6.837 COMPUTER GRAPHICS: GEOMETRIC ALGEBRA

Jovan Popović
Computer graphics simulates worlds that look real, act real, sound real, and feel real.
Computer graphics simulates worlds that look real, act real, sound real, and feel real.

Develop a computational model of pinhole camera
   Modeling and rendering

Identify primitive elements, rules for their combination, and common patterns of use
   Geometric points, vectors, and coordinate systems

Improve computational model of pinhole camera
   Matrices and vectors
PRIMITIVE ELEMENTS, MEANS OF COMBINATION, AND COMMON PATTERNS

Geometric points, vectors, and coordinate systems
Graphics programming becomes simpler with improved representation of composition

```plaintext
map(image_to_raster(size),
    map(perspective_projection(1.),
        map(move_shape((0., 0., -10.)), cube())))
```

This functional composition requires general-purpose computation and general-purpose programming.
Points and vectors are fundamental geometric primitives.

Points describe positions in space.

Vectors describe directions and magnitudes, but not positions.
Basic geometry defines the ground rules for computation with points and vectors.

Vector + Vector

Point + Vector

Computer graphics performs same operations using algebraic manipulation of coordinates.
A coordinate system (frame) contains the origin (a point) and the coordinate axes (vectors)
A coordinate frame assigns a unique set of coordinates to all points and vectors.

\[ p = p_1 e_1 + p_2 e_2 + e_3 = (p_1, p_2, 1) \]

\[ v = v_1 e_1 + v_2 e_2 = (v_1, v_2, 0) \]

\[ e_1 = (1,0,0) \]
\[ e_2 = (0,1,0) \]
\[ e_3 = (0,0,1) \]
Matrix-vector multiplication defines a convenient shorthand

\[ p = \begin{bmatrix} e_1 & e_2 & e_3 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ 1 \end{bmatrix} \]

\[ v = \begin{bmatrix} e_1 & e_2 & e_3 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ 0 \end{bmatrix} \]

The axes and the origin form the basis for the space of all points and vectors.
Different choices of coordinate frame assigns different coordinates to same point or vector.

\[(4,2,0)_E = (4,2,0)_A\]

\[(2,3,1)_E = (-3,2,1)_A\]
Linear transformations are defined by their action on the coordinate frame.

\[ l(x) = l \left( \sum_{i=1}^{n} x_i e_i \right) = \sum_{i=1}^{n} x_i l(e_i) = \sum_{i=1}^{n} x_i l_i \]

\[ l(x) = \begin{bmatrix} l_1 & l_2 & l_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = Lx \]
Scaling is a linear transformation

\[ s(x) = \begin{bmatrix} s_1 & 0 & 0 \\ 0 & s_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} = Sx \]
Rotation is a linear transformation

\[ r_\theta(x) = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} = Rx \]
In carefully chosen vector space, translation is a linear transformation.

\[ t_d(x) = \begin{bmatrix} 1 & 0 & d_1 \\ 0 & 1 & d_2 \\ 0 & 0 & 1 \end{bmatrix} = Tx \]
In carefully chosen vector space, projection is a linear transformation.

Define the equivalence class for any point:

\[ p = (p_1, p_2, 1) = (wp_1, wp_2, w) \]
In carefully chosen vector space, projection is a linear transformation

A simple* projection function:

\[ p_1 \mapsto \frac{d}{-p_3}, \quad p_2 \mapsto \frac{d}{-p_3}, \quad p_3 \mapsto * - d \]

\[
\begin{bmatrix}
  d & 0 & 0 & 0 \\
  0 & d & 0 & 0 \\
  0 & 0 & d & 0 \\
  0 & 0 & -1 & 0 \\
\end{bmatrix}
\begin{bmatrix}
  p_1 \\
  p_2 \\
  p_3 \\
  w \\
\end{bmatrix}
= \begin{bmatrix}
  d \\
  -p_3 \\
  d \\
  -p_3 \\
\end{bmatrix}
\equiv \begin{bmatrix}
  d p_1 \\
  -p_3 \\
  d p_2 \\
  -p_3 \\
  -d \\
  1 \\
\end{bmatrix}
\]

*This simple projection does not preserve depth distances. See Buss II.3.2 for more detailed analysis.
Composition of linear transformation defines the rule for matrix multiplication

\[ t(r(x)) = t \left( \sum_{i=1}^{n} x_i r_i \right) = \sum_{i=1}^{n} x_i t(r_i) \]

\[ = \sum_{i=1}^{n} x_i t_i \]

\[ TR = \begin{bmatrix} Tr_1 & \ldots & Tr_n \end{bmatrix} \]
Matrices and vectors
Model space
(Object space)

scale, translate,
rotate, ...

World space
(Object space)

rotate, translate

Eye space
(View space)
PINHOLE CAMERA WITH OpenGL

Eye space
(View space)

Projective transformation,
scale, translate

Normalized projection space

Project,
scale, translate

Normalized device space
(Screen space)

scale

Image space
(Window space)
(Raster space)
(Screen space)
(Device space)
Graphics programming becomes simpler with improved representation of composition

```python
map(image_to_raster(size),
    map(perspective_projection(1.),
        map(move_shape((0., 0., -10.)), cube())))

glMatrixMode(GL_PROJECTION)
glLoadIdentity()
gluPerspective(...)
glMatrixMode(GL_MODELVIEW)
glLoadIdentity()
glTranslate(0., 0., -10.)
cube()
```