Lecture 03:
2D Transformations

September 3, 2019

Previous Lecture & Today

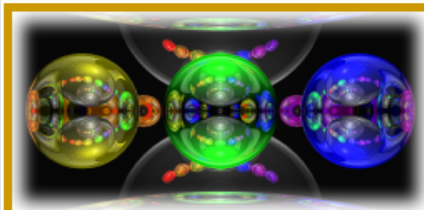
- Last Thursday
  - Scalars, Vectors and Points
  - Vector Spaces
  - Affine Spaces
  - Euclidean Spaces
  - Know and love the dot product
- Today
  - Projection (dot product) and rotation,
  - Homogeneous Coordinates for 2D

About SageMath

Getting Notebooks

localhost/courses/cs410/yr2019fa/home_resources

CS 410 : Resources



Introduction to Computer Graphics

Fall 2019

CS 410 : Resources

Home | Syllabus | Progress | Assignments | Resources | Canvas

For the most part this page is used to distribute SageMath notebooks used to illustrate key concepts. Students must during the course of the semester download and actually run these notebooks. It is not enough to passivly review them, it is essential to get into the examples, make changes, and learn from interaction.

There are links to the 'dead' HTML output of notebooks below. This is useful for a quick read through. However, to actually run the notebook, you must download the IPYNB file and upload it to a SageMath Jupyter Server. Server software is installed on the CS Department Machines (instructions coming soon) and you may also download SageMath to your own machine.

Lecture 3 Notebook 1
Notebook cs410lec03n01 illustrates the dot product as a projection operator in 2D. In other words, measuring the distance from the origin to a point in the direction of a vector u.
Zipped IPYNB Notebook File

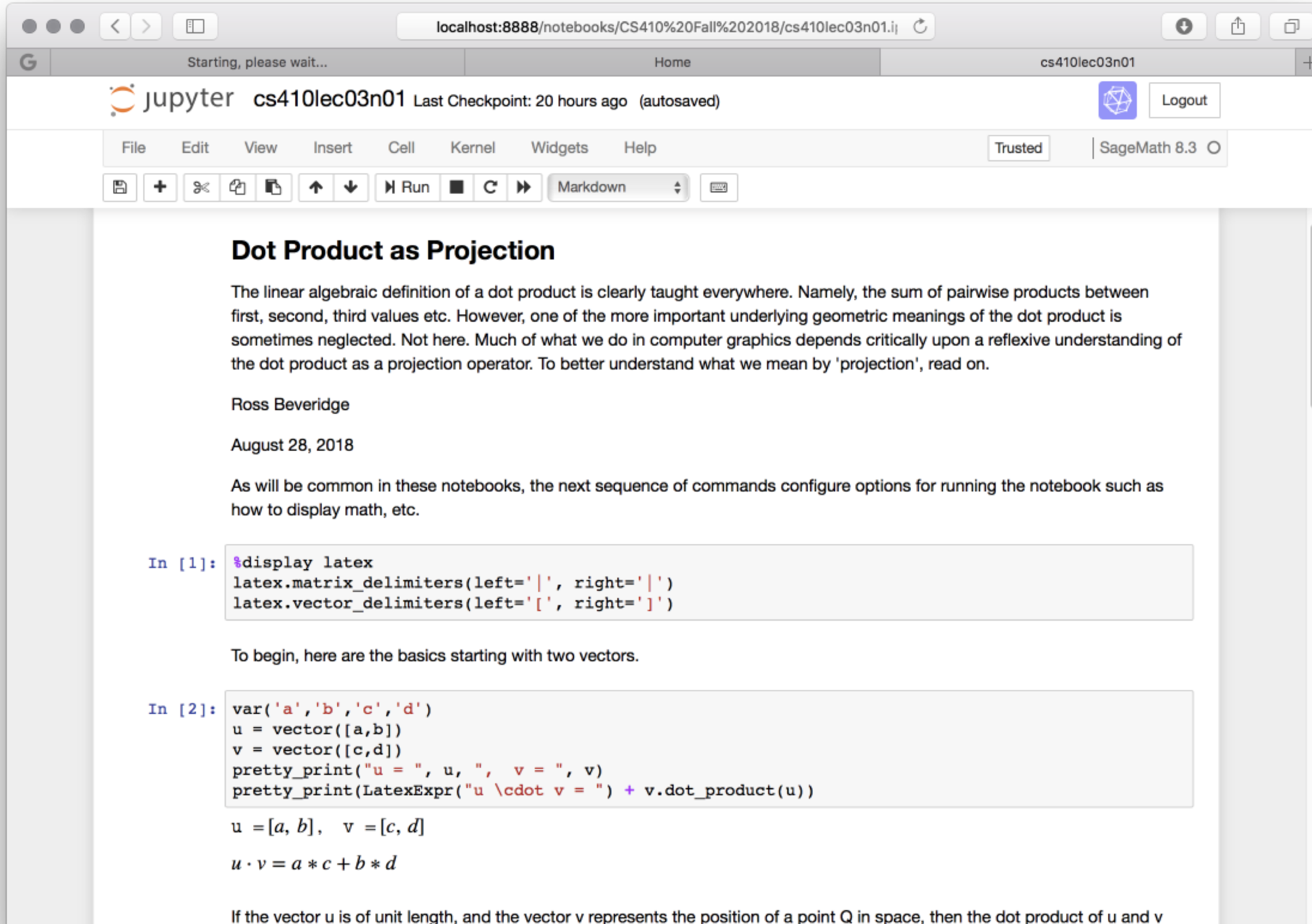
Lecture 3 Notebook 2
Notebook cs410lec03n02 builds upon the previous notebood carrying through to illustrate 2D rotation as simply projection onto a pair of orthongonal unit length basis vectors.
Zipped IPYNB Notebook File

Lecture 3 Notebook 3
Notebook cs410lec03n03 illustrates the composition of 2D rotation, translation and scaling using 2D homogeneous transforms.
Zipped IPYNB Notebook File

Session Time 2 Secs.
Originating IP 127.0.0.1
User: Guest

Apply to CSU | Contact CSU | Disclaimer | Equal Opportunity
Colorado State University, Fort Collins, CO 80523 USA

SageMath on Projection



Dot Product as Projection

The linear algebraic definition of a dot product is clearly taught everywhere. Namely, the sum of pairwise products between first, second, third values etc. However, one of the more important underlying geometric meanings of the dot product is sometimes neglected. Not here. Much of what we do in computer graphics depends critically upon a reflexive understanding of the dot product as a projection operator. To better understand what we mean by 'projection', read on.

Ross Beveridge

August 28, 2018

As will be common in these notebooks, the next sequence of commands configure options for running the notebook such as how to display math, etc.

In [1]:
```
%display latex
latex.matrix_delimiters(left='|', right='|')
latex.vector_delimiters(left='[', right=']')
```

To begin, here are the basics starting with two vectors.

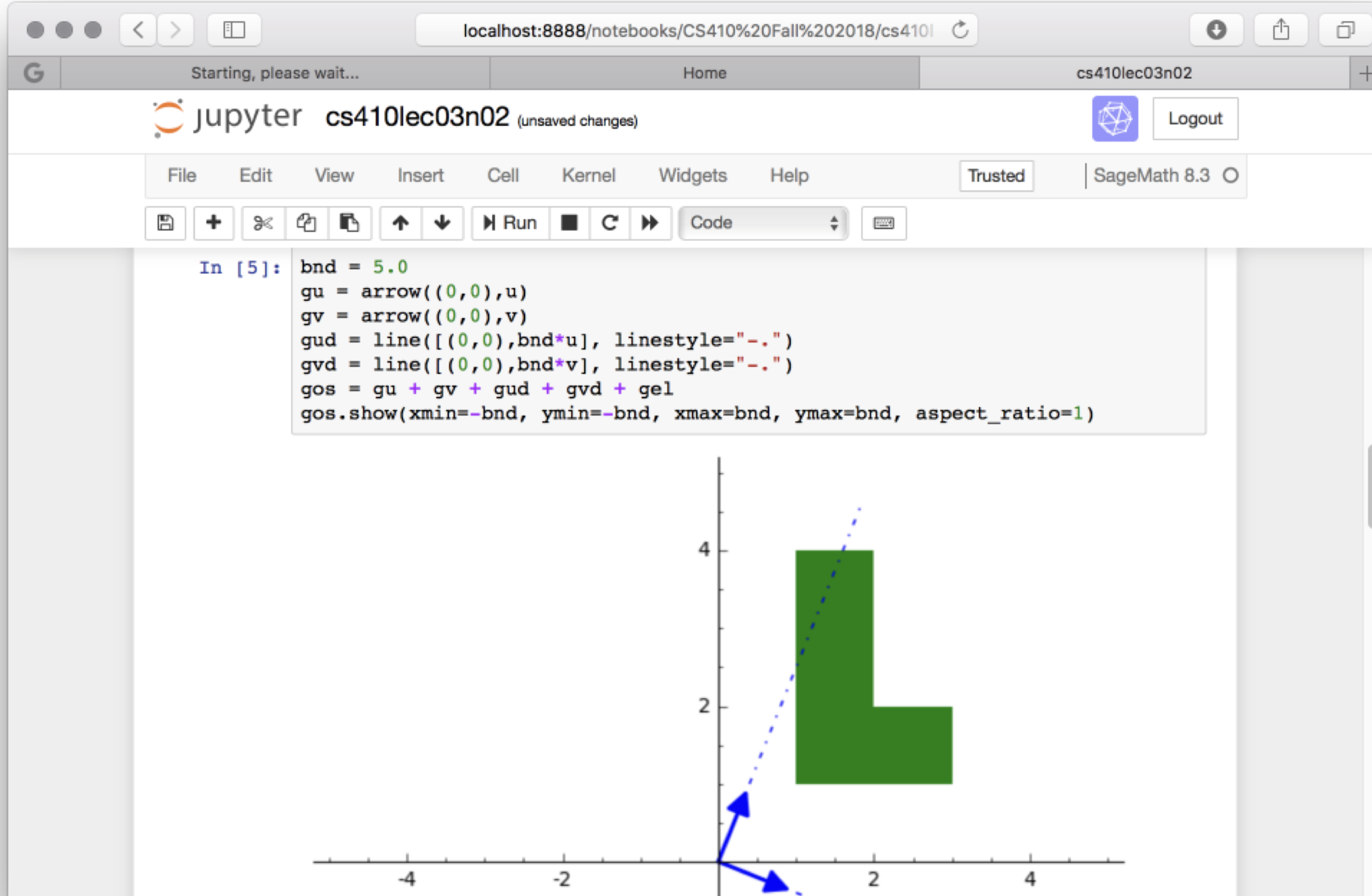
In [2]:
```
var('a','b','c','d')
u = vector([a,b])
v = vector([c,d])
pretty_print("u = ", u, ",  v = ", v)
pretty_print(LatexExpr("u \cdot v = ") + v.dot_product(u))
```

u $=[a, b]$, v $=[c, d]$

$u \cdot v = a * c + b * d$

If the vector u is of unit length, and the vector v represents the position of a point Q in space, then the dot product of u and v

SageMath Rotation is Projection

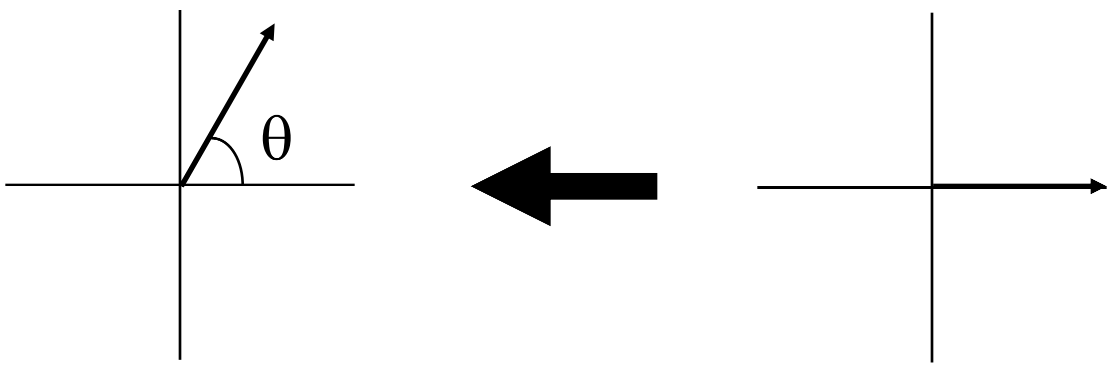


Rotate by θ

$$M = RP$$

$$R = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \qquad P = \begin{bmatrix} x \\ y \end{bmatrix}$$

$$\begin{bmatrix} \cos(\theta)\, x - \sin(\theta)\, y \\ \sin(\theta)\, x + \cos(\theta)\, y \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$



Does this make sense, given the geometry of the dot product?

Derivation of Rotation Matrix

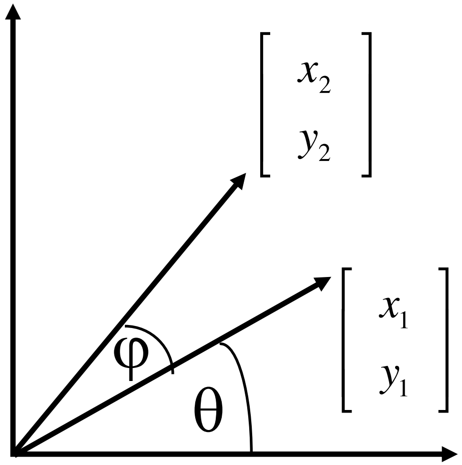
$$x_1 = r\cos(\theta) \qquad x_2 = r\cos(\theta + \phi)$$

$$y_1 = r\sin(\theta) \qquad y_2 = r\sin(\theta + \phi)$$

Pronunciation Guide ☺

θ theta

φ phi (fee)



Derivation (cont.)

Trig. Identity:

$$\cos(a+b) = \cos(a)\cos(b) - \sin(a)\sin(b)$$

$$\sin(a+b) = \sin(a)\cos(b) + \sin(b)\cos(a)$$

$$x_2 = r\cos(\theta+\phi)$$

$$x_2 = r\cos(\theta)\cos(\phi) - r\sin(\theta)\sin(\phi)$$

$$x_2 = x_1\cos(\phi) - y_1\sin(\phi)$$

Remember by Definition

$$x_1 = r\cos(\theta)$$

$$y_1 = r\sin(\theta)$$

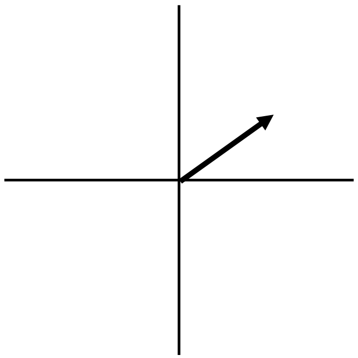
Y is left for you. Really, try it

Uniform Scaling

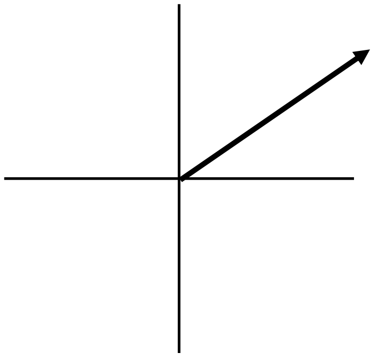
The first of several 2D canonical matrices…

$$S = \begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix}, \quad P = \begin{bmatrix} x \\ y \end{bmatrix}, \quad M = S\,P$$

$$\begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} s\,x \\ s\,y \end{bmatrix}$$

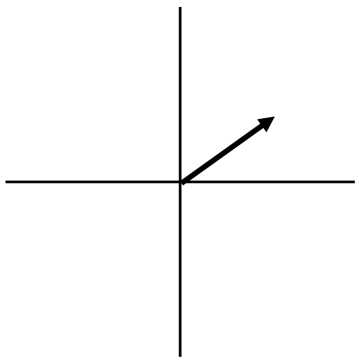


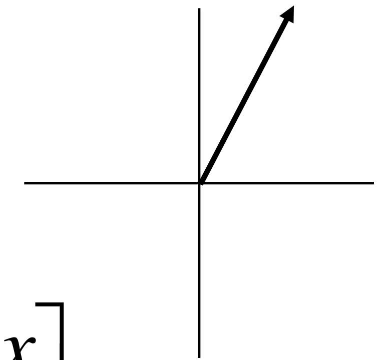
$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad M = s\;I\;P, \quad M = s \begin{bmatrix} x \\ y \end{bmatrix}$$



Non-uniform Scaling

$$S = \begin{bmatrix} s_1 & 0 \\ 0 & s_2 \end{bmatrix} \qquad P = \begin{bmatrix} x \\ y \end{bmatrix}$$



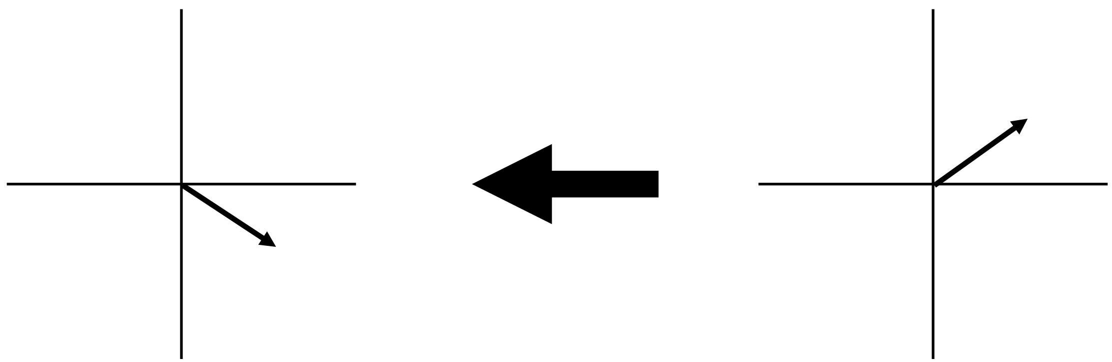


$$M = S\,P \qquad \begin{bmatrix} s_1 & 0 \\ 0 & s_2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} s_1\,x \\ s_2\,y \end{bmatrix}$$

Note orientation shift in line

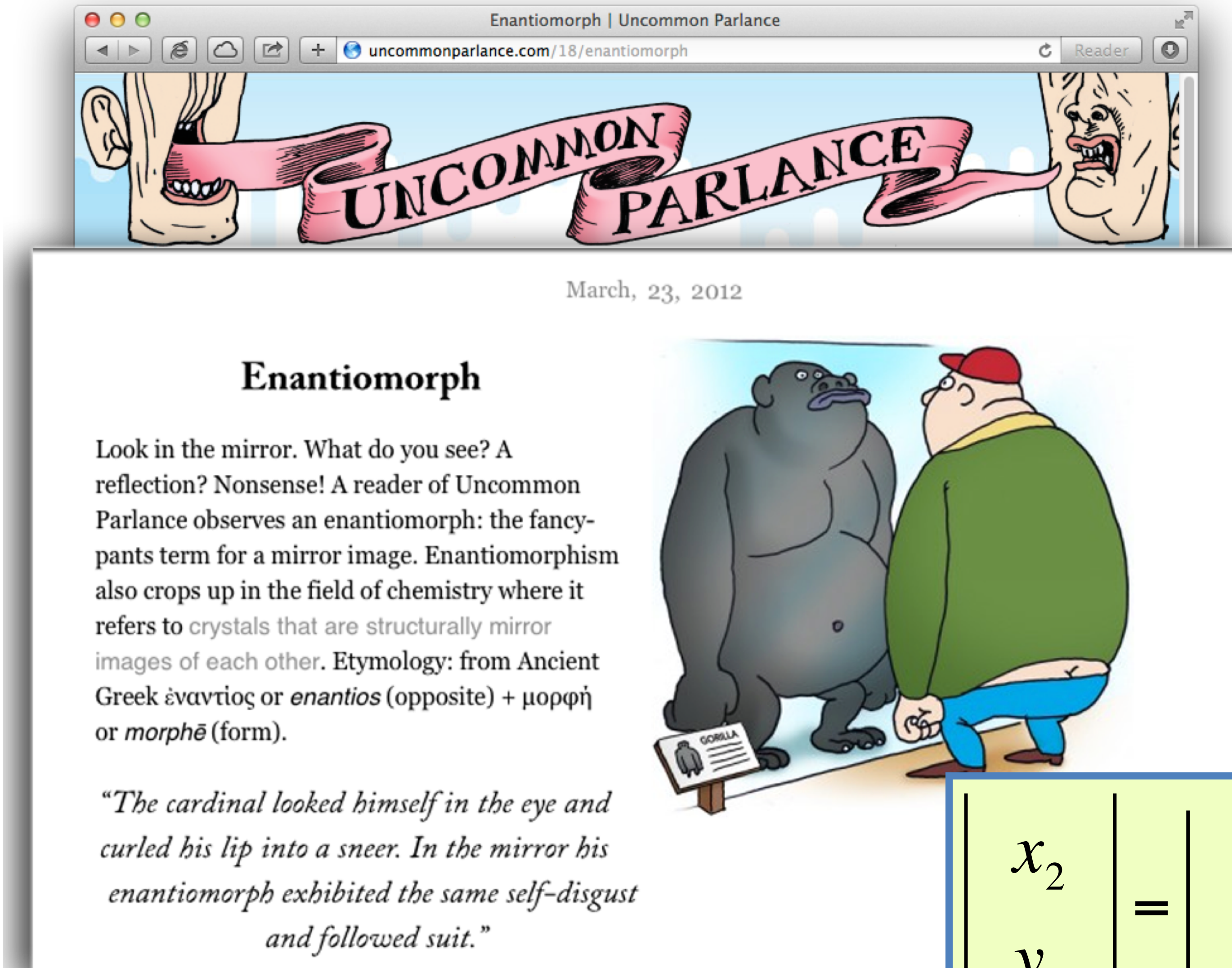
Flip an Axis...

$$\begin{bmatrix} x \\ -y \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$



What does this do to appearance of objects?

Flip the Horizontal


Enantiomorph | Uncommon Parlance

uncommonparlance.com/18/enantiomorph

UNCOMMON PARLANCE

March, 23, 2012

Enantiomorph

Look in the mirror. What do you see? A reflection? Nonsense! A reader of Uncommon Parlance observes an enantiomorph: the fancy-pants term for a mirror image. Enantiomorphism also crops up in the field of chemistry where it refers to crystals that are structurally mirror images of each other. Etymology: from Ancient Greek ἐναντίος or *enantios* (opposite) + μορφή or *morphē* (form).

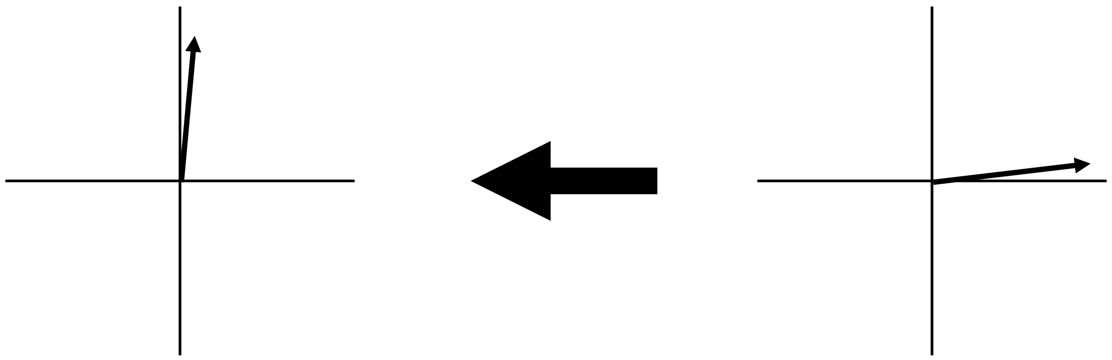
“The cardinal looked himself in the eye and curled his lip into a sneer. In the mirror his enantiomorph exhibited the same self-disgust and followed suit.”

Credits: Uncommon Parlance

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$$

Swap Axes

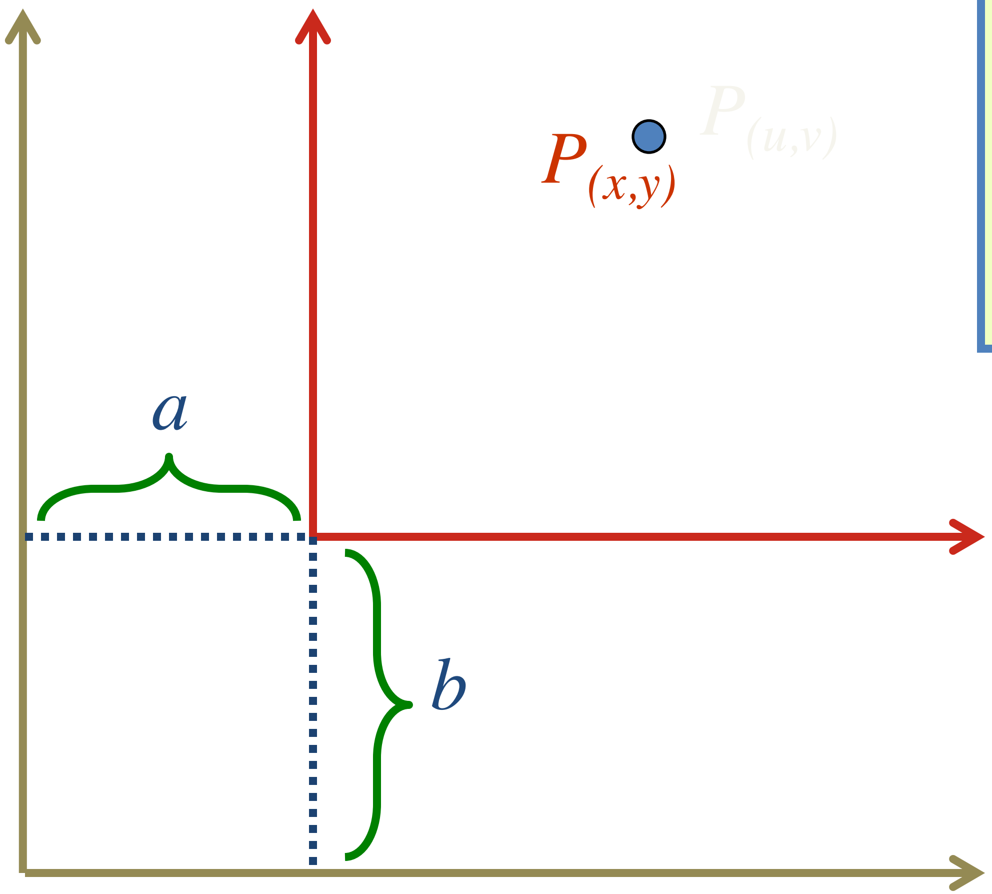
$$\begin{bmatrix} y \\ x \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$



Why would you do this?

Translation

Addition,
not multiplication!



$$P_{(x,y)} = \left| \begin{matrix} x \\ y \end{matrix} \right|$$

$$P_{(u,v)} = \left| \begin{matrix} u \\ v \end{matrix} \right| = \left| \begin{matrix} x \\ y \end{matrix} \right| + \left| \begin{matrix} a \\ b \end{matrix} \right|$$

I am intentionally drawing the alternative geometry, i.e. move the origin not the point.

Plug in some values and draw yourself some pictures.

Canonical Transformations

$$Rotate = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

$$Scale = \begin{bmatrix} s_1 & 0 \\ 0 & s_2 \end{bmatrix}$$

$$Flip = \begin{bmatrix} \pm 1 & 0 \\ 0 & \pm 1 \end{bmatrix}$$

$$Translate = \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

Add, don't multiply

Composition …

To apply transformation A to point p, and then transform the result by transformation B:

$$p' = (B\,A)\,p = B\,(A\,p)$$

Question: why is this important?

Problem: Translation

Unfortunately, we often need to translate points, and translation is matrix addition, not multiplication

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} + \begin{bmatrix} t_x \\ y_y \end{bmatrix}$$

We need some way to make translation into a matrix multiplication operation, so that all transformations can be composed...

Solution: Homogeneous Coordinates

In homogeneous coordinates, a two-dimensional point is represented as a vector of length 3.

In homogeneous coordinates, a three-dimensional point is represented as a vector of length 4

In general, homogeneous coordinates represent an N-dimensional point with a vector of length N+1.

Homogeneous Coordinates (cont.)

In particular, the 2D point (x,y) is:

$$\begin{vmatrix} x \\ y \\ 1 \end{vmatrix} = \begin{vmatrix} 2x \\ 2y \\ 2 \end{vmatrix} = \begin{vmatrix} nx \\ ny \\ n \end{vmatrix}$$

For any $n \neq 0$

Question: what is the last coordinate (conceptually)?

Homogeneous Coordinates (cont...)

- Note that homogeneous coordinates are non-unique, but

- Translation in homogeneous coordinates is multiplication:

$$\begin{vmatrix} x + t_x \\ y + t_y \\ 1 \end{vmatrix} = \begin{vmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} x \\ y \\ 1 \end{vmatrix}$$

Canonical Homogeneous Matrices

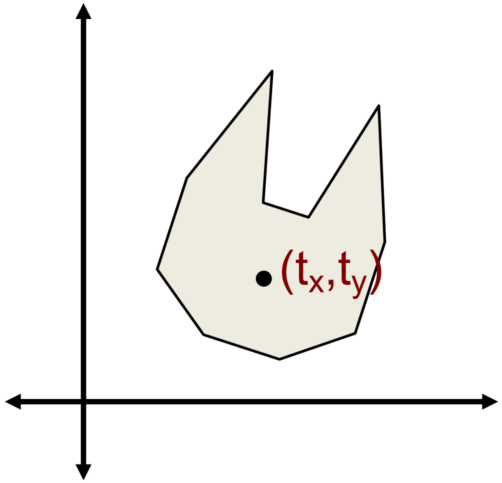
- 2D Rotation looks pretty much the same:

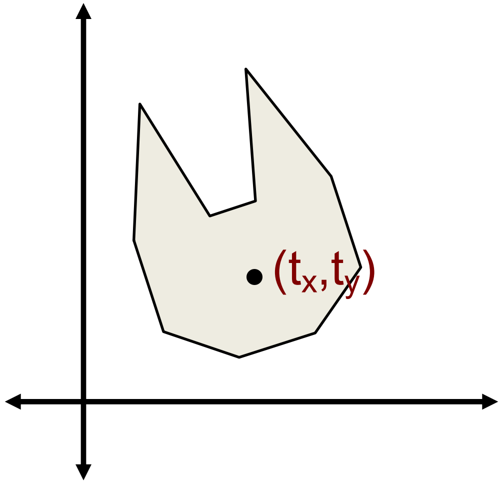
$$\begin{bmatrix} x_2 \\ y_2 \\ w_2 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ w_1 \end{bmatrix}$$

- As does 2D non-uniform scaling:

$$\begin{bmatrix} x_2 \\ y_2 \\ w_2 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ w_1 \end{bmatrix}$$

Rotate about a Point





- Rotate the Cat's Head about its Nose
  1. Translate the Nose to the Origin
  2. Rotate by the desired amount
  3. Invert the translation

Rotation about a Point (II)

- Translate to origin

  Note the negations: we want to bring (t_x, t_y) to the origin, so subtract t_x, t_y.

$$M_T = \begin{bmatrix} 1 & 0 & -t_x \\ 0 & 1 & -t_y \\ 0 & 0 & 1 \end{bmatrix}$$

- Rotate about origin

  What was the point (t_x, t_y) is now at the origin.

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

- Translate back

  Finally, what started as (t_x, t_y) is again (t_x, t_y).

$$M_{T^{-1}} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

Rotation about a Point (III)

$$M = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -t_x \\ 0 & 1 & -t_y \\ 0 & 0 & 1 \end{bmatrix}$$

Translate Back — Rotate — Translate to Origin

- Think about order!
- Operations right before those on left.
- Therefore, read from right to left.

Rotation about a Point (IV)

- Reminder, in matrix multiplication:

$$AB \neq BA$$

- The equation to rotate a matrix of points P around (t_x, t_y) is:

$$P' = T^{-1}RTP$$

Rotation about a Point (V)

- Compose the three transformations.

$$P' = \left(T^{-1}RT\right)P$$

$$P' = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & \sin(\theta)t_y + \left(1-\cos(\theta)\right)t_x \\ \sin(\theta) & \cos(\theta) & -\sin(\theta)t_x + \left(1-\cos(\theta)\right)t_y \\ 0 & 0 & 1 \end{bmatrix} P$$

Scaling About Point P

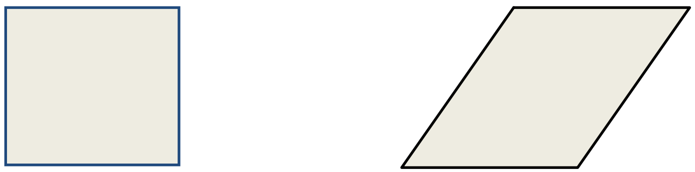
- Scaling also operates relative to the origin.
- To make an object bigger without moving it
  - Translate origin to object centroid.
  - Apply scaling.
  - Invert the translation.

$$M_4 = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -t_x \\ 0 & 1 & -t_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & -s_x t_x + t_x \\ 0 & s_y & -s_y t_y + t_y \\ 0 & 0 & 1 \end{bmatrix}$$

One More Transform - Shear

Shearing in the X dimension

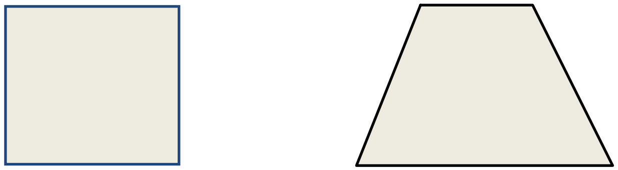
- Metaphor - Wind Blows Figure.



- Basic Matrix Form.

$$\begin{vmatrix} 1 & sh_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

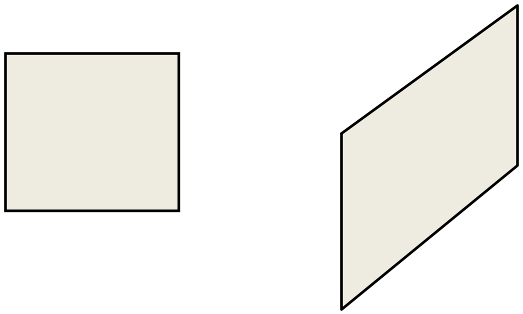
- Can X-Shear do This?



Shear (cont.)

Shearing in the Y dimension

- Metaphor - Same thing, other direction.



- Basic Matrix Form.

$$\begin{vmatrix} 1 & 0 & 0 \\ sh_y & 1 & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

- Note: Parallel Lines Stay Parallel

Use Notebook 3 to develop an intuition as well as mechanical understanding!

Starting, please wait | Home | cs410lec03n03

jupyter cs410lec03n03 (autosaved)

Logout

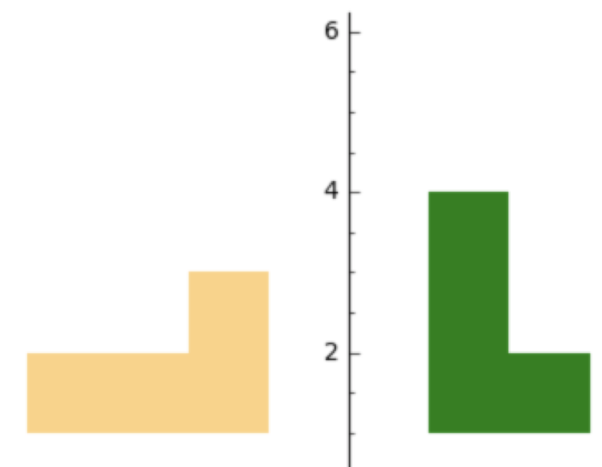
File Edit View Insert Cell Kernel Widgets Help | Trusted | SageMath 8.3

Run | Code

$$\text{Before: Pts} = \begin{vmatrix} 1 & 3 & 3 & 2 & 2 & 1 \\ 1 & 1 & 2 & 2 & 4 & 4 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{vmatrix}$$

$$\text{After: Pts} = \begin{vmatrix} -1.0000 & -1.0000 & -2.0000 & -2.0000 & -4.0000 & -4.0000 \\ 1.0000 & 3.0000 & 3.0000 & 2.0000 & 2.0000 & 1.0000 \\ 1.0000 & 1.0000 & 1.0000 & 1.0000 & 1.0000 & 1.0000 \end{vmatrix}$$

In [42]:
```
ptsA = list(elA[0:2,:].transpose())
ptsB = list(elB[0:2,:].transpose())
gelA = polygon(ptsA,color='green')
gelB = polygon(ptsB,color='orange',alpha=0.5)
bnd = 6.0
gos = gelA + gelB
gos.show(xmin=-bnd, ymin=-bnd, xmax=bnd, ymax=bnd, aspect_ratio=1)
```



The End