Chapter 2.
Scheme Extensions Aa thru Mz

Topic: Ignore

Scheme is a public domain programming language, based on the LISP language, that uses an interpreter to run commands. ACIS provides extensions (written in C++) to the native Scheme language that can be used by an application to interact with ACIS through its Scheme Interpreter. The C++ source files for ACIS Scheme extensions are provided with the product. *Spatial*'s Scheme based demonstration application, Scheme ACIS Interface Driver Extension (Scheme AIDE), also uses these Scheme extensions and the Scheme Interpreter. Refer to the *3D ACIS Online Help User's Guide* for a description of the fields in the reference template.

arc:set–center

Scheme Extension: Model Geometry

Action: Sets the center position of a circular curve or edge or an elliptical curve or edge.

Filename: cstr/cstr_scm/geom_scm.cxx

APIs: api_get_ellipse_parameters, api_modify_ellipse

Syntax: `(arc:set-center curve center)`

Arg Types:

curve	elliptical–curve | elliptical–edge
center	position

Returns: elliptical–curve | elliptical–edge

Errors: None

Description: This extension returns the modified elliptical curve or edge. Circular curves and edges are subsets of elliptical curves and edges.

curve specifies an elliptical curve or edge

center specifies the new center position of the ellipse.

Limitations: None

Example:

```
; arc:set–center
; Create a circular edge.
(define arc
    (edge:circular
    (position 15 25 0) 25 0 185))
;; arc
; OUTPUT Original

; Set the center position of the edge.
(define set (arc:set–center arc
    (position –10 –10 –10)))
;; set
; OUTPUT Result
```

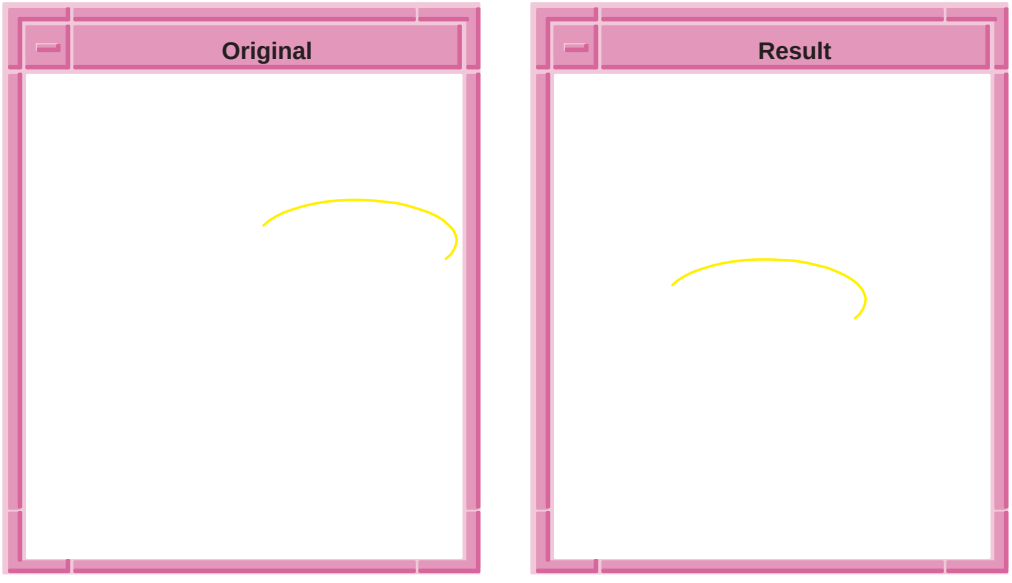


Figure 2-1. arc:set–center

arc:set–direction

Scheme Extension: Model Geometry

Action: Sets the major axis direction of an elliptical curve or edge.

Filename: cstr/cstr_scm/geom_scm.cxx

APIs: api_get_ellipse_parameters, api_modify_ellipse

Syntax: `(arc:set–direction curve vector)`

Arg Types:

curve	elliptical–curve | elliptical–edge
vector	gvector

Returns: elliptical–curve | elliptical–edge

Errors: None

Description: This extension realigns the curve's major axis to the given input vector gvector. If the vector gvector is normal to the input curve curve, the minor axis remains unchanged. Otherwise, the minor axis is realigned based on the cross product of the curve normal with the specified vector gvector.

curve specifies an elliptical curve or edge.

vector specifies the new major axis direction of the ellipse.

Limitations: None

Example:

```
; arc:set–direction
; Create a circular edge.
(define arc1
    (edge:circular
    (position 15 25 0) 25 0 185))
;; arc1
; OUTPUT Original

; Set the direction of the edge.
(define set (arc:set–direction arc1 (gvector 0 1 0)))
;; set
; OUTPUT Result
```

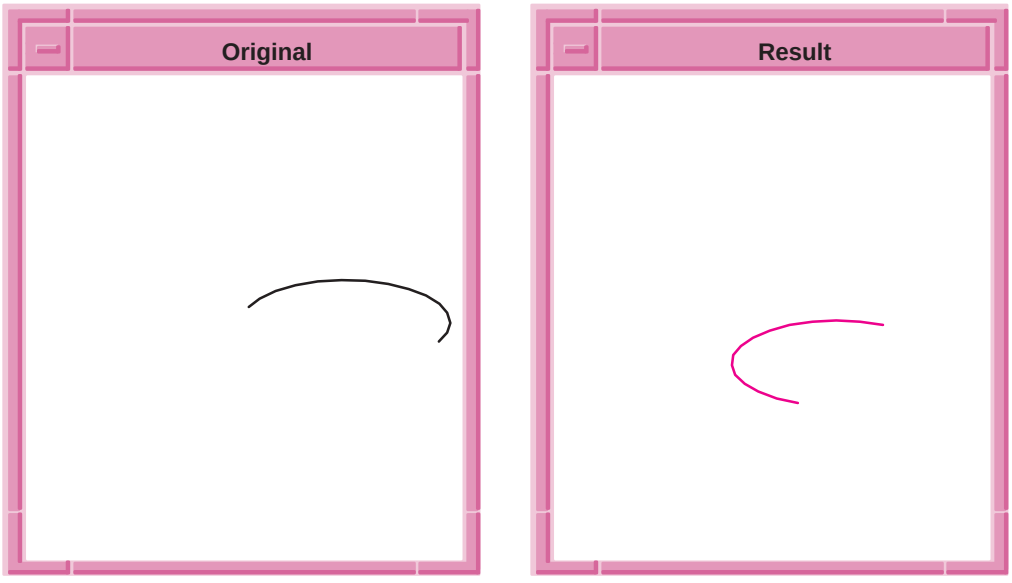


Figure 2-2. arc:set–direction

arc:set–end

Scheme Extension: Model Geometry

Action: Sets the end angle of a circular curve or edge or an elliptical curve or edge.

Filename: cstr/cstr_scm/geom_scm.cxx

APIs: api_get_ellipse_parameters, api_modify_ellipse

Syntax: `(arc:set-end curve angle)`

Arg Types:		
	curve	elliptical–curve | elliptical–edge
	angle	real

Returns: elliptical–curve | elliptical–edge

Errors: None

Description: This extensions extends or shortens the input curve curve to the given angle angle along the curve. The end position is determined using the right hand rule with respect to the curve's normal, the start position, and the desired angle. Circular curves and edges are subsets of elliptical curves and edges.

curve specifies an elliptical curve or edge.

angle specifies the new end angle of the ellipse in degrees.

Limitations: None

Example:

```
; arc:set–end
; Create a circular edge.
(define arc1
    (edge:circular
    (position 15 25 0) 25 0 185))
;; arc1
; OUTPUT Original

; Set the end angle of the edge.
(define set (arc:set–end arc1 45))
;; set
; OUTPUT Result
```

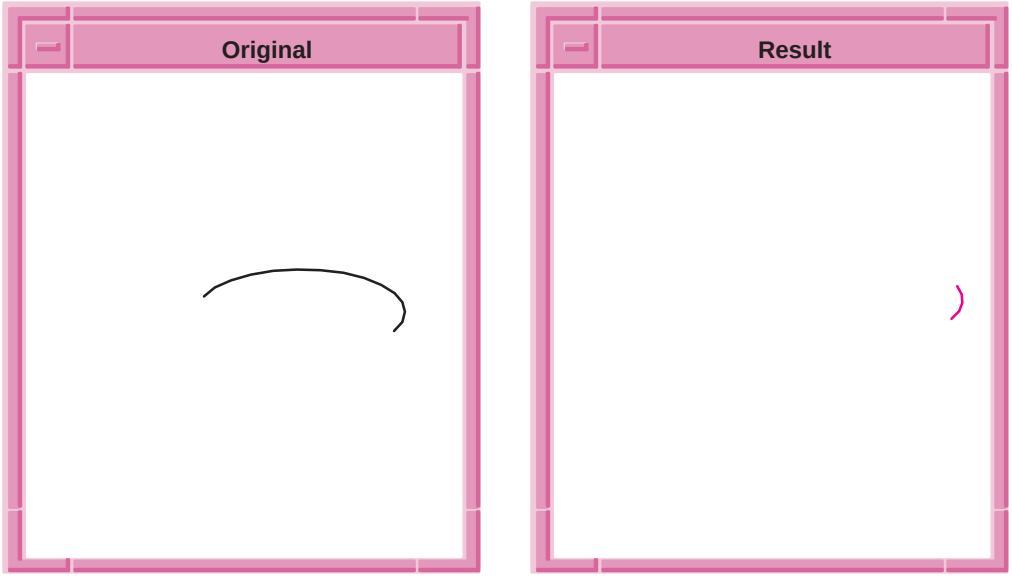


Figure 2-3. arc:set–end

arc:set–normal

Scheme Extension: Model Geometry

Action: Sets the normal of a circular curve or edge or an elliptical curve or edge.

Filename: cstr/cstr_scm/geom_scm.cxx

APIs: api_get_ellipse_parameters, api_modify_ellipse

Syntax: (**arc:set–normal** curve vector)

Arg Types:

curve	elliptical–curve | elliptical–edge
vector	gvector

Returns: elliptical–curve | elliptical–edge

Errors: None

Description: Circular curves and edges are subsets of elliptical curves and edges. This extension realigns the input curve so that its normal vector is the given input vector vector. If the vector gvector is parallel to the input curve curve's normal vector, the minor axis remains unchanged. Otherwise, the minor axis is realigned based on the cross product the specified vector gvector with the curve's major axis. After the minor axis is aligned with respect to the vector normal vector, the major axis is realigned as the cross product between the new minor axis and the vector normal.

curve specifies an elliptical curve or edge.

vector specifies the new normal of the ellipse.

Limitations: None

Example:

```
; arc:set–normal
; Create a circular edge.
(define arc1
    (edge:circular
    (position 15 25 0) 25 0 185))
;; arc1
; OUTPUT Original

; Set the normal of the edge.
(define set (arc:set–normal arc1 (gvector 1 0 0)))
;; set
; OUTPUT Result
(curve:normal arc1)
;; #[gvector 1 0 0]
```

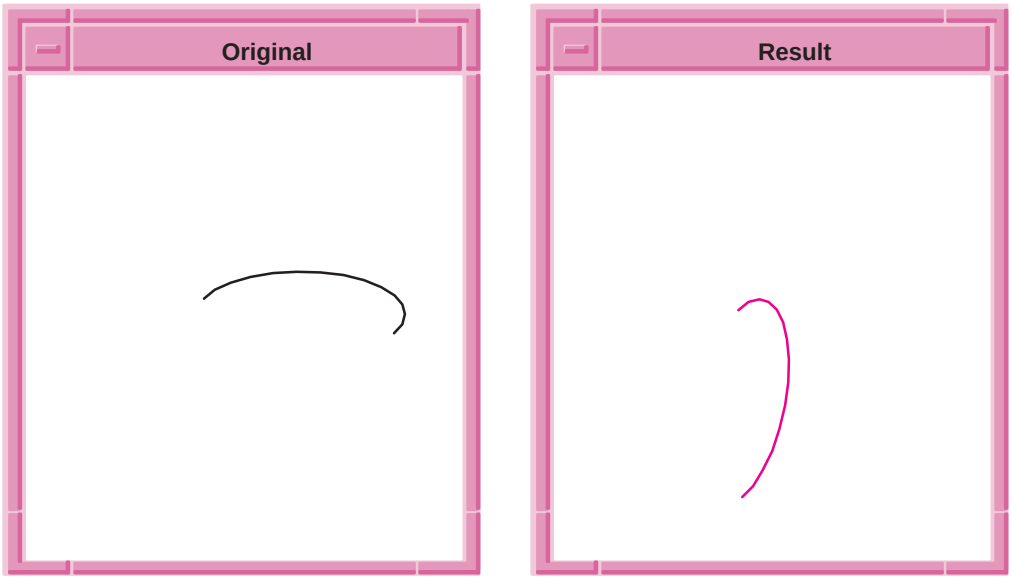


Figure 2-4. arc:set–normal

arc:set–radius

Scheme Extension: Model Geometry

Action: Sets the radius of a circular curve or edge or an elliptical curve or edge.

Filename: cstr/cstr_scm/geom_scm.cxx

APIs: api_get_ellipse_parameters, api_modify_ellipse

Syntax: `(arc:set-radius curve radius)`

Arg Types:

curve	elliptical–curve | elliptical–edge
radius	real

Returns: elliptical–curve | elliptical–edge

Errors: None

Description: Circular curves and edges are subsets of elliptical curves and edges. This extension scales the curve about its center such that the distance from the center to the starting position is equal to the input radius.

curve specifies an elliptical curve or edge.

radius specifies the new radius of the ellipse.

Limitations: None

Example:

```
; arc:set–radius
; Create a circular edge.
(define arc1
    (edge:circular
    (position 15 25 0) 25 0 185))
;; arc1
; OUTPUT Original

; Set the radius of the edge.
(define set (arc:set–radius arc1 12.3))
;; set
(arc:radius arc1)
;; 12.3
; OUTPUT Result
```

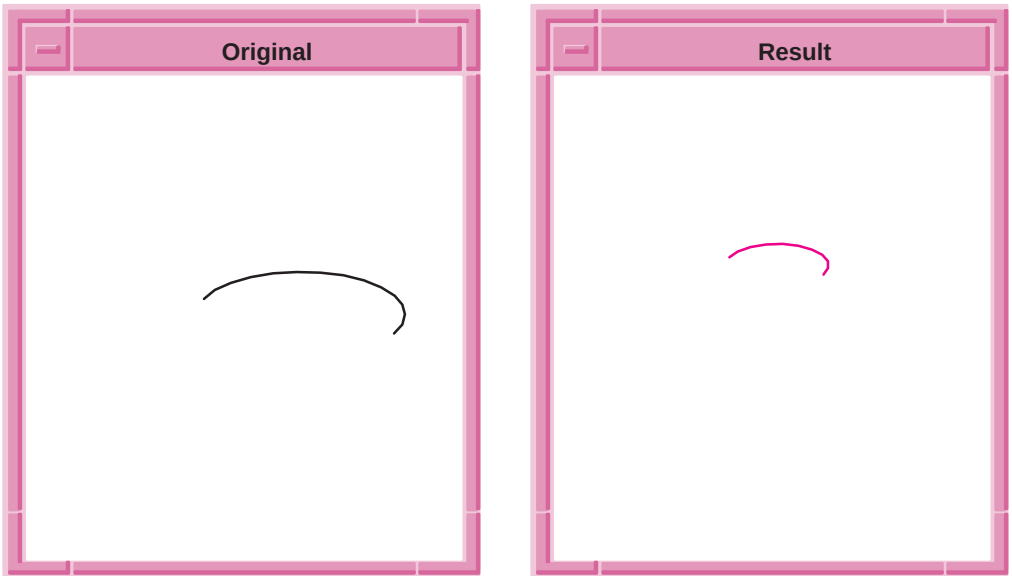


Figure 2-5. arc:set–radius

arc:set–radius–ratio

Scheme Extension: Model Geometry

Action: Sets the radius ratio of an elliptical curve or edge.

Filename: cstr/cstr_scm/geom_scm.cxx

APIs: api_get_ellipse_parameters, api_modify_ellipse

Syntax: (**arc:set–radius–ratio** curve ratio)

Arg Types:

curve	elliptical–curve | elliptical–edge
ratio	real

Returns: elliptical–curve | elliptical–edge

Errors: None

Description: Circular curves and edges are subsets of elliptical curves and edges. The input curve curve is changed such that the minor axis is ratio times as long as the major axis. The curve's center, major axis, start angle, end angle, and normal remain unchanged.

curve specifies elliptical curve or edge.

ratio specifies the new radius ratio of the ellipse.

Limitations: None

Example:

```
; arc:set–radius–ratio
; Create an elliptical circular edge.
(define arc1
    (edge:circular
    (position 15 25 0) 25 0 185))
;; arc1
; OUTPUT Original

; Set the elliptical radius ratio of the edge.
(define set (arc:set–radius–ratio arc1 0.3))
;; set
; OUTPUT Result
```

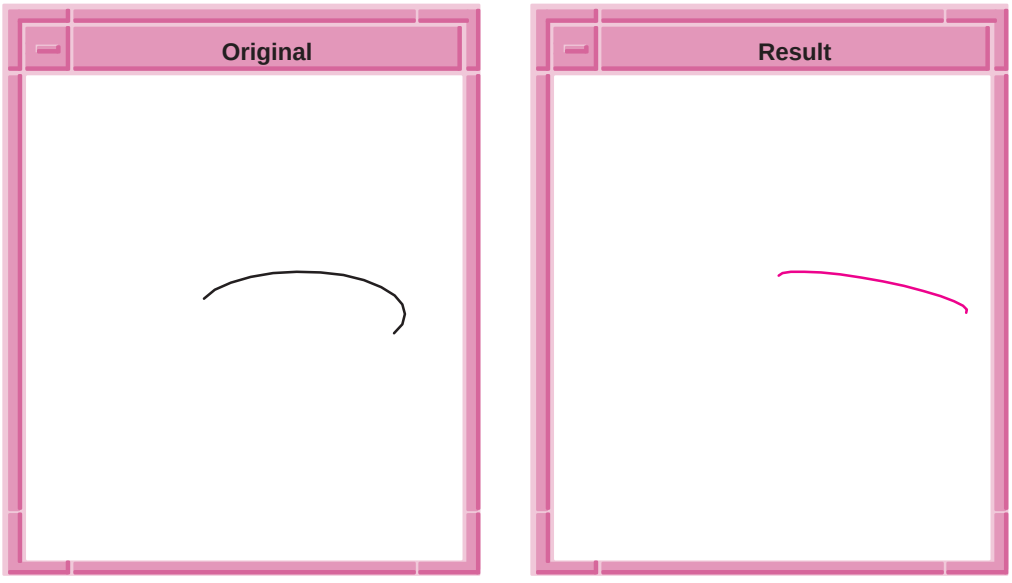


Figure 2-6. arc:set–radius–ratio

arc:set–start

Scheme Extension: Model Geometry

Action: Sets the start angle of a circular curve or edge or an elliptical curve or edge.

Filename: cstr/cstr_scm/geom_scm.cxx

APIs: api_get_ellipse_parameters, api_modify_ellipse

Syntax: `(arc:set-start curve angle)`

Arg Types:

curve	elliptical–curve | elliptical–edge
angle	real

Returns: elliptical–curve | elliptical–edge

Errors: None

Description: Circular curves and edges are subsets of elliptical curves and edges. This extension extends or shortens the input curve curve to the given angle angle along the curve. The start position is determined using the right hand rule with respect to the curve's normal, the end position, and the desired angle.

curve specifies an elliptical curve or edge.

angle specifies the new start angle of the ellipse in degrees.

Limitations: None

Example:

```
; arc:set-start
; Create a circular edge.
(define arc1
    (edge:circular
    (position 15 25 0) 25 0 185))
;; arc1
; OUTPUT Original

; Set the start angle of the edge.
(define set (arc:set-start arc1 45))
;; set
; OUTPUT Result
```

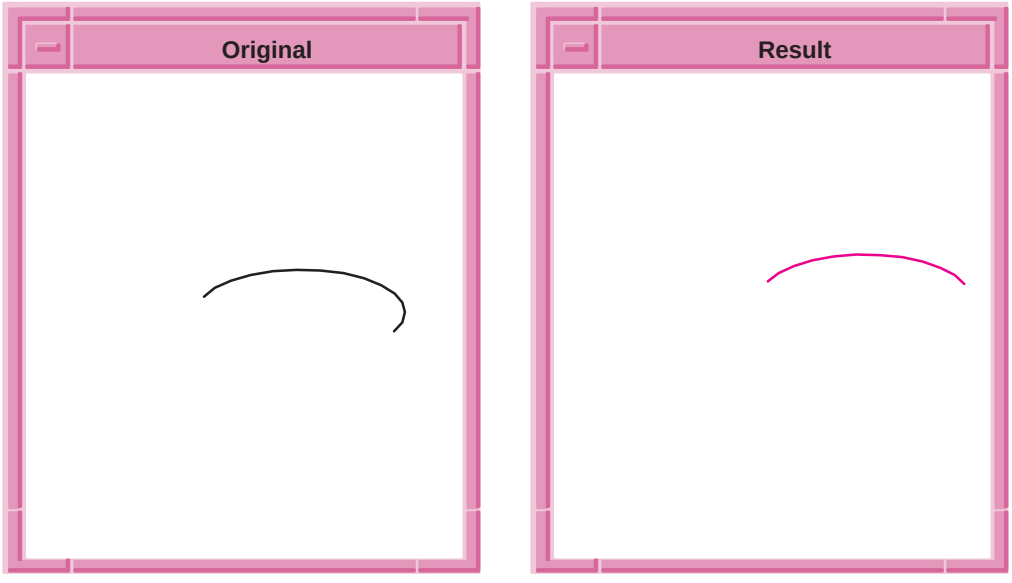


Figure 2-7. arc:set–start

body:find–face

Scheme Extension: Model Geometry, Model Object

Action: Returns a face of the given body.

Filename: cstr/cstr_scm/cstr_scm.cxx

APIs: api_find_face

Syntax: `(body:find-face body direction)`

Arg Types:		
	body	body
	direction	gvector

Returns: entity | boolean

Errors: None

Description: Returns the face of the given body that is the most extreme position in the given direction. For example, passing a direction of (0,0,1) returns the "top" face. Passing a direction of (0,0,–1) returns the "bottom" face. This is a direct interface to the api_find_face function.

body is an input body.

direction is the input direction.

Limitations: None

Example:

```
; body:find-face
; Create topology to illustrate command.
(define block
    (solid:block
    (position 0 0 0) (position 10 10 10)))
;; block
; Find face of block.
(body:find-face block (gvector 0 0 1))
;; #[entity 3 1]
```

body:reverse

Scheme Extension: Model Topology

Action: Reverses the sense on all faces, lumps,wire coedges, and wires within a body.

Filename: cstr/cstr_scm/sld_scm.cxx

APIs: api_reverse_body

Syntax: `(body:reverse body)`

Arg Types:		
	body	body

Returns: body

Errors: None

Description: This extension reverses the sense of each of its faces, as well as the sense of all of face and wire coedges. This is accomplished by switching FORWARD and REVERSED sense.

The sense of a face indicates which side is considered “outside” and which is “inside”. In the typical situation from solid modeling, the “inside” is adjacent to solid material while the “outside” is void. The sense of face is determined by the normal of the underlying surface. A FORWARD sense means the normal points from the face to the “outside”.

body is an input body.

Limitations: Will not take a body as input.

Example:

```
; body:reverse
; Create a planar disk.
(define block (solid:block (position –25 –25 –25)
    (position 25 25 25)))
;; block
; Retrieve list of faces.
(face:types)
; entity:(entity 1 1)
; entity (entity 2 1)
; face:(entity 3 1) face_type:Plane
; face:(entity 4 1) face_type:Plane
; face:(entity 5 1) face_type:Plane
; face:(entity 6 1) face_type:Plane
; face:(entity 7 1) face_type:Plane
; face:(entity 8 1) face_type:Plane
;; #t
; determine if sense is forward.
(entity:sense (entity 3 1))
;; #t
(entity:sense (entity 4 1))
;; #f
; Reverse the coedge, face, and loop orientations.
(define reverse (body:reverse block))
;; reverse
; Verify sense has been reversed.
(entity:sense (entity 3 1))
;; #f
(entity:sense (entity 4 1))
;; #t
```

coedge:add–pcurve

Scheme Extension: Model Geometry

Action: Adds a pcurve to a coedge.

Filename: cstr/cstr_scm/edge_scm.cxx

APIs: None

Syntax: (**coedge:add–pcurve** coedge)

Arg Types: coedge entity

Returns: boolean

Errors: None

Description: Calls sg_add_pcurve_to_coedge to add a pcurve to the defined coedge.

coedge defines the entity where the pcurve is to be added.

Limitations: None

Example:

```
; coedge:add–pcurve
; create a solid sphere.
(define sphere (solid:sphere (position 0 0 0) 20))
;; sphere
; create a solid block.
(define block (solid:block (position –30 –30 15)
    (position 30 30 30)))
;; block
; Subtract the block from the sphere.
(define subtract (bool:subtract sphere block))
;; subtract
(define face1 (cdr (entity:faces sphere)))
;; face1
(define coedge1 (car (entity:coedges face1)))
;; coedge1
; Add a pcurve to the coedge.
(coedge:add–pcurve coedge1)
;; #t
```

curve:circular

Scheme Extension: Construction Geometry

Action: Creates a temporary circular curve.

Filename: cstr/cstr_scm/geom_scm.cxx

APIs: None

Syntax: (**curve:circular** center radius [normal=z–axis])

Arg Types:

center	position
radius	real
normal	gvector

Returns: circular–curve

Errors: None

Description: This extension creates a Scheme object containing an ACIS geometry curve class. Geometry classes must be attached to derived ENTITY class items (e.g., edge) to be displayed and saved. Use the edge:from–curve extension to do this. By themselves, curve Scheme objects are useful for evaluating positional, directional, or curvature information without creating a (topology) entity.

center specifies the center position of the circle.

radius is an implicit line between the center position and the edge of the circle.

normal is a gvector, and it defaults to the z–axis of the active WCS.

Limitations: None

Example:

```
; curve:circular
; Create a circular curve.
; The resultant curve is not an entity.
(define curve1
    (curve:circular
    (position 0 0 0) 25 (gvector 1 –2 1)))
;; curve1
; To view this curve, attach it to an entity.
(define edge1 (edge:from–curve curve1))
;; edge1
```

curve:elliptical

Scheme Extension: Construction Geometry

Action: Creates a temporary elliptical curve.

Filename: cstr/cstr_scm/geom_scm.cxx

APIs: None

Syntax:
```
(curve:elliptical center major–axis [normal=z–axis]
    [ratio])
```

Arg Types:

center	position
major–axis	gvector
normal	gvector
ratio	real

Returns: elliptical–curve

Errors: None

Description: This extension creates a Scheme object containing an ACIS geometry curve class. Geometry classes must be attached to derived ENTITY class items (e.g., edge) to be displayed and saved. Use the edge:from–curve extension to do this. By themselves, curve Scheme objects are useful for evaluating positional, directional, or curvature information without creating a (topology) entity.

center specifies the center position of the ellipse.

major–axis specifies the length and direction of the major axis.

normal is a gvector and defaults to the *z*–axis of the active WCS.

ratio is the ratio between the length of the minor axis and that of the major axis. It must be positive and less than or equal to 1.

Limitations: None

Example:
```
; curve:elliptical
; Create an elliptical circular curve
; with the major axis 2x the minor axis.
; The resultant curve is not an entity.
(define curve1
    (curve:elliptical (position 0 0 0)
    (gvector 10 0 0) (gvector 0 1 0) 0.5))
;; curve1
; To view this curve, attach it to an entity.
(define edge1 (edge:from–curve curve1))
;; edge1
```

curve:linear

Scheme Extension: Construction Geometry

Action: Creates a temporary linear–curve.

Filename: cstr/cstr_scm/geom_scm.cxx

APIs: None

Syntax: `(curve:linear pos–start pos–gvec)`

Arg Types:

pos–start	position
pos–gvec	position | gvector

Returns: linear–curve

Errors: None

Description: This extension creates a Scheme object containing an ACIS geometry curve class. Geometry classes must be attached to derived ENTITY class items (e.g., edge) to be displayed and saved. To accomplish this, use the edge:from–curve extension. By themselves, curve Scheme objects are useful for evaluating positional, directional, or curvature information without creating a (topological) entity.

pos–start must be a position.

pos–gvec can be a position or a gvector.

Limitations: None

Example:

```
; curve:linear
; Create a linear curve from two positions.
(define curve1
    (curve:linear
    (position 0 0 0) (position 30 30 30)))
;; curve1
; Create a linear curve from a
; position and a gvector.
(define curve2
    (curve:linear
    (position 0 0 0) (gvector 0 10 0)))
;; curve2
; Temporary curves must be attached to an entity to
; be viewed.
(define from1 (edge:from–curve curve1))
;; from1
(define from2 (edge:from–curve curve2))
;; from2
```

edge:arclength–metric

Scheme Extension: Construction Geometry, Model Topology

Action: Calculates an arclength metric for an edge.

Filename: cstr/cstr_scm/edge_scm.cxx

APIs: api_edge_arclength_metric

Syntax: (**edge:arclength–metric** edge)

Arg Types:	edge	entity

Returns: real

Errors: None

Description: Arc length parameterization means that length for each set of coedges is normalized and parameterized. This technique is often used when skinning or lofting between coedge lists with unequal numbers of coedges.

edge is an input entity.

Limitations: None

Example:

```
; edge:arclength–metric
; Create an edge given a law with domain dimension 1
; and range dimension 3.
(define edge1 (edge:law ”vec(cos(x),sin(x),x/5)” 0
    (law:eval ”20*pi”)))
;; edge1
; The edge is a helix that turns around ten times.
(edge:arclength–metric edge1)
;; 0
```

edge:arclength–param

Scheme Extension: Model Topology, Construction Geometry

Action: Creates an arclength parameterized edge from an edge.

Filename: cstr/cstr_scm/edge_scm.cxx

APIs: api_edge_arclength_param, api_transform_entity

Syntax: (**edge:arclength–param** edge [approx–ok])

Arg Types:	edge	entity
	approx–ok	boolean

Returns: edge

Errors: None

Description: Arc length parameterization is the normalizing and parameterizing of the length for each set of coedges.

edge is an input entity.

Limitations: None

Example:
```
; edge:arclength–param
; Create an edge given a law with domain dimension 1
; and range dimension 3. The edge is a helix that
; turns around ten times.
(define edge1 (edge:law "vec(cos(x),sin(x),x/5)" 0
    (law:eval "20*pi")))
;; edge1
(define arc–edge1 (edge:arclength–param edge1))
;; arc–edge1
```

edge:bezier

Scheme Extension: Model Object, Construction Geometry

Action: Creates a cubic bezier curved edge from four control points.

Filename: cstr/cstr_scm/edge_scm.cxx

APIs: api_curve_bezier

Syntax: (**edge:bezier** point1 point2 point3 point4)

Arg Types:

point1	position
point2	position
point3	position
point4	position

Returns: edge

Errors: None

Description: The edge:bezier extension creates an edge defined by four specified control point positions (point1, point2, point3, and point4) that creates a spline. The bezier is created inside the edge using those control point positions as an outside boundary for the Bezier curve.

point1, point2, point3, and point4 are four specified control point positions that creates a spline.

Limitations: None

Example:

```
; edge:bezier
; Set the view's eye position, target position,
; and up vector.
(define viewset (view:set (position 0 –100 0)
    (position 0 0 0) (gvector 0 0 1)))
;; viewset
; Define a WCS.
(define wcs1
    (wcs (position 0 0 0) (position 5 0 0)
    (position 0 5 0)))
;; wcs1
; Create bezier edge 1.
(define edge1
    (edge:bezier (position 10 0 0)
    (position 10 0 30) (position 30 0 30)
    (position 30 0 0)))
;; edge1
(define color1 (entity:set–color edge1 6))
;; color1
; Create bezier edge 2.
(define edge2
    (edge:bezier (position 5 0 0)
    (position 15 0 20) (position 25 0 20)
    (position 35 0 0)))
;; edge2
(define color2 (entity:set–color edge2 3))
;; color2
; Create bezier edge 3.
(define edge3
    (edge:bezier (position –5 0 0)
    (position 0 0 25) (position –20 0 20)
    (position –20 0 0)))
;; edge3
(define color (entity:set–color edge3 4))
;; color
; OUTPUT Example
```

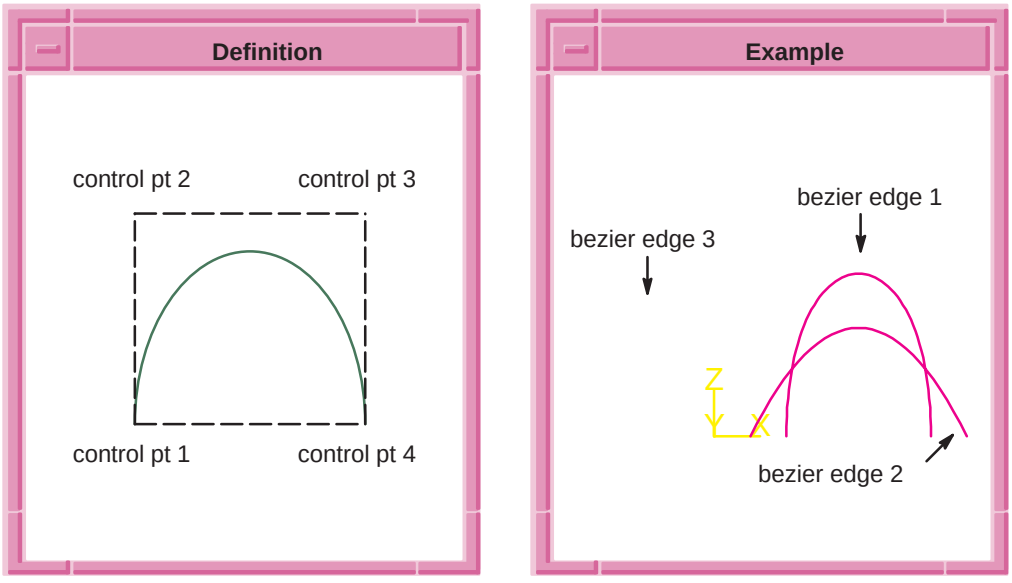


Figure 2-8. edge:bezier

edge:bezier–ndeg

Scheme Extension: Model Object, Construction Geometry

Action: Creates a bezier curved edge of arbitrary degree (ctrlpts–1) from a list of control points.

Filename: cstr/cstr_scm/edge_scm.cxx

APIs: api_mk_ed_int_ctrlpts

Syntax: `(edge:bezier–ndeg (list–points))`

Arg Types: list–points position

Returns: edge

Errors: Error if number of points is less than 2.

Description: The edge:bezier–ndeg extension creates a bezier curved edge defined by a list of control point positions (point1, point2, point3, and point4) that creates a spline. The bezier is created inside the edge using those control point positions as an outside boundary for the Bezier curve.

list–points is a list of positions.

Limitations: None

Example:

```
; edge:bezier–ndeg
; edge1: 1st Degree Bezier Curve from a list of 2
; control points
(define edge1
    (edge:bezier–ndeg (list (position 5 5 0)
    (position 15 50 0))))
;; edge1
(entity:set–color edge1 1)
;; ()
; edge2: 2nd Degree Bezier Curve from a list of
; three control points
(define edge2 (edge:bezier–ndeg (list
    (position 5 5 0) (position 15 50 0)
    (position 55 40 0))))
;; edge2
(entity:set–color edge2 1)
;; ()
; edge3: 3rd Degree Bezier Curve from a list of 4
; control points
(define edge3 (edge:bezier–ndeg (list
    (position 5 5 0) (position 15 50 0)
    (position 55 40 0 ) (position  85 5 0))))
;; edge3
(entity:set–color edge3 3)
;; ()
; edge4: 4th Degree Bezier Curve from a list of 5
; control points
(define edge4 (edge:bezier–ndeg (list
    (position 5 5 0) (position 15 50 0)
    (position 55 40 0) (position  85 5 0)
    (position 45 –15 0))))
;; edge4
(entity:set–color edge4 4)
;; ()
; edge5: 5th Degree Bezier Curve from a list of 6
; control points
(define edge5 ( edge:bezier–ndeg (list
    (position 5 5 0) (position 15 50 0)
    (position 55 40 0) (position  85 5 0)
    (position 45 –15 0) (position 60 0 36))))
;; edge5
```

```
(entity:set–color edge5 5)
;; ()
; edge6: 6th Degree Bezier Curve from a list of 7
; control points
(define edge6 ( edge:bezier–ndeg (list
    (position 5 5 30) (position 15 –50 0)
    (position 55 40 –10 ) (position  85 5 0)
    (position 45 –15 0) (position 60 5 16)
     (position  –10 –20 –45))))
;; edge6
(entity:set–color edge6 6)
;; ()
```

edge:circular

Scheme Extension: Model Object, Construction Geometry

Action: Creates an arc with the specified center position and radius.

Filename: cstr/cstr_scm/edge_scm.cxx

APIs: api_curve_arc

Syntax:
```
(edge:circular center–position radius
    [start–angle=0 [end–angle=360]])
```

Arg Types:

center–position	position
radius	real
start–angle	real
end–angle	real

Returns: entity

Errors: None

Description: center–position specifies the center position of the arc.

radius is an implicit line between the center position and the edge of the arc.

start–angle specifies the arc's starting point in degrees.

end–angle specifies the arc's end angle in degrees. The angle's start and end are measured counterclockwise from the *x*–axis of the current WCS. The start and end locations must be in the current *xy* construction plane.

Limitations: None

Example:

```
; edge:circular
; Set the view's eye position, target position,
; and up vector.
(define viewset (view:set (position 0 0 –100)
    (position 0 0 0) (gvector 1 0 0)))
;; viewset
; Create circular edge 1.
(define edge1
    (edge:circular
    (position 0 0 0) 30 0 90))
;; edge1
; Create circular edge 2.
(define edge2
    (edge:circular
    (position 25 25 0) 20))
;; edge2
; Create circular edge 3.
(define edge3
    (edge:circular
    (position –25 –25 0) 10 180 270))
;; edge3
; Create circular edge 4.
(define edge4
    (edge:circular
    (position –10 –10 0) 15 90 270))
;; edge4
(define zoom (zoom–all))
;; zoom
; OUTPUT Example
```

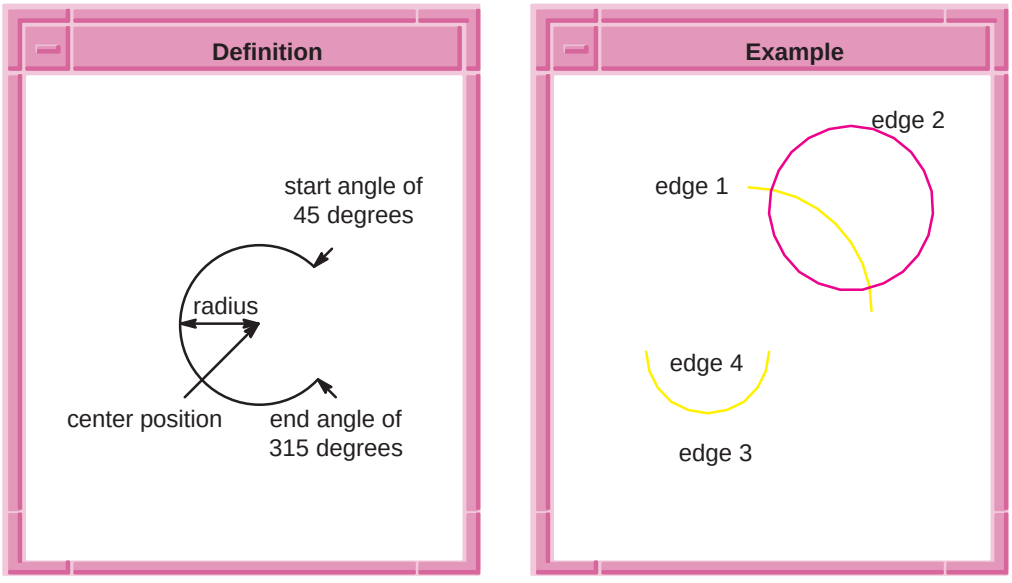


Figure 2-9. edge:circular

edge:circular–3curve

Scheme Extension: Model Object, Construction Geometry

Action: Creates an edge specified by an edge:circular tangent to three curves.

Filename: cstr/cstr_scm/edge_scm.cxx

APIs: api_curve_arc_3curve

Syntax:
```
(edge:circular-3curve entray1 entray2 entray3
    [full=#t])
```

Arg Types:

entray1	entray
entray2	entray
entray3	entray
full	boolean

Returns: entity

Errors: None

Description: At least three entrays (entities with a ray) for the tangent arc must be created prior to using this extension. Create the arcs within range of each other so the tangent circle finds and touches each entity when it is created. All created entities must lie in the same plane.

entray is an entity and a ray.

full creates a circle.

Limitations: None

Example:

```
; edge:circular–3curve
; Set the view’s eye position, target position,
; and up vector.
(define viewset (view:set (position 0 0 –100)
    (position 0 0 0)
    (gvector 1 0 0)))
;; viewset
; Create circular edge 1.
(define circedge1
    (edge:circular (position 0 10 10) 10))
;; circedge1
; Create circular edge 2.
(define circedge2
    (edge:circular (position 20 10 10) 5))
;; circedge2
; Create spline edge 3.
(define splineedge
    (edge:spline (list (position 10 0 10)
    (position 20 10 10) (position 30 10 10)
    (position 40 10 10) (position 42 10 10))))
;; splineedge
; OUTPUT Original

; Create a circular edge tangent to the three curves.
(define 3curve (edge:circular–3curve
    (entray circedge1
    (pick:ray (event 362 242 1 (env:active–view))))
    (entray circedge2
    (pick:ray (event 362 209 1 (env:active–view))))
    (entray splineedge
    (pick:ray (event 302 241 1 (env:active–view))))
    #t))
;; 3curve
; OUTPUT Result
```

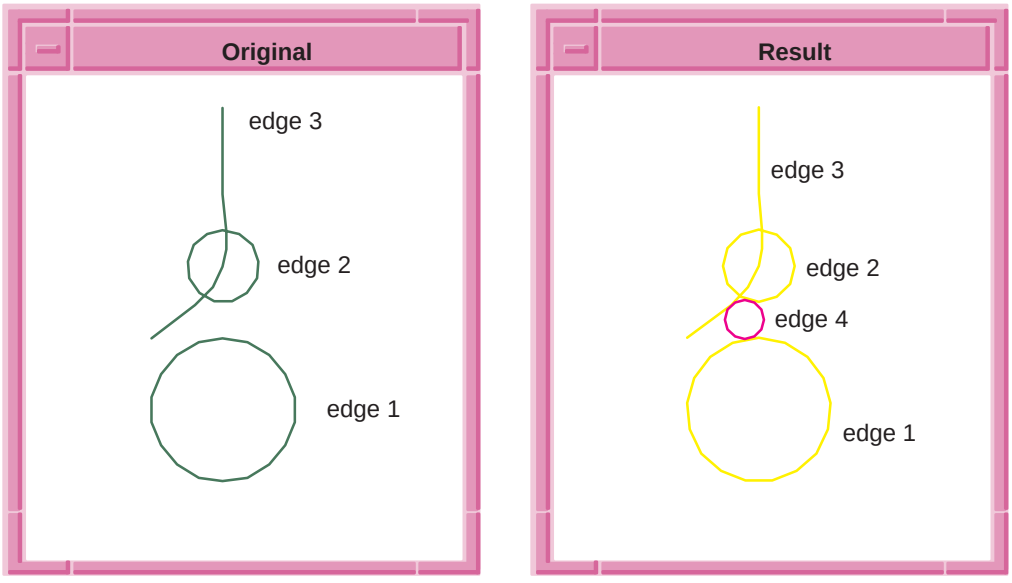


Figure 2-10. edge:circular–3curve

edge:circular–3pt

Scheme Extension: Model Object, Construction Geometry

Action: Creates an arc passing through three positions.

Filename: cstr/cstr_scm/edge_scm.cxx

APIs: api_curve_arc_3pt

Syntax:
```
(edge:circular-3pt edge-pos1 edge-pos2
    edge-pos3 [full=#f])
```

Arg Types:

edge–pos1	position
edge–pos2	position
edge–pos3	position
full	boolean

Returns: entity

Errors: None

Description: edge–pos1 specifies the start location and angle.

edge–pos2 specifies a location on the edge of the arc.

edge–pos3 specifies the end location and angle if the optional argument full is not used.

full determines whether or not to complete the arc in a full circle. If full is set to #t, the extension creates a full circle. The order of the arguments specifies the arc's direction.

Limitations: None

Example:

```
; edge:circular–3pt
; Set the view's eye position, target position,
; and up vector.
(define viewset (view:set
    (position 0 0 –100)
    (position 0 0 0)
    (gvector 1 0 0)))
;; viewset
; Create circular edge 1 passing through 3 points.
(define edge1
    (edge:circular–3pt (position 5 0 0)
    (position 0 5 0) (position –5 0 0)))
;; edge1
; Create circular edge 2 passing through 3 points.
(define edge2
    (edge:circular–3pt (position 15 0 0)
    (position 0 10 0) (position 0 –11 0) #t))
;; edge2
; Create circular edge 3 passing through 3 points.
(define edge3
    (edge:circular–3pt (position 20 0 0)
    (position 0 –20 0) (position –15 –15 0) #f))
;; edge3
; OUTPUT Example
```

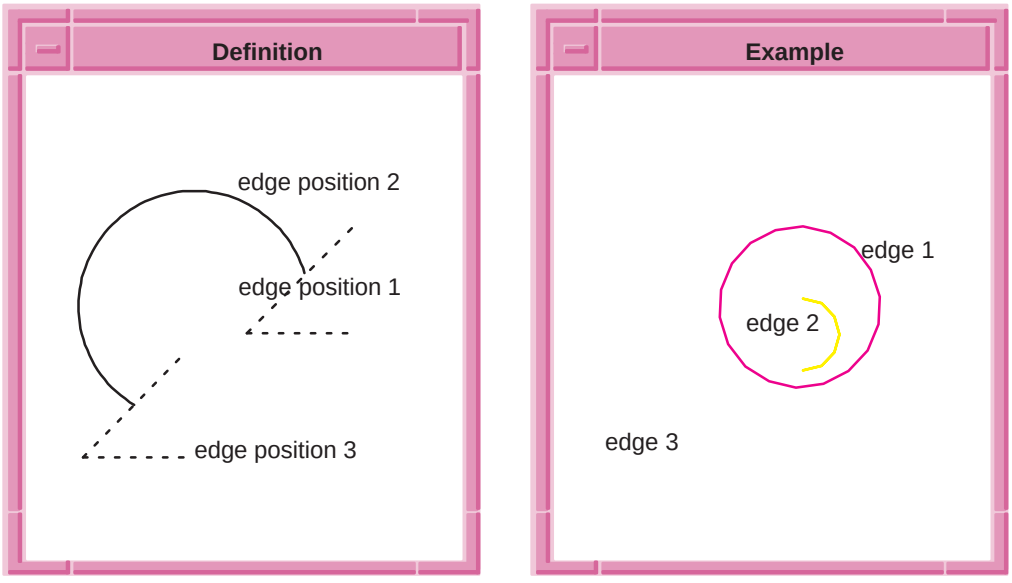


Figure 2-11. edge:circular–3pt

edge:circular–center–rim

Scheme Extension: Model Object, Construction Geometry

Action: Creates an arc given a center position and one or two positions on the arc.

Filename: cstr/cstr_scm/edge_scm.cxx

APIs: api_curve_arc_center_edge

Syntax:
```
(edge:circular–center–rim center–pos
    edge–pos1 [edge–pos2=360])
```

Arg Types:

center–pos	position
edge–pos1	position
edge–pos2	position

Returns: entity

Errors: None

Description: center–pos specifies the center position of the arc.

edge–pos1 specifies the radius based on the center position, along with the starting position of the arc.

edge–pos2 specifies the end position of the arc. If edge–pos2 is not specified, this extension creates a full circle. The direction of the arc is counterclockwise.

Limitations: None

Example:

```
; edge:circular–center–rim
; Set the view’s eye position, target position,
; and up vector.
(define viewset (view:set
    (position 0 0 –100)
    (position 0 0 0) (gvector 1 0 0)))
;; viewset
; Create circular edge 1 using the arc center
position
; and one edge position.
(define edge1
    (edge:circular–center–rim
    (position 20 20 0) (position 30 20 0)))
;; edge1
; Create circular edge 2 using the center position
; and two edge positions.
(define edge2
    (edge:circular–center–rim (position 0 0 0)
    (position 10 0 0) (position 0 –10 0)))
;; edge2
; Create circular edge 3 using the center position
; and one edge position.
(define edge3
    (edge:circular–center–rim
    (position 0 0 0) (position 28 28 0)))
;; edge3
; Create circular edge 4 using the enter position
; and two edge positions.
(define edge4
    (edge:circular–center–rim
    (position 0 0 0) (position 15 15 0)
    (position 0 –8 –8)))
;; edge4
; OUTPUT Example
```

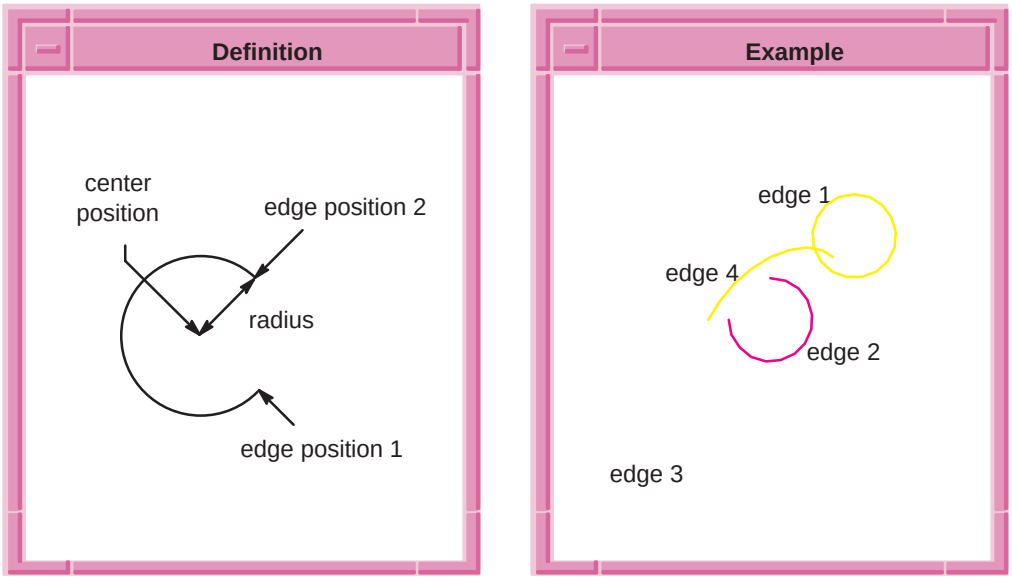


Figure 2-12. edge:circular–center–rim

edge:circular–diameter

Scheme Extension: Model Object, Construction Geometry

Action: Creates an arc passing through two positions based on the diameter.

Filename: cstr/cstr_scm/edge_scm.cxx

APIs: api_curve_arc_diagonal

Syntax:
```
(edge:circular-diameter edge-pos1
    edge-pos2 [full=#f])
```

Arg Types:		
	edge–pos1	position
	edge–pos2	position
	full	boolean

Returns: entity

Errors: None

Description: edge–pos1 specifies the start location of the arc.

edge–pos2 specifies the end location of the arc. The diameter of the arc is defined by the first and second edge positions.

full determines whether to complete the arc. If the argument full is set to #t, a full circle is created.

Limitations: None

Example:

```
; edge:circular–diameter
; Set the view’s eye position, target position,
; and up vector.
(define viewset (view:set (position 0 0 –100)
    (position 0 0 0) (gvector 1 0 0)))
;; viewset
; Create circular edge 1 based on two positions.
(define edge1
    (edge:circular–diameter
    (position 0 0 0) (position 20 0 0)))
;; edge1
; Create circular edge 2 based on two positions.
(define edge2
    (edge:circular–diameter
    (position 5 5 0) (position 25 15 0) #t))
;; edge2
; Create circular edge 3 based on two positions.
(define edge3
    (edge:circular–diameter
    (position –5 –5 3) (position 5 4 12)))
;; edge3
; OUTPUT Example
```

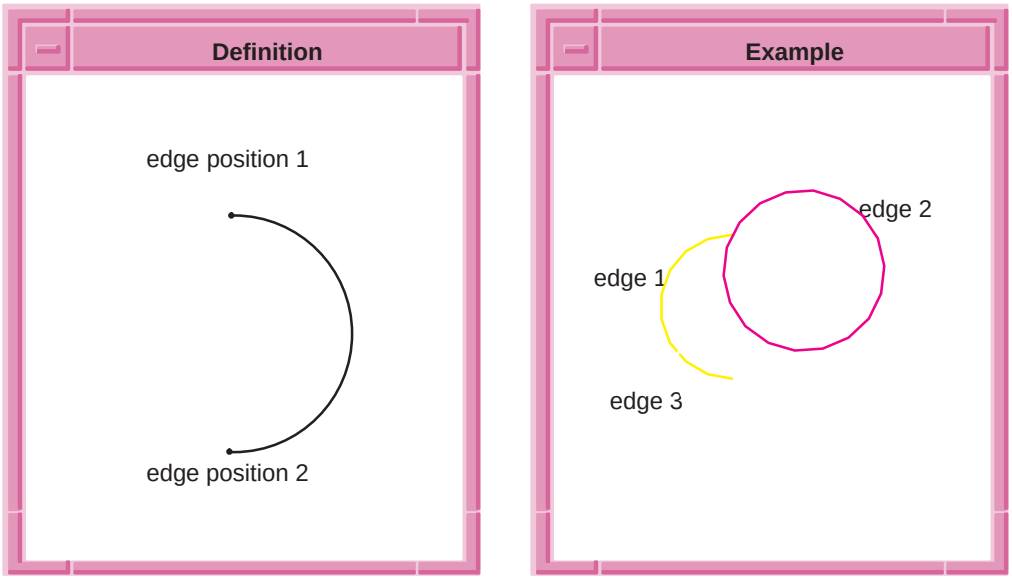


Figure 2-13. edge:circular–diameter

edge:conic

Scheme Extension: Model Object, Construction Geometry

Action: Creates a rho conic edge in which the geometrical definition represents a hyperbola or parabola.

Filename: cstr/cstr_scm/edge_scm.cxx

APIs: api_mk_ed_conic

Syntax: `(edge:conic point1 point2 point3 rho)`

Arg Types:		
	point1	position
	point2	position
	point3	position
	rho	real

Returns: edge

Errors: None

Description: The direction of the conic curve is from the start point to the end point.

point1 specifies the start position of the curve.

point2 specifies the end position of the curve.

point3 specifies the shoulder position of the curve.

rho specifies a number between 0 and 1. The argument rho has values ranging from 0.0 to 1.0, exclusive and defines the type of curve. Values less than 0.5 result in an elliptical appearance, while values greater than 0.5 result in a hyperbola; values equal to 0.5 result in a parabola.

Note *The value of* rho *must be 0 <* rho *< 1; an edge is not constructed for any other value of* rho*.*

Limitations: None

Example:

```
; edge:conic
; Create a rho conic parabola.
(define edge1
    (edge:conic (position –50 0 0)
    (position 50 0 0) (position 0 50 0) 0.5))
;; edge1
; Create a hyperbola.
(define edge2
    (edge:conic (position –50 0 0)
    (position 50 0 0) (position 0 50 0) 0.9))
;; edge2
; Create a conic ellipse.
(define edge3
    (edge:conic (position –50 0 0)
    (position 50 0 0) (position 0 50 0) 0.2))
;; edge3
; OUTPUT Example
```

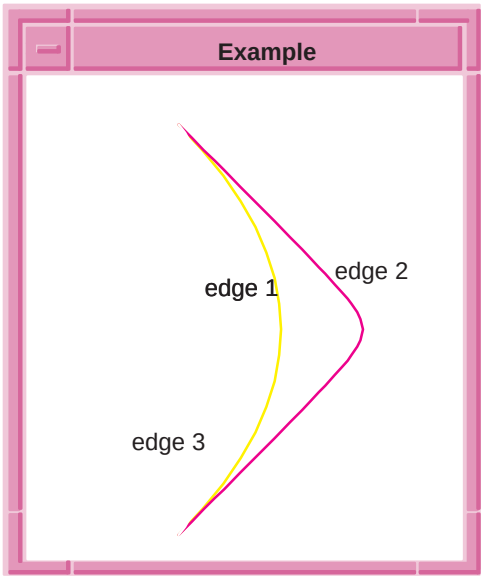


Figure 2-14. edge:conic

edge:ellipse

Scheme Extension: Model Object, Construction Geometry

Action: Creates an edge that is part of an ellipse with a given center, normal, major axis, and radius ratio.

Filename: cstr/cstr_scm/edge_scm.cxx

APIs: api_mk_ed_ellipse

Syntax:
```
(edge:ellipse center normal major–axis
    [radius–ratio=1 [start–angle=0 end–angle=360]])
```

Arg Types:

center	position
normal	gvector
major–axis	gvector
radius–ratio	real
start–angle	real
end–angle	real

Returns: entity

Errors: None

Description: If start–angle and end–angle are not given then a fill ellipse is created. If the ratio is not given it is assumed to be 1. The start and end angles are given in degrees.

center is the center of ellipse.

normal is a normal w.r.t. plane of the ellipse.

major–axis is a major axis of ellipse.

radius–ratio is a ratio between major axis and minor axis.

start–angle specifies the start of the ellipse given as an angle (in degrees) in relation to the major axis..

end–angle specifies the end of the ellipse given as an angle (in degrees) in relation to the major axis.

Limitations: None

Example:

```
; edge:ellipse
; Create an elliptic edge that is the unit circle.
(define edge1
    (edge:ellipse (position 0 0 0)
    (gvector 0 0 10) (gvector 10 0 0)))
;; edge1
; Create elliptic edge that is not the unit circle.
(define edge2
    (edge:ellipse (position 0 0 0)
    (gvector 0 0 10) (gvector 10 0 0) 0.5))
;; edge2
; Create part of an elliptic edge that is not
; the unit circle.
(define edge3
    (edge:ellipse (position 0 0 0)
    (gvector 0 10 10) (gvector 10 0 0) 0.5 0 90))
;; edge3
; OUTPUT Example
```

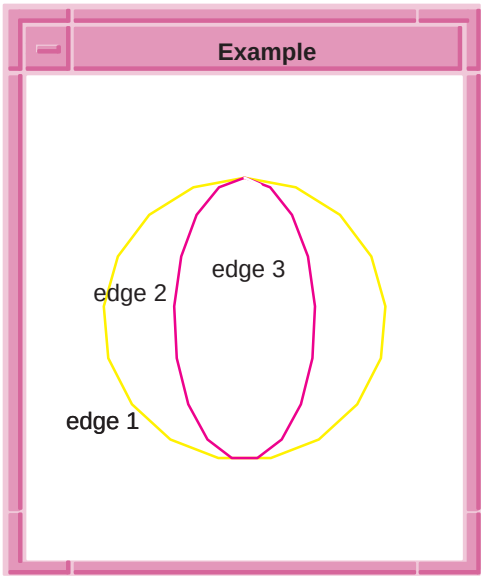


Figure 2-15. edge:ellipse

edge:elliptical

Scheme Extension: Model Object, Construction Geometry

Action: Creates a full or partial ellipse by specifying the start and end angles.

Filename: cstr/cstr_scm/edge_scm.cxx

APIs: api_curve_ellipse

Syntax:
```
(edge:elliptical center–pos start–pos ratio
    [start–angle=0 [end–angle=360]])
```

Arg Types:		
	center–pos	position
	start–pos	position
	ratio	real
	start–angle	real
	end–angle	real

Returns: entity

Errors: None

Description: This command displays circles and ellipses on any plane.

center–pos specifies the center position of the ellipse.

start–pos specifies the start location of one axis on the ellipse (including the magnitude of that axis).

ratio specifies the radius ratio of the other axis length to the major axis length; in a circle, the radius ratio is exactly 1.

start–angle specifies the start location of the ellipse in degrees.

end–angle specifies the end location to create a partial ellipse in degrees.

Limitations: None

Example:

```
; edge:elliptical
; Create elliptical edge by specifying
; the center position, start position, and ratio.
(define center1 (position 0 0 0))
;; center1
(define start (position 16.9705627484771
    –8.48528137423857 –8.48528137423857))
;; start
(define edge1 (edge:elliptical center1 start 0.25))
;; edge1
; Create elliptical edge 2 by specifying the center
; and start positions, the ratio, and
; the start and end angles.
(define center2 (position 15 15 0))
;; center2
(define start (position 15 –5 0))
;; start
(define edge2
    (edge:elliptical center2 start 0.5 0 270))
;; edge2
; OUTPUT Example
```

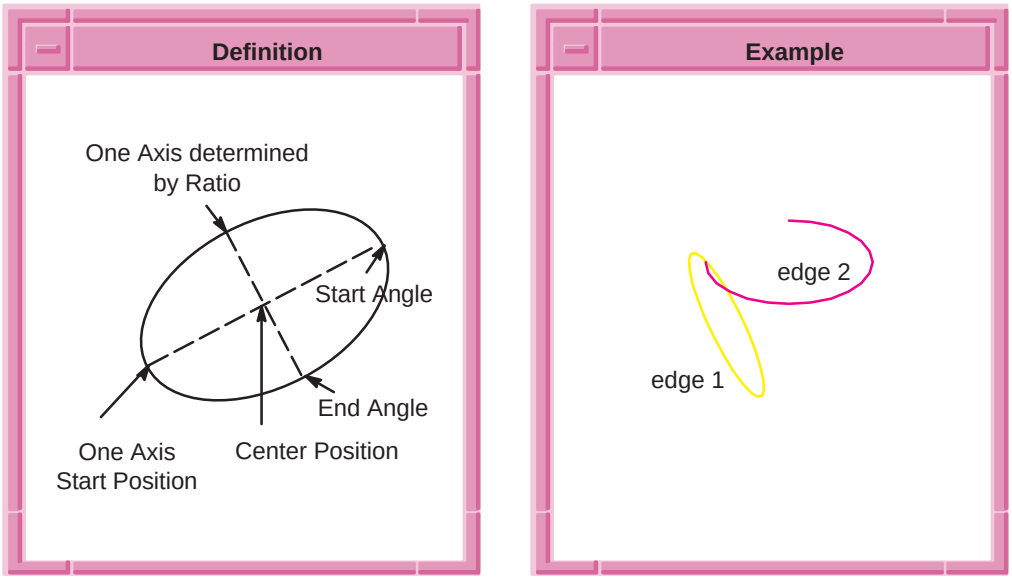


Figure 2-16. edge:elliptical

edge:end

Scheme Extension: Construction Geometry

Action: Returns the ending position of the edge.

Filename: cstr/cstr_scm/edge_scm.cxx

APIs: None

Syntax: `(edge:end edge)`

Arg Types: edge entity

Returns: vertex

Errors: None

Description: Returns the ending position of the edge. The ending position of the edge for this activity is not the position defined when creating the circle. The ending position of the edge (in this instance) is the end of the edge.

edge is an input edge.

Limitations: None

Example:

```
; edge:end
; Create a circle.
(define circle(edge:ellipse (position 0 0 0)
    (gvector 0 0 1) (gvector 10 0 0) 1 0 360))
;; circle
;Define the start point.
(define start–point(edge:end circle))
;; start–point
; Returns the ending position of the edge.
(edge:end circle)
;; #[position 10 0 0]
```

edge:end–dir

Scheme Extension: Construction Geometry

Action: Returns the end direction of the curve.

Filename: cstr/cstr_scm/edge_scm.cxx

APIs: None

Syntax: (**edge:end–dir** edge)

Arg Types:	edge	entity

Returns: gvector

Errors: None

Description: Refer to Action.

edge is an input entity.

Limitations: None

Example:

```
; edge:end–dir
; Create elliptical edge by specifying the center
; and start positions, the ratio, and the start and
; end angles.
(define center (position 15 15 0))
;; center
(define start (position 15 –5 0))
;; start
(define hello–edge
    (edge:elliptical center start 0.5 0 270))
;; hello–edge
; Get end direction of the curve.
(edge:end–dir hello–edge)
;; #[gvector 1.83690953072103e–15 –20 0]
```

edge:extend

Scheme Extension: Construction Geometry

Action: Creates a new edge that is an extension of an existing edge.

Filename: cstr/cstr_scm/edge_scm.cxx

APIs: None

Syntax: `(edge:extend in–entity in–start in–end)`

Arg Types:

in–entity	edge
in–start	real
in–end	real

Returns: entity

Errors: None

Description: in–entity specifies an edge entity that is to be copied.

in–start and in–end are the copied edge provides new starting and ending parameters. These effectively extend or contract the edge.

Limitations: None

Example:

```
; edge:extend
; Extend a edge from an existing edge.
(define center (position 15 15 0))
;; center
(define start (position 15 –5 0))
;; start
(define edge1
    (edge:elliptical center start 0.5 0 270))
;; edge1
(define extend (edge:extend edge1 25 10))
;; extend
```

edge:fillet

Scheme Extension: Construction Geometry

Action: Creates a fillet blend on two edge entrays (entities with rays).

Filename: cstr/cstr_scm/trim_scm.cxx

APIs: api_curve_fillet

Syntax: `(edge:fillet entray1 entray2 radius [trim1=#t [trim2=#t]])`

Arg Types:		
	entray1	entray
	entray2	entray
	radius	real
	trim1	boolean
	trim2	boolean

Returns: entity

Errors: None

Description: The entities picked to be filleted must be edge entities. Edges on the face of a solid cannot be filleted with this command. However, edges on the face of a solid can be blended and chamfered.

entray1 specifies the first entity and pick–ray.

entray2 specifies the second entity and a pick–ray.

radius specifies the curve of the fillet between the two entities. The ray of the selected entity determines the portion of the entity that remains. The ray relates to a specific position on the entity.

trim1 and trim2 indicate whether to trim the edges to the fillet ends. The created arc is tangent to the entities.

Limitations: None

Example:

```
; edge:fillet
; Set the view's eye position, target position,
; and up vector.
(define viewset (view:set (position 0 –100 0)
    (position 0 0 0)
    (gvector 0 0 1)))
;; viewset
; Create linear edge 1.
(define edge1
    (edge:linear (position 0 0 0)
    (position 40 0 0)))
;; edge1
; Create linear edge 2.
(define edge2
    (edge:linear (position 35 0 –5)
    (position 35 0 25)))
;; edge2
; Create linear edge 3.
(define edge3
    (edge:linear (position 40 0 20)
    (position 20 0 20)))
;; edge3
; Create linear edge 4.
(define edge4
    (edge:linear (position 25 0 25)
    (position 25 0 10)))
;; edge4
; OUTPUT Original
```

```
; Fillet the corner where edge 1 and 2 intersect
; and trim the edges.
(define fillet1 (edge:fillet (entray edge1
    (ray (position 5 0 0) (gvector 0 1 0)))
    (entray edge2 (ray (position 35 0 20)
    (gvector 1 0 0))) 3))
;; fillet1
; Fillet the corner where edge 2 and 3 intersect
; but do not trim the fillets.
(define fillet2 (edge:fillet (entray edge2
    (ray (position 35 0 20) (gvector 1 0 0)))
    (entray edge3 (ray (position 25 0 20)
    (gvector 0 1 0))) 3 #f #f))
;; fillet2
; Fillet the corner where edge 3 and 4 intersect,
; and partially trim the fillets.
(define fillet3 (edge:fillet (entray edge3
    (ray (position 35 0 20) (gvector 1 0 0)))
    (entray edge4 (ray (position 25 0 15)
    (gvector 0 1 0))) 3 #f))
;; fillet3
; OUTPUT Result
```

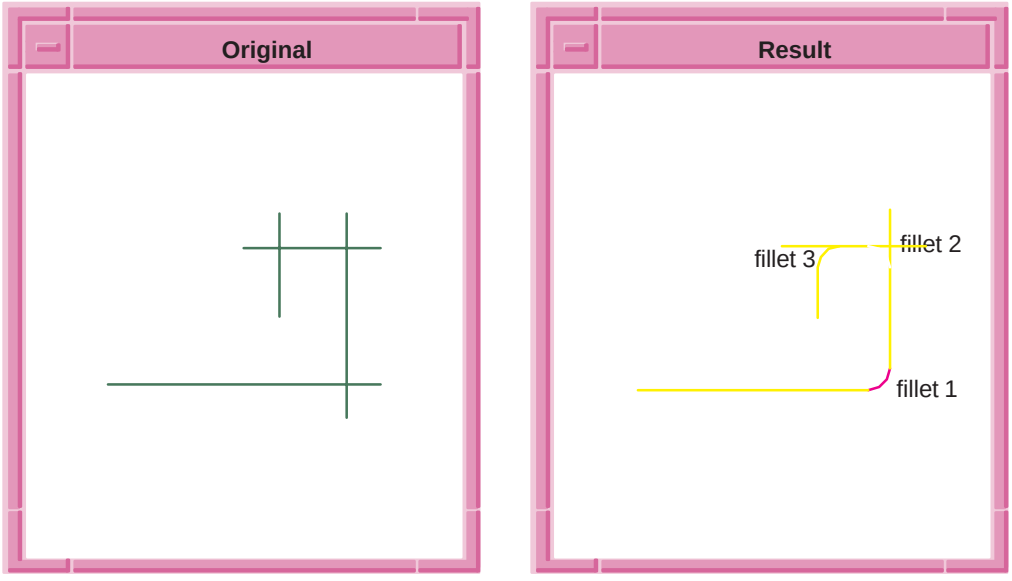


Figure 2-17. edge:fillet

edge:from–curve

Scheme Extension: Construction Geometry

Action: Creates an edge from a curve.

Filename: cstr/cstr_scm/geom_scm.cxx

APIs: None

Syntax: `(edge:from–curve curve [start end])`

Arg Types:		
	curve	curve
	start	real
	end	real

Returns: edge

Errors: None

Description: Creates an edge from a bounded curve or a CURVE entity. The optional start and end parameters are ignored when using a bounded curve.

curve is an input curve.

start is the starting point of the curve.

end is the ending point of the curve.

Limitations: None

Example:

```
; edge:from–curve
; Create a circular curve data structure, and
; convert it to an edge entity.
(define edge (edge:from–curve (curve:circular
    (position 0 0 0) 25)))
;; edge
```

edge:get–tolerance

Scheme Extension: Construction Geometry

Action: Gets the tolerance of a TEDGE.

Filename: cstr/cstr_scm/tmod_scm.cxx

APIs: None

Syntax: `(edge:get–tolerance edge [force–update])`

Arg Types:	edge force–update	edge boolean
Returns:	real	
Errors:	Edge not tolerant	
Description:	This extension gets the tolerance of a TEDGE. If the logical force–update is present and TRUE, the tolerance is recomputed. edge is an input edge. force–update is a logical argument set to recompute the tolerance.	
Limitations:	None	

Example:

```
; edge:get–tolerance
; Create something with tolerant topology.
(define block (solid:block (position –25 –25 –25)
    (position 25 25 25)))
;; block
; Make a tolerant edge with non–zero tolerance.
(define edge (pick:edge (ray (position 0 0 0)
    (gvector 1 0 1))))
;; edge
; make the edge a different color
(entity:set–color edge 4)
;; ()
(define tol–edge (edge:tolerant edge))
;; tol–edge
(define move (tolerant:move tol–edge
    (gvector 5 0 5)))
;; move
; Get tolerance of edge.
(edge:get–tolerance tol–edge)
;; 5
```

edge:law

Scheme Extension: Construction Geometry, Laws

Action: Creates an edge from a law.

Filename: cstr/cstr_scm/edge_scm.cxx

APIs: api_edge_law

Syntax: `(edge:law law start end [perp–law])`

Arg Types:		
	law	law
	start	real
	end	real
	perp–law	law

Returns: edge

Errors: None

Description: Refer to Action.

law is of the type law.

start is the start parameter at which the law will be evaluated.

end is the end parameter at which the law will be evaluated.

perp–law is a helper law.

Limitations: None

Example:

```
; edge:law
; Create an edge given a law with domain dimension 1
; and range dimension 3.
(define law
    (edge:law "vec (cos (x),sin (x),x/5)" 0
    (law:eval "20*pi")))
;; law
; The edge is a helix that turns around ten times.
```

edge:length

Scheme Extension: Construction Geometry

Action: Returns the length of the edge.

Filename: cstr/cstr_scm/edge_scm.cxx

APIs: None

Syntax: `(edge:length edge)`

Arg Types:		
	edge	entity

Returns: real

Errors: None

Description: Returns the length of the edge.

edge is an input entity.

Limitations: None

Example:
```
; edge:length
; Create a circle.
(define circle(edge:ellipse (position 0 0 0)
    (gvector 0 0 1) (gvector 10 0 0) 1 0 360))
;; circle
; Calculate the length of the edge.
(edge:length circle)
;; 62.8318530717959
```

edge:linear

Scheme Extension: Model Object, Construction Geometry

Action: Creates a linear–edge between two locations.

Filename: cstr/cstr_scm/edge_scm.cxx

APIs: api_curve_line, api_curve_line_tangent

Syntax:
```
(edge:linear {position | point | entray}
    {position | point | entray})
```

Arg Types:		
	position	position
	point	point
	entray	entray

Returns: entity

Errors: None

Description: Specify locations as positions, points, or entrays.

position specifies the start location of the line. The second position argument specifies the end location of the line.

point argument specifies the start location of the line. The second point argument specifies the end location of the line.

entray specifies an entity and a pick ray. The ray specifies the start position and gvector. The corresponding end of the line is tangent to the curve at the point nearest to the pick location.

Limitations: None

Example:
```
; edge:linear
; Create two linear edges given two positions.
(define edge1
    (edge:linear (position 0 0 0)
    (position 30 30 0)))
;; edge1
(define edge2
    (edge:linear (position 30 30 0)
    (position 0 30 0)))
;; edge2
; Define point 1.
(define pt1 (point (position 30 0 0)))
;; pt1
; Define point 2.
(define pt2 (point (position 0 30 0)))
;; pt2
; Create linear edge 3 from the two points.
(define edge3 (edge:linear pt1 pt2))
;; edge3
```

edge:mid–point

Scheme Extension: Construction Geometry

Action: Returns the middle point position of an edge.

Filename: cstr/cstr_scm/edge_scm.cxx

APIs: None

Syntax: (**edge:mid–point** edge [approximation=#t])

Arg Types:

edge	entity
approximation	boolean

Returns: gvector

Errors: None

Description: Approximation set to TRUE returns the exact geometrical midpoint of the edge, while FALSE returns the midpoint in parameter space. Default is TRUE.

edge is an input entity.

approximation is a boolean argument set to return the exact geometrical midpoint of the edge.

Limitations: None

Example:

```
; edge:mid–point
; Create elliptical edge by specifying the center
; and start positions, the ratio, and the start and
; end angles.
(define center (position 15 15 0))
;; center
(define start (position 15 –5 0))
;; start
(define hello–edge
    (edge:elliptical center start 0.5 0 270))
;; hello–edge
; Get the midpoint of the edge.
(edge:mid–point hello–edge)
;; #[position 7.92893218813452 29.142135623731 0]
```

edge:mid–point–dir

Scheme Extension: Construction Geometry

Action: Returns the direction of the midpoint of the curve.

Filename: cstr/cstr_scm/edge_scm.cxx

APIs: None

Syntax: (**edge:mid–point–dir** edge [approximation=#t])

Arg Types:		
	edge	entity
	approximation	boolean

Returns: gvector

Errors: None

Description: Approximation set to TRUE returns the direction at the exact geometrical midpoint of the edge, while FALSE returns the midpoint in parameter space. Default is TRUE.

edge is an input entity.

approximation is a boolean argument set to return the exact geometrical midpoint of the edge.

Limitations: None

Example:
```
; edge:mid–point–dir
; Create elliptical edge by specifying the center
; and start positions, the ratio, and the start and
; end angles.
(define center (position 15 15 0))
;; center
(define start (position 15 –5 0))
;; start
(define hello–edge
    (edge:elliptical center start 0.5 0 270))
;; hello–edge
; Get the midpoint direction of the edge.
(edge:mid–point–dir hello–edge)
;; #[gvector 7.07106781186547 14.142135623731 0]
```

edge:plaw

Scheme Extension: Construction Geometry, Laws

Action: Creates an edge from a law defining a parameter space curve on a surface.

Filename: cstr/cstr_scm/edge_scm.cxx

APIs: api_edge_plaw

Syntax: (**edge:plaw** surface law start end)

Arg Types:		
	surface	face
	law	law
	start	real
	end	real

Returns: edge

Errors: None

Description: Creates an edge from a law that defines a parameter space curve on a surface. The parameter space law is limited within a given domain, a subset of the real numbers and range, a subset of 3D space.

surface is an input surface.

law is of the type law.

start is start of law parameter.

end is end of law parameter.

Limitations: None

Example:
```
; edge:plaw
; Create a block to get a planar face
(define block1 (solid:block
    (position –60 –60 –10) (position 30 30 40)))
;; block1
(define face1 (car (entity:faces block1)))
;; face1
; Define a circle in parameter space
(define law1 (law ”vec (10*cos(x), 10*sin (x))”))
;; law1
; create a 3d curve
(define edge1 (edge:plaw face1 law1 0 (* 2 PI)))
;; edge1
```

edge:reverse

Scheme Extension: Model Topology, Construction Geometry

Action: Reverses the sense of an edge.

Filename: cstr/cstr_scm/edge_scm.cxx

APIs: None

Syntax: (**edge:reverse** edge)

Arg Types:	edge	edge

Returns: entity

Errors: None

Description: Refer to Action.

edge is an input edge.

Limitations: None

Example:

```
; edge:reverse
; Create an edge and reverse its sense.
(define block1
    (solid:block (position 0 0 0)
    (position 10 10 10)))
;; block1
; Get a list of the block's edges.
(define edges1
    (entity:edges block1))
;; edges1
; Get a list of the block's faces.
(define faces1
    (entity:faces block1))
;; faces1
(define one-edge (car edges1))
;; one-edge
(define one-face (car faces1))
;; one-face
; Determine if an edge is reverse
; with respect to a face.
(edge:reversed? one-edge one-face)
;; #f
; Reverse the sense of the edge.
(define reverse (edge:reverse one-edge))
;; reverse
; Determine if the edge has been reversed.
(edge:reversed? one-edge one-face)
;; #t
```

edge:spiral

Scheme Extension: Construction Geometry, Model Object

Action: Creates an edge from a spiral definition.

Filename: cstr/cstr_scm/edge_scm.cxx

APIs: api_edge_spiral

Syntax:

```
(edge:spiral center normal [start-position |
    start-direction] width angle [handedness
    start-radius])
```

Arg Types:	center	position
	normal	gvector
	start–position	position
	start–direction	gvector
	width	real
	angle	real
	handedness	boolean
	start–radius	real

Returns: edge

Errors: Specifying a starting position or direction on the normal axis.

Description: This command creates an Archimedian spiral edge. This implies that the radius expands at a constant rate. The resulting edge is perpendicular to the normal axis specified by the center and normal arguments.

The edge returned has information associated with it that indicates where the axis of the spiral is located. If the edge is then swept, a rail law is created that orients the profile relative to the center axis, rather than using a minimum rotation rail. This improves the speed of sweeping, and results in the intuitive sweeping of a spiral.

The command is overloaded to accept the start of the spiral as either a start position or a start direction. The start–position is the position where the spiral starts. The start–direction is a vector specifying an offset from the center. It should be noted that the resulting edge is perpendicular to the axis, but is not necessarily in the plane of the center position, depending on the starting position or direction specified.

The rate at which the spiral expands is specified with the width argument. The width is the perpendicular distance between consecutive loops of the spiral. The angle argument specifies how many revolutions the spiral should make, in degrees.

The optional argument handedness determines whether the direction of movement is right handed or left handed with respect to the normal vector. The default of TRUE is right handed.

The optional argument start–radius allows a specification of the start point at a desired distance in the start direction. One use for this argument is to specify a zero start radius so the spiral begins at the center.

Refer to Appendix D, *Spring Scheme Examples*, for additional illustrations and Scheme examples.

center and normal are the arguments used for defining edge which is perpendicular to the normal axis.

start–position is the position where the spiral starts.

start–direction is a vector specifying an offset from the center.

width specifies the rate at which the spiral expands.

angle argument specifies how many revolutions the spiral should make, in degrees.

handedness determines whether the direction of movement is right handed or left handed with respect to the normal vector.

start–radius allows a specification of the start point at a desired distance in the start direction.

Limitations: None

Example:

```
; edge:spiral
; Make a spiral edge with a center start
(define handedness #t)
;; handedness
(define width 2)
;; width
(define rot–angle 1800)
;; rot–angle
(define start–radius 0)
;; start–radius
(define center (position 0 0 0))
;; center
(define normal (gvector 0 0 10))
;; normal
(define start (gvector 5 0 5))
;; start
; Define the spiral
(define spiral (edge:spiral center normal start width
    rot–angle handedness start–radius))
;; spiral
; View the spiral edge
(define dl (view:dl))
;; dl
(define color (view:set–bg–color 7))
;; color
(define viewset (view:set (position 1 0.75 2)
    (position 0 0 0)
    (gvector 0 0 1)))
;; viewset
(define zoom (zoom–all))
;; zoom
(render:rebuild)
;; ()
; OUTPUT Example
```

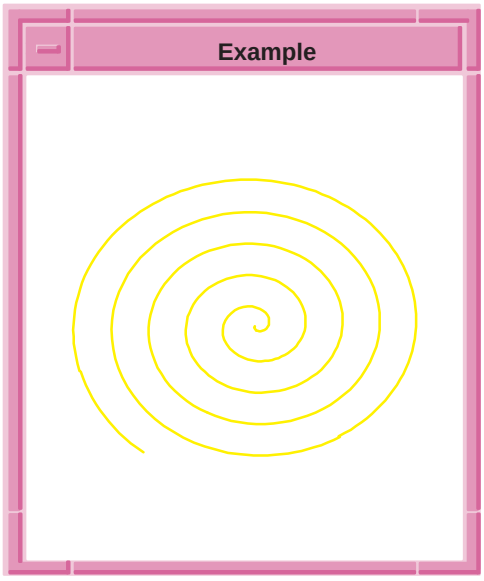


Figure 2-18. edge:spiral

edge:spline

Scheme Extension: Model Object, Spline Interface

Action: Creates a continuous spline edge from a list of positions.

Filename: cstr/cstr_scm/edge_scm.cxx

APIs: api_curve_spline

Syntax:

```
(edge:spline position–list
    [[start–angle | start–direction |
spline–condition]
    [end–angle | end–direction]] [”periodic”]
    [”exact”])
```

Arg Types:		
	position–list	position | (position ...)
	start–angle	real
	start–direction	gvector
	spline–condition	boolean
	end–angle	real
	end–direction	gvector
	”periodic”	string
	“exact”	string

Returns: spline–edge

Errors: None

Description: The argument position–list is a list of locations through which the spline should pass. The number of positions in the position–list must be at least two. The optional argument start–direction is the gvector of the tangent to the spline at the start position. The optional argument end–direction is the gvector of the tangent to the spline at the end position. If either start–direction or end–direction is not specified, a free end condition is used at that end. A “free” or “natural” spline condition means that the second derivative at that end is 0.

To force a “free” end condition, either assign #f to the argument spline–condition, or do not specify an argument for the end condition. Specifying #f results in a “free” end condition at the start, so the tangent direction needs to be specified at the end.

Specifying an end–direction vector indicates the end direction of the curve tangent at the end.

Specify the start–angle and end–angle in degrees. If an angle is specified, the curve is tangent to a vector in the *xy* plane of the active WCS. The angle specifies the angle between the tangent vector and the *x*–axis of the active WCS measured counterclockwise.

When "periodic" is included, the interpolation produces a periodic spline, as long as the first and last position in the list are within resabs, and no end conditions are specified. This check is performed lower down the stack.

If the string “exact” is included, the interpolation is as accurate as possible. Otherwise, it will be within resfit of the points being interpolated.

position–list is a list of locations through which the spline should pass.

start–direction is the gvector of the tangent to the spline at the start position.

start–angle and end–angle in degrees specifies the angle between the tangent vector and the *x*–axis of the active WCS measured counterclockwise.

spline–condition forces a “free” end condition by assigning #f to the argument spline–condition.

end–direction vector indicates the end direction of the curve tangent at the end.

”periodic” is included, the interpolation produces a periodic spline, as long as the first and last position in the list are within resabs, and no end conditions are specified. This check is performed lower down the stack.

“exact” is included, the interpolation is as accurate as possible. Otherwise, it will be within resfit of the points being interpolated.

Limitations: None

Example:

```
; edge:spline
; Set the view’s eye position, target position,
; and up vector.
(define viewset (view:set (position 0 0 100)
    (position 0 0 0)
    (gvector 0 1 0)))
;; viewset
; Define a new WCS.
(define wcs (wcs (position 0 0 0) (position 5 0 0)
    (position 0 5 0)))
;; wcs
; Create 3 spline edges.
(define spline1 (edge:spline (list (position 0 0 0)
    (position 5 5 0) (position 10 15 0)
    (position 15 20 0) (position 20 15 0)
    (position 25 5 0) (position 30 0 0))))
;; spline1
; spline 1 is shown in green.


(define spline2 (edge:spline (list (position 0 0 0)
    (position 5 5 0) (position 10 15 0)
    (position 15 20 0) (position 20 15 0)
    (position 25 5 0) (position 30 0 0)) 0 45))
;; spline2
; spline 2 is shown in blue.
(define spline3 (edge:spline (list (position 0 0 0)
    (position 5 5 0) (position 10 15 0)
    (position 15 20 0) (position 20 15 0)
    (position 25 5 0) (position 30 0 0))
    (gvector 0 1 0) (gvector 0 1 0)))
;; spline3
; spline 3 is shown in magenta.
; OUTPUT Example
```

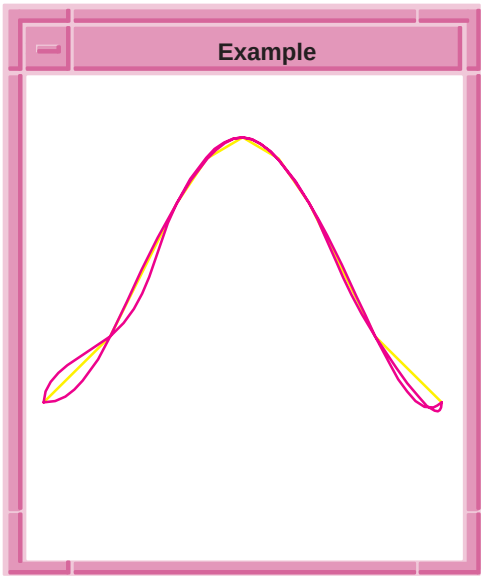


Figure 2-19. edge:spline

edge:spline2

Scheme Extension: Model Object, Spline Interface

Action: Creates a continuous spline edge from a list of positions and parameter values.

Filename: cstr/cstr_scm/edge_scm.cxx

APIs: api_curve_spline2

Syntax:

```
(edge:spline position–list param–list
[start–direction | spline–condition]
[end–direction])
```

Arg Types:		
	position–list	position | (position ...)
	param–list	real | (real ...)
	start–direction	gvector
	spline–condition	boolean
	end–direction	gvector

Returns: spline-edge

Errors: None

Description: The argument position-list is a list of locations through which the spline should pass. The argument param-list is the parameter value at which the spline should pass the corresponding position. The number of positions in the position-list must be at least two and must be equal to the number of reals in the param-list. The optional argument start-direction is the gvector of the tangent to the spline at the start position. The optional argument end-direction is the gvector of the tangent to the spline at the end position. If either start-direction or end-direction is not specified, a free end condition is used at that end. A ”free” or ”natural” spline condition means that the second derivative at that end is 0.

To force a ”free” end condition, either assign #f to the argument spline-condition, or do not specify an argument for the end condition. Specifying #f results in a ”free” end condition at the start, so the tangent direction needs to be specified at the end.

Specifying an end-direction vector indicates the end direction of the curve tangent at the end.

position-list is a list of locations through which the spline should pass.

param-list is the parameter value at which the spline should pass the corresponding position.

start-direction is the gvector of the tangent to the spline at the start position.

spline–condition forces a “free” end condition by assigning #f to the argument spline–condition.

end-direction is the gvector of the tangent to the spline at the end position.

Limitations: None

Example:

```
; edge:spline2
; Set the view
(view:gl)
;; #[view 3607154]
; Set the view's eye position, target position,
; and up vector.
(define viewset (view:set (position 0 0 100)
    (position 0 0 0) (gvector 0 1 0)))
 ;; viewset
 ; Create 3 spline edges.
(define spline1 (edge:spline2 (list (position 0 0 0)
    (position 5 5 0) (position 10 15 0)
```

```
    (position 15 20 0) (position 20 15 0)
    (position 25 5 0) (position 30 0 0))
    (list 0 1 2 3 4 5 6)))
;; spline1
;; set up so we can see edges
(view:edges ON)
;; #[view 3607154]
(zoom–all)
;; #[view 3607154]
(render:rebuild)
;; #[view 3607154]
(entity:set–color spline1 GREEN)
; spline 1 is shown in green.
(define spline2 (edge:spline2 (list
    (position 0 0 0) (position 5 5 0)
    (position 10 15 0) (position 15 20 0)
    (position 20 15 0) (position 25 5 0)
    (position 30 0 0))
    (list 0 .1666 .3333 .5 .6666 .8333 1)
    (gvector 0 30 0) (gvector 0 –30 0)))
;; spline2
(entity:set–color spline2 BLUE)
; spline 2 is shown in blue.
(define spline3 (edge:spline2 (list
    (position 0 0 0) (position 5 5 0)
    (position 10 15 0) (position 15 20 0)
    (position 20 15 0) (position 25 5 0)
    (position 30 0 0))
    (list 0 .1666 .3333 .5 .6666 .8333 1)
    (gvector 30 0  0) (gvector 30 0  0)))
;; spline3
(entity:set–color spline3 MAGENTA)
; spline 3 is shown in magenta.
(define spline4 (edge:spline2 (list
    (position 0 0 0) (position 5 5 0)
    (position 10 15 0) (position 15 20 0)
    (position 20 15 0) (position 25 5 0)
    (position 30 0 0))
    (list 0 .1666 .3333 .5 .6666 .8333 1)
    (gvector 30 0  0)))
;; spline4
(entity:set–color spline4 RED)
; spline4 is shown in RED (may be hidden by spline3)
(define spline5 (edge:spline2 (list
    (position 0 0 0) (position 5 5 0)
```

```
    (position 10 15 0) (position 15 20 0)
    (position 20 15 0) (position 25 5 0)
    (position 30 0 0))
    (list 0 .1666 .3333 .5 .6666 .8333 1) #f
    (gvector 30 0  0)))
;; spline5
(entity:set-color spline5 YELLOW)
; spline 5 is shown in YELLOW (may be hidden by
spline3)
; OUTPUT Example
```

edge:split

Scheme Extension: Model Topology, Construction Geometry

Action: Splits an edge into two entities at a specified position.

Filename: cstr/cstr_scm/trim_scm.cxx

APIs: api_split_curve

Syntax: `(edge:split edge1 {position | edge2})`

Arg Types:

edge1	edge
position	position
edge2	edge

Returns: (edge ...)

Errors: None

Description: The argument edge1 specifies the edge to split. The argument position specifies the location to split the edge. The argument edge2 specifies the edge with which to split. For closed curves, the current entity is trimmed to the position and at 0 degrees, and a new edge is created from the position to 360 degrees.

edge1 specifies the edge to split.

position specifies the location to split the edge.

edge2 specifies the edge with which to split.

Limitations: None

Example:

```
; edge:split
; Set the view's eye position, target position,
; and up vector.
(define viewset (view:set (position 0 –100 0)
    (position 0 0 0)
    (gvector 0 0 1)))
;; viewset
; Create a circular edge.
(define edge1
    (edge:circular–3pt (position 10 0 0)
    (position 0 0 10) (position –10 0 0)))
;; edge1
; Create a linear edge.
(define edge2
    (edge:linear (position 0 0 0)
    (position 20 0 20)))
;; edge2
; OUTPUT Original

; Split the first edge into two edges.
(edge:split edge1 (position 0 0 10))
;; (#[entity 2 1] #[entity 4 1])
; Split the second edge into two edges.
(edge:split edge2 edge1)
;; (#[entity 3 1] #[entity 5 1])
; OUTPUT Result
```

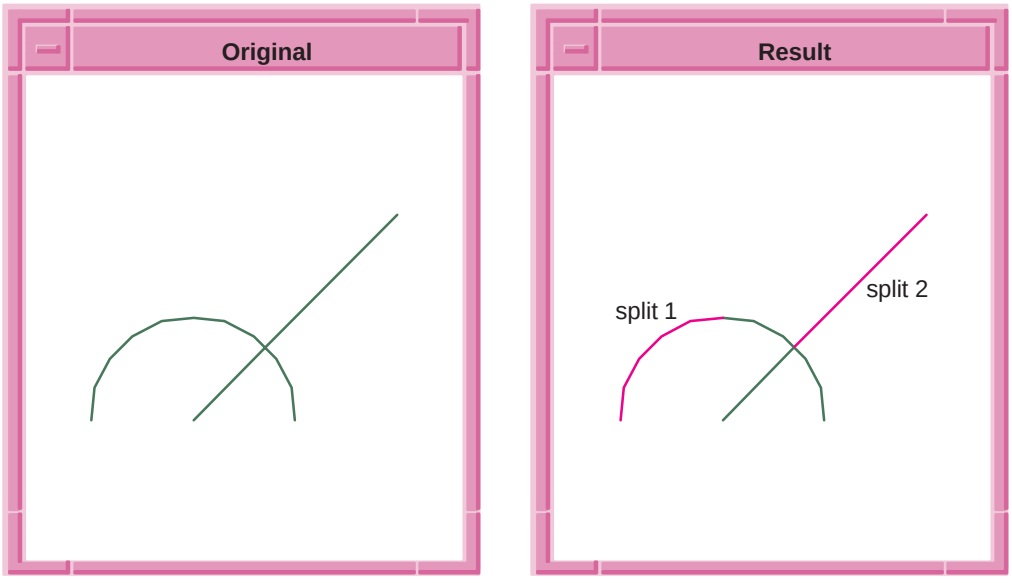


Figure 2-20. edge:split

edge:split–at–disc

Scheme Extension: Model Topology, Construction Geometry

Action: Splits an edge up to G1 or G2 discontinuities.

Filename: cstr/cstr_scm/trim_scm.cxx

APIs: api_split_edge_at_disc

Syntax: `(edge:split–at–disc edge [disc–order | acis–opts])`

Arg Types:		
	edge	edge
	disc–order	integer
	acis–opts	acis–options

Returns: (edge ...)

Errors: None

Description: The input edge is split up to the G1 or G2 (disc–order = 1 or 2) by accessing the discontinuity_info stored in the curve. The result is a list of edges that are split, including the original.

edge is an input edge.

disc–order is the discontinuity order.

acis–opts contains journaling and versioning information.

Limitations: None

Example:

```
; edge:split–at–disc
; We create a discontinuous curve by creating a
; spline edge using duplicate knots at internal
; points.
(part:clear)
(define ctrlpts_pos
        (list (position –2 0 0)
        (position –1.5 0 0)
        (position 0 0 0)
        (position .75 0.5 0)
        (position 1 0.5 0)
        (position 1.1 1 0)
        (position 1.2 1 0)
        (position 1.5 1 0)
        (position 1.9 .75 0)
        (position 2 0.5 0)
        (position 2.1 .25 0)
        (position 2.5 0 0)
        (position 3 0 0)
        (position 4 0 0)))
(define knot_v
    (list  0 0 0 1 2 3 4 4 4 5 6 7 7 7 8 9 9 9))
(define spline
    (edge:spline–from–ctrlpts ctrlpts_pos knot_v) )
(entity:check spline)
;; (#[entity 1 1])
(define se (edge:split–at–disc spline))
(part:entities)
;;  (#[entity 1 1] #[entity 2 1] #[entity 3 1])
```

edge:spring

Scheme Extension: Construction Geometry, Model Object

Action: Creates an edge from a spring definition.

Filename: cstr/cstr_scm/edge_scm.cxx

APIs: api_edge_spring

Syntax:
```
(edge:spring axis–point axis–vector
    start–position handedness thread–distance
    rotation–degrees {[trans–height trans–degrees
    thread–distance rotation–degrees]...})
```

Arg Types:

axis–point	position
axis–vector	gvector
start–position	position
handedness	boolean
thread–distance	real
rotation–degrees	real
trans–height	real
trans–degrees	real

Returns: edge

Errors: None

Description: This extension links together several helix pieces and transition regions to make one edge. The transition regions between the helix pieces ensure G2 continuity. The final edge consists of a helix followed by zero or more transition/helix pairs.

The edge returned has information associated with it that indicates where the axis of the helix is located. If the edge is then swept, a rail law is created that orients the profile relative to the center axis, rather than using a minimum rotation rail. This improves the speed of sweeping, and results in the intuitive sweeping of a spring.

All helix and subsequent transition/helix pairs are explicitly listed as parameters. As a minimum, one helix (thread distance and angle) must be specified. After that, any number of transition pieces (height and angle) and helix pieces may be specified, as long as they are specified in transition/helix pairs.

The axis–point, axis–vector, and start–position arguments determine the location of the multi–helix. The radius of the helix is calculated as the distance from the start position to the axis.

The handedness argument determines the spring direction with respect to the axis vector. TRUE is right handed, and FALSE is left handed.

The rotation–degrees argument determines how far the given spring section rotates about the axis, and can be greater than one revolution (360 degrees). The thread–distance argument determines the distance between adjacent loops, assuming that the rotation is greater than one revolution.

The trans–height argument determines the distance for a transition piece, while trans–degrees dictates the rotation (in degrees) that occurs over the transition piece. Following the transition piece, another rotation section can be specified using thread–distance and rot–degrees parameters.

The specification of a transition region and second spring area is optional. When specified, all four parameters trans–height, trans–degrees, a second thread–distance, and a second rot–degrees are required. Any number of additional transition regions and additional helical sections can be specified as long as the set of all four parameters is given for each section.

Refer to Appendix D, *Spring Scheme Examples*, for additional illustrations and Scheme examples.

axis–point, axis–vector, and start–position arguments determine the location of the multi–helix.

handedness argument determines the spring direction with respect to the axis vector. TRUE is right handed, and FALSE is left handed.

thread–distance argument determines the distance between adjacent loops, assuming that the rotation is greater than one revolution.

rotation–degrees argument determines how far the given spring section rotates about the axis, and can be greater than one revolution (360 degrees).

trans–height argument determines the distance for a transition piece.

trans–degrees dictates the rotation (in degrees) that occurs over the transition piece.

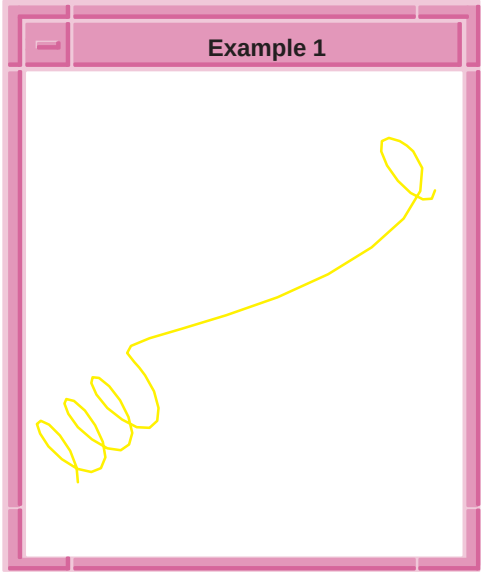
Limitations: None

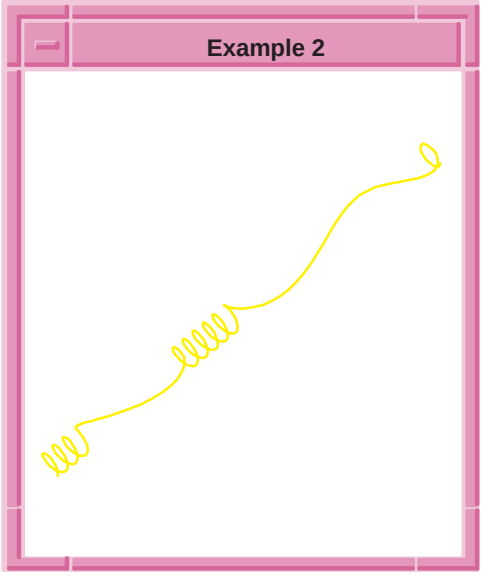
Example:

```
; edge:spring
; Create axis points.
; Set the view’s eye position, target position,
; and up vector.
(define viewset (view:set (position –50 –50 –100)
    (position 0 0 0)
    (gvector 0 0 1)))
;; viewset
(define axis–point (position 0 0 0))
;; axis–point
(define axis–vector(gvector 0 1 0))
;; axis–vector
(define start–position (position 1 0 0))
;; start–position
(define spring (edge:spring axis–point axis–vector
    start–position #t 0 270
    10 360 1 (law:eval ”3*360+90”) ))
;; spring
(define zoom (zoom–all))
;; zoom
; OUTPUT Example 1

; Additional Example
(part:clear)
;; #t
; Set the view’s eye position, target position,
; and up vector.
(define viewset (view:set (position –50 –50 –100)
    (position 0 0 0)
    (gvector 0 0 1)))
;; viewset
(define spring (edge:spring axis–point axis–vector
    start–position #t 0 270
    20 (law:eval ”2*360”) 1 (law:eval ”5*360”)
    10 360 1 (law:eval ”3*360+90”) ))
;; spring
(define zoom (zoom–all))
;; zoom
; OUTPUT Example 2
```

```
; Additional Example
(part:clear)
;; #t
; Set the view's eye position, target position,
; and up vector.
(define viewset (view:set (position –50 –50 –100)
    (position 0 0 0)
    (gvector 0 0 1)))
;; viewset
(define spring (edge:spring axis–point axis–vector
    start–position #t 0 270
    20 (law:eval "2*360") 1 (law:eval "3*360")
    20 (law:eval "2*360") 1 (law:eval "3*360")
    20 (law:eval "2*360") 1 (law:eval "5*360+90")
    20 (law:eval "2*360") 1 (law:eval "3*360")
    20 (law:eval "2*360") 1 (law:eval "3*360") ))
;; spring
(define zoom (zoom–all))
;; zoom
; OUTPUT Example 3
```





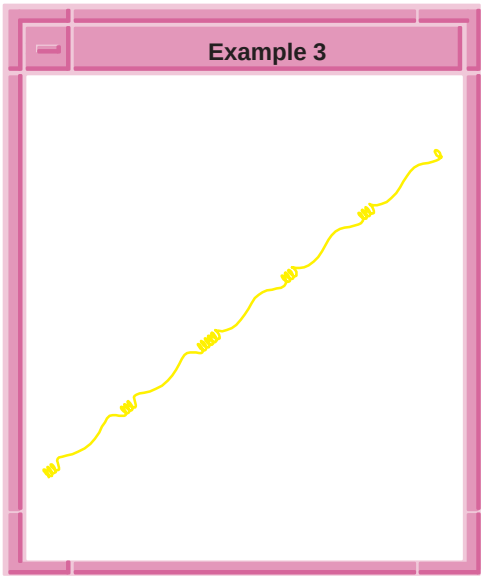


Figure 2-21. edge:spring

edge:spring–law

Scheme Extension: Construction Geometry, Laws

Action: Creates an edge from a spring definition and a radius law.

Filename: cstr/cstr_scm/edge_scm.cxx

APIs: api_edge_spring_law

Syntax:

```
(edge:spring–law axis–point axis–vector
    start–position radius–law handedness
    thread–distance rotation–degrees
    {[trans–height trans–degrees
    thread–distance rotation–degrees]...})
```

Arg Types:

axis–point	position
axis–vector	gvector
start–position	position
radius–law	law
handedness	boolean
thread–distance	real
rotation–degrees	real
trans–height	real
trans–degrees	real

Returns: edge

Errors: An incorrect law could result in a self–intersecting edge.

Description: This extension links together several helix pieces and transition regions to make one edge. This extension differs from edge:spring in that a law can be used to specify the radius of the helix. For example, a spring that expands and contracts can be created. Like edge:spring, the final edge consists of a helix followed by zero or more transition/helix pairs.

The edge returned has information associated with it that indicates where the axis of the helix is located. If the edge is then swept, a rail law is created that orients the profile relative to the center axis, rather than using a minimum rotation rail. This improves the speed of sweeping, and results in the intuitive sweeping of a spring.

All helix and subsequent transition/helix pairs are explicitly listed as parameters. As a minimum, one helix (thread distance and angle) must be specified. After that, any number of transition pieces (height and angle) and helix pieces may be specified, as long as they are specified in transition/helix pairs.

The axis–point, axis–vector, and start–position arguments determine the location of the multi–helix. The radius of the helix is calculated as the distance from the start position to the axis.

The radius–law argument allows specification of the radius as a function of the height of the spring. While this extension accepts a start position, the actual spring may start in a different position, depending on the starting value of the law function.

The handedness argument determines the spring direction with respect to the axis vector. TRUE is right handed, and FALSE is left handed.

The rotation–degrees argument determines how far the given spring section rotates about the axis, and can be greater than one revolution (360 degrees). The thread–distance argument determines the distance between adjacent loops, assuming that the rotation is greater than one revolution.

The trans–height argument determines the distance for a transition piece, while trans–degrees dictates the rotation (in degrees) that occurs over the transition piece. Following the transition piece, another rotation section can be specified using thread–distance and rot–degrees parameters.

The specification of a transition region and second spring area is optional. When specified, all four parameters trans–height, trans–degrees, a second thread–distance, and a second rot–degrees are required. Any number of additional transition regions and additional helical sections can be specified as long as the set of all four parameters is given for each section.

Refer to Appendix D, *Spring Scheme Examples*, for additional illustrations and Scheme examples.

axis–point, axis–vector, and start–position arguments determine the location of the multi–helix.

radius–law argument allows specification of the radius as a function of the height of the spring.

handedness argument determines the spring direction with respect to the axis vector.

thread–distance argument determines the distance between adjacent loops, assuming that the rotation is greater than one revolution.

rotation–degrees argument determines how far the given spring section rotates about the axis, and can be greater than one revolution (360 degrees).

trans–height argument determines the distance for a transition piece.

trans–degrees dictates the rotation (in degrees) that occurs over the transition piece.

Limitations: None

Example:

```
; edge:spring–law
; Define a variable radius spring
(define axis–start (position 0 0 0))
;; axis–start
(define axis–dir (gvector 0 0 10))
;; axis–dir
(define start (position 0 10 0))
;; start
(define radius–law (law ”5+(4/(1+((x/4)–5)^2))”))
;; radius–law
(define spring1 (edge:spring–law axis–start axis–dir
    start radius–law #t 4 3600))
;; spring1
; OUTPUT Result
```

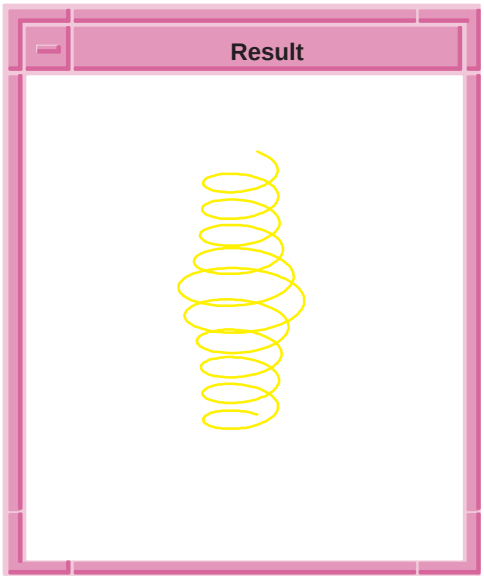


Figure 2-22. edge:spring–law

edge:spring–taper

Scheme Extension: Construction Geometry, Model Object

Action: Creates an edge from a tapered spring definition.

Filename: cstr/cstr_scm/edge_scm.cxx

APIs: api_edge_spring_taper

Syntax:

```
(edge:spring–tapte axis–point axis–vector
start–position
    taper–angle handedness thread–distance
    rotation–degrees {[trans–height trans–degrees
    thread–distance rotation–degrees]...})
```

Arg Types:		
	axis–point	position
	axis–vector	gvector
	start–position	position
	taper–angle	real
	handedness	boolean
	thread–distance	real
	rotation–degrees	real
	trans–height	real
	trans–degrees	real

Returns: edge

Errors: A negative taper–angle could result in a self–intersecting edge.

Description: This extension links together several helix pieces and transition regions to make one edge. This extension differs from edge:spring in that an angle can be used to specify the taper of the helix. For example, a conical spring can be created. Like edge:spring, the final edge consists of a helix followed by zero or more transition/helix pairs.

The edge returned has information associated with it that indicates where the axis of the helix is located. If the edge is then swept, a rail law is created that orients the profile relative to the center axis, rather than using a minimum rotation rail. This improves the speed of sweeping, and results in the intuitive sweeping of a spring.

All helix and subsequent transition/helix pairs are explicitly listed as parameters. As a minimum, one helix (thread distance and angle) must be specified. After that, any number of transition pieces (height and angle) and helix pieces may be specified, as long as they are specified in transition/helix pairs.

axis–point, axis–vector, and start–position arguments determine the location of the multi–helix. The radius of the helix is calculated as the distance from the start position to the axis.

taper–angle argument gives the angle by which the spring radius changes in relation to the height of the spring. The radius increases for positive angles and decreases for negative angles.

handedness argument determines the spring direction with respect to the axis vector. TRUE is right handed, and FALSE is left handed.

rotation–degrees argument determines how far the given spring section rotates about the axis, and can be greater than one revolution (360 degrees).

thread–distance argument determines the distance between adjacent loops, assuming that the rotation is greater than one revolution.

trans–height argument determines the distance for a transition piece.

trans–degrees dictates the rotation (in degrees) that occurs over the transition piece. Following the transition piece, another rotation section can be specified using thread–distance and rot–degrees parameters.

The specification of a transition region and second spring area is optional. When specified, all four parameters trans–height, trans–degrees, a second thread–distance, and a second rotation–degrees are required. Any number of additional transition regions and additional helical sections can be specified as long as the set of all four parameters is given for each section.

Refer to Appendix D, *Spring Scheme Examples*, for additional illustrations and Scheme examples.

Limitations: None

Example:

```
; edge:spring–taper
; Construct a tapered spring edge
(define axis–start (position 0 0 0))
;; axis–start
(define axis–dir (gvector 0 0 10))
;; axis–dir
(define start (position 0 10 0))
;; start
(define spring1 (edge:spring–taper axis–start
    axis–dir start 45 #t 5 1020 5 90 2 720))
;; spring1
; OUTPUT Result
```

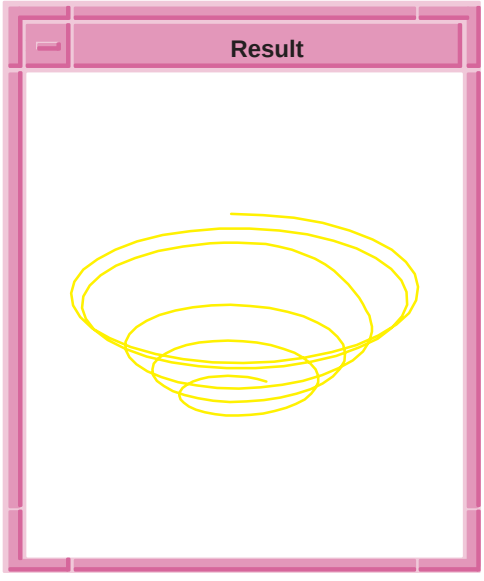


Figure 2-23. edge:spring–taper

edge:start

Scheme Extension: Construction Geometry

Action: Returns the starting position of the edge.

Filename: cstr/cstr_scm/edge_scm.cxx

APIs: None

Syntax: (**edge:start** edge)

Arg Types:	edge	entity

Returns: gvector

Errors: None

Description: Refer to Action.

edge is an input edge.

Limitations: None

Example:

```
; edge:start
; create a circle.
(define circle (edge:ellipse (position 0 0 0)
    (gvector 0 0 1) (gvector 10 0 0) 1 0 360))
;; circle
; define the starting point.
(define start–point (edge:start circle))
;; start–point
; Create/define a line.
(define line (edge:linear start–point
    (position 0 0 10)))
;; line
;Get the start point of the line.
(edge:start line)
;; #[position 10 0 0]
```

edge:start–dir

Scheme Extension: Construction Geometry

Action: Returns the start direction of the curve.

Filename: cstr/cstr_scm/edge_scm.cxx

APIs: None

Syntax: (**edge:start-dir** edge)

Arg Types: edge entity

Returns: gvector

Errors: None

Description: Refer to Action.

edge is an input edge.

Limitations: None

Example:
```
; edge:start-dir
; Create elliptical edge by specifying the center
; and start positions, the ratio, and the start and
; end angles.
(define center (position 15 15 0))
;; center
(define start (position 15 –5 0))
;; start
(define hello-edge
    (edge:elliptical center start 0.5 0 270))
;; hello-edge
; Get the start direction of the edge.
(edge:start-dir hello-edge)
;; #[gvector –10 0 0]
```

edge:tolerant

Scheme Extension: Construction Geometry, Tolerant Modeling

Action: Replaces an edge with a tolerant edge.

Filename: cstr/cstr_scm/tmod_scm.cxx

APIs: api_replace_edge_with_tedge

Syntax: (**edge:tolerant** edge)

Arg Types: edge edge

Returns: entity

Errors: None

Description: edge specifies an edge to be converted to a tolerant edge.

Limitations: None

Example:
```
; edge:tolerant
; Define a block.
(define block1 (solid:block (position 0 0 0)
    (position 50 50 50)))
;; block1
(define tol–edge (car (entity:edges block1)))
;; tol–edge
; tol–edge => #[entity 2 1]
; Define a tolerant edge
(define edge1 (edge:tolerant tol–edge))
;; edge1
```

edge:trim

Scheme Extension: Model Topology, Construction Geometry

Action: Trims an entray to the bounds of the specified position or another edge (entray).

Filename: cstr/cstr_scm/trim_scm.cxx

APIs: api_trim_curve

Syntax: (**edge:trim** entray {position | entray})

Arg Types:

entray	entray
position	position

Returns: entity

Errors: None

Description: The argument entray specifies the entity and a ray. The rays associated with the entities determine which intersection to choose for entities with multiple intersections. The ray associated with the edge to be trimmed also determines the portion of the edge to be kept.

entray specifies the entity and a ray.

position specifies the location where the edge is trimmed. The entray of the first edge is trimmed to the intersection closest to the ray.

Limitations: None

Example:

```
; edge:trim
; Set the view's eye position, target position,
; and up vector.
(define viewset (view:set (position 0 –100 0)
    (position 0 0 0)
    (gvector 0 0 1)))
;; viewset
; Create linear edge 1.
(define edge1
    (edge:linear (position –10 5 0)
    (position 30 5 0)))
;; edge1
; Create linear edge 2.
(define edge2
    (edge:linear (position 10 0 20)
    (position 10 0 –20)))
;; edge2
; Create linear edge 3.
(define edge3
    (edge:linear (position 20 0 10)
    (position –10 0 10)))
;; edge3
; OUTPUT Original

; Trim edge 1 to a particular position.
(define trim1 (edge:trim (entray edge1 (ray
    (position –10 0 0) (gvector 0 1 0)))
    (position 5 5 0)))
;; trim1
; Trim edge 2 to edge 3.
(define trim2 (edge:trim (entray edge2 (ray
    (position –10 0 0) (gvector 1 0 0)))
    (entray edge3 (ray (position 18 10 0)
    (gvector 1 0 0)))))
;; trim2
; OUTPUT Result
```

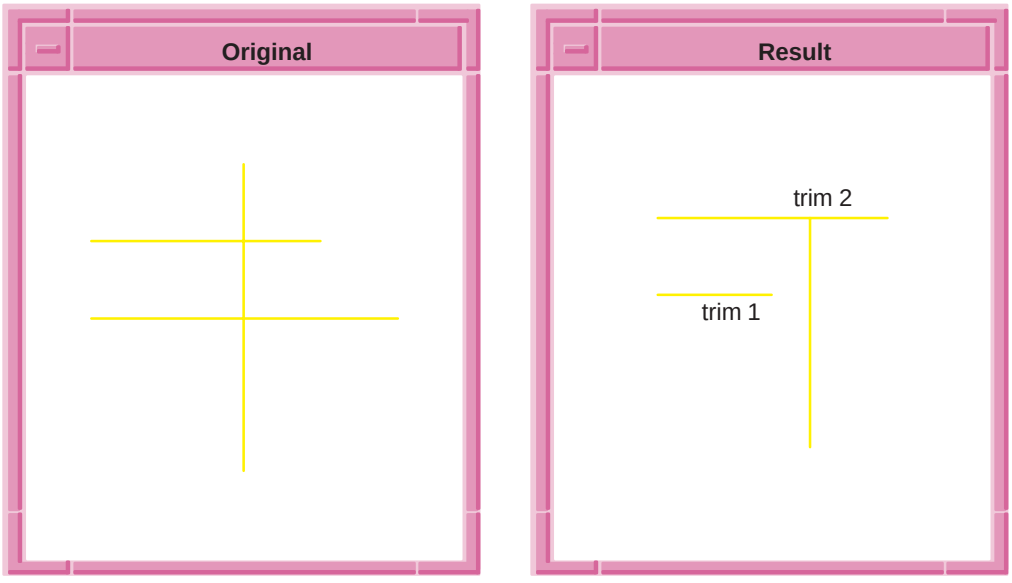


Figure 2-24. edge:trim

edge:trim–chain

Scheme Extension: Model Topology

Action: Trims a list of edges to form a continuous chain with each pair of edges having a common end point.

Filename: cstr/cstr_scm/trim_scm.cxx

APIs: api_trim_chain

Syntax: `(edge:trim–chain entray–list close)`

Arg Types:		
	entray–list	(entray ...)
	close	boolean

Returns: (entity ...)

Errors: None

Description: The argument entray–list specifies a list of edge entities and pick rays. The number of entrays in the list must be greater than two. Non–edge selection entities for inclusion in the trim are not trimmed, and their adjacent entities are not trimmed to them. Also, the first and last edge in the list are trimmed to each other. The argument close determines whether to extend the lines to form the intersection of two edges.

entray–list specifies a list of edge entities and pick rays.

close determines whether to extend the lines to form the intersection of two edges.

Note *If a spline or splines are used in the* edge:trim–chain *extension as part of the list, the splines must intersect the other edges; splines are not automatically extended.*

Limitations: None

Example:

```
; edge:trim–chain
; Set the view's eye position, target position,
; and up vector.
(define viewset (view:set (position 0 –100 0)
    (position 0 0 0)
    (gvector 0 0 1)))
;; viewset
; Create linear edge 1.
(define edge1
    (edge:linear (position 5 0 5)
    (position 20 0 5)))
;; edge1
; Create linear edge 2.
(define edge2
    (edge:linear (position 18 0 0)
    (position 18 0 20)))
;; edge2
; Create linear edge 3.
(define edge3
    (edge:linear (position 6 0 0)
    (position 15 0 15)))
;; edge3
; OUTPUT Original

; Trim the three edges to form a continuous chain.
(define trim (edge:trim–chain (list
    (entray edge1 (ray (position 17 0 5)
    (gvector 0 –1 0))) (entray edge2
    (ray (position 18 0 15) (gvector 0 –1 0)))
    (entray edge3 (ray (position 15 0 15)
    (gvector 0 –1 0)))) #t))
;; trim
; OUTPUT Result
```

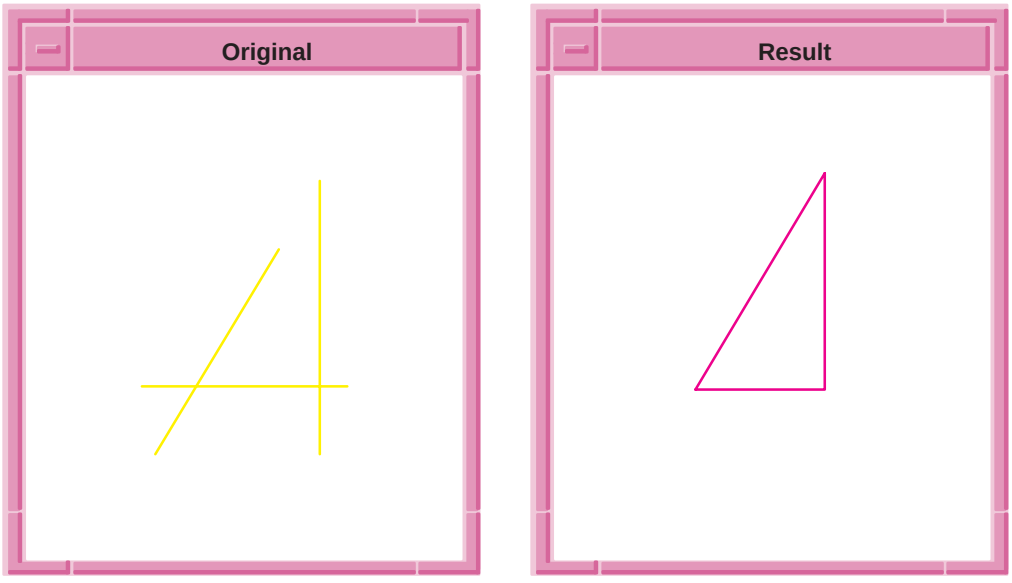


Figure 2-25. edge:trim–chain

edge:trim–intersect

Scheme Extension: Booleans, Intersectors

Action: Trims both edges to the intersection of the edges.

Filename: cstr/cstr_scm/trim_scm.cxx

APIs: api_trim_2curves

Syntax: `(edge:trim–intersect entray entray)`

Arg Types: entray entray

Returns: entity

Errors: None

Description: Trims both edges to the intersection of the edges. The arguments entray specify the edge entities and their corresponding rays. In the trimming process, the ends of the edges nearest the ray are kept.

entray specify the edge entities and their corresponding rays.

Limitations: None

Example:

```
; edge:trim–intersect
; Set the view's eye position, target position,
; and up vector.
(define viewset (view:set (position 0 –100 0)
    (position 0 0 0)
    (gvector 0 0 1)))
;; viewset
; Create linear edge 1.
(define edge1
    (edge:linear (position –10 0 0)
    (position 30 0 0)))
;; edge1
; Create linear edge 2.
(define edge2
    (edge:linear (position 0 0 –10)
    (position –8 0 30)))
;; edge2
; OUTPUT Original

; Trim the edges to their intersection point.
(define trim (edge:trim–intersect (entray edge1
    (ray (position –10 0 0) (gvector 0 –1 0)))
    (entray edge2 (ray (position 0 0 –10)
    (gvector 0 –1 0)))))
;; trim
; OUTPUT Result
```

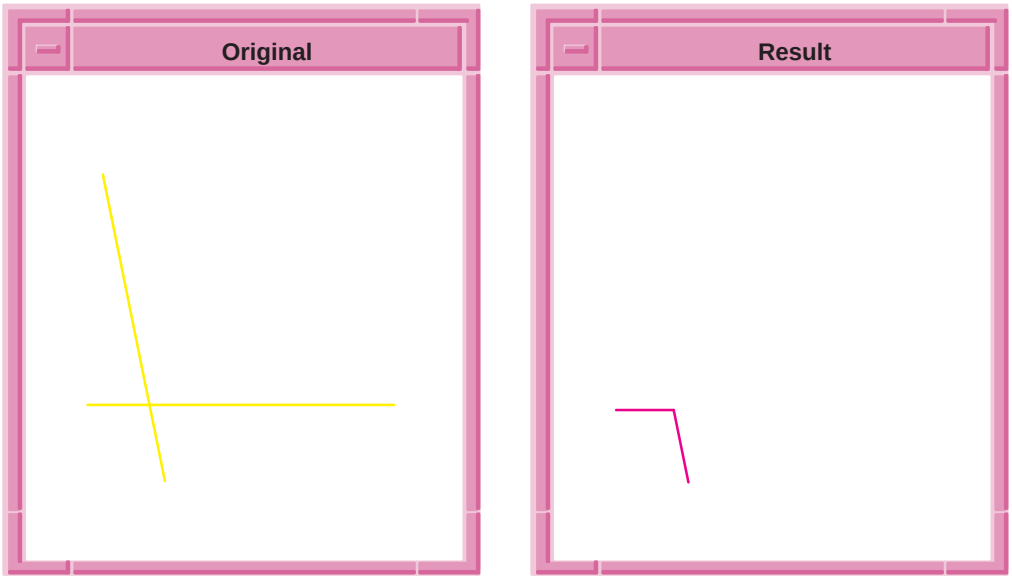


Figure 2-26. edge:trim–intersect

edge:trim–middle

Scheme Extension: Model Topology, Construction Geometry

Action: Trims the middle out of an edge.

Filename: cstr/cstr_scm/trim_scm.cxx

APIs: api_trim_middle

Syntax:
```
(edge:trim–middle entray {position | entray}
    {position | entray})
```

Arg Types:	entray position	entray position

Returns: entity

Errors: None

Description: Each entray specifies an edge entity and a ray. The first position argument specifies the start location of the edge to be trimmed. The second position argument specifies the end location of the edge to be trimmed.

entray specifies an edge entity and a ray.

position argument specifies the start location of the edge to be trimmed.

Limitations: None

Example:

```
; edge:trim–middle
; Set the view’s eye position, target position,
; and up vector.
(define viewset (view:set (position 0 –100 0)
    (position 0 0 0)
    (gvector 0 0 1)))
;; viewset
; Create a linear edge.
(define edge1
    (edge:linear (position –10 0 5)
    (position 30 0 5)))
;; edge1
; OUTPUT Original

; Trim a segment out of the middle of the edge.
(define trim (edge:trim–middle (entray edge1
    (ray (position 6 0 5) (gvector 0 –1 0)))
    (position 6 0 5) (position 15 0 5)))
;; trim
; OUTPUT Result
```

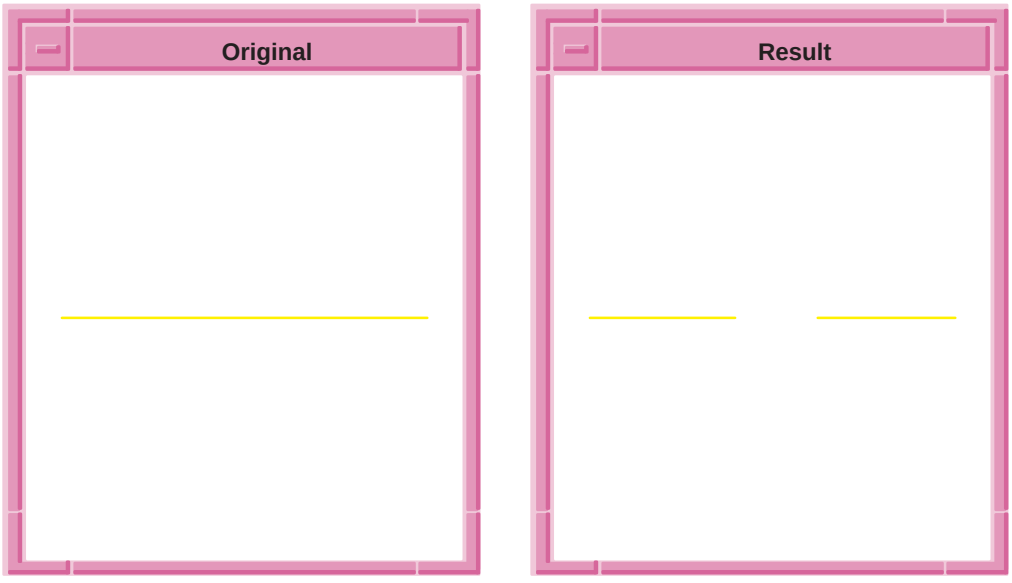


Figure 2-27. edge:trim–middle

entity:accurate–approx

Scheme Extension: Model Geometry

Action: Regenerates bs3 surfaces, using a slower but more robust approach.

Filename: cstr/cstr_scm/face_scm.cxx

APIs: api_accurate_bs3_approximation, api_get_faces

Syntax: `(entity:accurate–approx entity [fit=resfit])`

Arg Types:

entity	entity | face
fit	real

Returns: (entity | entity...) | (face | face...)

Errors: None

Description: This command recomputes the bs3 approximation of all spline surfaces on the input entity. If the fit is nonpositive or unspecified, resfit is used.

entity is an input entity.

Limitations: None

Example:

```
; entity:accurate–approx
; Create an entity
(define block1 (solid:block (position 10 20 30)
    (position 0 0 –30)))
;; block1
(define pos1 (position 5 10 20))
;; pos1
(define pos2 (position 5 10 –20))
;; pos2
(define theta2 (law:eval ”2*360”))
;; theta2
(define twist1 (entity:twist block1
    pos1 pos2 0 theta2 2))
;; twist1
(define approx (entity:accurate–approx twist1))
;; approx
```

entity:box

Scheme Extension: Model Object, Geometric Analysis

Action: Gets an entity’s extrema box.

Filename: cstr/cstr_scm/sld_scm.cxx

APIs: api_get_entity_box, api_wcs_get_active

Syntax: (**entity:box** entity–list [create–box])

Arg Types:		
	entity–list	entity | (entity ...)
	create–box	boolean

Returns: pair

Errors: None

Description: This extension obtains the minimum and maximum values corresponding to a diagonal across the body’s bounding box relative to the active working coordinate system. The argument entity–list comprises any entity or list of entities. Each entity or list of entities may be a body, wire, face, or edge. This extension returns a pair consisting of (min–pt . max–pt). If create–box is given a solid box will be returned.

entity–list comprises any entity or list of entities.

create–box is a boolean argument, if given, a solid box will be returned.

Limitations: None

Example:
```
; entity:box
; Create solid block 1.
(define block1
    (solid:block (position 0 0 0)
    (position 10 10 10)))
;; block1
; Determine the bounding box of solid block 1.
(entity:box block1)
;; (#[position 0 0 0] . #[position 10 10 10])
; Create solid block 2.
(define block2
    (solid:block (position 5 5 5)
    (position 25 25 25)))
;; block2
; Determine the bounding box of solid block 2.
(entity:box block2)
;; (#[position 5 5 5] . #[position 25 25 25])
; Determine the bounding box of solid blocks 1 and 2.
(entity:box (list block1 block2))
;; (#[position 0 0 0] . #[position 25 25 25])
```

entity:tolerize

Scheme Extension: Debugging

Action: Finds all the edges and vertices in a part that are not accurate to within a given tolerance, and turns them into tolerant entities.

Filename: cstr/cstr_scm/tmod_scm.cxx

APIs: api_check_edge_errors, api_check_vertex_errors, api_get_edges, api_get_vertices

Syntax: (**entity:tolerize** entity)

Arg Types:	entity	entity

Returns: entity | (entity...)

Errors: None

Description: If the tolerance is omitted, resabs is used. Reports how many edges and vertices are checked and highlights anything 'tolerized'.

entity is an input entity.

Limitations: None

Example:
```
; entity:tolerize
; Create topology to demonstrate command.
(define block (solid:block (position –10 –10 –10)
    (position 10 10 10)))
;; block
(define edge (pick:edge (ray (position 0 0 0)
    (gvector 1 1 0))))
;; edge
(define tedge (edge:tolerant edge))
;; tedge
; Move one edge away from block.
(define move (tolerant:move tedge (gvector 1 1 0)))
;; move
; note, this corrupts the block.
(define tolerant (tolerant:none tedge))
;; tolerant
; now fix the block
(define tolerize (entity:tolerize block))
;; tolerize
; check block information
(define check (entity:check block))
;; check
(define report (tolerant:report block))
;; report
```

face:area

Scheme Extension: Physical Properties

Action: Gets the area of a face.

Filename: cstr/cstr_scm/face_scm.cxx

APIs: api_ent_area

Syntax: (**face:area** face)

Arg Types:	face	face

Returns: real

Errors: None

Description: This extension calculates the area of a given face. The accuracy of the calculation is fixed at 0.001 for an area of geometry that cannot be determined analytically.

face is an input face.

Limitations: None

Example:
```
; face:area
; Create face 1.
(define face1 (face:plane (position 0 0 0) 30 30))
;; face1
; Determine the area of face1.
(face:area face1)
;; 900
; Create face 2.
(define face2
    (face:cylinder (position 0 0 0)
    (position 0 30 0) 10 0 180))
;; face2
; Determine the area of face2.
(face:area face2)
;; 942.477796076938
```

face:cone

Scheme Extension: Model Object

Action: Creates a conical face relative to the active WCS.

Filename: cstr/cstr_scm/face_scm.cxx

APIs: api_face_cylinder_cone

Syntax:
```
(face:cone position1 position2 radius1 radius2
    [start–angle=0 end–angle=360 ratio position3])
```

Arg Types:		
	position1	position
	position2	position
	radius1	real
	radius2	real
	start–angle	real
	end–angle	real
	ratio	real
	position3	position

Returns: face

Errors: None

Description: This extension creates a conical face relative to the active WCS.

position1 and position2 are opposite ends of the cone axis and define the cone height.

radius1 is the cone radius at the base

radius2 is the cone radius at the top. If radius2 is greater than 0, the object created is a frustum.

start–angle and end–angle angles default to 0 and 360 degrees, creating a completed conical face. A face of a quarter cone may be defined, for example, by specifying 0 for start–angle and 90 for end–angle. The angles must lie in the range of –360 to+ 360 degrees.

ratio allows the creation of an elliptical conical face. radius1 becomes the major axis and a minor axis is determined by the ratio (major/minor).

The cone axis defines an alternate *z*–axis (as described in face:plane). This *z*–axis defines the top and bottom planes that become the top and bottom elliptical faces of the cone. The start (start–angle) and end (end–angle) angles and the elliptical major axis are based from the alternate *x*–axis.

position3 if specified, the *x*–axis becomes the line drawn to the projection of this point on the top and bottom planes. The cone axis defines the top and bottom planes; therefore, the *x*–axis simply rotates about the *z*–axis until it aligns with position3.

Limitations: None

Example:

```
; face:cone
; Create conical face 1.
(define face1
    (face:cone (position 0 0 0)
    (position 0 50 0) 30 10))
;; face1
; Create conical face 2.
(define face2
    (face:cone (position 0 0 0)
    (position 0 40 0) 40 20 0 135))
;; face2
; OUTPUT Example
```

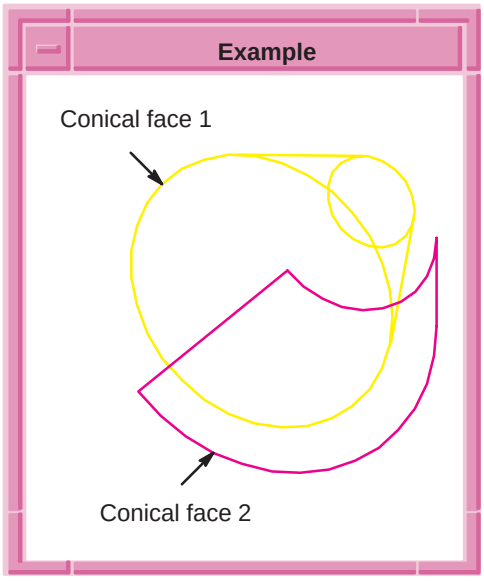


Figure 2-28. face:cone

face:conic

Scheme Extension: Model Object, Ray Testing

Action: Creates a conic surface with a given radius R, a conic constant K, and an extent E such that x^2+y^2<E.

Filename: cstr/cstr_scm/face_scm.cxx

APIs: api_face_conic

Syntax: `(face:conic radius conic–constant extent [length])`

Arg Types:		
	radius	real
	conic–constant	real
	extent	real
	length	real

Returns: entity

Errors: None

Description: This creates a parabolic dish using law surfaces. It automatically creates the required helper laws. This makes ray firing for such a surface significantly faster.

The radius is measured at the vertex. The conic–constant is the constant for the conic. The extent limits the conic size such that x^2 + y^2 < extent, or if the length is greater than 0, the y^2 < extent. when the length is specified and is greater than 0, then a conic trough is formed.

radius is the input radius measured at the vertex.

conic–constant is the constant for the conic.

extent limits the conic size such that x^2 + y^2 < extent.

length if greater than 0, the y^2 < extent. when the length is specified and is greater than 0, then a conic trough is formed.

Limitations: None

Example:
```
; face:conic
; This makes a half sphere of radius 10
(define face1 (face:conic 10 0 9))
;; face1
```

face:cylinder

Scheme Extension: Model Object

Action: Creates a cylindrical face relative to the active WCS.

Filename: cstr/cstr_scm/face_scm.cxx

APIs: api_face_cylinder_cone

Syntax:
```
(face:cylinder position1 position2 radius
    [start–angle=0 end–angle=360 ratio position3])
```

Arg Types:

position1	position
position2	position
radius	real
start–angle	real
end–angle	real
ratio	real
position3	position

Returns: face

Errors: None

Description: This extension creates a cylindrical face relative to the active WCS.

position1 and position2 are opposite ends of the cylinder axis and define the cylinder height. The cylinder radius is self–explanatory.

start–angle and end–angle angles default to 0 and 360 degrees creating a completed cylindrical face. A face of a quarter cylinder may be defined, for example, by specifying 0 for start–angle and 90 for end–angle. The angles must lie in the range of –360 to 360.

ratio allows the creation of an elliptical cylindrical face. The radius becomes the major axis and a minor axis is determined by the ratio (major/minor).

The cylinder axis defines an alternate *z*–axis as previously described in face:plane. This *z*–axis defines the top and bottom planes that become the top and bottom elliptical faces of the cylinder. The start (start–angle) and end (end–angle) angles and the elliptical major axis are based from the alternate *x*–axis.

position3 if specified, the *x*–axis becomes the line drawn to the projection of this point on the top and bottom planes. The cylinder axis defines the top and bottom planes; therefore, *x*–axis simply rotates about the *z*–axis until it aligns with position3.

Limitations: None

Example:

```
; face:cylinder
; Create cylindrical face 1.
(define face1
    (face:cylinder (position 0 0 0)
    (position 0 50 0) 10))
;; face1
; Create cylindrical face 2.
(define face2
    (face:cylinder (position 0 0 0)
    (position 0 40 0) 15 45 135))
;; face2
; OUTPUT Example
```

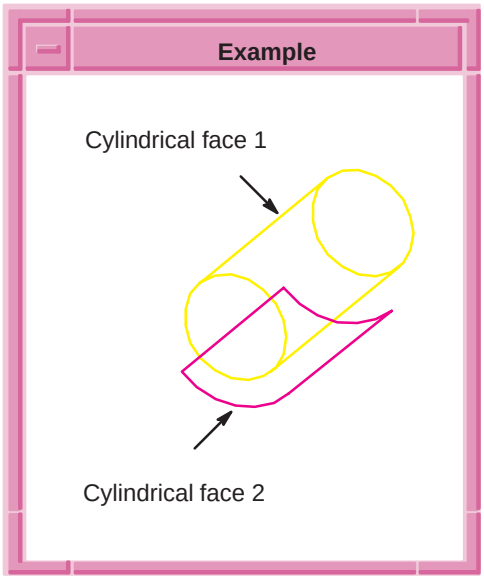


Figure 2-29. face:cylinder

face:extend

Scheme Extension: Model Object

Action: Creates a new face that is an extension of an existing face.

Filename: cstr/cstr_scm/face_scm.cxx

APIs: None

Syntax: `(face:extend entity start–u end–u start–v end–v)`

Arg Types:		
	entity	face
	start–u	real
	end–u	real
	start–v	real
	end–v	real

Returns: entity

Errors: None

Description: This extends the bounding parameters of the underlying surface of the given face entity, entity, based on the supplied new starting and ending parameters for *u* and *v*. It returns an entity which is a new face.

To do this, face:extend creates a law from the surface and constant laws for the parameter values. The constant laws are turned into vectors which are then combined with the surface law. They create curve laws as the bounding edges of a new surface, and are then used to create a new face.

entity is an input entity.

start–u is the starting parameter in the u direction.

end–u is the ending parameter in the u direction.

start–v is the starting parameter in the v direction.

end–v is the ending parameter in the v direction.

Limitations: None

Example:

```
; face:extend
; Extend an existing face.
(define face1 (face:plane (position 0 0 0) 30 30))
;; face1
(define extend (face:extend face1 0 40 –10 25))
;; extend
```

face:law

Scheme Extension: Laws

Action: Creates a law face.

Filename: cstr/cstr_scm/face_scm.cxx

APIs: api_face_law

Syntax: (**face:law** law start–u end–u start–v end–v)

Arg Types:		
	law	law
	start–u	real
	end–u	real
	start–v	real
	end–v	real

Returns: face

Errors: None

Description: This extension creates a law face with the given *uv* bounds from a law that takes two values and returns three.

law is of the type law.

start–u is the starting parameter in the u direction.

end–u is the ending parameter in the u direction.

start–v is the starting parameter in the v direction.

end–v is the ending parameter in the v direction.

Limitations: None

Example:
```
; face:law
; Create a face from a law with a domain of dimension
(define face-law1
    (face:law
    "vec(cos(x),sin(x),x/5)" -10 10 -10 10))
;; face-law1
```

face:par–box

Scheme Extension: Physical Properties

Action: Gets the parameter ranges of a face.

Filename: cstr/cstr_scm/face_scm.cxx

APIs: None

Syntax: (**face:par-box** face)

Arg Types:	face	face
Returns:	par–pos par–pos	

Errors: None

Description: This extension calculates the *u* and *v* parameter ranges of a given face.

face is an input face argument.

Limitations: None

Example:

```
; face:par–box
; Create face 1.
(define face1 (face:plane (position 0 0 0) 30 30))
;; face1
; Determine the parameters of face1.
(face:par–box face1)
;; (#[par–pos 0 30] . #[par–pos 0 30])
; Clear for 2nd example.
(part:clear)
;; #t
; Create face 2.
(define face2
    (face:cylinder (position 0 0 0)
    (position 0 30 0) 10 0 180))
;; face2
; Determine the parameters of face2.
(face:par–box face2)
;; (#[par–pos 0 3] . #[par–pos 0 3.14159265358979])
```

face:planar–disk

Scheme Extension: Model Object

Action: Creates a planar disk of a given center, normal, and radius.

Filename: cstr/cstr_scm/face_scm.cxx

APIs: api_make_planar_disk

Syntax: (**face:planar–disk** center normal radius)

Arg Types:

center	position
normal	gvector
radius	real

Returns: face

Errors: None

Description: This extension creates a planar disk of a given center, normal, and radius.

center is the center position of a disk.

normal is a direction vector.

radius is the disk radius.

Limitations: None

Example:
```
; face:planar–disk
; Create a planar disk
(define pdisk
    (face:planar–disk
    (position 0 0 0) (gvector 0 0 10)10))
;; pdisk
```

face:plane

Scheme Extension: Model Object

Action: Creates a face that is a parallelogram specified by three points: origin, left, and right.

Filename: cstr/cstr_scm/face_scm.cxx

APIs: api_face_plane

Syntax: (**face:plane** position width height [normal])

Arg Types:		
	position	position
	width	real
	height	real
	normal	gvector

Returns: face

Errors: A plane cannot have a zero width or height. This means that both (left – origin) and (right – origin) must be greater than resabs.

A plane cannot have a zero length normal. This means that the length of ((left – origin) * (right – origin)) must be greater than resabs.

Description: This extension creates a planar face of the given width and height, relative to the active WCS, with its lower left–hand corner at the given position.

normal becomes the *z*–axis of an alternate coordinate system, (*xyz*). If the *z*–axis and the active *x*–axis are parallel, the new *y*–axis is the active *y*–axis. If they are not parallel, the *y*–axis is the normal to the plane defined by the *z*–axis and the *x*–axis. The *x*–axis is then determined as the normal to the plane defined *y*–axis and the *z*–axis.

If the normal is in the direction of the z–axis, then the width and height are in the direction of the WCS x–axis and y–axis respectively. Orientation of width and height in any other case is not guaranteed.

position is the input position argument.

width is the width of the planar face.

height is the height of the planar face.

normal becomes the *z*–axis of an alternate coordinate system, (*xyz*).

Limitations: None

Example:

```
; face:plane
; Create planar face 1.
(define face1 (face:plane (position 0 0 0) 30 30))
;; face1
; Create planar face 2.
(define face2
    (face:plane (position 0 0 30) 30 30))
;; face2
; Create planar face 3.
(define face3
    (face:plane
    (position 0 0 30) 30 30 (gvector 1 0 0)))
;; face3
; Create planar face 4.
(define face4
    (face:plane
    (position 30 0 30) 30 30 (gvector 1 0 0)))
;; face4
; OUTPUT Example
```

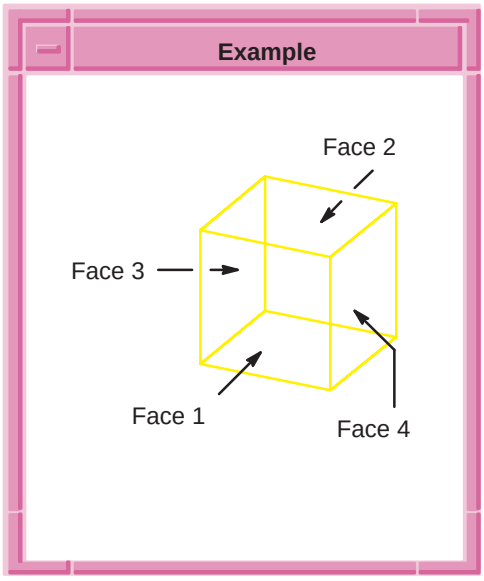


Figure 2-30. face:plane

face:prop

Scheme Extension: Physical Properties

Action: Analyzes the properties of a face.

Filename: cstr/cstr_scm/face_scm.cxx

APIs: api_ent_area, api_planar_face_pr

Syntax: `(face:prop face [tolerance=0.01])`

Arg Types:	face	face
	tolerance	real

Returns: (position | gvector | real | integer ...)

Errors: None

Description: For a planar face this extension calculates the center of area, principal moments, and principal axes. For all types of faces it also returns the area and sidedness of a face. The results are in a global coordinate space of an owning body. Use tolerance to specify an accuracy other than the default accuracy of 0.01. The accuracy achieved will be returned.

face is a input planar face.

tolerance to specify an accuracy other than the default accuracy of 0.01.

Limitations: None

Example:

```
; face:prop
; Create planar face 1.
(define face1 (face:plane (position 0 0 0)
    40 40 (gvector 1 –1 –1)))
;; face1
; Calculate the properties of a planar face.
(face:prop face1)
; Face is SINGLE_SIDED and FORWARD.
; PLANAR face properties:
;; ((”area” . 1600.0) (”accuracy achieved” . 0)
;; (”center of face” . #[position 16.3299316185545
;; –5.97716981445369 22.3071014330082]) (”principal
;; moment x” . 213333.333333334) (”principal axis x”
;; . #[gvector –0.707106781186548 0
;; –0.707106781186548]) (”principal moment y” .
;; 213333.333333333) (”principal axis y” . #[gvector
;; 0.408248290463863 0.816496580927726
;; –0.408248290463863]))
; Create planar face 2.
(define face2 (face:sphere (position 10 10 10) 20))
;; face2
; Calculate the properties of a non–planar face.
(face:prop face2)
; Face is SINGLE_SIDED and FORWARD.
; NON–PLANAR face properties:
;; ((”area” . 5026.54824574367) (”accuracy achieved”
;; . 0) (”center of face” . #f) (”principal moment x”
;; . #f) (”principal axis x” . #f) (”principal moment
;; y” . #f) (”principal axis y” . #f))
```

face:reverse

Scheme Extension: Model Topology

Action: Reverses the sense of a face.

Filename: cstr/cstr_scm/face_scm.cxx

APIs: api_reverse_face

Syntax: (**face:reverse** face)

Arg Types: face face

Returns: face

Errors: None

Description: This extension reverses the sense of the given face. The sense is used in conjunction with the surface normal to determine which side of the face is solid material, and is applicable only to single–sided faces of solids. This extension affects the sense of the face's coedges.

face is an input argument face.

Limitations: None

Example:

```
; face:reverse
; Create a plane
(define face1 (face:plane (position 0 0 0) 10 10))
;; face1
(face:plane–normal face1)
;; #[gvector 0 0 1]
(define reverse (face:reverse face1))
;; reverse
(face:plane–normal face1)
;; #[gvector 0 0 –1]

; Example 2
; Create a solid cylinder.
(define cyl1
    (solid:cylinder (position 0 0 0)
    (position 0 8 0) 10))
;; cyl1
; Get a list of the faces for the cylinder.
(define faces1 (entity:faces cyl1))
;; faces1
; (#[entity 4 1] #[entity 5 1] #[entity 6 1])
; Reverse the sense of the face.
(define reverse (face:reverse (list–ref faces1 0)))
;; reverse
```

face:sphere

Scheme Extension: Model Object

Action: Creates a spherical face.

Filename: cstr/cstr_scm/face_scm.cxx

APIs: api_face_sphere

Syntax:
```
(face:sphere position radius
    [latitude-start=-90 [latitude-end=90
    [longitude-start=0 [longitude-end=360
[poledir]]]]])
```

Arg Types:		
	position	position
	radius	real
	latitude–start	real
	latitude–end	real
	longitude–start	real
	longitude–end	real
	poledir	vector

Returns: face

Errors: None

Description: This extension creates a spherical face relative to the active WCS. The center is the center of a sphere with the given radius.

angle1 and angle2 make up the latitude and angle3 and angle4 make up the longitude. Latitude moves from the equator 90 degrees to the North Pole or from the equator –90 degrees to the South Pole. Longitude moves from the zero meridian to the east 360 degrees (one complete revolution of the sphere). The default angles are latitude, –90 to 90, and longitude, 0 to 360. A partial sphere may be created, for example, by setting the latitude angles to 0 and 90, which results in the Northern hemisphere.

The north pole points in the direction of the active *y*–axis; the equator lies in the *xz* plane; and the zero meridian lies in the *xy* plane. The zero meridian begins at a positive radius length on the *x*–axis. Setting the poledir vector creates an alternate *z*–axis as described in face:plane.

position is the input position argument.

radius is the radius of the sphere.

latitude–start is the starting parameter of the latitude.

latitude–end is the ending parameter of the latitude.

longitude–start is the starting parameter of the longitude.

longitude–end is the ending parameter of the longitude.

poledir vector creates an alternate z–axis as described in face:plane.

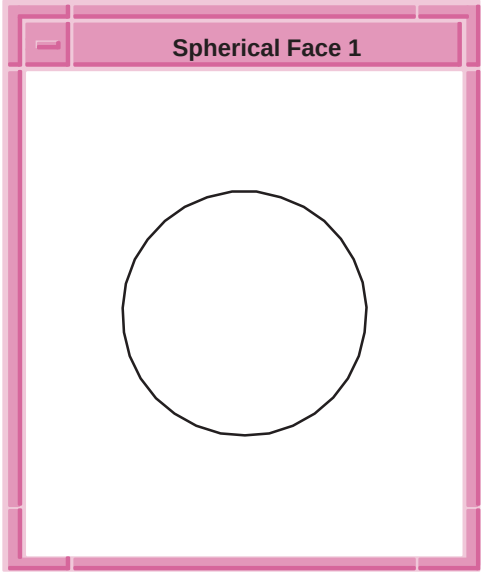
Limitations: None

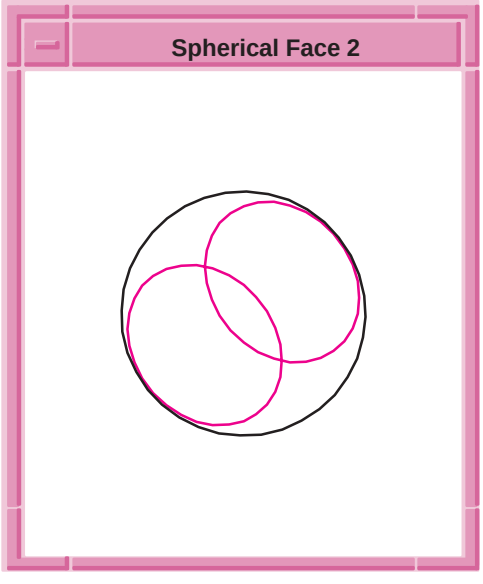
Example:

```
; face:sphere
; Create spherical face 1.
(define face1 (face:sphere (position 0 0 0) 30))
;; face1
; OUTPUT Spherical Face 1

; Clear the first face
(part:clear)
;; #t
; Create spherical face 2.
(define face2
    (face:sphere (position 0 0 0) 40 –45 45 0 360))
;; face2
; OUTPUT Spherical Face 2

; Render to show the face.
(render)
;; ()
; OUTPUT Rendered Face
```





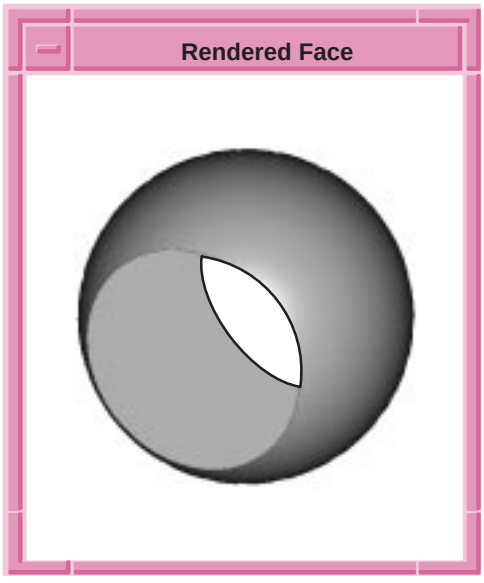


Figure 2-31. face:sphere

face:spline–ctrlpts

Scheme Extension: Spline Interface

Action: Creates a spline face using control points.

Filename: cstr/cstr_scm/face_scm.cxx

APIs: api_face_spl_ctrlpts

Syntax: `(face:spline–ctrlpts surface–data)`

Arg Types:

surface–data	splsurf

Returns: face

Errors: None

Description: This extension creates a spline surface face with the given surface control points, weights, knot data, and parametric values.

surface–data is the input surface data values.

Limitations: None

Example:

```
; face:spline-ctrlpts
; Create a spline surface data type.
(define surf1 (splsurf))
;; surf1
; Set the control points.
(define ctrlpoints1 (list
    (position -10 -10 20) (position -5 0 10)
    (position -20 20 30) (position -20 -15 10)
    (position -30 -10 0) (position -30 25 -30)
    (position 0 -10 -20) (position 5 -12 -30)
    (position 0 15 15) (position 30 -6 -5)
    (position 25 -10 0) (position 30 25 -30)
    (position 10 -10 10) (position 25 -10 20)
    (position 20 20 30)))
;; ctrlpoints1
(splsurf:set-ctrlpt-list surf1 ctrlpoints1 5 3)
;; #[splsurf bc4700]
; Set the u and v parameter values.
(splsurf:set-u-param surf1 3 0 0 0)
;; #[splsurf bc4700]
(splsurf:set-v-param surf1 2 0 0 0)
;; #[splsurf bc4700]
; set the u and v knot values.
(define uknots (list 0 0 0 0 0.5 1 1 1 1))
;; uknots
(splsurf:set-u-knot-list surf1 uknots 9)
;; #[splsurf bc4700]
(define vknots (list 0 0 0 1 1 1))
;; vknots
(splsurf:set-v-knot-list surf1 vknots 6)
;; #[splsurf bc4700]
; Create the spline face.
(define spline (face:spline-ctrlpts surf1))
;; spline
; OUTPUT Example
```

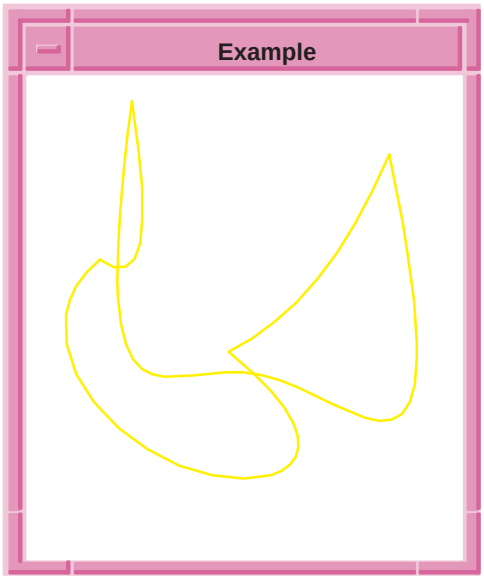


Figure 2-32. face:spline–ctrlpts

face:spline–grid

Scheme Extension: Spline Interface

Action: Creates a spline face using a grid.

Filename: cstr/cstr_scm/face_scm.cxx

APIs: api_face_spl_apprx, api_face_spl_intrp

Syntax: `(face:spline-grid grid-data interpolate)`

Arg Types:		
	grid–data	splgrid
	interpolate	boolean

Returns: face

Errors: None

Description: This extension creates a spline surface face by interpolating (#t) or approximating (#f) the b–spline surface given a grid of points (grid–data) on the surface.

grid–data is the input grid data values.

interpolate is a boolean argument which can be set to interpolating the b–spline surface.

Limitations: None

Example:

```
; face:spline–grid
; Create a spline grid data type.
(define grid (splgrid))
;; grid
(define points1 (list
    (position –50 –50 0) (position 0 –50 50)
    (position 50 –50 0) (position –50 0 50)
    (position 0 0 100) (position 50 0 50)
    (position –50 50 0) (position 0 50 50)
    (position 50 50 0)))
;; points1
(splgrid:set–point–list grid points1 3 3)
;; #[splgrid bc1510]
; Set the start and end tangent vector lists for u.
(define vectors1
    (list (gvector 1 0 1)
    (gvector 1 0 1) (gvector 1 0 1)))
;; vectors1
(splgrid:set–u–tanvec–list grid vectors1 #t)
;; #[splgrid bc1510]
(set! vectors1
    (list (gvector 1 0 –1)
    (gvector 1 0 –1)
    (gvector 1 0 –1)))
;; (#[gvector 1 0 1] #[gvector 1 0 1]
;; #[gvector 1 0 1])
(splgrid:set–u–tanvec–list grid vectors1 #f)
;; #[splgrid bc1510]
; Set the start and end tangent vector lists for v.
(define vectors2
    (list (gvector 0 1 1)
    (gvector 0 1 1) (gvector 0 1 1)))
;; vectors2
(splgrid:set–v–tanvec–list grid vectors2 #t)
;; #[splgrid bc1510]
(set! vectors2
    (list (gvector 0 1 –1)
    (gvector 0 1 –1) (gvector 0 1 –1)))
;; (#[gvector 0 1 1] #[gvector 0 1 1]
;; #[gvector 0 1 1])
```

```
(splgrid:set-v-tanvec-list grid vectors2 #f)
;; #[splgrid bc1510]
; Interpolate the B-spline surface.
(define grid1 (face:spline-grid grid #t))
;; grid1
(zoom-all))
;; ()
; Approximate the B-spline surface.
(splgrid:set-tolerance grid 0.01)
;; #[splgrid bc1510]
(define grid2 (face:spline-grid grid #f))
;; grid2
; OUTPUT Example
```

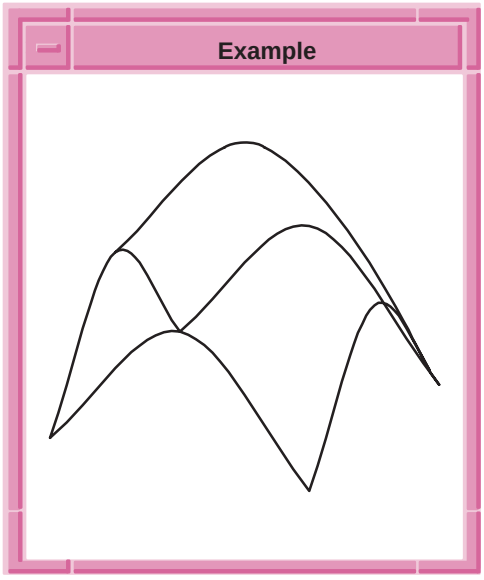


Figure 2-33. face:spline–grid

face:torus

Scheme Extension: Construction Geometry

Action: Creates a toroidal face.

Filename: cstr/cstr_scm/face_scm.cxx

APIs: api_face_torus

Syntax:

```
(face:torus position spine–radius tube–radius
    [tube–start=0 tube–end=360 spine–start=0
    spine–end=360 vector])
```

Arg Types:

position	position
spine–radius	real
tube–radius	real
tube–start	real
tube–end	real
spine–start	real
spine–end	real
vector	vector

Returns: face

Errors: None

Description: Creates a toroidal face relative to the active WCS. The torus axis is in the direction of the active *y*–axis.

position is the center of the torus.

spine–radius or major radius is the radius of the spine.

tube–radius or minor radius is the radius of the tube, defined by the positive *x*–axis with the spine in the *xz* plane

tube–start is the starting angle of the tube, defined by the positive *x*–axis with the spine in the *xy* plane. All angles are measured counter–clockwise and respect the right–hand rule.

spine–end. is the end angle for the spine.

tube–end is the end angle for the tube.

vector creates an alternate *z*–axis as described in face:plane.

Limitations: None

Example:

```
; face:torus
; Create toroidal face 1.
(define face1 (face:torus (position 0 0 0) 30 10))
;; face1
; Create toroidal face 2.
(define face2
    (face:torus
    (position –10 –10 –10)
    10 20 –90 90 0 180))
;; face2
; OUTPUT Example
```

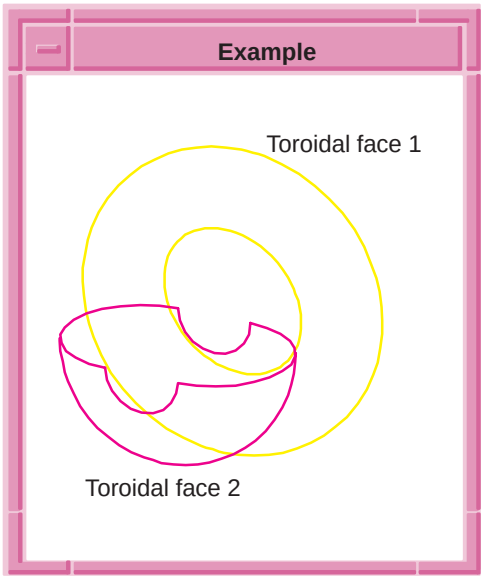


Figure 2-34. face:torus

line:set–direction

Scheme Extension: Model Geometry

Action: Sets the direction of a linear–curve or edge.

Filename: cstr/cstr_scm/geom_scm.cxx

APIs: api_get_curve_ends, api_modify_line

Syntax: `(line:set-direction curve direction)`

Arg Types:		
	curve	linear–curve | linear–edge
	direction	gvector

Returns: linear–curve | linear–edge

Errors: None

Description: curve specifies a linear–curve or edge.

direction is a gvector that specifies the new direction of the line.

Limitations: None

Example:

```
; line:set–direction
; Create a linear curve.
(define curve1
    (curve:linear (position 0 0 0)
    (position 40 40 40)))
;; curve1
; Set the direction of a linear curve.
(line:set–direction curve1 (gvector 0 1 0))
;; #[curve 401b75a8]
; Perform same example with an edge
(define edge1
    (edge:linear (position 10 0 0)
    (position 40 40 40)))
;; edge1
; Set the direction of a linear edge.
(define direction (line:set–direction edge1
    (gvector 0 1 0)))
;; direction
```

line:set–end

Scheme Extension: Model Geometry

Action: Sets the end position of a linear–curve or edge.

Filename: cstr/cstr_scm/geom_scm.cxx

APIs: api_get_curve_ends, api_modify_line

Syntax: (**line:set–end** curve position)

Arg Types:	curve	linear–curve | linear–edge
	position	position

Returns: linear–curve | linear–edge

Errors: None

Description: curve specifies a linear–curve or edge.

position specifies the end position of the line.

Limitations: None

Example:

```
; line:set–end
; Create a linear curve.
(define curve1
    (curve:linear (position 0 0 0)
    (position 40 40 40)))
;; curve1
; Set the end position of a linear curve.
(line:set–end curve1 (position 35 35 35))
;; #[curve 401b75a8]
; Perform same example with an edge
(define edge1
    (edge:linear (position 10 0 0)
    (position 40 40 40)))
;; edge1
; Set the end position of a linear edge.
(define setend (line:set–end edge1
    (position 35 35 35)))
;; setend
```

line:set–start

Scheme Extension: Model Geometry

Action: Sets the start position of a linear–curve or edge.

Filename: cstr/cstr_scm/geom_scm.cxx

APIs: api_get_curve_ends, api_modify_line

Syntax: (**line:set–start** curve position)

Arg Types:	curve	linear–curve | linear–edge
	position	position

Returns: linear–curve | linear–edge

Errors: None

Description: curve specifies a linear–curve or edge.

position modifies the start position of the line.

Limitations: None

Example:

```
; line:set-start
; Create a linear curve.
(define curve1
    (curve:linear (position 0 0 0)
    (position 40 40 40)))
;; curve1
; Set the start position of a linear curve.
(line:set-start curve1 (position 20 20 20))
;; #[curve 401b75a8]
; Perform same example with an edge
(define edge1
    (edge:linear (position 10 0 0)
    (position 40 40 40)))
;; edge1
; Set the start position of a linear edge.
(define setstart (line:set-start edge1
    (position 20 20 20)))
;; setstart
```

loop:external?

Scheme Extension: Model Geometry

Action: Determines if a loop is internal or external.

Filename: cstr/cstr_scm/loop_cstr_scm.cxx

APIs: None

Syntax: (**loop:external?** loop)

Arg Types: loop loop

Returns: boolean

Errors: The argument loop is not a LOOP.

Description: Refer to Action.

loop is an input argument of type LOOP.

Limitations: None

Example:

```
; loop:external?
; make a block with a hole in it
(define body (solid:block
    (position 0 0 0) (position 20 5 20)))
;; body
(define tool (solid:cylinder 10 0 10 10 5 10 2))
;; tool
(define new_body (solid:subtract body tool))
;; new_body
; identify the top face and its list of two loops
(define top_face (list–ref (entity:faces body) 5))
;; top_face
(define top_loops (entity:loops top_face))
;; top_loops
; one of the loops is internal, one is external
(loop:external? (list–ref top_loops 0))
;; #f
(loop:external? (list–ref top_loops 1))
;; #t
```