D curve on a surface
- surface

All elements can be freely intermixed in the file.

Referencing vertex data

For all elements, reference numbers are used to identify geometric vertices, texture vertices, vertex normals, and parameter space vertices.

Each of these types of vertices is numbered separately, starting with 1. This means that the first geometric vertex in the file is 1, the second is 2, and so on. The first texture vertex in the file is 1, the second is 2, and so on. The numbering continues sequentially throughout the entire file. Frequently, files have multiple lists of vertex data. This numbering sequence continues even when vertex data is separated by other data.

In addition to counting vertices down from the top of the first list in the file, you can also count vertices back up the list from an element's position in the file. When you count up the list from an element, the reference numbers are negative. A reference number of -1 indicates the vertex immediately above the element. A reference number of -2 indicates two references above and so on.

Referencing groups of vertices

Some elements, such as faces and surfaces, may have a triplet of numbers that reference vertex data.These numbers are the reference numbers for a geometric vertex, a texture vertex, and a vertex normal.

Each triplet of numbers specifies a geometric vertex, texture vertex, and vertex normal. The reference numbers must be in order and must separated by slashes (/).

- The first reference number is the geometric vertex.
- The second reference number is the texture vertex. It follows the first slash.
- The third reference number is the vertex normal. It follows the second slash.

There is no space between numbers and the slashes. There may be more than one series of geometric vertex/texture vertex/vertex normal numbers on a line.

The following is a portion of a sample file for a four-sided face element:

```
f 1/1/1 2/2/2 3/3/3 4/4/4
```

Using v, vt, and vn to represent geometric vertices, texture vertices, and vertex normals, the statement would read:

```
f v/vt/vn v/vt/vn v/vt/vn v/vt/vn
```

If there are only vertices and vertex normals for a face element (no texture vertices), you would enter two slashes (//). For example, to specify only the vertex and vertex normal reference numbers, you would enter:

```
f 1//1 2//2 3//3 4//4
```

When you are using a series of triplets, you must be consistent in the way you reference the vertex data. For example, it is illegal to give vertex normals for some vertices, but not all.

The following is an example of an **illegal statement**.

```
f 1/1/1 2/2/2 3//3 4//4
```

Syntax

The following syntax statements are listed in order of complexity of geometry.

p ***v1*** *v2 v3 . . .*

Polygonal geometry statement.
Specifies a point element and its vertex. You can specify multiple points with this statement. Although points cannot be shaded or rendered, they are used by other Advanced Visualizer programs.

v is the vertex reference number for a point element. Each point element requires one vertex. Positive values indicate absolute vertex numbers. Negative values indicate relative vertex numbers.

l ***v1****/vt1* ***v2****/vt2* ***v3****/vt3 . . .*

Polygonal geometry statement.
Specifies a line and its vertex reference numbers. You can optionally include the texture vertex reference numbers. Although lines cannot be shaded or rendered, they are used by other Advanced Visualizer programs.

The reference numbers for the vertices and texture vertices must be separated by a slash (/). There is no space between the number and the slash.

v is a reference number for a vertex on the line. A minimum of two vertex numbers are required. There is no limit on the maximum. Positive values indicate absolute vertex numbers. Negative values indicate relative vertex numbers.

vt is an optional argument.
vt is the reference number for a texture vertex in the line element. It must always follow the first slash.

f ***v1****/vt1/vn1* ***v2****/vt2/vn2* ***v3****/vt3/vn3 . . .*

Polygonal geometry statement.
Specifies a face element and its vertex reference number. You can optionally include the texture vertex and vertex normal reference numbers.

The reference numbers for the vertices, texture vertices, and vertex normals must be separated by slashes (/). There is no space between the number and the slash.

v is the reference number for a vertex in the face element. A minimum of three vertices are required.

vt is an optional argument.
vt is the reference number for a texture vertex in the face element. It always follows the first slash.

vn is an optional argument.
vn is the reference number for a vertex normal in the face element. It must always follow the second slash.

Face elements use surface normals to indicate their orientation. If vertices are ordered counterclockwise around the face, both the face and the normal will point toward the viewer. If the vertex ordering is clockwise, both will point away from the viewer. If vertex normals are assigned, they should point in the general direction of the surface normal, otherwise unpredictable results may occur.

If a face has a texture map assigned to it and no texture vertices are assigned in the *f* statement, the texture map is ignored when the element is rendered.

NOTE: Any references to *fo* (face outline) are no longer valid as of version 2.11. You can use *f* (face) to get the same results. References to *fo* in existing .obj files will still be read, however, they will be written out as *f* when the file is saved.

curv *u0 u1 v1 v2* . . .

Element statement for free-form geometry.
Specifies a curve, its parameter range, and its control vertices. Although curves cannot be shaded or rendered, they are used by other Advanced Visualizer programs.

u0 is the starting parameter value for the curve. This is a floating point number.

u1 is the ending parameter value for the curve. This is a floating point number.

v is the vertex reference number for a control point. You can specify multiple control points. A minimum of two control points are required for a curve.

For a non-rational curve, the control points must be 3D. For a rational curve, the control points are 3D or 4D. The fourth coordinate (weight) defaults to 1.0 if omitted.

curv2 ***vp1 vp2*** *vp3. . .*

Free-form geometry statement.
Specifies a 2D curve on a surface and its control points. A 2D curve is used as an outer or inner trimming curve, as a special curve, or for connectivity.

vp is the parameter vertex reference number for the control point. You can specify multiple control points. A minimum of two control points is required for a 2D curve.

The control points are parameter vertices because the curve must lie in the parameter space of some surface. For a non-rational curve, the control vertices can be 2D. For a rational curve, the control vertices can be 2D or 3D. The third coordinate (weight) defaults to 1.0 if omitted.

surf ***s0 s1 t0 t1 v1****/vt1/vn1 v2/vt2/vn2 . . .*

Element statement for free-form geometry.
Specifies a surface, its parameter range, and its control vertices. The surface is evaluated within the global parameter range from *s0* to *s1* in the *u* direction and *t0* to *t1* in the *v* direction.

s0 is the starting parameter value for the surface in the u direction.

s1 is the ending parameter value for the surface in the u direction.

t0 is the starting parameter value for the surface in the v direction.

t1 is the ending parameter value for the surface in the v direction.

v is the reference number for a control vertex in the surface.

*v*t is an optional argument.
vt is the reference number for a texture vertex in the surface. It must always follow the first slash.

vn is an optional argument.
vn is the reference number for a vertex normal in the surface. It must always follow the second slash.

For a non-rational surface the control vertices are 3D. For a rational surface the control vertices can be 3D or 4D. The fourth coordinate (weight) defaults to 1.0 if omitted.

NOTE: For more information on the ordering of control points for surfaces, refer to the section on surfaces and control points in "Mathematics of free-form curves/surfaces" at the end of this appendix.

Examples

These are examples for polygonal geometry.

For examples using free-form geometry, see the examples at the end of the next section, "Free-form curve/surface body statements."

1. Square

This example shows a square that measures two units on each side and faces in the positive direction (toward the camera). Note that the ordering of the vertices is counterclockwise. This ordering determines that the square is facing forward.

```
v 0.000000 2.000000 0.000000
v 0.000000 0.000000 0.000000
v 2.000000 0.000000 0.000000
v 2.000000 2.000000 0.000000
f 1 2 3 4
```

2. Cube

This is a cube that measures two units on each side. Each vertex is shared by three different faces.

```
v 0.000000 2.000000 2.000000
v 0.000000 0.000000 2.000000
v 2.000000 0.000000 2.000000
v 2.000000 2.000000 2.000000
v 0.000000 2.000000 0.000000
v 0.000000 0.000000 0.000000
v 2.000000 0.000000 0.000000
v 2.000000 2.000000 0.000000
f 1 2 3 4
f 8 7 6 5
f 4 3 7 8
f 5 1 4 8
f 5 6 2 1
f 2 6 7 3
```

3. Cube with negative reference numbers

This is a cube with negative vertex reference numbers. Each element references the vertices stored immediately above it in the file. Note that vertices are not shared.

```
v 0.000000 2.000000 2.000000
v 0.000000 0.000000 2.000000
v 2.000000 0.000000 2.000000
v 2.000000 2.000000 2.000000
f -4 -3 -2 -1

v 2.000000 2.000000 0.000000
v 2.000000 0.000000 0.000000
v 0.000000 0.000000 0.000000
v 0.000000 2.000000 0.000000
f -4 -3 -2 -1

v 2.000000 2.000000 2.000000
v 2.000000 0.000000 2.000000
v 2.000000 0.000000 0.000000
v 2.000000 2.000000 0.000000
f -4 -3 -2 -1

v 0.000000 2.000000 0.000000
v 0.000000 2.000000 2.000000
v 2.000000 2.000000 2.000000
v 2.000000 2.000000 0.000000
f -4 -3 -2 -1

v 0.000000 2.000000 0.000000
v 0.000000 0.000000 0.000000
v 0.000000 0.000000 2.000000
v 0.000000 2.000000 2.000000
f -4 -3 -2 -1

v 0.000000 0.000000 2.000000
v 0.000000 0.000000 0.000000
v 2.000000 0.000000 0.000000
v 2.000000 0.000000 2.000000
f -4 -3 -2 -1
```

Free-form curve/surface body statements

You can specify additional information for free-form curve and surface elements using a series of statements called body statements. The series is concluded by an *end* statement.

Body statements are valid only when they appear between the free-form element statement (*curv*, *curv2*, *surf*) and the *end* statement. If they are anywhere else in the .obj file, they do not have any effect.

You can use body statements to specify the following values:

- parameter
- knot vector
- trimming loop
- hole
- special curve
- special point

You cannot use any other statements between the free-form curve or surface statement and the *end* statement. Using any other of type of statement may cause unpredictable results.

This portion of a sample file shows the knot vector values for a rational B-spline surface with a trimming loop. Notice the *end* statement to conclude the body statements.

```
cstype rat bspline
deg 2 2
surf -1.0 2.5 -2.0 2.0 -9 -8 -7 -6 -5 -4 -3 -2 -1
parm u -1.00 -1.00 -1.00 2.50 2.50 2.50
parm v -2.00 -2.00 -2.00 -2.00 -2.00 -2.00
trim 0.0 2.0 1
end
```

Parameter values and knot vectors

All curve and surface elements require a set of parameter values.

For polynomial curves and surfaces, this specifies global parameter values. For B-spline curves and surfaces, this specifies the knot vectors.

For surfaces, the parameter values must be specified for both the u and v directions. For curves, the parameter values must be specified for only the u direction.

If multiple parameter value statements for the same parametric direction are used inside a single curve or surface body, the last statement is used.

Trimming loops and holes

The trimming loop statement builds a single outer trimming loop as a sequence of curves which lie on a given surface.

The hole statement builds a single inner trimming loop as a sequence of curves which lie on a given surface. The inner loop creates a hole.

The curves are referenced by number in the same way vertices are referenced by face elements.

The individual curves must lie end-to-end to form a closed loop which does not intersect itself and which lies within the parameter range specified for the surface. The loop as a whole may be oriented in either direction (clockwise or counterclockwise).

To cut one or more holes in a region, use a trim statement followed by one or more hole statements. To introduce another trimmed region in the same surface, use another trim statement followed by one or more hole statements. The ordering that associates holes and the regions they cut is important and must be maintained.

If the first trim statement in the sequence is omitted, the enclosing outer trimming loop is taken to be the parameter range of the surface. If no trim or hole statements are specified, then the surface is trimmed at its parameter range.

This portion of a sample file shows a non-rational Bezier surface with two regions, each with a single hole:

```
cstype bezier
deg 1 1
surf 0.0 2.0 0.0 2.0 1 2 3 4
parm u 0.00 2.00
parm v 0.00 2.00
trim 0.0 4.0 1
hole 0.0 4.0 2
trim 0.0 4.0 3
hole 0.0 4.0 4
end
```

Special curve

A special curve statement builds a single special curve as a sequence of curves which lie on a given surface.

The curves are referenced by number in the same way vertices are referenced by face elements.

A special curve is guaranteed to be included in any triangulation of the surface. This means that the line formed by approximating the special curve with a sequence of straight line segments will actually appear as a sequence of triangle edges in the final triangulation.

Special point

A special point statement specifies that special geometric points are to be associated with a curve or surface. For space curves and trimming curves, the parameter vertices must be 1D. For surfaces, the parameter vertices must be 2D.

These special points will be included in any linear approximation of the curve or surface.

For space curves, this means that the point corresponding to the given curve parameter is included as one of the vertices in an approximation consisting of a sequence of line segments.

For surfaces, this means that the point corresponding to the given surface parameters is included as a triangle vertex in the triangulation.

For trimming curves, the treatment is slightly different: a special point on a trimming curve is essentially the same as a special point on the surface it trims.

The following portion of a sample files shows special points for a rational Bezier 2D curve on a surface.

```
vp -0.675  1.850  3.000
vp  0.915  1.930
vp  2.485  0.470  2.000
vp  2.485 -1.030
vp  1.605 -1.890 10.700
vp -0.745 -0.654  0.500
cstype rat bezier
curv2 -6 -5 -4 -3 -2 -1 -6
parm u 0.00 1.00 2.00
sp 2 3
end
```

Syntax

The following syntax statement are listed in order of normal use.

parm *u p1 p2* *p3. . .*
parm *v p1 p2* *p3 . . .*

Body statement for free-form geometry.
Specifies global parameter values. For B-spline curves and surfaces, this specifies the knot vectors.

u is the u direction for the parameter values.

v is the v direction for the parameter values.

To set *u* and *v* values, use separate command lines.

p is the global parameter or knot value. You can specify multiple values. A minimum of two parameter values are required. Parameter values must increase monotonically. The type of surface and the degree dictate the number of values required.

trim *u0 u1 curv2d* *u0 u1 curv2d . . .*

Body statement for free-form geometry.
Specifies a sequence of curves to build a single outer trimming loop.

u0 is the starting parameter value for the trimming curve *curv2d*.

u1 is the ending parameter value for the trimming curve *curv2d*.

curv2d is the index of the trimming curve lying in the parameter space of the surface. This curve must have been previously defined with the *curv2* statement.

hole *u0 u1 curv2d* *u0 u1 curv2d . . .*

Body statement for free-form geometry.
Specifies a sequence of curves to build a single inner trimming loop (hole).

u0 is the starting parameter value for the trimming curve *curv2d*.

u1 is the ending parameter value for the trimming curve *curv2d*.

curv2d is the index of the trimming curve lying in the parameter space of the surface. This curve must have been previously defined with the *curv2* statement.

scrv *u0 u1 curv2d* *u0 u1 curv2d . . .*

Body statement for free-form geometry.
Specifies a sequence of curves which lie on the given surface to build a single special curve.

u0 is the starting parameter value for the special curve *curv2d*.

u1 is the ending parameter value for the special curve *curv2d*.

curv2d is the index of the special curve lying in the parameter space of the surface. This curve must have been previously defined with the *curv2* statement.

sp *vp1* *vp. . .*

Body statement for free-form geometry.
Specifies special geometric points to be associated with a curve or surface. For space curves and trimming curves, the parameter vertices must be 1D. For surfaces, the parameter vertices must be 2D.

vp is the reference number for the parameter vertex of a special point to be associated with the parameter space point of the curve or surface.

end

Body statement for free-form geometry.
Specifies the end of a curve or surface body begun by a *curv, curv2,* or *surf* statement.

Examples

1. Taylor curve

For creating a single-segment Taylor polynomial curve of the form:

$$x = 3 + 2.3t + 7.98t^2 + 8.3t^3 + 6.34t^4$$
$$y = 1 - 10.1t + 5.4t^2 - 4.7t^3 + 2.03t^4$$
$$z = -2.5 + 0.5t - 7.0t^2 + 18.1t^3 + 0.08t^4$$

and evaluated between the global parameters 0.5 and 1.6:

```
v   3.000    1.000   -2.500
v   2.300  -10.100    0.500
v   7.980    5.400   -7.000
v   8.300   -4.700   18.100
v   6.340    2.030    0.080
cstype taylor
deg 4
curv 0.500 1.600 1 2 3 4 5
parm u 0.000 2.000
end
```

2. Bezier curve

This example shows a non-rational Bezier curve with 13 control points.

```
v -2.300000 1.950000 0.000000
v -2.200000 0.790000 0.000000
v -2.340000 -1.510000 0.000000
v -1.530000 -1.490000 0.000000
v -0.720000 -1.470000 0.000000
v -0.780000 0.230000 0.000000
v 0.070000 0.250000 0.000000
v 0.920000 0.270000 0.000000
v 0.800000 -1.610000 0.000000
v 1.620000 -1.590000 0.000000
v 2.440000 -1.570000 0.000000
v 2.690000 0.670000 0.000000
v 2.900000 1.980000 0.000000
# 13 vertices

cstype bezier
ctech cparm 1.000000
deg 3
curv 0.000000 4.000000 1 2 3 4 5 6 7 8 9 10 \
11 12 13
parm u 0.000000 1.000000 2.000000 3.000000  \
4.000000
end
# 1 element
```

3. B-spline surface

This is an example of a cubic B-spline surface.

```
g bspatch
v -5.000000 -5.000000 -7.808327
v -5.000000 -1.666667 -7.808327
v -5.000000 1.666667 -7.808327
v -5.000000 5.000000 -7.808327
v -1.666667 -5.000000 -7.808327
v -1.666667 -1.666667 11.977780
v -1.666667 1.666667 11.977780
v -1.666667 5.000000 -7.808327
v 1.666667 -5.000000 -7.808327
v 1.666667 -1.666667 11.977780
v 1.666667 1.666667 11.977780
v 1.666667 5.000000 -7.808327
v 5.000000 -5.000000 -7.808327
v 5.000000 -1.666667 -7.808327
v 5.000000 1.666667 -7.808327
v 5.000000 5.000000 -7.808327
# 16 vertices

cstype bspline
stech curv 0.5 10.000000
deg 3 3
8surf 0.000000 1.000000 0.000000 1.000000 13 14 \
15 16 9 10 11 12 5 6 7 8 1 2 3 4
parm u -3.000000 -2.000000 -1.000000 0.000000  \
1.000000 2.000000 3.000000 4.000000
parm v -3.000000 -2.000000 -1.000000 0.000000  \
1.000000 2.000000 3.000000 4.000000
end
# 1 element
```

4. Cardinal surface

This example shows a Cardinal surface.

```
v -5.000000 -5.000000 0.000000
v -5.000000 -1.666667 0.000000
v -5.000000 1.666667 0.000000
v -5.000000 5.000000 0.000000
v -1.666667 -5.000000 0.000000
v -1.666667 -1.666667 0.000000
v -1.666667 1.666667 0.000000
v -1.666667 5.000000 0.000000
v 1.666667 -5.000000 0.000000
v 1.666667 -1.666667 0.000000
v 1.666667 1.666667 0.000000
v 1.666667 5.000000 0.000000
v 5.000000 -5.000000 0.000000
```

```
                  v 5.000000 -1.666667 0.000000
```

scrv *u0 u1 curv2d* *u0 u1 curv2d . . .*

Body statement for free-form geometry.
Specifies a sequence of curves which lie on the given surface to build a single special curve.

u0 is the starting parameter value for the special curve *curv2d*.

u1 is the ending parameter value for the special curve *curv2d*.

curv2d is the index of the special curve lying in the parameter space of the surface. This curve must have been previously defined with the *curv2* statement.

sp *vp1* *vp. . .*

Body statement for free-form geometry.
Specifies special geometric points to be associated with a curve or surface. For space curves and trimming curves, the parameter vertices must be 1D. For surfaces, the parameter vertices must be 2D.

vp is the reference number for the parameter vertex of a special point to be associated with the parameter space point of the curve or surface.

end

Body statement for free-form geometry.
Specifies the end of a curve or surface body begun by a *curv, curv2,* or *surf* statement.

Examples

1. Taylor curve

For creating a single-segment Taylor polynomial curve of the form:

$$
\begin{aligned}
x &= 3 + 2.3t + 7.98t^2 + 8.3t^3 + 6.34t^4 \\
y &= 1 - 10.1t + 5.4t^2 - 4.7t^3 + 2.03t^4 \\
z &= -2.5 + 0.5t - 7.0t^2 + 18.1t^3 + 0.08t^4
\end{aligned}
$$

and evaluated between the global parameters 0.5 and 1.6:

```
v   3.000    1.000   -2.500
v   2.300  -10.100    0.500
v   7.980    5.400   -7.000
v   8.300   -4.700   18.100
v   6.340    2.030    0.080
cstype taylor
deg 4
curv 0.500 1.600 1 2 3 4 5
parm u 0.000 2.000
end
```

2. Bezier curve

This example shows a non-rational Bezier curve with 13 control points.

```
v -2.300000 1.950000 0.000000
v -2.200000 0.790000 0.000000
v -2.340000 -1.510000 0.000000
v -1.530000 -1.490000 0.000000
v -0.720000 -1.470000 0.000000
v -0.780000 0.230000 0.000000
v 0.070000 0.250000 0.000000
v 0.920000 0.270000 0.000000
v 0.800000 -1.610000 0.000000
v 1.620000 -1.590000 0.000000
v 2.440000 -1.570000 0.000000
v 2.690000 0.670000 0.000000
v 2.900000 1.980000 0.000000
# 13 vertices

cstype bezier
ctech cparm 1.000000
deg 3
curv 0.000000 4.000000 1 2 3 4 5 6 7 8 9 10 \
11 12 13
parm u 0.000000 1.000000 2.000000 3.000000  \
4.000000
end
# 1 element
```

3. B-spline surface

This is an example of a cubic B-spline surface.

```
g bspatch
v -5.000000 -5.000000 -7.808327
v -5.000000 -1.666667 -7.808327
v -5.000000 1.666667 -7.808327
v -5.000000 5.000000 -7.808327
v -1.666667 -5.000000 -7.808327
v -1.666667 -1.666667 11.977780
v -1.666667 1.666667 11.977780
v -1.666667 5.000000 -7.808327
v 1.666667 -5.000000 -7.808327
v 1.666667 -1.666667 11.977780
v 1.666667 1.666667 11.977780
v 1.666667 5.000000 -7.808327
v 5.000000 -5.000000 -7.808327
v 5.000000 -1.666667 -7.808327
v 5.000000 1.666667 -7.808327
v 5.000000 5.000000 -7.808327
# 16 vertices

cstype bspline
stech curv 0.5 10.000000
deg 3 3
8surf 0.000000 1.000000 0.000000 1.000000 13 14 \
15 16 9 10 11 12 5 6 7 8 1 2 3 4
parm u -3.000000 -2.000000 -1.000000 0.000000  \
1.000000 2.000000 3.000000 4.000000
parm v -3.000000 -2.000000 -1.000000 0.000000  \
1.000000 2.000000 3.000000 4.000000
end
# 1 element
```

4. Cardinal surface

This example shows a Cardinal surface.

```
v -5.000000 -5.000000 0.000000
v -5.000000 -1.666667 0.000000
v -5.000000 1.666667 0.000000
v -5.000000 5.000000 0.000000
v -1.666667 -5.000000 0.000000
v -1.666667 -1.666667 0.000000
v -1.666667 1.666667 0.000000
v -1.666667 5.000000 0.000000
v 1.666667 -5.000000 0.000000
v 1.666667 -1.666667 0.000000
v 1.666667 1.666667 0.000000
v 1.666667 5.000000 0.000000
v 5.000000 -5.000000 0.000000
v 5.000000 -1.666667 0.000000
v 5.000000 1.666667 0.000000
v 5.000000 5.000000 0.000000
# 16 vertices

cstype cardinal
stech cparma 1.000000 1.000000
deg 3 3
surf 0.000000 1.000000 0.000000 1.000000 13 14 \
15 16 9 10 11 12 5 6 7 8 1 2 3 4
parm u 0.000000 1.000000
parm v 0.000000 1.000000
end
# 1 element
```

5. Rational B-spline surface

This example creates a second-degree, rational B-spline surface using open, uniform knot vectors. A texture map is applied to the surface.

```
v -1.3 -1.0  0.0
v  0.1 -1.0  0.4  7.6
v  1.4 -1.0  0.0  2.3
v -1.4  0.0  0.2
v  0.1  0.0  0.9  0.5
v  1.3  0.0  0.4  1.5
v -1.4  1.0  0.0  2.3
v  0.1  1.0  0.3  6.1
v  1.1  1.0  0.0  3.3
vt 0.0  0.0
vt 0.5  0.0
vt 1.0  0.0
vt 0.0  0.5
vt 0.5  0.5
vt 1.0  0.5
vt 0.0  1.0
vt 0.5  1.0
vt 1.0  1.0
cstype rat bspline
deg 2 2
surf 0.0 1.0 0.0 1.0 1/1 2/2 3/3 4/4 5/5 6/6 \
7/7 8/8 9/9
parm u 0.0 0.0 0.0 1.0 1.0 1.0
parm v 0.0 0.0 0.0 1.0 1.0 1.0
end
```

6. Trimmed NURB surface

This is a complete example of a file containing a trimmed NURB surface with negative reference numbers for vertices.

```
# trimming curve
vp -0.675  1.850  3.000
vp  0.915  1.930
vp  2.485  0.470  2.000
vp  2.485 -1.030
vp  1.605 -1.890 10.700
vp -0.745 -0.654  0.500
cstype rat bezier
deg 3
curv2 -6 -5 -4 -3 -2 -1 -6
parm u 0.00 1.00 2.00
end
# surface
v -1.350 -1.030 0.000
v  0.130 -1.030 0.432 7.600
v  1.480 -1.030 0.000 2.300
v -1.460  0.060 0.201
v  0.120  0.060 0.915 0.500
v  1.380  0.060 0.454 1.500
v -1.480  1.030 0.000 2.300
v  0.120  1.030 0.394 6.100
v  1.170  1.030 0.000 3.300
cstype rat bspline
deg 2 2
surf -1.0 2.5 -2.0 2.0 -9 -8 -7 -6 -5 -4 -3 -2 -1
parm u -1.00 -1.00 -1.00 2.50 2.50 2.50
parm v -2.00 -2.00 -2.00 -2.00 -2.00 -2.00
trim 0.0 2.0 1
end
```

7. Two trimming regions with a hole

This example shows a Bezier surface with two trimming regions, each with a hole in them.

```
# outer loop of first region
deg 1
cstype bezier
vp 0.100 0.100
vp 0.900 0.100
vp 0.900 0.900
vp 0.100 0.900
curv2 1 2 3 4 1
parm u 0.00 1.00 2.00 3.00 4.00
end
# hole in first region
vp 0.300 0.300
vp 0.700 0.300
vp 0.700 0.700
vp 0.300 0.700
curv2 5 6 7 8 5
parm u 0.00 1.00 2.00 3.00 4.00
end
# outer loop of second region
vp 1.100 1.100
vp 1.900 1.100
vp 1.900 1.900
vp 1.100 1.900
curv2 9 10 11 12 9
parm u 0.00 1.00 2.00 3.00 4.00
end
# hole in second region
vp 1.300 1.300
vp 1.700 1.300
vp 1.700 1.700
vp 1.300 1.700
curv2 13 14 15 16 13
parm u 0.00 1.00 2.00 3.00 4.00
end
# surface
v 0.000 0.000 0.000
v 1.000 0.000 0.000
v 0.000 1.000 0.000
v 1.000 1.000 0.000
deg 1 1
```

Free-form curve/surface body statements

```
cstype bezier
surf 0.0 2.0 0.0 2.0 1 2 3 4
parm u 0.00 2.00
parm v 0.00 2.00
trim 0.0 4.0 1
hole 0.0 4.0 2
trim 0.0 4.0 3
hole 0.0 4.0 4
end
```

8. Trimming with a special curve

This example is similar to the trimmed NURB surface example (6), except there is a special curve on the surface. This example uses negative vertex numbers.

```
# trimming curve
vp -0.675  1.850  3.000
vp  0.915  1.930
vp  2.485  0.470  2.000
vp  2.485 -1.030
vp  1.605 -1.890 10.700
vp -0.745 -0.654  0.500
cstype rat bezier
deg 3
curv2 -6 -5 -4 -3 -2 -1 -6
parm u 0.00 1.00 2.00
end
# special curve
vp -0.185  0.322
vp  0.214  0.818
vp  1.652  0.207
vp  1.652 -0.455
curv2 -4 -3 -2 -1
parm u 2.00 10.00
end
# surface
v -1.350 -1.030 0.000
v  0.130 -1.030 0.432 7.600
v  1.480 -1.030 0.000 2.300
v -1.460  0.060 0.201
v  0.120  0.060 0.915 0.500
v  1.380  0.060 0.454 1.500
v -1.480  1.030 0.000 2.300
v  0.120  1.030 0.394 6.100
v  1.170  1.030 0.000 3.300
cstype rat bspline
deg 2 2
surf -1.0 2.5 -2.0 2.0 -9 -8 -7 -6 -5 -4 -3 -2 -1
parm u -1.00 -1.00 -1.00 2.50 2.50 2.50
parm v -2.00 -2.00 -2.00 2.00 2.00 2.00
trim 0.0 2.0 1
scrv 4.2 9.7 2
end
```

9. Trimming with special points

This example extends the trimmed NURB surface example (6) to include special points on both the trimming curve and surface. A space curve with a special point is also included. This example uses negative vertex numbers.

```
# special point and space curve data
vp 0.500
vp 0.700
vp 1.100
vp 0.200 0.950
v  0.300 1.500 0.100
v  0.000  0.000  0.000
v  1.000  1.000  0.000
v  2.000  1.000  0.000
v  3.000  0.000  0.000
cstype bezier
deg 3
curv 0.2 0.9 -4 -3 -2 -1
sp 1
parm u 0.00 1.00
end
# trimming curve
vp -0.675  1.850  3.000
vp  0.915  1.930
vp  2.485  0.470  2.000
vp  2.485 -1.030
vp  1.605 -1.890 10.700
vp -0.745 -0.654  0.500
cstype rat bezier
curv2 -6 -5 -4 -3 -2 -1 -6
parm u 0.00 1.00 2.00
sp 2 3
end
# surface
v -1.350 -1.030 0.000
v  0.130 -1.030 0.432 7.600
v  1.480 -1.030 0.000 2.300
v -1.460  0.060 0.201
v  0.120  0.060 0.915 0.500
v  1.380  0.060 0.454 1.500
v -1.480  1.030 0.000 2.300
v  0.120  1.030 0.394 6.100
v  1.170  1.030 0.000 3.300
```

```
cstype rat bspline
deg 2 2
surf -1.0 2.5 -2.0 2.0 -9 -8 -7 -6 -5 -4 -3 -2 -1
parm u -1.00 -1.00 -1.00 2.50 2.50 2.50
parm v -2.00 -2.00 -2.00 2.00 2.00 2.00
trim 0.0 2.0 1
sp 4
end
```

Connectivity between free-form surfaces

Connectivity connects two surfaces along their trimming curves.

The *con* statement specifies the first surface with its trimming curve and the second surface with its trimming curve. This information is useful for edge merging. Without this surface and curve data, connectivity must be determined numerically at greater expense and with reduced accuracy using the *mg* statement.

Connectivity between surfaces in different merging groups is ignored. Also, although connectivity which crosses points of C^1discontinuity in trimming curves is legal, it is not recommended. Instead, use two connectivity statements which meet at the point of discontinuity.

The two curves and their starting and ending parameters should all map to the same curve and starting and ending points in object space.

Syntax

con *surf_1 q0_1 q1_1 curv2d_1 surf_2 q0_2 q1_2 curv2d_2*

Free-form geometry statement.
Specifies connectivity between two surfaces.

surf_1 is the index of the first surface.

q0_1 is the starting parameter for the curve referenced by *curv2d_1*.

q1_1 is the ending parameter for the curve referenced by *curv2d_1*.

curv2d_1 is the index of a curve on the first surface. This curve must have been previously defined with the *curv2* statement.

surf_2 is the index of the second surface.

q0_2 is the starting parameter for the curve referenced by *curv2d_2*.

q1_2 is the ending parameter for the curve referenced by *curv2d_2*.

curv2d_2 is the index of a curve on the second surface. This curve must have been previously defined with the *curv2* statement.

Example

1. Connectivity between two surfaces

This example shows the connectivity between two surfaces with trimming curves.

```
cstype bezier
deg 1 1

v 0 0 0
v 1 0 0
v 0 1 0
v 1 1 0

vp 0 0
vp 1 0
vp 1 1
vp 0 1

curv2 1 2 3 4 1
parm u 0.0 1.0 2.0 3.0 4.0
end

surf 0.0 1.0 0.0 1.0 1 2 3 4
parm u 0.0 1.0
parm v 0.0 1.0
trim 0.0 4.0 1
end

v 1 0 0
v 2 0 0
v 1 1 0
v 2 1 0

surf 0.0 1.0 0.0 1.0 5 6 7 8
parm u 0.0 1.0
parm v 0.0 1.0
trim 0.0 4.0 1
end

con 1 2.0 2.0 1 2 4.0 3.0 1
```

Grouping

There are four statements in the .obj file to help you manipulate groups of elements:

- Group name statements are used to organize collections of elements and simplify data manipulation for operations in Model.
- Smoothing group statements let you identify elements over which normals are to be interpolated to give those elements a smooth, non-faceted appearance. This is a quick way to specify vertex normals.
- Merging group statements are used to identify free-form elements that should be inspected for adjacency detection. You can also use merging groups to exclude surfaces which are close enough to be considered adjacent but should not be merged.
- Object name statements let you assign a name to an entire object in a single file.

All grouping statements are state-setting. This means that once a group statement is set, it applies to all elements that follow until the next group statement.

This portion of a sample file shows a single element which belongs to three groups. The smoothing group is turned off.

```
g square thing all
s off
f 1 2 3 4
```

This example shows two surfaces in merging group 1 with a merge resolution of 0.5.

```
mg 1 .5
surf 0.0 1.0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
surf 0.0 1.0 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
```

Syntax

g *group_name1* *group_name2 . . .*

Polygonal and free-form geometry statement.
Specifies the group name for the elements that follow it. You can have multiple group names. If there are multiple groups on one line, the data that follows belong to all groups. Group information is optional.

group_name is the name for the group. Letters, numbers, and combinations of letters and numbers are accepted for group names. The default group name is *default*.

s *group_number*

Polygonal and free-form geometry statement.
Sets the smoothing group for the elements that follow it. If you do not want to use a smoothing group, specify *off* or a value of 0.

To display with smooth shading in Model and PreView, you must create vertex normals after you have assigned the smoothing groups. You can create vertex normals with the *vn* statement or with the Model program.

To smooth polygonal geometry for rendering with Image, it is sufficient to put elements in some smoothing group. However, vertex normals override smoothing information for Image.

group_number is the smoothing group number. To turn off smoothing groups, use a value of 0 or *off*. Polygonal elements use group numbers to put elements in different smoothing groups. For free-form surfaces, smoothing groups are either turned on or off; there is no difference between values greater than 0.

mg *group_number res*

Free-form geometry statement.
Sets the merging group and merge resolution for the free-form surfaces that follow it. If you do not want to use a merging group, specify *off* or a value of 0.

Adjacency detection is performed only within groups, never between groups. Connectivity between surfaces in different merging groups is not allowed. Surfaces in the same merging group are merged together along edges that are within the distance *res* apart.

NOTE: Adjacency detection is an expensive numerical comparison process. It is best to restrict this process to as small a domain as possible by using small merging groups.

---

group_number is the merging group number. To turn off adjacency detection, use a value of 0 or *off*.

res is the maximum distance between two surfaces that will be merged together. The resolution must be a value greater than 0. This is a required argument only when using merging groups.

o *object_name*

Polygonal and free-form geometry statement.
Optional statement; it is not processed by any Wavefront programs. It specifies a user-defined object name for the elements defined after this statement.

object_name is the user-defined object name. There is no default.

Examples

1. Cube with group names

The following example is a cube with each of its faces placed in a separate group. In addition, all elements belong to the group *cube*.

```
v 0.000000 2.000000 2.000000
v 0.000000 0.000000 2.000000
v 2.000000 0.000000 2.000000
v 2.000000 2.000000 2.000000
v 0.000000 2.000000 0.000000
v 0.000000 0.000000 0.000000
v 2.000000 0.000000 0.000000
v 2.000000 2.000000 0.000000
# 8 vertices

g front cube
f 1 2 3 4
g back cube
f 8 7 6 5
g right cube
f 4 3 7 8
g top cube
f 5 1 4 8
g left cube
f 5 6 2 1
g bottom cube
f 2 6 7 3
# 6 elements
```

2. Two adjoining squares with a smoothing group

This example shows two adjoining squares that share a common edge. The squares are placed in a smoothing group to ensure that their common edge will be smoothed when rendered with Image.

```
v 0.000000 2.000000 0.000000
v 0.000000 0.000000 0.000000
v 2.000000 0.000000 0.000000
v 2.000000 2.000000 0.000000
v 4.000000 0.000000 -1.255298
v 4.000000 2.000000 -1.255298
# 6 vertices

g all
s 1
f 1 2 3 4
f 4 3 5 6
# 2 elements
```

3. Two adjoining squares with vertex normals

This example also shows two squares that share a common edge. Vertex normals have been added to the corners of each square to ensure that their common edge will be smoothed during display in Model and PreView and when rendered with Image.

```
v 0.000000 2.000000 0.000000
v 0.000000 0.000000 0.000000
v 2.000000 0.000000 0.000000
v 2.000000 2.000000 0.000000
v 4.000000 0.000000 -1.255298
v 4.000000 2.000000 -1.255298
vn 0.000000 0.000000 1.000000
vn 0.000000 0.000000 1.000000
vn 0.276597 0.000000 0.960986
vn 0.276597 0.000000 0.960986
vn 0.531611 0.000000 0.846988
vn 0.531611 0.000000 0.846988
# 6 vertices

# 6 normals

g all
s 1
f 1//1 2//2 3//3 4//4
f 4//4 3//3 5//5 6//6
# 2 elements
```

4. Merging group

This example shows two Bezier surfaces that meet at a common edge. They have both been placed in the same merging group to ensure continuity at the edge where they meet. This prevents "cracks" from appearing along the seam between the two surfaces during rendering. Merging groups will be ignored during flat-shading, smooth-shading, and material shading of the surface.

```
v -4.949854 -5.000000 0.000000
v -4.949854 -1.666667 0.000000
v -4.949854 1.666667 0.000000
v -4.949854 5.000000 0.000000
v -1.616521 -5.000000 0.000000
v -1.616521 -1.666667 0.000000
v -1.616521 1.666667 0.000000
v -1.616521 5.000000 0.000000
v 1.716813 -5.000000 0.000000
v 1.716813 -1.666667 0.000000
v 1.716813 1.666667 0.000000
v 1.716813 5.000000 0.000000
v 5.050146 -5.000000 0.000000
v 5.050146 -1.666667 0.000000
v 5.050146 1.666667 0.000000
v 5.050146 5.000000 0.000000
v -15.015566 -4.974991 0.000000
v -15.015566 -1.641658 0.000000
v -15.015566 1.691675 0.000000
v -15.015566 5.025009 0.000000
v -11.682233 -4.974991 0.000000
v -11.682233 -1.641658 0.000000
v -11.682233 1.691675 0.000000
v -11.682233 5.025009 0.000000
v -8.348900 -4.974991 0.000000
v -8.348900 -1.641658 0.000000
v -8.348900 1.691675 0.000000
v -8.348900 5.025009 0.000000
v -5.015566 -4.974991 0.000000
v -5.015566 -1.641658 0.000000
v -5.015566 1.691675 0.000000
v -5.015566 5.025009 0.000000

mg 1 0.500000

cstype bezier
deg 3 3
surf 0.000000 1.000000 0.000000 1.000000 13 14 \
```

```
15 16 9 10 11 12 5 6 7 8 1 2 3 4
parm u 0.000000 1.000000
parm v 0.000000 1.000000
end
surf 0.000000 1.000000 0.000000 1.000000 29 30 31
32 25 26 27 28 21 22 \
 23 24 17 18 19 20
parm u 0.000000 1.000000
parm v 0.000000 1.000000
end
```

Display/render attributes

Display and render attributes describe how an object looks when displayed in Model and PreView or when rendered with Image.

Some attributes apply to both free-form and polygonal geometry, such as material name and library, ray tracing, and shadow casting. Interpolation attributes apply only to polygonal geometry. Curve and surface resolutions are used for only free-form geometry.

The following chart shows the display and render statements available for polygonal and free-form geometry.

Table B1-1. Display and render attributes

polygonal only	**polygonal or free-form**	**free-form only**
bevel	lod	ctech
c_interp	usemtl	stech
d_interp	mtllib	
	shadow_obj	
	trace_obj	

All display and render attribute statements are state-setting. This means that once an attribute statement is set, it applies to all elements that follow until it is reset to a different value.

The following sample shows rendering and display statements for a face element.:

```
s 1
usemtl blue
usemap marble
f 1 2 3 4
```

Syntax

The following syntax statements are listed by the type of geometry. First are statements for polygonal geometry. Second are statements for both free-form and polygonal geometry. Third are statements for free-form geometry only.

bevel *on/off*

Polygonal geometry statement.
Sets bevel interpolation on or off. It works only with beveled objects, that is, objects with sides separated by beveled faces.

Bevel interpolation uses normal vector interpolation to give an illusion of roundness to a flat bevel. It does not affect the smoothing of non-bevelled faces.

Bevel interpolation does not alter the geometry of the original object.

on turns on bevel interpolation.

off turns off bevel interpolation. The default is *off*.

NOTE: Image cannot render bevel-interpolated elements that have vertex normals.

c_interp *on/off*

Polygonal geometry statement.
Sets color interpolation on or off.

Color interpolation creates a blend across the surface of a polygon between the materials assigned to its vertices. This creates a blending of colors across a face element.

To support color interpolation, materials must be assigned per vertex, not per element. The illumination models for all materials of vertices attached to the polygon must be the same. Color interpolation applies to the values for ambient (Ka), diffuse (Kd), specular (Ks), and specular highlight (Ns) material properties.

on turns on color interpolation.

off turns off color interpolation. The default is *off*.

d_interp *on/off*

Polygonal geometry statement.
Sets dissolve interpolation on or off.

Dissolve interpolation creates an interpolation or blend across a polygon between the dissolve (d) values of the materials assigned to its vertices. This feature is used to create effects exhibiting varying degrees of apparent transparency, as in glass or clouds.

To support dissolve interpolation, materials must be assigned per vertex, not per element. All the materials assigned to the vertices involved in the dissolve interpolation must contain a dissolve factor command to specify a dissolve.

on turns on dissolve interpolation.

off turns off dissolve interpolation. The default is *off*.

lod *level*

Polygonal and free-form geometry statement.
Sets the level of detail to be displayed in a PreView animation. The level of detail feature lets you control which elements of an object are displayed while working in PreView.

level is the level of detail to be displayed. When you set the level of detail to 0 or omit the *lod* statement, all elements are displayed. Specifying an integer between 1 and 100 sets the level of detail to be displayed when reading the .obj file.

maplib *filename1* *filename2 . . .*

This is a rendering identifier that specifies the map library file for the texture map definitions set with the *usemap* identifier. You can specify multiple filenames with *maplib*. If multiple filenames are specified, the first file listed is searched first for the map definition, the second file is searched next, and so on.

When you assign a map library using the Model program, Model allows only one map library per .obj file. You can assign multiple libraries using a text editor.

filename is the name of the library file where the texture maps are defined. There is no default.

usemap *map_name/off*

This is a rendering identifier that specifies the texture map name for the element following it. To turn off texture mapping, specify *off* instead of the map name.

If you specify texture mapping for a face without texture vertices, the texture map will be ignored.

map_name is the name of the texture map.

off turns off texture mapping. The default is *off*.

usemtl *material_name*

Polygonal and free-form geometry statement.
Specifies the material name for the element following it. Once a material is assigned, it cannot be turned off; it can only be changed.

material_name is the name of the material. If a material name is not specified, a white material is used.

mtllib *filename1* *filename2 . . .*

Polygonal and free-form geometry statement.
Specifies the material library file for the material definitions set with the *usemtl* statement. You can specify multiple filenames with *mtllib*. If multiple filenames are specified, the first file listed is searched first for the material definition, the second file is searched next, and so on.

When you assign a material library using the Model program, only one map library per .obj file is allowed. You can assign multiple libraries using a text editor.

filename is the name of the library file that defines the materials. There is no default.

shadow_obj *filename*

Polygonal and free-form geometry statement.
Specifies the shadow object filename. This object is used to cast shadows for the current object. Shadows are only visible in a rendered image; they cannot be seen using hardware shading. The shadow object is invisible except for its shadow.

An object will cast shadows only if it has a shadow object. You can use an object as its own shadow object. However, a simplified version of the original object is usually preferable for shadow objects, since shadow casting can greatly increase rendering time.

filename is the filename for the shadow object. You can enter any valid object filename for the shadow object. The object file can be an .obj or .mod file. If a filename is given without an extension, an extension of .obj is assumed.

Only one shadow object can be stored in a file. If more than one shadow object is specified, the last one specified will be used.

trace_obj *filename*

Polygonal and free-form geometry statement.
Specifies the ray tracing object filename. This object will be used in generating reflections of the current object on reflective surfaces. Reflections are only visible in a rendered image; they cannot be seen using hardware shading.

An object will appear in reflections only if it has a trace object. You can use an object as its own trace object. However, a simplified version of the original object is usually preferable for trace objects, since ray tracing can greatly increase rendering time.

filename is the filename for the ray tracing object. You can enter any valid object filename for the trace object. You can enter any valid object filename for the shadow object. The object file can be an .obj or .mod file. If a filename is given without an extension, an extension of .obj is assumed.

Only one trace object can be stored in a file. If more than one is specified, the last one is used.

ctech *technique resolution*

Free-form geometry statement.
Specifies a curve approximation technique. The arguments specify the technique and resolution for the curve.

You must select from one of the following three techniques.

ctech cparm *res*

Specifies a curve with constant parametric subdivision using one resolution parameter. Each polynomial segment of the curve is subdivided *n* times in parameter space, where *n* is the resolution parameter multiplied by the degree of the curve.

res is the resolution parameter. The larger the value, the finer the resolution. If *res* has a value of 0, each polynomial curve segment is represented by a single line segment.

ctech cspace *maxlength*

Specifies a curve with constant spatial subdivision. The curve is approximated by a series of line segments whose lengths in real space are less than or equal to the *maxlength*.

maxlength is the maximum length of the line segments. The smaller the value, the finer the resolution.

ctech curv *maxdist maxangle*

Specifies curvature-dependent subdivision using separate resolution parameters for the maximum distance and the maximum angle.

The curve is approximated by a series of line segments in which 1) the distance in object space between a line segment and the actual curve must be less than the *maxdist* parameter and 2) the angle in degrees between tangent vectors at the ends of a line segment must be less than the *maxangle* parameter.

maxdist is the distance in real space between a line segment and the actual curve.

maxangle is the angle (in degrees) between tangent vectors at the ends of a line segment.

The smaller the values for *maxdist* and *maxangle*, the finer the resolution.

NOTE: Approximation information for trimming, hole, and special curves is stored in the corresponding surface. The *ctech* statement for the surface is used, not the *ctech* statement applied to the *curv2* statement. Although untrimmed surfaces have no explicit trimming loop, a loop is constructed which bounds the legal parameter range. This implicit loop follows the same rules as any other loop and is approximated according to the *ctech* information for the surface.

---

stech *technique resolution*

Free-form geometry statement.
Specifies a surface approximation technique. The arguments specify the technique and resolution for the surface.

You must select from one of the following techniques:

stech cparma *ures vres*

Specifies a surface with constant parametric subdivision using separate resolution parameters for the u and v directions. Each patch of the surface is subdivided *n* times in parameter space, where *n* is the resolution parameter multiplied by the degree of the surface.

ures is the resolution parameter for the u direction.

vres is the resolution parameter for the v direction.

The larger the values for *ures* and *vres*, the finer the resolution. If you enter a value of 0 for both *ures* and *vres*, each patch is approximated by two triangles.

stech cparmb *uvres*

Specifies a surface with constant parametric subdivision, with refinement using one resolution parameter for both the u and v directions.

An initial triangulation is performed using only the points on the trimming curves. This triangulation is then refined until all edges are of an appropriate length. The resulting triangles are not oriented along isoparametric lines as they are in the *cparma* technique.

uvres is the resolution parameter for both the u and v directions. The larger the value, the finer the resolution.

stech cspace *maxlength*

Specifies a surface with constant spatial subdivision.

The surface is subdivided in rectangular regions until the length in real space of any rectangle edge is less than the *maxlength*. These rectangular regions are then triangulated.

maxlength is the length in real space of any rectangle edge. The smaller the value, the finer the resolution.

stech curv *maxdist maxangle*

Specifies a surface with curvature-dependent subdivision using separate resolution parameters for the maximum distance and the maximum angle.

The surface is subdivided in rectangular regions until 1) the distance in real space between the approximating rectangle and the actual surface is less than the *maxdist* (approximately) and 2) the angle in degrees between surface normals at the corners of the rectangle is less than the *maxangle*. Following subdivision, the regions are triangulated.

maxdist is the distance in real space between the approximating rectangle and the actual surface.

maxangle is the angle in degrees between surface normals at the corners of the rectangle.

The smaller the values for *maxdist* and *maxangle*, the finer the resolution.

Examples

1. Cube with materials

This cube has a different material applied to each of its faces.

```
mtllib master.mtl

v 0.000000 2.000000 2.000000
v 0.000000 0.000000 2.000000
v 2.000000 0.000000 2.000000
v 2.000000 2.000000 2.000000
v 0.000000 2.000000 0.000000
v 0.000000 0.000000 0.000000
v 2.000000 0.000000 0.000000
v 2.000000 2.000000 0.000000
# 8 vertices

g front
usemtl red
f 1 2 3 4
g back
usemtl blue
f 8 7 6 5
g right
usemtl green
f 4 3 7 8
g top
usemtl gold
f 5 1 4 8
g left
usemtl orange
f 5 6 2 1
g bottom
usemtl purple
f 2 6 7 3
# 6 elements
```

2. Cube casting a shadow

In this example, the cube casts a shadow on the other objects when it is rendered with Image. The cube, which is stored in the file *cube.obj*, references itself as the shadow object.

```
mtllib master.mtl
shadow_obj cube.obj

v 0.000000 2.000000 2.000000
v 0.000000 0.000000 2.000000
v 2.000000 0.000000 2.000000
v 2.000000 2.000000 2.000000
v 0.000000 2.000000 0.000000
v 0.000000 0.000000 0.000000
v 2.000000 0.000000 0.000000
v 2.000000 2.000000 0.000000
# 8 vertices

g front
usemtl red
f 1 2 3 4
g back
usemtl blue
f 8 7 6 5
g right
usemtl green
f 4 3 7 8
g top
usemtl gold
f 5 1 4 8
g left
usemtl orange
f 5 6 2 1
g bottom
usemtl purple
f 2 6 7 3
# 6 elements
```

3. Cube casting a reflection

This cube casts its reflection on any reflective objects when it is rendered with Image. The cube, which is stored in the file *cube.obj*, references itself as the trace object.

```
mtllib master.mtl
trace_obj cube.obj

v 0.000000 2.000000 2.000000
v 0.000000 0.000000 2.000000
v 2.000000 0.000000 2.000000
v 2.000000 2.000000 2.000000
v 0.000000 2.000000 0.000000
v 0.000000 0.000000 0.000000
v 2.000000 0.000000 0.000000
v 2.000000 2.000000 0.000000
# 8 vertices

g front
usemtl red
f 1 2 3 4
g back
usemtl blue
f 8 7 6 5
g right
usemtl green
f 4 3 7 8
g top
usemtl gold
f 5 1 4 8
g left
usemtl orange
f 5 6 2 1
g bottom
usemtl purple
f 2 6 7 3
# 6 elements
```

4. Texture-mapped square

This example describes a 2 x 2 square. It is mapped with a 1 x 1 square texture. The texture is stretched to fit the square exactly.

```
mtllib master.mtl

v 0.000000 2.000000 0.000000
v 0.000000 0.000000 0.000000
v 2.000000 0.000000 0.000000
v 2.000000 2.000000 0.000000
vt 0.000000 1.000000 0.000000
vt 0.000000 0.000000 0.000000
vt 1.000000 0.000000 0.000000
vt 1.000000 1.000000 0.000000
# 4 vertices

usemtl wood
f 1/1 2/2 3/3 4/4
# 1 element
```

5. Approximation technique for a surface

This example shows a B-spline surface which will be approximated using curvature-dependent subdivision specified by the *stech* command.

```
g bspatch
v -5.000000 -5.000000 -7.808327
v -5.000000 -1.666667 -7.808327
v -5.000000 1.666667 -7.808327
v -5.000000 5.000000 -7.808327
v -1.666667 -5.000000 -7.808327
v -1.666667 -1.666667 11.977780
v -1.666667 1.666667 11.977780
v -1.666667 5.000000 -7.808327
v 1.666667 -5.000000 -7.808327
v 1.666667 -1.666667 11.977780
v 1.666667 1.666667 11.977780
v 1.666667 5.000000 -7.808327
v 5.000000 -5.000000 -7.808327
v 5.000000 -1.666667 -7.808327
v 5.000000 1.666667 -7.808327
v 5.000000 5.000000 -7.808327
# 16 vertices

g bspatch
cstype bspline
stech curv 0.5 10.000000
deg 3 3
surf 0.000000 1.000000 0.000000 1.000000 13 14 \
15 16 9 10 11 12 5 6 7 8 1 2 3 4
parm u -3.000000 -2.000000 -1.000000 0.000000  \
1.000000 2.000000 3.000000 4.000000
parm v -3.000000 -2.000000 -1.000000 0.000000  \
1.000000 2.000000 3.000000 4.000000
end
# 1 element
```

6. Approximation technique for a curve

This example shows a Bezier curve which will be approximated using constant parametric subdivision specified by the *ctech* command.

```
v -2.300000 1.950000 0.000000
v -2.200000 0.790000 0.000000
v -2.340000 -1.510000 0.000000
v -1.530000 -1.490000 0.000000
v -0.720000 -1.470000 0.000000
v -0.780000 0.230000 0.000000
v 0.070000 0.250000 0.000000
v 0.920000 0.270000 0.000000
v 0.800000 -1.610000 0.000000
v 1.620000 -1.590000 0.000000
v 2.440000 -1.570000 0.000000
v 2.690000 0.670000 0.000000
v 2.900000 1.980000 0.000000
# 13 vertices

g default
cstype bezier
ctech cparm 1.000000
deg 3
curv 0.000000 4.000000 1 2 3 4 5 6 7 8 9 10 \
11 12 13
parm u 0.000000 1.000000 2.000000 3.000000  \
4.000000
end
# 1 element
```

Comments

Comments can appear anywhere in an .obj file. They are used to annotate the file; they are not processed.

Here is an example:

```
# this is a comment
```

The Model program automatically inserts comments when it creates .obj files. For example, it reports the number of geometric vertices, texture vertices, and vertex normals in a file.

```
# 4 vertices
# 4 texture vertices
# 4 normals
```

Mathematics for free-form curves/surfaces

General forms

Rational and non-rational curves and surfaces

In general, any non-rational curve segment may be written as:

$$\mathrm{C}(t) = \sum_{i=0}^{K} \mathbf{d}_i N_{i,n}(t)$$

where

$K + 1$	is the number of control points
$\mathbf{d}_i$	are the control points
n	is the degree of the curve
$N_{i,n}(t)$	are the degree n basis functions

Extending this to the bivariate case, any non-rational surface patch may be written as:

$$S(u, v) = \sum_{i=0}^{K_1} \sum_{j=0}^{K_2} \mathbf{d}_{i,j} N_{i,m}(u) N_{j,n}(v)$$

where:

$K_1 + 1$	is the number of control points in the u direction
$K_2 + 1$	is the number of control points in the v direction
$\mathbf{d}_{i,j}$	are the control points
m	is the degree of the surface in the u direction
n	is the degree of the surface in the v direction
$N_{i,m}(u)$	are the degree m basis functions in the u direction
$N_{j,n}(v)$	are the degree n basis functions in the v direction

NOTE: The *front* of the surface is defined as the side where the u parameter increases to the right and the v parameter increases upward.

We may extend this curve to the rational case as:

$$C(t) = \frac{\sum_{i=0}^{K} w_i \mathbf{d}_i N_{i,n}(t)}{\sum_{i=0}^{K} w_i N_{i,n}(t)}$$

where w_i are the weights associated with the control points $\mathbf{d}_i$. Similarly, a rational surface may be expressed as:

$$S(u,v) = \frac{\sum_{i=0}^{K_1} \sum_{j=0}^{K_2} w_{i,j} \mathbf{d}_{i,j} N_{i,m}(u) N_{j,n}(v)}{\sum_{i=0}^{K_1} \sum_{j=0}^{K_2} w_{i,j} N_{i,m}(u) N_{j,n}(v)}$$

where $w_{i,j}$ are the weights associated with the control points $\mathbf{d}_{i,j}$.

NOTE: If a curve or surface in an .obj file is rational, it must use the *rat* option with the *cstype* statement and it requires some weight values for each control point.

The weights for the rational form are given as a third control point coordinate (for trimming curves) or fourth coordinate (for space curves and surfaces). These weights are optional and default to 1.0 if not given.

This default weight is only reasonable for curves and surfaces whose basis functions sum to 1.0, such as Bezier, Cardinal, and NURB. It does not make sense for Taylor and may or may not make sense for a representation given in basis-matrix form.

For all forms other than B-spline, the final curve or surface is constructed by piecing together the individual curve segments or surface patches. A global parameter space is then defined over the entire composite curve or surface using the *parameter vector* given with the *parm* statement.

The parameter vector for a curve is a list of *p* global parameter values $\{\tau_1, \ldots, \tau_p\}$. If $\tau_1 \le \tau < \tau_{\iota+1}$ is a point in global parameter space, then:

$$t = \frac{\tau - \tau_i}{\tau_{i+1} - \tau_i}$$

is the corresponding point in local parameter space for the *i*th polynomial segment. It is this *t* which is used when evaluating a given segment of the piecewise curve. For surfaces, this mapping from global to local parameter space is applied independently in both the *u* and *v* parametric directions.

B-splines require a *knot vector* rather than a parameter vector, although this is also given with the *parm* statement. Refer to the description of B-splines below.

The following discussion of each type is expressed in terms of the above definitions.

NOTE: The maximum degree for all curve and surface types is currently set at 20, which is high enough for most purposes.

Free-form curve and surface types

B-spline

Type *bspline* specifies arbitrary degree non-uniform B-splines which are commonly referred to as NURBs in their rational form. The basis functions are defined by the Cox-deBoor recursion formulas as:

$$N_{i,0}(t) = \begin{cases} 1 & \text{if} \quad x_i \le t < x_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

and:

$$N_{i,n}(t) = (t - x_i)\frac{N_{i,n-1}(t)}{x_{i+n} - x_i} + (x_{i+n+1} - t)\frac{N_{i+1,n-1}(t)}{x_{i+n+1} - x_{i+1}}$$

where, by convention, $0/0 = 0$.

The $x_i \in \{x_{0,\ldots,}x_q\}$ form a set known as the knot vector which is given by the *parm* statement. It is required that

1. $x_i \le x_{i+1}$,
2. $x_0 < x_{n+1}$,
3. $x_{q-n-1} < x_q$,
4. $x_i < x_{i+n}$ for $0 < i < q - n - 1$,
5. $x_n \le t_{min} < t_{max} \le x_{K+1}$, where $[t_{min}, t_{max}]$ is the parameter over which the B-spline is to be evaluated, and
6. $K = q - n - 1$.

A knot is said to be of multiplicity r if its value is repeated r times in the knot vector. The second through fourth conditions above restrict knots to be of at most multiplicity $n + 1$ at the ends of the vector and at most n everywhere else.

The last condition requires that the number of control points is equal to one less than the number of knots minus the degree. For surfaces, all of the above conditions apply independently for the u and v parametric directions.

Bezier

Type *bezier* specifies arbitrary degree Bezier curves and surfaces. This basis function is defined as:

$$N_{i,n}(t) = \binom{n}{i} t^i (1-t)^{n-i}$$

where:

$$\binom{n}{i} = \frac{n!}{i!\,(n-1)!}$$

When using type *bezier*, the number of global parameter values given with the *parm* statement must be $K/n + 1$, where K is the number of control points. For surfaces, this requirement applies independently for the u and v parametric directions.

Cardinal

Type *cardinal* specifies a cubic, first derivative, continuous curve or surface. For curves, this interpolates all but the first and last control points. For surfaces, all but the first and last row and column of control points are interpolated.

Cardinal splines, also known as Catmull-Rom splines, are best understood by considering the conversion from Cardinal to Bezier control points for a single curve segment:

$$\begin{aligned}
\mathbf{b}_i &= \mathbf{c}_{i+1} \\
\mathbf{b}_{i+1} &= \mathbf{c}_{i+1} + \frac{1}{6}(\mathbf{c}_{i+2} - \mathbf{c}_i) \\
\mathbf{b}_{i+2} &= \mathbf{c}_{i+2} + \frac{1}{6}(\mathbf{c}_{i+1} - \mathbf{c}_{i+3}) \\
\mathbf{b}_{i+3} &= \mathbf{c}_{i+2}
\end{aligned}$$

Here, the $\mathbf{c}_\mathrm{i}$ variables are the Cardinal control points and the $\mathbf{b}_\mathrm{i}$ variables are the Bezier control points. We see that the second and third Cardinal points are the beginning and ending points for the segment, respectively. Also, the beginning tangent lies along the vector from the first to the third point, and the ending tangent along the vector from the second to the last point.

If we let $B_i(t)$ be the cubic Bezier basis functions (i.e. what was given above for Bezier as $N_{i,n}(t)$ with $n = 3$), then we may write the Cardinal basis functions as:

$$N_{0,3}(t) = -\frac{1}{6}B_1(t)$$

$$N_{1,3}(t) = B_0(t) + B_1(t) + \frac{1}{6}B_2(t)$$

$$N_{2,3}(t) = \frac{1}{6}B_1(t) + B_2(t) + B_3(t)$$

$$N_{3,3}(t) = -\frac{1}{6}B_2(t)$$

Note that Cardinal splines are only defined for the cubic case.

When using type *cardinal*, the number of global parameter values given with the *parm* statement must be $K - n + 2$, where K is the number of control points. For surfaces, this requirement applies independently for the u and v parametric directions.

Taylor

Type *taylor* specifies arbitrary degree Taylor polynomial curves and surfaces. The basis function is simply:

$$N_{i,n}(t) = t^i$$

NOTE: The control points in this case are the polynomial coefficients and have no obvious geometric significance.

When using type *taylor*, the number of global parameter values given with the *parm* statement must be $(K + 1)/(n + 1) + 1$, where K is the number of control points. For surfaces, this requirement applies independently for the u and v parametric directions.

Basis matrix

Type *bmatrix* specifies general, arbitrary-degree curves defined through the use of a basis matrix rather than an explicit type such as Bezier. The basis functions are defined as:

$$N_{i,n}(t) = \sum_{j=0}^{n} b_{i,j} t^{j}$$

where the basis matrix is the $b_{i,j}$. In order to make the matrix nature of this more obvious, we may also write:

$$\begin{bmatrix} N_{0,n}(t) \\ \\ N_{n,n}(t) \end{bmatrix} = \begin{bmatrix} b_{0,0} & \cdots & b_{0,n} \\ \\ b_{n,0} & \cdots & b_{n,n} \end{bmatrix} \begin{bmatrix} t^{0} \\ \\ t^{n} \end{bmatrix}$$

When constructing basis matrices, you should keep this definition in mind, as different authors write this in different ways. A more common matrix representation is:

$$\begin{bmatrix} N_{0,n}(t) \\ \\ N_{n,n}(t) \end{bmatrix} = \begin{bmatrix} b_{0,0} & \cdots & b_{n,0} \\ \\ b_{0,n} & \cdots & b_{n,n} \end{bmatrix} \begin{bmatrix} t^{n} \\ \\ t^{0} \end{bmatrix}$$

To use such matrices in the .obj file, simply transpose the matrix and reverse the column ordering.

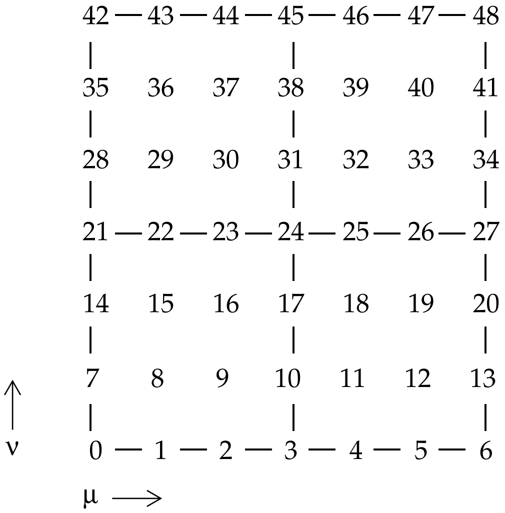
When using type *basis*, the number of global parameter values given with the *parm* statement must be $(K - n)/s + 2$, where K is the number of control points and s is the step size given with the *step* statement. For surfaces, this requirement applies independently for the u and v parametric directions.

Surface vertex data

Control points

The control points for a surface consisting of a single patch are listed in the order *i = 0* to K_1 for j = 0, followed by i = 0 to K_1 for j = 1, and so on until j = K_2.

For surfaces made up of many patches, which is the usual case, the control points are ordered as if the surface were a single large patch. For example, the control points for a bicubic Bezier surface consisting of four patches would be arranged as follows:



where (μ, ν) is the global parameter space of the surface and the numbers indicate the ordering of the vertex indices in the *surf* statement.

Texture vertices and texture mapping

When texture vertices are not supplied, the original surface parameterization is used for texture mapping. However, if texture vertices are supplied, they are interpreted as additional information to be interpolated or approximated separately from, but using the same interpolation functions as the control vertices.

That is, whereas the surface itself, in the non-rational case, was given in the section "Rational and non-rational curves and surfaces" as:

$$S(u, v) = \sum_{i=0}^{K_1} \sum_{j=0}^{K_2} \mathbf{d}_{i,j} N_{i,m}(u) N_{j,n}(v)$$

the texture vertices are interpolated or approximated by:

$$T(u, v) = \sum_{i=0}^{K_1} \sum_{j=0}^{K_2} \mathbf{t}_{i,j} N_{i,m}(u) N_{j,n}(v)$$

where $\mathbf{t}_{i,j}$ are the texture vertices and the basis functions are the same as for $S(u,v)$. It is $T(u,v)$, rather than the surface parameterization (u,v), which is used when a texture map is applied.

Vertex normals and normal mapping

Vertex normals are treated exactly like texture vertices. When vertex normals are not supplied, the true surface normals are used. If vertex normals are supplied, they are calculated as:

$$Q(u,v) = \sum_{i=0}^{K_1} \sum_{j=0}^{K_2} \mathbf{q}_{i,j} N_{i,m}(u) N_{j,n}(v)$$

where $\mathbf{q}_{i,j}$ are the vertex normals and the basis functions are the same as for S(*u*,*v*) and T(*u*,*v*).

NOTE: Vertex normals do not affect the shape of the surface; they are simply associated with the triangle vertices in the final triangulation. As with faces, supplying vertex normals only affects lighting calculations for the surface.

The treatment of both texture vertices and vertex normals in the case of rational surfaces is identical. It is important to notice that even when the surface *S(u,v)* is rational, the texture and normal surfaces, *T(u,v)* and *Q(u,v)*, are **not** rational. This is because the control points (the texture vertices and vertex normals) are never rational.

Curve and surface operations

Special points

The following equations give a more precise description of special points for space curves and discuss the extension to trimming curves and surfaces.

Let $C(\tau)$ be a space curve with the global parameter τ. We can approximate this curve by a set of *k-1* line segments which connect the points:

$$\{\mathbf{x}_1, \ldots, \mathbf{x}_k\} = \{C(\tau_1), \ldots, C(\tau_k)\}$$

for some set of k global parameter values $\{\tau_1,\ldots,\tau_k\}$

Given a special point τ_σ in the parameter space of the curve (referenced by *vp*), we guarantee that $\tau_\sigma \in \{\tau_1, \ldots, \tau_k\}$. More specifically, we approximate the curve by:

$$\{C(\tau_1), \ldots, C(\tau_\sigma), \ldots, C(\tau_k)\}$$

where, at the point i where τ_σ is inserted, we have $\tau_i \le \tau_\sigma < \tau_{i+1}$.

Special curves

The following equations give a more precise description of a special curve.

Let $T(\tau)$ be a special curve with the global parameter τ. We have:

$$(\mu, \nu) = T(\tau)$$

where (μ,ν) is a point in the global parameter space of a surface. We can approximate this curve by a set of k-1 line segments which connect the points:

$$\{T(\tau_1), \ldots, T(\tau_k)\}$$

for some set of k global parameter values.

Let $S(\mu,\nu)$ be a surface with the global parameters μ and ν. We can approximate this surface by a triangulation of a set of p points.

$$\{S(\mu_1 \nu_1), \ldots, S(\mu_p, \mu_p)\}$$

which lie on the surface. We further define E as the set of all edges such that $e_{i,j} \in E$ implies that $S(\mu_i,\nu_i)$ and $S(\mu_j,\nu_j)$ are connected in the triangulation. Finally, we guarantee that there exists some subset of *E:*

$$\{e_{1,2}, e_{2,3}, \ldots, e_{k-1,k}\}$$

such that the points:

$$\{S(T(\tau_1)), \ldots, S(T(\tau_k))\}$$

are connected in the triangulation.

Connectivity

Recall that the syntax of the *con* statement is:

```
con surf_1 q0_1 q1_1 curv2d_1 surf_2 q0_2 q1_2 curv2d_2
```

If we let:

$T_1(\tau_1)$	be the curve referenced by *curv2d_1*
$S_1(\mu_1, \nu_1)$	be the surface referenced by *surf1* on which $T_1(\tau_1)$ lies
$T_2(\tau_2)$	be the curve referenced by *curv2d_2*
$S_2(\mu_2, \nu_2)$	be the surface referenced by *surf2* on which $T_2(\tau_2)$ lies

then $S_1(T_1(\tau_1))$, $S_2(T_2(\tau_2))$ must be identical up to reparameterization. Moreover, it must be the case that:

$$S_1(T_1(q0\_1)) = S_2(T_2(q0\_2))$$

and:

$$S_1(T_1(q1\_1)) = S_2(T_2(q1\_2))$$

It is along the curve $S_1(T_1(\tau_1))$ between $\tau_1 = q0\_1$ and $\tau_1 = q1\_1$, and the curve $S_2(T_2(\tau_2))$ between $\tau_2 = q0\_2$ and $\tau_2 = q1\_2$ that the surface $S_1(\mu_1, \nu_1)$ is connected to the surface $S_2(\mu_2, \nu_2)$.

Superseded statements

The new .obj file format has eliminated the need for several patch and curve statements. These statements have been replaced by free-form geometry statements.

In the 3.0 release, the following keywords have been superseded:

- bsp
- bzp
- cdc
- cdp
- res

You can still read these statements in this version 3.0, however, the system will no longer write files in this format.

This release is the last release that will read these statements. If you want to save any data from this format, read in the file and write it out. The system will convert the data to the new .obj format.

For more information on the new syntax statements, see "Specifying free-form curves and surfaces."

Syntax

The following syntax statements are for the superseded keywords.

bsp *v1 v2 . . . v16*

Specifies a B-spline patch. B-spline patches have sixteen control points, defined as vertices. Only four of the control points are distributed over the surface of the patch; the remainder are distributed around the perimeter of the patch.

Patches must be tessellated in Model before they can be correctly shaded or rendered.

v is the vertex number for a control point. Sixteen vertex numbers are required. Positive values indicate absolute vertex numbers. Negative values indicate relative vertex numbers.

bzp *v1 v2 . . . v16*

Specifies a Bezier patch. Bezier patches have sixteen control points, defined as vertices. The control points are distributed uniformly over its surface.

Patches must be tessellated in Model before they can be correctly shaded or rendered.

v is the vertex number for a control point. Sixteen vertex numbers are required. Positive values indicate absolute vertex numbers. Negative values indicate relative vertex numbers.

cdc *v1 v2 v3 v4* *v5 . . .*

Specifies a Cardinal curve. Cardinal curves have a minimum of four control points, defined as vertices.

Cardinal curves cannot be correctly shaded or rendered. They can be tessellated and then extruded in Model to create 3D shapes.

v is the vertex number for a control point. A minimum of four vertex numbers are required. There is no limit on the maximum.