Module 3

Windows to Viewport Transformations

This is the process of transforming 2D world-coordinate objects to device coordinates. Objects inside the world or clipping window are mapped to the viewport which is the area on the screen where world coordinates are mapped to be displayed.

Important Terminology

- World Coordinates: This is the cartesian coordinate w.r.t which we define the diagram.
- Device coordinate: It is the screen coordinate where the objects are to be displayed.
- Window: It is the area on the world coordinate selected for display
- Viewport: It is the area on the device coordinate where graphics is to be displayed.

In some case the viewport may not be the same size as the windows coordinate, this is where we need to perform windows to viewport transformation.

There are two main steps for this transformation:

Steps

1: Window coordinate to normalized coordinate translation.
This can be done by the following method

$$X_{normalised}, Y_{normalised} = \frac{X_w - X_{wmin}}{X_{wmax} - X_{wmin}}, \frac{Y_w - Y_{wmin}}{Y_{wmax} - Y_{wmin}}$$

Clipping

When we have to display a large portion of the picture, then not only scaling & translation is necessary, the visible part of picture is also identified. This process is not easy. Certain parts of the image are inside, while others are partially inside. The lines or elements which are partially visible will be omitted.

Applications of clipping:

- It will extract part we desire.
- For identifying the visible and invisible area in the 3D or 2D object.
- For drawing operations.
- For deleting, copying, moving part of an object.

Cohen Sutherland Line Clipping

This is the most popular clipping algorithm

All lines come under any one of the following categories

- Visible: If a line lies within the window, i.e., both endpoints of the line lie within the window. A line is visible and will be displayed as it is.
- Not Visible: If a line lies outside the window, it will be invisible and rejected.
- Clipping Case: If the line is neither visible case nor invisible case. It is considered to be the clipped case. Reject the portion of line that are outside the clipping window and accept portion that are inside the clipping window.

In case of clipping, we need to find the intersecting point of line with window boundary this can be done by finding the point of intersection between the windows boundary lines and the line to be displayed.

This can be done using the formula

$$X = \frac{Y_{wmin} - Y_1}{m} + X_1$$

$$Y = (X_{wmin} - X_1) \cdot m + Y_1$$

where m is the slope of the line in question

Every line endpoint in a picture is assigned a four-digit binary code, called a region code that identifies the location of the point relative to the boundaries of the clipping window. Regions are set up in reference to the boundaries of the clipping window.

Steps to Follow

Step 1 : Assign a region code for two endpoints of given line.

Step 2 : If both endpoints have a region code 0000 then given line is completely inside.

Step 3 : Else, perform the logical AND operation for the region codes of those end points.

- **Step 3.1** : If the result is not 0000, then given line is completely outside.
- **Step 3.2** : Else line is partially inside.
  - **Step 3.2.1** : Choose an endpoint of the line that is outside the given rectangle
  - **Step 3.2.2** : Find the intersection point of the rectangular boundary .
  - **Step 3.2.3** : Replace endpoint with the intersection point and update the region code.
  - **Step 3.2.4** : Repeat step 2 until we find a clipped line either trivially accepted or trivially rejected.

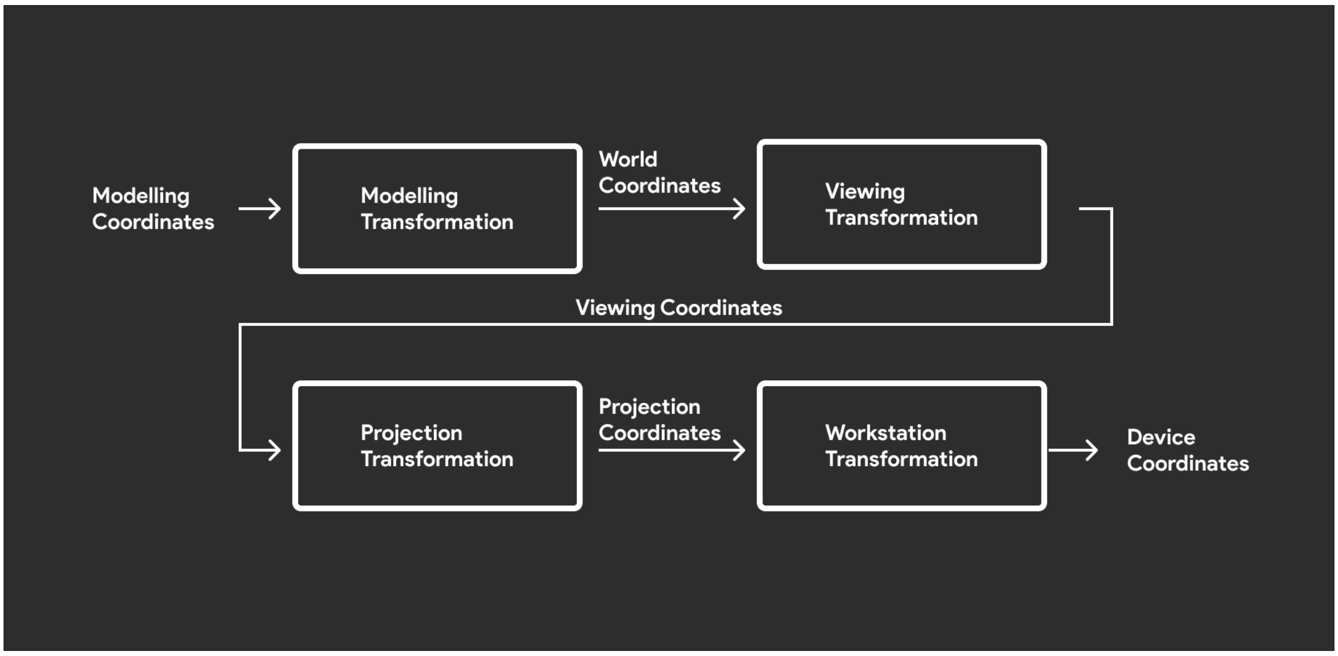
Step 4 : Repeat step 1 for other lines .

Sutherland-Hodgeman Polygon Clipping

When clipping a polygon to a window using the Sutherland-Hodgeman algorithm, the following rules apply: if the first vertex is outside and the second is inside, the intersection point with the window boundary and the second vertex are added to the output list (outside to inside). If both vertices are inside, only the second vertex is added (inside to inside). If the first vertex is inside and the second is outside, the intersection point is added (inside to outside). If both vertices are outside, nothing is added.

TLDR

Just common sense should be fine



Creating a scene involves several steps:
First, build the objects within the modeling coordinate system.
Next, determine the scene dimensions (modeling transformation) and position the objects within the frame (world coordinate).
Set the camera's position and orientation (viewing transformation) and scale the view properly (viewing coordinate).
Adjust the scene to fit the projection plane (projection transformation), then convert this to the camera's image frame (device coordinate).

Projections

Once the world –coordinate descriptions of the objects in the scene are converted to viewing coordinates , we can project the 3D objects onto the 2D view plane.

There are two basic projection methods: **Parallel Projections** and **Perspective Projection**

Perspective Projection
Coordinate positions are transferred to the view plane along lines that converge to a point called the projection reference point (or center of projection). This produces a realistic representation but does not preserve relative proportions.

Parallel Projections

Coordinate positions are transformed to the view plane along parallel lines, it also preserves relative proportions of objects, and accurate views of the various sides of an object are obtained with a parallel projection, but does not give a realistic representation of the 3D object.

Orthographic Projections

In orthographic projection the direction of projection is normal to the projection plane. There are three types of orthographic projections :

- Front Projection
- Top Projection (plan view)
- Side Projection

Isometric Projections

Orthographic projections that show more than one side of an object are called axonometric orthographic projections. The most common axonometric projection is an isometric projection where the projection plane intersects each coordinate axis in the model coordinate system at an equal distance. In this projection parallelism of lines are preserved but angles are not preserved.

Oblique Projection

Oblique projection involves projecting along parallel lines that are not perpendicular to the projection plane. There are two types:

- **Cavalier Projection**: This preserves the lengths of lines perpendicular to the projection plane, with projection lines at a 45-degree angle to the plane.
- **Cabinet Projection**: Lines perpendicular to the viewing surface are projected at half their original size, with projection lines at a 63.4-degree angle to the plane.

Perspective Projections

In perspective projections, distances and angles are not preserved, and parallel lines converge at a single point called the center of projection or projection reference point. There are three types of perspective projections. Lines parallel to the view plane are projected as parallel lines, while lines not parallel to the projection plane converge at a vanishing point. Different sets of parallel lines converge at different vanishing points. The vanishing point for lines parallel to the principal axes of an object is known as the principal vanishing point.

Visible Surface Detection

Visible surface detection, or hidden-surface removal, is the process of identifying and eliminating surfaces of objects that are not visible because they are obscured by closer objects. This ensures a realistic image by removing hidden surfaces. There are two methods to solve hidden surface problems: the object-space method, which works in the physical coordinate system, and the image-space method, which works in the screen coordinate system.

There are a few methods to identify the visible surfaces, like:

- Depth Buffer Algorithm
- Scan Line Algorithm

Depth Buffer Algorithm (Z-Buffer Algorithm)

The Depth Buffer Algorithm, also known as the Z-Buffer Algorithm, is a widely used technique in computer graphics for hidden surface removal. This method ensures that the correct surfaces are visible in a rendered scene by keeping track of the depth of objects at each pixel position on the screen. Here's a detailed explanation of how it works:

1. **Initialization**: Two buffers, the depth buffer (Z-buffer) and the frame buffer, are created and initialized: the depth buffer is set to the maximum depth value (usually 1.0 in normalized coordinates) to signify the farthest distance, while the frame buffer is initialized with the background color.
2. **Processing Each Polygon**: The algorithm computes the depth (z-value) of each pixel (x, y) covered by every polygon in the scene using the polygon's plane equation.
3. **Depth Comparison and Update**: The algorithm compares the calculated depth value with the existing depth value stored in the depth buffer for each pixel, and if the calculated depth is less (indicating the polygon is closer to the viewer), it updates both the depth buffer and the frame buffer with the new depth and color values.
4. **Final Image**: After all polygons have been processed, the frame buffer contains the color values of the visible surfaces, and the depth buffer contains the depth values of these surfaces.

This process ensures that only the closest (visible) surfaces are rendered, resulting in a correctly displayed scene.
The depth of a polygon at a given pixel can be calculated using the plane equation:

$$Ax + By + Cz + D = 0$$

Solving for z

$$z = \frac{-(Ax + By + D)}{C}$$

For incremental calculations, the depth at the next pixel can be computed efficiently:

$$z' = z - \frac{A}{C}$$

Along the Z axis:

$$z' = z + \frac{A/m + B}{C}$$

This allows the algorithm to update depth values with minimal computation.

The Scan Line Algorithm

This is an image-space method for identifying visible surfaces, extends the polygon filling algorithm by processing multiple surfaces simultaneously using depth information for each scan line, requiring the grouping and processing of all polygons intersecting a given scan line before moving to the next, and maintaining two important tables:

- The Edge Table (containing line endpoints, inverse slopes, and polygon pointers) and
- The Polygon Table (containing plane coefficients, color information, and a flag).

Algorithm:

1. All polygons (surfaces) intersecting a scan line are processed from left to right.
2. If there are overlapping faces, the algorithm determines the depth to identify the nearest face to the view plane, and then enters the intensities of that face into the refresh buffer.
3. An active list of edges (edges intersecting the scan line) for each scan line is maintained, sorted in the order of increasing x values.