13. Solid Modeling

13.1 Constructive Solid Geometric (CSG) method

- Objects are created as combination of simple primitives, such as rectangular blocks or cylinders, via (regularized) Boolean set operations ($\cup^*$, $\cap^*$, $-^*$)
- The construction process is usually recorded in a CSG tree
- A CSG tree is an ordered binary tree: non-terminal nodes are operators ($\cup^*, \cap^*, -^*$), terminal nodes are primitives.

An example of 2D CSG object:
An example of 3D CSG object:

Non-terminal nodes are also called **composites**
Terminal nodes are also called **primitives**
13.2 Regular Sets & Regular Set Operations

- A set $X$ is a **regular set** if it equals the closure of its interior, i.e.,

$$X = Cl (Int (X))$$

where $Cl$ and $Int$ denote the closure and interior of a set, respectively.

**Example:** sets which are not regular

![Diagram showing dangling face and edge]

- The **regularization** of a set $X$ is the closure of its interior, i.e.,

$$Re(X) = Cl (Int (X))$$

where Re denotes the regularization of a set.
Regularized set operators:

\[ X \ <\ rop > \ Y = Cl (Int ( X \ <\ op > \ Y)) \]

"op": conventional set operator (\(\cup\), \(\cap\) or \(-\))

"rop": regularized set operator (\(\cup^*\), \(\cap^*\), \(-^*\))

Example:

The regularized set intersection of \(A\) and \(B\) below is empty but the conventional intersection of \(A\) and \(B\) is not.
13.3 Data Structures

**Composite:**
- **Name**
- **Operator**
- **L_T_PTR**
- **R_T_PTR**

**Primitive:**
- **Name**
- **Type**
- **Local_To_Global_Transformation**
- **Global_To_Local_Transformation**
- **Surface_PTRs**

where

- **Name:** string identifier, provided by user during creation of the solid
- **Operator:** ∪*, ∩*, or −*
- **Type:** block, cylinder, cone, sphere, wedge, ..., etc
- **Local_To_Global_Transformation:** 4×4 matrix
- **Global_To_Local_Transformation:** inverse of
- **Surface_PTRs:**

Each primitive type has a local coordinate system:
To construct solids, we position primitives in proper location and scale and rotate them to get the desired size and orientation in global coordinate system.

For instance, to position a block in the global coordinate system, we need:

Reference point (R):
X direction coordinate (X):
Y direction coordinate (Y):
Z length:

These parameters are used to construct the Local_To_Global_Transformation so that a copy of the required primitive can be move from the local coordinate system to the desired location (R) in global coordinate system and then rotated and scaled to obtain the required size and orientation.
13.4 Set Membership Classification

Most algorithms that operate on a CSG representation use the technique of divide-and-conquer.

The technique of divide-and-conquer will be used in finding solutions to the following problems:

1) Given a regular set \( X \) (candidate set) and a regular set \( S \) (reference set), find out if \( X \) a member of \( S \).

2) Given a regular set \( X \) and a regular set \( S \), divide \( X \) into parts that are inside, on the boundary of, and outside \( S \).

These are the so-called membership classification problems

Membership classification is the fundamental building block of algorithms in solid modeling
The membership classification function $M[\cdot, \cdot]$ is defined as follows:

$$M[X, S] = (XinS, XonS, XoutS)$$

where

- $S$: reference set
- $X$: candidate set

$$XinS = X \cap^* (Int(S))$$
$$XonS = X \cap^* B(S)$$
$$XoutS = X \cap^* C(S)$$

The classification process is usually performed using the technique of divide-and-conquer, i.e., if $S$ is not a primitive, we classify $X$ with respect to the subtrees of $S$ and then "combine" the results to get the classification of $X$ with respect to $S$. If $S$ is primitive, the problem can not be decomposed further, a "primitive evaluator" is used.
procedure \( M[X, S] \)  
\[
\text{if ( } S \text{ is a primitive ) then} \\
\quad \text{return } Prim-M[X, S] \\
\text{else} \\
\quad \text{combine}(M[X, LT(S)], M[X, RT(S)], Root(S))
\]

Example:

\[
S = A <\text{rop}> B \\
\text{Combine}(M[X,A], M[X,B], <\text{rop}>) \\
\text{Prim-M[ X , B ]} \\
\text{Prim-M[ X , A ]}
\]

The primitive classification procedure "Prim-M" and the marriage procedure "Combine" must be designed.
13.4.1 Point Membership Classification

**P**: a point, \( S \): a CSG object

procedure \( PMC [P,S] \)

\[
\text{if ( } S \text{ is a primitive ) then}
\]

\[
\text{return } Prim -PMC [P,S]
\]

\[
\text{else}
\]

\[
\text{Combine}(PMC [P, LT(S)], PMC [P, RT(S)], Root(S))
\]

Prim-PMC classifies a point against a primitive. This is the trivial part.

"Combine" procedure is to determine whether a point is "in", "on", or "out" a CSG tree (object)

\[
S = A \ <rop > B
\]

by combining the classification of the point with respect to \( A \) and \( B \).
This can be done by using a look-up table. The tables for $\cap^*$, $\cup^*$, and $-^*$ are shown below.

<table>
<thead>
<tr>
<th>$\cap^*$</th>
<th>in</th>
<th>on</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>in</td>
<td>on</td>
<td>out</td>
</tr>
<tr>
<td>on</td>
<td>on</td>
<td>on/out</td>
<td>out</td>
</tr>
<tr>
<td>out</td>
<td>out</td>
<td>out</td>
<td>out</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\cup^*$</th>
<th>in</th>
<th>on</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>in</td>
<td>in</td>
<td>in</td>
</tr>
<tr>
<td>on</td>
<td>in</td>
<td>in/on</td>
<td>on</td>
</tr>
<tr>
<td>out</td>
<td>in</td>
<td>on</td>
<td>out</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$-^*$</th>
<th>in</th>
<th>on</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>out</td>
<td>out</td>
<td>out</td>
</tr>
<tr>
<td>on</td>
<td>on</td>
<td>on/out</td>
<td>out</td>
</tr>
<tr>
<td>out</td>
<td>in</td>
<td>on</td>
<td>out</td>
</tr>
</tbody>
</table>
Ambiguous cases: \( P \) is "on" \( A \) and "on" \( B \) in both cases

\[(a) \quad S = A \cap^* B \]

\( P \) is "on" \( S = A \cap^* B \) in case (a)
\( P \) is "out" \( S = A \cap^* B \) in case (b)

\[(b) \quad S = A \cup^* B \]

\( P \) is "on" \( S = A \cap^* B \) in case (a)
\( P \) is "in" \( S = A \cap^* B \) in case (b)

\[(3) \quad S = A -^* B \]

\( P \) is "out" \( S = A \cap^* B \) in case (a)
\( P \) is "on" \( S = A \cap^* B \) in case (b)

Remedy:
using the *regular neighborhood method*
13.4.2 Line Membership Classification

L: a line, S: a CSG object

procedure $LMC[L, S]$

for each primitive $P$ in $S$ do
intersect $L$ with $P$
enter point(s) of intersections into $PList$

if $PList$ contains more than one point then
order the points by sorting them along $L$

Add the endpoints of $L$ into $PList$ at the ends

for each segment* in $PList$ do

$m$ ← midpoint of the segment

case $PMC[m, S]$ of
"in": add segment to $in-list$
"on": add segment to $on-list$
"out": add segment to $out-list$

* segment: a portion of $L$ defined by two consecutive points of $PList$
Example for LMC

\[ \text{L intersects } A \text{ at: } P_3, P_4 \]
\[ \text{L intersects } B \text{ at: } \text{null} \]
\[ \text{L intersects } C \text{ at: } P_5, P_6 \]
\[ \text{L intersects } C \text{ at: } P_7, P_8 \]

After sorting, \( PList = P_3, P_5, P_4, P_6, P_7, P_8 \)

After adding \( P_1 \) and \( P_2 \) into \( PList \), we have
\[ P_1, P_3, P_5, P_4, P_6, P_7, P_8, P_2 \]
Mid-point of $[P_1, P_3]$ is "out" S
$[P_3, P_5]$ "in"
$[P_5, P_4]$ "in"
$[P_4, P_6]$ "in"
$[P_6, P_7]$ "out"
$[P_7, P_8]$ "in"
$[P_8, P_2]$ "out"

Hence, $\text{in-list} = [P_3, P_5], [P_5, P_4], [P_4, P_6], [P_7, P_8]$  
$\text{on-list} = \text{null}$  
$\text{out-list} = [P_1, P_3], [P_6, P_7], [P_8, P_2]$  

(If necessary, merge connecting segments in the same category into one segment)

After merging,

$\text{in-list} = [P_3, P_6], [P_7, P_8]$  
$\text{on-list} = \text{null}$  
$\text{out-list} = [P_1, P_3], [P_6, P_7], [P_8, P_2]$
13.5 Object Generation and Display

- Using a **ray casting** based approach

### 13.5.1 Ray Casting Model

- For a focal point (view point) and a rectangular pixel array (screen), generate a **ray** through each pixel and find the first surface the ray intersects.

![Diagram of ray casting](image)

**Example of a cast ray.**
Definition of a ray:

\[(x, y, z) = (x_0, y_0, z_0) + t^* (D_x, D_y, D_z)\]

For each ray, the following information will be returned:

**Ray parameters:** \[t[1], t[2], \ldots, t[n]\]

**Surface pointers:** \[S[1], S[2], \ldots, S[n]\]

where \(n\) is the number of ray-solid intersections.

The ordered list of ray parameters denotes the enter-exit points.

The list of surface pointers are pointers to the surfaces through which the ray passes.
13.5.2 How to use the information

- Wire Frame Drawings
- Shaded Images
- Volume Computation

**Wire Frame Drawings**

To generate visible edges of a solid, generate a ray per pixel. Evaluate each ray to identify the visible surface $S[1]$. If the visible surface at pixel $(x, y)$ is different from the visible at pixel $(x-1, y)$, display a vertical line one pixel long centered at $(x-0.5, y)$. If the visible surface at pixel $(x, y)$ is different from the visible at pixel $(x, y-1)$, display a horizontal line one pixel long centered at $(x, y-0.5)$. The resulting drawing will consist of horizontal and vertical edges.

![Given situation]
Shaded Images
To generated a shaded image, cast a ray per pixel in the screen. Use the visible surface pointer \( S[1] \) to access the definition of the surface. Compute the surface normal at the visible point \( t[1] \). The intensity value of the pixel is proportional to the \( \cosine \) of the angle between the surface normal and the light-to-surface vector.

Volume Computation
The volume of a solid can be computed using the "approximating sum" integration method.