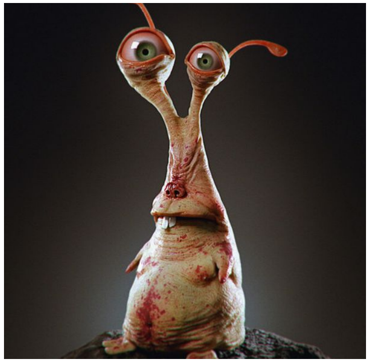
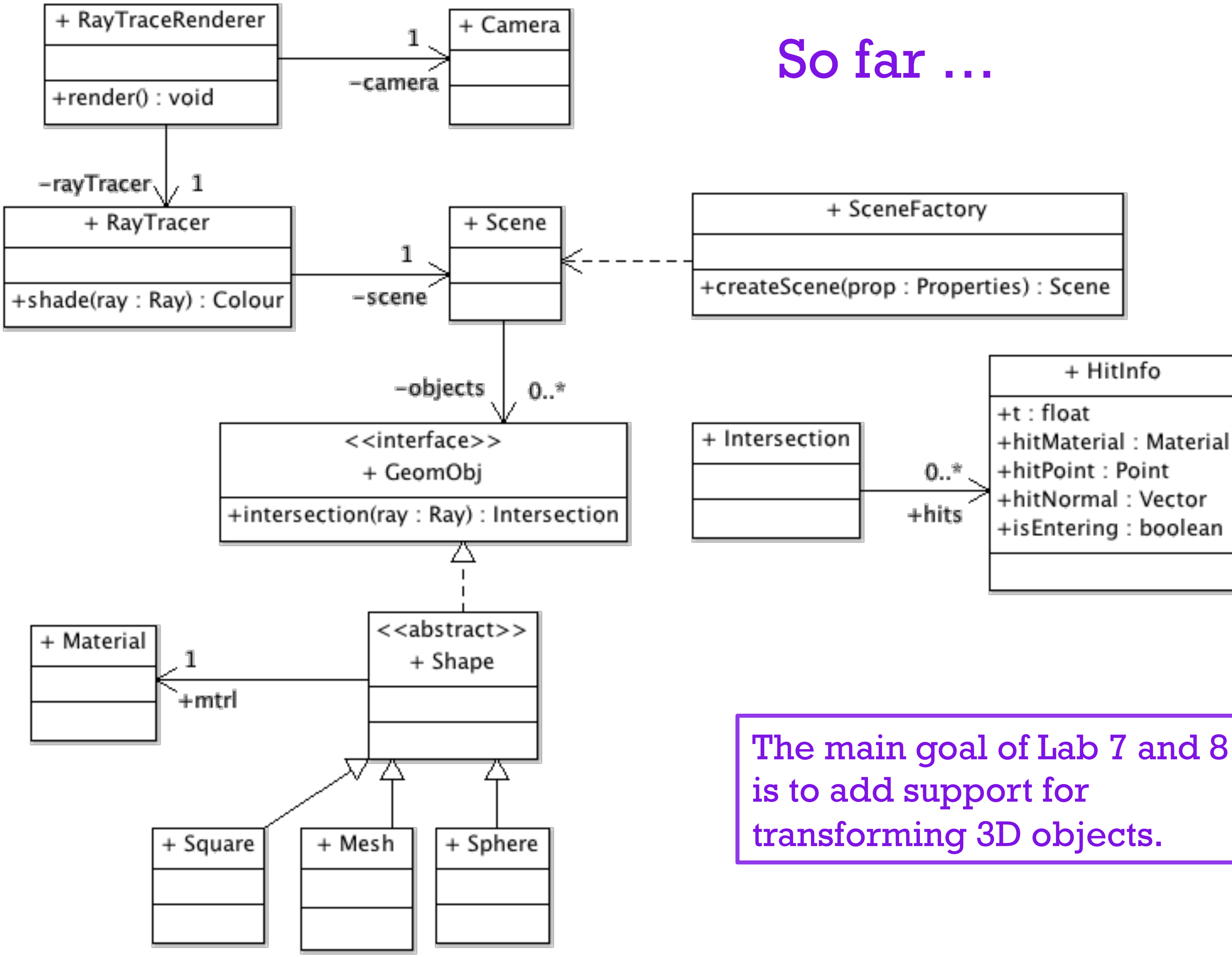
3D transformations

3D Computer Graphics (Lab 7)



So far …



The main goal of Lab 7 and 8 is to add support for transforming 3D objects.

Matrices

Matrix multiplication

- Example

$$A = \begin{pmatrix} 1 & 2 & -1 \\ 3 & 0 & -2 \end{pmatrix} \qquad B = \begin{pmatrix} 4 & -2 & 0 & 1 \\ -3 & 0 & 1 & -1 \\ 1 & 2 & -2 & 0 \end{pmatrix}$$

$C = A.B = ?$

- Condition: the number of columns of A is equal to the number of rows of B.

$A_{2x\mathbf{3}}.B_{\mathbf{3}x4}$

- The size of C=A.B?

$A_{\mathbf{2}x3}.B_{3x\mathbf{4}} = C_{2x4}$

-

$$C_{1,2}$$

$$\begin{pmatrix} 1 & 2 & -1 \\ 3 & 0 & -2 \end{pmatrix} . \begin{pmatrix} 4 & -2 & 0 & 1 \\ -3 & 0 & 1 & -1 \\ 1 & 2 & -2 & 0 \end{pmatrix} = \begin{pmatrix} . & \boxed{.} & . & . \\ . & . & . & . \end{pmatrix}$$

$1.(-2) + 2.0 + (-1).2 = -4$

Matrix multiplication

- Example

$$A = \begin{pmatrix} 1 & 2 & -1 \\ 3 & 0 & -2 \end{pmatrix} \qquad B = \begin{pmatrix} 4 & -2 & 0 & 1 \\ -3 & 0 & 1 & -1 \\ 1 & 2 & -2 & 0 \end{pmatrix}$$

C = A.B = ?

- Condition: the number of columns of A is equal to the number of rows of B.

$A_{2x\mathbf{3}}.B_{\mathbf{3}x4}$

- The size of C=A.B?

$A_{\mathbf{2}x3}.B_{3x\mathbf{4}} = C_{2x4}$

-

$$\begin{pmatrix} 1 & 2 & -1 \\ 3 & 0 & -2 \end{pmatrix} . \begin{pmatrix} 4 & -2 & 0 & 1 \\ -3 & 0 & 1 & -1 \\ 1 & 2 & -2 & 0 \end{pmatrix} = \begin{array}{c} \begin{array}{cccc} & C_{1,2} & & \end{array} \\ \begin{pmatrix} . & \boxed{-4} & . & . \\ . & . & . & . \end{pmatrix} \end{array}$$

1.(-2) + 2.0 + (-1).2 = -4

Matrix multiplication

- Example

$$A = \begin{pmatrix} 1 & 2 & -1 \\ 3 & 0 & -2 \end{pmatrix} \qquad B = \begin{pmatrix} 4 & -2 & 0 & 1 \\ -3 & 0 & 1 & -1 \\ 1 & 2 & -2 & 0 \end{pmatrix}$$

C = A.B = ?

- Condition: the number of columns of A is equal to the number of rows of B.

$A_{2x\mathbf{3}}.B_{\mathbf{3}x4}$

- The size of C=A.B?

$A_{\mathbf{2}x3}.B_{3x\mathbf{4}} = C_{2x4}$

-

$$\begin{pmatrix} 1 & 2 & -1 \\ 3 & 0 & -2 \end{pmatrix} . \begin{pmatrix} 4 & -2 & 0 & 1 \\ -3 & 0 & 1 & -1 \\ 1 & 2 & -2 & 0 \end{pmatrix} = \begin{pmatrix} . & -4 & . & . \\ . & . & \boxed{.} & . \end{pmatrix}$$

3.0 + 0.1 + (-2).(-2) = 4

$C_{2,3}$

Matrix multiplication

- Example

$$A = \begin{pmatrix} 1 & 2 & -1 \\ 3 & 0 & -2 \end{pmatrix} \qquad B = \begin{pmatrix} 4 & -2 & 0 & 1 \\ -3 & 0 & 1 & -1 \\ 1 & 2 & -2 & 0 \end{pmatrix}$$

C = A.B = ?

- Condition: the number of columns of A is equal to the number of rows of B.

$A_{2x\mathbf{3}}.B_{\mathbf{3}x4}$

- The size of C=A.B?

$A_{\mathbf{2}x3}.B_{3x\mathbf{4}} = C_{2x4}$

$$\begin{pmatrix} 1 & 2 & -1 \\ 3 & 0 & -2 \end{pmatrix} . \begin{pmatrix} 4 & -2 & 0 & 1 \\ -3 & 0 & 1 & -1 \\ 1 & 2 & -2 & 0 \end{pmatrix} = \begin{pmatrix} . & -4 & . & . \\ . & . & \boxed{4} & . \end{pmatrix}$$

$C_{2,3}$

3.0 + 0.1 + (-2).(-2) = 4

Matrix multiplication

- Example

$$A = \begin{pmatrix} 1 & 2 & -1 \\ 3 & 0 & -2 \end{pmatrix} \qquad B = \begin{pmatrix} 4 & -2 & 0 & 1 \\ -3 & 0 & 1 & -1 \\ 1 & 2 & -2 & 0 \end{pmatrix}$$

C = A.B = ?

- Condition: the number of columns of A is equal to the number of rows of B.

$A_{2x\mathbf{3}}.B_{\mathbf{3}x4}$

- The size of C=A.B?

$A_{\mathbf{2}x3}.B_{3x\mathbf{4}} = C_{2x4}$

-

$$\begin{pmatrix} 1 & 2 & -1 \\ 3 & 0 & -2 \end{pmatrix} . \begin{pmatrix} 4 & -2 & 0 & 1 \\ -3 & 0 & 1 & -1 \\ 1 & 2 & -2 & 0 \end{pmatrix} = \begin{pmatrix} . & -4 & . & . \\ . & . & 4 & . \end{pmatrix}$$

Matrix multiplication

- Example

$$A = \begin{pmatrix} 1 & 2 & -1 \\ 3 & 0 & -2 \end{pmatrix} \qquad B = \begin{pmatrix} 4 & -2 & 0 & 1 \\ -3 & 0 & 1 & -1 \\ 1 & 2 & -2 & 0 \end{pmatrix}$$

C = A.B = ?

- Condition: the number of columns of A is equal to the number of rows of B.

$A_{2x\mathbf{3}}.B_{\mathbf{3}x4}$

- The size of C=A.B? $A_{\mathbf{2}x3}.B_{3x\mathbf{4}} = C_{2x4}$

-

$$\begin{pmatrix} 1 & 2 & -1 \\ 3 & 0 & -2 \end{pmatrix} . \begin{pmatrix} 4 & -2 & 0 & 1 \\ -3 & 0 & 1 & -1 \\ 1 & 2 & -2 & 0 \end{pmatrix} = \begin{pmatrix} -3 & -4 & 4 & -1 \\ 10 & -10 & 4 & 3 \end{pmatrix}$$

Matrix multiplication

- Example

$$A = \begin{pmatrix} 1 & 2 & -1 \\ 3 & 0 & -2 \end{pmatrix} \qquad B = \begin{pmatrix} 4 & -2 & 0 & 1 \\ -3 & 0 & 1 & -1 \\ 1 & 2 & -2 & 0 \end{pmatrix}$$

C = A.B = ?

- Condition: the number of columns of A is equal to the number of rows of B.

$$A_{2x\mathbf{3}}.B_{\mathbf{3}x4}$$

- The size of C=A.B?

$$A_{\mathbf{2}x3}.B_{3x\mathbf{4}} = C_{2x4}$$

-

$$\begin{pmatrix} 1 & 2 & -1 \\ 3 & 0 & -2 \end{pmatrix} . \begin{pmatrix} 4 & -2 & 0 & 1 \\ -3 & 0 & 1 & -1 \\ 1 & 2 & -2 & 0 \end{pmatrix} = \begin{pmatrix} -3 & -4 & 4 & -1 \\ 10 & -10 & 4 & 3 \end{pmatrix}$$

Matrix multiplication

- Does the following property hold?

$$A.B = B.A$$

No, matrix multiplication is not commutative.

- A matrix does not change when it is multiplied with a unit matrix

$$A.I = I.A = A$$

- What is a unit matrix?

A square matrix with ones on the main diagonal and zeros elsewhere

$$I = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Matrices

- B is the inverse of the matrix A if and only if

  $$A.B = B.A = I$$

  A^{-1} cancels the effect of A

  The inverse of the matrix A is denoted by A^{-1}

- B is the transpose of the matrix A if and only if

  B is obtained by reflecting A over its main diagonal.

  The transpose of the matrix A is denoted by A^{T}

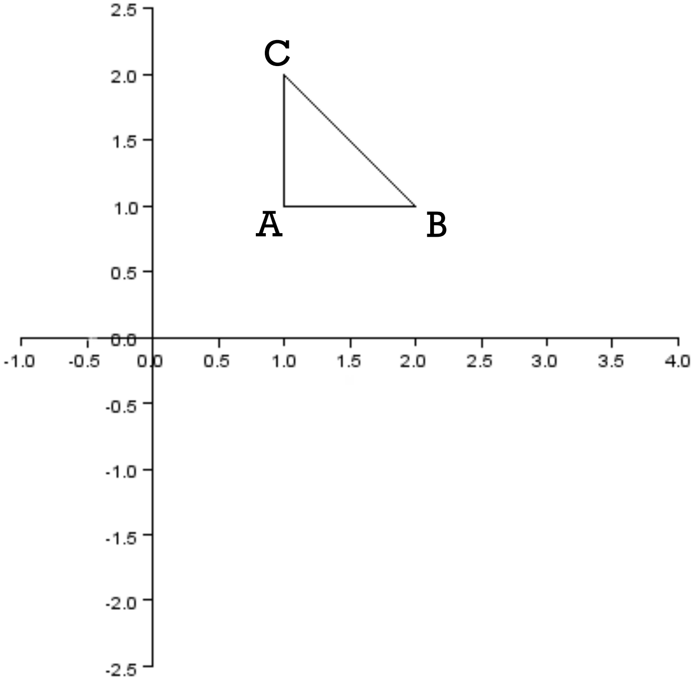
  Example

$$A = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{pmatrix} \Rightarrow A^T = \begin{pmatrix} 1 & 5 & 9 & 13 \\ 2 & 6 & 10 & 14 \\ 3 & 7 & 11 & 15 \\ 4 & 8 & 12 & 16 \end{pmatrix}$$

Applying one transformation to one shape

Example

- Consider the matrix $T = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$
- Consider the following triangle:



- It is completely determined by

$$A = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \qquad B = \begin{pmatrix} 2 \\ 1 \end{pmatrix} \qquad C = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

- We compute three new points

$$A' = T.A = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}\begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$$

$$B' = T.B = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}\begin{pmatrix} 2 \\ 1 \end{pmatrix} = \begin{pmatrix} 2 \\ -1 \end{pmatrix}$$

$$C' = T.C = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}\begin{pmatrix} 1 \\ 2 \end{pmatrix} = \begin{pmatrix} 1 \\ -2 \end{pmatrix}$$

Example

- Consider the matrix $T = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$
- Consider the following triangle:
- It is completely determined by

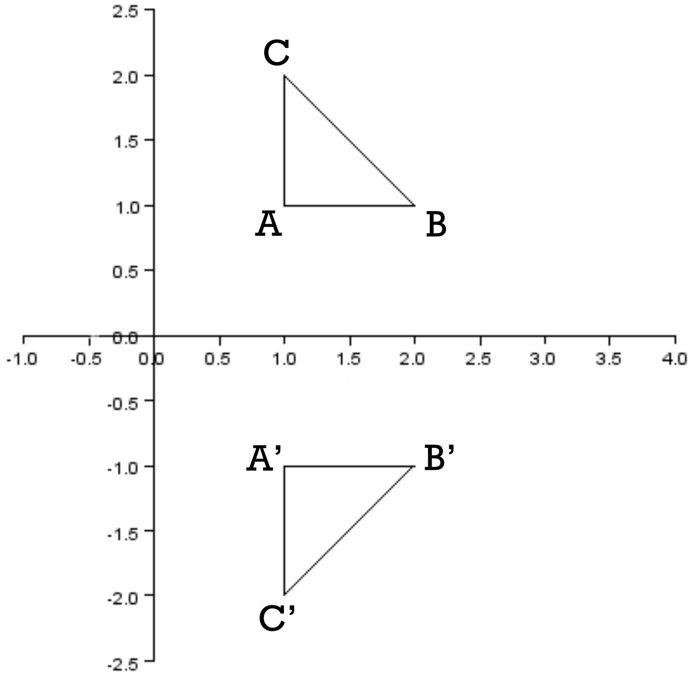
$$A = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \qquad B = \begin{pmatrix} 2 \\ 1 \end{pmatrix} \qquad C = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

- We compute three new points

$$A' = T.A = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}\begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$$

$$B' = T.B = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}\begin{pmatrix} 2 \\ 1 \end{pmatrix} = \begin{pmatrix} 2 \\ -1 \end{pmatrix}$$

$$C' = T.C = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}\begin{pmatrix} 1 \\ 2 \end{pmatrix} = \begin{pmatrix} 1 \\ -2 \end{pmatrix}$$



T is a transformation matrix which corresponds to a reflection with respect to the x-axis.

Idea

- Multiplying a matrix with a point results in a new point.
- A shape can be transformed by applying a matrix to all its vertices.
- This idea applies both in 2D and 3D.

Which transformations are we going to support?

- Rotation around the origin $\longrightarrow$ orientation of the shape
- Scaling $\longrightarrow$ size of the shape
- Translation $\longrightarrow$ position of the shape

We want a uniform way to represent these transformations.

- Matrices seem a good choice.
- What are the matrices corresponding to these transformations?

Scaling

- What is the matrix corresponding to a scaling in 2D by a factor s_x and s_y in the x- and y-direction, resp.?

$$\begin{pmatrix} s_x & 0 \\ 0 & s_y \end{pmatrix}$$

Example The matrix corresponding to a scaling by a factor 2 and 3 in the x- and y-direction, respectively is $\begin{pmatrix} 2 & 0 \\ 0 & 3 \end{pmatrix}$

So, scaling a point, say $\begin{pmatrix} 1 \\ 1 \end{pmatrix}$, by 2 and 3 in the x- and y-direction, resp., comes down to computing $\begin{pmatrix} 2 & 0 \\ 0 & 3 \end{pmatrix}\begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 2 \\ 3 \end{pmatrix}$

- What is the matrix corresponding to a scaling in 3D by a factor s_x, s_y and s_z in the x-, y- and z-direction, resp.?

$$\begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{pmatrix}$$

Example The matrix corresponding to a scaling by a factor 2, 4 and 5 in the x-, y- and z-direction, respectively is $\begin{pmatrix} 2 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 5 \end{pmatrix}$

So, scaling a point, say $\begin{pmatrix} 3 \\ 2 \\ 1 \end{pmatrix}$, by 2, 4 and 5 in the x-, y- and z-direction, resp., comes down to computing

$$\begin{pmatrix} 2 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 5 \end{pmatrix}\begin{pmatrix} 3 \\ 2 \\ 1 \end{pmatrix} = \begin{pmatrix} 6 \\ 8 \\ 5 \end{pmatrix}$$

Rotation

- The matrix corresponding to a rotation in 2D by an angle θ around the origin is given by

$$\begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

Example The matrix corresponding to a rotation by an angle 90° around the origin is $\begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$

So rotating a point, say $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ by 90° around the origin, comes down to computing

$$\begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

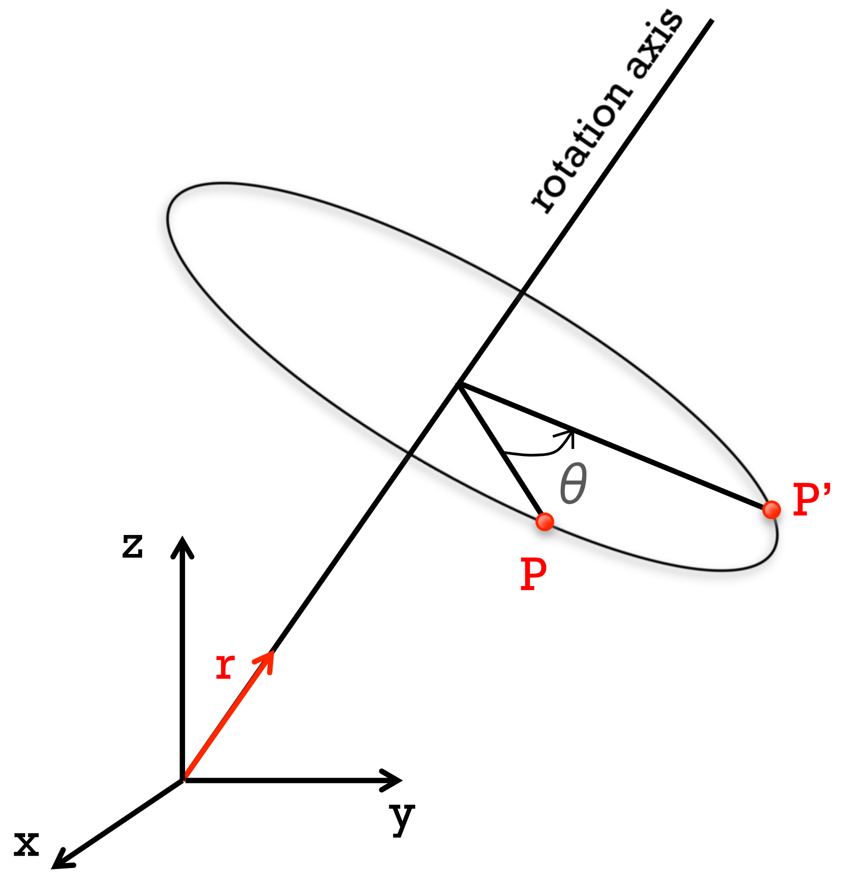
- For rotations in 3D, one also needs to specify the rotation axis.
  - A rotation in 3D by an angle θ around the x-axis $\begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{pmatrix}$
  - A rotation in 3D by an angle θ around the y-axis $\begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix}$
  - A rotation in 3D by an angle θ around the z-axis $\begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$

But we want to carry out rotations in 3D around an arbitrary axis.

How?

Remember …

Rotations in 3D around an arbitrary axis.



$r = (r_x, r_y, r_z)$ with $|r|=1$

θ

$P = (x, y, z)$

$P' = ?$

We want one uniform way to represent the 3 transformations: rotation, scaling and translation in 3D.

We are not going to use quaternions to carry out rotations in 3D around an arbitrary axis as scaling and translation operations cannot be represented by quaternions.

Rotation

- The matrix corresponding to a rotation in 3D by an angle θ around an arbitrary axis (indicated by the vector r = (r_x, r_y, r_z) with $|r|=1$) is given by

$$\begin{pmatrix} \cos\theta + r_x^2(1-\cos\theta) & r_x r_y(1-\cos\theta) - r_z\sin\theta & r_x r_z(1-\cos\theta) + r_y\sin\theta \\ r_x r_y(1-\cos\theta) + r_z\sin\theta & \cos\theta + r_y^2(1-\cos\theta) & r_y r_z(1-\cos\theta) - r_x\sin\theta \\ r_x r_z(1-\cos\theta) - r_y\sin\theta & r_y r_z(1-\cos\theta) + r_x\sin\theta & \cos\theta + r_z^2(1-\cos\theta) \end{pmatrix}$$

Example What is the matrix corresponding to a rotation by an angle 90° around the y-axis?

In this case, r = (0, 1, 0) and θ = 90°.

Furthermore, cos(90°) = 0 and sin(90°) = 1,

so the above matrix can be simplified to

$$\begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{pmatrix}$$

Hence rotating a point, say $\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$ by 90° around the y-axis, comes down to computing

$$\begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{pmatrix}\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

Summary

- A scaling in 3D by a factor s_x, s_y and s_z in the x-, y- and z-direction, resp.

$$\begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{pmatrix}$$

- A rotation in 3D by an angle θ around an arbitrary axis (indicated by the vector $r = (r_x, r_y, r_z)$ with $|r|=1$)

$$\begin{pmatrix} \cos\theta + r_x^2(1-\cos\theta) & r_x r_y(1-\cos\theta) - r_z \sin\theta & r_x r_z(1-\cos\theta) + r_y \sin\theta \\ r_x r_y(1-\cos\theta) + r_z \sin\theta & \cos\theta + r_y^2(1-\cos\theta) & r_y r_z(1-\cos\theta) - r_x \sin\theta \\ r_x r_z(1-\cos\theta) - r_y \sin\theta & r_y r_z(1-\cos\theta) + r_x \sin\theta & \cos\theta + r_z^2(1-\cos\theta) \end{pmatrix}$$

- Unfortunately, one can prove that a translation in 3D cannot be represented by a 3x3-matrix.

- But a translation in 3D can be carried out by means of a 4x4-matrix.

Translation

- A translation in 3D can be carried out by means of a 4x4-matrix as follows.

Example In order to translate a point $\begin{pmatrix} 3 \\ 0 \\ 2 \end{pmatrix}$ by 2, 4 and 5 units in the x-, y- and z-direction, resp.,

- Consider the transformation matrix $\begin{pmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 4 \\ 0 & 0 & 1 & 5 \\ 0 & 0 & 0 & 1 \end{pmatrix}$
- Use the homogeneous coordinates of the point.
  These can be obtained by adding a "1" to the cartesian coordinates $\begin{pmatrix} 3 \\ 0 \\ 2 \\ 1 \end{pmatrix}$
- Compute

$$\begin{pmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 4 \\ 0 & 0 & 1 & 5 \\ 0 & 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} 3 \\ 0 \\ 2 \\ 1 \end{pmatrix} = \begin{pmatrix} 5 \\ 4 \\ 7 \\ 1 \end{pmatrix}$$

Cartesian coordinates of the transformed point

Homogeneous coordinates of the transformed point

- What is the matrix corresponding to a translation by t_x, t_y and t_z units in the x-, y- and z-direction, resp.?

$$\begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

4x4-matrices

Scaling and rotations in 3D can be carried out by 3x3-matrices.

Translations in 3D can be carried out by 4x4-matrices.

Because we want a uniform approach, we will represent ALL transformations by 4x4-matrices.

So we will not represent a scaling in 3D by a factor s_x, s_y and s_z in the x-, y- and z-direction by a 3x3-matrix

$$\begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{pmatrix}$$

and apply it to the cartesian coordinates, say $\begin{pmatrix} 2 \\ 3 \\ 4 \end{pmatrix}$, of a point as follows

$$\begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{pmatrix}\begin{pmatrix} 2 \\ 3 \\ 4 \end{pmatrix} = \begin{pmatrix} 2s_x \\ 3s_y \\ 4s_z \end{pmatrix}$$

Instead, we will use the 4x4-matrix

$$\begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

and apply it to the homogeneous coordinates $\begin{pmatrix} 2 \\ 3 \\ 4 \\ 1 \end{pmatrix}$ as follows

$$\begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} 2 \\ 3 \\ 4 \\ 1 \end{pmatrix} = \begin{pmatrix} 2s_x \\ 3s_y \\ 4s_z \\ 1 \end{pmatrix}$$

4x4-matrices

Scaling and rotations in 3D can be carried out by 3x3-matrices.

Translations in 3D can be carried out by 4x4-matrices.

Because we want a uniform approach, we will represent ALL transformations by 4x4-matrices.

Similarly, we will not represent a rotation in 3D by an angle θ around an arbitrary axis (indicated by the vector r = (r_x, r_y, r_z) with $|r|=1$) by a 3x3-matrix and apply it to the cartesian coordinates of a point.

$$\begin{pmatrix} \cos\theta + r_x^2(1-\cos\theta) & r_x r_y(1-\cos\theta) - r_z\sin\theta & r_x r_z(1-\cos\theta) + r_y\sin\theta \\ r_x r_y(1-\cos\theta) + r_z\sin\theta & \cos\theta + r_y^2(1-\cos\theta) & r_y r_z(1-\cos\theta) - r_x\sin\theta \\ r_x r_z(1-\cos\theta) - r_y\sin\theta & r_y r_z(1-\cos\theta) + r_x\sin\theta & \cos\theta + r_z^2(1-\cos\theta) \end{pmatrix}$$

Instead, we will use the 4x4-matrix

$$\begin{pmatrix} \cos\theta + r_x^2(1-\cos\theta) & r_x r_y(1-\cos\theta) - r_z\sin\theta & r_x r_z(1-\cos\theta) + r_y\sin\theta & 0 \\ r_x r_y(1-\cos\theta) + r_z\sin\theta & \cos\theta + r_y^2(1-\cos\theta) & r_y r_z(1-\cos\theta) - r_x\sin\theta & 0 \\ r_x r_z(1-\cos\theta) - r_y\sin\theta & r_y r_z(1-\cos\theta) + r_x\sin\theta & \cos\theta + r_z^2(1-\cos\theta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

and apply it to the homogeneous coordinates of this point.

Summary

Our rendering framework will support three transformations in 3D.

These transformations are represented in a uniform way (4x4-matrices).

- A scaling in 3D by a factor s_x, s_y and s_z in the x-, y- and z-direction, resp.

$$\begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- A rotation in 3D by an angle θ around an arbitrary axis (indicated by the vector $r = (r_x, r_y, r_z)$ with $|r|=1$)

$$\begin{pmatrix} \cos\theta + r_x^2(1-\cos\theta) & r_x r_y(1-\cos\theta) - r_z\sin\theta & r_x r_z(1-\cos\theta) + r_y\sin\theta & 0 \\ r_x r_y(1-\cos\theta) + r_z\sin\theta & \cos\theta + r_y^2(1-\cos\theta) & r_y r_z(1-\cos\theta) - r_x\sin\theta & 0 \\ r_x r_z(1-\cos\theta) - r_y\sin\theta & r_y r_z(1-\cos\theta) + r_x\sin\theta & \cos\theta + r_z^2(1-\cos\theta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- A translation by t_x, t_y and t_z units in the x-, y- and z-direction, resp.

$$\begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Inverse transformation

- In order to transform a 3D object, the corresponding transformation matrix T is applied to all its vertices.

- How can we cancel the effect of the transformation?

  By applying T^{-1}, the inverse of the matrix T, to all vertices, because

  $$T^{-1}.T = I$$

- So we also need to know the inverse of all the transformation matrices our rendering framework supports.

  - Scaling in 3D

$$T_s = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad T_s^{-1} = \begin{pmatrix} 1/s_x & 0 & 0 & 0 \\ 0 & 1/s_y & 0 & 0 \\ 0 & 0 & 1/s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Inverse transformation

- Rotation in 3D

$$T_r = \begin{pmatrix} \cos\theta + r_x^2(1-\cos\theta) & r_x r_y(1-\cos\theta) - r_z\sin\theta & r_x r_z(1-\cos\theta) + r_y\sin\theta & 0 \\ r_x r_y(1-\cos\theta) + r_z\sin\theta & \cos\theta + r_y^2(1-\cos\theta) & r_y r_z(1-\cos\theta) - r_x\sin\theta & 0 \\ r_x r_z(1-\cos\theta) - r_y\sin\theta & r_y r_z(1-\cos\theta) + r_x\sin\theta & \cos\theta + r_z^2(1-\cos\theta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Simply change θ by $-\theta$ and make use of the property that $\cos(-\theta)=\cos(\theta)$ and $\sin(-\theta)=-\sin(\theta)$

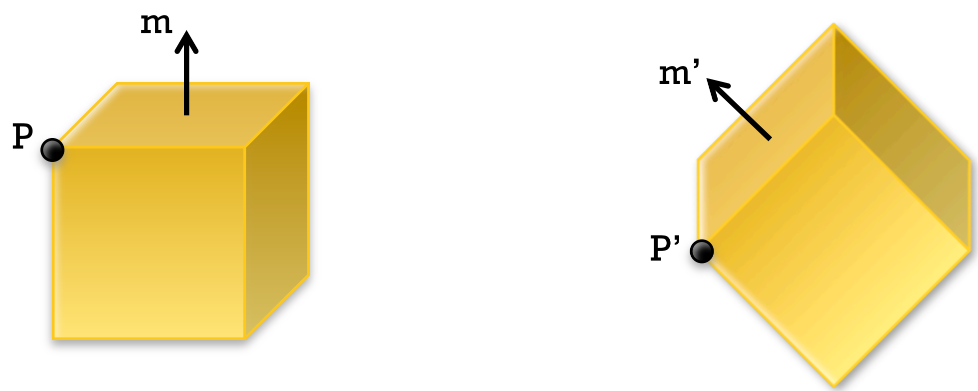
$$T_r^{-1} = \begin{pmatrix} \cos\theta + r_x^2(1-\cos\theta) & r_x r_y(1-\cos\theta) + r_z\sin\theta & r_x r_z(1-\cos\theta) - r_y\sin\theta & 0 \\ r_x r_y(1-\cos\theta) - r_z\sin\theta & \cos\theta + r_y^2(1-\cos\theta) & r_y r_z(1-\cos\theta) + r_x\sin\theta & 0 \\ r_x r_z(1-\cos\theta) + r_y\sin\theta & r_y r_z(1-\cos\theta) - r_x\sin\theta & \cos\theta + r_z^2(1-\cos\theta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Translation in 3D

$$T_t = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad T_t^{-1} = \begin{pmatrix} 1 & 0 & 0 & -t_x \\ 0 & 1 & 0 & -t_y \\ 0 & 0 & 1 & -t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Transforming normal vectors

Assume we want to rotate a generic cube by 45° around the z-axis.



Hereto, we will apply the corresponding rotation matrix T to the eight vertices of the generic cube to get the eight vertices of the rotated cube.

For example $T.P = P'$

Note that the normal vectors should also be transformed! How?

Transforming normal vectors

- Theorem

> If the vertices of a 3D object are transformed by a matrix T,
> then the normal vectors of this 3D object are transformed by the matrix $(T^{-1})^T$.

- What does it mean that a normal vector m is transformed by the matrix $(T^{-1})^T$?
  - The new normal vector m' is given by $m' = (T^{-1})^T.m$.
  - What is T? a 4x4-matrix
  - What is T^{-1}? a 4x4-matrix
  - What is $(T^{-1})^T$? a 4x4-matrix
  - What is m? a vector with 3 coordinates (= a 3x1-matrix)
- But we cannot multiply a 4x4-matrix with a 3x1-matrix!

Solution? Use the homogeneous coordinates of the vector m.

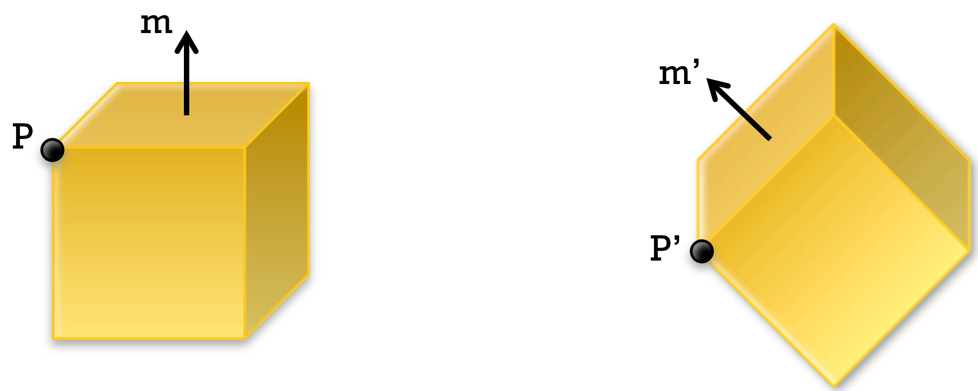
What are the homogeneous coordinates of a vector?

Add "0" to the cartesian coordinates of the vector.

- Note that a normal vector which is transformed in this way, does not necessarily have length 1 anymore!

Transforming normal vectors

Assume we want to rotate a generic cube by 45° around the z-axis.



Hereto, we will apply the corresponding rotation matrix T to the eight vertices of the generic cube to get the eight vertices of the rotated cube.

For example $T.P = P'$

Note that the normal vectors should also be transformed! How?

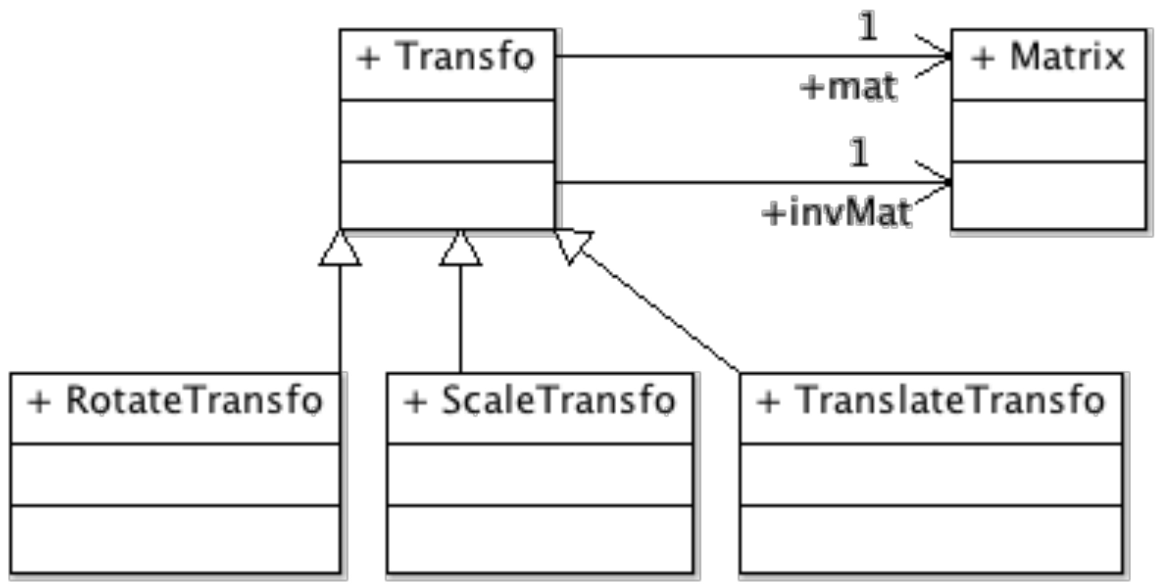
$$(T^{-1})^T.m = m'$$

normalize m'

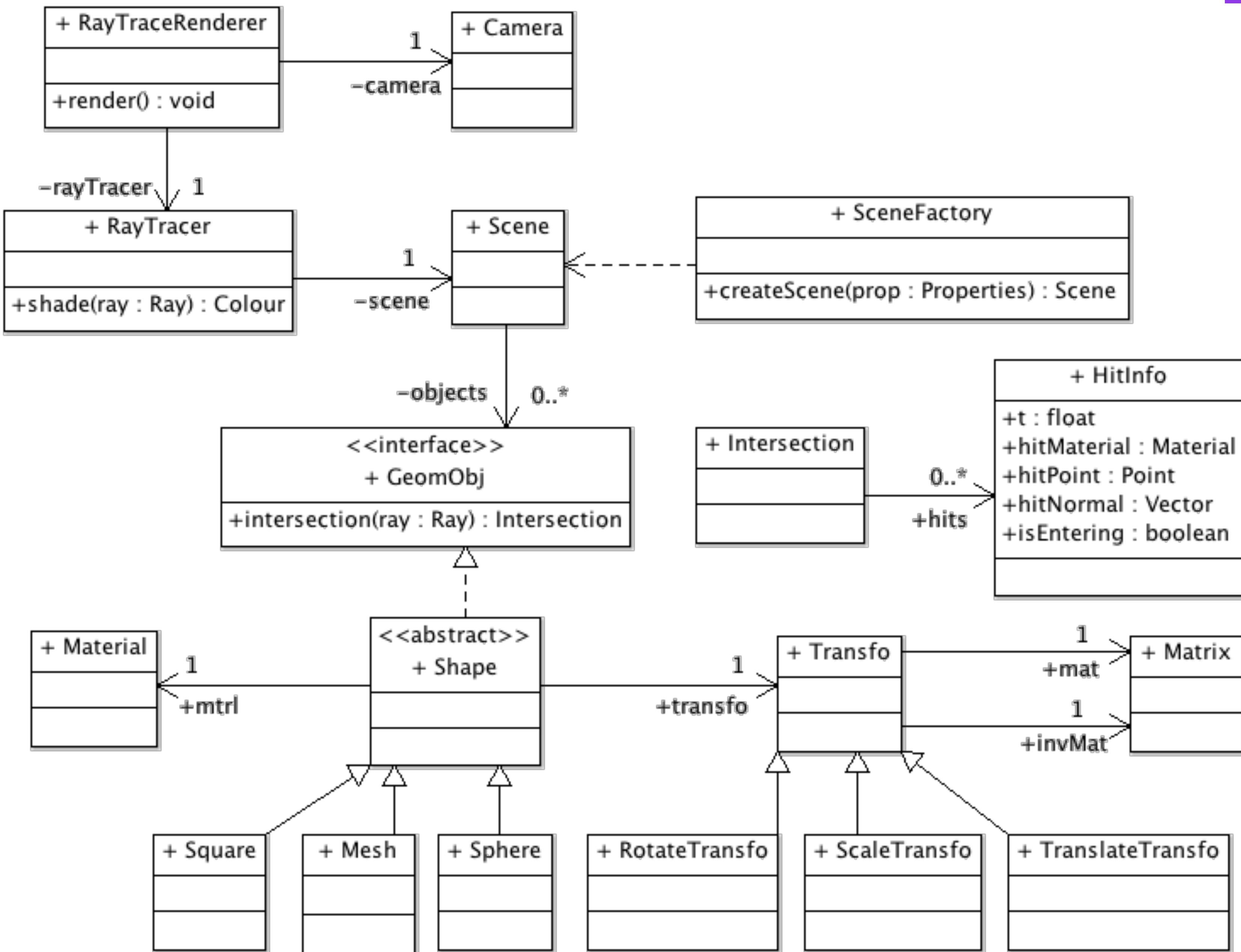
Summary

- 3D objects can be transformed by premultiplying all vertices with the corresponding transformation matrix T.
- The normal vectors should also be updated in this case by premultiplying them with $(T^{-1})^T$.
- All transformations will be represented in our code by 4x4-matrices.
- In order to be able to multiply a 4x4-matrix with a point or a vector, we need to consider the homogeneous coordinates of this point or vector.
    - for a point: cartesian coordinates + "1"
    - for a vector: cartesian coordinates + "0"

Implementation



Where will we plug these classes
into our rendering framework?


+ RayTraceRenderer
+render() : void
+ Camera
1
-camera
-rayTracer
1
+ RayTracer
+shade(ray : Ray) : Colour
1
-scene
+ Scene
+ SceneFactory
+createScene(prop : Properties) : Scene
-objects
0..*
<<interface>>
+ GeomObj
+intersection(ray : Ray) : Intersection
+ Intersection
0..*
+hits
+ HitInfo
+t : float
+hitMaterial : Material
+hitPoint : Point
+hitNormal : Vector
+isEntering : boolean
+ Material
1
+mtrl
<<abstract>>
+ Shape
1
+transfo
+ Transfo
1
+mat
1
+invMat
+ Matrix
+ Square
+ Mesh
+ Sphere
+ RotateTransfo
+ ScaleTransfo
+ TranslateTransfo

3D transformations and sdl file

Example 1

```
background 1 1 1
light -10 10 10 1 1 1
diffuse 0 0 1
ambient 0 0 0.5
scale 2 1 1
sphere
```

The sphere should be rendered after being scaled by a factor 2 in the x-direction.

Example 2

```
background 1 1 1
light -10 10 10 1 1 1
diffuse 0 0 1
ambient 0 0 0.5
rotate 45 0 1 0
sphere
```

The sphere should be rendered after being rotated by 45° around the y-axis.

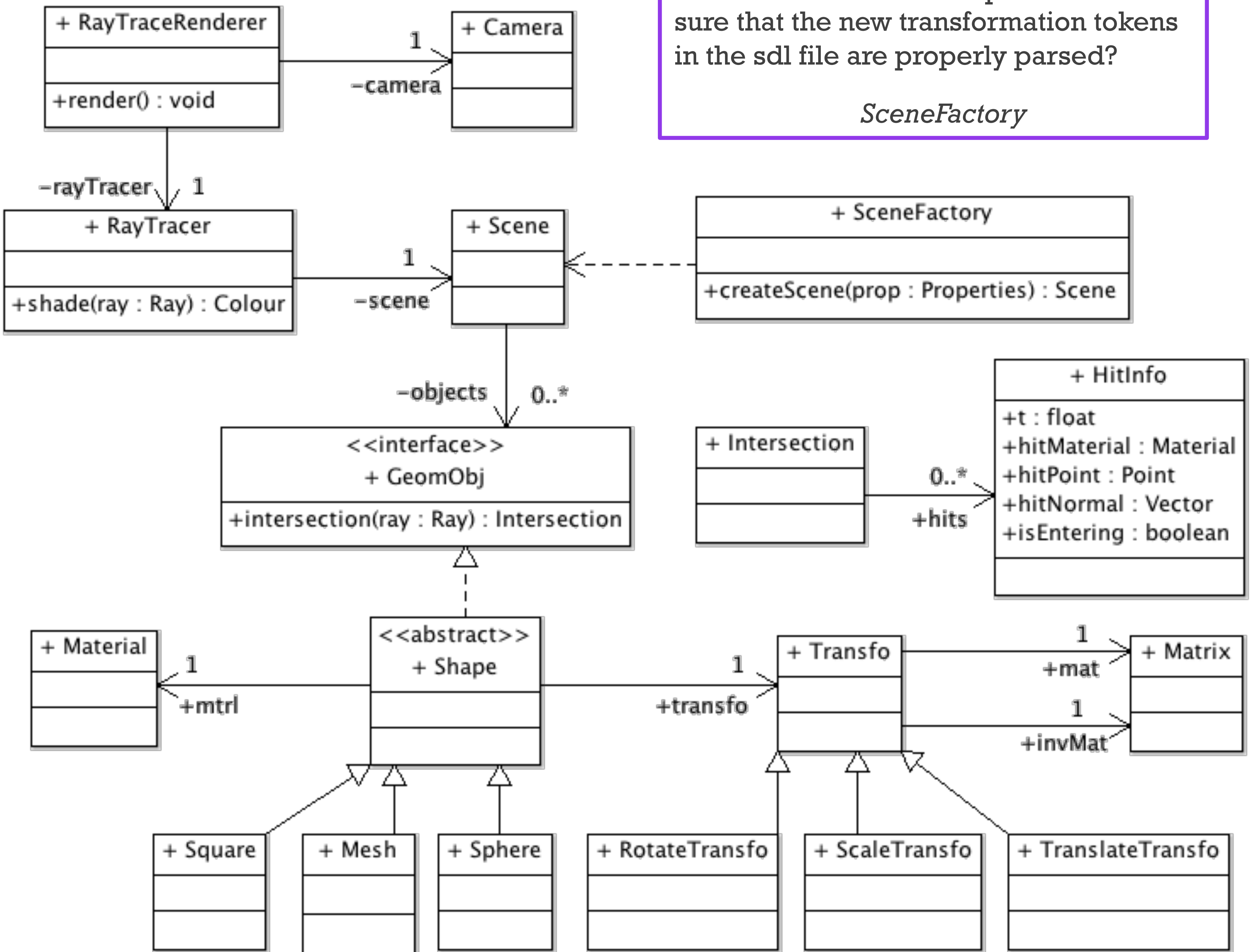
Example 3

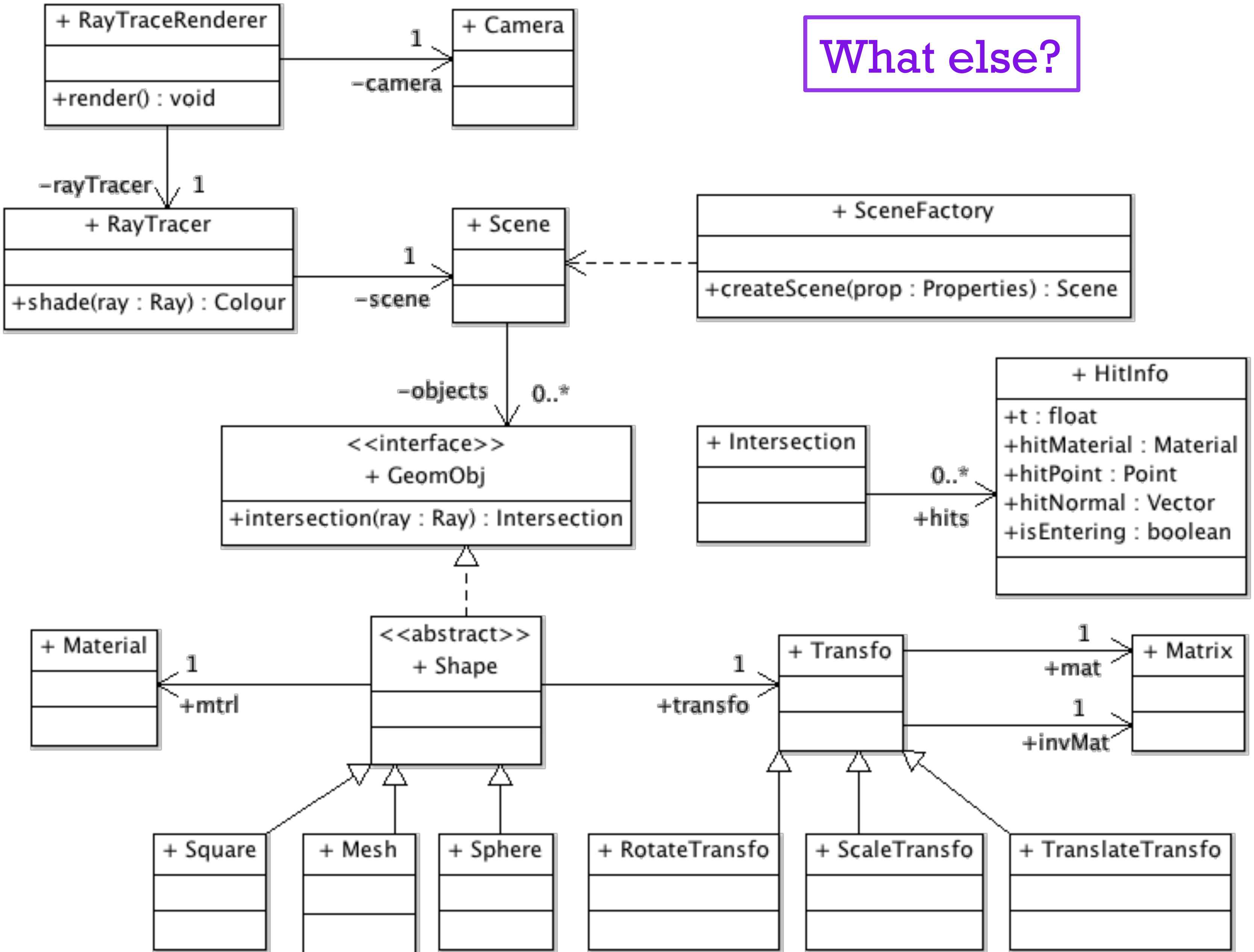
```
background 1 1 1
light -10 10 10 1 1 1
diffuse 0 0 1
ambient 0 0 0.5
translate 0 0 3
sphere
```

The sphere should be rendered after being translated 3 units in the z-direction.

Which class should be updated to make sure that the new transformation tokens in the sdl file are properly parsed?

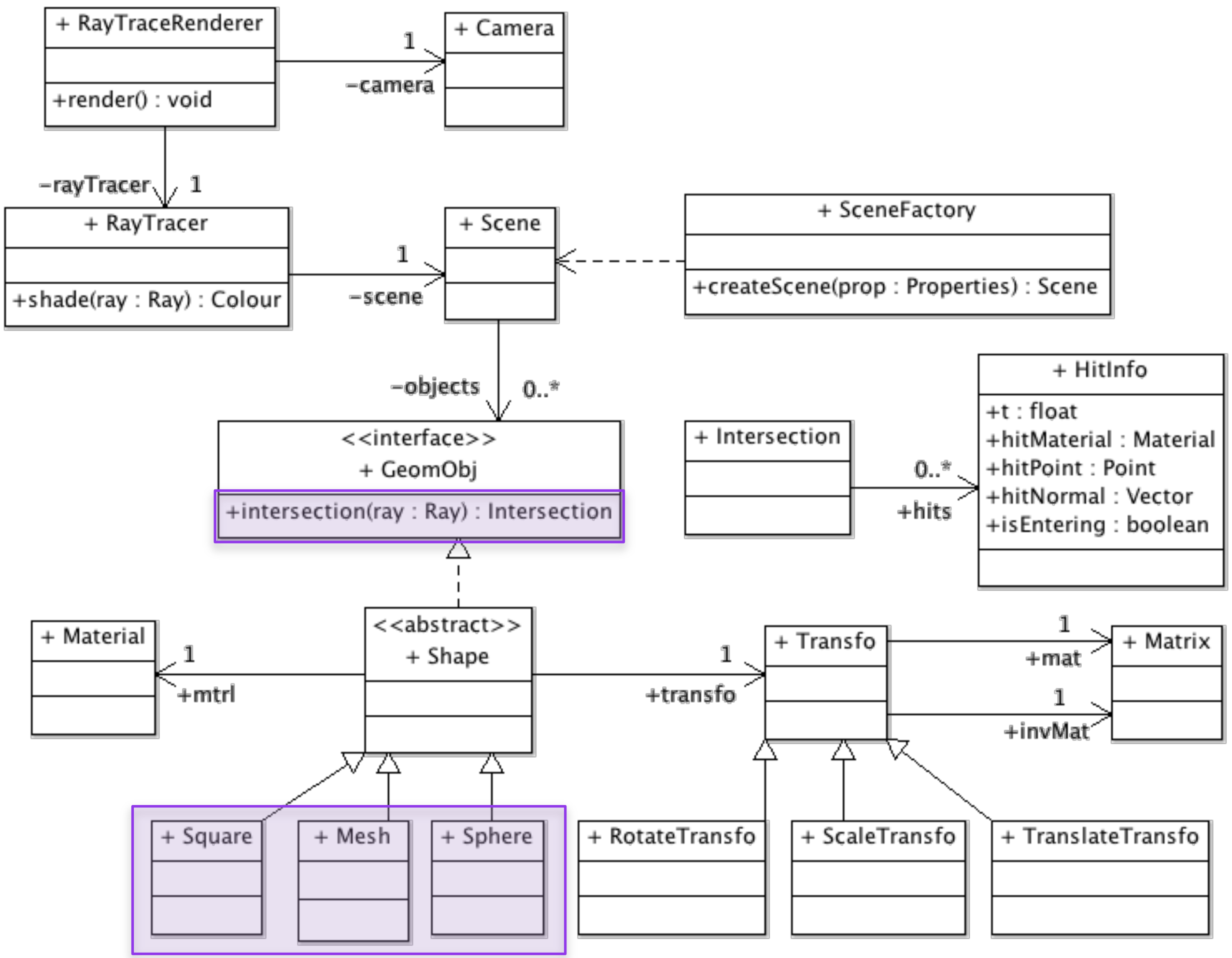
SceneFactory





+ RayTraceRenderer
+render() : void
1
-camera
+ Camera
-rayTracer
1
+ RayTracer
+shade(ray : Ray) : Colour
1
-scene
+ Scene
+ SceneFactory
+createScene(prop : Properties) : Scene
-objects
0..*
<<interface>>
+ GeomObj
+intersection(ray : Ray) : Intersection
+ Intersection
0..*
+hits
+ HitInfo
+t : float
+hitMaterial : Material
+hitPoint : Point
+hitNormal : Vector
+isEntering : boolean
+ Material
1
+mtrl
<<abstract>>
+ Shape
1
+transfo
+ Transfo
1
+mat
1
+invMat
+ Matrix
+ Square
+ Mesh
+ Sphere
+ RotateTransfo
+ ScaleTransfo
+ TranslateTransfo

What else?



A 3D object can only be rendered if its position and shape is known.

How does our raytracer know the position and shape of a 3D object?

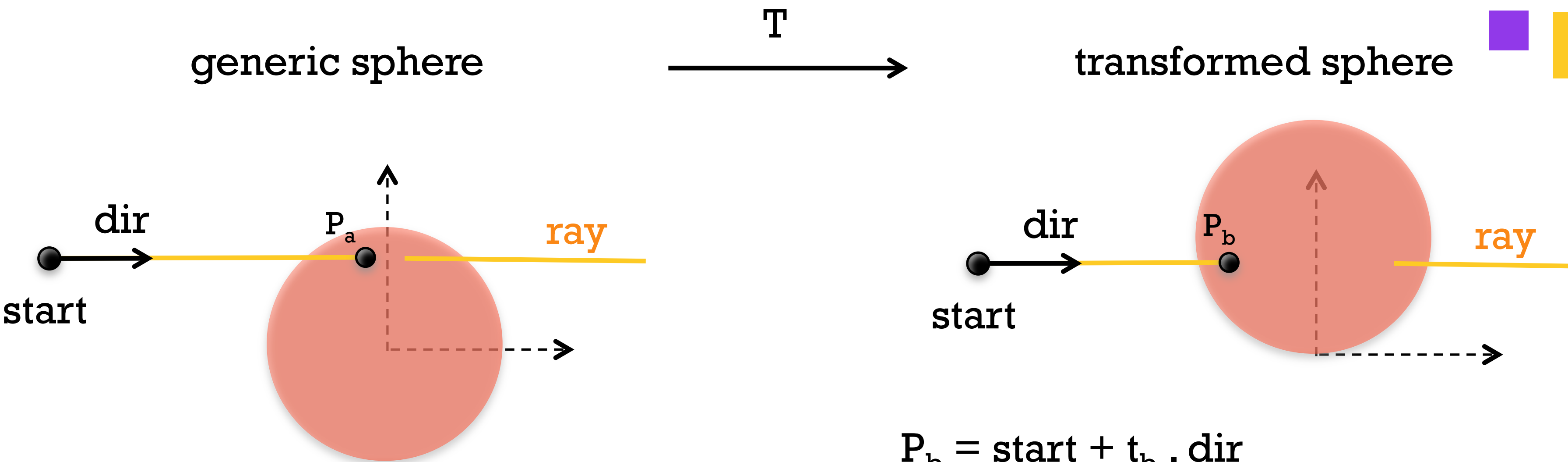
By casting rays from the eye of the camera into the scene and computing the intersection points between these rays and the 3D object.

However, the current intersection calculations ignore the transformation applied to the 3D object.

For example, in the intersection method of the Sphere class, we solved the equation

$$(dir.dir)\ t^2 + (2.start.dir)\ t + start.start - 1 = 0$$

The solution only yields the t-values of the intersection points between a generic sphere and a ray and not the t-values of the intersection points between a transformed sphere and a ray.



$P_a = start + t_a \,.\, dir$

t_a, the t-value of the closest hitpoint (P_a) between the ray and the generic sphere can be found as a solution of

$(dir.dir)\, t^2 + (2.start.dir)\, t + start.start - 1 = 0$

This equation is only valid for the intersection between a ray and a generic sphere!

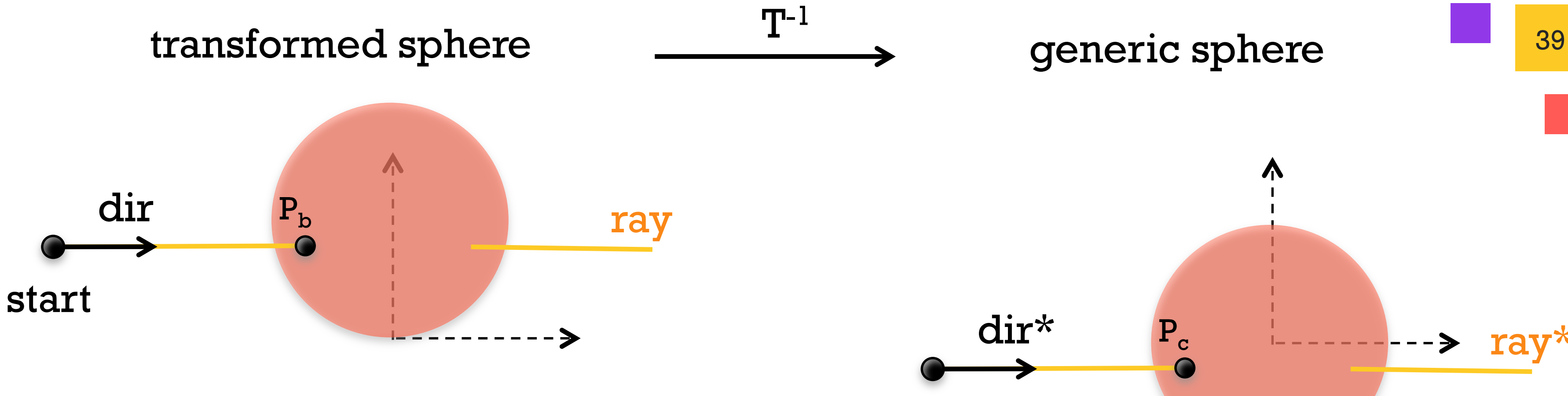
$P_b = start + t_b \,.\, dir$

Now, we want to find t_b, the t-value of the closest hitpoint (P_b) between the ray and the transformed sphere.

Therefore, we need a new equation which is valid for the intersection between a ray and the transformed sphere.

Does this mean we need a new equation for every possible transformation T ???

Luckily not …



$P_b = start + t_b \,.\, dir$

$P_c = start^* + t_c \,.\, dir^*$

Apply T^{-1} to both the transformed sphere and the ray.

- Applying T^{-1} to the transformed sphere yields the generic sphere.
- Applying T^{-1} to the ray yields ray*, the inverse transformed ray.

ray* has startpoint $start^* = T^{-1}(start)$

and direction $dir^* = T^{-1}(dir)$

Note that because we apply T^{-1} to BOTH the transformed sphere and the ray, $\boxed{t_b = t_c}$

The value t_c is the t-value of the closest hitpoint between ray* and the generic sphere.

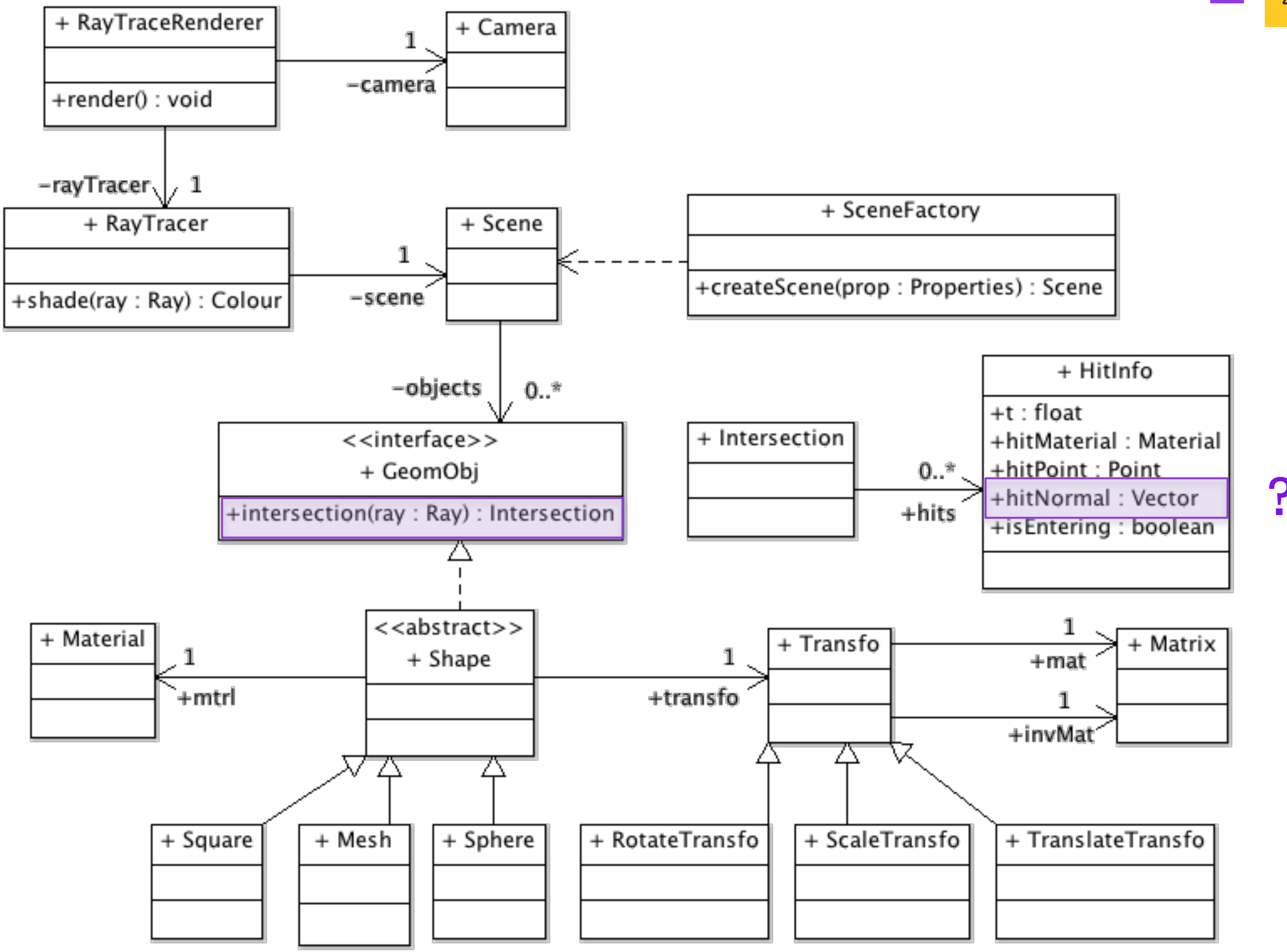
t_c can be computed because we have the equation for the intersection between a ray and the generic sphere.

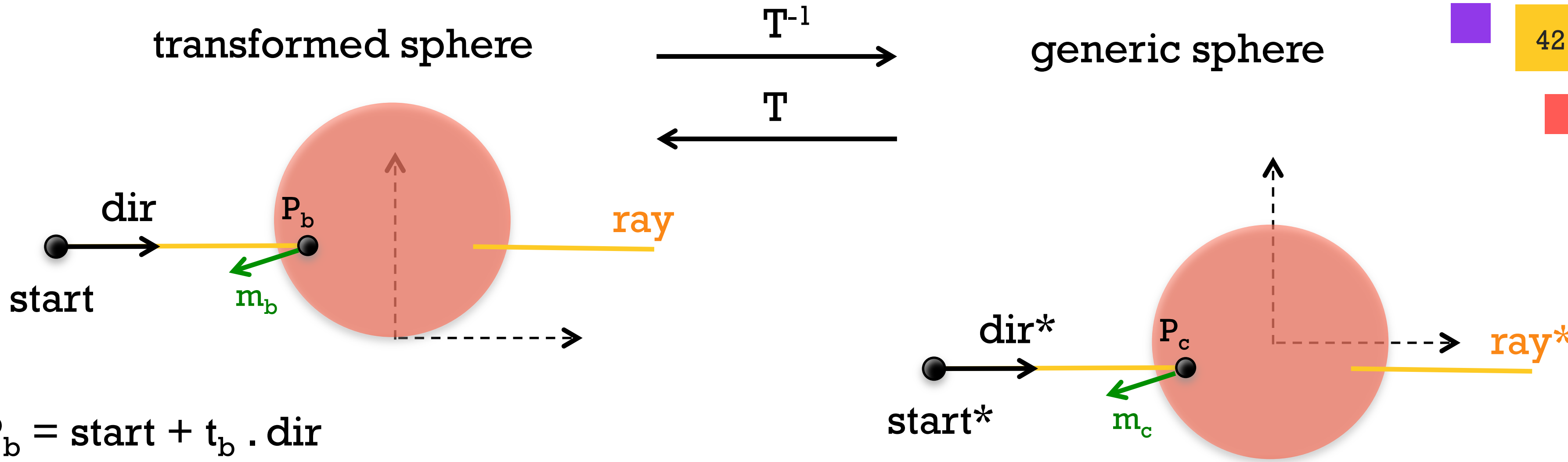
As soon as we have $t_b = t_c$, we can also compute the hitpoint between the ray and the transformed sphere

$$P_b = start + t_b \,.\, dir$$

Conclusion

- Assume you know how to find the t-values of the hitpoints between a generic object and a ray.
- How can you find the hitpoints between the object obtained by applying a transformation T to the generic object, and a ray?
  - Compute ray*, the inverse transformed ray,
    - start* = T^{-1}(start)
    - dir* = T^{-1}(dir)
  - Compute the t-values between the generic object and ray*.
  - Use the original ray to get the coordinates of the corresponding hitpoint.


+ RayTraceRenderer
+render() : void
+ Camera
1
-camera
-rayTracer
1
+ RayTracer
+shade(ray : Ray) : Colour
+ Scene
1
-scene
+ SceneFactory
+createScene(prop : Properties) : Scene
-objects
0..*
<<interface>>
+ GeomObj
+intersection(ray : Ray) : Intersection
+ Intersection
0..*
+hits
+ HitInfo
+t : float
+hitMaterial : Material
+hitPoint : Point
+hitNormal : Vector
+isEntering : boolean
?
+ Material
1
+mtrl
<<abstract>>
+ Shape
1
+transfo
+ Transfo
1
+mat
1
+invMat
+ Matrix
+ Square
+ Mesh
+ Sphere
+ RotateTransfo
+ ScaleTransfo
+ TranslateTransfo



P_b = start + t_b . dir

P_c = start* + t_c . dir*

And what about the hitNormal m_b, the normal vector to the transformed sphere at P_b?

- It is straightforward to compute the normal vector to the generic sphere at a particular point.
  So you can compute the m_c, the normal vector to the generic sphere at P_c

- You obtained the transformed sphere by applying T to the generic sphere.
- Hence, the hitNormal m_b can be computed as follows:

$$\boxed{m_b = (T^{-1})^T . m_c}$$

+ normalization

Final conclusion

- Assume you know
  - how to find the t-values of the hitpoints between a generic object and a ray,
  - how to compute the hitNormal to the generic object at the hitpoints.
- How can you find the hitpoints between the object obtained by applying a transformation T to the generic object, and a ray?
  - Compute ray*, the inverse transformed ray,
    - start* = T^{-1}(start)
    - dir* = T^{-1}(dir)
  - Compute the t-values between the generic object and ray*.
  - Use the original ray to get the coordinates of the corresponding hitpoint.
  - Compute the hitNormal to the generic object, premultiply it with $(T^{-1})^T$, and normalize the result.

Questions?