Fundamentals of 3D Graphics

ECE 6397 (GPU Programming) – David Mayerich
Principals in Graphics

• **rasterization**: conversion of vector graphics into a raster image
• **ray-tracing**: tracing the path of light contributing to each pixel
• Most real-time graphics applications rely on *rasterization* of triangles
  – Defined as 3 points given a specific winding (CW or CCW)
Rasterizing Lines

• Bresenham’s Line Drawing Algorithm
  – determine the points of an image that can be filled to approximate a line segment given by \((x_0, y_0)\) and \((x_1, y_1)\)
  – assumes that the line goes down and to the right with slope \(|m| \leq 1\)

• Given the equation for a line:
  \[
  \frac{y - y_0}{y_1 - y_0} = \frac{x - x_0}{x_1 - x_0}
  \]

• Start at \((x_0, y_0)\) and take integer steps along \(x\):
  \[
  y = \left[ \frac{y_1 - y_0}{x_1 - x_0} (x - x_0) + y_0 + \frac{1}{2} \right]
  \]

• If \(|m| > 1\), flip the coordinates
Rasterizing Polygons

• Scanline Rasterization
Modern Approaches: Anti-Aliasing

• Wu’s Line Algorithm
Interpolation and Barycentric Coordinates

• Barycentric coordinates define a unique point as a weighted value of vertices within a simplex

• Given a triangle composed of points \((p_0, p_1, p_2)\):

\[
p = \alpha_0 p_0 + \alpha_1 p_1 + \alpha_2 p_2
\]

\[
\alpha_0 + \alpha_1 + \alpha_2 = 1
\]

• The barycentric coordinates \((\alpha_0, \alpha_1, \alpha_2)\) can be used to interpolate any parameter specified at the vertices
Linear Transformations

- Objects composed of triangles are manipulated by applying a linear transformation to each vertex:

\[
\begin{align*}
\begin{bmatrix}
sp_0 \\
sp_1 \\
sp_2
\end{bmatrix} &= \begin{bmatrix}
s_x p_x \\
sp_y \\
s_z p_z
\end{bmatrix} \\
&= (s_x p_x, s_y p_y, s_z p_z)
\end{align*}
\]

- **translation**: add \( t_x, t_y, t_z \) to each component
- **isotropic scaling**: multiply each coordinate \( s \)
- **anisotropic scaling**: multiply each coordinate by a corresponding \( s_x, s_y, s_z \)
- **rotation**: linear combination of components based on trigonometric functions of \( \theta \)
Transformations as Linear Systems

• A point is represented as a vector in $p \in \mathbb{R}^3$
• A transformation is represented by a matrix:

$$\begin{bmatrix}
    s & 0 & 0 \\
    0 & s & 0 \\
    0 & 0 & s
\end{bmatrix} \begin{bmatrix}
    p_x \\
    p_y \\
    p_z
\end{bmatrix}$$

isotropic scaling

$$\begin{bmatrix}
    s_x & 0 & 0 \\
    0 & s_y & 0 \\
    0 & 0 & s_z
\end{bmatrix} \begin{bmatrix}
    p_x \\
    p_y \\
    p_z
\end{bmatrix}$$

anisotropic scaling

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

rotation about $z$

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

rotation about $y$

• 3D matrix operations can handle scaling and rotation, but not translation...
Affine Transformations and Augmented Matrices

• A larger group of transformations – affine transformations – can be represented as matrices

• affine transformation: linear transformation of the form \( y = Mx + b \)
  – represents a linear transformation and a translation
  – preserves parallel lines and ratios of distances along a line
  – all linear transformation is affine, but not all affine transforms are linear

• Augmented matrices can be used to represent a translation and linear transformation:

\[
p' = Mp + b
\]

\[
\begin{bmatrix}
p'_x \\
p'_y \\
p'_z \\
1
\end{bmatrix} =
\begin{bmatrix}
m_{11} & m_{12} & m_{13} & b_x \\
m_{21} & m_{22} & m_{23} & b_y \\
m_{31} & m_{32} & m_{33} & b_z \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
p_x \\
p_y \\
p_z \\
1
\end{bmatrix}
\]
Projection

- Assume that all coordinates for a set of triangles are specified
- These coordinates are transformed and displayed on a 2D screen
Projection Matrices

• Orthographic projection

\[ P_{ortho} = \begin{bmatrix}
\frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\
0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\
0 & 0 & \frac{2}{f-n} & -\frac{f+n}{f-n} \\
0 & 0 & 0 & 1 \\
\end{bmatrix} \]

• Perspective projection

\[ P_{per} = \begin{bmatrix}
\frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\
0 & \frac{2n}{t-b} & \frac{r-l}{t-b} & 0 \\
0 & 0 & \frac{f+n}{f-n} & -\frac{2fn}{f-n} \\
0 & 0 & -1 & 0 \\
\end{bmatrix} \]
Compound Transformations

• Sequential linear transformations can be applied through multiple matrix multiplications:

1. Rotate the vertices by $\theta_1$
2. Translate by $t_1$
3. Scale by $s_1$
4. Apply the camera rotation $\theta_c$
5. Apply the camera translation $t_c$
6. Apply a perspective projection

$$R_{\theta_1} \cdot T_{t_1} \cdot S_{s_1} \cdot R_{\theta_c} \cdot T_{t_c} \cdot P \cdot p = p'$$
Vertex Pipeline

- Vertices are submitted for rendering hierarchically:

\[ M_{11} \cdot M_{12} \cdot M_{13} \cdot M_{14} \cdot M_{15} \cdot M_{16} \cdot M_{17} \cdot P = R_{55} S_{55} T_{55} M_{17} \]

- calculate \( P \) and push( )
- calculate \( M_7 \) and push( )
- draw7( )
- pop( )
- calculate \( M_8 \) and push( )
- draw8( )
- pop( )
- draw7( ):
  - calculate \( M_4 \) and push( )
  - draw4( )
  - calculate \( M_5 \) and push( )
  - ...

\[ M_{5} M_{7} P v_n \]