Advanced Computer Graphics

Transformations

Matthias Teschner

Computer Science Department
University of Freiburg
Motivation

- transformations are used
  - to convert between arbitrary spaces, e.g. world space and other spaces, such as object space, camera space
  - to position and animate objects, lights, and the virtual camera
- transformations are applied to points, normals, rays
Outline

- coordinate spaces
- homogeneous coordinates
- transformations
- transformations in ray tracing
- animating transformations
Coordinate Spaces

- **object space**
  - coordinate system in which geometric primitives are defined
  - object spaces are object-dependent

- **world space**
  - objects, lights are placed / transformed into world space
  - object-to-world transformations allow to arbitrarily place objects, lights relative to each other

- **camera space**
  - a space with a specific camera setting, e.g. camera at the origin, viewing along z-axis, y-axis is up direction
  - useful for simplified computations (similar to the rendering pipeline)
  - camera is placed in world space with a view transformation
  - inverse view transform is used to get from world to camera space
From Object to Camera Space

- $\mathbf{M}_1$, $\mathbf{M}_2$, $\mathbf{M}_3$, $\mathbf{M}_4$, $\mathbf{V}$ are matrices representing transformations
- $\mathbf{M}_1$, $\mathbf{M}_2$, $\mathbf{M}_3$, $\mathbf{M}_4$ are object-to-world transforms to place the objects in the scene
- $\mathbf{V}$ places and orientates the camera in space
  - $\mathbf{V}^{-1}$ transforms the camera to the origin looking along the z-axis
- $\mathbf{V}^{-1}\mathbf{M}_{1..4}$ transforms all objects or lights from object to camera space
Coordinate Spaces

- normalized device coordinate space (NDC space)
  - x-, y-, z-coordinates range from zero (e.g. left, bottom, near) to one (e.g. right, top, far)
  - camera-to-NDC transformation with, e.g., perspective projection, parallel projection

- raster space
  - x-, y-coordinates range from (0,0) to (xResolution-1, yResolution-1)
  - NDC-to-raster transformation with scaling and translation
Outline

- coordinate spaces
- homogeneous coordinates
- transformations
- transformations in ray tracing
- animating transformations
Motivation

- using homogeneous coordinates,
  - affine transformations can be represented with a matrix
  - points, vectors, ray can be transformed in a unified way with one matrix-vector multiplication
Affine Transformations

- affine transformations of a 3D point $\mathbf{p}$: $\mathbf{p}' = \mathbf{T}(\mathbf{p}) = \mathbf{A}\mathbf{p} + \mathbf{t}$
- affine transformations preserve affine combinations
  $\mathbf{T}(\sum_i \alpha_i \cdot \mathbf{p}_i) = \sum_i \alpha_i \cdot \mathbf{T}(\mathbf{p}_i)$ for $\sum_i \alpha_i = 1$
- e.g., a line can be transformed by transforming its control points

\[ x = \alpha_1 \mathbf{p}_1 + \alpha_2 \mathbf{p}_2 \]

\[ x' = \mathbf{T}(x) = \alpha_1 \mathbf{T}(\mathbf{p}_1) + \alpha_2 \mathbf{T}(\mathbf{p}_2) \]
Points, Vectors, Rays

- points specify a location (x, y, z) in space
e.g., vertices of a triangulated object

- vectors specify a direction (x, y, z)
  - e.g., surface normals
    (the not necessarily normalized direction perpendicular to a surface)

- rays
  - a half-line specified by an origin / position \( \mathbf{o} \) and a direction \( \mathbf{d} \)
  - parametric form \( \mathbf{r}(t) = \mathbf{o} + t\mathbf{d} \) with \( 0 \leq t \leq \infty \)
  - various additional properties in ray tracers, e.g.
    - parametric range, time, recursion depth
Homogeneous Coordinates of Points

- \((x, y, z, w)^T\) with \(w \neq 0\) are the homogeneous coordinates of the 3D point \((\frac{x}{w}, \frac{y}{w}, \frac{z}{w})^T\)

- \((\lambda x, \lambda y, \lambda z, \lambda w)^T\) represents the same point \((\frac{\lambda x}{\lambda w}, \frac{\lambda y}{\lambda w}, \frac{\lambda z}{\lambda w})^T = (\frac{x}{w}, \frac{y}{w}, \frac{z}{w})^T\) for all \(\lambda\) with \(\lambda \neq 0\)

- examples
  - \((2, 3, 4, 1) \sim (2, 3, 4)\)
  - \((2, 4, 6, 1) \sim (2, 4, 6)\)
  - \((4, 8, 12, 2) \sim (2, 4, 6)\)
Homogeneous Coordinates of Vectors

- for varying $w$, a point $(x, y, z, w)^T$ is scaled and the points $(\frac{x}{w}, \frac{y}{w}, \frac{z}{w})^T$ represent a line in 3D space.

- the direction of this line is characterized by $(x, y, z)^T$

- for $w \to 0$, the point $(\frac{x}{w}, \frac{y}{w}, \frac{z}{w})^T$ moves to infinity in the direction $(x, y, z)^T$

- $(x, y, z, 0)^T$ is a point at infinity in the direction of $(x, y, z)^T$

- $(x, y, z, 0)^T$ is a vector in the direction of $(x, y, z)^T$
Homogeneous Representation of Transformations

- linear transformation
  \[
  \begin{pmatrix}
  m_{00} & m_{01} & m_{02} \\
  m_{10} & m_{11} & m_{12} \\
  m_{20} & m_{21} & m_{22}
  \end{pmatrix}
  \begin{pmatrix}
  p_x \\
  p_y \\
  p_z
  \end{pmatrix}
  \sim
  \begin{pmatrix}
  m_{00} & m_{01} & m_{02} & 0 \\
  m_{10} & m_{11} & m_{12} & 0 \\
  m_{20} & m_{21} & m_{22} & 0 \\
  0 & 0 & 0 & 1
  \end{pmatrix}
  \begin{pmatrix}
  p_x \\
  p_y \\
  p_z \\
  1
  \end{pmatrix}
  \]

- affine transformation
- representing rotation, scale, shear, translation
- projective components \( p \) are zero for affine transformations
Outline

- coordinate spaces
- homogeneous coordinates
- transformations
- transformations in ray tracing
- animating transformations
Translation

- of a point
  \[ T(t)p = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix} = \begin{pmatrix} p_x + t_x \\ p_y + t_y \\ p_z + t_z \\ 1 \end{pmatrix} \]

- of a vector
  \[ T(t)v = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} v_x \\ v_y \\ v_z \\ 0 \end{pmatrix} = \begin{pmatrix} v_x \\ v_y \\ v_z \\ 0 \end{pmatrix} \]

- inverse (\( T^{-1} \) "undoes" the transform \( T \))
  \[ T^{-1}(t) = T(-t) \]
Rotation

- positive (anticlockwise) rotation with angle $\phi$
  around the $x$-, $y$-, $z$-axis

\[
\mathbf{R}_x(\phi) = \begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & \cos \phi & -\sin \phi & 0 \\
0 & \sin \phi & \cos \phi & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}
\]

\[
\mathbf{R}_y(\phi) = \begin{pmatrix}
\cos \phi & 0 & \sin \phi & 0 \\
0 & 1 & 0 & 0 \\
-\sin \phi & 0 & \cos \phi & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}
\]

\[
\mathbf{R}_z(\phi) = \begin{pmatrix}
\cos \phi & -\sin \phi & 0 & 0 \\
\sin \phi & \cos \phi & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}
\]
Inverse Rotation

- \( \mathbf{R}_x(-\phi) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \phi & -\sin \phi & 0 \\ 0 & \sin \phi & \cos \phi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \mathbf{R}_x^T(\phi) \)

- the inverse of a rotation matrix corresponds to its transpose
Compositing Transformations

- composition is achieved by matrix multiplication
  \[ M_2 (M_1 p) = (M_2 M_1) p \]
  \[ \text{note that generally } M_1 M_2 \neq M_2 M_1 \]
  \[ \text{the inverse is } (M_2 M_1)^{-1} = M_1^{-1} M_2^{-1} \]

- examples
  - rotation about an arbitrary axes
  - scaling with respect to arbitrary directions
  - object-to-view space transformation \( V^{-1} M \)
Outline

- coordinate spaces
- homogeneous coordinates
- transformations
- transformations in ray tracing
- animating transformations
Objects

- Transformations can be represented in a graph or hierarchy
- E.g., for a car
- T1 is applied to "chassis", T1 \cdot T2 \ldots T1 \cdot T5 are applied to the wheels
Instancing

- to save memory
Planes and Normals

- Planes can be represented by a surface normal $\mathbf{n}$ and a point $\mathbf{r}$. All points $\mathbf{p}$ with $\mathbf{n} \cdot (\mathbf{p} - \mathbf{r}) = 0$ form a plane.

  $$nxpx + nypy + nzpz + \left(-nxrx - nyry - nzrz\right) = 0$$

  $$nxpx + nypy + nzpz + d = 0$$

  $$(nx \ ny \ nz \ d)(px \ py \ pz \ 1)^T = 0$$

- The transformed points $A(px \ py \ pz \ 1)^T$ are on the plane represented by

  $$(nx \ ny \ nz \ d)A^{-1} = ((A^{-1})^T(nx \ ny \ nz \ d)^T)^T$$

- If a surface is transformed by $A$, its normal is transformed by $(A^{-1})^T$
Normals

- Normals generally point outside of a surface.
- If a transformation changes the handedness of the coordinate system, the normal $\mathbf{n}$ might need to be flipped to $-\mathbf{n}$.
- The handedness changes if the determinant is negative.
- E.g., for a reflection $\mathbf{R}$, $\det \mathbf{R} = -1$. 
Rays

- for ray-object intersections,
  - objects are commonly not transformed
  - instead, rays are transformed with the inverse of the object-to-camera space transformation

- algorithm
  - apply the inverse transform to the ray
  - compute intersection and normal
  - transform the intersection and the normal
Rays

\[ p' = o + td \]
\[ T^{-1} p' = p = T^{-1} o + tT^{-1} d \]
\[ p' = Tp = o + td \]

if computed in camera space, \( o \) is \((0,0,0,1)^T\)

[Suffern: Ray Tracing]
Outline

- coordinate spaces
- homogeneous coordinates
- transformations
- transformations in ray tracing
- animating transformations
Animating Transformations

- keyframe matrix animation
  - for camera and objects
  - defined by a number of keyframe transformations
  - allows camera / object movements, e.g. for motion blur

- challenge
  - linear combination of two corresponding matrix values does not provide useful results for general transformations
Animating Translation, Scale, and Shear

- Linear interpolation of matrices, representing translation, is meaningful:
  \[
  \mathbf{T}(\lambda) = (1 - \lambda) \begin{pmatrix} 1 & 0 & 0 & s_x \\ 0 & 1 & 0 & s_y \\ 0 & 0 & 1 & s_z \\ 0 & 0 & 0 & 1 \end{pmatrix} + \lambda \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}
  \]

- Interpolation of components also works for scale and shear:

\[
\mathbf{K} = \begin{pmatrix} x & 0 & 0 & 0 \\ 0 & y & 0 & 0 \\ 0 & 0 & z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad \mathbf{H} = \begin{pmatrix} 1 & h_{xy} & h_{xz} & 0 \\ h_{yx} & 1 & h_{yz} & 0 \\ h_{zx} & h_{zy} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}
\]
Animating Rotations

- linear combination of matrix components does not work
  - interpolated matrix is not orthogonal, i.e. object can be distorted
  - determinant of the interpolated matrix is not one, lengths are not preserved, object can be stretched, compressed or degenerate to a line or a point

[Shoemake, Duff]
Animating Rotations

- a useful approach
  - convert the rotation matrices to a quaternion representation
  - perform a spherical linear interpolation (slerp)
  - convert the interpolated quaternion to a rotation matrix

- motivation
  - rate of change of the rotation / orientation can be linear in the interpolation parameter when using quaternions
Quaternions

- are four-tuples $q = (w, x, y, z) = w + xi + yj + zk$
  with $i^2 = j^2 = k^2 = ijk = -1$ and $ij = k$, $ji = -k$, ...
- quaternion multiplication
  $qr = (q_w + q_xi + q_yj + q_zk)(r_w + r_xi + r_yj + r_zk)$
- unit quaternions represent rotations ($u$ is a unit vector)
  $q = w + xi + yj + zk = w + (x, y, z) = \cos(\frac{\alpha}{2}) + u \sin(\frac{\alpha}{2})$
- if $v$ is a 3D vector, the product $v' = q(0, v)q^{-1}$ results in a vector $v'$ rotated by $\alpha$ around $u$
- $q^n$ is a rotation by $n$ times the angle $\alpha$ around $u$
Spherical Linear Interpolation

- \text{slerp} \ s (q_1, q_2, t) \text{ computes intermediate orientations between } q_1 \text{ and } q_2
- Orientation changes are linear in } t

\[
\text{Slerp}(q_1, q_2, t) = \frac{\sin(1-t)\theta}{\sin \theta} q_1 + \frac{\sin t\theta}{\sin \theta} q_2
\]

- Still leads to discontinuous orientation changes in case of changing rotation axes between key frames

[Shoemake]
Matrix Decomposition

- if keyframe transformations are composed of translation, rotation, and scale, these components have to be decomposed and interpolated independently
- projective components are not considered, (but could be extracted easily)
- translation can be extracted as

\[
\begin{pmatrix}
    m_{11} & m_{12} & m_{13} & t_x \\
    m_{21} & m_{22} & m_{23} & t_y \\
    m_{31} & m_{32} & m_{33} & t_z \\
    0 & 0 & 0 & 1
\end{pmatrix}
= \begin{pmatrix}
    1 & 0 & 0 & t_x \\
    0 & 1 & 0 & t_y \\
    0 & 0 & 1 & t_z \\
    0 & 0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
    m_{11} & m_{12} & m_{13} & 0 \\
    m_{21} & m_{22} & m_{23} & 0 \\
    m_{31} & m_{32} & m_{33} & 0 \\
    0 & 0 & 0 & 1
\end{pmatrix}
\]
Rotation Extraction

- approaches
  - QR decomposition (extracted rotations are not meaningful)
  - SVD (extracted rotations are not meaningful, rather unstable)
  - Polar decomposition

- Polar decomposition
  - efficient to compute
  - extracts rotation $R$ that is closest to the original transf. $M$
  - Find $R$ minimizing $\|R - M\|_F^2$
    subject to $R^T R - I = 0$
    with $\|R - M\|_F^2 = \sum_{i,j} (r_{i,j} - m_{i,j})^2$
    being the Frobenius matrix norm

- $M = R(-I)S$
  - $S$ is symmetric, positive definite (scale in a potentially rotated frame)
  - shear cannot be extracted
Polar Decomposition

- iterative computation [Higham]
  - $R_0 = M$
  - $R_{i+1} = \frac{1}{2}(R_i + (R_i^T)^{-1})$
  - until $R_{i+1} - R_i \approx 0$
Summary

- coordinate spaces
  - object, world, camera

- homogeneous coordinates
  - points, vectors, rays can be transformed in a unified way
  - matrices for affine transformations and perspective projections

- transformations
  - translation, rotation, scale, shear, inverse, composition

- transformations in ray tracing
  - instancing, normals, rays

- animating transformations
  - Polar decomposition, quaternions, linear and spherical linear interpolation