Geometric Modeling System

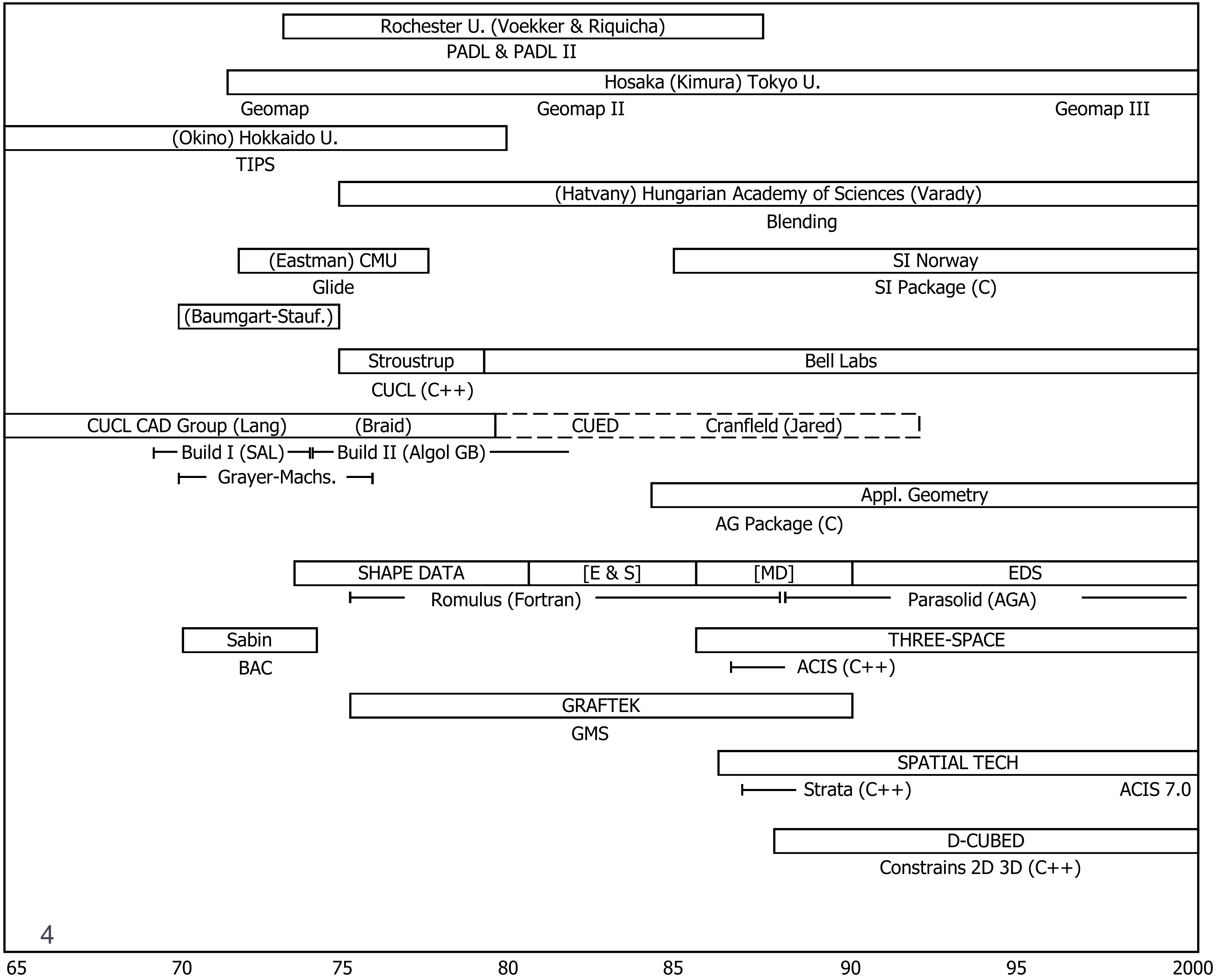
Human-centered CAD Lab.

Geometric modeling system

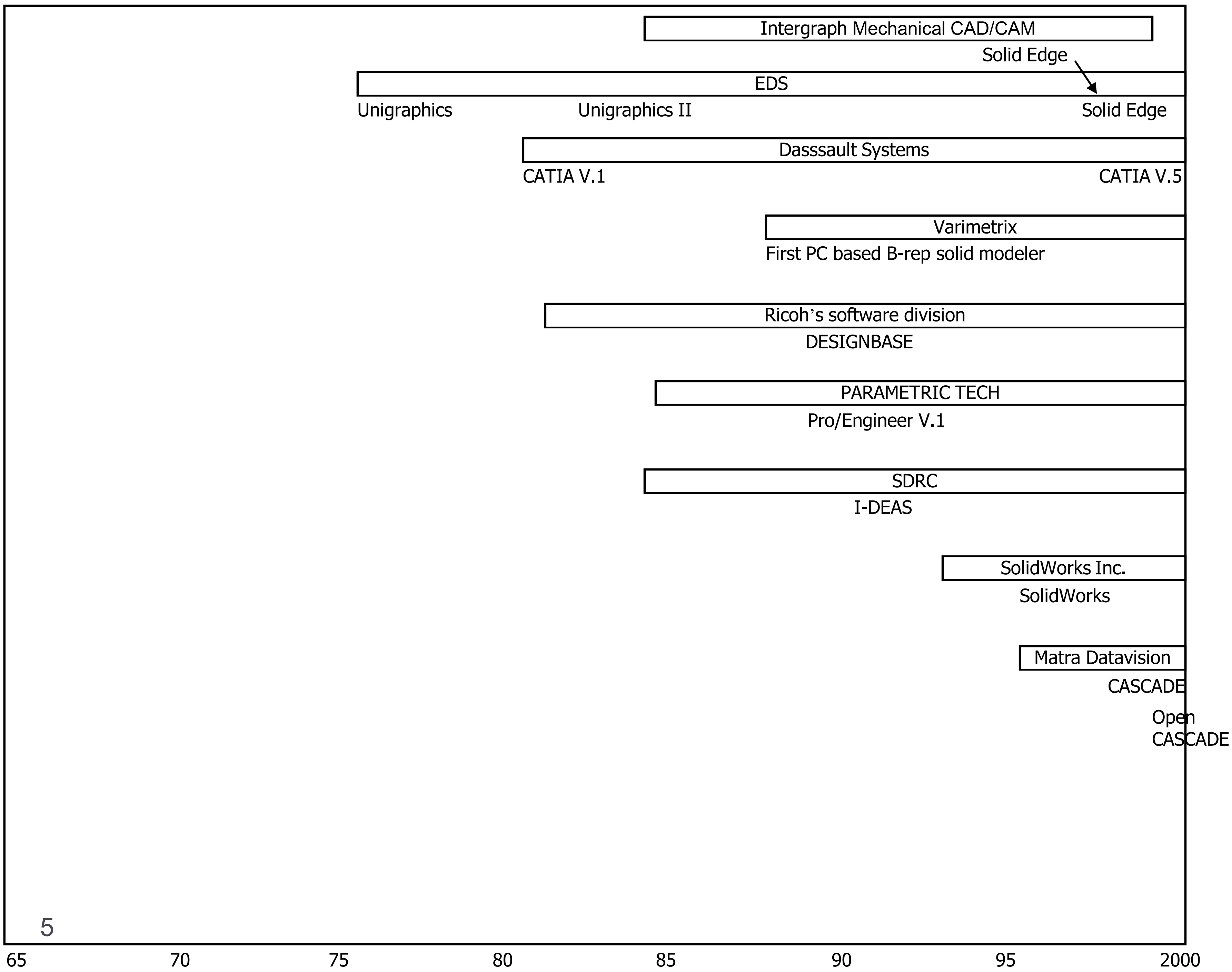
- Software enabling shape creation and visulaization in the design process
- Designer realizes the shape in his mind while the shape data are stored inside
  - Wireframe Modeling System
  - Surface Modeling System
  - Solid Modeling System
  - Non-manifold Modeling System

History of Geometric Modeling

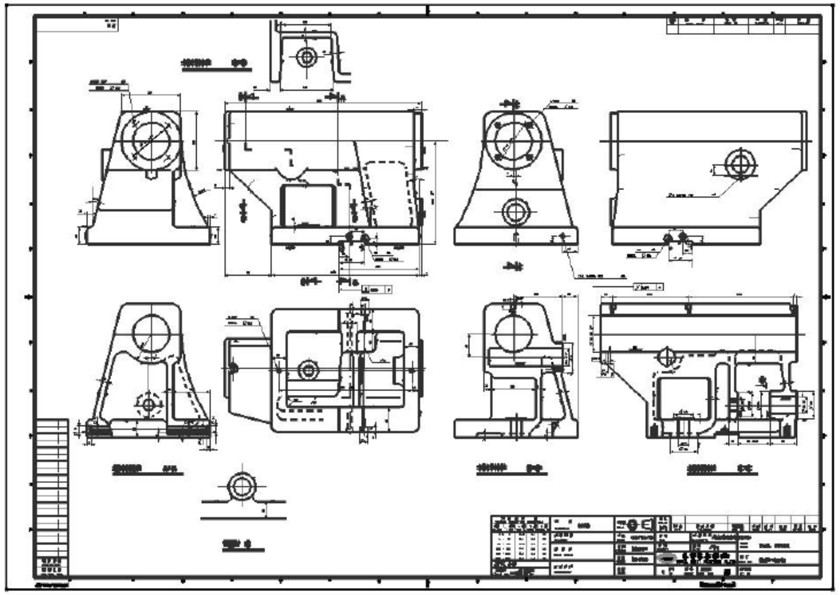
- Tips
  - Okino, Kubo at Hokaido University, 1973
  - Constructive Solid Geometry (CSG)
- Build
  - Braid, Lang at Cambridge University, 1973
  - Boundary Representation (B-rep)
- CADAM, Unigraphics, CATIA, I-DEAS, BRAVO, ME10/30, Pro/ENGINEER, DesignBASE, SolidEdge, SolidWorks, …

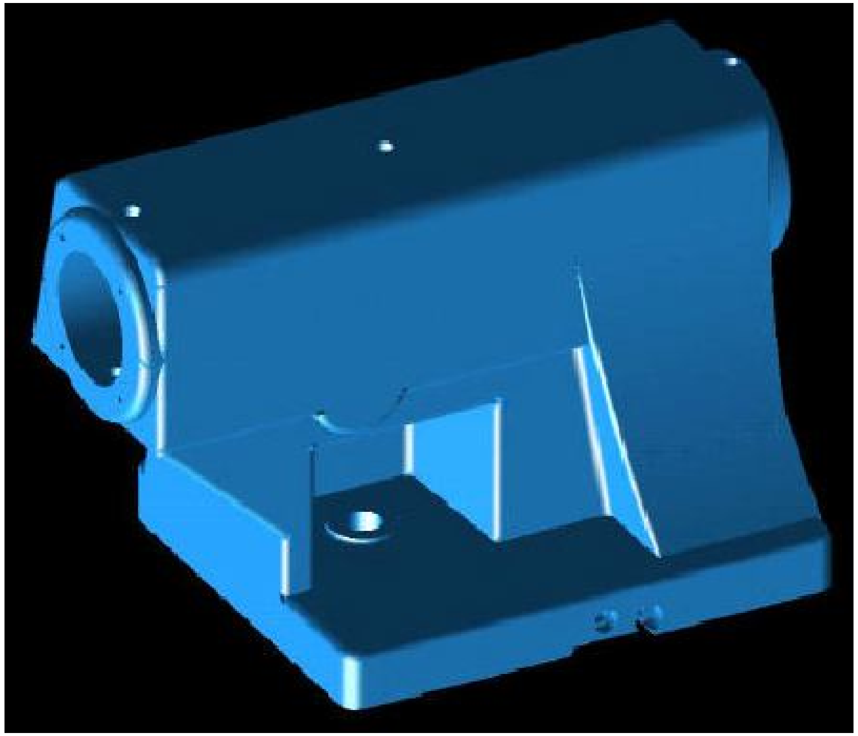


Rochester U. (Voekker & Riquicha)
PADL & PADL II
Hosaka (Kimura) Tokyo U.
Geomap
Geomap II
Geomap III
(Okino) Hokkaido U.
TIPS
(Hatvany) Hungarian Academy of Sciences (Varady)
Blending
(Eastman) CMU
Glide
SI Norway
SI Package (C)
(Baumgart-Stauf.)
Stroustrup
Bell Labs
CUCL (C++)
CUCL CAD Group (Lang)
(Braid)
CUED
Cranfleld (Jared)
Build I (SAL)
Build II (Algol GB)
Grayer-Machs.
Appl. Geometry
AG Package (C)
SHAPE DATA
[E & S]
[MD]
EDS
Romulus (Fortran)
Parasolid (AGA)
Sabin
BAC
THREE-SPACE
ACIS (C++)
GRAFTEK
GMS
SPATIAL TECH
Strata (C++)
ACIS 7.0
D-CUBED
Constrains 2D 3D (C++)
65
70
75
80
85
90
95
2000


Intergraph Mechanical CAD/CAM
Solid Edge
EDS
Unigraphics
Unigraphics II
Solid Edge
Dasssault Systems
CATIA V.1
CATIA V.5
Varimetrix
First PC based B-rep solid modeler
Ricoh's software division
DESIGNBASE
PARAMETRIC TECH
Pro/Engineer V.1
SDRC
I-DEAS
SolidWorks Inc.
SolidWorks
Matra Datavision
CASCADE
Open CASCADE
65
70
75
80
85
90
95
2000

Why 3 Dimensional Model?

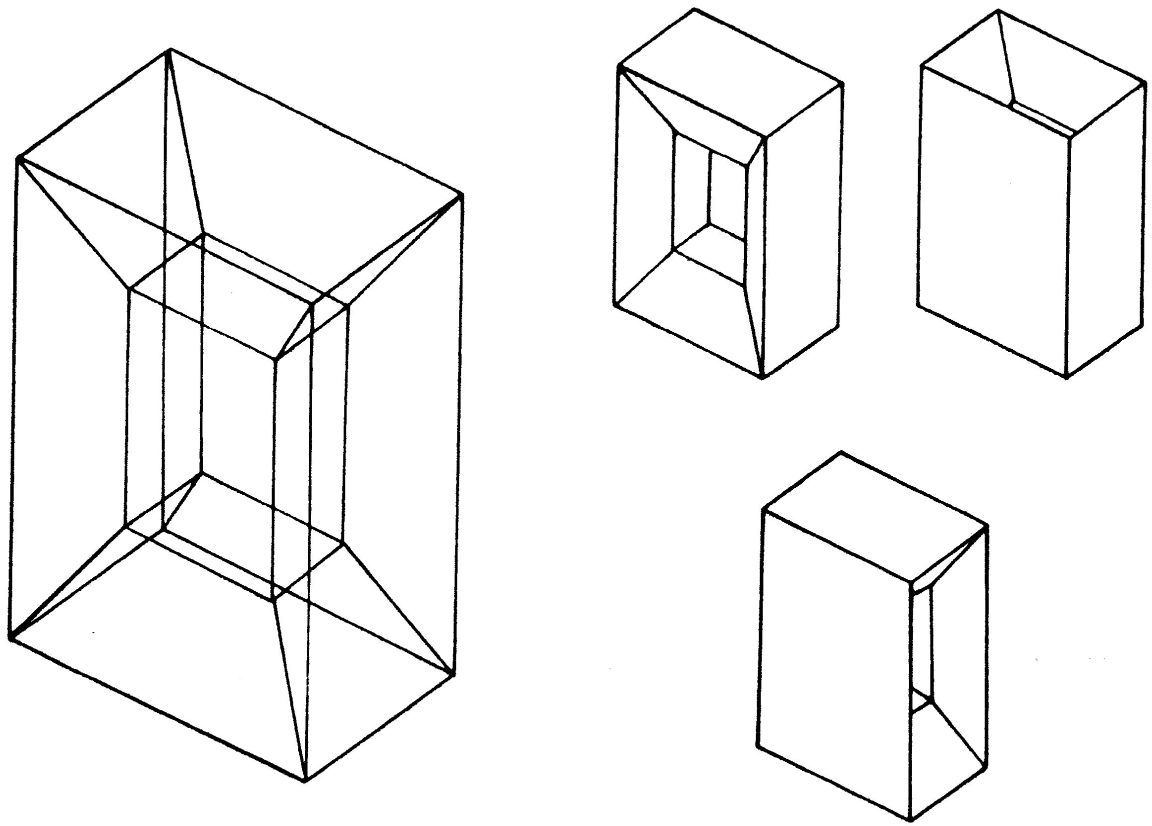




Wireframe Modeling System

- User inputs characteristic points and curves
- Good for simple visualization
- Ambiguous situations may occur
- Impossible to automatically calculate mass properties, NC tool paths, and finite elements

Ambiguous wireframe models



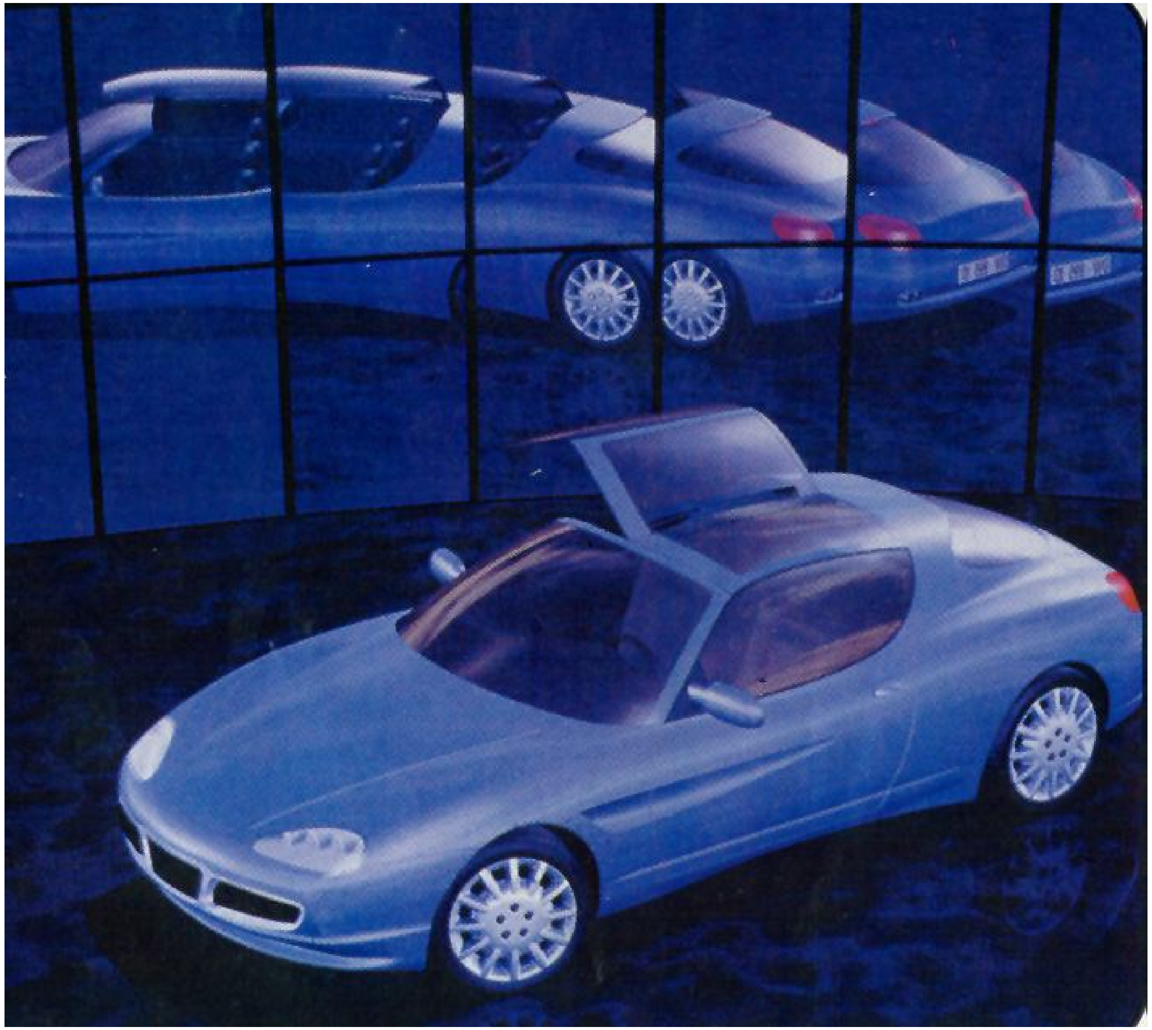
Surface Modeling System

- Surface information in addition to wireframe model
- Usually user specify the curves on a surface, then system stores the surface equation
- Adjacency information between surfaces are not stored in general
- Intersection calculation is needed to derive the boundary curves
- Some surface modeling systems store boundary curves also

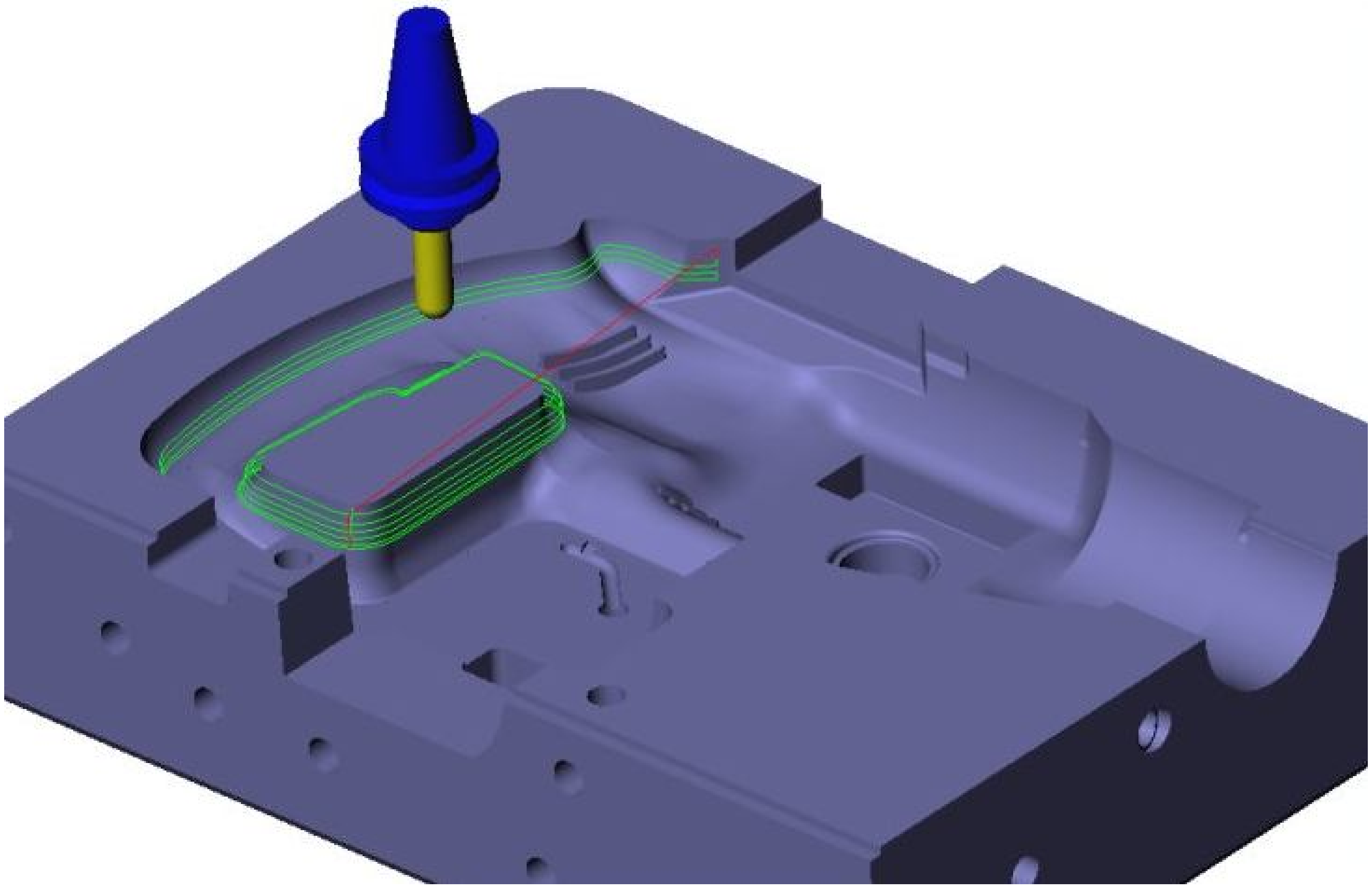
Surface Modeling System – cont’

- Point set
- Curve net
- Curve movement (Sweeping, Skinning)
- Good for aesthetic evaluation, Styling CAD
- Input for NC tool path generation
- Good for modeling object bounded by complicated surfaces

Modeling of automobile body by surface modeling system



Calculation and verification of NC tool paths



Solid Modeling System

- Adjacency information between faces, and inside-outside information of each face are stored in addition
- Volume inside modeled object is defined
- Volumetric operations are possible
  - Automatic generation of solid elements for FEA
  - Automatic generation of tool paths for rough cut

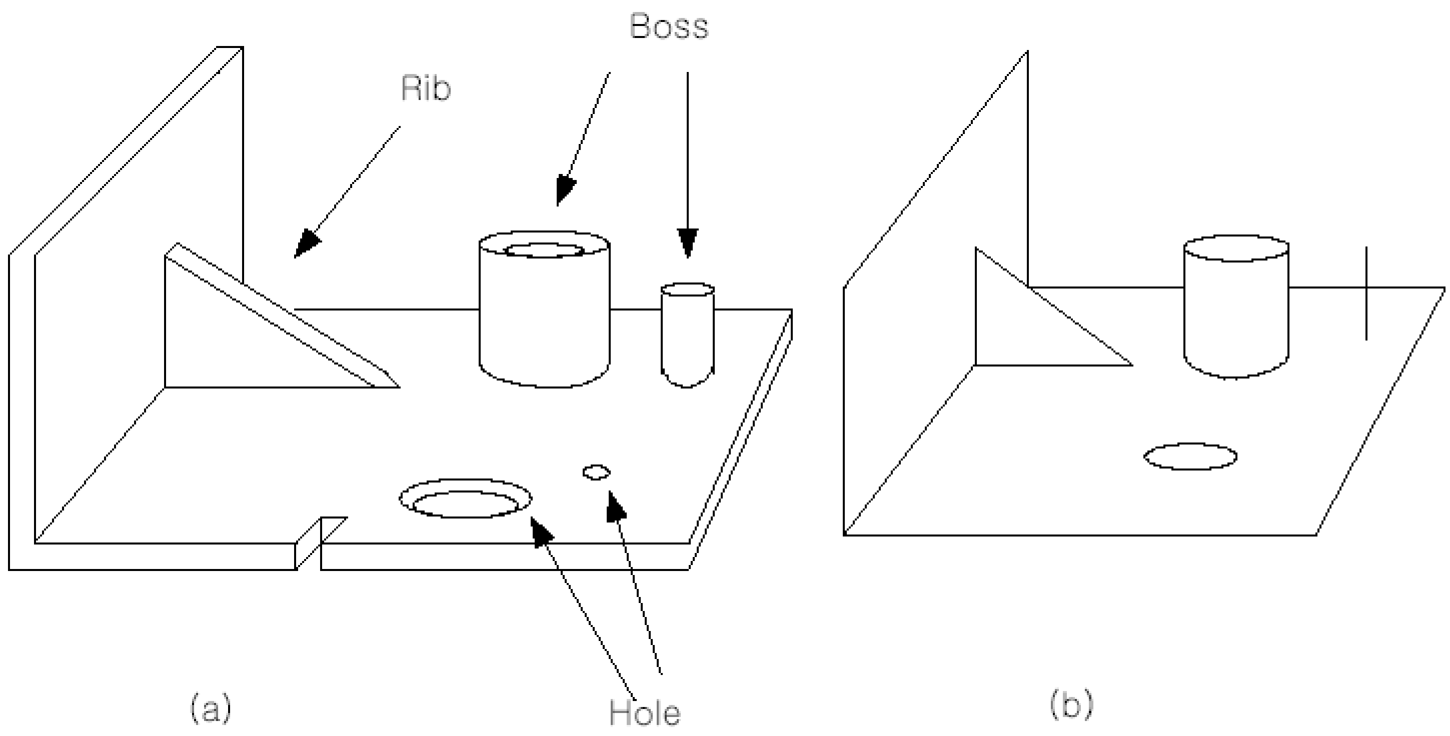
Solid Modeling System – cont'

- Partial modeling is not allowed, complete solid model should be made
- More modeling tasks
- Many convenient modeling commands are provided
  - Face adjacency, in-out information, etc. are generated by the system

Non-manifold Modeling System

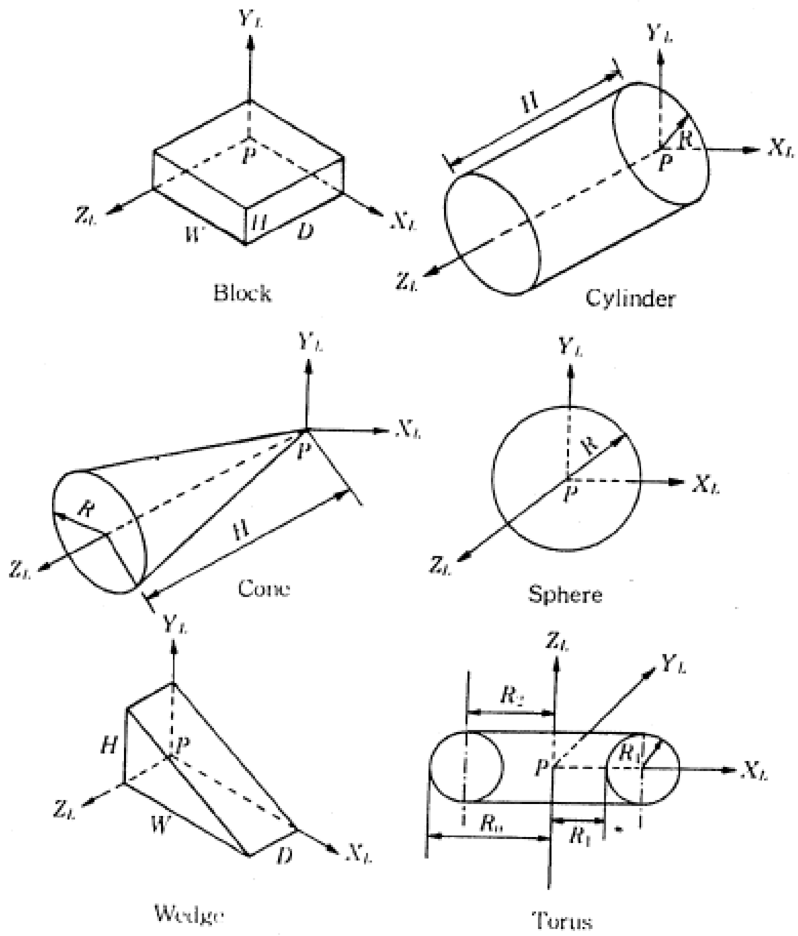
- Accommodate all different levels of geometric model
  - Wireframe model : Wireframe modeling system
  - Surface model : Surface modeling system
  - Solid model : Solid modeling system
- Models of mixed dimension, incomplete models are allowed (support design process, analysis model)

Non-manifold Modeling System



Modeling Functions
(1) Primitives Creation

- Retrieves a solid of a simple shape
- Primitives are stored by the procedures how they are created.
- Parameters specifying the size are passed to the corresponding procedure as arguments.

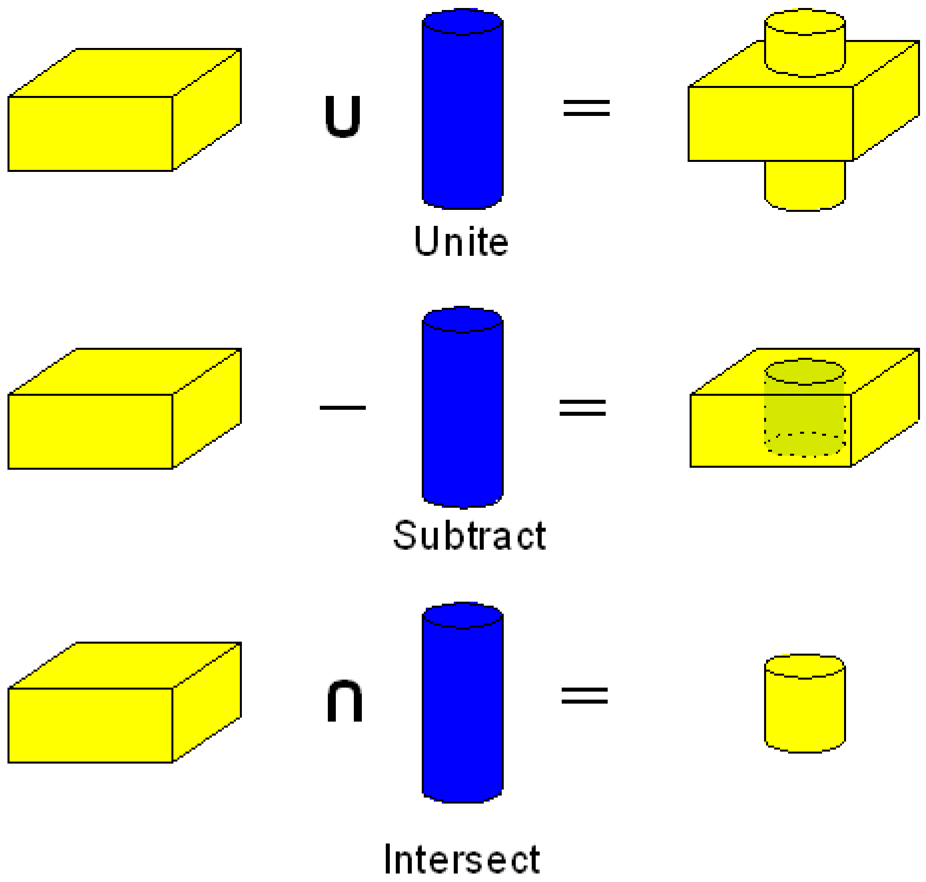


Primitives generally supported

Modeling Functions
(2) Boolean operation

- Primitive solid is assumed to be a set of points
- Boolean operation is performed between the point sets
- The result is the solid composed of the points resulting from the set operation.

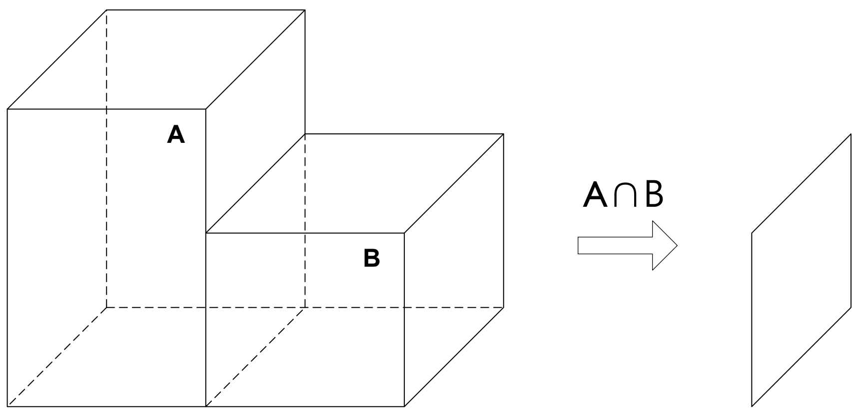
Modeling Functions
(2) Boolean operation

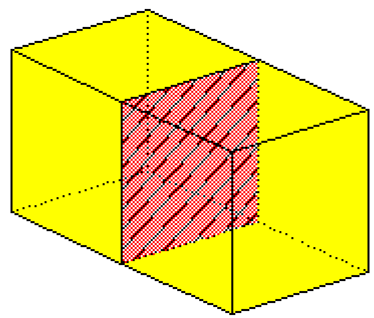


Modeling Functions
(2) Boolean operation

- Boolean operation may result an invalid solid
- Non-manifold modeling systems can handle Boolean operations between objects of mixed dimension.

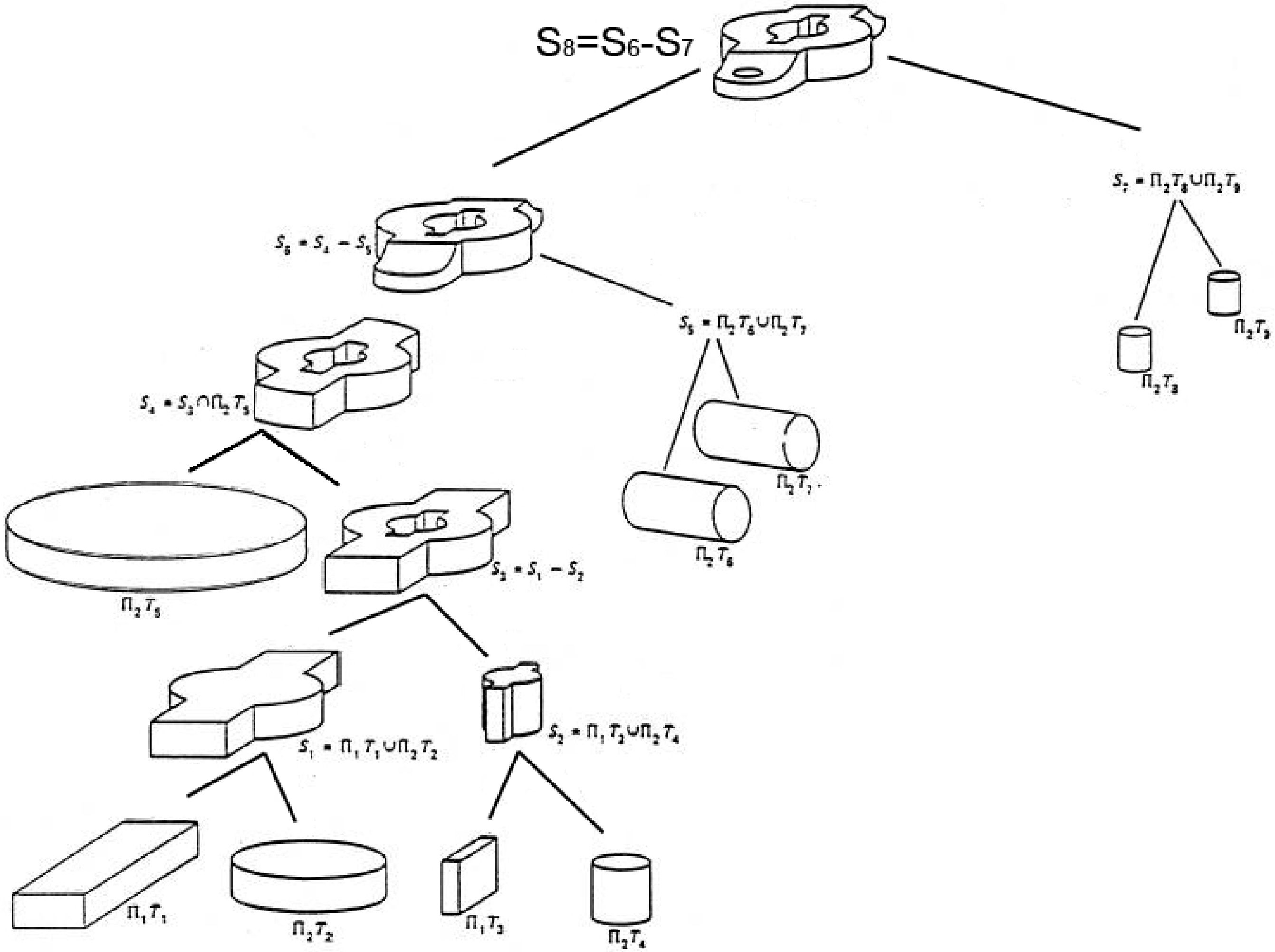
Example of Boolean operation to be avoided





A ∩ B

Example of modeling in CSG approach

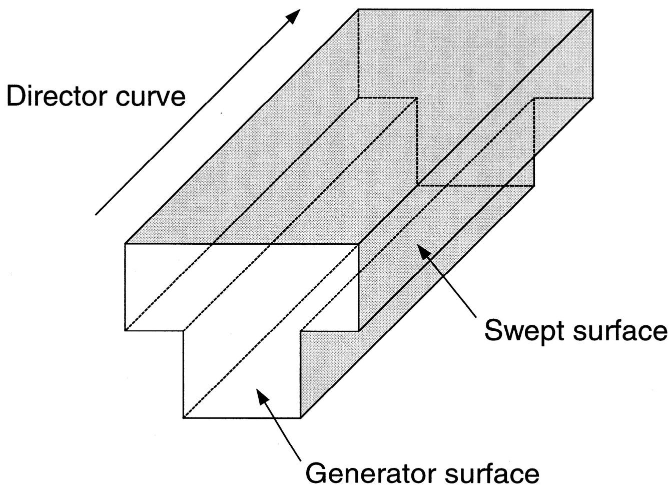


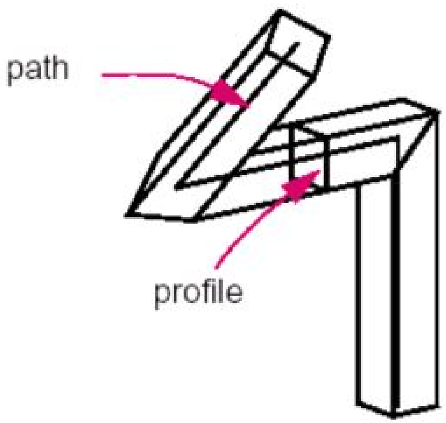
Modeling Functions
(3) Sweeping

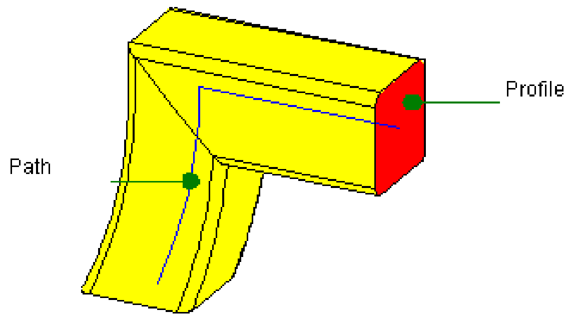
- Planar closed domain is translated or revolved to form a solid
- When the planar shape is not closed, the result is a surface
  - Used in surface modeling system

Modeling Functions
(3) Sweeping – Example.1

- Example of translation sweeping

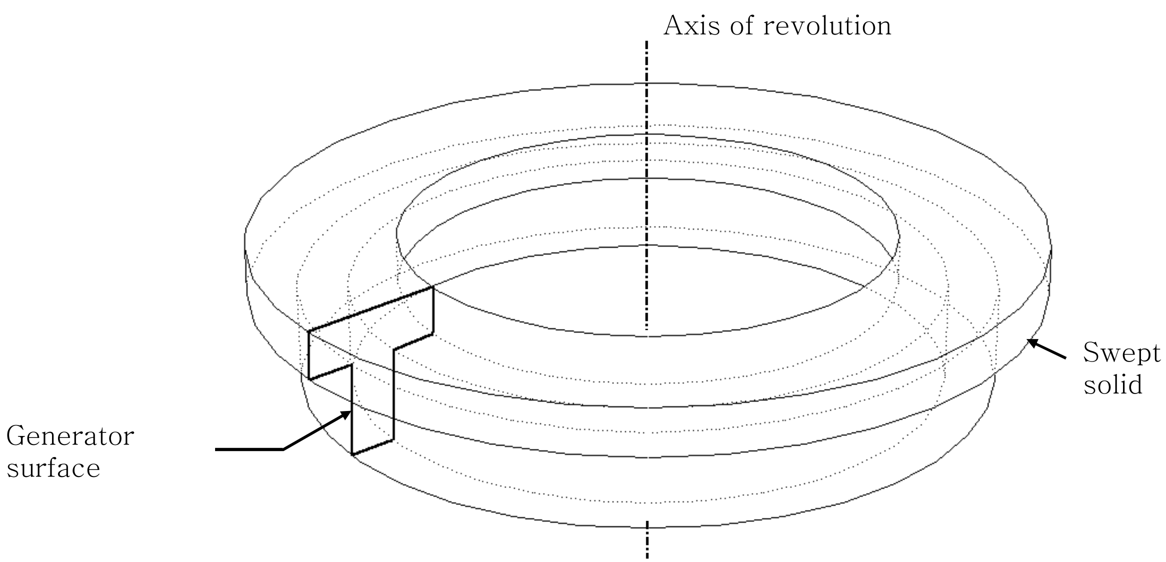


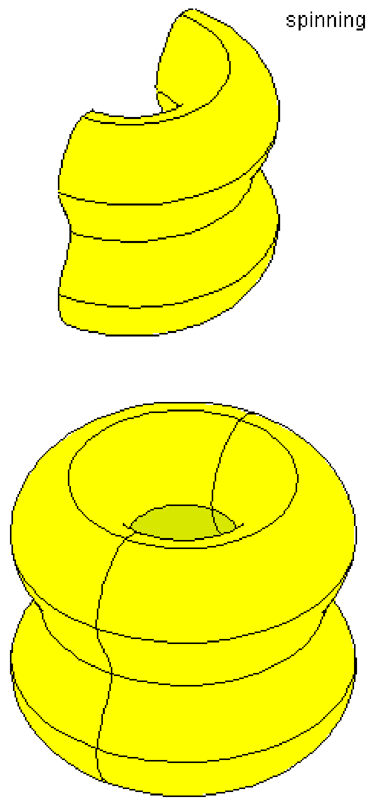




Modeling Functions
(3) Sweeping – Example.2

- Example of rotational sweeping





Modeling Functions
(4) Skinning

- Form a closed volume by creating a skin surface over pre-specified cross sectional planar curves
- If two end faces corresponding to the two end cross sections are not added, the result would be a surface
  - Used in surface modeling system

Modeling Functions
(4) Skinning (Lofting) - Example

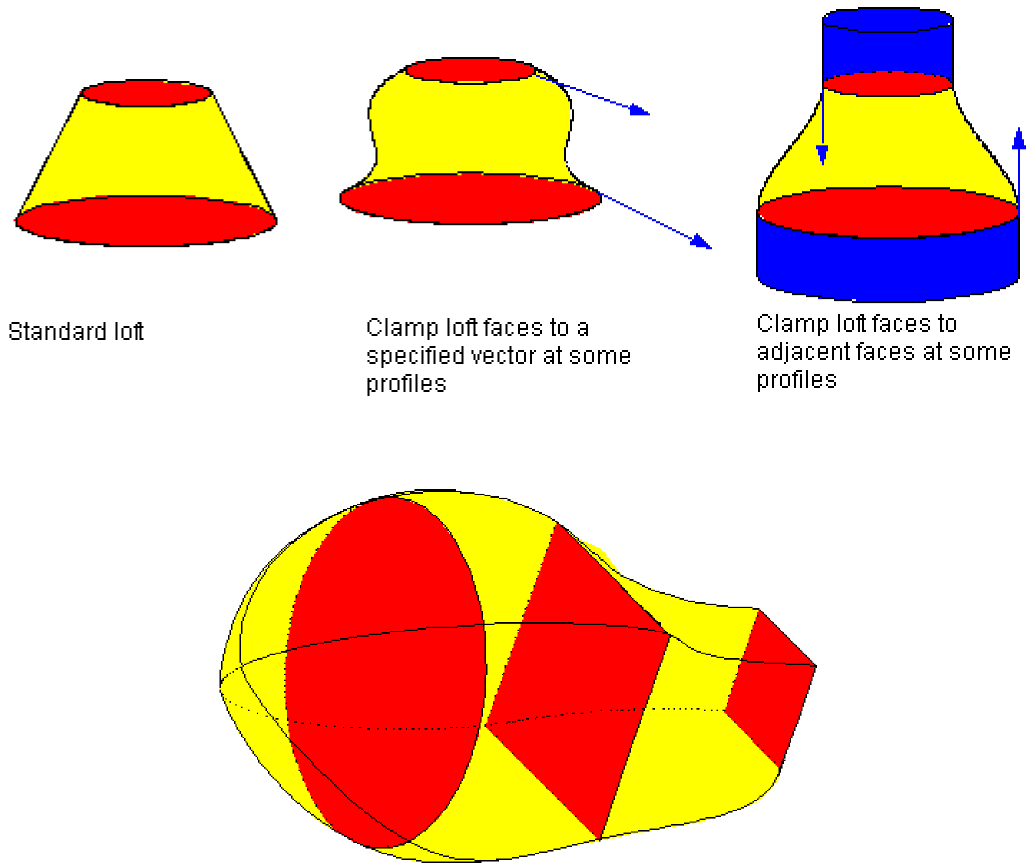
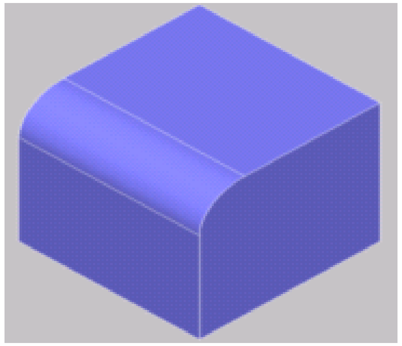


Figure 8-6 Creating a lofted body using several different profiles

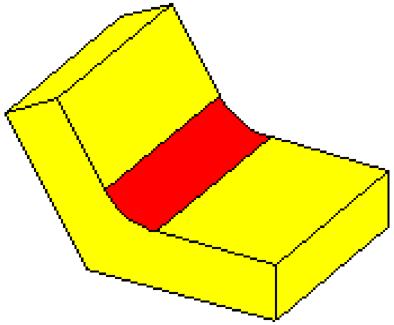
Modeling Functions
(5) Blending

- Sharp edge or sharp vertex is replaced by a smooth curved surface
- Normal vector is continuous across the surfaces meeting at the original sharp edge or vertex

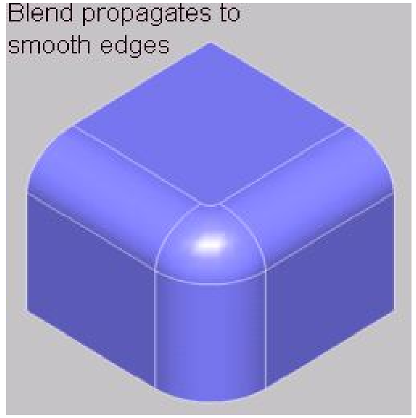
Modeling Functions
(5) Blending – Example



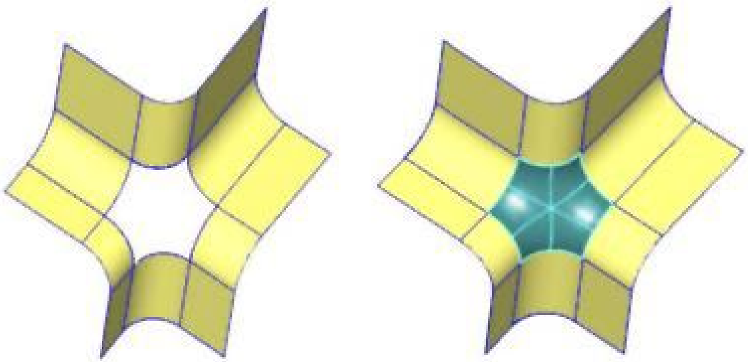
Edge rounding



Edge filleting



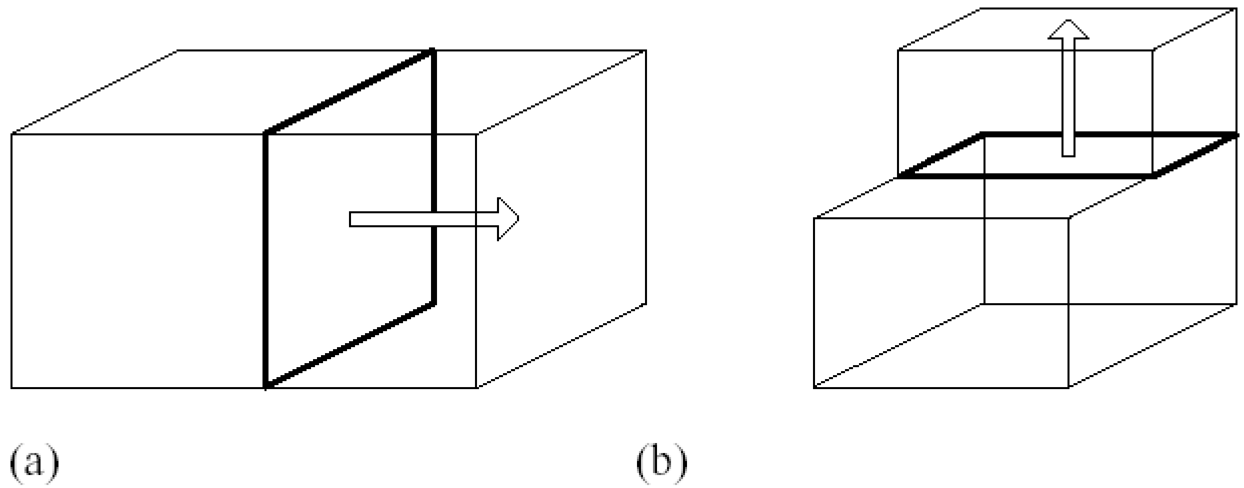
Vertex rounding



complex intersecting blends

Modeling Functions
(6) Lifting

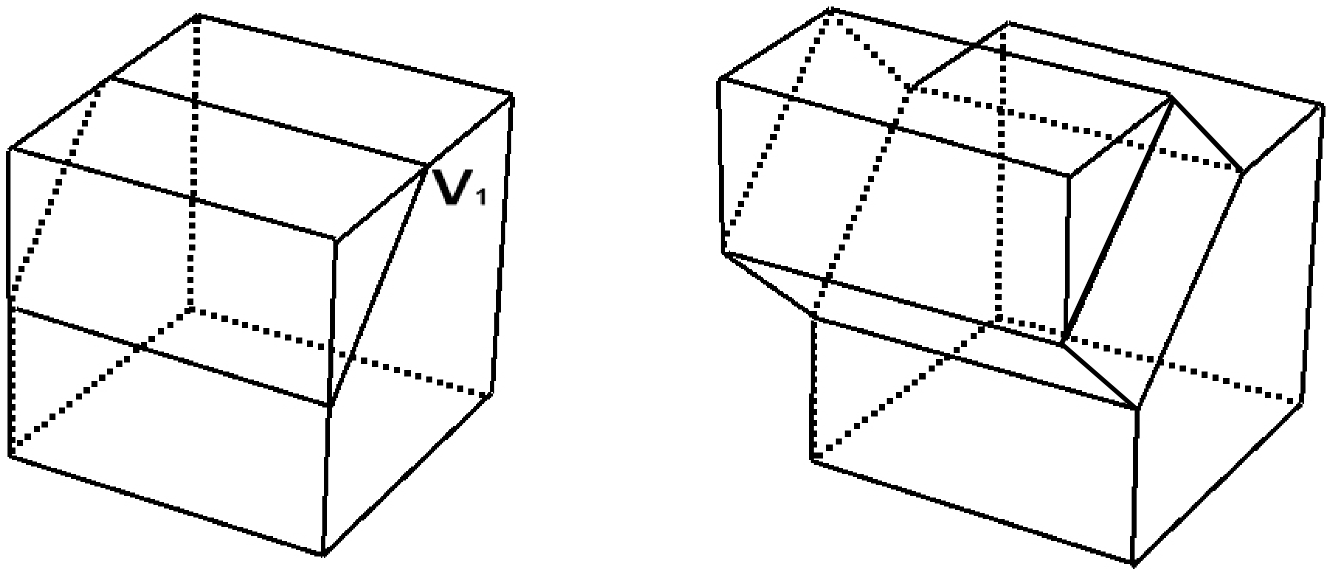
- Pull a portion or whole face of a solid



Example of lifting

Modeling Functions (6) Lifting

- Face lifting

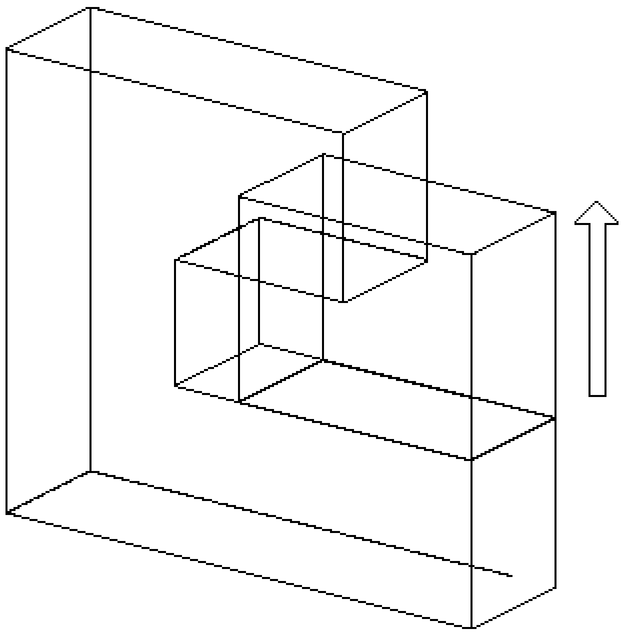


Modeling Functions
(6) Lifting

- When a portion of a face is lifted, the face should be split beforehand
  - Add a splitting edge
  - Update face connectivity
  - Update edge adjacency, …
- Euler operators will handle these tasks

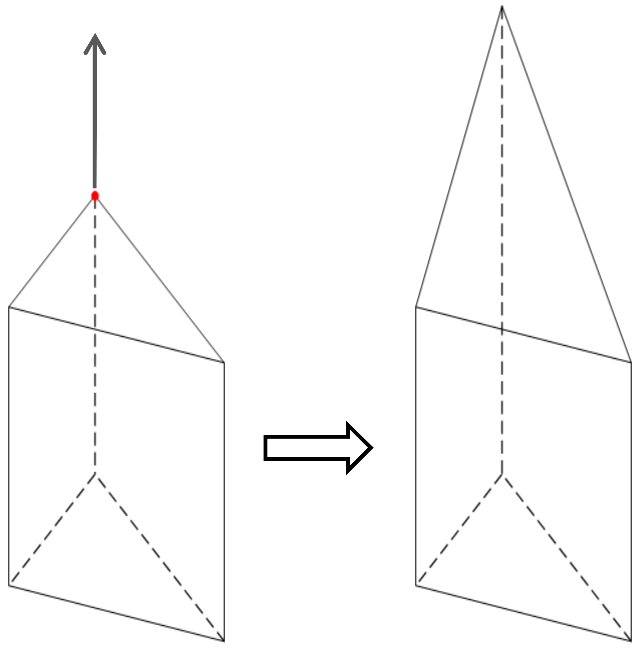
Modeling Functions
(6) Lifting

- Self interference caused by lifting

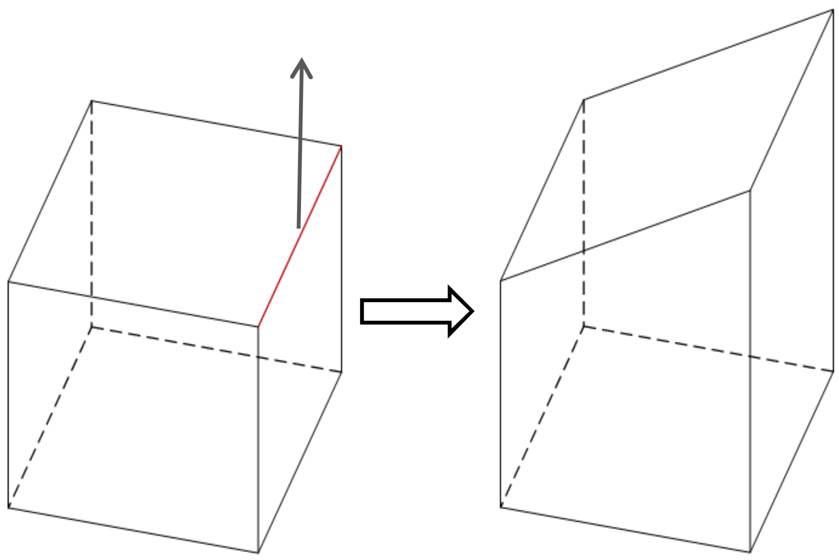


Modeling Functions
(7) Tweaking

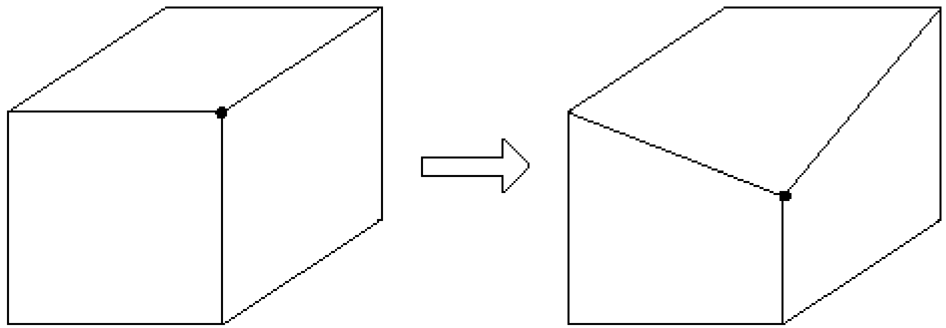
- Vertex Tweaking



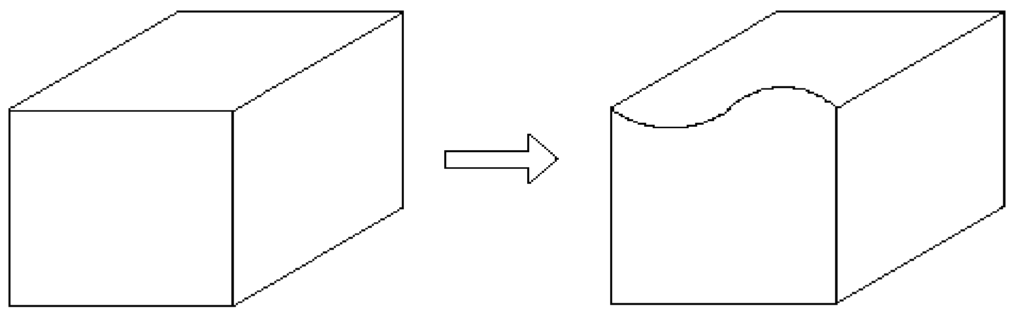
- Edge Tweaking



Modeling Functions
(7) Tweaking



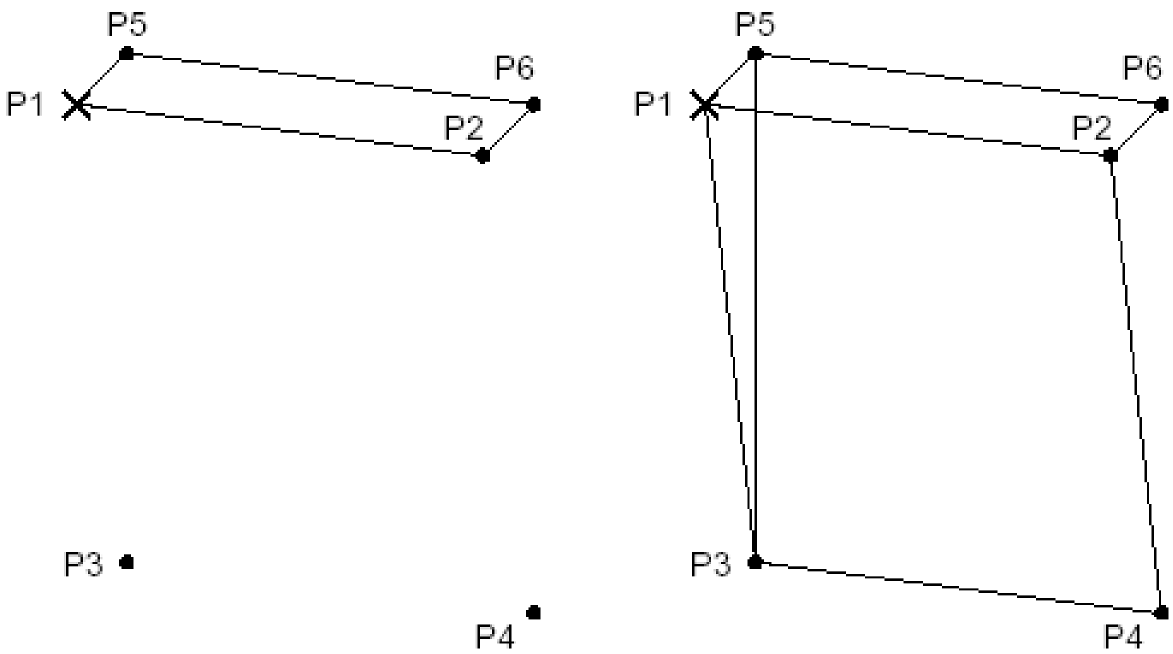
Modification by vertex moving



Modification by edge replacement

Modeling Functions
(8) Boundary Modeling

- Add, delete, modify entities such as vertices, edges, and faces directly



Modeling Functions
(8) Boundary Modeling

- Very tedious operation
- Boundary modeling functions are mainly used to create only up to two dimensional shapes which are used for sweeping or skinning
- Can be effectively applied to modify a shape of an existing solid
  - Tweaking operation

Modeling Functions
(9) Feature based modeling

- Let designers model a solid by the shape units familiar to them
- The resulting solid carries the information on the existence of these shape units in addition to the elementary shape entities such as vertices, edges, faces, etc.

Modeling Functions (9) Feature based modeling

- E.g.
  - ‘ Make a hole of a certain size at a certain place ’
  - ‘ Make a chamfer of a certain size at a certain place ’
    - Existence of hole and chamfer is added to model information
- Set of features varies depending upon the frequent applications of the system

Modeling Functions
(9) Feature based modeling

- Popular feature

chamfer, hole, fillet, slot, pocket, …

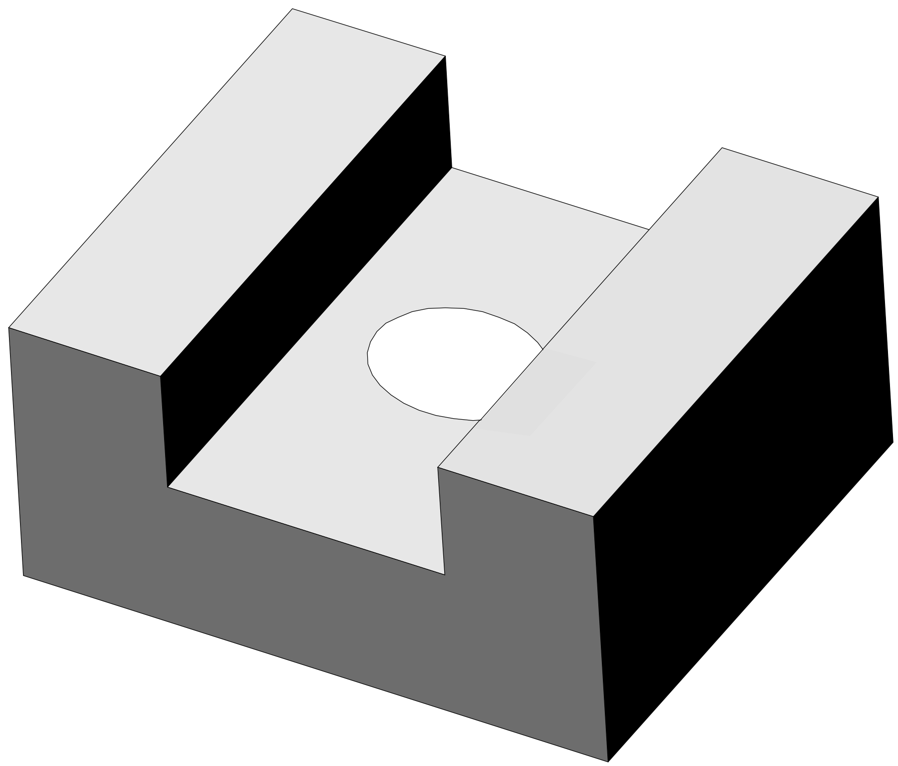
↓

manufacturing features

↓

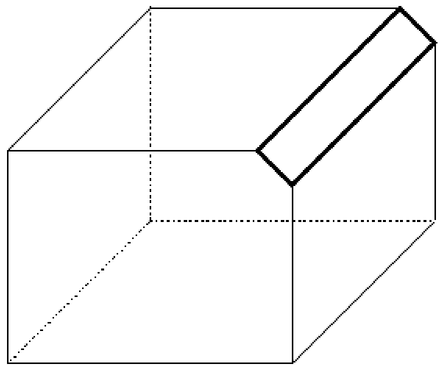
These features can be matched to
a specific machining process

Modeling Functions
(9) Feature based modeling

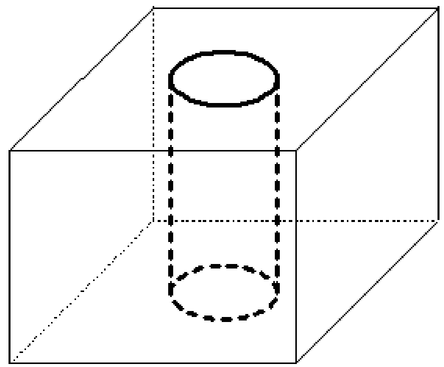


Example of modeling using “slot” and “hole” features

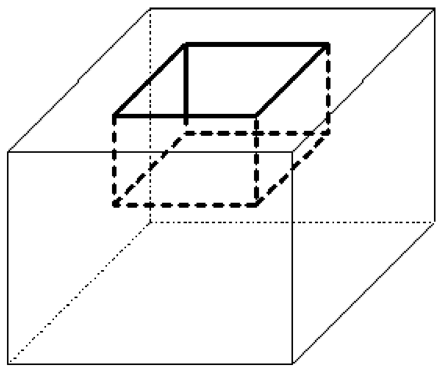
Example modeling using machining features



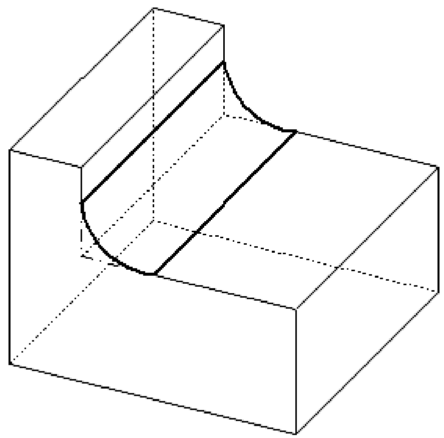
(a) Chamfering



(b) Hole



(c) Pocket



(d) Fillet

Modeling Functions
(9) Feature based modeling

- Any feature based modeling system cannot provide all the features necessary for all the specific applications
- The desirable set of features is different between applications
- Many systems provide feature definition language so that any specific feature can be defined
- When they are defined, they are parameterized as the primitives

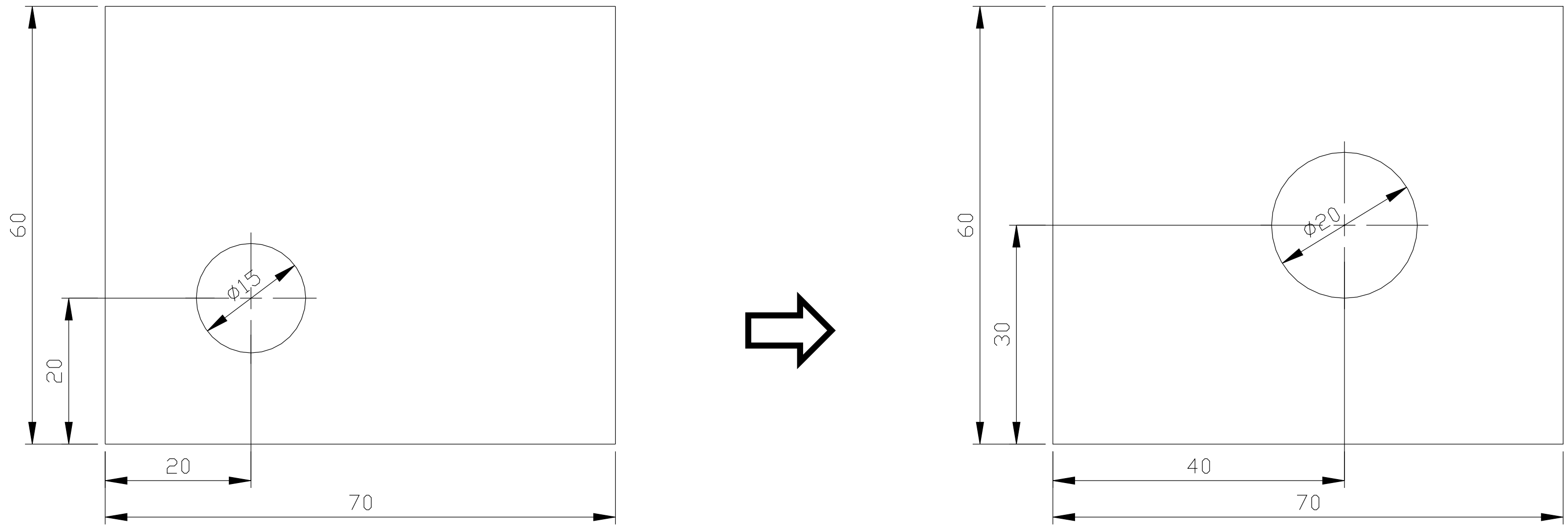
Modeling Functions
(10) Parametric Modeling

- Model a shape by using the geometric constraints and the dimension data
- Geometric constraints describe the relation between shape elements
- Dimensional data include dimensions and relations between the dimensions

Modeling Functions
(10) Parametric Modeling

- Input two dimensional shape roughly
- Input geometric constraints and dimension data
- Reconstruct the two dimensional shape
- Create 3D shape by sweeping or swinging

Modeling Functions
(10) Parametric Modeling



Data structure of solid model

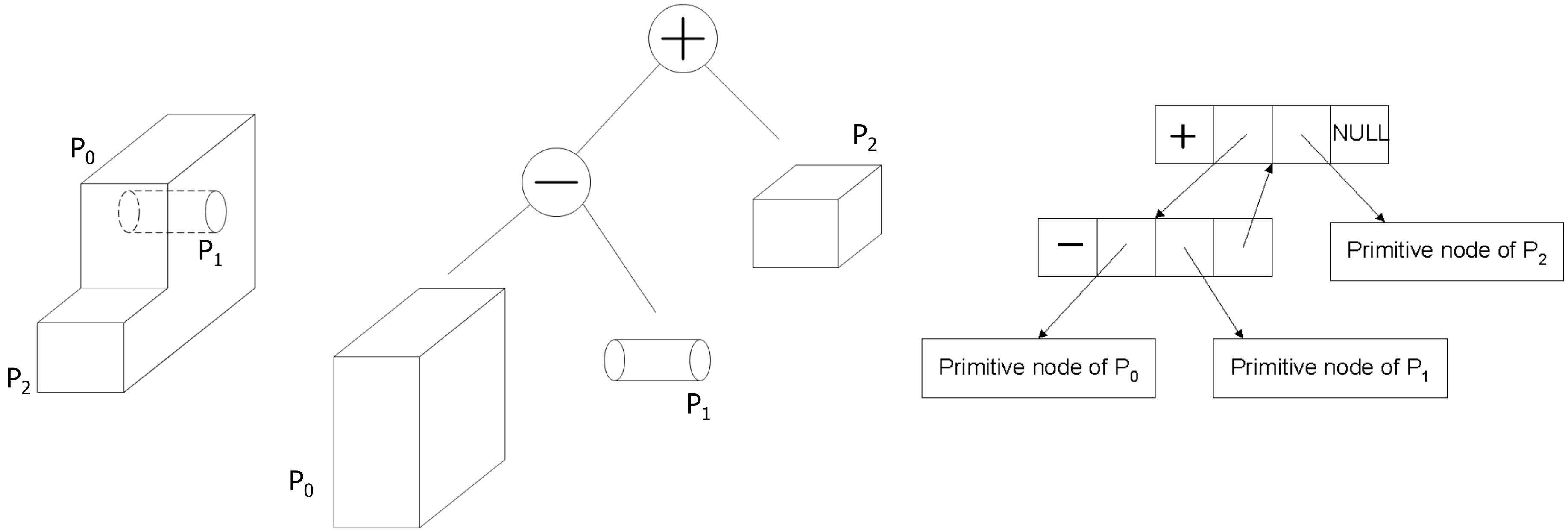
- CSG Representation storing CSG tree
  - Store procedure Boolean operation in tree structure
- Boundary Representation (B-Rep)
  - Data structure vertex, edge, face tables
  - Data structure using half edge
  - Data structure using Winged-edge

Data structure of solid model – cont'

- Data structure storing decomposition model
  - Octree representation
  - Voxel representation
  - Cell decomposition model
  - Similar to finite element

CSG tree

- Stores the procedure in which Boolean operations are applied



Example of CSG tree

Implementation of CSG tree structure in C language

```
struct operator {
    int     op_type,                 /* union, intersection or difference operator */
            L_type;                  /* left node type: 0=operator, 1=primitive */
            R_type                   /* right node type: 0=operator, 1=primitive */
    void    *L_ptr;                  /* left node */
            *R_ptr;                  /* right node */
            *p_ptr;                  /* parent node */
}

struct primitive {
    int     prim_type;               /* type of primitive */
    double  pos_x, pos_y, pos_z;     /* position of instance */
    double  ori_x, ori_y, ori_z;     /* orientation of instance */
    void    *attribute;              /* the value of dimensions of the primitive */
}
```

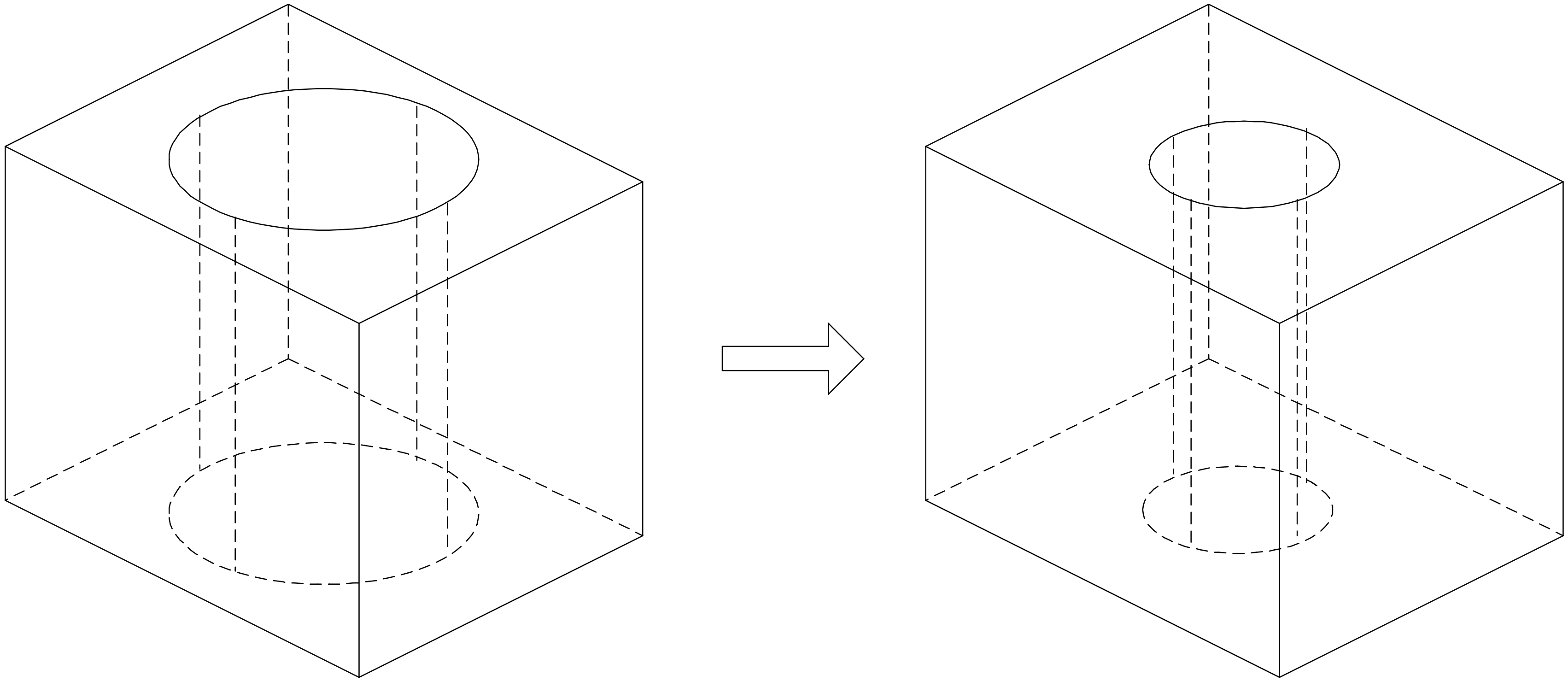
CSG tree representation– advantages

- Compact data, Easy to maintain
- Represent only valid object
- Possible to be converted to B-Rep
  - Many applications can be integrated
  - Model can be easily changed by changing parameter values of primitives

CSG tree representation – disadvantages

- Allows only Boolean operations
- Shapes to be modeled are limited
- Impossible to modify locally
- Significant computation is required for boundary evaluation
  → bad for interactive display
- Trends are to store B-Rep and Feature tree together

Modification of solid by changing parameters

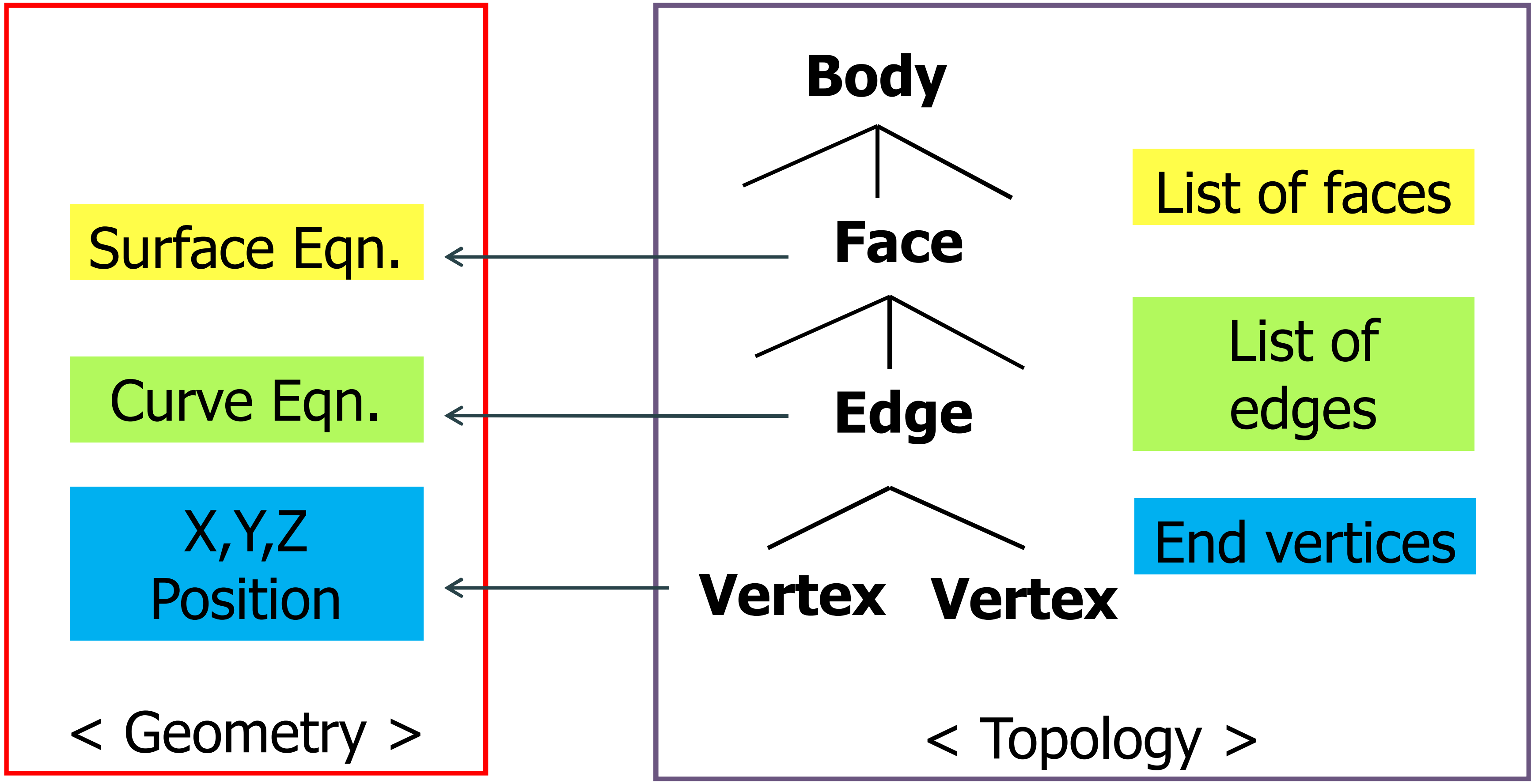


B-Rep(Boundary Representation)

- Shape is expressed by its bounding entities such as faces, edges, and vertices
- Bounding entities and their connectivity are stored in graph structure
  → Graph-based model

B-Rep Structure – cont'

Topology vs. Geometry



B-Rep – advantages

- Boundary data are stored explicitly and enables quick interactive response
- Topology information can be easily derived
- Supports various modeling commands (local operations in addition to Boolean)

B-Rep – disadvantages

- Complicated data structure with a large amount of data
- Invalid solid may result

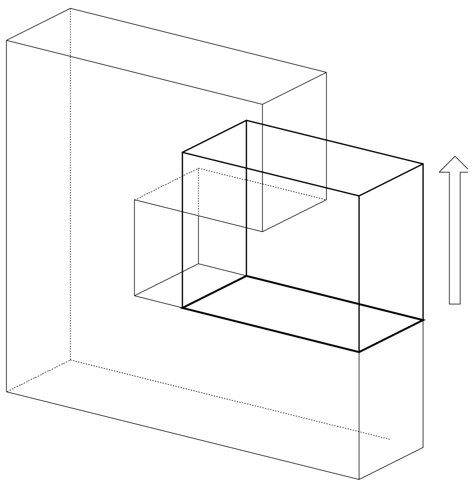
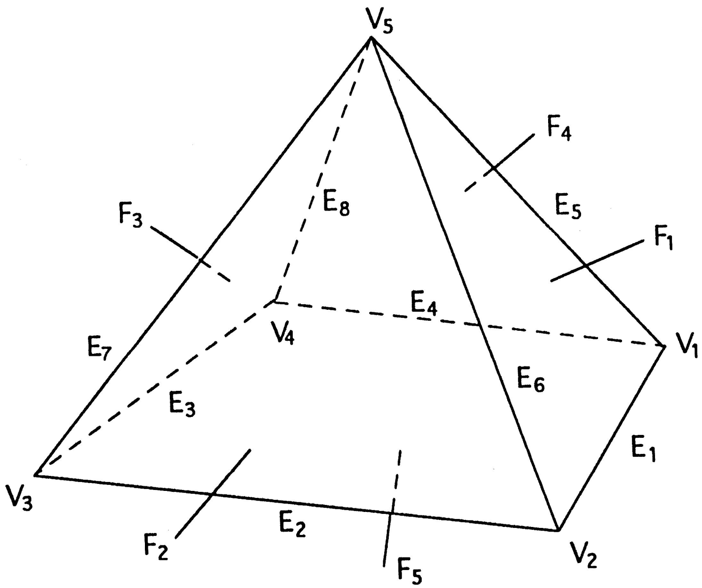


Table-based structure for storing B-Rep

Face table	
Face	Edges
F_1	E_1, E_5, E_6
F_2	E_2, E_6, E_7
F_3	E_3, E_7, E_8
F_4	E_4, E_8, E_5
F_5	E_1, E_2, E_3, E_4

Edge table	
Edge	Vertices
E_1	V_1, V_2
E_2	V_2, V_3
E_3	V_3, V_4
E_4	V_4, V_1
E_5	V_1, V_5
E_6	V_2, V_5
E_7	V_3, V_5
E_8	V_4, V_5

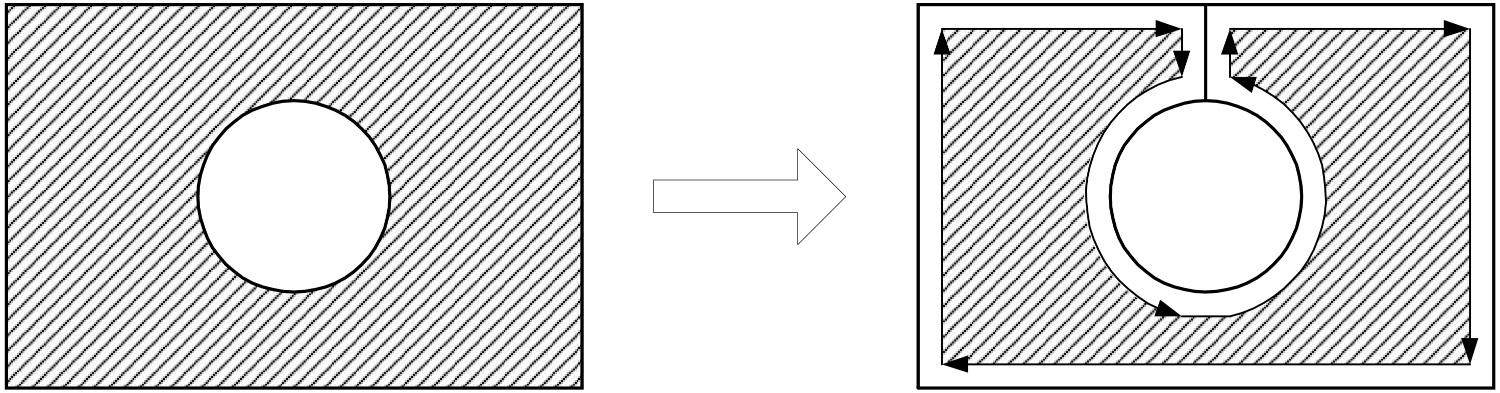
Vertex table	
Vertex	Coordinates
V_1	x_1, y_1, z_1
V_2	x_2, y_2, z_2
V_3	x_3, y_3, z_3
V_4	x_4, y_4, z_4
V_5	x_5, y_5, z_5
V_6	x_6, y_6, z_6



Things to be cosidered

- Balance between structure compactness and effectiveness in data retrieval
- Basically used for polyhedron models
- For objects with curved surfaces and curved edges, information on surface equations are stored in the Face table, information on curve equations are stored in the Edge table
- If there are faces with holes, the current Face table cannot be used

Treatment of face with multiple boundaries



Adding bridge-edge is one way to
handle hole

B-Rep – Things to be considered

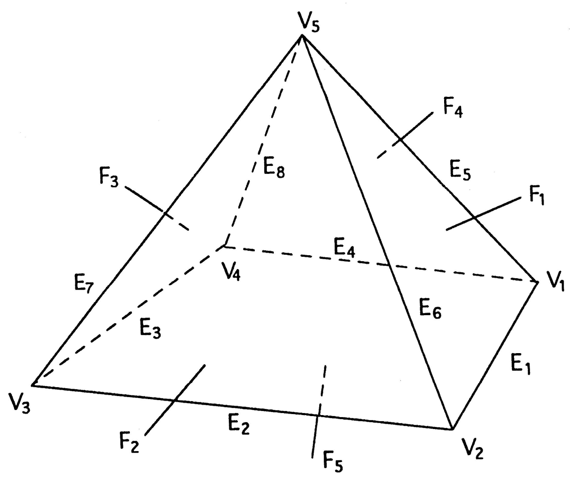
- Length of edge table in the Face table varies
  → Loss of memory usage
- Deriving adjacency among Vertex, Edge, Face requires a heavy search

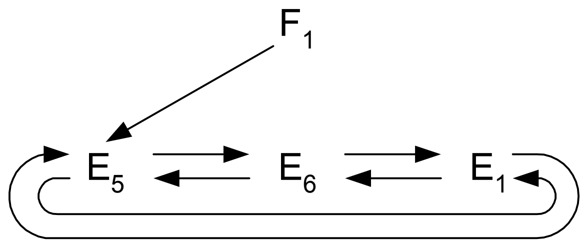
Ex) Which faces share a given edge?

Which edges share a given vertex?

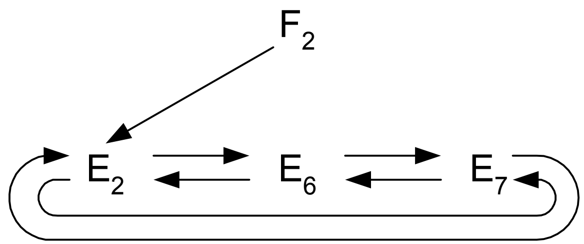
Half Edge Data Structure

- Varying length of edge list in the Face table can be solved by linked list





Doubly linked list for face F_1



Doubly linked list for face F_2

Half Edge Data Structure – cont'

- Every face points to any one edge, every edge points to its next edge

→ The number of edges bounding a face has no effect

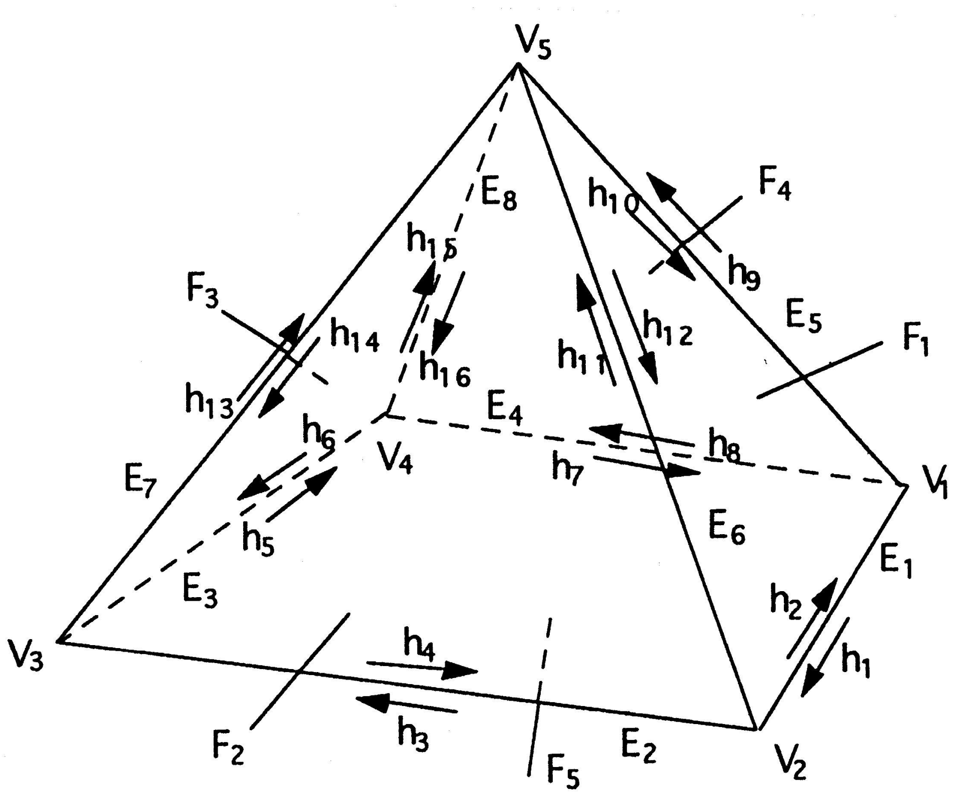
- The next edge of edge E_6 changes depending on the face being considered

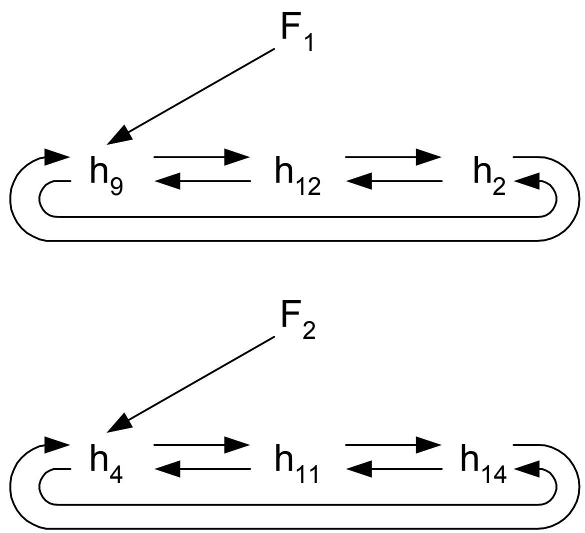
→ Data for F_2 are deleted when data for F_3 is stored

- Each edge is split into two halves, and each split edge is used for each Face

→ half edge

Half edges of the example solid





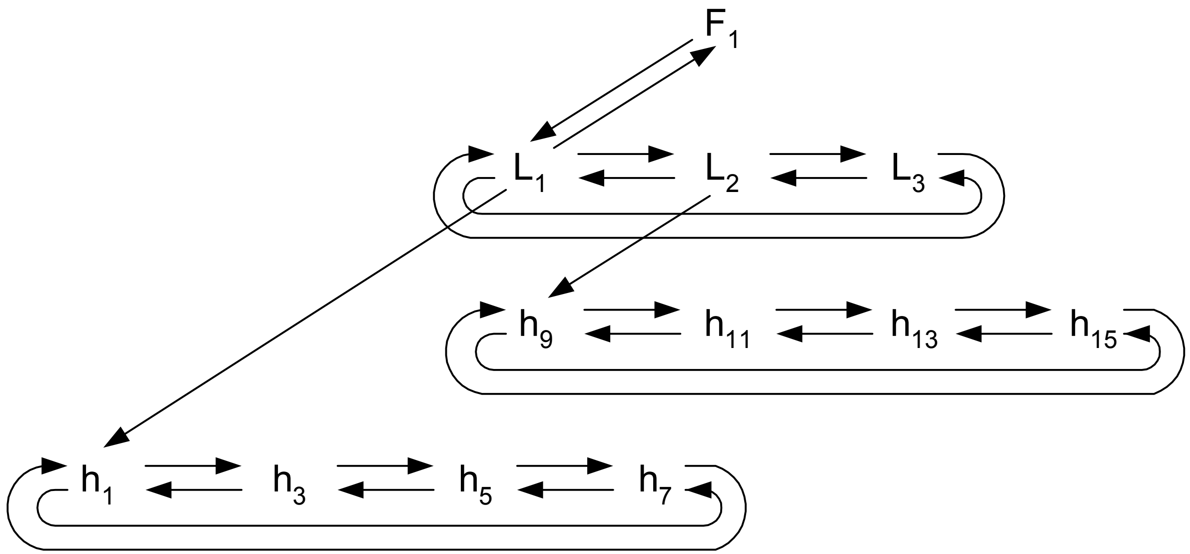
Doubly linked list using half edges

Half Edge Data Structure – cont'

- Face with holes has a peripheral boundary and several inner boundary
  → Attach the inner boundaries to the peripheral

boundary using bridge-edges
  → Introduce the Loop concept

Treatment of a face with holes using loops



Half Edge Data Structure – cont’

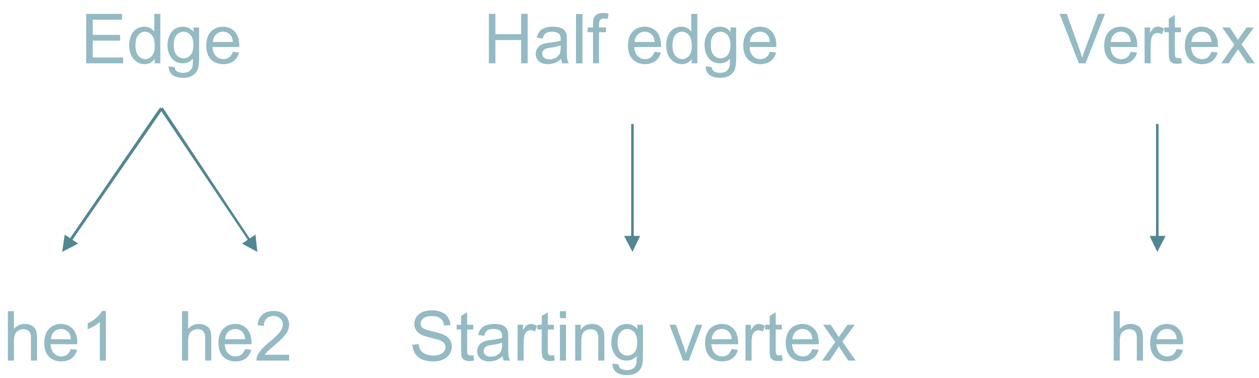
- Assign opposite directions for peripheral boundary and inner boundary

→ Inside of face always exists on the left-hand side as one proceed along the boundary

→ Inside and outside of face is specified

Half Edge Data Structure – cont’

- For connectivity among vertex, edge, face



- Ex) Which loops share a given edge?
  Which edges share a given vertex?

Half Edge Data Structure (represented by C)

```
struct solid
   {
          Id          solidno ;     /* solid identifier */
          Face        *sfaces ;     /* pointer to list of face */
          Edge        *sedges ;     /* pointer to list of edges */
          Vertex      *sverts ;     /* pointer to list of vertices */
          Solid       *nexts ;      /* pointer to next solid */
          Solid       *prevs ;      /* pointer to previous solid */
   } ;
struct face
   {
          Id          faceno ;      /* face identifier */
          Solid       *fsolid ;     /* back pointer to solid */
          Loop        *flout ;      /* pointer to outer loop */
          Loop        *floops ;     /* pointer to list of loops */
          vector      feq ;         /* face equation */
          Face        *nextf ;      /* pointer to next face */
          Face        *prevf ;      /* pointer to previous face */
   } ;
```

Half Edge Data Structure (represented by C) – cont'

```
struct loop
    {
            HalfEdge     *ledg ;      /* ptr to ring of halfedges */
            Face         *lface ;     /* back pointer to face */
            Loop         *nextl ;     /* pointer to next loop */
            Loop         *prevl ;     /* pointer to previous loop */
    } ;
struct edge
    {
            HalfEdge     *he1 ;       /* pointer to right halfedge */
            HalfEdge     *he2 ;       /* pointer to left halfedge */
            Edge         *nexte ;     /* pointer to next edge */
            Edge         *preve ;     /* pointer to previous edge */
    } ;
```

Half Edge Data Structure (represented by C) – cont’

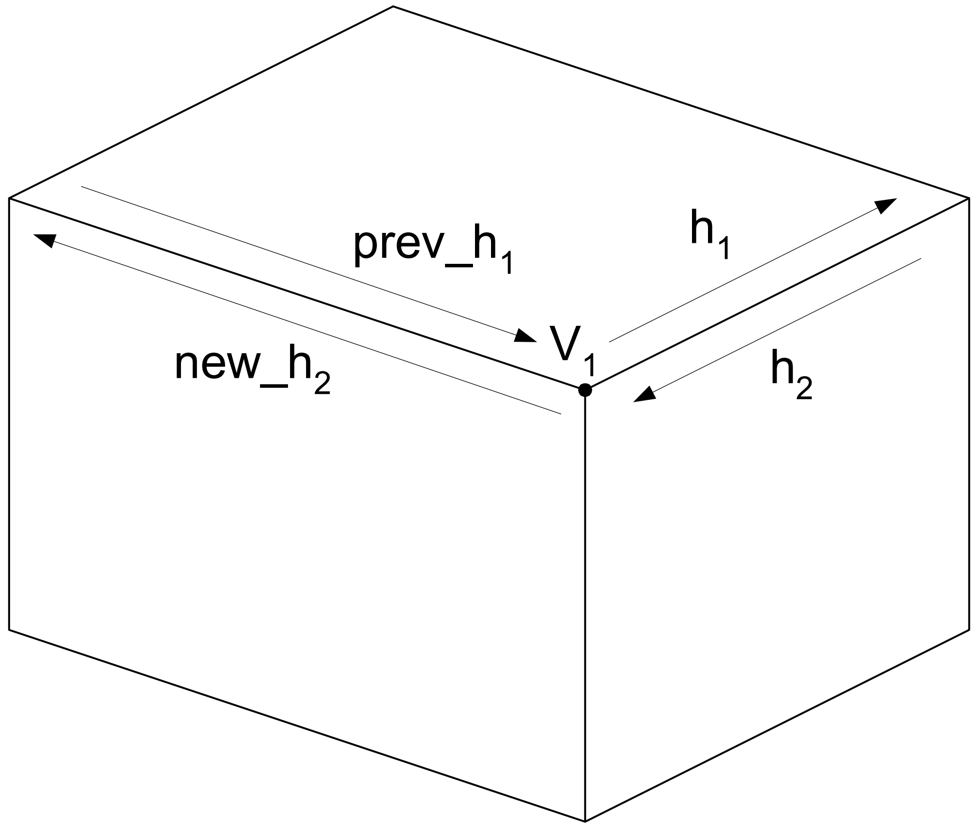
```
struct halfedge
    {
            Edge        *edg ;        /* pointer to parent edge */
            Vertex      *vtx ;        /* pointer to starting vertex */
            Loop        *wloop ;      /* back pointer to loop */
            Halfedge    *nxt ;        /* pointer to next halfedge */
            Halfedge    *prev ;       /* pointer to previous halfedge */
    } ;

struct vertex
    {
            Id          *vertexno ;   /* vertex identifier */
            HalfEdge    *vedge ;      /* pointer to a halfedge */
            vector      *vcoord ;     /* vertex coordinates */
            Vertex      *nextv ;      /* pointer to next vertex */
            Vertex      *prevv ;      /* pointer to previous vertex */
    } ;
```

Half Edge Data Structure (represented by C) – cont’

```
union nodes
    {
            Solid       s ;
            Face        f ;
            Loop        l ;
            HalfEdge    h ;
            Vertex      v ;
            Edge        e ;
    } ;
```

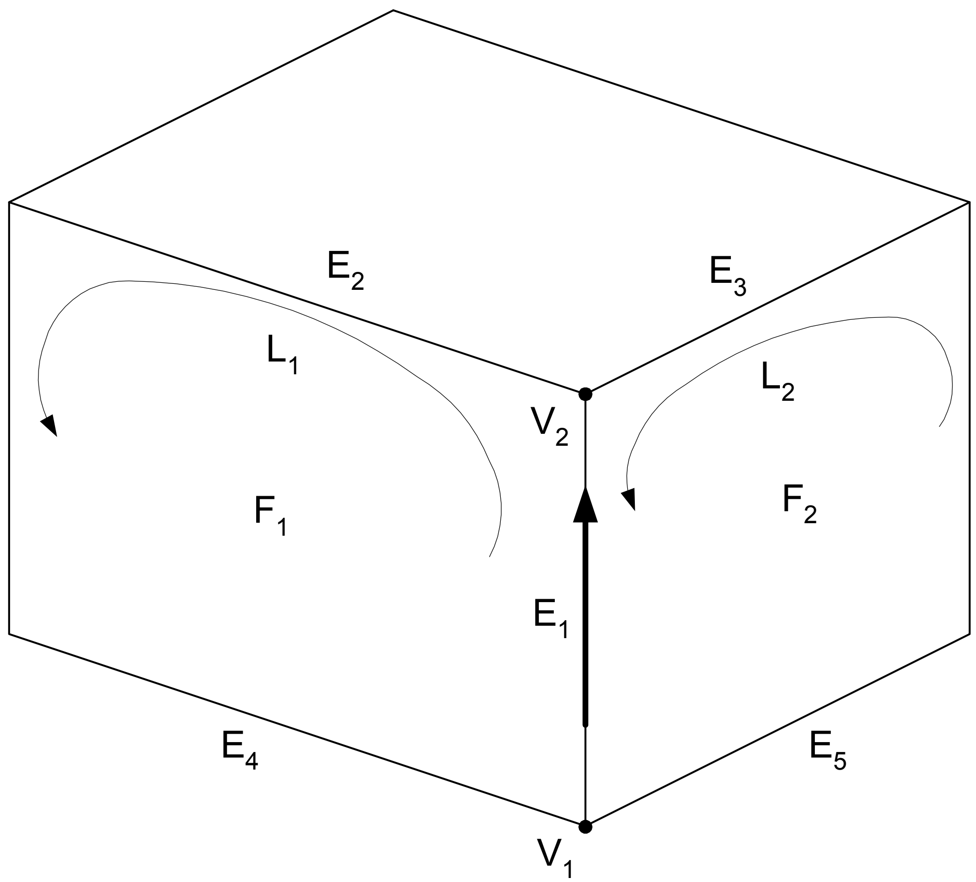
Example of finding an adjacency information between edges and vertices



Winged Edge Data Structure

- Half edge data structure
  - Face is the agent to provide the connectivity
- Winged edge data structure
  - Edge is the agent to provide the connectivity
  - Edge list of faces are derived when needed
  - Proposed by Baumgart in 1974
  - Extended by Braid in 1979
    - Loop concept is introduced to handle faces with holes

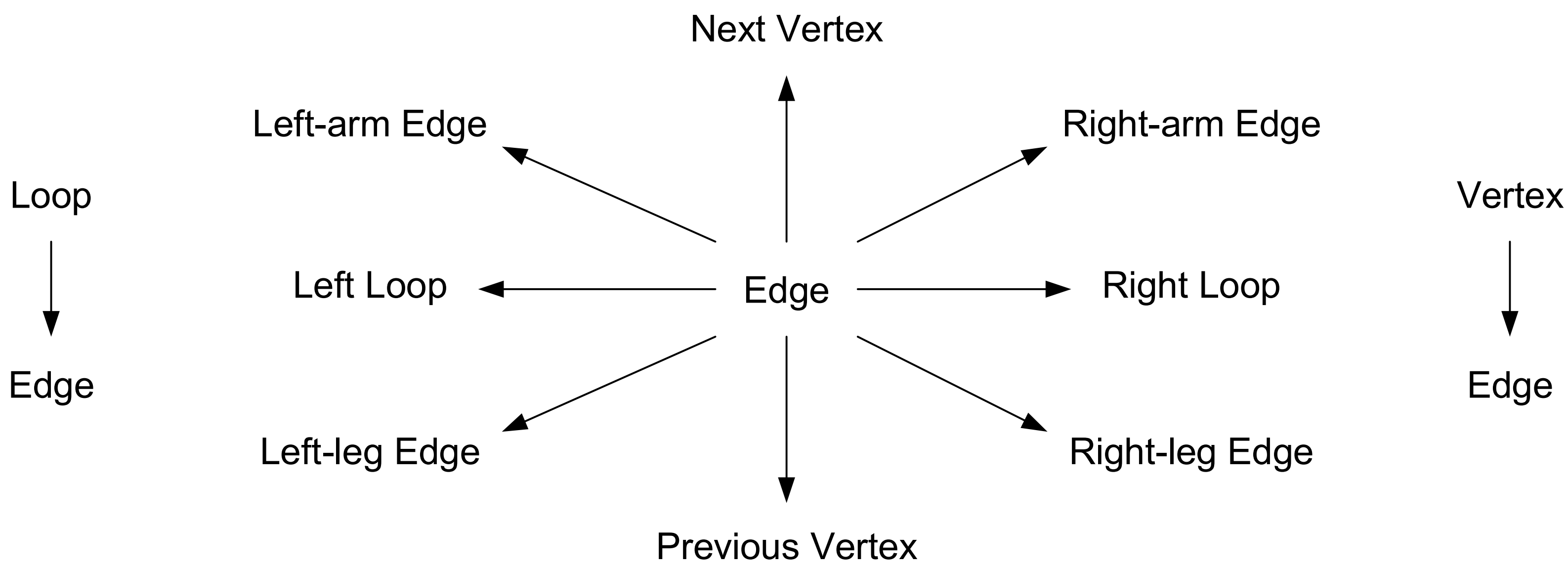
Definition of winged edges



Winged Edge Data Structure – cont'

- E_2, E_3, E_4, E_5: Winged edges of E_1
- Four winged edges stored with specific names
  → connectivity defined explicitly
- Every edge is assigned direction

Connections between vertices, edges, and faces



Winged Edge Data Structure – cont'

- Neighboring faces of an edge have specific names

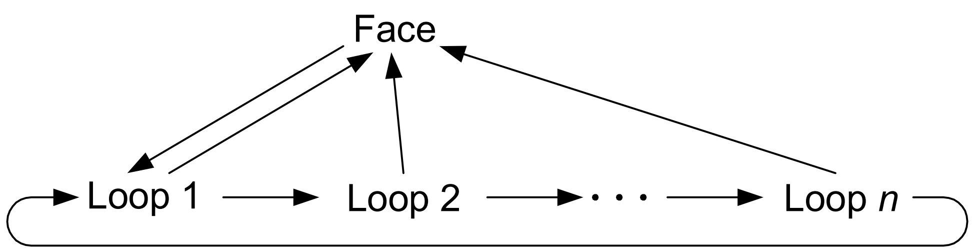
- F_1 Left face
- F_2 Right face

Winged Edge Data Structure – cont'

- Loop is used to handle faces with holes

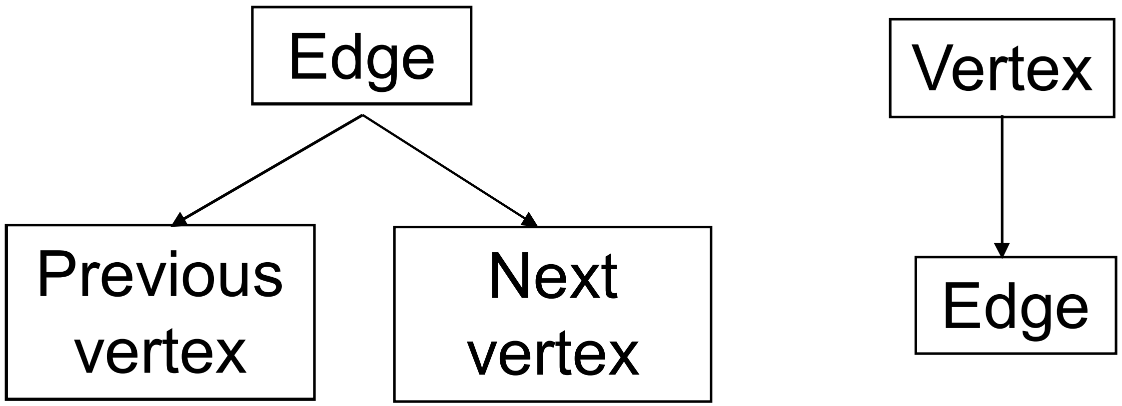
$$L_1 \leftarrow E_1 \rightarrow L_2$$

Left loop Right Loop



Winged Edge Data Structure – cont'

- Every Loop points to any one edge
- Edge list of a loop can be derived by tracing winged edges
- Connectivity between edges and vertices are also stored



Winged Edge Data Structure (represented by C)

```
typedef     struct snu_body      Body;
typedef     struct snu_shell     Shell;
typedef     struct snu_face      Face;
typedef     struct snu_loop      Loop;
typedef     struct snu_edge      Edge;
typedef     struct snu_vertex    Vertex;
typedef     struct snu_surface   Surface;
typedef     struct snu_curve     Curve;
typedef     struct snu_point     Point;

struct snu_body
{
  int       id;        /*body indentifier*/
  Body      *next;     /*pointer to next body */
  Shell     *shell;    /*pointer to shell*/
  Char      *name;     /*pointer  to body name */
};
```

Winged Edge Data Structure (represented by C) – cont'

```
struct snu_shell
{
    int       id;      /*shell indentifier*/
    Body      *body;   /*pointer to body */
    Shell     *next;   /*pointer to next shell*/
    Face      *face;   /*pointer to face*/
};
struct snu_face
{
    int       id;       /*face indentifier*/
    Shell     *shell;   /*pointer to shell*/
    Face      *next;    /*pointer to next face*/
    Loop      *loop;    /*pointer to loop*/
    Surface   *surface; /*pointer to geometry data*/
};
```

Winged Edge Data Structure (represented by C) – cont’

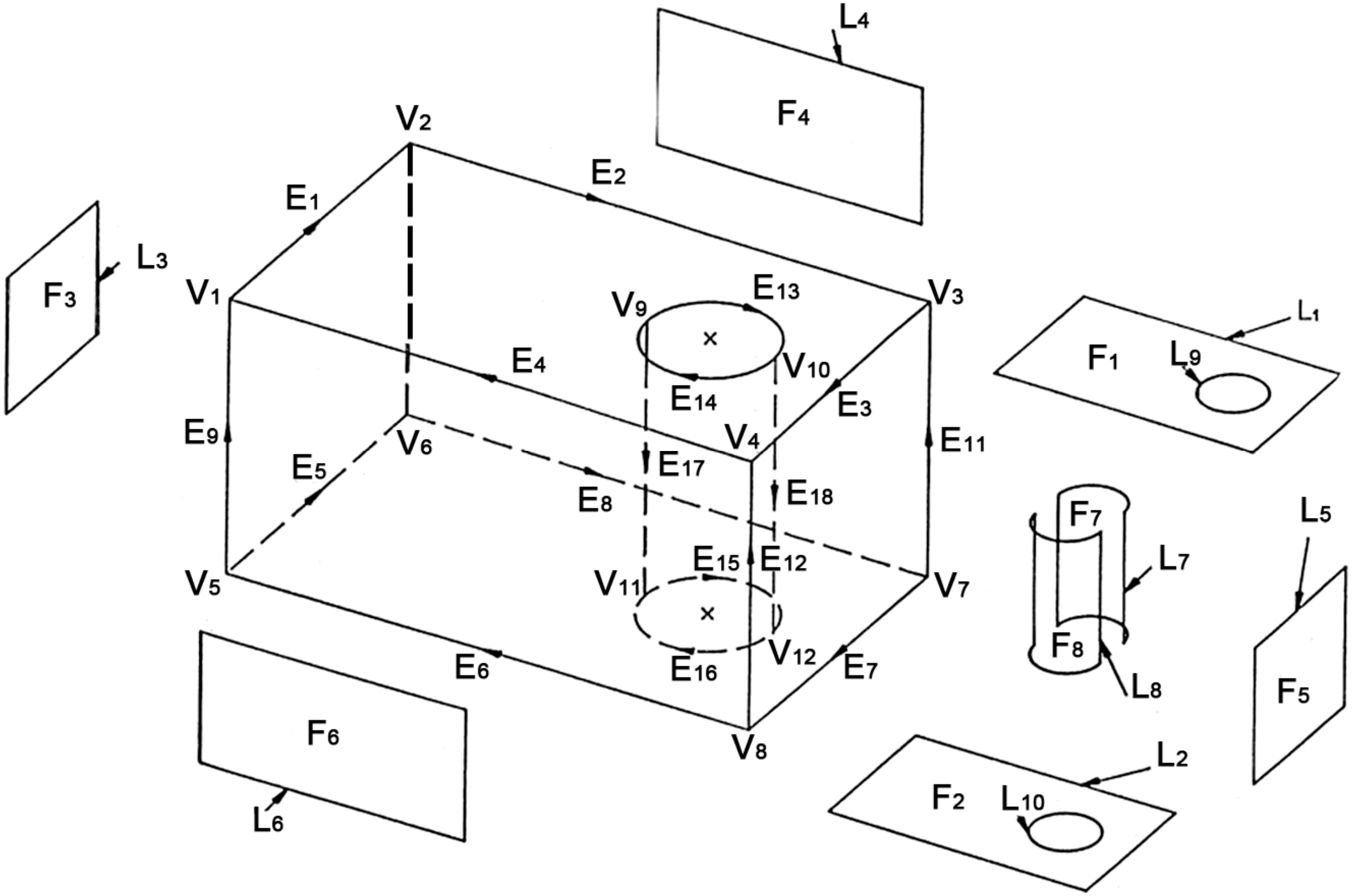
```
struct snu_loop
{
    int         id;      /*loop indentifier*/
    Face    *face;    /*pointer to face*/
    Loop    *next;    /*pointer to next loop*/
    Edge    *edge;    /*pointer to edge */
    int     type  ;   /*loop type*/
};
```

Winged Edge Data Structure (represented by C) – cont'

```
struct snu_edge
{
    int      id;          /*edge indentifier*/
    Loop     *left_loop;   /*pointer to left  loop*/
    Loop     *right_loop;  /*pointer to right loop*/
    Edge     *left_arm;    /*pointer to left arm ( ccw left edge )*/
    Edge     *left_leg;    /*pointer to left leg ( cw left edge )*/
    Edge     *right_leg;   /*pointer to right leg ( ccw right edge )*/
    Edge     *right_arm;   /*pointer to right arm ( cw right edge )*/
    Vertex   *tail_vertex;  /*pointer to tail vertex ( previous vertex)*;/
    Vertex   *head_vertex;  /*pointer to head vertex ( next vertex )*/
    Curve    *curve;       /*pointer to geometry data*/
};

struct snu_vertex
{
    int      id;          /*vertex indentifier*/
    Edge     *edge;       /*pointer to edge*/
    Point    *point;      /*pointer to geometry data */
};
```

Winged Edge Data Structure – cont'



Decomposition Model Data Structure

- Decomposition model:
  - Represent an object as an aggregation of simple objects such as cubes

Voxel model (Exhaustive enumeration)

- Space of interest is represented by a set of cubes (voxels) after being subdivided by grid planes
- Only the voxels embodied by the object are stored
- Use 3D array C(i, j, k), C(i, j, k) corresponding to the embodied voxels is set to 1. Others set to 0
- Popular in digital image processing

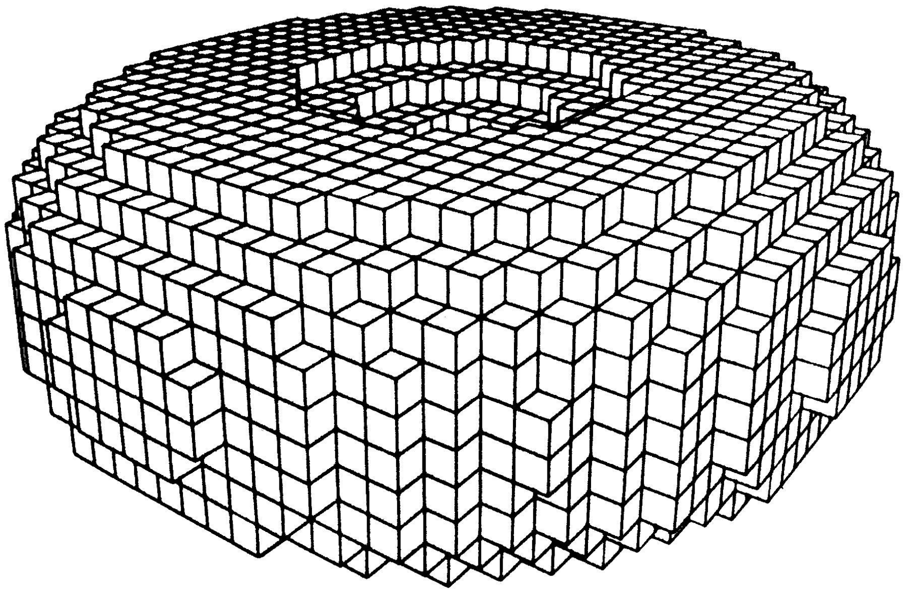
Voxel model – cont'

- Any shape can be represented, approximately at elast
- Used to model human bones and organs from digital topography
- Easy to implement mass property calculation and Boolean operation
- Information on empty space is also available

Voxel model – cont'

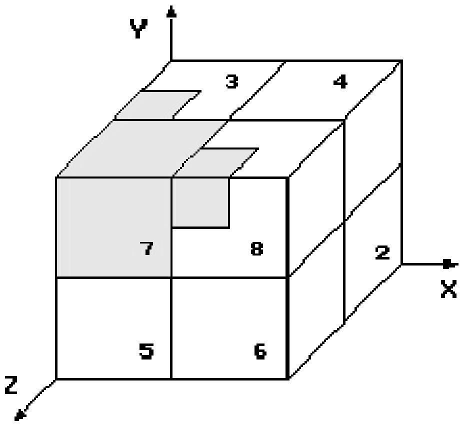
- Memory requirement varies drastically depending upon desired resolution
- Used as a secondary representation for computation convenience

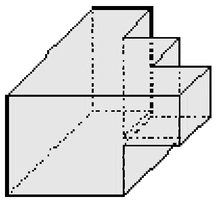
Visualization of voxel representation

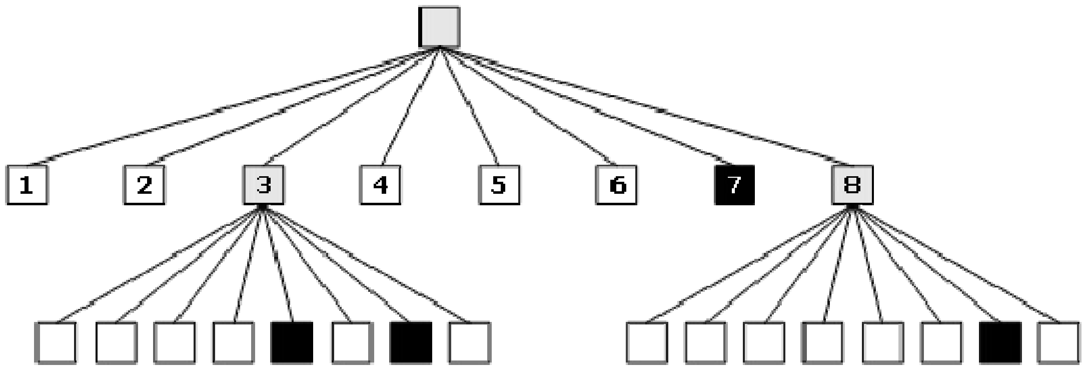


Octree representation

- Only voxels occupying the object space are subdivided, Extension of Quadtree to 3D







Data structure for storing octrees

```
struct      octreeroot
{
     float      xmin, ymin, zmin;             /* space of interest */
     float      xmax, ymax, zmax;
     struct     octree       *root;           /* root of the tree */
};

struct      octree
{
     char                    code;            /* BLACK, WHITE, GREY */
     struct     octree       *oct[8];         /* pointers to octants, present if
     GREY */
};
```

Procedure of octree generation

```
make_tree( p, t, depth )
primitive       *p;             /* p = the primitive to be modeled */
octree          *t;             /* t = node of the octree, initially
    the initial tree with one grey node */
int   depth;    /* initially max. depth of the recursion */
{
    int         i;
    switch( classify( p, t ) )
    {
                case WHITE:
                                t->code = WHITE;
                                break;
                case BLACK:
                                t->code = BLACK;
                                break;
```

Procedure of octree generation – cont’

```
            case GREY:
                        if( depth == 0 )
                        {
                                    t->code = BLACK;
                        }
                        else
                        {
                                    subdivide( t );
                                    for( i = 0; i < 8; i++ )
                                                make_tree( p, t->oct[i],
    depth-1 );
                        }
                        break;
            }
    }
}
/* classify octree node against primitives */
classify( … );
/* divide octree node into eight octants */
subdivide( … );
```

Cell decomposition model