GDC 09

learn
network
inspire

www.GDConf.com

Game Developers Conference®

March 23-27, 2009 | Moscone Center, San Francisco

Affine Transformations

Jim Van Verth
NVIDIA Corporation
(jim@essentialmath.com)

Topics

» What's an affine transformation?
» How to generate various forms
» Things to watch out for

What is it?

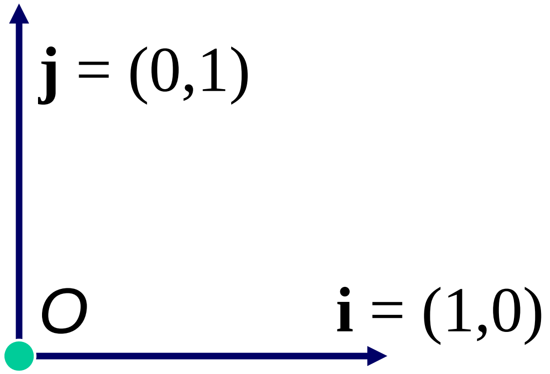
- A mapping between affine spaces
- Preserves lines (& planes)
- Preserves parallel lines
- But not angles or distances
- Can represent as

$$T(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{y}$$

- (note we're using column vectors)

Affine Space

- Collection of points and vectors
- Can be represented using *frame*



- Within frame, vector and point

$$\mathbf{v} = x\mathbf{i} + y\mathbf{j}$$

$$\mathbf{x} = x\mathbf{i} + y\mathbf{j} + O$$

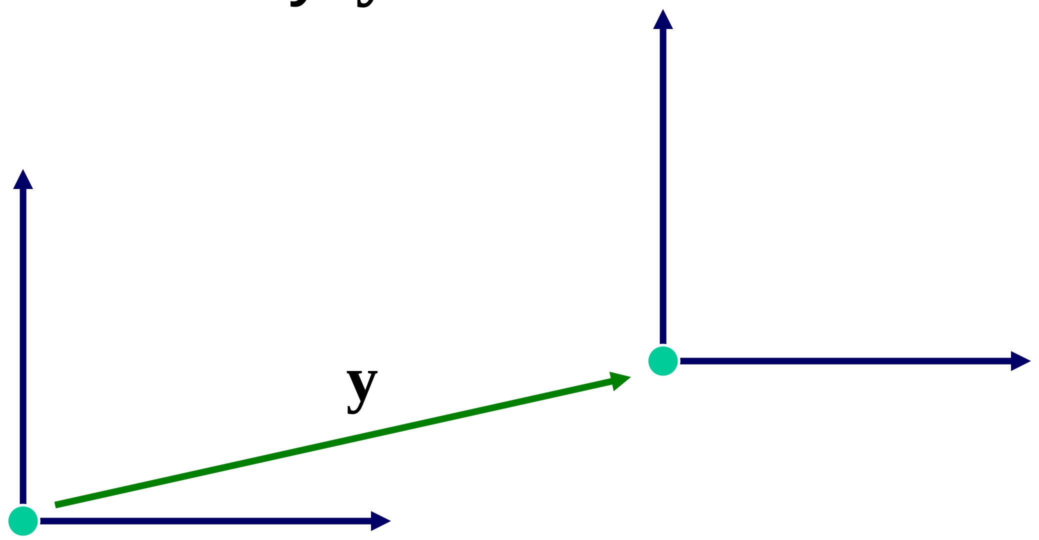
Affine Transformation

- Key idea: map from space to space by using frames
- Note how axes change ($\mathbf{A}$)
- Note how origin changes ($\mathbf{y}$)

$$T(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{y}$$

Example: Translation

» Axes don't change
» Origin moves by $\mathbf{y}$

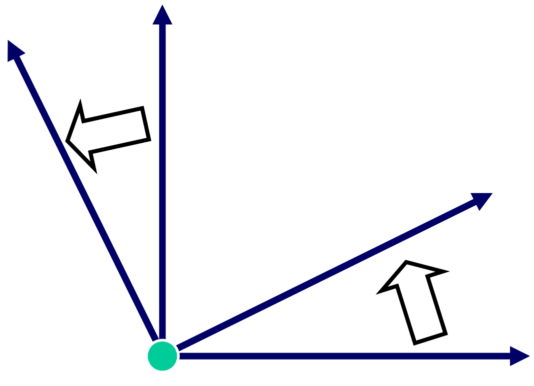


$$T(\mathbf{x}) = \mathbf{x} + \mathbf{y}$$

Example: Rotation

» Axes change

» Origin doesn't move

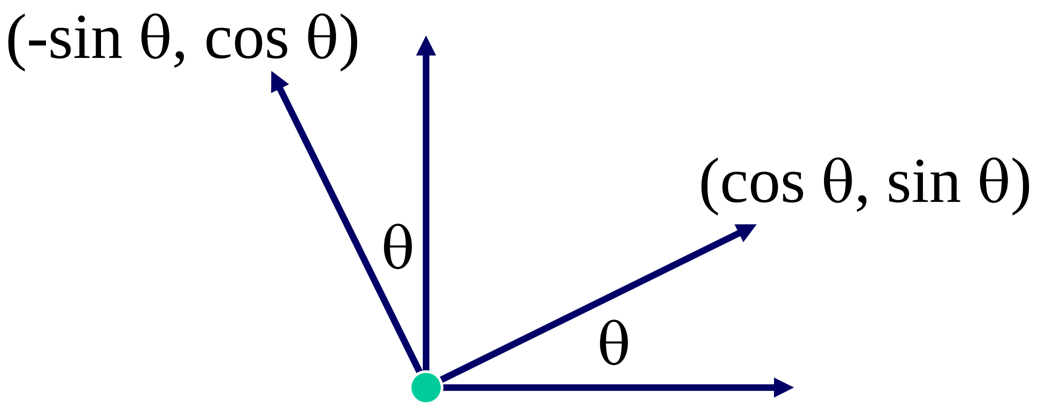


$$T(\mathbf{x}) = \mathbf{A}\mathbf{x}$$

» But what is $\mathbf{A}$?

Example: Rotation

» Follow the axes:

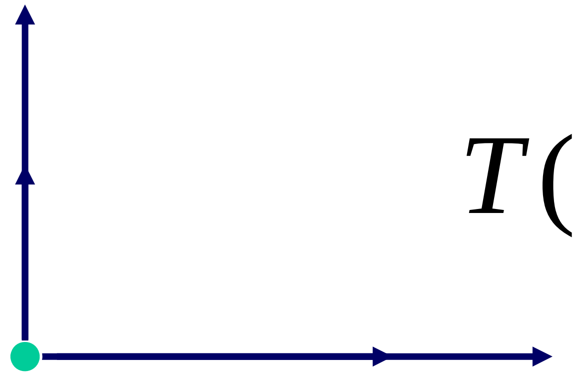


» New axes go in columns of matrix

$$\mathbf{A} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

Example: Scale

- Axes change
- Origin doesn't move

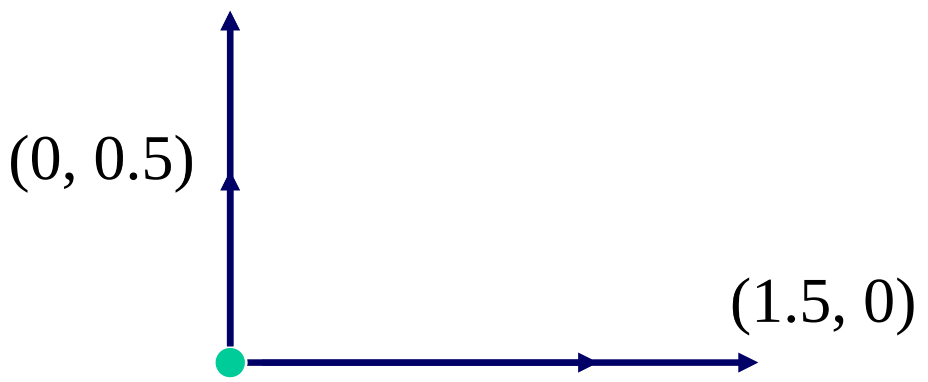


$$T(\mathbf{x}) = \mathbf{A}\mathbf{x}$$

- Again, what is $\mathbf{A}$?

Example: Scale

» Follow the axes:

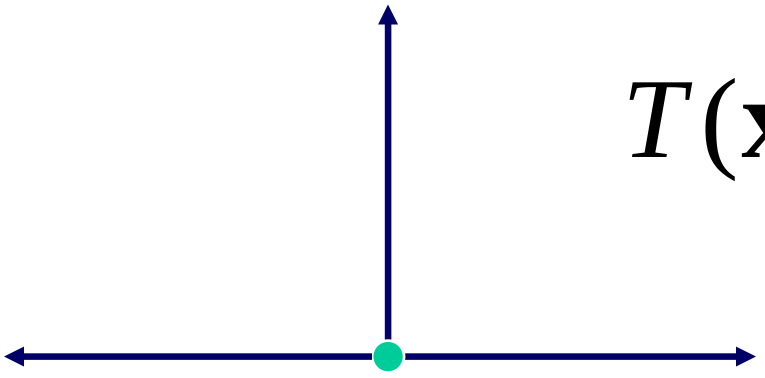


» New axes go in columns of matrix

$$\mathbf{A} = \begin{bmatrix} 1.5 & 0 \\ 0 & 0.5 \end{bmatrix}$$

Example: Reflection

- Axes change
- Origin doesn't move

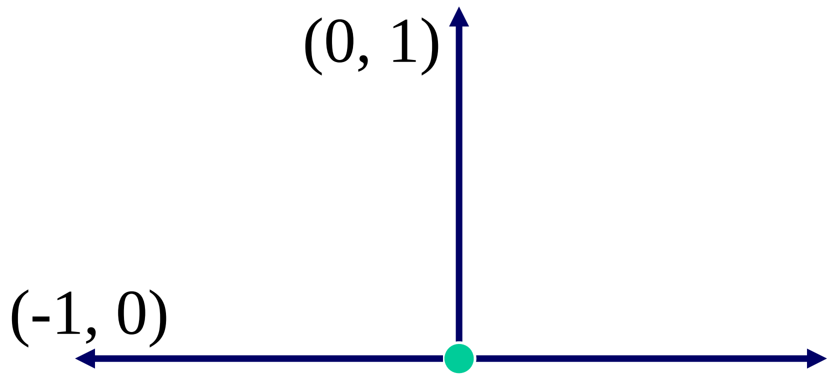


$$T(\mathbf{x}) = \mathbf{A}\mathbf{x}$$

- But what, oh what, is $\mathbf{A}$?

Example: Reflection

» Follow the axes:

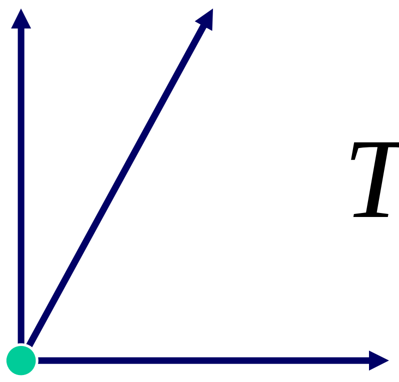


» New axes go in columns of matrix

$$\mathbf{A} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$$

Example: Shear

- Axes change
- Origin doesn't move

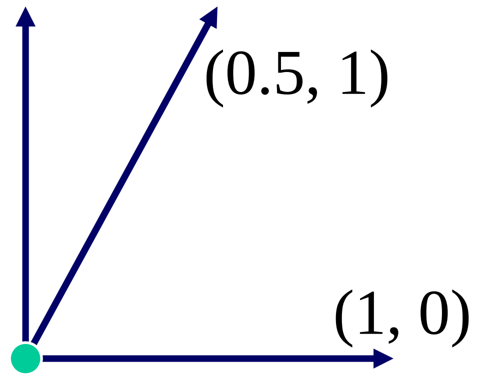


$$T(\mathbf{x}) = \mathbf{A}\mathbf{x}$$

- Hey, hey, what is $\mathbf{A}$?

Example: Shear

» Follow the axes:



» New axes go in columns of matrix

$$\mathbf{A} = \begin{bmatrix} 1 & 0.5 \\ 0 & 1 \end{bmatrix}$$

Transform Types

- Rigid-body transformation
  - Translation
  - Rotation
- Deformable transformation
  - Scale
  - Reflection
  - Shear

Combining Transforms

- Simple function composition

$$T(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{y}$$

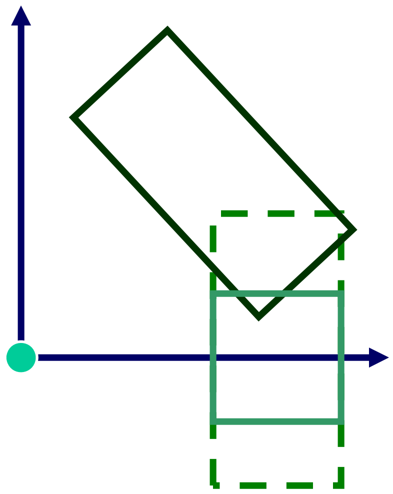
$$S(\mathbf{w}) = \mathbf{B}\mathbf{w} + \mathbf{z}$$

$$\begin{aligned} S(T(\mathbf{x})) &= \mathbf{B}(\mathbf{A}\mathbf{x} + \mathbf{y}) + \mathbf{z} \\ &= \mathbf{B}\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{y} + \mathbf{z} \end{aligned}$$

Combining Transforms

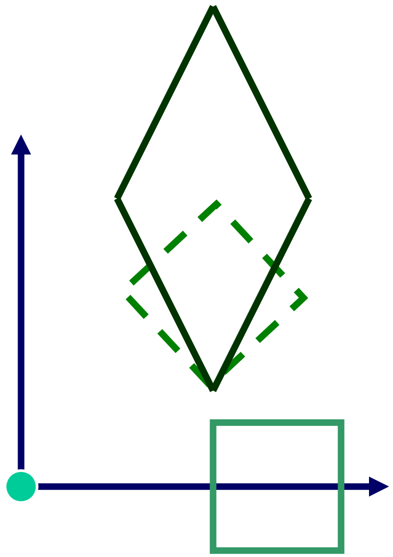
» Order is important!

» Scale, then rotate



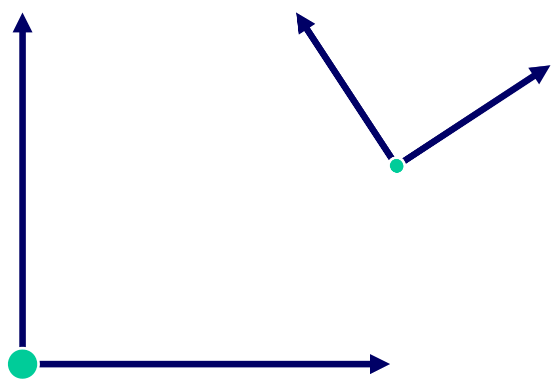
Combining Transforms

- Order is important!
- Rotate, then scale



Local and World Frames

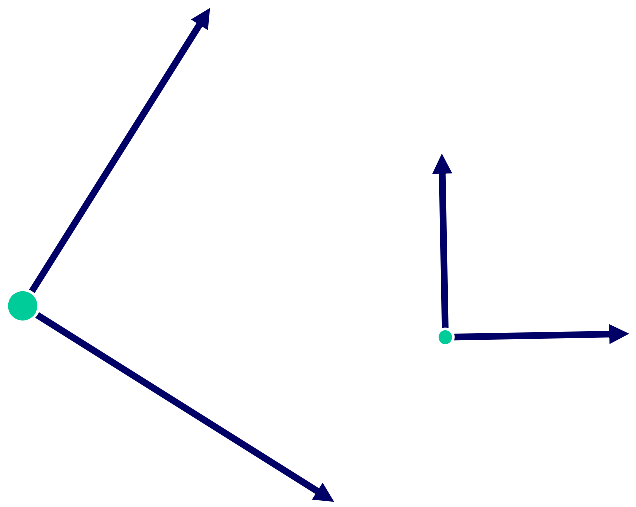
- Objects built in local frame
- Want to place in world frame
- Local-to-world transformation



- Same basic idea: determine where axes and origin end up in *world frame*

World to Local Frame

- Similar, but in reverse
- Want transformation relative to local frame



- Often easier to just take the inverse
- Example: world-to-view transformation

Inverse

- Reverses the effect of a transformation
- Easy to do with formula:

$$\mathbf{Ax} + \mathbf{y} = \mathbf{z}$$

$$\mathbf{Ax} = \mathbf{z} - \mathbf{y}$$

$$\mathbf{x} = \mathbf{A}^{-1}(\mathbf{z} - \mathbf{y})$$

$$\mathbf{T}^{-1}(\mathbf{z}) = \mathbf{A}^{-1}\mathbf{z} - \mathbf{A}^{-1}\mathbf{y}$$

- For matrix, just use matrix inverse

Matrix Form

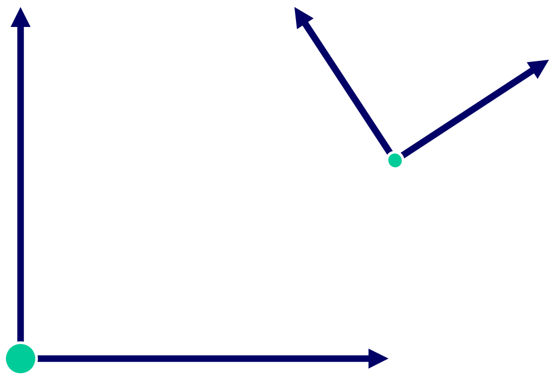
» Necessary for many APIs
» Is easy

$$\begin{bmatrix} \mathbf{A} & \mathbf{y} \\ \mathbf{0}^T & 1 \end{bmatrix}$$

» Concatenate by matrix multiply
» Can be faster on vector architectures
» Takes more storage, though

Object-centered Transform

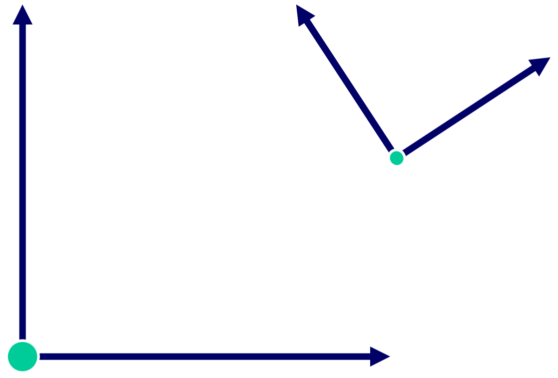
» Often get this case
» Already have local-to-world transform



» Want rotate/scale/whatever around local origin, not world origin
» How?

Object-Oriented Transform

- One way
- Translate to origin
- Rotate there
- Translate back



Object-Oriented Transform

- Using formula:
- Translate to origin

$$T(\mathbf{z}) = \mathbf{z} - \mathbf{y}$$

- Rotate there

$$S(T(\mathbf{z})) = \mathbf{B}(\mathbf{z} - \mathbf{y}) = \mathbf{Bz} - \mathbf{By}$$

- Translate back

$$R(S(T(\mathbf{z}))) = (\mathbf{Bz} - \mathbf{By}) + \mathbf{y} = \mathbf{Bz} + (\mathbf{I} - \mathbf{B})\mathbf{y}$$

- This works with arbitrary center in world frame!

Alternate Format

- Problem: want to
  - Translate object in space and change rotation and scale arbitrarily
  - Handle rotation/scale separately
- Can't do easily with matrix format or $\mathbf{Ax} + \mathbf{y}$ form
- Involves SVD, Polar decomposition
- Messy, but we can do better using...

Rigid Body Transforms

- Any sequence of translation and rotation transformations
- Not scale, reflection or shear
- Object shape is not affected (preserves angles and lengths)
- Usually include uniform scale despite this

Alternate Format

- Scale – one uniform scale factor s
- Rotation – matrix $\mathbf{R}$
- Translation – single vector $\mathbf{t}$

Alternate Format

» Want to concatenate transforms T_1, T_2 in this form, or

$$T = T_2 \circ T_1$$

» Do this by

$$s = s_2 s_1$$

$$\mathbf{R} = \mathbf{R}_2 \mathbf{R}_1$$

$$\mathbf{t} = \mathbf{t}_2 + s_2 \mathbf{R}_2 \mathbf{t}_1$$

Alternate Format

- Advantages
  - Clear what each part does
  - Easier to change individual elements
- Disadvantages
  - Eventually have to convert to 4x4 matrix anyway for renderer/video card
  - 4x4 faster on vector architecture

Alternate Format

» Matrix conversion

$$\begin{bmatrix} s\cdot r_{11} & s\cdot r_{12} & s\cdot r_{13} & t_x \\ s\cdot r_{21} & s\cdot r_{22} & s\cdot r_{23} & t_y \\ s\cdot r_{31} & s\cdot r_{32} & s\cdot r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Inverting Rigid Body Xforms

- We can easily invert our rigid body transforms symbolically:

$$\mathbf{y} = s\mathbf{R}\mathbf{x} + \mathbf{t}$$

$$\mathbf{y} - \mathbf{t} = s\mathbf{R}\mathbf{x}$$

$$\tfrac{1}{s}(\mathbf{y} - \mathbf{t}) = \mathbf{R}\mathbf{x}$$

$$\mathbf{R}^{\mathrm{T}} \tfrac{1}{s}(\mathbf{y} - \mathbf{t}) = \mathbf{R}^{\mathrm{T}}\mathbf{R}\mathbf{x}$$

$$\tfrac{1}{s}\mathbf{R}^{\mathrm{T}}\mathbf{y} - \tfrac{1}{s}\mathbf{R}^{\mathrm{T}}\mathbf{t} = \mathbf{x}$$

Inverting Rigid Body Xforms (2)

- In fact, the result itself may be written as a rigid body transform:

$$s^{-1} = \frac{1}{s}$$

$$\mathbf{R}^{-1} = \mathbf{R}^{\mathrm{T}}$$

$$\mathbf{t}^{-1} = -\tfrac{1}{s}\mathbf{R}^{\mathrm{T}}\mathbf{t}$$

References

- Van Verth, James M. and Lars M. Bishop, *Essential Mathematics for Games and Interactive Applications*, 2nd Ed, Morgan Kaufmann, 2008.
- Rogers, F. David and J. Alan Adams, *Mathematical Elements for Computer Graphics,* 2nd Ed, McGraw-Hill, 1990.
- Watt, Alan, *3D Computer Graphics*, Addison-Wesley, Wokingham, England, 1993.

GDC 09
learn network inspire
www.GDConf.com