Chapter 4
Geometric Objects & Transformations

 Scalars, Points, Vectors

- Scalars = real numbers
- A point is a fixed location in space
  - Need a coordinate system to specify location
    - Cartesian coordinate
- A vector indicates direction in space
  - has magnitude / length
  - does not have a fixed location
  - Unit vector: magnitude = 1

Notation

- Point \( A (5, -3) \)
- Vector \( v = \begin{bmatrix} 4 \\ 3 \end{bmatrix} \)

Length of \( v \): \( |v| = \sqrt{4^2 + 3^2} = \sqrt{25} = 5 \)

Operations

- Point-Point subtraction \( v = P - Q \)
- Point-Vector addition: point displacement \( Q + v = P \)
- Scalar-Vector multiplication \( s v = w \)
- Vector-Vector addition \( t = u + v \)

Vector “Multiplication”: Dot Product

- Dot Product / Inner Product \( s = v \cdot w = |v||w| \cos(\Theta) \)
- \( a \cdot b = 0 \) if \( a \) is perpendicular to \( b \)
- \( v = \begin{bmatrix} a \\ b \\ c \end{bmatrix}, \ \ w = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \)
  \( v \cdot w = ax + by + cz \)

How to calculate dot product in 3D?

Vector “Multiplication”: Cross Product

- Cross Product / Outer Product \( u = v \times w \)
  \( |u| = |v||w| \sin(\Theta) \)
  \( u \) is perpendicular to both \( v \) and \( w \)
- Three dimensional vectors

\[
\begin{align*}
v &= \begin{bmatrix} a \\ b \\ c \end{bmatrix}, \ \ w &= \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \ \ u &= v \times w = \begin{bmatrix} by - cz \\ cz - ax \\ ax - by \end{bmatrix}
\end{align*}
\]
Geometric Interpretation

- Dot product \( \mathbf{a} \cdot \mathbf{b} = |\mathbf{a}| |\mathbf{b}| \cos \alpha \)
  - \( |\mathbf{b}| \cos \alpha \) is the length of \( \mathbf{b} \) projection to \( \mathbf{a} \)
- Cross product \( \mathbf{a} \times \mathbf{b} \):
  - \( |\mathbf{a} \times \mathbf{b}| = |\mathbf{a}| |\mathbf{b}| \sin \alpha \) is the area of parallelogram formed by \( \mathbf{a} \) and \( \mathbf{b} \), or twice the area of triangle \( \text{OAB} \)

Lines

- Slope-intercept form: \( y = mx + c \)
  - Gradient vector: \( [1 \ m]' \)
  - Normal vector: \( [-m \ 1]' \)
- Standard form equation: \( ax + by = c \)
  - Normal vector \( [a \ b] \)

Lines: Parametric Equation

- Any point \( \mathbf{Z} \) on the line can be expressed as \( \mathbf{Z}(\alpha) = \mathbf{P} + \alpha (\mathbf{Q} - \mathbf{P}) \)
- Parameter \( \alpha \) determines position of \( \mathbf{Z} \)

Parametric Equation of Straight Lines

\[
\begin{align*}
\mathbf{Z}(\alpha) &= \mathbf{P} + \alpha (\mathbf{Q} - \mathbf{P}) \quad \text{or} \quad \mathbf{Z}(\alpha) = \mathbf{P} + \alpha \mathbf{v} \\
\mathbf{Z}(\alpha) &= (1-\alpha) \mathbf{P} + \alpha \mathbf{Q}
\end{align*}
\]

\[
\begin{align*}
\mathbf{Z}(0) &= \mathbf{P} \\
\mathbf{Z}(1) &= \mathbf{Q} \\
\mathbf{Z}(-0.4) &= ? \\
\mathbf{Z}(2.6) &= ?
\end{align*}
\]

Convexity

- A convex object is an object with no “indentation”
- Mathematically, an object is convex if for any two points in the object, the line connecting the points is entirely inside the object

Planes

- 3 arbitrary points make a unique plane
- Each plane has a unique normal vector (perpendicular to the plane)
- Calculate the normal vector from three points?
- Standard equation: \( ax + by + cz = d \)
- Parametric equation of plane:
  - \( \mathbf{Z}(\alpha, \beta) = \mathbf{P} + \alpha (\mathbf{Q} - \mathbf{P}) + \beta (\mathbf{R} - \mathbf{P}) \)
**Coordinate System**

- Any coordinate frame can be represented as 4x4 matrix (in homogeneous coordinate)
- A point measured by two coordinate frames will yield two different coordinates
  - One coordinate can be converted to the other by using matrix operation (matrix vector multiplication, inverse, ...)
- Homogeneous Coordinate

**Homogeneous Coordinate**

- Point 2D Cartesian
  \((x, y)\)
- Vector 2D
  \[
  \begin{bmatrix}
  p \\
  q 
  \end{bmatrix}
  \]
- Point in 2DH (Homogeneous)
  \((x, y, 1)\)
- Direction 2DH
  \((p, q, 0)\)

**Homogeneous Coordinate**

- In general a 2D point \((x, y)\) in Cartesian Coordinate is expressed as \((sx, sy, s)\) in 2DH homogeneous coordinate
  - The scaling factor \(s\) is non-zero
- For 3D \((x, y, z) \Rightarrow (sx, sy, sz, s)\)

**Advantage of Homogeneous Coordinate**

- Easy to express a point at infinity (“direction”) using homogeneous coordinate
  - Example \((2, 1, 0)\) is a point at “positive” infinity along the infinite line connecting \((0,0)\) and \((2,1)\)
  - \((-2, -1, 0)\) is a point at “negative” infinity, the opposite side of \((2,1,0)\)

**Matrix**

- In Computer Graphics matrices are used for transformation of points, objects, ...
- In general matrices can be any size (rows and columns)
  - In 3D Computer Graphics: 4x4 matrices
- A vector is a “special case” of matrix whose column size is 1

**Matrix Operations**

- Multiplication
  - Matrix-Vector Multiplication
  - Matrix-Matrix Multiplication
  - Matrix-Scalar Multiplication
- Matrix-Matrix Addition/Subtraction
- Inverse of Matrix
- Identity Matrix
Coordinate Frames in OpenGL

- Object / Model Coordinate Frame (CF)
- World CF
- Eye / Viewer CF
- Clip CF
- Normalized Device CF
- Screen CF

Object-World-Viewer

- Objects are designed in the object CF
- Objects are then placed in the world CF
  - A number of coordinate transformations can be applied to the object coordinate to place it at a desired position and orientation
- Viewer / Camera is also placed in the world
  - A number of coordinate transformations can be applied to the camera CF to place the camera to a desired position and orientation

Inside the Virtual Camera

- After change of coordinate (from object to camera) the following take place inside the virtual camera
  - Clipping (the clipping volume is transformed in to a cube centered at the origin)
  - Perspective normalization (more detail in Chapter 5)
  - Projection to Image Coordinate / Screen Coordinate

3D Transformation Matrices

- Rigid-Body Transformation
  - Rotation (around Origin)
  - Translation
- Non Rigid-Body Transformation
  - Scaling (from Origin)

Use of Transformations

- Modeling Transformation
  - To place the model at a desired orientation / position
- Viewing Transformation
  - To place the camera / viewer at a desired orientation / position
- In OpenGL both transformation are represented by ONE model view matrix

Dual Interpretation of ModelView Matrix Update

- Changing the content of ModelView matrix can be interpreted as either:
  - Moving the object to a different position / orientation
  - Moving the camera to the opposite position / orientation
- Both interpretations must be understood equally well
  - In computer graphics, both interpretations are used
Composite Transformations

- Some operations cannot be expressed as a single linear transformation
  - Door opening / closing
  - Ball rolling
  - Ceiling fan spinning
  - Car turning on a curve
    - car body translates and rotates
    - tires rotate, spin, and translate
    - steering wheel rotates

Concatenation of Transformations

- Transformation matrices can be concatenated (multiplied) in some order to create a composite (complex) transformation
- Understanding the right order is important
- OpenGL postmultiplies each transformation matrix to the current transformation matrix
  - The product then replaces the current

Two Ways of Interpreting Composite Transformations

- Grand Fixed Coordinate Frame
  - All transformations refer to the fixed coordinate frame
  - Assume transformations are taking place in bottom to top order of the source code
- Local Moving Coordinate Frame
  - Each transformation refers to an instantaneous coordinate (locally moving) frame
  - Assume transformations are taking place in top to bottom order of the source code

Hints for Memorization

- Fall Back, Spring Forward
- Go Back vs. Leap Forward
  - Global CF reads Backward
  - Local CF reads Forward

Two ways of Interpreting Composite Transformation

- Regardless of which interpretation you use the visual effect (order of operation by GL engine) remains the same
- Local moving coordinate approach
  - The instantaneous coordinate frame is used for the reference of the next transformation

Interpreting Transformation Sequence: Example

```c
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();  /* M = I */
glRotate(30, 0, 0, 1);  /* M = M Rot */
glTranslate(2, 0, 0);  /* M = M Tran */
glVertex3f (....);  /* vertex P */
```

a) Vertex P is first translated 2 units along grand x-axis and then rotated 30 degrees around the grand z-axis
b) Vertex P is first rotate 30 degrees around a local z axis and then translated 2 units along the x-axis (of the new 30-degree rotated coordinate frame)
Matrix Manipulation in OpenGL

- `glLoadIdentity()`
  - initialize the current matrix to identity
- `glLoadMatrix()`
  - initialize (“upload”) the current matrix from a 16-element array (float or double)
  - Column major order
  - Use `GLfloat[16] mat;` not `GLfloat mat[4][4];`
- `glGetFloat()`, `glGetDoublev()`
  - retrieve (“download”) the current matrix to a 16-element array

OpenGL Matrix Multiplication

- `glMultMatrix (M)`
  - multiply the current matrix (C) with the given matrix (16 elements), save the product back to C
  - Post multiplication: `C = C * M`

Order of Statements

```
gLoadIdentity();
statements to move the object(s) from the origin to the desired pose
statement to draw the object(s) around the origin
```
Point A will be drawn at (100, 40) and line AB points in the NE direction

Matrix Stacks

- ModelView Matrix Stack (up to 32 matrices)
- Projection Matrix Stack (up to 2 matrices)
- OpenGL commands
  - `glPushMatrix()`: push the current matrix to its stack
  - `glPopMatrix()`: pop the topmost matrix from a stack and use it the current matrix
- Practical use: selective transformation

Selective Transformation

```
gLoadIdentity();
gTranslatef(100, 40, 0);
gPushMatrix();
gRotatef(45.0, 0, 0, 1);
BEGIN(GL_LINES);
  glVertex2f(0, 0); /* point A */
  glVertex2f(0, 3); /* point B */
END();
gPopMatrix();
gBegin(GL_LINES);
  glVertex2f(10, 10); /* point C */
  glVertex2f(10, 15); /* point D */
END();
```
// Rotation affects only line AB
Nested Push-Pop

glLoadIdentity();
going {100, 40, 0};
glPushMatrix();
  glRotatef(45, 0, 0, 1);
  glPushMatrix();
  glRotatef(20, 0, 0, 1);
  drawObjectA();
  glPopMatrix();
  drawObjectB();
  glPushMatrix();
  drawObjectC();
  // ObjectA: translated and rotated 65 degrees
  // ObjectB: translated and rotated 45 degrees
  // ObjectC: translated only