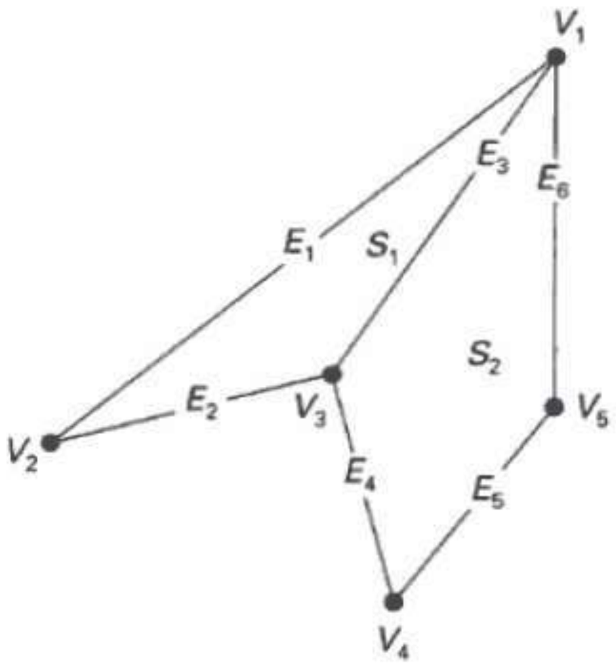
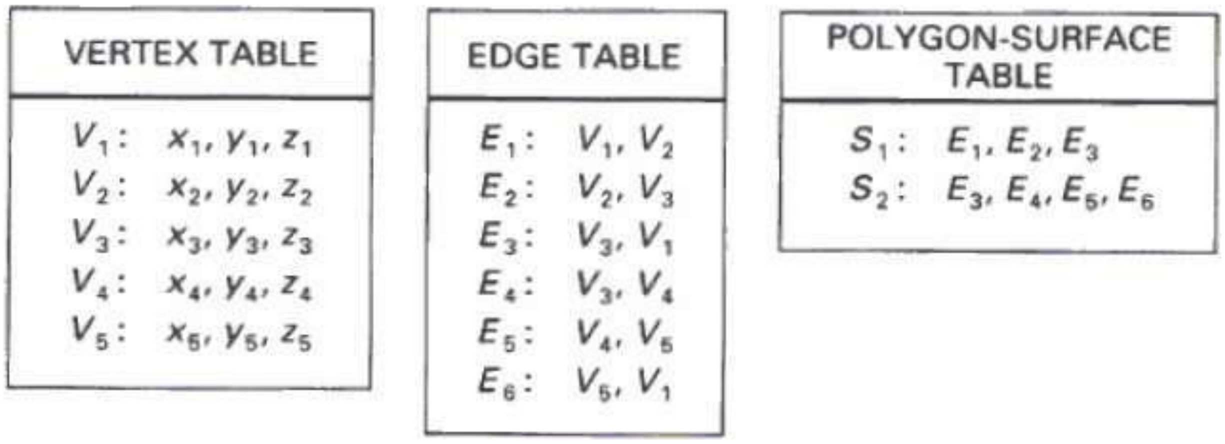
Three dimensional object representations

1. **Polygon Surfaces**:

Polygon surfaces represent 3D objects as a collection of interconnected polygons. These polygons are typically triangles or quadrilaterals, although triangles are more common due to their simplicity and efficiency. Each polygon is defined by a set of vertices, which specify its shape and size in 3D space. Polygon surfaces are widely used in computer graphics and 3D modeling because they are versatile and relatively easy to work with.





VERTEX TABLE	
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

EDGE TABLE	
E_1:	V_1, V_2
E_2:	V_2, V_3
E_3:	V_3, V_1
E_4:	V_3, V_4
E_5:	V_4, V_5
E_6:	V_5, V_1

POLYGON-SURFACE TABLE	
S_1:	E_1, E_2, E_3
S_2:	E_3, E_4, E_5, E_6

Key components of polygon surfaces include:

- **Vertex**: A point in 3D space that defines one corner of a polygon.
- **Edge**: A line segment connecting two vertices.
- **Face (or Polygon)**: A flat surface bounded by edges.

- **Normal**: A vector perpendicular to the surface of a polygon, used for shading and lighting calculations.

Polygon surfaces can represent complex shapes by tessellating them into smaller polygons. They are often stored and manipulated using data structures like vertex arrays or indexed arrays.

2. **Polygon Tables**:

Polygon tables are data structures used to efficiently store and manage information about polygonal meshes. They provide a systematic way to organize and access data related to vertices, edges, and faces in a mesh.

Key components of polygon tables include:

- **Vertex Table**: Stores information about the vertices in the mesh, such as their coordinates and attributes.
- **Edge Table**: Stores information about the edges in the mesh, such as the vertices it connects and adjacent polygons.
- **Face Table**: Stores information about the faces (polygons) in the mesh, such as the vertices that define each face and the edges that bound it.

Polygon tables facilitate various operations on polygonal meshes, such as rendering, collision detection, and mesh editing. They are often implemented using arrays, linked lists, or more complex data structures like half-edge meshes.

In summary, polygon surfaces and polygon tables are fundamental to representing and working with 3D objects in computer graphics and related fields. They provide a flexible and efficient way to model and manipulate complex shapes in three-dimensional space.

Introduction to Hidden Line and Hidden Surface Removal Techniques

Hidden line and hidden surface removal techniques can be classified based on whether they operate in object space (before projection to the screen) or image space (after projection to the screen).

1. **Object Space Methods**:

Object space methods operate on the 3D geometric objects themselves, before they are projected onto the 2D screen space. These techniques analyze the geometry of objects in their original 3D coordinate systems to determine which lines or surfaces are hidden from the viewer's perspective.

Common object space methods include:

  - **Back-face Culling**: This method identifies and eliminates polygons that are facing away from the viewer, assuming they are obscured by other objects in the scene. It relies on the orientation of surface normals relative to the viewer's viewpoint.

- **Depth Sorting**: Objects are sorted based on their distance from the viewer along the viewing direction. This allows the renderer to draw objects from back to front, ensuring that closer objects obscure farther ones.
- **Object-space Partitioning**: Objects or scenes are divided into regions or cells, and visibility is determined within each region. This approach can include techniques like space partitioning using bounding volume hierarchies (e.g., octrees) or spatial subdivision.

2. **Image Space Methods**:

Image space methods operate on the 2D image generated after projection onto the screen space. These techniques analyze the rendered image to determine which lines or surfaces are visible to the viewer and should be displayed.

Common image space methods include:

- **Depth Buffer (Z-buffer) Algorithm**: This method uses a buffer (typically called the Z-buffer) to store the depth (Z-coordinate) of each pixel in the scene. As polygons are rendered, their depths are compared to the values in the Z-buffer, and only the closest fragments are displayed, effectively removing hidden surfaces.
- **Scan-line Algorithm**: This method processes each scan line (horizontal line) of the rendered image, identifying visible segments of lines or surfaces. It can use techniques like edge coherence to optimize visibility determination along scan lines.
- **Painter's Algorithm**: While traditionally considered an object space method, Painter's Algorithm can also be adapted to image space. In this case, the depth of each pixel in the image is determined, and objects are drawn in order from back to front, with later objects potentially overwriting earlier ones.

Three dimensional viewing pipeline

The three-dimensional (3D) viewing pipeline is the process of transforming a 3D scene into a 2D image for display on a screen or other output device. It encompasses several stages that convert geometric information from its original 3D representation into a final 2D image. Here's an overview of the steps involved in the 3D viewing pipeline:

1. **Object Definition**: The 3D scene is defined using geometric primitives such as points, lines, curves, and polygons. Objects may also have associated attributes like color, texture, and material properties.
2. **Modeling Transformation**: Geometric objects are transformed from their local object coordinate systems to a global world coordinate system. This involves operations like translation, rotation, scaling, and shearing, which position and orient objects within the 3D scene.
3. **Viewing Transformation**: The world coordinate system is transformed into a view or camera coordinate system, simulating the position and orientation of a virtual camera or viewer within

the scene. This transformation allows us to specify the viewpoint and direction from which the scene will be viewed.

4. **Projection**: The 3D scene is projected onto a 2D plane to create a perspective or orthographic projection. Perspective projection simulates the way objects appear smaller as they move farther away from the viewer, while orthographic projection maintains the size of objects regardless of their distance from the viewer.
5. **Clipping**: Portions of objects that lie outside the view frustum (the volume representing the visible region of space) are clipped, removing them from further processing. This ensures that only objects within the viewing volume are rendered.
6. **Hidden Surface Removal**: Surfaces or portions of surfaces that are obscured by other objects in the scene are identified and removed. This step ensures that only visible surfaces contribute to the final image.
7. **Rasterization**: The remaining 3D primitives (points, lines, polygons) are converted into 2D pixel values on the screen. This process, known as rasterization or scan conversion, determines which pixels are covered by each primitive and assigns color values based on attributes such as shading, texture mapping, and lighting.
8. **Rendering**: Finally, the 2D image is rendered onto the display screen or other output device. This involves combining the pixel values generated during rasterization to produce the final image that represents the 3D scene from the specified viewpoint.

The 3D viewing pipeline is a fundamental concept in computer graphics, providing a systematic framework for transforming and rendering 3D scenes into 2D images. Each stage of the pipeline plays a critical role in the process, and optimization techniques are often applied to improve performance and visual quality.