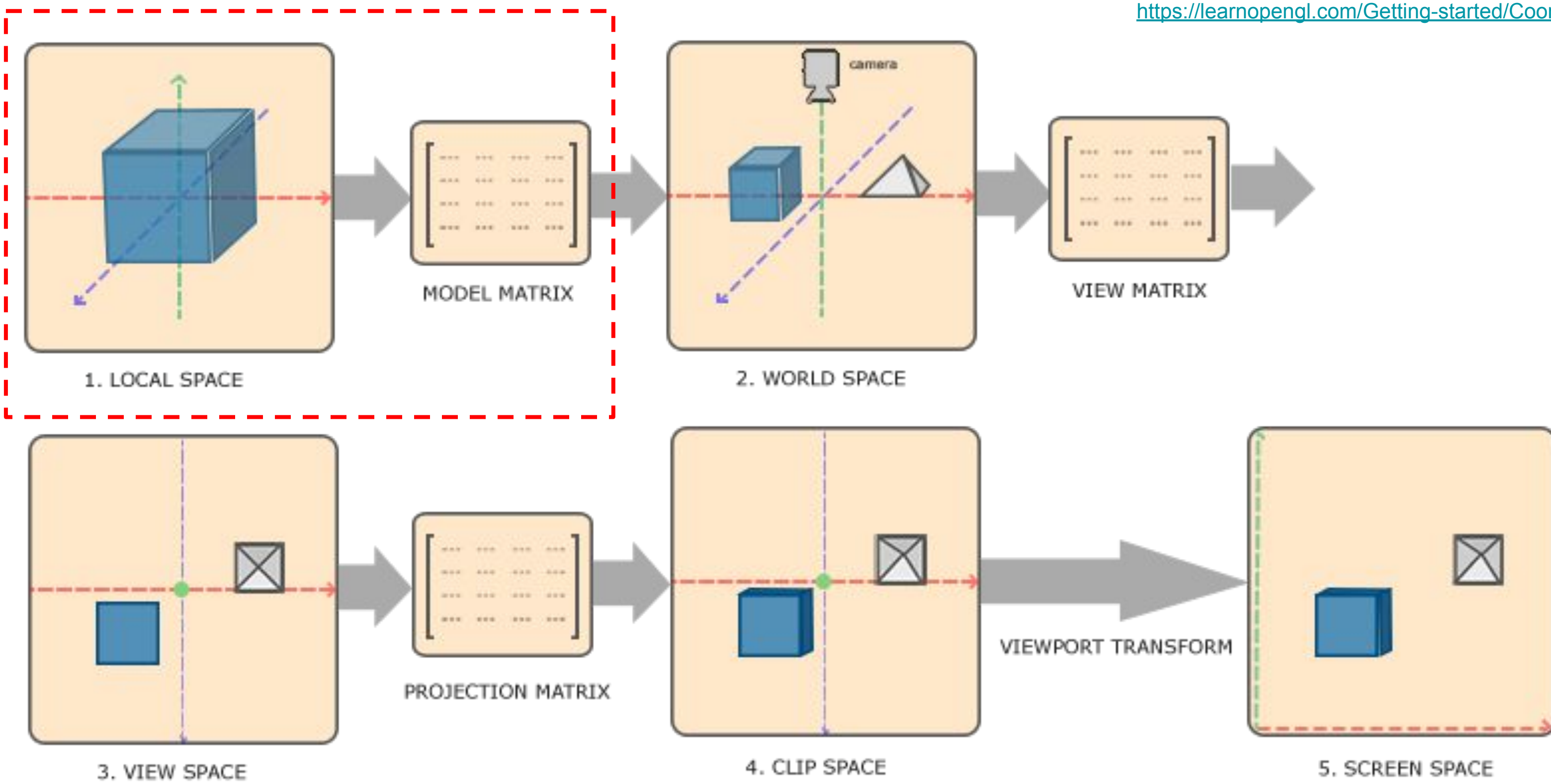
Affine Transformations and Modeling-Rendering Paradigm

CSE606: Computer Graphics
Jaya Sreevalsan Nair, IIIT Bangalore
January 27, 2025

Affine Transformations

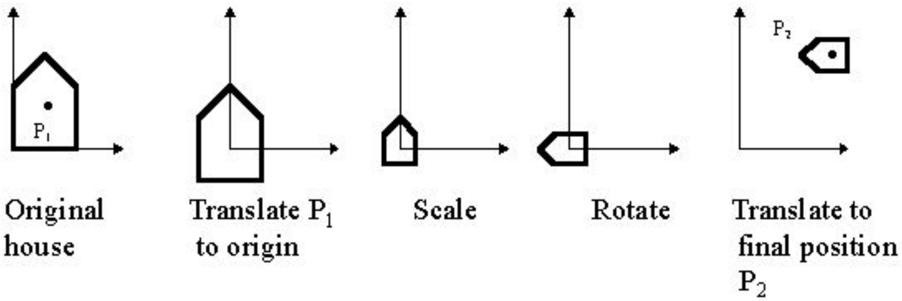
Coordinate Systems in Computer Graphics: Implementation

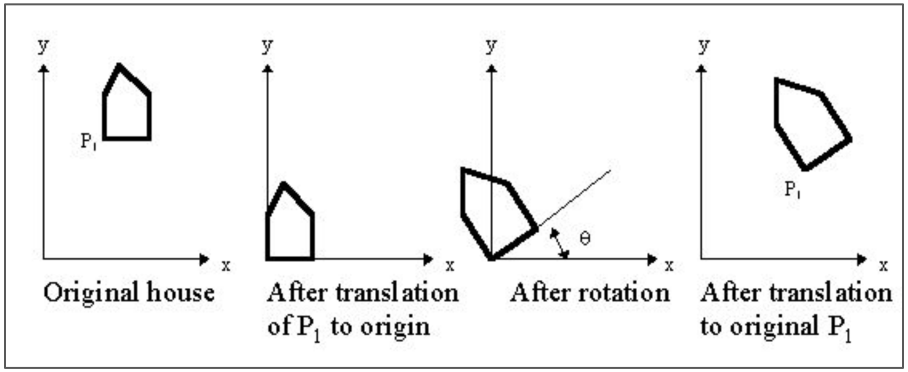
https://learnopengl.com/Getting-started/Coordinate-Systems

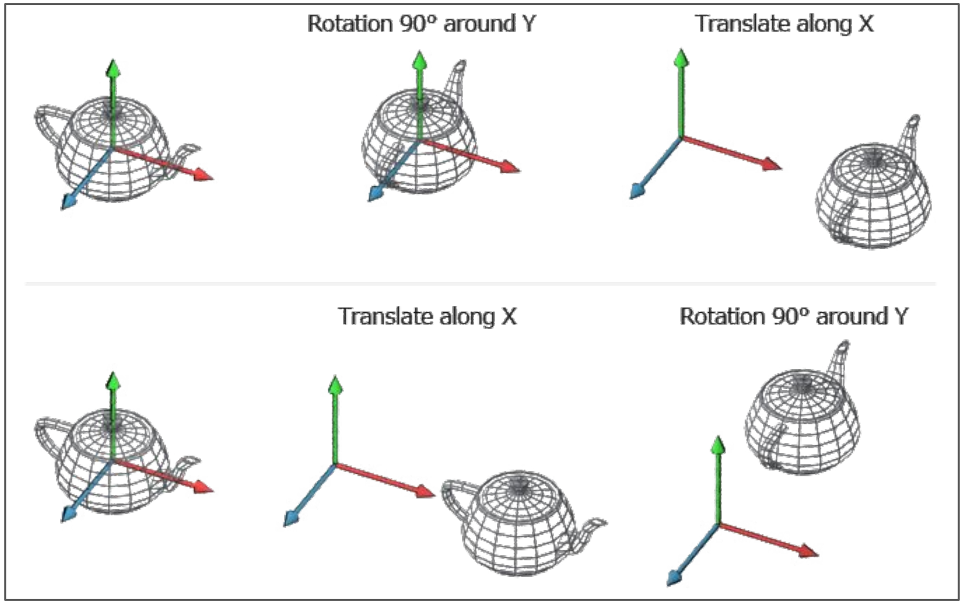


Model Transformations: Examples - 2D and 3D

Model transformations fall under the class of affine transformations.



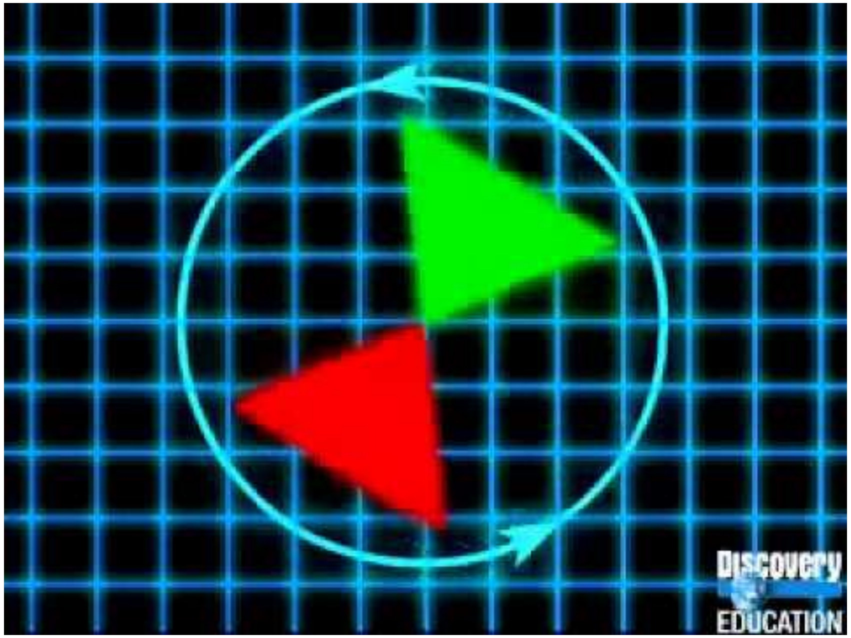




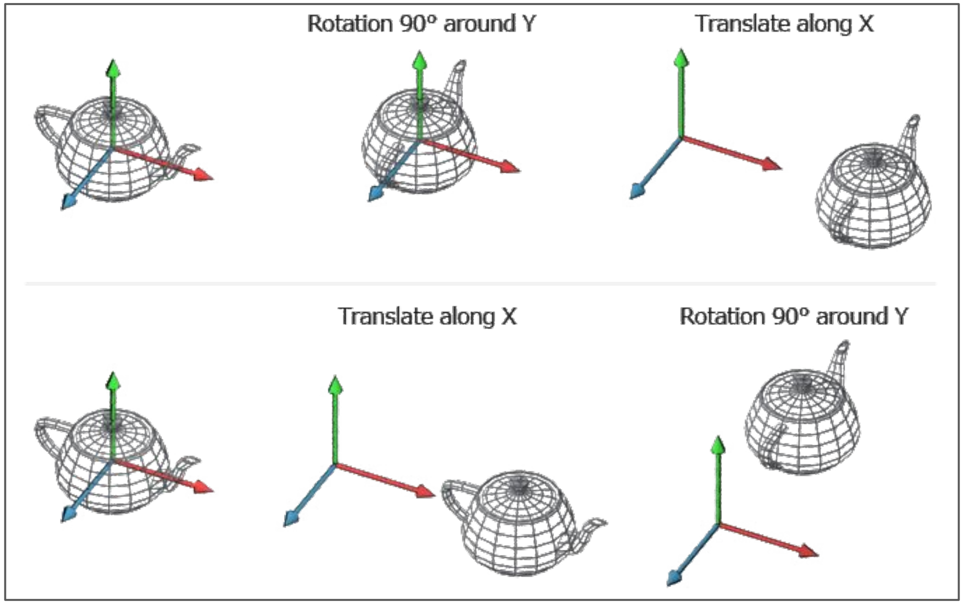
http://www.codinglabs.net/article_world_view_projection_matrix.aspx

https://www.docjava.com/java/javalect/ub/cs411x/lect11/lect11.htm

Model Transformations: Examples - 2D and 3D



Model transformations fall under the class of affine transformations.



https://www.youtube.com/watch?v=NY2cDTpsvBA

http://www.codinglabs.net/article_world_view_projection_matrix.aspx

Introduction to Transformations

Definition: A transformation is a function that maps a point (or vector) to another point (or vector).

In functional form, Q = T(P), **v** = R(**u**)

Transformations are too general to be useful, hence a (restricted) class of transformations is used in computer graphics, namely, the **linear transformations**.

Introduction to Transformations

Definition: A transformation is a function that maps a point (or vector) to another point (or vector).

In functional form, Q = T(P), **v** = R(**u**)

Transformations are too general to be useful, hence a (restricted) class of transformations is used in computer graphics, namely, the **linear transformations**.

Linear transformation is defined as: $f(\alpha p + \beta q) = \alpha f(p) + \beta f(q)$

Linear transformations can be implemented using **matrix multiplications**.

- Using matrix representation theorem of linear transformations;
- Now, extending vector algebra (of handling vertices) to matrix algebra.

Examples of linear transformations: rotation, scaling

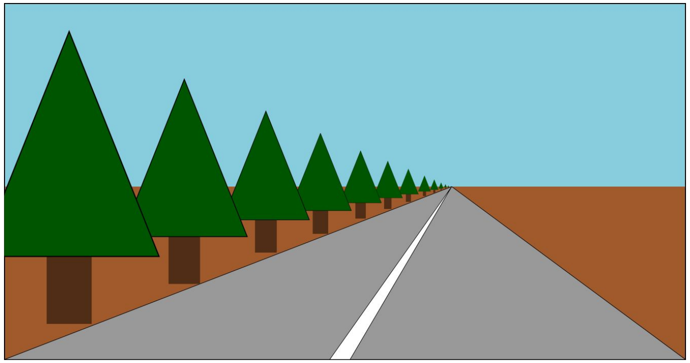
Linear Transformations

Linear transformation is defined as: $f(\alpha p + \beta q) = \alpha f(p) + \beta f(q)$

Implication: **Transformations of linear combination** of entities is a **linear combination of transformation** of the entities.

It has the advantage of saving several recalculations.

However, linearity is a restriction, as computer graphics also has non-linear behavior, e.g. perspective projection.



https://webglfundamentals.org/webgl/lessons/webgl-3d-perspective.html

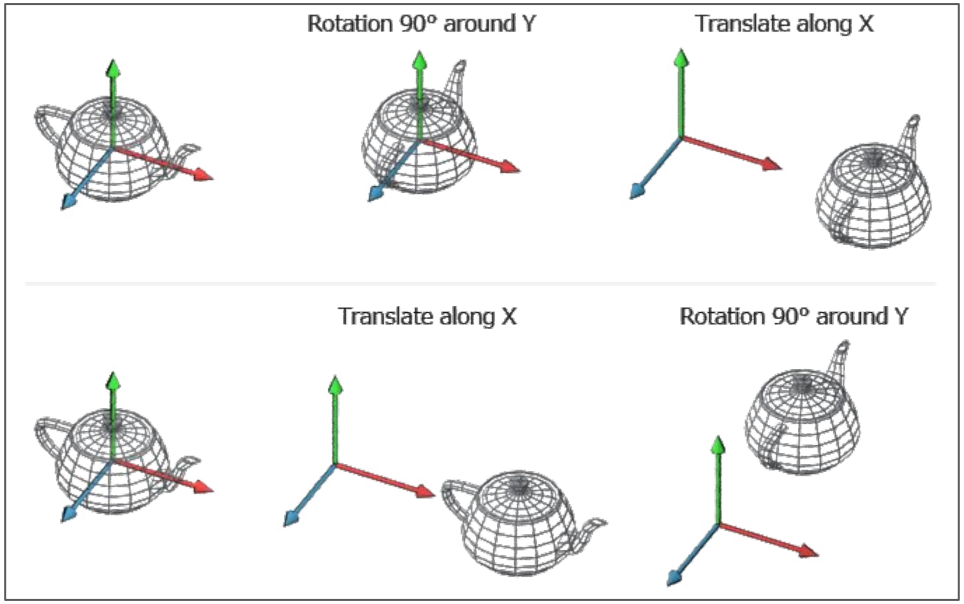
Affine Transformations

An **affine transformation** is equivalent to a linear transformation, but combined with vector addition.

$$f(\mathbf{u}) = \mathbf{v} = T(\mathbf{u}) + \mathbf{w}$$

for $f : X \longrightarrow Y$ for affine spaces, X and Y, such that $\mathbf{u} \in X$ and $\mathbf{w} \in Y$.

Model transformations fall under the class of affine transformations.



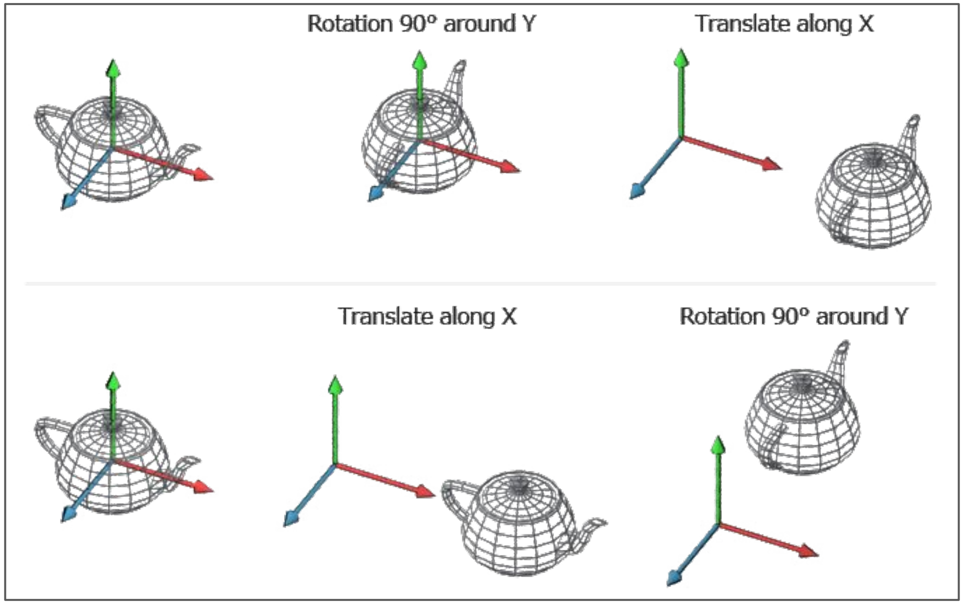
http://www.codinglabs.net/article_world_view_projection_matrix.aspx

Affine Transformations

Properties of Affine Transformations:

- Parallel lines remain parallel.
- The midpoint of a line segment remains a midpoint.
- All points on a straight line remain on a straight line.

Model transformations fall under the class of affine transformations.



http://www.codinglabs.net/article_world_view_projection_matrix.aspx

Affine Transformation of Lines

Affine transformations preserve lines.

For a line, $P(\alpha) = P_0 + \alpha\mathbf{d}$, affine transformation gives a line

$$Q(\alpha)=C(P(\alpha))=C(P_0)+\alpha C(\mathbf{d})$$

Similarly, affine transformation of line-segment gives

$$Q(\alpha)=C(P(\alpha))=(1-\alpha)C(P_0)+\alpha C(P_1)$$

Significance: OpenGL implements *affine transformations of end-points/vertices* to determine the *transformed line.*

Affine Transformations in Computer Graphics

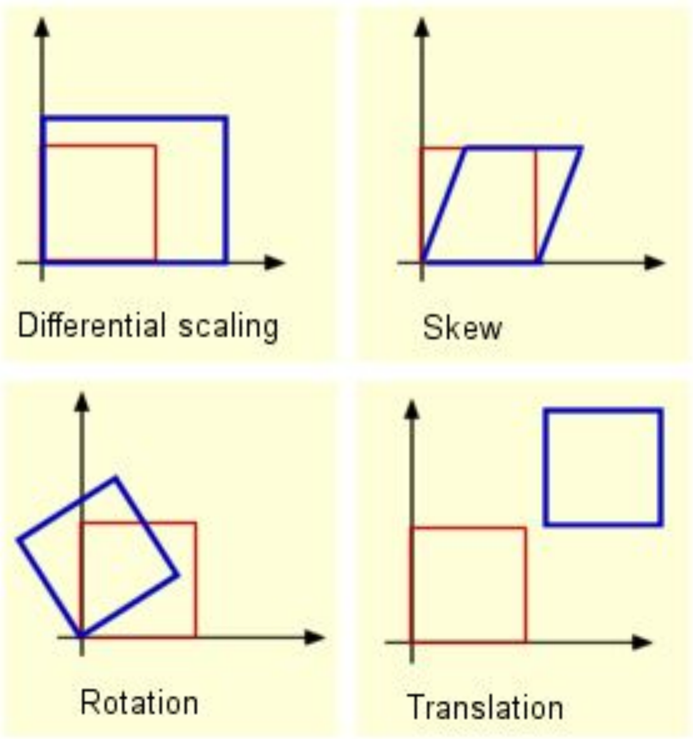
Most of the transformations in Computer Graphics are affine.

- Basic types of affine transformations include translation, rotation, scaling, shear.
- All affine transformations can be constructed as sequence of basic types.

Rigid-body transformations: Transformations with no alteration to shape or volume of object. e.g., Translation, Rotation.

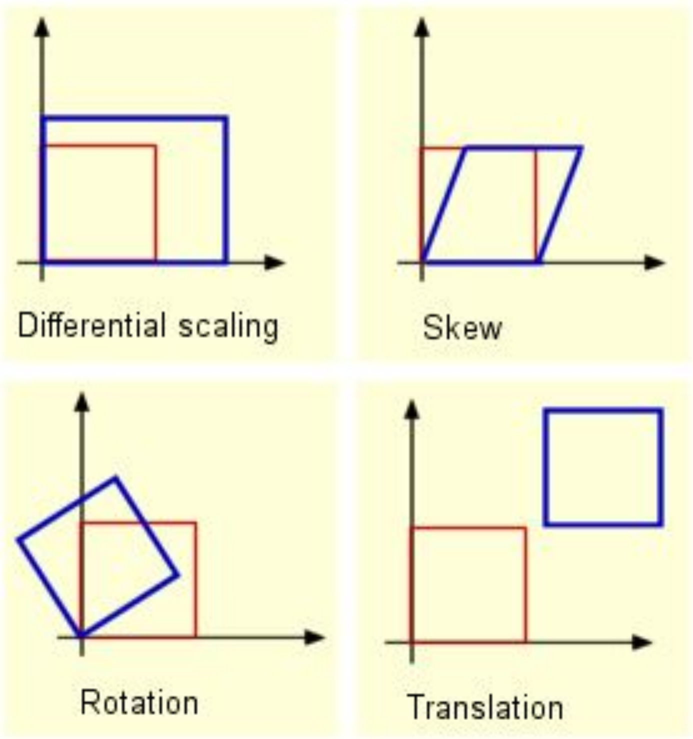
Non-rigid-body transformations: Transformations that alter shape or volume of an object. e.g., Uniform & Non-uniform scaling, Shear.

Affine Transformations



(Image courtesy: OpenGlobe).

Affine Transformations



(Image courtesy: OpenGlobe).

Scaling: requires fixed point, and a scale factor α.

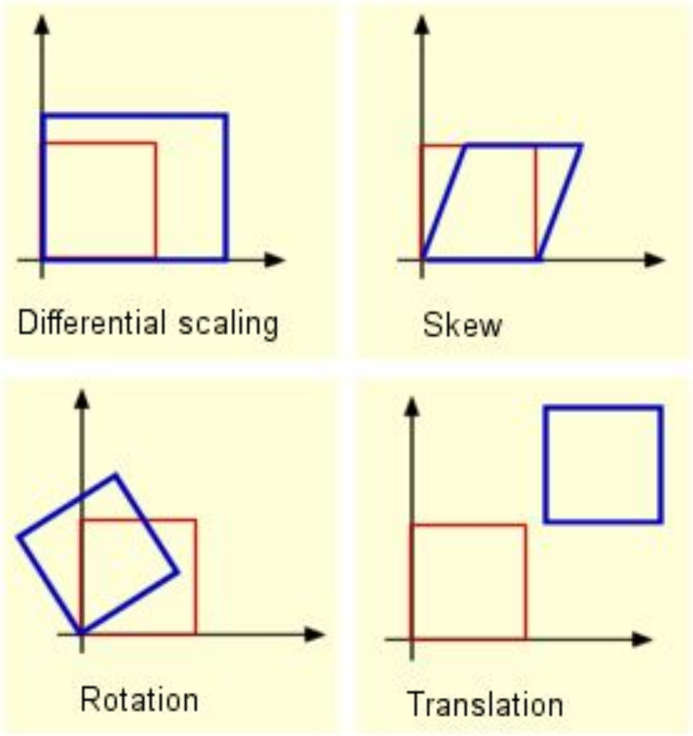
For a specific direction of scaling.

- Normal scaling: $\alpha \geq 0$
- Reflection: $\alpha < 0$

For a specific size of scaling:

- Expansion: $|\alpha| > 1$
- Identity: $|\alpha| = 1$
- Contraction: $0 \leq |\alpha| < 1$

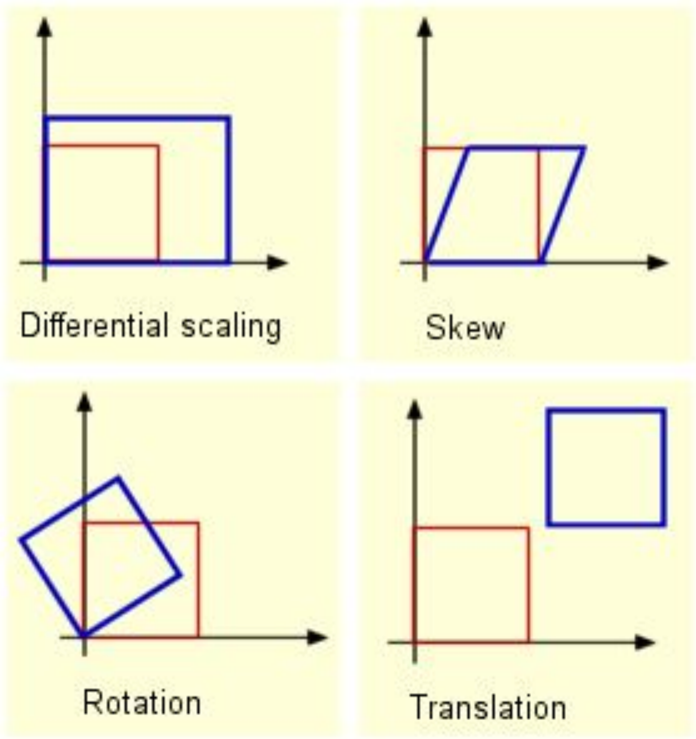
Affine Transformations



Translation: P' = P + **d**

(Image courtesy: OpenGlobe).

Affine Transformations



(Image courtesy: OpenGlobe).

Rotation by a fixed angle about fixed point about a fixed axis:

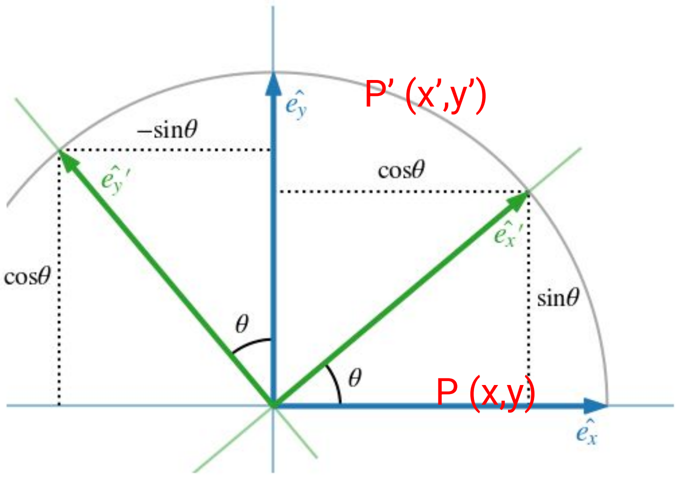
The fixed point is the point unchanged by rotation.

Two-dimensional rotation ≡ Three-dimensional rotation about the z-axis.

In OpenGL, the origin is always the **fixed point.**

Rotation

What happens when we rotate a coordinate system?



https://scipython.com/book/chapter-6-numpy/examples/creating-a-rotation-matrix-in-numpy/

Rotation

What happens when we rotate a coordinate system?

Applying the same concept to rotating a point -- analogous to a vertex in graphics.

Since rotations are linear transformations, the effect of rotating a vector from the origin to some arbitrary point, $P = (x, y)$, can be established by considering the rotation of the basis vectors $\hat{\boldsymbol{e}}_x \equiv (1, 0)$ and $\hat{\boldsymbol{e}}_y \equiv (0, 1)$. In the figure below, a rotation by θ takes

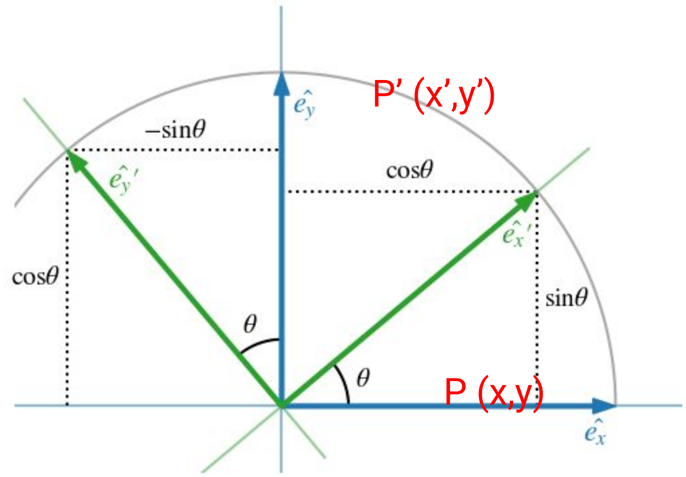
$$\begin{aligned}\hat{\boldsymbol{e}}_x \rightarrow \hat{\boldsymbol{e}}_x' &= \cos\theta\hat{\boldsymbol{e}}_x + \sin\theta\hat{\boldsymbol{e}}_y,\\ \hat{\boldsymbol{e}}_y \rightarrow \hat{\boldsymbol{e}}_y' &= -\sin\theta\hat{\boldsymbol{e}}_x + \cos\theta\hat{\boldsymbol{e}}_y.\end{aligned}$$

Our point P is therefore transformed from $(x, y) \equiv x\hat{\boldsymbol{e}}_x + y\hat{\boldsymbol{e}}_y$ to:

$$P' = x\hat{\boldsymbol{e}}_x' + y\hat{\boldsymbol{e}}_y' = (x\cos\theta - y\sin\theta)\hat{\boldsymbol{e}}_x + (x\sin\theta + y\sin\theta)\hat{\boldsymbol{e}}_y.$$

That is,

$$P' = \mathbf{R}P = \begin{pmatrix}\cos\theta & -\sin\theta\\ \sin\theta & \cos\theta\end{pmatrix}\begin{pmatrix}x\\ y\end{pmatrix}.$$



https://scipython.com/book/chapter-6-numpy/examples/creating-a-rotation-matrix-in-numpy/

Rotation

What happens when we rotate a coordinate system?

Applying the same concept to rotating a point -- analogous to a vertex in graphics.

Since rotations are linear transformations, the effect of rotating a vector from the origin to some arbitrary point, $P = (x, y)$, can be established by considering the rotation of the basis vectors $\hat{\boldsymbol{e}}_x \equiv (1, 0)$ and $\hat{\boldsymbol{e}}_y \equiv (0, 1)$. In the figure below, a rotation by θ takes

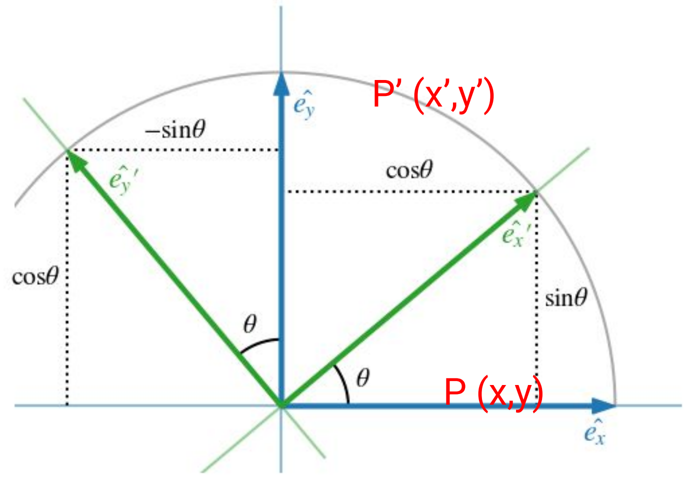
$$\hat{\boldsymbol{e}}_x \rightarrow \hat{\boldsymbol{e}}'_x = \cos\theta\hat{\boldsymbol{e}}_x + \sin\theta\hat{\boldsymbol{e}}_y,$$
$$\hat{\boldsymbol{e}}_y \rightarrow \hat{\boldsymbol{e}}'_y = -\sin\theta\hat{\boldsymbol{e}}_x + \cos\theta\hat{\boldsymbol{e}}_y.$$

Our point P is therefore transformed from $(x, y) \equiv x\hat{\boldsymbol{e}}_x + y\hat{\boldsymbol{e}}_y$ to:

$$P' = x\hat{\boldsymbol{e}}'_x + y\hat{\boldsymbol{e}}'_y = (x\cos\theta - y\sin\theta)\hat{\boldsymbol{e}}_x + (x\sin\theta + y\sin\theta)\hat{\boldsymbol{e}}_y.$$

That is,

$$P' = \mathbf{R}P = \boxed{\begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}} \begin{pmatrix} x \\ y \end{pmatrix}.$$



https://scipython.com/book/chapter-6-numpy/examples/creating-a-rotation-matrix-in-numpy/

Rotation Matrix

Extending 2D matrices to 3D

Rotation in 3D

Rotation around coordinate axes

$$\mathbf{R}_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$

$$\mathbf{R}_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

$$\mathbf{R}_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

http://ivl.calit2.net/wiki/images/8/8c/02_TransformationsF12.pdf

Rotation Matrix

Extending 2D matrices to 3D

Rotation in 3D

- Concatenation of rotations around x, y, z axes

$$\mathbf{R}_{x,y,z}(\theta_x, \theta_y, \theta_z) = \mathbf{R}_x(\theta_x)\mathbf{R}_y(\theta_y)\mathbf{R}_z(\theta_z)$$

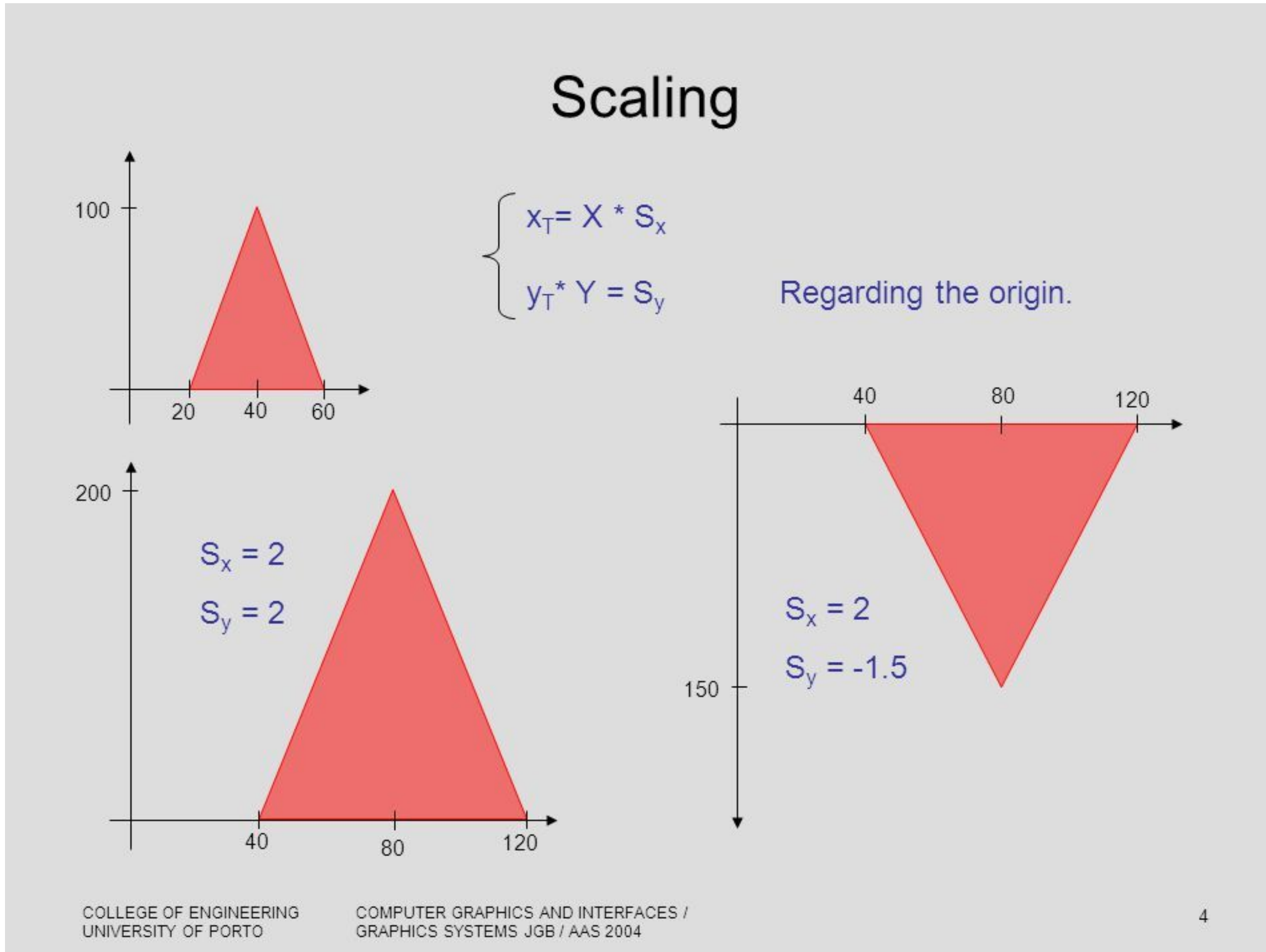
- $\theta_x, \theta_y, \theta_z$ are called Euler angles
- Result depends on matrix order!

$$\mathbf{R}_x(\theta_x)\mathbf{R}_y(\theta_y)\mathbf{R}_z(\theta_z) \neq \mathbf{R}_z(\theta_z)\mathbf{R}_y(\theta_y)\mathbf{R}_x(\theta_x)$$

http://ivl.calit2.net/wiki/images/8/8c/02_TransformationsF12.pdf

Scaling

An example.



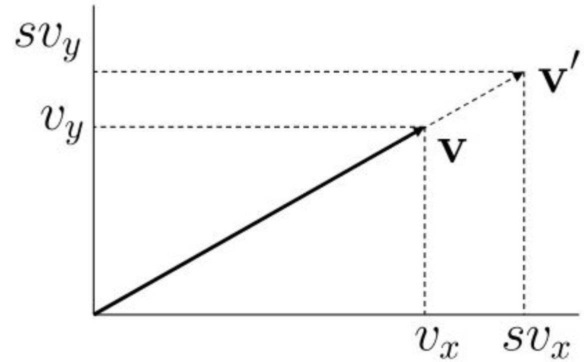
https://slideplayer.com/slide/3269641/

Scaling Matrix

Uniform or non-differential scaling:
Same scaling factor along all principal axes

Scaling

- Uniform scaling matrix in 2D



- Analogous in 3D

$$\begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix} \mathbf{v} = \begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix} \begin{bmatrix} v_x \\ v_y \end{bmatrix} = \begin{bmatrix} v'_x \\ v'_y \end{bmatrix} = \mathbf{v}'$$

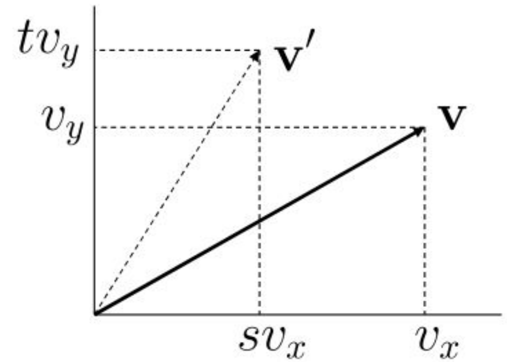
http://ivl.calit2.net/wiki/images/8/8c/02_TransformationsF12.pdf

Scaling Matrix

Uniform or non-differential scaling:
Same scaling factor along all principal axes

Scaling

- Nonuniform scaling matrix in 2D



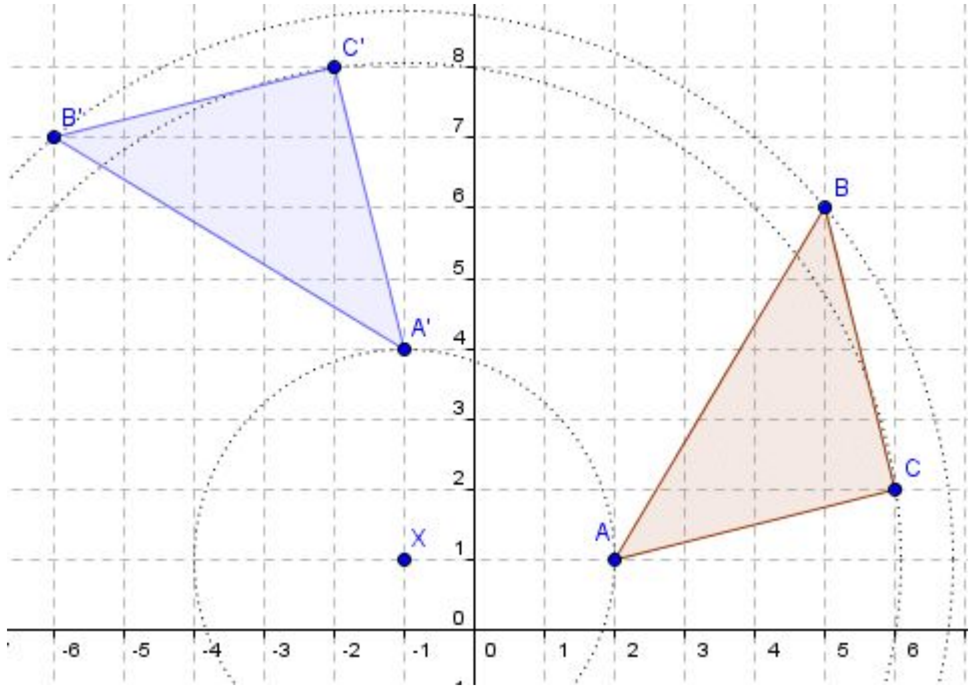
- Analogous in 3D

$$\begin{bmatrix} s & 0 \\ 0 & t \end{bmatrix} \mathbf{v} = \begin{bmatrix} s & 0 \\ 0 & t \end{bmatrix} \begin{bmatrix} v_x \\ v_y \end{bmatrix} = \begin{bmatrix} v'_x \\ v'_y \end{bmatrix} = \mathbf{v}'$$

http://ivl.calit2.net/wiki/images/8/8c/02_TransformationsF12.pdf

Rotation & Scaling

These transformations involve a **fixed point**, which is the point that does not transform.



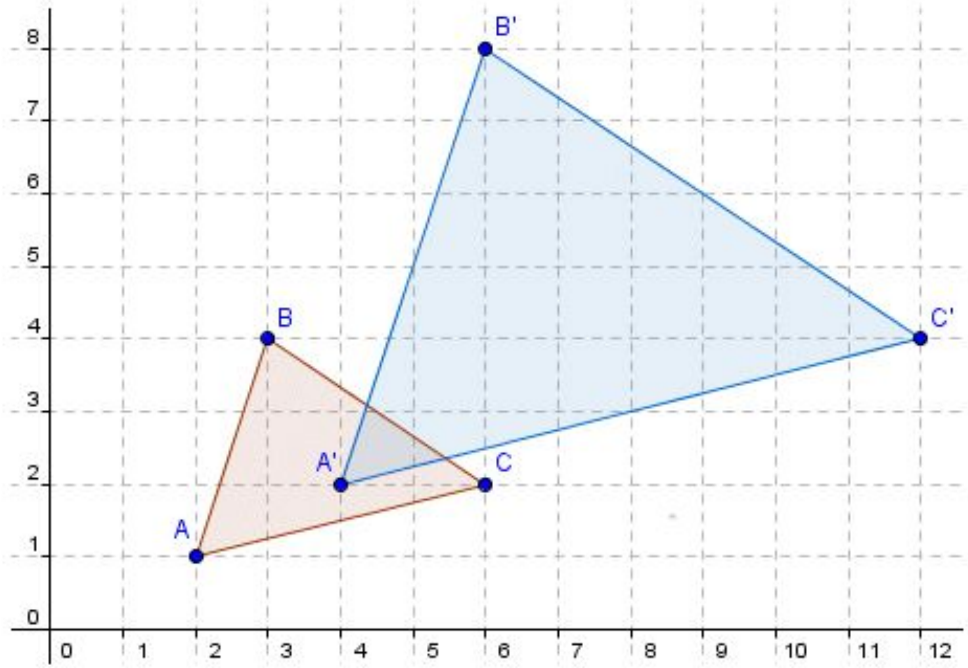
X is a fixed point for rotation, also called “center of rotation”.

Rotation is thus defined by the center, radius, and axis of rotation.

http://www.technologyuk.net/mathematics/geometry/transformations.shtml

Rotation & Scaling

These transformations involve a **fixed point**, which is the point that does not transform.



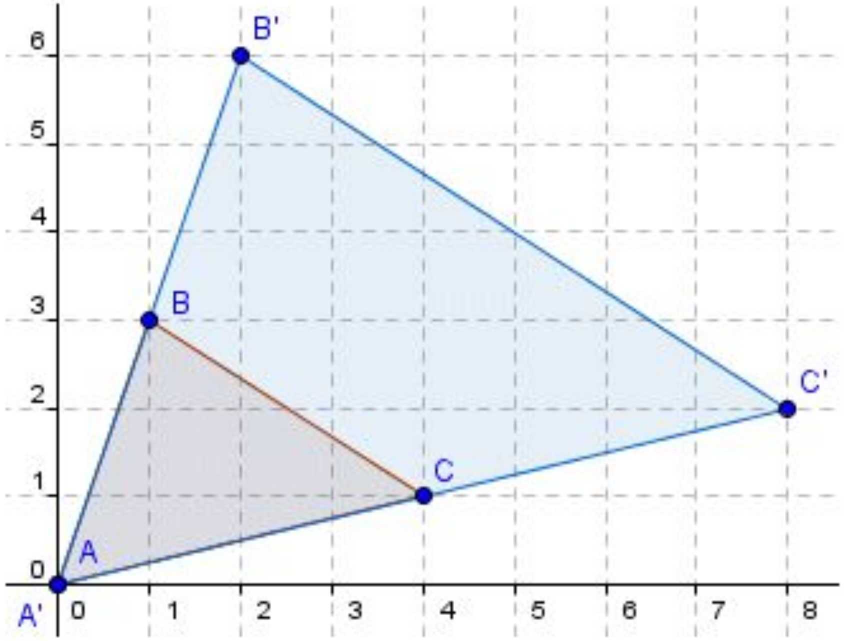
(0,0) is a fixed point for scaling here.

Scaling is thus defined by the fixed point and scaling factors.

http://www.technologyuk.net/mathematics/geometry/transformations.shtml

Rotation & Scaling

These transformations involve a **fixed point**, which is the point that does not transform.



(0,0) is a fixed point for scaling here.

Scaling is thus defined by the fixed point and scaling factors.

http://www.technologyuk.net/mathematics/geometry/transformations.shtml

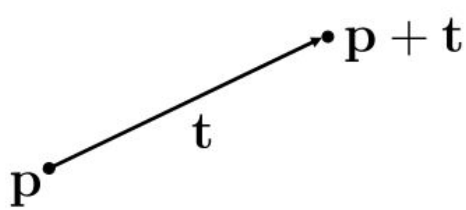
Translation

Translation is an affine transformation, but not a linear transformation.

Hence it cannot be represented in the form of matrices.

Translation

Using homogeneous coordinates



$$\mathbf{p} = \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} \qquad \mathbf{t} = \begin{bmatrix} t_x \\ t \\ \imath \\ 0 \end{bmatrix} \qquad \mathbf{p} + \mathbf{t} \quad \begin{bmatrix} p_x + t_x \\ p_y + t_y \\ p_z + t_z \\ 1 \end{bmatrix}$$

http://ivl.calit2.net/wiki/images/8/8c/02_TransformationsF12.pdf

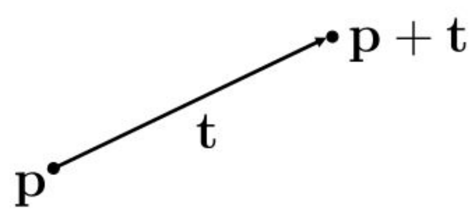
Translation

How to make a translation matrix?

Increase dimensionality of coordinate system by 1 to create **Homogeneous Coordinate System.**

Translation

Using homogeneous coordinates



$$\mathbf{p} = \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} \qquad \mathbf{t} = \begin{bmatrix} t_x \\ t \\ \imath \\ 0 \end{bmatrix} \qquad \mathbf{p} + \mathbf{t} \quad \begin{bmatrix} p_x + t_x \\ p_y + t_y \\ p_z + t_z \\ 1 \end{bmatrix}$$

http://ivl.calit2.net/wiki/images/8/8c/02_TransformationsF12.pdf

Translation

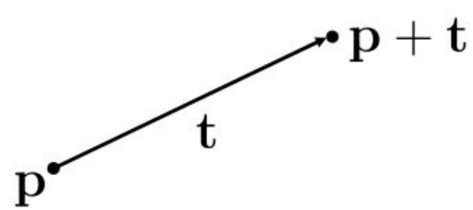
How to make a translation matrix?

Increase dimensionality of coordinate system by 1 to create **Homogeneous Coordinate System.**

Translation

Using homogeneous coordinates

Homogeneous Coordinate System is to be understood as an ***alternative*** representation of the Cartesian coordinate system.



$$\mathbf{p} = \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} \qquad \mathbf{t} = \begin{bmatrix} t_x \\ t \\ \imath \\ 0 \end{bmatrix} \qquad \mathbf{p} + \mathbf{t} \quad \begin{bmatrix} p_x + t_x \\ p_y + t_y \\ p_z + t_z \\ 1 \end{bmatrix}$$

http://ivl.calit2.net/wiki/images/8/8c/02_TransformationsF12.pdf

Summary: 2D Affine Transformations

Rotation by an angle θ about +z axis, scaling, translation.

$$Rotation: \begin{bmatrix} x \\ y \end{bmatrix} \rightarrow \begin{bmatrix} x\cos\theta - y\sin\theta \\ x\sin\theta + y\cos\theta \end{bmatrix}; \Rightarrow M_{R_z} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

$$Scaling: \begin{bmatrix} x \\ y \end{bmatrix} \rightarrow \begin{bmatrix} k_x x \\ k_y y \end{bmatrix}; \Rightarrow M_S = \begin{bmatrix} k_x & 0 \\ 0 & k_y \end{bmatrix}$$

$$Translation: \begin{bmatrix} x \\ y \end{bmatrix} \rightarrow \begin{bmatrix} x + t_x \\ y + t_y \end{bmatrix}; \Rightarrow M_T = ?$$

Rotation and scaling can be represented as (matrix representations of) linear transformations, but translation cannot be represented such.

Summary: Homogeneous Coordinates for 2D Affine Transformations

Rotation by an angle θ about +z axis, scaling, translation.

$$\mathrm{R}: \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} x\cos\theta - y\sin\theta \\ x\sin\theta + y\cos\theta \\ 1 \end{bmatrix}; \Rightarrow M_{R_z} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathrm{S}: \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} k_x x \\ k_y y \\ 1 \end{bmatrix}; \Rightarrow M_S = \begin{bmatrix} k_x & 0 & 0 \\ 0 & k_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathrm{T}: \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} x + t_x \\ y + t_y \\ 1 \end{bmatrix}; \Rightarrow M_T = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

Modeling-Rendering Paradigm

Recap - Ingredients in Graphics Programming

Graphics programming is the main part of the rendering module.

Ingredients:

1. Objects -- output of the modeling module, geometry (3D, complex, etc.)
2. A viewer (or camera)
3. Light sources (important for 3D realism)
4. Material properties (needed for rendering)

[In-class demo of Nate Robins OpenGL Tutorial
http://user.xmission.com/~nate/tutors.html]

Object Transformation

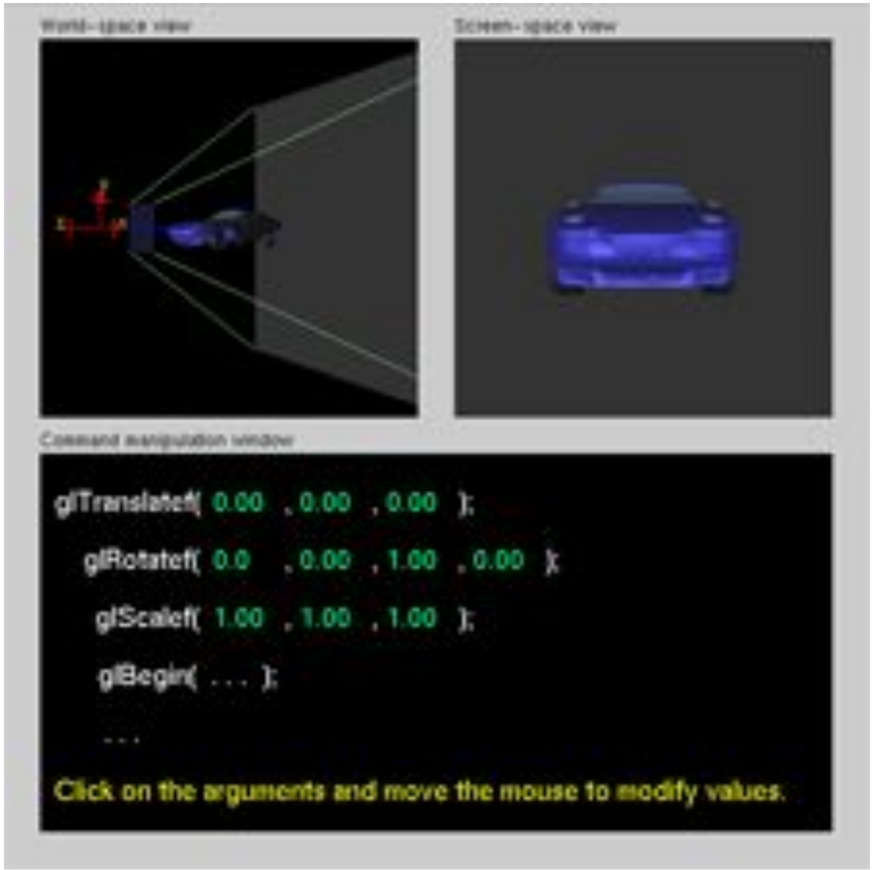


Image courtesy: Nate Robins

Object Projection

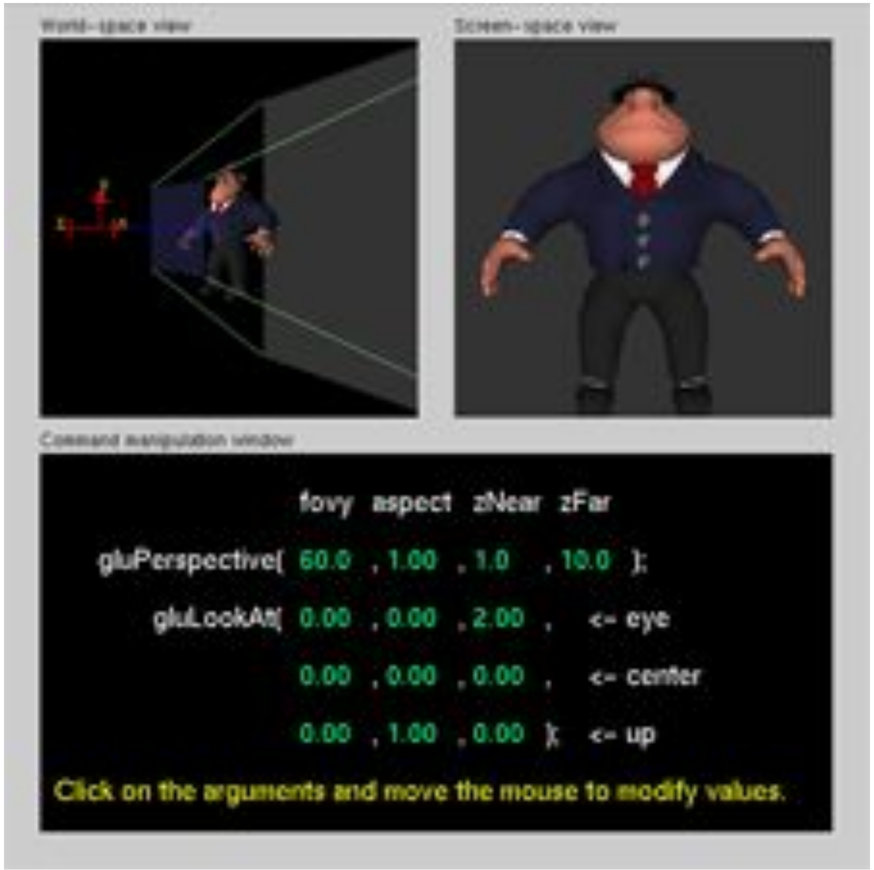


Image courtesy: Nate Robins

Lighting

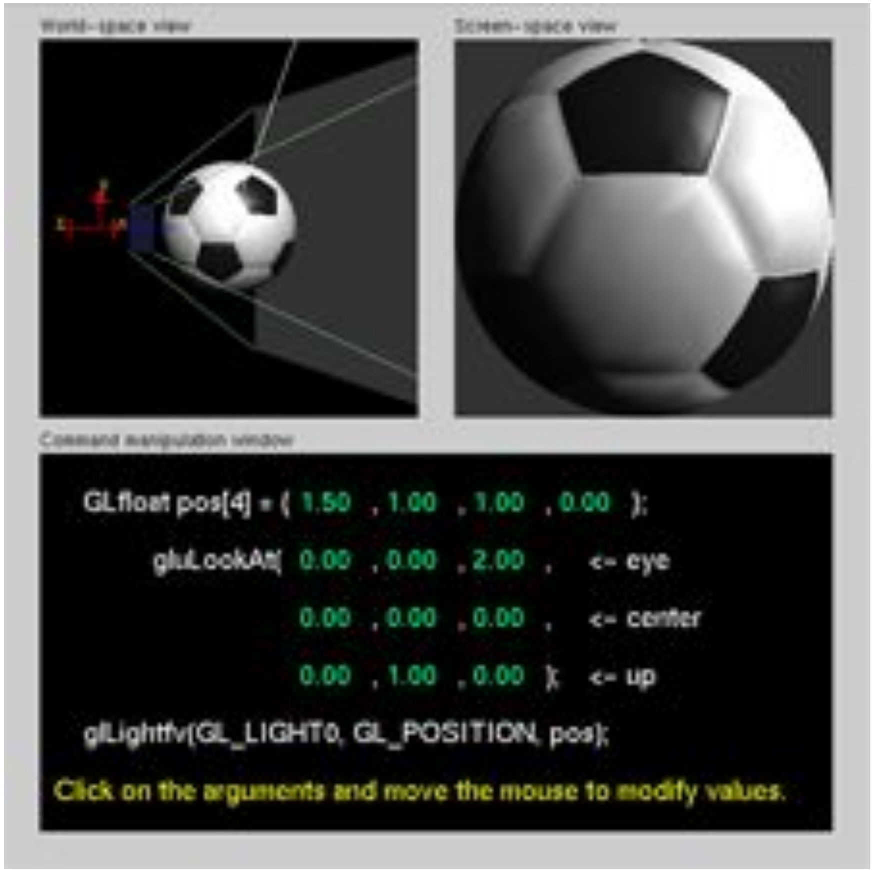


Image courtesy: Nate Robins

Material Properties

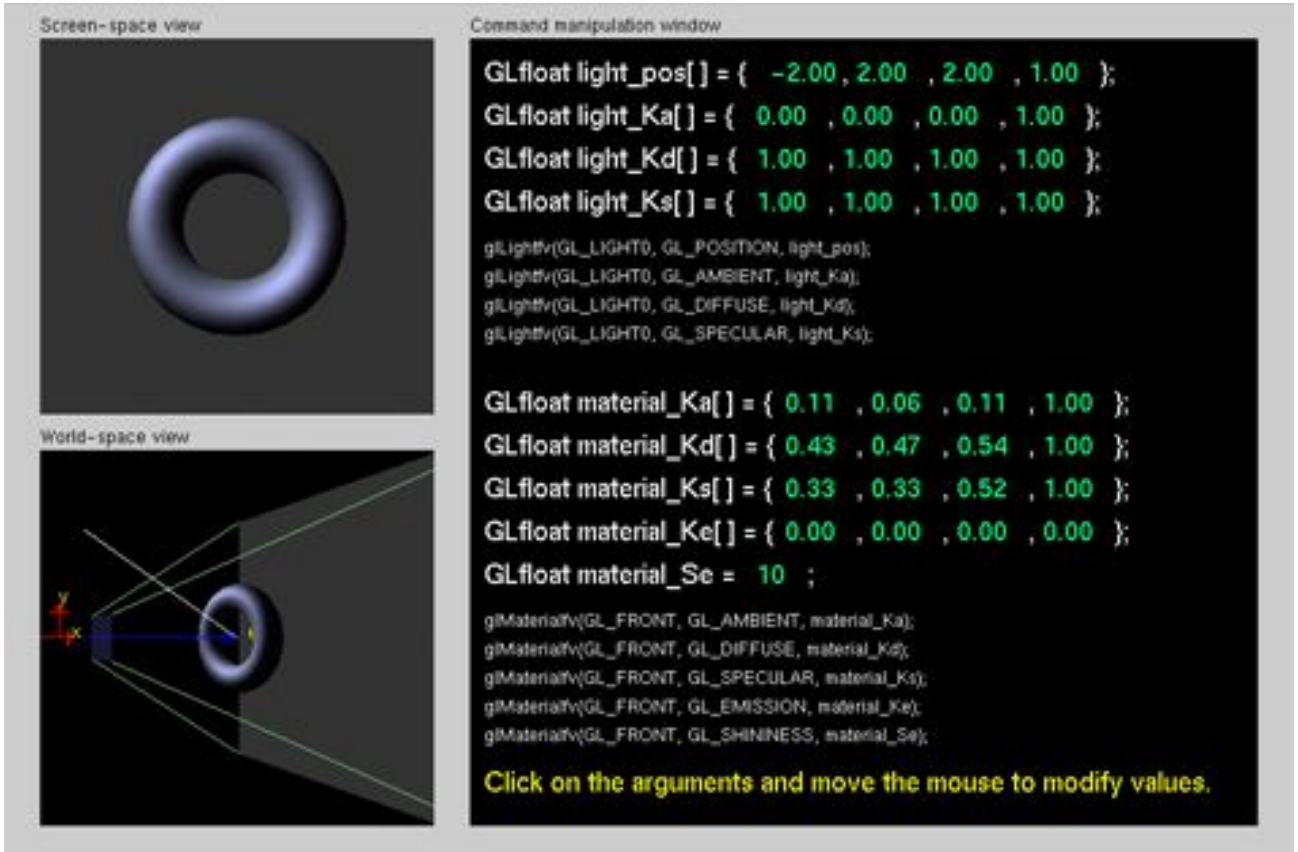


Image courtesy: Nate Robins

Summary

- Affine transformations
  - Its significance in computer graphics
  - Posing them as linear transformations for efficiency, through the use of homogeneous coordinates
  - Rotation, scaling, translation
- Modeling-rendering paradigm
  - Examples from OpenGL1.0 implementation in Nate Robins tutorials
  - http://user.xmission.com/~nate/tutors.html