We shall use the term "meshed geometry" to denote the discrete data structure used to encode a geometric object, using a mesh.

A "meshed geometry" is a composite structure, containing 2 data structures:

"Meshed geometry" (or "geometry", for short)  "Particles"  "Mesh" (or topology, or connectivity)

example (A quadrilateral mesh)

This geometric object is decomposed as follows:
Particle data structure

<table>
<thead>
<tr>
<th>Particle ID</th>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>(1,1)</td>
</tr>
<tr>
<td>p2</td>
<td>(1,-1)</td>
</tr>
<tr>
<td>p3</td>
<td>(-1,-1)</td>
</tr>
<tr>
<td>p4</td>
<td>(-1,1)</td>
</tr>
<tr>
<td>p5</td>
<td>(3,3)</td>
</tr>
<tr>
<td>p6</td>
<td>(3,-3)</td>
</tr>
<tr>
<td>p7</td>
<td>(-3,-3)</td>
</tr>
<tr>
<td>p8</td>
<td>(-3,3)</td>
</tr>
</tbody>
</table>

Mesh data structure (essentially, a graph)

Symbolically:
On the computer

<table>
<thead>
<tr>
<th>Element (Quad) ID</th>
<th>Vertices</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q₁</td>
<td>(P₁, P₂, P₃, P₄)</td>
</tr>
<tr>
<td>Q₂</td>
<td>(P₁, P₅, P₆, P₇)</td>
</tr>
<tr>
<td>Q₃</td>
<td>(P₂, P₆, P₇, P₃)</td>
</tr>
<tr>
<td>Q₄</td>
<td>(P₃, P₇, P₈, P₉)</td>
</tr>
<tr>
<td>Q₅</td>
<td>(P₄, P₈, P₅, P₁)</td>
</tr>
</tbody>
</table>

Note: We prefer encoding the vertices in a principled way, here, e.g. in clockwise order.

Why "particles" and not just "points" or "vertices"?

A "particle" is a placeholder for more information than just a position coordinate. It may contain:

- Position
- Velocity
- Acceleration, or force
- Mass
- Texture coordinates
- Color, etc

→ Physical attributes
→ Secondary attributes

Here, we used the term "attribute" to denote the constituent physical/practical quantities that comprise a particle.
Implementation | storage of particles

**Approach 1:**

```c
struct Particle {
    float position [3];
    float velocity [3];
    float mass;
};

struct Particle particle_array[N];
```

**#1 Benefits**

- Particles are self-contained
- Easy to construct subsets of particles
- Can extend to accommodate particles with different attributes, on the same array

**#2 Benefits**

- Simulation algorithms typically stream through the arrays of positions & velocities at different possess: keeping them separate exploits bandwidth better.
- Easy to construct a subset of attributes
  - e.g. for visualization, we may only need positions, not velocities or masses.

**Approach 2:**

```c
struct Particles {
    float positions[N][3];
    float velocities[N][3];
    float masses[N];

} particle_array;
```
Other examples:

A surface in 3D (e.g. cloth)

Particles:

\[
\begin{array}{|c|}
\hline
p_1 & (x_1, y_1, z_1) \\
\hline
p_2 & (x_2, y_2, z_2) \\
\hline
\vdots & \vdots \\
\hline
p_q & (x_q, y_q, z_q) \\
\hline
\end{array}
\]

(Triangle) Mesh:

\[
\begin{array}{|c|}
\hline
T_1 & (1, 2, 4) \\
\hline
T_2 & (2, 5, 4) \\
\hline
T_3 & (2, 3, 5) \\
\hline
\vdots & \vdots \\
\hline
T_8 & (6, 9, 8) \\
\hline
\end{array}
\]

(for CW ordering)

A linear segmented curve (in 2D) (e.g. hair, or a crack pattern)

Note: Geometry is disconnected!

Particles:

\[
\begin{array}{|c|}
\hline
p_1 & (x_1, y_1) \\
\hline
p_2 & (x_2, y_2) \\
\hline
\vdots & \vdots \\
\hline
p_{12} & (x_{12}, y_{12}) \\
\hline
\end{array}
\]

Segment Mesh:

\[
\begin{array}{|c|}
\hline
S_1 & (1, 2) \\
\hline
S_2 & (2, 3) \\
\hline
S_3 & (3, 4) \\
\hline
S_4 & (4, 5) \\
\hline
S_5 & (4, 6) \\
\hline
\vdots & \vdots \\
\hline
S_{10} & (11, 12) \\
\hline
\end{array}
\]
Naming conventions (not widely standardized, just for CS381):

"Meshed " "object"

1st part (constituent element type)

\{ Tetrahedralized
  Triangulated
  Segmented
  Polyhedralized
  Polygonized
  Hexahedralized
  Quadrangulated \}

+ \{ Volume
  Surface (3D)
  Area (2D)
  Curve (2D or 3D) \} = Geometry name

e.g. Tetrahedralized volume → 3D object with interior structure, tessellated with tetrahedra.

Triangulated surface → 3D surface composed of triangles, without interior structure.

Polygonized area →

e.g. etc.
Derivative geometries

- A triangle contains 3 segments (edges), and
- A tetrahedron contains 4 triangles (faces) and 6 segments (edges)

As a result, any triangle mesh implicitly defines a segment mesh of its edges or, respectively, any triangulated area implicitly contains a segmented curve.

Likewise, in 3D, e.g.

\[
\begin{array}{|c|c|}
\hline
T_1 & (1, 2, 3, 4) \\
\hline
T_2 & (1, 2, 4, 5) \\
\hline
\end{array}
\]
This volume also contains the following triangle mesh:

\[
\begin{array}{c|c}
  t_1 & (2,3,4) \\
  t_2 & (1,3,1) \\
  t_3 & (1,2,4) \\
  \vdots & \\
  t_6 & (5,1,4) \\
  t_7 & (1,3,4) \\
\end{array}
\]

...or the segment mesh

Observations

- The resulting derivative geometries share the exact same particle array.
- The derivative geometries are not requiring any user input for their construction, i.e., their information is fully dependent on the primary geometry (with the exception of the numbering choice of the derivative elements).
- No need to store them (other than preserving the numbering). Can rebuild them on demand.
When using a mesh-based geometry (with, or without derivative geometries) for a simulation algorithm, questions such as the following often arise:

- Which tetrahedra contain $p_i$ as one of their vertices?
- Which triangles (on a triangulated surface) share a common edge with triangle $t_i$?
- Which segments lie on the boundary of a triangulated area?

**Observations**

- The answers to these queries are independent of the exact particle locations (i.e., queries are purely topological in nature).
- The answers can be precomputed, from the information in the meshed geometry (and its derivatives), e.g.,

  $\text{int } ** \text{incident-tetrahedra}$  $\rightarrow \text{incident-tetrahedra } [i][j]$ is the $j$-th of the tetrahedra that share vertex $p_i$;

  $\text{int } (x \text{adjacent-triangles})[3]$  $\rightarrow \text{adjacent-triangles } [i][j]$ is the $j$-th triangle sharing an edge with $t_i$.