How to program it in programming

* can be described using
  - 8 vertexes
  - 12 edges
  - 6 faces
  - 32 dimensional

**Solution 1** - 8 vertexes;

```java
class cuboid {
  int vertex v[8];
}
```

**Solution 2** - 12 Edges → comprise of 2 vertexes → 3 coordinates

```java
class cuboid {
  int Edges [12];
}
class vertex {
  float coordinates [3];
}
```

**Solution 3** - 6 faces

```java
class cuboid {
  Face f[6];
}
class vertex {
  float coordinates [3];
}
class face {
  int vertex v[4];
  int num; // or just <vertex> v
}
```
Solution: Cuboid: length, width, breadth

All cubes cannot be described using only 3, 6, 9 and translation vector.

Here I am just defining something at origin.

Transformation can be used to describe the position where it needs to be visible.

describe it in terms of 3 dimensional vectors.

In two dimensions I need only one angle.

\[
\begin{align*}
\text{angle} & \quad \text{across all axis for} \\
3 \text{ dimension: } x, y, z\\n\end{align*}
\]

We need 3 angles.

Not flexible — Only define cubed.

Storage consumption improved efficiency is also improved for frame.

\[
\begin{align*}
&= 9 \text{ values} \\
\end{align*}
\]

class cuboid

float dimension [3];
float translation vector [3];
float rotation [3];

And: → 2 Way/Methods and structures to describe geometries / models

Wire Frame Models:

Part of 3D design.

Data Structures available: wire frame — edges

But edge is a topological concept

( connection between two points — line)

Curves B splines

(specific way to describe desired curve)

Lot of stuff is missing

We store info about 9 vertices, 16 edges

1. What we don't have is fix it a solid shape, what

2. No info of fore shape, material etc

Early computer graphics when computation problem was restrictive
2. CSG: Constructive Solid Geometry
   - We take primitive cuboid, pyramid
   - Basic operations done to primitive shape
     - union (U) = P1 U P2
     - intersection (N) = P1 ∩ P2
     - difference (−) = P1 − P2

   - We take 1 primitive cuboid

   - Data structure: cuboid, pyramid etc...
     - Position, rotation, scaling
     - Encoding of basic operation (union, intersection)

3. Triangle Meshes: Data Structure (Triangle)

4. Brep: Boundary Representation
   - Describes the volume (which is not defined in triangle mesh)
   - Is more flexible because we can have arbitrary surfaces

   - Data structures: vertex, edge, face (closed loop of edges), shell (boundary of a volume)
Advantage: I get info about volume

1 shell →

\[ f_1 = e_1 e_2 e_3 e_4 \]
\[ f_2 = e_4 e_15 e_16 \]

\[ f_9 = \]

Shell: \( f_1 \ldots f_9 \) (boundaries of my solid shape)

(1 face + 1 extrusion)

5 points + 8 edges + 5 faces

For 1 face an extrusion factor

ANS: 3

Semantically rich model: lot of modelling constructs, it has a lot of meaning encoded. Becomes easier to use. More primitives, more ways to combine to form complex shapes.

- easily understandable by humans
example file → Collada file (closed to BRep) (not used here)

Ans: → 4 - Collada file: bringing together geometry and topological info.
- File format for data description and exchange for automotive applications, e.g., robotics
- Controllable description of geometries at high level functionalities
- XML file that describes Collada data
  - Hierarchical structure: [Mixing of description of data with the data]
  - What it described
    * At: 8 vertices

Technique (description how position array should be read)

Metadata

Geometry mesh [combining different surface parts to make constructive work]

Polygon, e.g., triangle mesh

- In <vertices>
  - It is pointing towards source relationship between branches

- Polygon defines faces (6 polygons)

Cross references are required within an XML document
* Geometrical data is what things are
* Topological is how things are interleaved
Exercise 1

Imp:

Ans: 1

Brep:

Basic Topology:
- vertices: (vid, x, y, z)
- edges: (eid, vid1, vid2)
- faces: (fid, eid1, eid2...)
- shells: (sid, fid, fid2...)

* New construct methods = more semantics
* Easy for user to draw and understand
- CSG and sweeping allows more operation to change and life more easier

CSG:
- shell := (sid, fid, fid2...)
  * unionshell := (sid1, & & sid2)
  * Intersectshell := ( - - - )
  * Minusshell := ( - - - )
  (union class)

Sweeping:
Extrusion shell (sid, fid, x1, y, z) extrusion/protuding

Taking 2 dimension surface and extending into 3 dimm

→ Extrusion = face + vector
→ Relation = face + axis + angle
Winged Edge Data Structure?

- Explain BREP again
- Hierarchical topological concepts: vertexes → edges → faces → volume

→ winged Edge data structure: classic representation of BREP

* Redundant data
- Defines not only the vertexes but also faces and neighboring edges
- For every edge, we store more info

* We need direct connection to the faces for some algorithms to work. They always work with connected data, therefore for the faces, we define the neighboring edges

In programming language: objects are the reference

→ List<We.Edge> → list of edges that define vertex
→ List<We.Edge> → list of edges that define face

* How can the same data be represented in relational database

1. System

<table>
<thead>
<tr>
<th>eid</th>
<th>v1</th>
<th>v2</th>
<th>a1</th>
<th>b1</th>
<th>pver</th>
<th>anext</th>
<th>hprev</th>
<th>bnext</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>e1</td>
<td>v1</td>
<td>v2</td>
<td>s1</td>
<td>f10</td>
<td>e2</td>
<td>e3</td>
<td>e4</td>
<td>e5</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

   One table

2. Fancy table for every class

<table>
<thead>
<tr>
<th>vid</th>
<th>edge</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>v1</td>
<td>e3</td>
<td></td>
</tr>
<tr>
<td>v1</td>
<td>e1</td>
<td></td>
</tr>
<tr>
<td>v1</td>
<td>e4</td>
<td></td>
</tr>
</tbody>
</table>

   Not normalized

   Normalize: require

<table>
<thead>
<tr>
<th>vid</th>
<th>edg</th>
</tr>
</thead>
<tbody>
<tr>
<td>v1</td>
<td>e3</td>
</tr>
<tr>
<td>v1</td>
<td>e4</td>
</tr>
<tr>
<td>v1</td>
<td>e6</td>
</tr>
</tbody>
</table>

We face

<table>
<thead>
<tr>
<th>face</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>f1</td>
<td></td>
</tr>
<tr>
<td>f2</td>
<td></td>
</tr>
</tbody>
</table>

We edge

<table>
<thead>
<tr>
<th>edge</th>
<th>face</th>
</tr>
</thead>
<tbody>
<tr>
<td>e1</td>
<td>f1</td>
</tr>
<tr>
<td>e2</td>
<td>f2</td>
</tr>
<tr>
<td>e3</td>
<td>f3</td>
</tr>
</tbody>
</table>

Vertex edge

<table>
<thead>
<tr>
<th>vid</th>
<th>edg</th>
</tr>
</thead>
<tbody>
<tr>
<td>v1</td>
<td>e3</td>
</tr>
<tr>
<td>v1</td>
<td>e4</td>
</tr>
<tr>
<td>v1</td>
<td>e6</td>
</tr>
</tbody>
</table>
Advantages - less gons in the first approach
  * Could provide performance benefits but lot of inconsistencies

  At every step we store neighboring edges/vertices in order to make algorithms quicker.

Ans. – 3

Some part of STEP AP 203

e.g. how an edge which is a rounded curve in terms of being a spline

* SCHEMA de:finition in terms of Express:
  3 entities — edge curve, bspline curve, bspline curve with knots

* Have some control points that control the curvature
  * Rounded curve is computed based on the control points
  * Rounded curve is a computed partial polyform
  * These polyforms are called knots

  There is a certain control point and passing through it 3 times then I am forcing the curve to form the polyform

* #1518 = table and some references to vertex point

  Edge curve has an edge geometry — difference between topological and geometrical model.
  When mesh edges are defined by vertexes.

  Degree = 3

  Splines are consisted of partial function.
  The partial function is a polynomial of degree
Exercise 8

Exercise 8: Storage of CAD data in RDBMS

<table>
<thead>
<tr>
<th>Metadata</th>
<th>BLOB</th>
<th>Database</th>
<th>Filesystem</th>
<th>Structured Data in Tables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Concurrency</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Consistency</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Performance</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Use CAD systems to manage/manipulate geometrical data.

- RDBMS - state of the art technology.
- Metadata -> info about versioning, who, when, etc.
- Not the geometry itself and is not stored in the database but in CAD files anymore.
- We want to access metadata using SQL for data in CAD system to use/open files easily.

Advantages:
- Can many people access data at the same time.
- Not allowed to do operations in parallel (specifically write operations).
- In DB we can put integrity constraints on metadata.
- We can apply application logic in CAD system. Consistency can be achieved at a certain level. But one can also open a file with other editors and manipulate the file.
- We can quickly access data.

Disadvantages:
- Some data are stored outside database so we don't achieve concurrency and consistency.
* BLOB - Binary Large Object (Huge number of contents of BLOB will be in the file format)
  a) Disadvantage: Whenever we have to view a file it is difficult as we have to write SQL and access the file
  b) Concurrency: Level of control is same as file system because we typically put locks on the file whenever one is accessing it
  c) Consistency: We still don't have control on the inside of BLOB. But we can control whom accessing it. Grant access rights, revoke access rights. Opening from other collisons is not possible
  d) Performance: Will take a little while to read data

* Database File System: Virtual Database file system: (interface on top)
  a) This will be helpful for easy accessible since it no longer requires SQL to fetch the geometrical data.

* Storage as structured data in Tables
  a) Access: Access is not easy due to complex queries
  b) Concurrency: Change of data which is not inter-related
  c) Consistency: No violation of data is possible because of constraints
  d) Performance: Complex join operation to find one particular geometry Data is too complex and ends up in many relationships
Extensions to SQL 92 - structured data types

Ans: 2) SQL 2003 (CAD is object relational database systems)

- Type `geometry with label`
- Type `winged edge` type as a part of `geometry` type

* Difference from SQL 92
  - References: like a pointer, replacement of primary key
  - Arrays
  - Type `Hierarchy`
  - Table Hierarchy

- Inheritance =
  a) UNDER `geometry` type
     - Attributes of edge type will include label
  b) UNDER `geometry` object
     - Select * from `geometry object` → label,
       `geometry object, WE-Edges, WE-Faces`
     - Table Hierarchy means that we have one table
       having all such tables being accessible

Table Hierarchy means that we have one table
having all such tables being accessible

- Performance improves if we
  - use references instead of
    primary key/foreign key
  - because we are still using SQL
  - then we have access issues

Nested Table: possible within object relational system because of types

Advantages:
- just one table (no joins)

Disadvantages:
- Redundancy of data. Nested table models m:n relationship
- No description about geometrical types
  - e.g.: `node` can be `types` → `spines`
Ans! → 3

ACIS model: It's like a library, available software/kernel for modelling geometrical models.

| Left Side | Wire Frame Model |
| Right Side | B Rep Model |
|           | Topological geometrical concept |

→ How do we store objects??

a) Named objects: How to extract data

Programming interface to extract body will be like:

```
SELECT * FROM WHERE body = "body1"
```

Simple mechanism to store objects gives a unique name and fetch it back using unique name.

It is a bad idea if I give all objects name.

b) Persistence by reachability: Good if I also store all objects reachable from named object using pointer and reference.

Advantage → No need to give all objects name, only need to store entry points.

→ If I don't store L4 then it will have dangling references.

→ Best part for entry object is BODY
EXERCISE-9 (DMEA) [Electronic engineering]

Ans. → 1) a) Two phases of design for electronic circuits

- Schematic design: abstract depiction of components, connections, and functionalities, including details of implementing eg.: schematic design
- Design for board layout: sizes, dimensions, conductance paths, and real description of the final product.

b) Requirements to be addressed in both phases:

- Sizes, real geometrical aspects of the board (single, multilevel), paths and connection paths.
- 2D design: design, size of components.

- Schematic connections are important instead of diagrams.
- Behavior of certain components in terms of logic gates for examples, functionalities, properties (charge behavior of a capacitor).

- In schematic connections, area limits are important instead of diagrams.

- Behavior of certain components in terms of logic gates for example, functionalities, properties (charge behavior of a capacitor).

- Functionality description of all important concepts in terms of schematics is required.

- Schematic diagram is a subset of board layout.

- Representing component data and their properties.
- Connection paths (traces), conductance paths, electronic circuit.

- EDA (Electronic Design Automation) tools help moving from schematic to board layout.

Board [size (x, y, z)] - location of pins

- Layout [size of components]

Ans. → 2) DTD to Schemata of XML file

a) Parts and discuss about device = ‘name’

- (device) attribute which requires some reference to the description file

HIMPLIED: - Optional
DTD describes hierarchical structure of a document.

b) what if I have complex components and have new values, how can I describe them.

* create subparts and have several attributes of a part
  `<part>`
  `<attributes>`
  `<attribute>`
  `<attributes>`

  `</attributes>`
  `</part>`

* every part there can be variant and attributes
  `-*` $\rightarrow$ `0...\infty`
  `-+` $\rightarrow$ `1...\infty`
  `-?` $\rightarrow$ `optional(0 or 1)`

Ans.: 3) STEP data $\rightarrow$ semantic of STEP data
  * first part contains the actual data items according to type definitions.

  * component in terms of STEP is interpreted as package
  
  $\rightarrow$ Last two lines don't contain any reference (Cartesian, polar, directed.
  $\rightarrow$ #5116 $\rightarrow$ reference to dimension, Cartesian from
  $\rightarrow$ usage concept $\rightarrow$ establishing relationship between
  generic description of a package and putting at some position.
  schematic $\rightarrow$ board

Reference as what kind of package do we have and where we place it
EXERCISE 10 (DMEA)

Ans. → 1 a) Foundation of Product Lifecycle Management

Marketing → Research → Demand → Development of Product

Product design → Simulation → Testing → Manufacturing → Product

Product Development

* Depending on kind of product we have iterative cycles and loops

Recycle

Disposal → Maintenance → Distribution

(Controlling way of disposing) → (Guaranteed fixation/service) → (How to get things to the customer)

(Reuse of materials)

Entire life span of a product (Product lifecycle)

- Activities like Marketing, Research, requirement analysis and works on certain data that support the product, indirectly. Product Management deals with all the aspects related to the lifecycle.

b) Database systems that cover entire life span of a product is being dealt by product lifecycle management

Why don’t we integrate all data in CAD itself (Design phase)?

By data we mean; structured data, diagrams, market analysis, product files, etc.

* Functionality of CAD system is specific and deep (it describes how the product looks like)

* Lot of functionalities are needed in PLM for all these to be used. We need specific tools/solutions eg CAD, EDA
Main functions of PLM:
- Product Structure Management: What are parts made of?
- Document Management: How are documents related to products?
- Process Management: Guidelines for documents for interaction of people in order to move from one phase to another (e.g., requirement to design, etc. to check)
- User Management: Which users are responsible for carrying out processes/
documentation/ products, i.e., grant rights to users
- Configuration Management: What is the valid configuration of a product? What parts interact well?

It is a very complex task! What kind of version variants could be handled to specify dependencies between several parts?

Status Management - Product status (tested, in development, on production)

* is a relationship (loss of semantics)

Relational Database
Tables: Product, Assemblies, Parts
COP, Consist A, Consist Pa, Consist of ship
Document Management

- Part and document entity type - why do we have n:m relationship?
- One part can be included in many documents showing aspects of that part in different points of view (e.g., requirement phase, conceptual design, cost evaluation).
- Marketing - two characteristics for marketing analysis:
  - Cheaper, cheaper product, distribution of different products using same logistics facilities.
  - Board layout, schedule design.
Ans. 74. a) Configuration Management:

- Lot of products to choose from

b) Variance - describe dependencies between variance and tell if we use V, then it is compatible with a particular variance.

Version - built at different time spans. We need to describe the dependencies within.