Advanced Computer Graphics
Transformations

Matthias Teschner

UNI 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
  - Space 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 and lights relative to each other

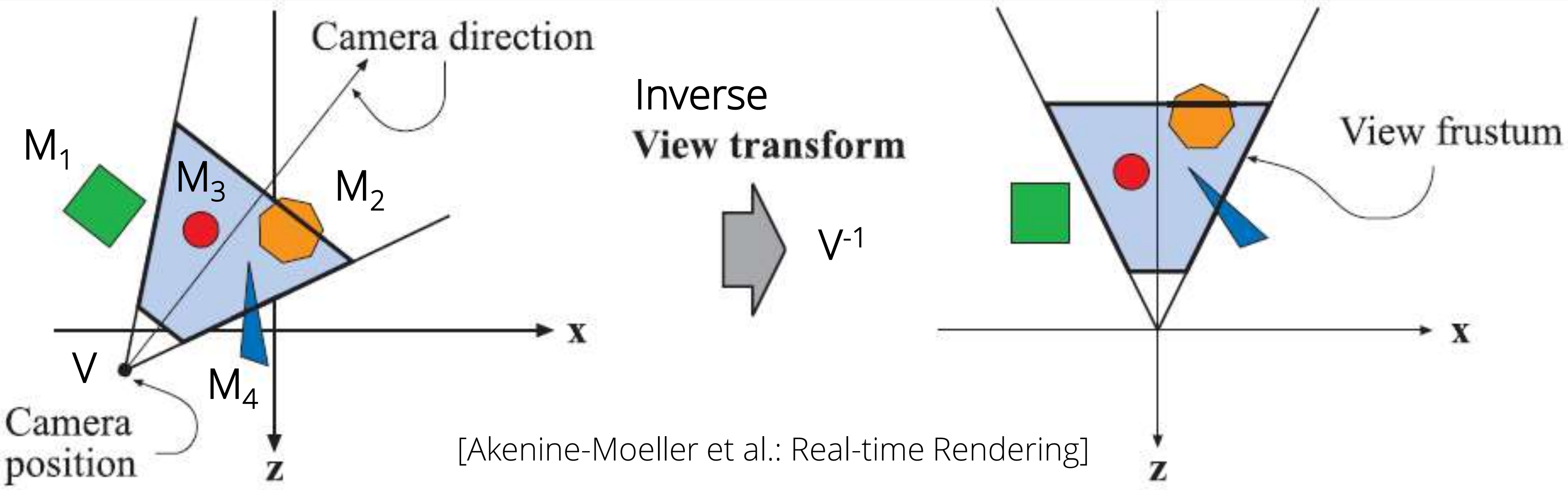
UNI FREIBURG

Coordinate Spaces

- Camera space
  - 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 converts from world to camera space

UNI FREIBURG

From Object to Camera Space



[Akenine-Moeller et al.: Real-time Rendering]

- M_1, M_2, M_3, M_4, V are transformation matrices
- M_1, M_2, M_3, M_4 are object-to-world transforms placing objects in the scene
- V places and orientates the camera in space
- V^{-1} transforms the camera to the origin looking along the z-axis
- $V^{-1}M_{1..4}$ transforms all objects or lights from object to camera space

Outline

- Coordinate spaces
- Homogeneous coordinates
- Transformations
- Transformations in ray tracing
- Animating transformations

Motivation

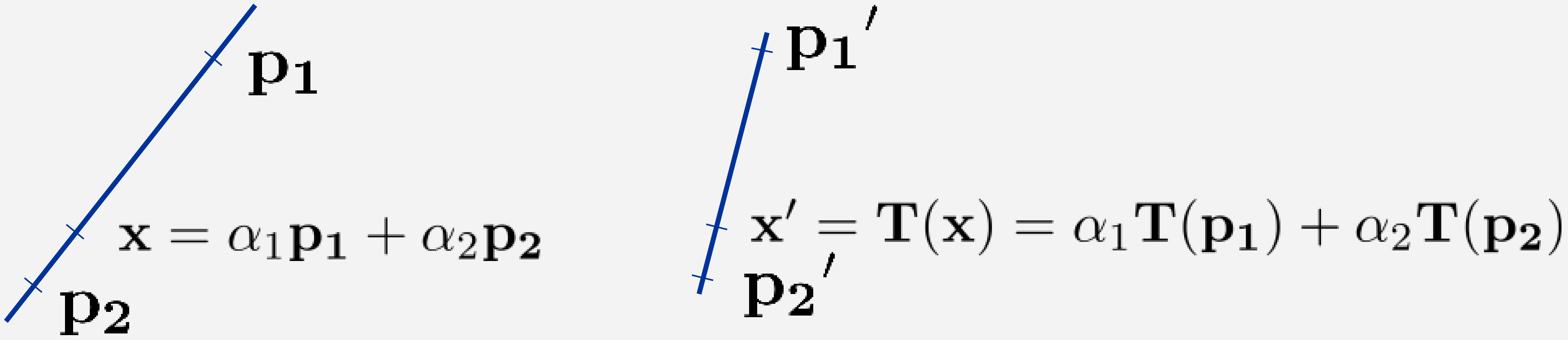
- Using homogeneous coordinates,
  - Affine transformations can be represented with a matrix
  - Points, vectors, rays can be transformed in a unified way with one matrix-vector multiplication

Affine Transformations

– Affine transformations of a 3D point p: $\mathbf{p}' = \mathbf{T}(\mathbf{p}) = \mathbf{A}\mathbf{p} + \mathbf{t}$

– Affine transformations preserve affine combinations
$\mathbf{T}\left(\sum_i \alpha_i \cdot \mathbf{p}_i\right) = \sum_i \alpha_i \cdot \mathbf{T}(\mathbf{p}_i)$ for $\sum_i \alpha_i = 1$

– E.g., a line can be transformed
by transforming its control points



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
- Rays
  - A half-line specified by origin / position **o** and direction **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 ≠ 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 λ with λ ≠ 0
- Examples
  - (2, 3, 4, 1) ~ (2, 3, 4)
  - (2, 4, 6, 1) ~ (2, 4, 6)
  - (4, 8, 12, 2) ~ (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 $w \rightarrow 0$, 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

$$\begin{pmatrix} m_{00} & m_{01} & m_{02} & t_0 \\ m_{10} & m_{11} & m_{12} & t_1 \\ m_{20} & m_{21} & m_{22} & t_2 \\ p_0 & p_1 & p_2 & w \end{pmatrix}$$

Outline

- Coordinate spaces
- Homogeneous coordinates
- Transformations
- Transformations in ray tracing
- Animating transformations

Translation

– Point

$$\mathbf{T}(\mathbf{t})\mathbf{p} = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix} = \begin{pmatrix} p_x + t_x \\ p_y + t_y \\ p_z + t_z \\ 1 \end{pmatrix}$$

– Vector

$$\mathbf{T}(\mathbf{t})\mathbf{v} = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} v_x \\ v_y \\ v_z \\ 0 \end{pmatrix} = \begin{pmatrix} v_x \\ v_y \\ v_z \\ 0 \end{pmatrix}$$

– Inverse (T^{-1} "undoes" the transform T)

$$\mathbf{T}^{-1}(\mathbf{t}) = \mathbf{T}(-\mathbf{t})$$

Rotation

– Positive (anticlockwise) rotation with angle ϕ 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

$$\mathbf{R_x}(-\phi) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos -\phi & -\sin -\phi & 0 \\ 0 & \sin -\phi & \cos -\phi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

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

$$\mathbf{R_x}^{-1} = \mathbf{R_x}^T \qquad \mathbf{R_y}^{-1} = \mathbf{R_y}^T \qquad \mathbf{R_z}^{-1} = \mathbf{R_z}^T$$

– The inverse of a rotation matrix corresponds to its transpose

Compositing Transformations

- Composition is achieved by matrix multiplication
  - $\mathbf{M_2}(\mathbf{M_1}\mathbf{p}) = (\mathbf{M_2}\mathbf{M_1})\mathbf{p}$
  - Note that generally $\mathbf{M_1}\mathbf{M_2} \neq \mathbf{M_2}\mathbf{M_1}$
  - The inverse is $(\mathbf{M_2}\mathbf{M_1})^{-1} = \mathbf{M_1}^{-1}\mathbf{M_2}^{-1}$
- Examples
  - Rotation about an arbitrary axes
  - Scaling with respect to arbitrary directions
  - Object-to-view space transformation $\mathbf{V}^{-1}\mathbf{M}$

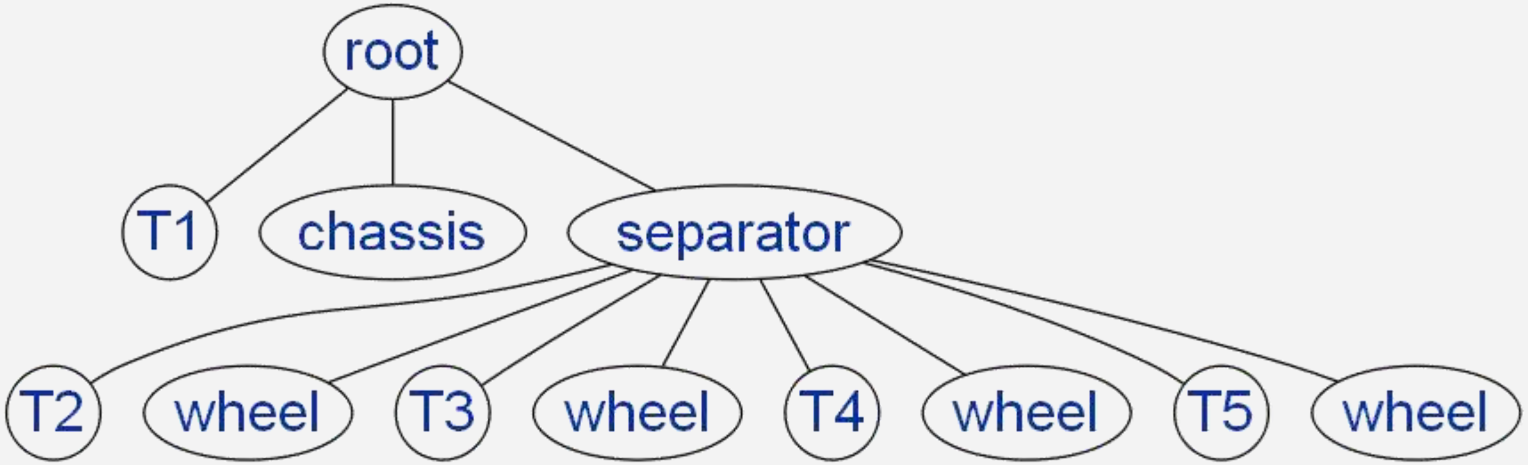
Outline

- Coordinate spaces
- Homogeneous coordinates
- Transformations
- Transformations in ray tracing
- Animating transformations

UNI FREIBURG

Objects

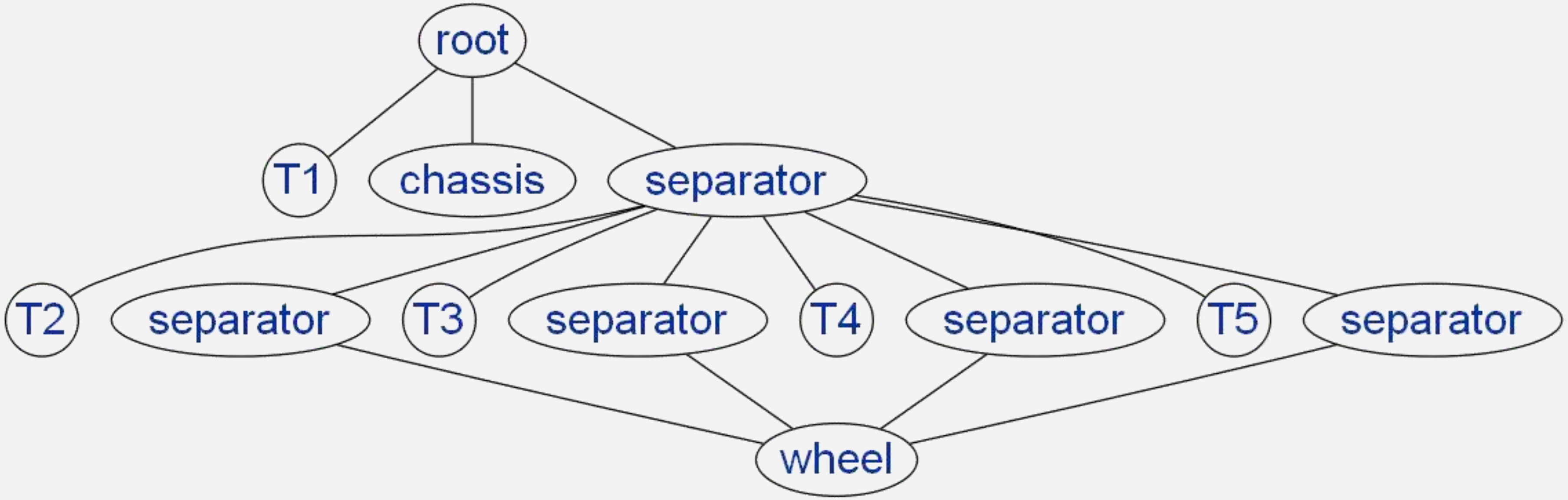
– Transformations can be represented in a graph or hierarchy, e.g., for a car



– T1 is applied to "chassis", T1 · T2 ... T1 · T5 are applied to the wheels

Instancing

– To save memory



Planes and Normals

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

$$n_x p_x + n_y p_y + n_z p_z + (-n_x r_x - n_y r_y - n_z r_z) = 0$$

$$n_x p_x + n_y p_y + n_z p_z + d = 0$$

$$(n_x \; n_y \; n_z \; d)(p_x \; p_y \; p_z \; 1)^T = 0$$

$$(n_x \; n_y \; n_z \; d)\mathbf{A}^{-1}\mathbf{A}(p_x \; p_y \; p_z \; 1)^T = 0$$

– The transformed points $\mathbf{A}(p_x \; p_y \; p_z \; 1)^T$ are on the plane represented by $(n_x \; n_y \; n_z \; d)\mathbf{A}^{-1} = ((\mathbf{A}^{-1})^T (n_x \; n_y \; n_z \; d)^T)^T$

– 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 **n** might need to be flipped to -**n**
- The handedness changes if the determinant is negative
- E.g., for a reflection **R**, $\det \mathbf{R} = -1$

Rays

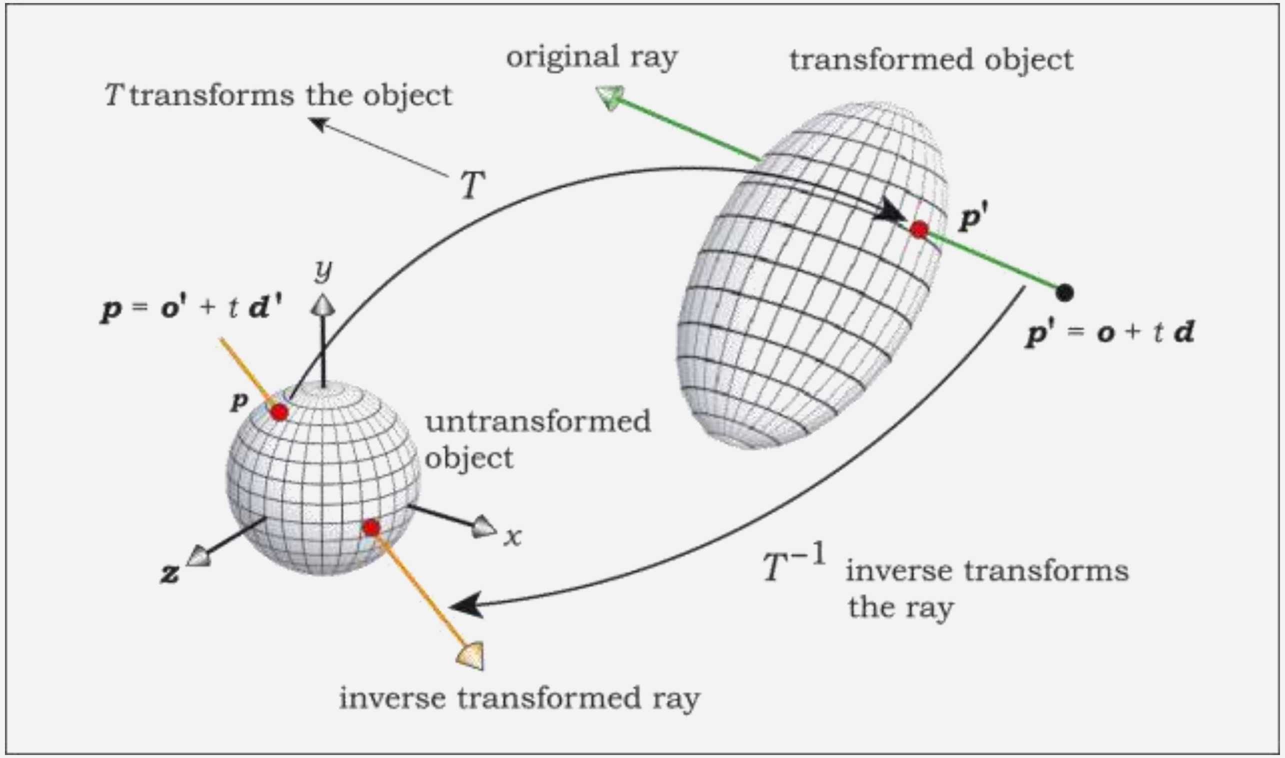
- For ray-object intersections,
  - Objects are commonly not transformed
  - Instead, rays are transformed with the inverse of the object-to-camera space transformation
- Algorithm
  - Apply the inverse transform to the ray
  - Compute intersection and normal
  - Transform the intersection and the normal

Rays

$$\mathbf{p}' = \mathbf{o} + t\mathbf{d}$$

$$\mathbf{T}^{-1}\mathbf{p}' = \mathbf{p} = \mathbf{T}^{-1}\mathbf{o} + t\mathbf{T}^{-1}\mathbf{d}$$

$$\mathbf{p}' = \mathbf{T}\mathbf{p} = \mathbf{o} + t\mathbf{d}$$



If computed in camera space, **o** is $(0,0,0,1)^T$

[Suffern: Ray Tracing]

UNI FREIBURG

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

$$\mathbf{T}(\lambda) = (1-\lambda)\begin{pmatrix} 1 & 0 & 0 & s_x \\ 0 & 1 & 0 & s_y \\ 0 & 0 & 1 & s_z \\ 0 & 0 & 0 & 1 \end{pmatrix} + \lambda \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

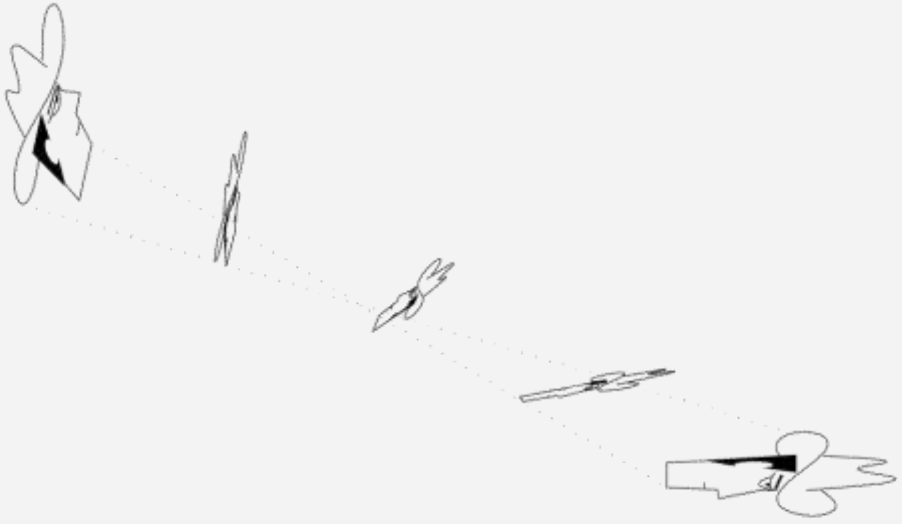
T(λ) is a translation

– Interpolation of components also works for scale and shear

$$\mathbf{K} = \begin{pmatrix} x & 0 & 0 & 0 \\ 0 & y & 0 & 0 \\ 0 & 0 & z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad \mathbf{H} = \begin{pmatrix} 1 & h_{xy} & h_{xz} & 0 \\ h_{yx} & 1 & h_{yz} & 0 \\ h_{zx} & h_{zy} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Animating Rotations

- Linear combination of matrix values 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 $\mathbf{q} = (w, x, y, z) = w + x\mathbf{i} + y\mathbf{j} + z\mathbf{k}$ with $\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1$ and $\mathbf{ij} = \mathbf{k},\ \mathbf{ji} = -\mathbf{k},\ \ldots$
- Quaternion multiplication

$$\mathbf{qr} = (q_w + q_x\mathbf{i} + q_y\mathbf{j} + q_z\mathbf{k})(r_w + r_x\mathbf{i} + r_y\mathbf{j} + r_z\mathbf{k})$$

- Unit quaternions represent rotations (**u** is a unit vector)

$$\mathbf{q} = w + x\mathbf{i} + y\mathbf{j} + z\mathbf{k}) = w + (x, y, z) = \cos(\tfrac{\alpha}{2}) + \mathbf{u}\sin(\tfrac{\alpha}{2})$$

- If **v** is a 3D vector, the product $\mathbf{v}' = \mathbf{q}(0, \mathbf{v})\mathbf{q}^{-1}$ results in a vector **v'** rotated by α around **u**
- $\mathbf{q}^n$ is a rotation by *n* times the angle α around **u**

Spherical Linear Interpolation

- Slerp($\mathbf{q}_1$, $\mathbf{q}_2$, t) computes intermediate orientations between $\mathbf{q}_1$ and $\mathbf{q}_2$
- Orientation changes are linear in t

$$\mathrm{Slerp}(\mathbf{q_1}, \mathbf{q_2}, t) = \frac{\sin(1-t)\theta}{\sin\theta}\mathbf{q_1} + \frac{\sin t\theta}{\sin\theta}\mathbf{q_2}$$

[Shoemake]

$$\mathbf{q_1} \cdot \mathbf{q_2} = \cos\theta$$

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

UNI FREIBURG

Matrix Decomposition

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

$$\mathbf{M} = \begin{pmatrix} m_{11} & m_{12} & m_{13} & t_x \\ m_{21} & m_{22} & m_{23} & t_y \\ m_{31} & m_{32} & m_{33} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} m_{11} & m_{12} & m_{13} & 0 \\ m_{21} & m_{22} & m_{23} & 0 \\ m_{31} & m_{32} & m_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotation Extraction

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

-I in case of a negative determinant
S is symmetric, positive definite (scale in a potentially rotated frame)
shear cannot be extracted

UNI FREIBURG

Polar Decomposition

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

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