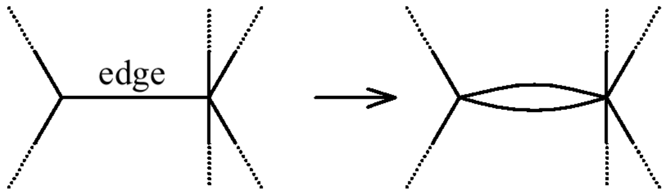
Parasolid Euler Operators

www.parasolid.com

PARASOLID Euler operators

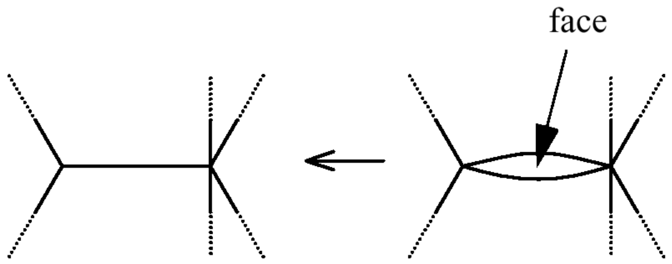
- PK_EDGE_euler_slit



```
PK_ERROR_code_t  PK_EDGE_euler_slit
(
--- received arguments ---
PK_EDGE_t        edge,        --- Edge to be slit
PK_LOGICAL_t     on_left,     --- New face is on left of edge

--- returned arguments ---
PK_FACE_t *const new_face,    --- New face created by slit
PK_EDGE_t *const new_edge     --- New edge created by slit
)
```

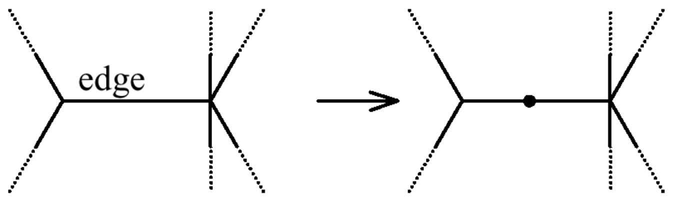
- PK_FACE_euler_unslit



```
PK_ERROR_code_t PK_FACE_euler_unslit
(
--- received arguments ---
PK_FACE_t       face,       --- Face to be unslit
PK_EDGE_t       surviving   --- Edge to survive the unslit
)
```

PARASOLID Euler operators – cont’

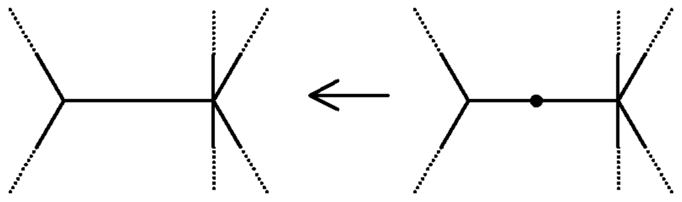
- PK_EDGE_euler_split



```
PK_ERROR_code_t    PK_EDGE_euler_split
(
--- received arguments ---
PK_EDGE_t           edge,         --- Edge to be split
PK_LOGICAL_t        forward,      --- New vertex is forward vertex

--- returned arguments ---
PK_VERTEX_t *const new_vertex,    --- New vertex create by split
PK_EDGE_t   *const new_edge       --- New edge created by split
)
```

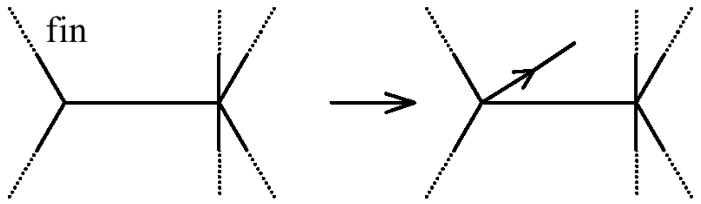
- PK_VERTEX_euler_merge_edges



```
PK_ERROR_code_t PK_VERTEX_euler_merge_edges
(
--- received arguments ---
PK_VERTEX_t     vertex, --- Vertex to be deleted
PK_EDGE_t       edge    --- Edge to survive
)
```

PARASOLID Euler operators – cont’

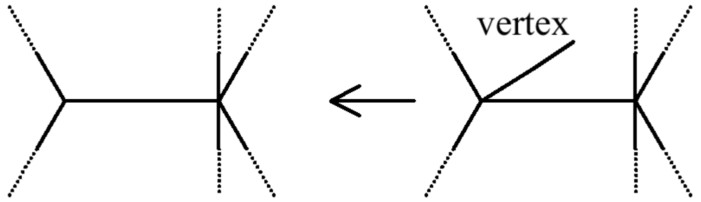
- PK_LOOP_euler_make_edge



```
PK_ERROR_code_t    PK_LOOP_euler_make_edge
(
--- received arguments ---
PK_LOOP_t          loop,         --- Loop in which to create the edge
PK_FIN_t           fin,          --- Fin to create edge at

--- returned arguments ---
PK_VERTEX_t *const new_vertex,   --- New vertex created
PK_EDGE_t   *const new_edge      --- New edge created
)
```

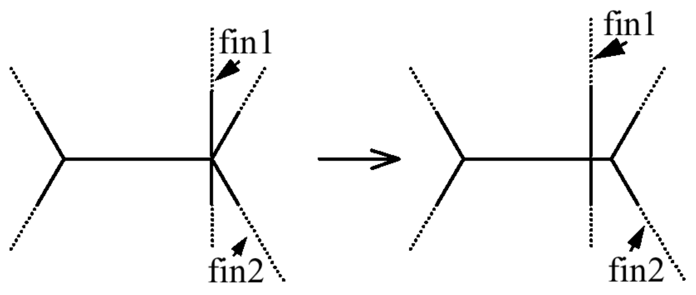
- PK_VERTEX_euler_delete



```
PK_ERROR_code_t PK_VERTEX_euler_delete
(
--- received arguments ---
PK_VERTEX_t     vertex   --- Vertex to be deleted
)
```

PARASOLID Euler operators – cont’

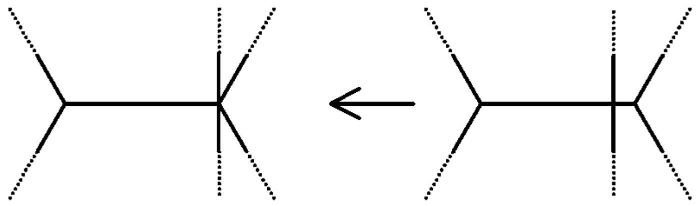
- PK_VERTEX_euler_split



```
PK_ERROR_code_t    PK_VERTEX_euler_split
(
--- received arguments ---
PK_VERTEX_t        vertex,       --- Vertex to be split
PK_FIN_t           fin1,         --- First fin at vertex
PK_FIN_t           fin2,         --- Second fin at vertex

--- returned arguments ---
PK_VERTEX_t *const new_vertex,   --- New vertex created
PK_EDGE_t   *const new_edge      --- New edge created
)
```

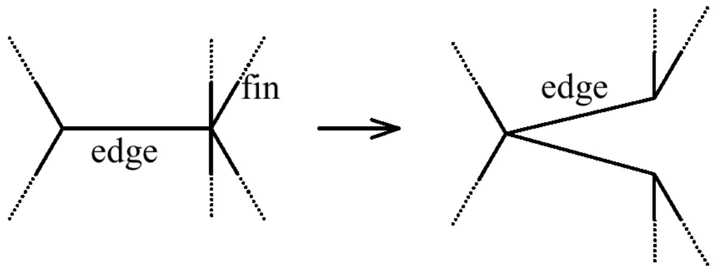
- PK_EDGE_euler_merge_vertices



```
PK_ERROR_code_t PK_EDGE_euler_merge_vertices
(
--- received arguments ---
PK_EDGE_t          edge,    --- Edge to delete
PK_VERTEX_t        vertex   --- Vertex to delete
)
```

PARASOLID Euler operators – cont’

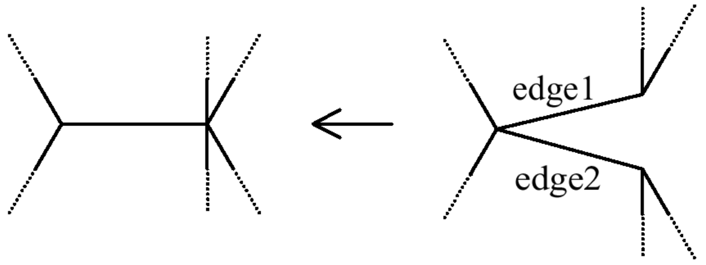
- PK_EDGE_euler_open_zip



```
PK_ERROR_code_t    PK_EDGE_euler_open_zip
(
--- received arguments ---
PK_EDGE_t            edge,         --- Edge to split
PK_FIN_t             fin,          --- Fin at vertex to split

--- returned arguments ---
PK_VERTEX_t *const new_vertex,     --- New vertex created
PK_EDGE_t   *const new_edge        --- New edge created
)
```

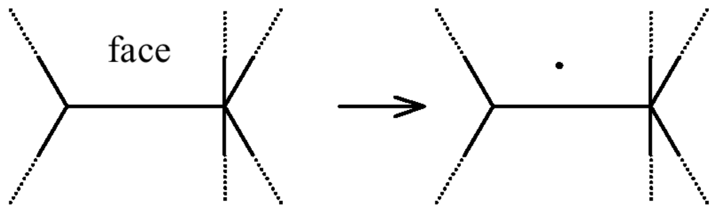
- PK_EDGE_euler_close_zip



```
PK_ERROR_code_t PK_EDGE_euler_close_zip
(
--- received arguments ---
PK_EDGE_t        edge1,   --- Edge to zip
PK_EDGE_t        edge2    --- Edge to delete
)
```

PARASOLID Euler operators – cont’

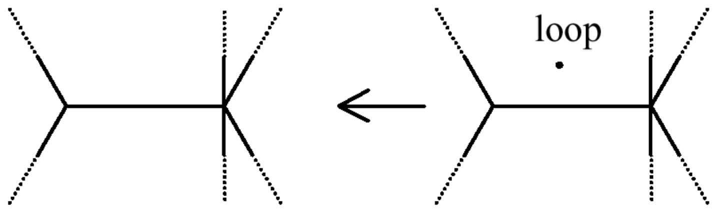
- PK_FACE_euler_make_loop



```
PK_ERROR_code_t  PK_FACE_euler_make_loop
(
--- received arguments ---
PK_FACE_t          face,       --- Face to contain new loop

--- returned arguments ---
PK_LOOP_t *const new_loop      --- New loop created
)
```

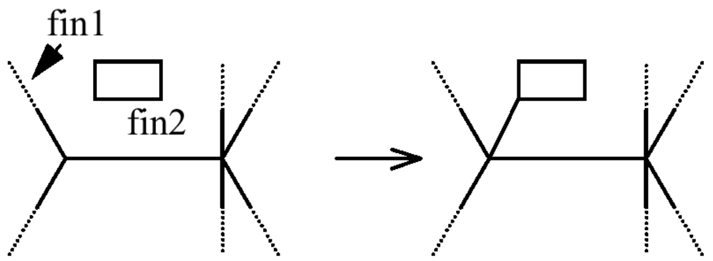
- PK_LOOP_euler_delete_isolated



```
PK_ERROR_code_t PK_LOOP_euler_delete_isolated
(
--- received arguments ---
PK_LOOP_t         loop     --- Loop to delete
)
```

PARASOLID Euler operators – cont'

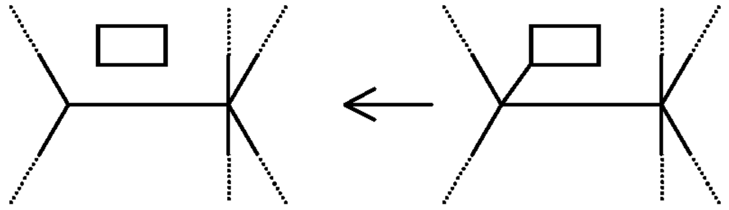
- PK_LOOP_euler_delete_make_edge



```
PK_ERROR_code_t  PK_LOOP_euler_delete_make_edge
(
--- received arguments ---
PK_LOOP_t          loop1, --- Loop of fin1
PK_FIN_t           fin1,  --- Fin at first vertex to join
PK_LOOP_t          loop2, --- Loop of fin2
PK_FIN_t           fin2,  --- Fin at second vertex to join

--- returned arguments ---
PK_EDGE_t *const new_edge
)
```

- PK_EDGE_euler_delete_make_loop

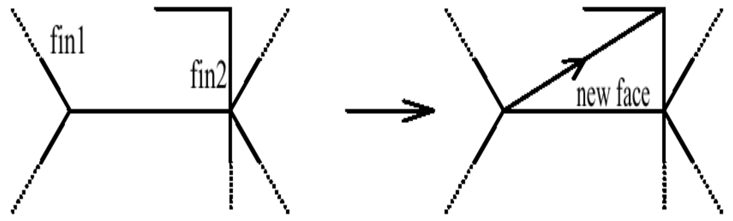


```
PK_ERROR_code_t  PK_EDGE_euler_delete_make_loop
(
--- received arguments ---
PK_EDGE_t        edge,      --- Edge to be deleted
PK_LOGICAL_t     forward,   --- Forward vertex connects to new loop

--- returned arguments ---
PK_LOOP_t *const new_loop   --- New loop created
)
```

PARASOLID Euler operators – cont’

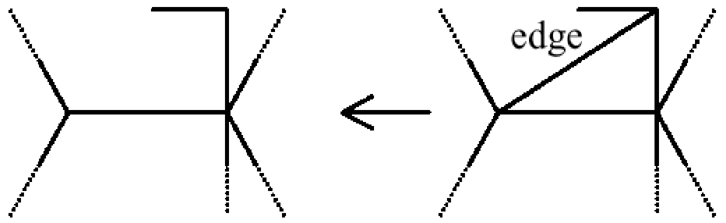
- PK_LOOP_euler_make_edge_face



```
PK_ERROR_code_t  PK_LOOP_euler_make_edge_face
(
--- received arguments ---
PK_LOOP_t          loop,       --- Loop of face
PK_FIN_t           fin1,       --- Fin at first vertex to join
PK_FIN_t           fin2,       --- Fin at second vertex to join

--- returned arguments ---
PK_FACE_t *const new_face,   --- New face created
PK_EDGE_t *const new_edge    --- New edge created
)
```

- PK_EDGE_euler_delete_with_face



```
PK_ERROR_code_t PK_EDGE_euler_delete_with_face
(
--- received arguments ---
PK_EDGE_t       edge,    --- Edge to be deleted
PK_LOGICAL_t    on_left --- Face to delete is on left of edge
)
```

PARASOLID Euler operators – cont'

PK_LOOP_euler_make_edge_loop

This function is similar to PK_LOOP_euler_make_edge_face, but instead of splitting the face in two, it creates a new loop in the face, increasing its genus.

```
PK_ERROR_code_t  PK_LOOP_euler_make_edge_loop
(
--- received arguments ---
PK_LOOP_t         loop,      --- Loop of face
PK_FIN_t          fin1,      --- Fin at first vertex to join
PK_FIN_t          fin2,      --- Fin at second vertex to join

--- returned arguments ---
PK_LOOP_t *const new_loop    --- New loop created
)
```

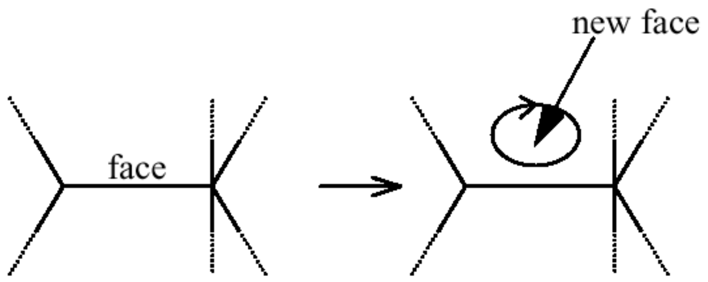
PK_EDGE_euler_delete_with_loop

This function is similar to PK_EDGE_euler_delete_with_face, except that it deletes an edge which has different loops in the same face on each side of it. It merges the loops into one, decreasing the genus of the face.

```
PK_ERROR_code_t PK_EDGE_euler_delete_with_loop
(
--- received arguments ---
PK_EDGE_t       edge,    --- Edge to be deleted
PK_LOGICAL_t    on_left  --- Loop to delete is on left of edge
)
```

PARASOLID Euler operators – cont’

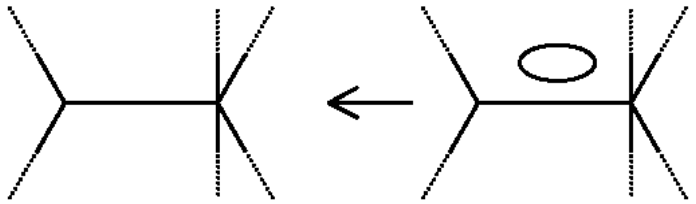
- PK_FACE_euler_make_ring_face



```
PK_ERROR_code_t  PK_FACE_euler_make_ring_face
(
--- received arguments ---
PK_FACE_t          face,       --- Face to contain new face

--- returned arguments ---
PK_FACE_t *const new_face    --- New face created
)
```

- PK_EDGE_euler_delete_ring_face



```
PK_ERROR_code_t PK_EDGE_euler_delete_ring_face
(
--- received arguments ---
PK_EDGE_t       edge,    --- Edge to be deleted
PK_LOGICAL_t    on_left --- Face on the left is to be deleted
)
```

PARASOLID Euler operators – cont'

- PK_FACE_euler_make_ring_loop

This function is similar to PK_FACE_euler_make_ring_face, except that instead of splitting the face in two, it creates a new loop in the face, thus increasing its genus.

```
PK_ERROR_code_t  PK_FACE_euler_make_ring_loop
(
--- received arguments ---
PK_FACE_t         face,       --- Face to contain new loop

--- returned arguments ---
PK_LOOP_t *const new_loop     --- One of the two loops created
)
```

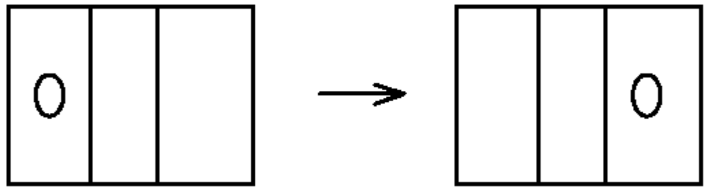
- PK_EDGE_euler_delete_ring_loop

This function is similar to PK_EDGE_euler_delete_with_face, except that it deletes a ring edge which has different loops from the same face on each side, thus decreasing the genus of the face.

```
PK_ERROR_code_t PK_EDGE_euler_delete_ring_loop
(
--- received arguments ---
PK_EDGE_t         edge     --- Edge to be deleted
)
```

PARASOLID Euler operators – cont’

- PK_LOOP_euler_transfer



```
PK_ERROR_code_t PK_LOOP_euler_transfer
(
--- received arguments ---
PK_LOOP_t        loop,     --- loop to transfer
PK_FACE_t        face      --- destination face
)
```

This function transfers a loop from one face to another. The two faces must have the same front and back shells.