Geometry in Graphics
Applications

Games/Movies

Engineering/Product design
Applications

Medicine/Biology

Architecture
Sources of Geometry

- Acquired real-world objects
  3D Scanning
Sources of Geometry

• Acquired real-world objects
  Point Clouds
Sources of Geometry

• Digital 3D modeling
Sources of Geometry

- Digital 3D modeling
Sources of Geometry

• Procedural Modeling
Geometry Representations

• Considerations
  – Storage
  – Acquisition of shapes
  – Creation of shapes
  – Editing shapes
  – Rendering shapes
Geometry Representations

- Parametric surfaces
Geometry Representations

• Subdivision surfaces
Geometry Representations

- Point set surfaces
Geometry Representations

- Implicit surfaces
Geometry Representations

• Polygonal Meshes
Geometry Representations

• Polygonal Meshes
  – Boundary of objects
  – Geometry + connectivity
Geometry Representations

• Polygonal Meshes
  – Fast rendering with optimized GPU pipeline
Polygonal Mesh

• What is a polygon

\[ v_0, v_1, \ldots, v_{n-1} \]

• Vertices

\[ \{ (v_0, v_1), \ldots, (v_{n-2}, v_{n-1}) \} \]

• Edges

• Planar and non-self-intersecting
Polygonal Mesh

- Set of connected polygons

\[ M = \langle V, E, F \rangle \]
Polygonal Mesh

- Properties

- Every edge belongs to at least one polygon
- The intersection of two polygons in $M$ is either empty, a vertex, or an edge
Polygonal Mesh

- Definitions

- **Vertex degree (Valence):** Number of edges incident to a vertex

- Example: Vertex with a degree of 4
• **Definitions**

- **Vertex degree (valence):** Number of edges incident to a vertex
- **Boundary:** the set of all edges that belong to only one polygon

![Polygonal Mesh Diagram](image)
Manifolds

- Surface locally homeomorphic to a disk
- Closed manifolds divides space into two
Manifold Mesh

• In a manifold mesh, not allowed:
Manifold Mesh

• Real-world data is often non-manifold
Mesh Data Structures

• Store geometry & topology
  – **Geometry**: vertex locations
  – **Topology**: how vertices are connected (edges/faces)
  – **Attributes**: Normal, color, etc.
Mesh Data Structures

• Operations to be supported
  – Rendering
Mesh Data Structures

• Operations to be supported
  – Rendering
  – Geometry queries

Example:
What are the vertices of face $F$?
Mesh Data Structures

• Operations to be supported
  – Rendering
  – Geometry queries
  – Modifications

Example:
Remove vertex $v$. 
Mesh Data Structures

• Triangle List

<table>
<thead>
<tr>
<th>Triangles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vertex Coord.</td>
</tr>
<tr>
<td>((x_0, y_0, z_0))</td>
</tr>
<tr>
<td>((x_3, y_3, z_3))</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

• Simple
• No connectivity
• Redundant
• STL file format
Mesh Data Structures

- Indexed Face Set

<table>
<thead>
<tr>
<th>Triangles</th>
<th>Vertices</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vertex Index</td>
<td>Index Coord.</td>
</tr>
<tr>
<td>1</td>
<td>(x₀, y₀, z₀)</td>
</tr>
<tr>
<td>2</td>
<td>(x₁, y₁, z₁)</td>
</tr>
<tr>
<td>0</td>
<td>(x₂, y₂, z₂)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

- Mesh data structures mention index face set with vertex indices and vertex coordinates.
Mesh Data Structures

• Indexed Face Set
  – Avoids redundancy
  – OBJ, OFF, WRL file formats
  – Stores connectivity, but:
    • Costly geometric queries
    • Costly mesh modifications
Mesh Textures

• Enhancing realism via textures

Lighting & Shading

Texture Mapping
Texture Mapping

• Enhance details without increasing geometric complexity
Texture Mapping

• Issues
  – Mapping between texture and geometry
Texture Mapping

• Issues
  – Anti-aliasing and filtering
Texture Mapping

• Issues
  – Level-of-detail
Texture Mapping

• One-to-one mapping between texture and geometry

Texture Coordinates

World Coordinates

\[ x(u, v), y(u, v), z(u, v) \]
Texture Mapping

- Requires parameterization

\[
\begin{align*}
\mathbb{R}^2 & \quad \Omega \\
(u, v) & \quad \xrightarrow{\text{Texture}} \\
\mathbb{R}^3 & \quad \begin{bmatrix}
x(u, v) \\
y(u, v) \\
z(u, v)
\end{bmatrix} \\
\text{Geometry}
\end{align*}
\]
Parameterization

- Example: sphere

\[ \begin{bmatrix} u \\ v \end{bmatrix} \rightarrow \begin{bmatrix} \sin(u) \sin(v) \\ \cos(v) \\ \cos(u) \sin(v) \end{bmatrix} \]
Parameterization

- Example: sphere

\[
\begin{bmatrix}
  u \\
  v
\end{bmatrix} \rightarrow
\begin{bmatrix}
  \sin(u) \sin(v) \\
  \cos(v) \\
  \cos(u) \sin(v)
\end{bmatrix}
\]
Parameterization

• Desirable properties
  – Low distortion
  – Bijective mapping
  – Efficient to compute
Parameterization

- Desirable properties – low distortion

Texture Map
Texture Mapping

• Additional Issues
  – Finding cuts
  – Texture atlases
Texture Mapping

- Additional Issues - Finding cuts
Texture Mapping

• Additional Issues - Finding cuts
Texture Mapping

• Additional Issues – Texture atlases
Texture Mapping

- **OBJ Files**

  - Vertex positions
  - Texture coordinates
  - Normals
  - Faces (triangles)

  ```plaintext
  v 0.131171 -0.113469 0.178314
  v 0.130945 -0.114951 0.182474
  ...
  vt 0.538446 0.4275
  vt 0.550132 0.41427
  ...
  vn 0.609697 0.486474 0.625789
  vn 0.799934 0.334347 0.498315
  ...
  f 22/209/22 220/210/220
  221/211/221
  f 21/213/21 219/214/219
  220/210/229
  ...
  ```
Texture Mapping

- OpenGL

Load and bind texture:
```c
loadImage(&texture_data);
glGenTextures(1, &texId);
glBindTexture(GL_TEXTURE_2D, texId);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, w, h, 0, GL_RGB, GL_UNSIGNED_BYTE, texture_data);
...```

Fragment shader: texture lookup:
```c
uniform sampler2D texWood;
varying vec2 texCoords;  // [u, v]

void main(void) {
    gl_FragColor = texture2D(texWood, texCoords);
}
```
Texture Mapping

• WebGL

Load and bind texture

```javascript
const texture = gl.createTexture();
gl.bindTexture(gl.TEXTURE_2D, texture);
const image = new Image();
image.onload = () => {
  gl.bindTexture(gl.TEXTURE_2D, texture);
  gl.texImage2D(gl.TEXTURE_2D, level, internalFormat, srcFormat, srcType, image);
  gl.generateMipmap(gl.TEXTURE_2D);
};
```

Fragment shader: texture lookup

```javascript
uniform sampler2D texWood;
varying vec2 texCoords;     // [u, v]

void main(void) {
  gl_FragColor = texture2D(texWood, texCoords);
}
```