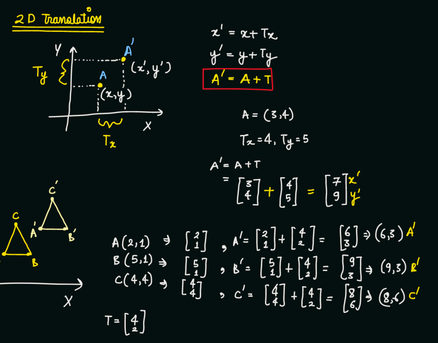
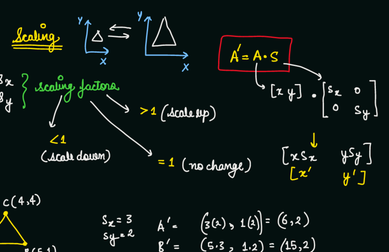
Unit 04

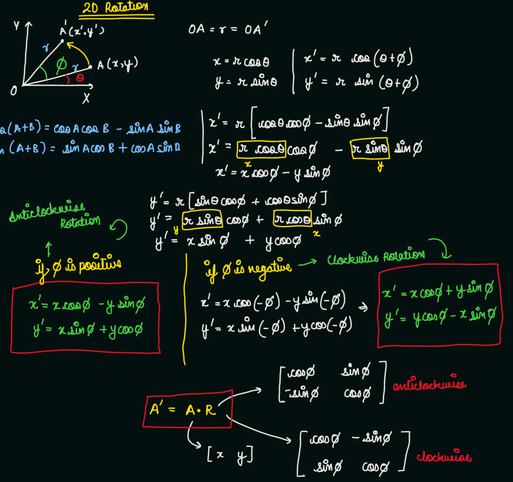
Monday, May 8, 2023 9:49 AM

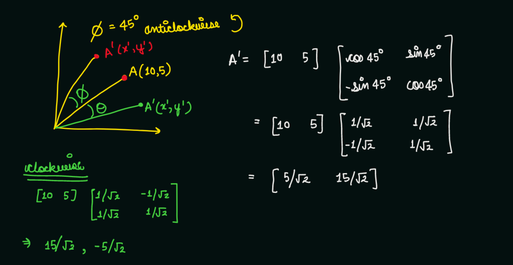
2D Transformation:
- 2D transformation,
- Basic Transformations,
- Composite transformations:
  - Reflection,
  - Shearing,
  - Transformations between coordinate systems.

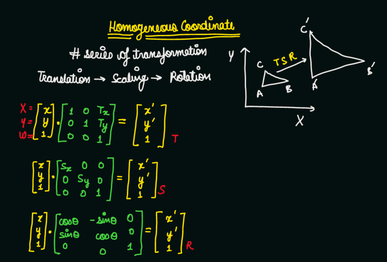
3D Transformation:
- 3D transformations,
- Parallel projection,
- Perspective projection,
- Visible lines and surfaces identification,
- Hidden surface removal algorithms.











Unit: Computer Graphics - 2D and 3D Transformations

1. 2D Transformation:

- 2D Transformation: In computer graphics, 2D transformations refer to the operations that modify the position, size, orientation, or shape of 2D objects. These transformations include translation, rotation, scaling, and

- Basic Transformations:
  - Translation: Moving an object in a specified direction by a certain
  - Rotation: Rotating an object around a fixed point or axis by a given
  - Scaling: Changing the size of an object by multiplying its coordinates

$x' = x + tx$, $y' = y + ty$

$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$

$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} Sx & 0 \\ 0 & Sy \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$

Reflection on X axis: $x' = x$, $y' = -y$

$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$

Reflection on Y axis: $x' = -x$, $y' = y$

$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$

Shearing

$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & Shx \\ Shy & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$

- Composite Transformations: Composite transformations involve combining multiple basic transformations to achieve complex transformations.

  - Reflection: Creating a mirror image of an object across a specified axis or line of reflection.
  - Shearing: Distorting the shape of an object by displacing its points along a specified direction.

- Transformations between Coordinate Systems: Transforming coordinates from one coordinate system to another, such as from object space to world space or from world space to screen space.

2. 3D Transformation:

- 3D Transformations: Similar to 2D transformations, 3D transformations involve modifying the position, size, orientation, or shape of 3D objects. These transformations include translation, rotation, scaling, and shearing in three-dimensional space.

- Parallel Projection: Parallel projection is a method used to project 3D objects onto a 2D plane without accounting for perspective. It maintains parallel lines but does not provide a realistic sense of depth.

- Perspective Projection: Perspective projection is a method used to project 3D objects onto a 2D plane while accounting for perspective and creating a sense of depth. It simulates how objects appear in the real world.

- Visible Lines and Surfaces Identification: In 3D graphics, identifying visible lines and surfaces is crucial for rendering realistic scenes. Various algorithms, such as the Painter's algorithm and the Z-buffer algorithm, are used to determine which lines and surfaces should be displayed.

- Hidden Surface Removal Algorithms: Hidden surface removal algorithms are used to determine which surfaces are occluded or hidden from view by other surfaces. Examples include the Depth Sort algorithm, the Scanline algorithm, and the BSP (Binary Space Partitioning) tree algorithm.

Importance of hidden line and surface Removal Algorithm

The importance of hidden line and surface removal algorithms in computer graphics cannot be overstated. These algorithms play a crucial role in creating realistic and visually appealing 3D renderings by determining which lines and surfaces should be visible or hidden in a given scene.

1. Enhancing Realism: Hidden line and surface removal algorithms are essential for achieving realism in 3D graphics. By accurately determining which lines and surfaces should be visible to an observer, these algorithms help create a more immersive and believable visual experience.

2. Improving Visualization: When rendering complex scenes with overlapping objects or intersecting surfaces, it can be challenging to discern the individual components without proper hidden line and surface removal. These algorithms improve the clarity and comprehensibility of visual representations by eliminating unnecessary clutter and revealing the intended structure of the scene.

3. Optimizing Performance: Rendering all lines and surfaces, including those that are obscured or hidden, can be computationally expensive and inefficient. Hidden line and surface removal algorithms help optimize rendering performance by reducing the amount of unnecessary calculations and rendering operations, resulting in faster and smoother visualizations.

4. Enabling Efficient Editing: In applications where editing or modifying 3D scenes is required, hidden line and surface removal algorithms facilitate a more efficient workflow. By accurately identifying hidden components, these algorithms aid in selecting, manipulating, and modifying objects without interference from obscured or occluded elements.

5. Supporting Visualization Techniques: Hidden line and surface removal algorithms are crucial for supporting various visualization techniques such as scientific visualization, architectural design, virtual reality, and augmented reality. These techniques heavily rely on accurate rendering of 3D scenes and the elimination of hidden lines and surfaces to provide a clear and meaningful representation of complex data.

6. Facilitating Communication and Design Evaluation: Hidden line and surface removal algorithms are invaluable in fields like industrial design and architecture, where communicating design concepts and evaluating spatial relationships are essential. By eliminating hidden lines and surfaces, these algorithms enable designers and stakeholders to better assess the visual aesthetics, functionality, and practicality of a proposed design.

Painter's Algorithm:
- The Painter's algorithm is a simple and intuitive hidden surface removal technique.
- It works by rendering objects from back to front based on their distance from the viewer.
- Objects that are closer to the viewer are rendered last, appearing on top and hiding the objects behind them.
- The Painter's algorithm is suitable for scenes without complex overlapping or transparency.
- However, it suffers from issues like "Z-fighting" (fluctuation of visible surfaces due to precision limitations) and incorrect rendering order for intersecting or occluded objects.
- It is relatively easy to implement but may not provide accurate results in all situations.

When it comes to hidden surface removal algorithms in computer graphics, two commonly used approaches are the Z-buffer algorithm and the A-buffer algorithm.

Z-Buffer Algorithm:
- The Z-buffer algorithm is a simple and efficient method for hidden surface removal.
- It utilizes a depth buffer, also known as a Z-buffer, to store the depth (distance) information of each pixel in the scene.
- During rendering, the algorithm compares the depth value of each pixel being processed with the corresponding depth value stored in the Z-buffer.
- If the depth value of the new pixel is closer to the viewer, it is considered as the visible surface and its color is stored in the frame buffer, and the Z-buffer is updated with the new depth value.
- Advantages:
  - Fast and straightforward implementation.
  - Works well for scenes with moderate to high depth complexity.
  - Requires minimal additional memory compared to other algorithms.
- Limitations:
  - Can suffer from artifacts like aliasing and pixelation.
  - In scenes with extreme depth complexity, Z-buffer precision issues may occur, leading to visual artifacts known as Z-fighting.
  - Inefficient for scenes with large numbers of overlapping polygons.

A-Buffer Algorithm:
- The A-buffer algorithm is a more memory-intensive but versatile approach for hidden surface removal.
- It maintains a separate buffer called the accumulation buffer (A-buffer) for each pixel in the scene.
- Instead of storing only the depth value as in the Z-buffer algorithm, the A-buffer stores all the attributes (e.g., color, transparency, normals) of the surfaces that contribute to a pixel.
- During rendering, the algorithm keeps track of the surfaces at each pixel by storing their attributes in the A-buffer.
- When multiple surfaces intersect at a pixel, the algorithm uses merging and sorting techniques to determine the visible surface and its corresponding attributes.
- Advantages:
  - Provides more accurate results and can handle complex scenes with overlapping polygons.
  - Resolves aliasing and Z-fighting issues more effectively than Z-buffer.
  - Enables advanced effects like transparency and anti-aliasing.
- Limitations:
  - Requires significantly more memory compared to the Z-buffer algorithm.
  - The sorting and merging operations can be computationally expensive.
  - Implementation complexity is higher compared to the Z-buffer algorithm.

1. Rotation around the X-Axis:
To derive the rotation matrix for rotation around the X-axis, we assume an angle of rotation θ.

The general form of the rotation matrix around the X-axis is:

```
[ 1    0        0        0 ]
[ 0    cos(θ)   -sin(θ)   0 ]
[ 0    sin(θ)   cos(θ)    0 ]
[ 0    0        0        1 ]
```

$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

This matrix rotates a point in 3D space around the X-axis by an angle θ.

2. Rotation around the Y-Axis:
To derive the rotation matrix for rotation around the Y-axis, we assume an angle of rotation θ.

The general form of the rotation matrix around the Y-axis is:

```
[ cos(θ)    0    sin(θ)    0 ]
[ 0         1    0         0 ]
[-sin(θ)    0    cos(θ)    0 ]
[ 0         0    0         1 ]
```

$\begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

This matrix rotates a point in 3D space around the Y-axis by an angle θ.

3. Rotation around the Z-Axis:
To derive the rotation matrix for rotation around the Z-axis, we assume an angle of rotation θ.

The general form of the rotation matrix around the Z-axis is:

```
[ cos(θ)   -sin(θ)   0    0 ]
[ sin(θ)    cos(θ)   0    0 ]
[ 0         0        1    0 ]
[ 0         0        0    1 ]
```

$\begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

This matrix rotates a point in 3D space around the Z-axis by an angle θ.

These are the general forms of the rotation matrices around the X, Y, and Z axes. Understanding these matrices and the principles of rotation around different axes will help you solve problems and apply these concepts in exams or assessments related to computer graphics or 3D transformations.

1. Oblique Parallel Projection Matrix:
Oblique parallel projection is a type of projection where the projection rays are parallel to each other but at an oblique angle to the projection plane. To derive the oblique parallel projection matrix, we need to define the parameters of the projection:

Let's assume the oblique angle to the projection plane is α, the projection plane is located at Z = d, and the orthographic projection planes are defined as:
- Left plane: x = x_min
- Right plane: x = x_max
- Bottom plane: y = y_min
- Top plane: y = y_max

The oblique parallel projection matrix can be derived as follows:

Translation matrix T1:

```
[1   0   0   0]
[0   1   0   0]
[0   0   1   -d]
[0   0   0   1]
```

$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -d \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Shear matrix S:

```
[1   0   -cot(α)   0]
[0   1   0         0]
[0   0   1         0]
[0   0   0         1]
```

$\begin{bmatrix} 1 & 0 & -\cot\alpha & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Scaling matrix Sc:

```
[2/(x_max - x_min)   0                   0          0]
[0                   2/(y_max - y_min)   0          0]
[0                   0                   1          0]
[0                   0                   0          1]
```

$\begin{bmatrix} \frac{2}{x_{max} - x_{min}} & 0 & 0 & 0 \\ 0 & \frac{2}{y_{max} - y_{min}} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Translation matrix T2:

```
[1   0   0   -(x_max + x_min)/2]
[0   1   0   -(y_max + y_min)/2]
[0   0   1   0]
[0   0   0   1]
```

$\begin{bmatrix} 1 & 0 & 0 & -(x_{max} + x_{min})/2 \\ 0 & 1 & 0 & -(y_{max} + y_{min})/2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

The oblique parallel projection matrix P can be obtained by multiplying these matrices in the following order:
P = T2 * Sc * S * T1

The resulting matrix P is the oblique parallel projection matrix.

2. Perspective Projection Matrix:
Perspective projection is a type of projection that simulates the way objects appear smaller as they move away from the viewer, creating a sense of depth. The perspective projection matrix can be derived using the following parameters:

Let's assume the parameters for the field of view (FOV), aspect ratio (width/height), near clipping plane (z_near), and far clipping plane (z_far).

The perspective projection matrix can be derived as follows:

Aspect ratio calculation:
aspect = width / height

Cotangent of half the field of view:
cot_half_fov = 1 / tan(FOV / 2)

The perspective projection matrix can be defined as:

```
[ cot_half_fov/aspect   0             0                                   0                          ]
[ 0                     cot_half_fov  0                                   0                          ]
[ 0                     0             (z_far + z_near)/(z_near - z_far)   (2 * z_far * z_near)/(z_near - z_far) ]
[ 0                     0             -1                                  0                          ]
```

This matrix performs the perspective projection transformation on the 3D coordinates.

These are the derived matrices for oblique parallel projection and perspective projection. Understanding the derivation process and the parameters involved will help you grasp the concepts better and apply them in exams or assessments related to computer graphics.