Chapter 1.
Faceter Component

Component: *Faceting

The Faceter Component (FCT), in the fct directory, generates and controls approximate polygonal representations. The polygon approximations are used in all active and static rendering, in clearance analysis, and in operations where approximations are acceptable to simplify calculations. FCT supplies refinements to the polygons which specify how they should be generated.

The ACIS faceter supports only view-independent faceting. Faceted representations are used in rendering, in clearance analysis, and in operations where an approximation is acceptable in order to simplify calculations.

Faceting generates approximate polygonal representations for the faces of entities while maintaining edge consistency between adjacent faces. Face faceting is performed by subdividing the face in parameter space with a grid whose increments are determined through *refinements*. Refinements specify the accuracy of the faceted representation. Refinements also control how triangulation is performed and whether smoothing is used to improve the aspect ratios of triangles. The faceted representation of a face is also called a *mesh*.

Faceting is used by all active and static renderers. Refinements for faceting have a significant effect on the rendering process. The tighter the facets, the smoother the rendered surface will be. However, the tighter the facets, the slower the rendering process becomes. On the other hand, the looser the facets, the faster the rendering process becomes, but the rendered surface will be rougher.

Entities and their refinement attributes are stored in the save file. Because facets can be regenerated from the entities and their refinements, facet information is not stored in the save file. However, facets can be output through *mesh managers* to an application-determined data structure, which the application can manage as it sees fit.

Step–by–Step Faceting

Topic: *Faceting

Rendering is one of the primary down-stream operations that requires facets. Faceting is used by all active and static renderers. In both the active and static renderers, the entity facets are what are rendered.

Information on active rendering can be found in the *OpenGL Rendering Component Manual* Information on static rendering can be found in the *Basic Rendering Component Manual*.

In the case of the active renderers, each time an entity is added to the rendering context, the faceter automatically facets the entity.

In the case of a static renderer, the faceter does not automatically facet the entity. It expects that each entity in their list already has the appropriate INDEXED_MESH. Therefore, at the beginning of an application, a call must be made to api_initialize_faceter. Later in the application, when the list of entities is available, a call is made to one of the facet APIs: api_facet_entities, api_facet_entity, api_facet_unfaceted_entities, or api_facet_unfaceted_entity.

The following lists the steps that an application could perform to facet and render a model using a static renderer.

1. When an application uses faceting, it must first determine whether or not the facet data will be saved. If saving the data is required, the application derives its own mesh manager which handles saving the facet data.

2. Because rendering is desired and this example does not save any data, create an instance of the INDEX_MESH_MANAGER.

3. Create or retrieve an ACIS model. This could be performed sooner or later in the process, but the model, obviously, must be available before attempting to facet its unfaceted entities.

4. Initialize the renderer that the application is using. Refer to the renderer's component manual. Each active renderer has slightly different requirements.

5. The mesh manager may be specified when faceting is initialized using the function api_initialize_faceter. It may also be set after initialization with the function api_set_mesh_manager. This is how the faceter is informed of which mesh manager is being used. For this example, call the api_set_mesh_manager function with a pointer to the mesh manager (e.g., INDEX_MESH_MANAGER).

6. After faceting is initialized and a mesh manager specified, the application must specify some refinements. Environment default refinements are set using the function api_set_default_refinement, while individual refinements are set using the function api_set_entity_refinement.

Note *Refinements attached to a face take precedence over the refinements attached to their owning body.*

7. If Graphic Interaction is used, call the api_initialize_graphic_interaction function to initialize the graphic interaction library.

8. If Part Management is used, call the api_initialize_part_manager function to initialize the part manager library.
9. Call the api_rh_initialise function to initialize the data structures required by the renderers. This must be called before any other rendering function.
10. Optionally, call the api_rh_initialize_supl_shaders function to initialize a larger set of shaders. The shaders supplied with the Core Shader Library have already been initialized prior to this step.
11. Call the api_prepare_to_render function. This tells the renderer the specific view3d that is to be overwritten with the rendered image. The view3d that is used for rendering can be the same view3d used by another rendering context, such as the display_list_context; the view3d is overwritten with the rendering data.
12. The application can send an entity or a list of entities to be faceted, using either function api_facet_entity or api_facet_entities, respectively. Each entity may be a body, a lump, a shell, or a face. Alternatively, one can choose to facet only those faces that have changed since they were last faceted, using the functions api_facet_unfaceted_entity or api_facet_unfaceted_entities. This is called *incremental faceting*. In this example, call the api_facet_unfaceted_entities function with a pointer to the entity list. The entity list comes from the ACIS model that was retrieved or created.
13. Call the api_rh_render_entities function to see the results of faceting. For the static renderers, this bypasses the normal viewing pipeline by rendering the supplied list of entities directly in a view3d. The api_rh_render_entities uses the rendering mode established by api_rh_set_mode.
14. At the end of an application or rendering section, the faceter should be turned off with a call to api_terminate_faceter.

Figure 1-1 shows the data flow during a faceting session:

- A REFINEMENT is created and attached to an ACIS entity using ATTRIB_EYE_REF_VT.
- The entity and the refinement are passed to the faceter.
- The faceter creates an instance of the MESH_MANAGER.
- The mesh manager creates a mesh for each face of the entity.
- If the mesh manager is an application-defined class, the application can store mesh information into one of its data structures.
- The face entities and their meshes are coupled using ATTRIB_EYE_ATTACHED_MESH.
- The mesh manager instance goes away after faceting the face entities.
- The generated mesh is available to down-stream operations, such as rendering. Rendering has a list of entities. The entities are aware of their attributes, one of which tells about the facet mesh.

- If the associated entities are deleted or the application terminates, the mesh goes away.
- The entities and their refinements can be saved and stored in an ACIS save file.

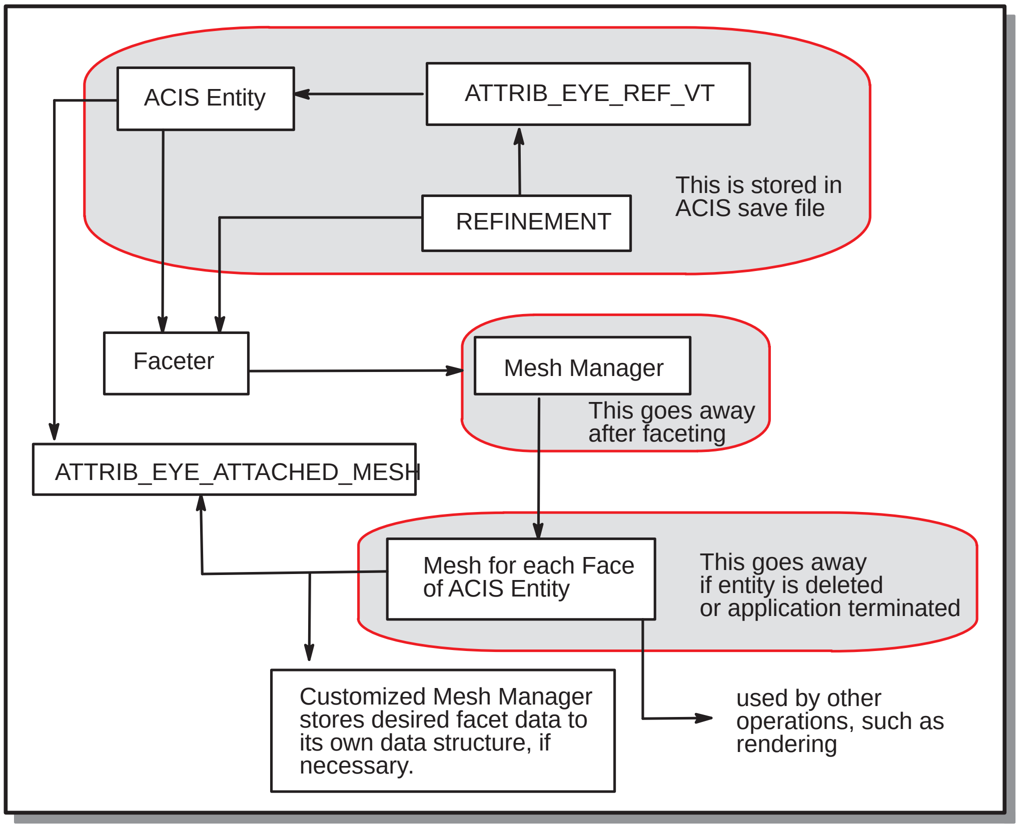

ACIS Entity
ATTRIB_EYE_REF_VT
REFINEMENT
This is stored in ACIS save file
Faceter
Mesh Manager
This goes away after faceting
ATTRIB_EYE_ATTACHED_MESH
Mesh for each Face of ACIS Entity
This goes away if entity is deleted or application terminated
Customized Mesh Manager stores desired facet data to its own data structure, if necessary.
used by other operations, such as rendering

Figure 1-1. Data flow among the application, faceter, and ACIS modeler

The Faceting Algorithm

Topic: *Faceting

When the api_facet_unfaceted_entities function is called with a pointer to the entity list, what does the faceter do? The process is divided into four general phases: *grid spacing determination*, *edge discretization*, *face subdivision*, and *triangulation*. These general phases are implemented in the faceting steps described in this section. The following assumptions are made in the description of the faceting algorithm steps:

- The faceter has been initialized.

- Default refinements are set.
- There is an appropriate mesh manager (e.g., INDEXED_MESH_MANAGER) for facet output.
- A list of working faces has been created for faceting out of an ACIS solid body.
- Refinements have been attached to the ACIS face, shell, lump, body, or default refinement.

Note *Refinements attached to a face take precedence over the refinements attached to their owning body. A common difficulty is sometimes encountered when changing a body's refinement may have no effect on one or more faces; those faces probably have refinements of their own that either need to be updated or removed.*

The faceter carries out the following operations:

1. A working face is created for each ACIS face.

2. A working edge is created for each ACIS edge. The working edge class contains the normal deviation, surface tolerance, and maximum edge length refinements. A working edge gets the lower of the two values from two adjacent faces.

3. Each working edge is discretized, or subdivided, using the three refinements in step 2. A list of points for each working edge is created. Spline edges are recursively subdivided until all the refinements are satisfied. When there are adjacent edges, the last point from the previous edge is the first point on the working edge. Likewise, the first point of the next edge is the last point on the working edge. In the *edge discretization* phase, the edges are discretized to be about the same as the smallest grid spacing among connected faces. Thus, it is crucial that the application provide all the faces that are to be faceted in one step (using one of the functions api_facet_unfaceted_entity or api_facet_unfaceted_entities).

4. In the *face subdivision* phase, a grid in parameter space is laid on each working face. A *uv* node set structure is created for each working face. The points from the point list of the working edges are added to this structure. Then, the initial *uv* node loops on each face are created.

   On planar faces and cylindrical faces, the cells on the grid are rectangular in the object space. For spherical, conical, and toroidal faces, the cells are not rectangular in the object space, but they are planar. For spline faces, the cells are not planar, but the grid is laid in an adaptive fashion that puts more cells in areas of higher curvature than in areas of lower curvature (*adaptive faceting*).

   When the grid is laid on the face, the grid lines in general do not intersect the edges at the discretization points. A refinement can be used to specify whether these intersection points will be inserted into the edge discretization. If they are not inserted, faceting will place triangles between the edge and the interior cells, regardless of the refinement.

5. If the grid mode is set to either AF_GRID_INTERIOR or AF_GRID_TO_EDGES, then the faceter determines the number of *uv* parametric lines to be inserted for each working face based on the normal tolerance, surface tolerance, the maximum facet edge length, and the maximum number of grid lines. The *uv* parametric lines are equally spaced for analytical surfaces and are variably space for spline surfaces. In the *grid spacing determination* phase, faceting determines the appropriate grid to lay on each face according to the refinements as specified.

6. If the grid mode is set to AF_GRID_TO_EDGES, the grid intersects the edges and creates facet nodes. This is accomplished by inserting the point list from the working edge into the face's *uv* point set.

7. If a facet edge lying on a face's edge is wider than the *uv* spacing, the facet edges are subdivided accordingly. This adds more points to the *uv* points.

8. If the grid mode is set to AF_GRID_TO_EDGES, the boundary points from the face's *uv* node set are inserted into the corresponding working edge point list.

9. The points from the adjacent faces are verified to be in the *uv* node set of the face. If not, and if the grid mode is set to AF_GRID_TO_EDGES, the point list from the working edge is inserted into the face's *uv* point set as in step 6.

10. The mesh for each face is created using the *uv* node set collected thus far and the *triangulation* phase.

    a. If the face has holes, bridge edges are added between exterior and interior edges.

    b. If the grid mode is set to either AF_GRID_INTERIOR or AF_GRID_TO_EDGES, then the bridge edges are split based on the *uv* grid. In addition, edges are inserted between existing *uv* nodes on opposing edges and are split by the *uv* grid.

    c. If triangulation mode is not AF_TRIANG_NONE, then the face nodes are triangulated up to the number of fringes specified in the mode (e.g., AF_TRIANG_FRINGE_1, AF_TRIANG_FRINGE_2, etc.). If the mode is AF_TRIANG_ALL, then the whole grid is triangulated. Triangle smoothing is eliminated with no loss in quality. An option permits smoothing the facets if desired. The adaptive mesh algorithm subdivides when no grid mode is used and AF_TRIANG_ALL is used.

    d. If triangulation mode is not AF_TRIANG_NONE and if the grid mode is not AF_GRID_TO_EDGES, then there are some triangles that need to be subdivided based on the refinement controls given by the surface tolerance, the normal tolerance, and the maximum edge length tolerance. The midpoint of a facet edge is now used when testing for compliance to the normal tolerance.

11. The resulting face nodes and polygons for each face are output through the appropriate mesh manager to a derived MESH class. For example, the INDEXED_MESH_MANAGER creates an INDEXED_MESH, while the POLYGON_POINT_MESH_MANAGER creates a POLYGON_POINT_MESH.

12. If the application derived its own mesh manager for the purposes of saving the facet data in its own data structure, that would be carried out now. The derived class would redefine the inherited methods and enhance them to carry out the special needs of the application. For example, a class derived from the INDEXED_MESH_MANAGER would need announce_counts, announce_indexed_node, and announce_indexed_polynode methods. In addition, this is where information from the VERTEX_TEMPLATE class would be filled out for each node.

13. A mesh (e.g., INDEXED_MESH instance) is attached to each face using ATTRIB_EYE_ATTACHED_MESH. ATTRIB_EYE_ATTACHED_MESH takes an entity (face) pointer and a mesh pointer.

14. At the very end of faceting, the mesh manager is freed.

Mesh Managers

Topic: *Faceting

The way facets are stored always depends on the particular application. A data format designed to meet the needs of all end-user applications would make the data storage requirements for faceting huge, if not unmanageable. Applications rarely use everything output by the faceter, anyway. If provided a huge data structure, applications would typically extract what they need and discard the original data structure to save on storage. Saving storage space might come at the expense of computation, because down-stream operations might need the original data structure and would be forced to regenerate (re-facet) the model.

The solution used by ACIS was not to store any facet data directly. Instead, ACIS provides *mesh managers* with simple virtual functions that can output the facet data. Deriving application specific mesh managers permits an application to redefine the functions, or class methods. In doing so, an application has the ability to store the information it requires directly into its own structures. An application can have multiple mesh managers that generate facet data for various purposes. For example, one mesh manager may be used for display and another for generating stereolithography files.

A mesh manager is simply a C++ class that is derived from the base class MESH_MANAGER. For a complete list of MESH_MANAGER methods, refer to the documentation of the MESH_MANAGER class. The mesh manager's virtual functions are divided into three different protocols. Each protocol represents a group of functions tailored for a commonly used data storage format.

The base MESH_MANAGER class defines numerous virtual member functions stubs that are grouped into protocols. Derived classes can redefine these functions to switch on the desired protocol. When the faceter has computed a set of facets, it outputs them according to the protocol specified.

Coordinate Protocol	Outputs the polygon vertices as explicit coordinates.
Indexed Protocol	First outputs the list of coordinates of all the vertices in a face mesh, then outputs the polygon vertices as indices referring to items of this list.
Global Indexed Protocol	Is similar to the indexed protocol, but considers the entire body as a single mesh.

The MESH_MANAGER has a virtual method announce_polygon_model_face to announce the model face from which the polygons are made. Being a virtual function, it works with all mesh managers. The method announces the face once before the polygons are announced. All mesh managers evaluate the coordinates on a node only once regardless of the number of polygon edges that share that node.

The general flow of output is:

1. The counts.
2. The nodes.
3. The edges.
4. The polygons for each mesh. The polygons are output node-by-node.

ACIS provides two main mesh managers: INDEXED_MESH_MANAGER and POLYGON_POINT_MESH_MANAGER.

The INDEXED_MESH_MANAGER is used primarily for rendering. It announces (declares) the nodes by their *xyz* positions. The nodes are created by the faceter. The nodes are used to create polygons. Each face receives a mesh based on the polygons. The INDEXED_MESH created by this mesh manager places its data in contiguous arrays, thereby minimizing memory fragmentation. It is stored as an array of nodes and a set of polygons defined by numerical indices. In this “packed” representation, both nodes and polygon are referenced using an index into the array.

The POLYGON_POINT_MESH_MANAGER is used primarily by the obsolete Faceted Hidden Line Component. It announces its mesh polygon by polygon. The polygons are announced based on the *xyz* coordinates of their nodes created by the faceter. The POLYGON_POINT_MESH created by this mesh manager places its data in a singly-linked, NULL terminated list. The mesh maintains pointers to the first and last polygons as well as the number of polygons in the mesh. The Polygon Point Mesh Manager calculates the node information once and stores it for future use.

There is a third mesh manager, called a GLOBAL_INDEX_MESH_MANAGER . It declares a single mesh for the entire body instead of a mesh for each face. It has no graphics or permanent storage services. The mesh is written to the standard output.

The mesh manager can be specified when faceting is initialized with the function api_facet_initialise. Or, after initialization, the mesh manager can be specified with the function api_set_mesh_manager.

Refinements

Topic: *Faceting

A *refinement* controls how the facets approximate the solid model and how accurately. A refinement is a C++ class that encapsulates all refining control parameters along with methods to read and modify the parameters. For a complete list of these parameters and their associated methods, refer to the documentation of the class REFINEMENT. Some parameters can take any numeric value (e.g., a normal deviation), while others can only take one value of a limited set (e.g., surface type and grid handling).

A refinement attached to a face applies only to that face. A refinement attached to a shell, a lump, or a body may apply to the contained faces. Based on the surface type of a face, the refinement is first searched from the face, then the shell, then the lump, then the body, then finally from the faceting defaults.

After faceting has determined the applicable refinement for a face, shell, lump, or body during the search, the application is allowed to change the refinement for a face through a method (member function) of the mesh manager. This implies that the application can use some context-dependent refinements without going through all the refinements to modify them explicitly.

In general, faceting tries to satisfy all refinement requirements set by the refinement parameters. However, in a case of conflict, higher order parameters will be satisfied and lower order conflicting ones will be ignored. Refinement parameters are considered in the following high-to-low order:

1. Maximum number of grid lines
2. Maximum facet edge length
3. Normal tolerance
4. Surface tolerance
5. Grid aspect ratio

The many parameters in the refinement are necessary to provide flexible control. They are used in combinations for the desirable effect.

Default Refinement Values

Topic: *Faceting

Default refinement values are set when a refinement is created (e.g., when a REFINEMENT class is instantiated). Refer to the class reference template for REFINEMENT for default settings for these refinements:

Surface Tolerance	Surface Tolerance
Grid Aspect Ratio	Maximum Edge Length
Maximum Grid Lines	Minimum U Grid Lines
Minimum V Grid Lines	Grid Mode
Triangulation Mode	Adjust Mode
Surface Mode	

Refinement values can be queried and set in C++ using various methods of the REFINEMENT class. Refinement values can also be queried and set in Scheme using Scheme extensions (such as refinement:default and refinement:set–default).

Surface Tolerance

Topic: *Faceting

The surface tolerance is the distance between the facet and the part of the surface it is representing. By setting this refinement, the user specifies how accurately the facets represent the surface. It is desirable to provide surface tolerance when facets obtained with normal tolerance do not represent the surface well.

The set_surface_tol method from the REFINEMENT class specifies the maximum distance between a facet and the true surface. The proper value is dependent on the model size. When the surface tolerance is set to –1, the faceter will use a surface tolerance that is independent of the model size. This is a fraction (e.g., 1/1000th) of the diagonal of the bounding box of the face. This results in faceting that is independent of the units of measurement of the face.

Figure 1-2 illustrates two cases of surface tolerance. The refinement values are specified in Table 1-1. In the first case the surface tolerance is zero, meaning that it is not taken into account in faceting. In the second case the surface tolerance is 0.1, meaning that every facet is not more than 0.1 unit away from the mathematical spline surface. In the second case there are smaller but more facets in order to satisfy the surface tolerance constraint.

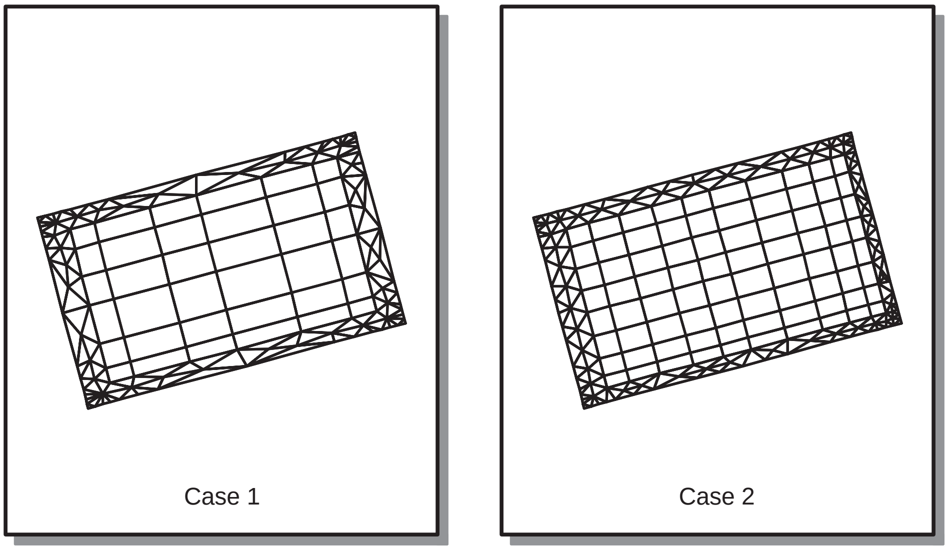


Figure 1-2. Surface Tolerance

Table 1-1. Surface Tolerance

Refinement	Case 1 Value	Case 2 Value
Surface Tolerance	0	0.1
Normal Tolerance	15	15
Grid Aspect Ratio	0	0
Maximum Edge Length	0	0
Maximum Grid Lines	10,000	300
Grid Mode	AF_GRID_INTERIOR	AF_GRID_INTERIOR
Triangulation Mode	AF_TRIANG_FRINGE_2	AF_TRIANG_FRINGE_2
Adjust Mode	AF_ADJUST_NON_GRID	AF_ADJUST_NON_GRID
Surface Mode	AF_SURF_ALL	AF_SURF_ALL

Normal Tolerance

Topic: *Faceting

The normal tolerance is the angle between the surface normals at the two adjacent nodes of a facet. By setting this *normal deviation* refinement, the user specifies how accurately the facets are representing the surface and the quality of rendering desired. It is desirable to use this refinement control because it is independent of the model size.

The set_normal_tol method from the REFINEMENT class specifies the maximum normal deviation allowed between two normals on a facet. The proper value is usually independent of the model size.

Figure 1-3 illustrates two cases of normal tolerance. The refinement values are specified in Table 1-2. In the first case the normal tolerance is 15 degrees. In the second case the normal tolerance is 25 degrees, which means that all the adjacent nodes' normals have at most a 25 degree angle between them. Because a higher normal tolerance is set, fewer facets are obtained in the second case.

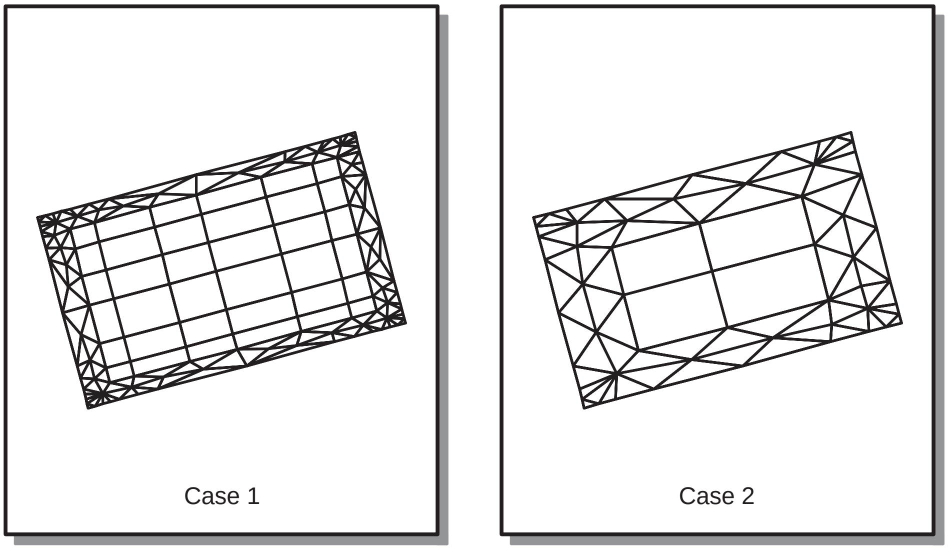


Figure 1-3. Normal Tolerance

Table 1-2. Normal Tolerance

Refinement	Case 1 Value	Case 2 Value
Surface Tolerance	0	0
Normal Tolerance	15	25
Grid Aspect Ratio	0	0

Refinement	Case 1 Value	Case 2 Value
Maximum Edge Length	0	0
Maximum Grid Lines	10,000	300
Grid Mode	AF_GRID_INTERIOR	AF_GRID_INTERIOR
Triangulation Mode	AF_TRIANG_FRINGE_2	AF_TRIANG_FRINGE_2
Adjust Mode	AF_ADJUST_NON_GRID	AF_ADJUST_NON_GRID
Surface Mode	AF_SURF_ALL	AF_SURF_ALL

Grid Aspect Ratio

Topic: *Faceting

The grid aspect ratio is the ratio of the *u* and *v* lengths of a *uv* grid cell. Thus, the aspect ratio is used only when the grid mode is something other than AF_GRID_NONE. The *u* and *v* lengths are in object space. The algorithm tries to make the aspect ratio equal to the one specified, but this is not guaranteed.

The grid aspect ratio is defined as the ratio of the geometric length of a grid in the *u* parameter direction to the geometric length of the grid in the *v* parameter direction. The faceter attempts to make grids having a ratio specified by the grid_aspect_ratio refinement parameter. For instance, setting the aspect ratio to 1.0 indicates that the grid should be as square as possible. The algorithm which tries to create grids having a specified aspect ratio has been modified to give better grids when the aspect ratio is 1.0.

The set_grid_aspect_ratio method from the REFINEMENT class specifies the approximate aspect ratio of each cell in the grid. If the value is close to 1, then the cell is close to a square. This does not guarantee the aspect ratio of the facet, which may consist of only a part of a cell.

Figure 1-4 illustrates two cases of grid aspect ratio. The refinement values are specified in Table 1-3.

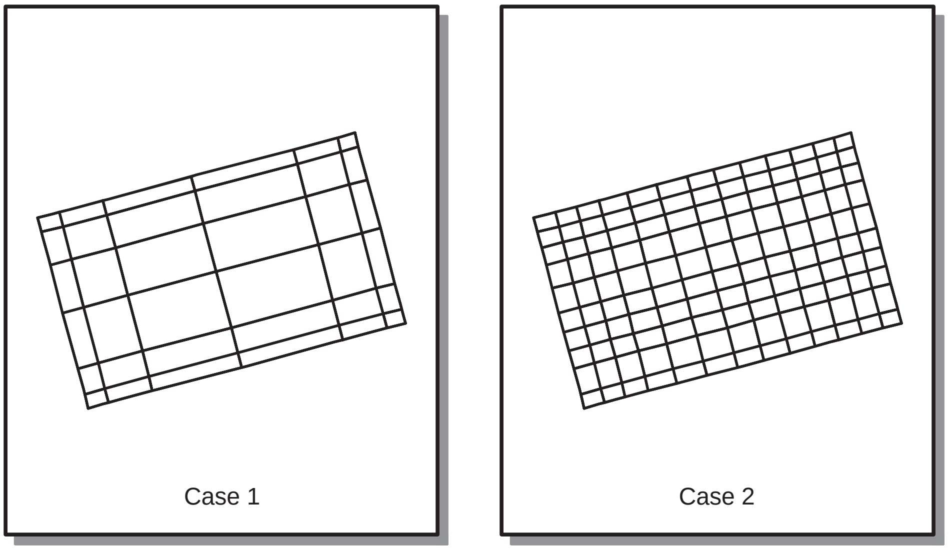


Figure 1-4. Grid Aspect Ratio

Table 1-3. Grid Aspect Ratio

Refinement	Case 1 Value	Case 2 Value
Surface Tolerance	0	0
Normal Tolerance	25	25
Grid Aspect Ratio	0	1
Maximum Edge Length	0	0
Maximum Grid Lines	10,000	300
Grid Mode	AF_GRID_TO_EDGES	AF_GRID_TO_EDGES
Triangulation Mode	AF_TRIANG_NONE	AF_TRIANG_NONE
Adjust Mode	AF_ADJUST_NON_GRID	AF_ADJUST_NON_GRID
Surface Mode	AF_SURF_ALL	AF_SURF_ALL

Maximum Edge Length

Topic: *Faceting

The edge length is the length of a facet edge. The maximum edge length value allows the facet edge length to be within the specified limit. This is the only way to subdivide facets further in planer faces.

The set_max_edge_length method from the REFINEMENT class specifies the maximum length of a side of a cell in object space. Since a facet cannot be larger than the cell, this determines the maximum size of the facet. If the value is zero, maximum edge length is not considered in faceting.

Figure 1-5 illustrates two cases of maximum edge length. The refinement values are specified in Table 1-4.

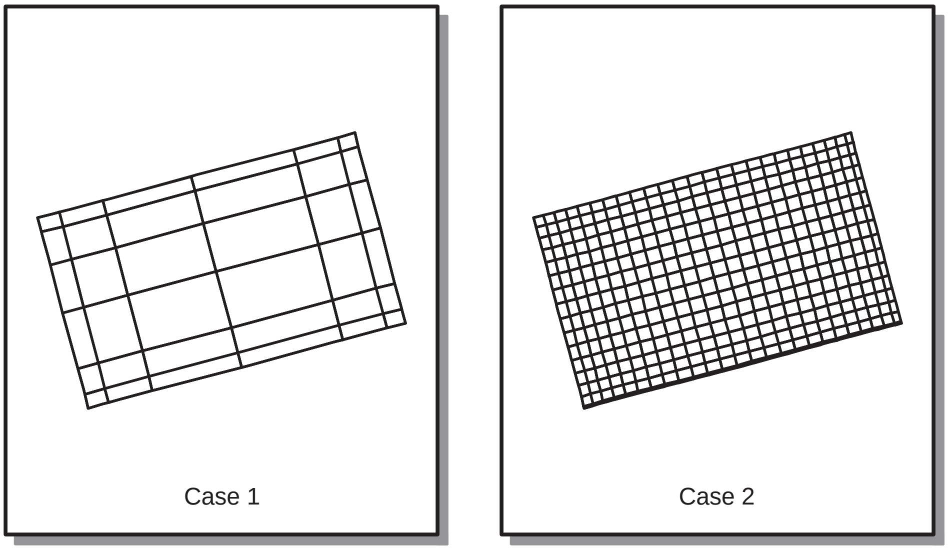


Figure 1-5. Maximum Edge Length

Table 1-4. Maximum Edge Length

Refinement	Case 1 Value	Case 2 Value
Surface Tolerance	0	0
Normal Tolerance	25	25
Grid Aspect Ratio	0	0
Maximum Edge Length	0	2
Maximum Grid Lines	10,000	300
Grid Mode	AF_GRID_TO_EDGES	AF_GRID_TO_EDGES
Triangulation Mode	AF_TRIANG_NONE	AF_TRIANG_NONE
Adjust Mode	AF_ADJUST_NON_GRID	AF_ADJUST_NON_GRID
Surface Mode	AF_SURF_ALL	AF_SURF_ALL

Maximum Grid Lines

Topic: *Faceting

The maximum number of grid lines can prevent the other constraints from being satisfied, so it is recommended to set it to a high value. This refinement has no meaning when grid mode is set to AF_GRID_NONE.

The set_max_grid_lines method from the REFINEMENT class specifies the maximum number of grid subdivisions. This prevents the facet data of a face from getting too big. It can also be used to specify the exact number of divisions on a face by using it in conjunction with another parameter; e.g., a very small normal deviation.

Figure 1-6 illustrates two cases of maximum grid lines. The refinement values are specified in Table 1-5. In the second case, although normal tolerance is set to 5 degrees, the maximum number of grid lines set to 20 does not allow normal tolerance to generate enough facets.

The maximum number of grid lines also effects the the edge split limit (the maximum number of times an edge may be split to meet faceting tolerances). If an edge does not meet tolerances, it is split in two, and if either half does not meet tolerance, the halves are split in two. This continues until tolerances are met, or until the edge split limit is met. The limit for splitting an edge is derived from the maximum number of grid lines. The derived value is sufficient to meet edge tolerances when the maximum grid lines number is sufficient for meeting tolerances along the surface. In some models, meeting edge tolerances may take excessive time and resources. In these cases this limit insures that excessive faceting will not drain system resources or make the system appear to hang.

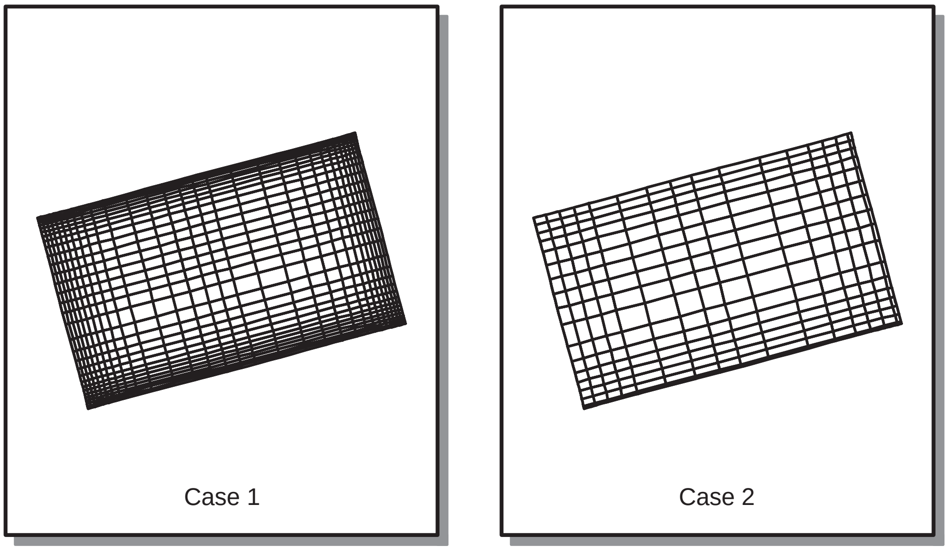


Figure 1-6. Maximum Grid Lines

Table 1-5. Maximum Grid Lines

Refinement	Case 1 Value	Case 2 Value
Surface Tolerance	0	0
Normal Tolerance	5	5
Grid Aspect Ratio	0	0
Maximum Edge Length	0	0
Maximum Grid Lines	10,000	20
Grid Mode	AF_GRID_TO_EDGES	AF_GRID_TO_EDGES
Triangulation Mode	AF_TRIANG_NONE	AF_TRIANG_NONE
Adjust Mode	AF_ADJUST_NON_GRID	AF_ADJUST_NON_GRID
Surface Mode	AF_SURF_ALL	AF_SURF_ALL

Grid Mode and Triangulation Mode

Topic: *Faceting

The grid mode determines whether a grid is used and whether the points where the grid cuts the edges should be inserted into the edge discretization. The grid mode uses the *uv* parameter grid of the surface for faceting. Method set_grid_mode of the REFINEMENT class is used to specify the grid mode.

The allowable values are:

AF_GRID_NONE	Does not subdivide face with a grid.
AF_GRID_TO_EDGES	Uses a grid and creates intersection points of the grid with the edges.
AF_GRID_INTERIOR	Uses grid only in the interior. Triangles are generated from the edge to the grid.

The triangulation mode specifies how much triangulation to perform. The set_triang_mode method of the REFINEMENT class is used to specify the mode. If AF_GRID_INTERIOR is specified, triangulation will be performed at least at the fringe cells.

The allowable values are:

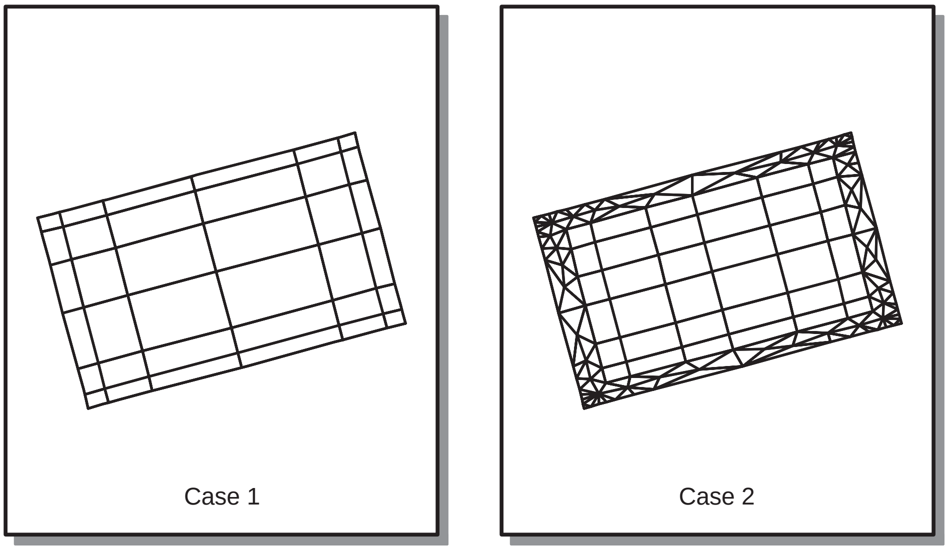
AF_TRIANG_NONE	No triangulation.
AF_TRIANG_FRINGE_1	Triangulate at the fringe layer.

AF_TRIANG_FRINGE_2	Triangulate two (2) fringe layers.
AF_TRIANG_FRINGE_3	Triangulate three (3) fringe layers.
AF_TRIANG_FRINGE_4	Triangulate four (4) fringe layers.
AF_TRIANG_ALL	Triangulate all facets.

The AF_GRID_INTERIOR mode generates triangles at least near the first fringe, even if the triangulation mode is AF_TRIANG_NONE. In this case, triangulation mode AF_TRIANG_NONE gives the same result as AF_TRIANG_FRINGE_1.

When triangulation mode is set to AF_TRIANG_FRINGE_2, two outermost peripheral grid facets are triangulated. Up to four fringe levels can be specified by the triangulation mode, or AF_TRIANG_ALL can be specified for all triangular facets.

Figure 1-7 illustrates three different cases of grid modes and triangulation modes. The refinement values are specified in Table 1-6.



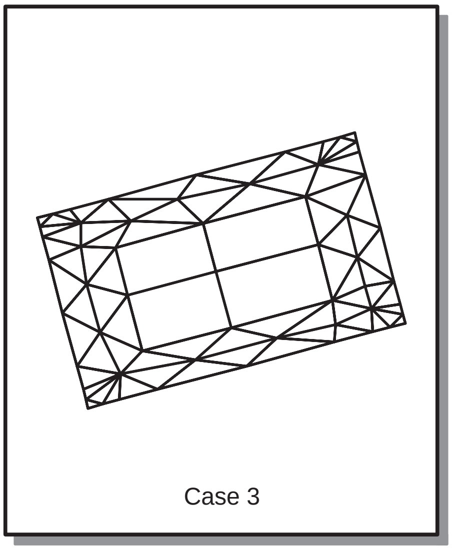


Figure 1-7. Grid and Triangulation Modes

Table 1-6. Grid and Triangulation Modes

Refinement	Case 1 Value	Case 2 Value	Case 3 Value
Surface Tolerance	0	0	0
Normal Tolerance	25	25	25
Grid Aspect Ratio	0	0	0
Maximum Edge Length	0	0	0
Maximum Grid Lines	10,000	300	300
Grid Mode	AF_GRID_ TO_EDGES	AF_GRID_ INTERIOR	AF_GRID_ INTERIOR
Triangulation Mode	AF_TRIANG_ NONE	AF_TRIANG_ NONE	AF_TRIANG_ FRINGE_2
Adjust Mode	AF_ADJUST_ NON_GRID	AF_ADJUST_ NON_GRID	AF_ADJUST_ NON_GRID
Surface Mode	AF_SURF_ALL	AF_SURF_ALL	AF_SURF_ALL

Adjust Mode

Topic: *Faceting

The adjust mode works when a *uv* grid is used along with triangulation of facets. It tries to adjust the facet node positions to smooth the triangles.

The set_adjust_mode method from the REFINEMENT class is used for triangle smoothing. It specifies whether triangles should be smoothed. The nongrid mode preserves the plane of cells by avoiding points that are corners of a cell. The allowable values are:

AF_ADJUST_NONE	No smoothing.
AF_ADJUST_NON_GRID	Applies to points and not part of a cell.
AF_ADJUST_ALL	Adjusts all points connected to triangles.

Figure 1-8 illustrates two cases of adjust mode. The refinement values are specified in Table 1-7. In the second case, smooth triangles are seen where each node is put in the center of surrounding nodes by iterative algorithm. The AF_ADJUST_NON_GRID option does not show much effect on the triangles.

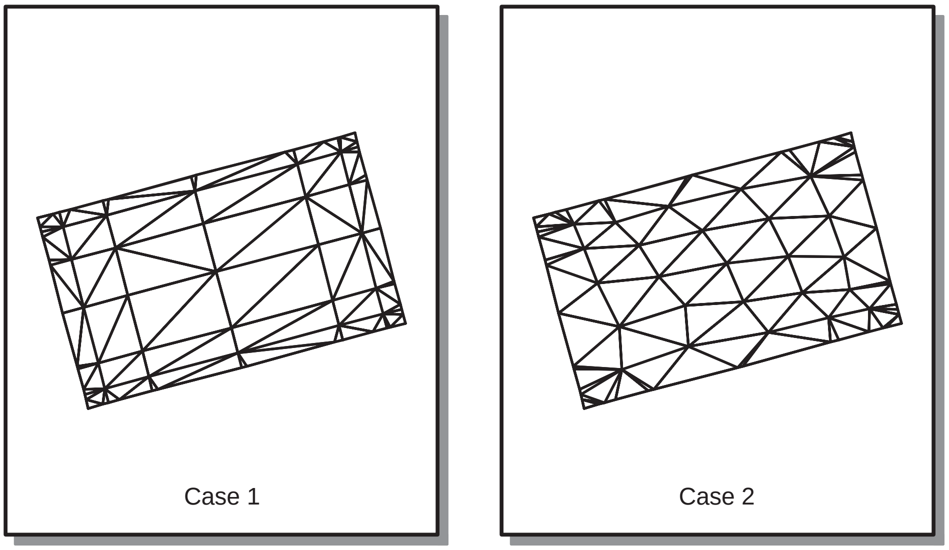


Figure 1-8. Adjust Mode

Table 1-7. Adjust Mode

Refinement	Case 1 Value	Case 2 Value
Surface Tolerance	0	0
Normal Tolerance	25	25
Grid Aspect Ratio	0	0

Refinement	Case 1 Value	Case 2 Value
Maximum Edge Length	0	0
Maximum Grid Lines	10,000	20
Grid Mode	AF_GRID_TO_EDGES	AF_GRID_TO_EDGES
Triangulation Mode	AF_TRIANG_ALL	AF_TRIANG_ALL
Adjust Mode	AF_ADJUST_NONE	AF_ADJUST_ALL
Surface Mode	AF_SURF_ALL	AF_SURF_ALL

Surface Type

Topic: *Faceting

It is very important to remember the surface type in refinement controls. It is safe to set this to AF_SURF_ALL unless one wants to set different refinement controls for different surface types and attach multiple refinements to the same entity.

The set_surf_mode method from the REFINEMENT class specifies the type of surface to which the refinement is applicable. The allowable types are listed below. If more than one refinement is applicable, the more specific one overrides the less specific one.

AF_SURF_ALL	All surface types.
AF_SURF_REGULAR	A surface with planar cells.
AF_SURF_IRREGULAR	A surface with possibly nonplanar cells.
AF_SURF_PLANE	A planar surface.
AF_SURF_CONE	A conical surface (including cylinder).
AF_SURF_SPHERE	A spherical surface.
AF_SURF_TORUS	A toroidal surface.
AF_SURF_SPLINE	A spline surface.

Figure 1-9 illustrates two cases of surface type. The refinement values are specified in Table 1-8. The surfaces are the B-splines surfaces. The surface type for the face in case 2 is AF_SURF_PLANE, so the given refinements are not applied at all. Instead, the ones which were applied earlier (same as in case 1) take effect. Because of this, we do not see the effect of changing the normal tolerance from 25 in case 1 to 5 degrees in case 2.

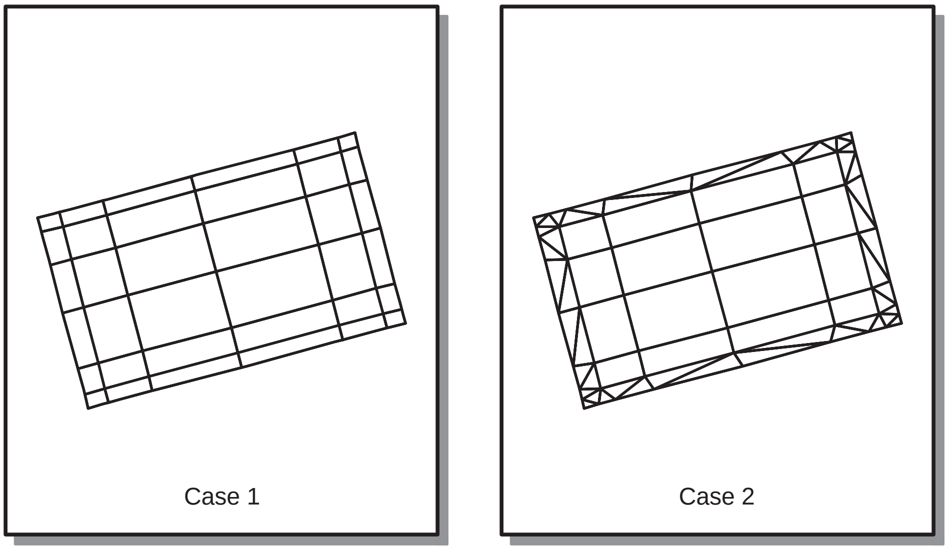


Figure 1-9. Surface Type

Table 1-8. Surface Type

Refinement	Case 1 Value	Case 2 Value
Surface Tolerance	0	0
Normal Tolerance	25	5
Grid Aspect Ratio	0	0
Maximum Edge Length	0	0
Maximum Grid Lines	10,000	20
Grid Mode	AF_GRID_TO_EDGES	AF_GRID_TO_EDGES
Triangulation Mode	AF_TRIANG_NONE	AF_TRIANG_NONE
Adjust Mode	AF_ADJUST_NON_GRID	AF_ADJUST_NON_GRID
Surface Mode	AF_SURF_ALL	AF_SURF_PLANE

Discarding or Keeping Facet Data

Topic: *Faceting

Faceting is required for rendering, but the facet data does not necessarily have to be saved. The facet data can be regenerated from the entity and the refinements attached to the entity. Even though the INDEXED_MESH is not saved to the save file, it is not deleted after rendering either. The mesh remains accessible to the application until either the entity is deleted or the application terminates. This improves performance, because faceting is not carried out repeatedly.

If an application needs to save facet information into its own data structure, it should derive its own mesh manager from either INDEXED_MESH_MANAGER or POLYGON_POINT_MESH_MANAGER. The derived class would redefine the inherited methods and enhance them to carry out the special needs of the application. For example, a class derived from the INDEXED_MESH_MANAGER would need announce_counts, announce_indexed_node, and announce_indexed_polynode methods.

Vertex Templates

Topic: *Faceting

A *vertex template* enables applications to request the mesh manager to attach data to the nodes of a mesh, if the mesh manager is designed to handle the request. The data, which is defined with the class VERTEX_TEMPLATE, includes:

- Coordinate data
- Surface normal
- *uv* parameter
- Partial derivatives and their magnitude
- RGB color
- Transparency
- Texture
- Pointer to user-defined data

Indexed (nonglobal) and coordinate mesh managers create meshes on a face basis. Therefore, they can be designed to attach vertex template data to the nodes. Because global indexed mesh managers share node data between faces, they cannot store face-related information such as attached vertex template data.

The VERTEX_TEMPLATE is typically used from a customized mesh manager. The template defines the information of importance from a mesh node. The template itself is stored in the save file, but not the resulting VERTEX_TEMPLATE instances for each node of the mesh. In other words, if a template is defined, then when the mesh is generated, a template for each node is filled out with information. When the mesh itself goes away, either because the owning entity was deleted or because the application was terminated, the individual templates go away. If the entity is saved to the save file, its refinements and master vertex template are saved as well. The mesh and the information in each individual vertex template are recreated from the entity, refinements, and master template when restored from the save file.

Adaptive Faceting

Topic: *Faceting

Because ACIS supports various types of geometric representation (e.g., planar, conical, toroidal, and spline surfaces), faceting relies on the curvature measurements of each faceted face to capture the underlying geometry. All types of surface geometry, except spline, can be well approximated with a regular grid whose lines are equidistant in parameter space. However, due to their intrinsic free-form nature, spline surfaces necessitate a different approach.

Adaptive faceting is employed to lay a grid of non-equidistant lines, rather than a regular grid, for spline surfaces. The nonregular grid is based on the curvature computation of the spline surface at some parameter space sampling locations. Sampled curvature distribution controls the density of grid lines in the parameter space. Higher curvature areas get a greater number of grid lines, while lower curvature areas get fewer lines. In all cases, faceting tries to satisfy the refinement criteria set by the application.

The adaptive faceting algorithm now subdivides internal triangles according to the specified refinements in the case of faceting a body with triangles when no *uv* grid was selected.

Incremental Faceting

Topic: *Faceting

Incremental faceting saves computation time when only a few faces of a model have undergone some changes. This is accomplished by attaching a *mark attribute* to each faceted face when passing its facets to the mesh manager.

A mark attribute is lost when its owner face is engaged in any Boolean operation or a transformation operation other than translation or rotation. Therefore, a faceted face that has undergone a change will no longer be marked as faceted.

The API function api_mark_faceted_faces allows the user to select whether or not to attach mark attributes to faceted faces. If marking faceted faces is selected, a subsequent call to either of the functions api_facet_unfaceted_entity or api_facet_unfaceted_entities causes only the unmarked faces (i.e., those that have changed since last faceted) to be faceted.

Faceting Functions and Classes

Topic: Faceting

Important API functions for faceting include:

api_initialise_faceter	Initializes the faceter.
api_terminate_faceter	Terminates the faceter.
api_create_refinement	Creates a refinement.
api_facet_entities	Creates facets for a list of entities.
api_facet_entity	Creates facets for an entity.
api_facet_unfaceted_entities	Facets unfaceted faces of a list of entities.
api_facet_unfaceted_entity	Facets unfaceted; i.e., unmarked, faces of an entity.
api_get_default_refinement	Gets the default refinement associated with a type of surface.
api_get_entity_refinement	Gets the refinement attached to the entity.
api_get_mesh_manager	Gets the current mesh manager of faceter.
api_modify_refinement	Modifies the parameters of a refinement.
api_set_default_refinement	Sets the default refinement in the faceter.
api_set_entity_refinement	Attaches a refinement to an entity (BODY, LUMP, SHELL, FACE).

Important classes for faceting include:

ATTRIB_EYE	Defines an organization attribute class.
ATTRIB_EYE_ATTACHED_MESH	Defines an attribute to attach facets to an entity as a MESH.
ATTRIB_EYE_FCTD_MARK	Indicates whether or not face has been faceted.

GLOBAL_MESH_MANAGER	Shows an example of a mesh manager.
INDEXED_MESH	Shows an example of a mesh that stores indices to a vertex array.
INDEXED_MESH_MANAGER	Shows an example of a mesh manager using the indexed protocol.
INDEXED_MESH_MANAGER_PRINTER	Shows an example of a mesh manager using the indexed protocol.
MESH	Defines a generic mesh class that can be attached to an entity.
MESH_MANAGER	A MESH_MANAGER class directs output of the facet data.
POLYGON	Defines a POLYGON.
POLYGON_POINT_MESH	Defines a collection of polygons.
POLYGON_POINT_MESH_MANAGER	Specializes the MESH_MANAGER class when a mesh is maintained as a POLYGON_POINT_MESH.
POLYGON_VERTEX	Defines a vertex on a polygon.
PPM_ON_FACE_MESH_MANAGER	Shows how a MESH-derived mesh is attached to a face.
REFINEMENT	Controls the accuracy and types of polygons generated in the faceter.
SIMPLE_INDEXED_MESH	Shows an example of a mesh that stores indices to a vertex array.
SIMPLE_INDEXED_MESH_MANAGER	To show an example of a mesh manager using the indexed protocol.
STL_MESH_MANAGER	Writes data to an STL formatted file.
VERTEX_TEMPLATE	Creates parameter templates.

Faceting using Scheme

Topic: *Faceting

When using Scheme to exercise faceting, much of the preparatory work is taken care of. The user must create entities, then refinements, and then attach these refinements to the entities. Finally, the entities are faceted and the facets displayed.

Faceting an Entity in Scheme

Topic: Entity, *Faceting

entity:facet generates facets for an entity or list of entities. The argument entity–list specifies the facets of an entity or list of entities to save. The facets produced become part of the part. When facets are saved with the part, refinements cannot be modified unless the entity:delete–facets extension is applied. New refinement modifications must then be faceted with the entity to be saved with that entity. The optional argument view specifies which view to facet the entities, and it is usually used when view–dependent refinements have been set; the default is the active view. When the optional argument unfaceted_only is set to #t, only unfaceted geometry is faceted. This extension returns the input entity-list.

entity:display–facets displays the facets of an entity or list of entities. The optional argument view specifies the view to display the entity; the default is the active view. This extension returns the number of facets displayed from the input entity–list.

The entity:count–facets extension counts the number of facets in an entity. This procedure does not facet the entities. The user must call entity:facet to facet them or nothing prints.

entity:delete–facets deletes the facets from an entity or list of entities. The argument entity–list specifies the facets of an entity or list of entities to delete. This extension returns the input entity–list.

The entity:refinement extension returns the entity's refinement. If no refinement is assigned, this extension returns #f. The entity:refinement gets all the faceting refinements attached to an ENTITY. The optional argument view is used as the base view to determine the orientation of the printed display. It defaults to the active view.

The entity:set–facet–color extension sets the color information to each facet vertex based on the input vector. The newly created VERTEX_TEMPLATE class sets up storage for the tokens used for position, normal, color, and *uv*. When the model is subsequently faceted, the polygon's vertices contain data for each of these tokens.

entity:print–facets prints the facets of an entity or list of entities. If no entities are specified, then all entities that are both displayed and faceted are printed. This procedure does not facet the entities. The user must call entity:facet to facet them or nothing prints.

entity:write–facets writes the facets to a file. The argument entity–list specifies the entity or list of entities to include facet information. The argument file–name specifies the name of the file to create. This extension returns #t if the operation is successful.

Scheme Example

```
; entity:display-facets
; Create and facet a block, then display the facets.
(define ablock
    (solid:block (position 0 0 0)
    (position 25 25 25)))
;; ablock
; Facet the solid block.
(entity:facet ablock)
;; 12
; Display the facets of the solid block.
(entity:display-facets ablock)
;; ()
; Count the number of facets in the block.
(entity:count-facets ablock)
;; 12
(entity:delete-facets ablock)
;; #[entity 2 1]
```

Faceting Refinements in Scheme

Topic: Entity, *Faceting

The two Scheme extensions, entity:set–refinement and refinement, are used in conjunction with each other. The refinement extension is used to create a new refinement for faceting. A refinement is an entity created in the active part that controls how closely the surfaces of entities are approximated by their faceted representation. A refinement is associated with a face, shell, lump, or solid body. The entity:set–refinement is used to attached a refinement to an entity or a group of entities.

refinement:set–prop sets a property of a refinement to control faceting.

"surface tolerance"	is the maximum distance between facet and true surface. Setting it to 0 turns it off.
"normal tolerance"	is the maximum difference between any two normals of facet.
"aspect ratio"	is the approximate aspect ratio of each cell in grid.
"max edge length"	is the maximum size of an edge of a facet.
"max grid lines"	is the maximum number of subdivisions on a face.
"min u_ grid lines"	is the minimum number of *u* grid lines used to display a face.
"min v_ grid lines"	is the minimum number of *v* grid lines used to display a face.

“grading”	adjusts facets to get better parametric aspect ratio of a grid cell.
“grid mode”	specifies whether a grid is used and whether the points where the grid cuts the edge is inserted to the edge.
“triang mode”	specifies how much triangulation to do.
“adjust mode”	specifies type of triangle smoothing to do. Determines if facet nodes should be adjusted for smoothing. Also determines if the grid points should be adjusted or not.
“surf mode”	specifies the type of surface.

Scheme Example

```
; refinement:set-prop
; Create a new refinement.
(define my_refine (refinement))
;; my_refine
; Set a property of the refinement.
(refinement:set-prop my_refine
    ”aspect ratio” 0.5)
;; ()
; Determine the properties of a refinement.
(refinement:props my_refine)
;; ((”surface tolerance” . -1)
;; (”normal tolerance” . 15)
;; (”aspect ratio” . 0.5)
;; (”max edge length” . 0)
;; (”max grid lines” . 512)
;; (”grading” . #f)
;; (”grid mode” . ”AF_GRID_INTERIOR”)
;; (”triang mode” . ”AF_TRIANG_FRINGE_2”)
;; (”adjust mode” . ”AF_ADJUST_NONE”)
;; (”surf mode” . ”AF_SURF_ALL”))
; Set the default property of the refinement.
(refinement:set-default my_refine)
;; ()
; entity:set-refinement
; Create a solid cylinder.
(define my_cyl
    (solid:cylinder (position 0 0 0)
    (position 8 8 0) 20))
;; my_cyl
; my_cyl => #[entity 2 1]
; Create a refinement.
(define my_refine (refinement))
;; my_refine
; my_refine => #[entity 3 1]
; Get the default refinement values.
(refinement:props my_refine)
;; ((”surface tolerance” . 0)
;; (”normal tolerance” . 15)
;; (”aspect ratio” . 0) (”max edge length” . 0)
;; (”max grid lines” . 10000) (”grading” . #f)
;; (”grid mode” . ”AF_GRID_INTERIOR”)
;; (”triang mode” . ”AF_TRIANG_FRINGE_2”)
;; (”adjust mode” . ”AF_ADJUST_NON_GRID”)
;; (”surf mode” . “AF_SURF_ALL”))
; Set the properties of the refinement.
(refinement:set–prop my_refine
    ”aspect ratio” 0.5)
;; ()
; Set the refinement on the cylinder.
(entity:set–refinement my_cyl my_refine)
;; #[entity 2 1]
; Remove the refinement from the cylinder.
(entity:set–refinement my_cyl #f)
;; #[entity 2 1]
```

Faceting Scheme Summary

Topic: Ignore

entity:count–facets	Counts the number of facets in an entity.
entity:delete–facets	Deletes the facets from an entity or list of entities.
entity:display–facets	Displays the facets of an entity or list of entities.
entity:facet	Facets an entity or list of entities.
entity:faceted?	Determines if an entity is faceted.
entity:print–facets	Prints the facets of an entity or list of entities.
entity:refinement	Gets an entity’s refinement.
entity:refinements	Gets all the refinements attached to an ENTITY.
entity:set–refinement	Attaches a refinement to an entity or list of entities.

entity:write–facets	Writes facets to a file.
refinement	Creates a refinement.
refinement?	Determines if a Scheme object is a refinement.
refinement:default	Gets the default refinement.
refinement:props	Gets the current property list of a refinement.
refinement:set–default	Sets the default refinement to the specified refinement.
refinement:set–prop	Sets a property of a refinement.