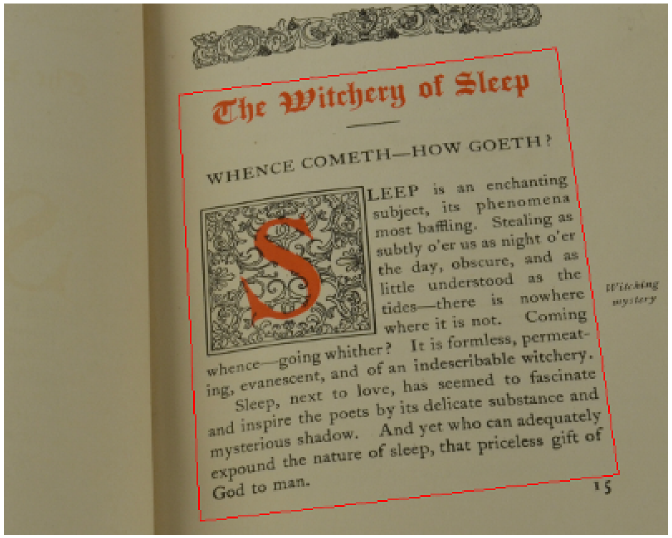
Homogeneous coordinates

Why to care?

1. Application in computer graphics.
2. Mathematical investment.

Pragmatic example



The Witchery of Sleep

WHENCE COMETH—HOW GOETH?

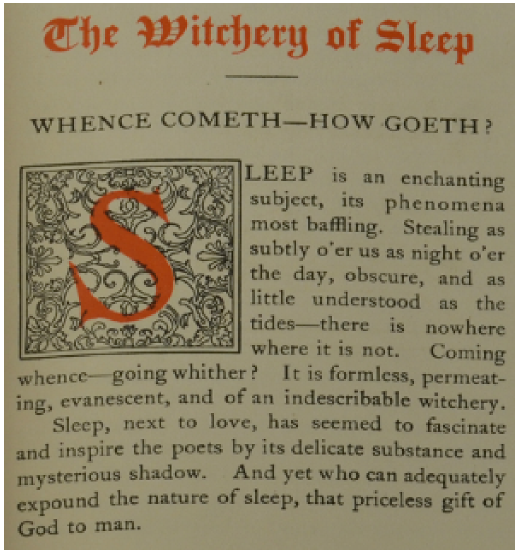
SLEEP is an enchanting subject, its phenomena most baffling. Stealing as subtly o'er us as night o'er the day, obscure, and as little understood as the tides—there is nowhere where it is not. Coming whence—going whither? It is formless, permeating, evanescent, and of an indescribable witchery.

Witching mystery

Sleep, next to love, has seemed to fascinate and inspire the poets by its delicate substance and mysterious shadow. And yet who can adequately expound the nature of sleep, that priceless gift of God to man.

15

m.TransformPoints(xy);



The Witchery of Sleep

WHENCE COMETH—HOW GOETH?

SLEEP is an enchanting subject, its phenomena most baffling. Stealing as subtly o'er us as night o'er the day, obscure, and as little understood as the tides—there is nowhere where it is not. Coming whence—going whither? It is formless, permeating, evanescent, and of an indescribable witchery.

Sleep, next to love, has seemed to fascinate and inspire the poets by its delicate substance and mysterious shadow. And yet who can adequately expound the nature of sleep, that priceless gift of God to man.

- System.Drawing.Drawing2D.Matrix.TransformPoints:
  - System.Drawing.SafeNativeMethods.Gdip.ConvertPointToMemory,
  - System.Drawing.SafeNativeMethods.Gdip.ConvertGPPOINTFArrayF:
    - System.Drawing.UnsafeNativeMethods.PtrToStructure:
      - System.Drawing.Internal.GPPOINTF..ctor (which is empty, by the way),
      - System.RuntimeType.CreateInstanceSlow:
        - System.Runtime.InteropServices.Marshal.PtrToStructure.

All and all: 41.18 seconds

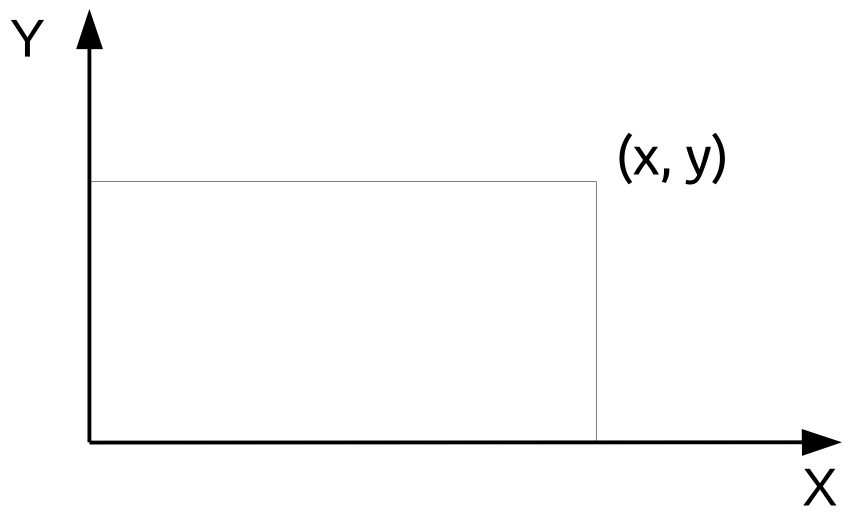
Pragmatic example

```
for (int i = 0; i < ciH; i++)
    {
        for (int j = 0; j < ciW; j++)
        {
            x = xy[i*ciW + j].x;
            y = xy[i*ciW + j].y;
            double d_ = 1.0f / (a * x + b * y + c);
            xy[i*ciW + j].x = (A * x + B * y + C) * d_;
            xy[i*ciW + j].y = (D * x + E * y + F) * d_;
        }
    }
```

```
ldloc.s   A
ldloc.s   x
mul
ldloc.s   B
ldloc.s   y
mul
add
ldloc.s   C
add
ldloc.s   d_
mul
```

Computations only: 0.18 seconds

Complication



$(x_a, y_a) = (4, 2)$

$(x_p, y_p, w_p) = (4, 2, 1)$

$x_a = x_p / w_p$

$y_a = y_p / w_p$

$(4, 2, 1) = (8, 4, 2)$

$= (12, 6, 3)$

$= (2, 1, 0.5)$

$= (4w, 2w, w)$

Cartesian and homogeneous coordinates

$$(X_a, Y_a) = (X_p / W_p, Y_p / W_p)$$

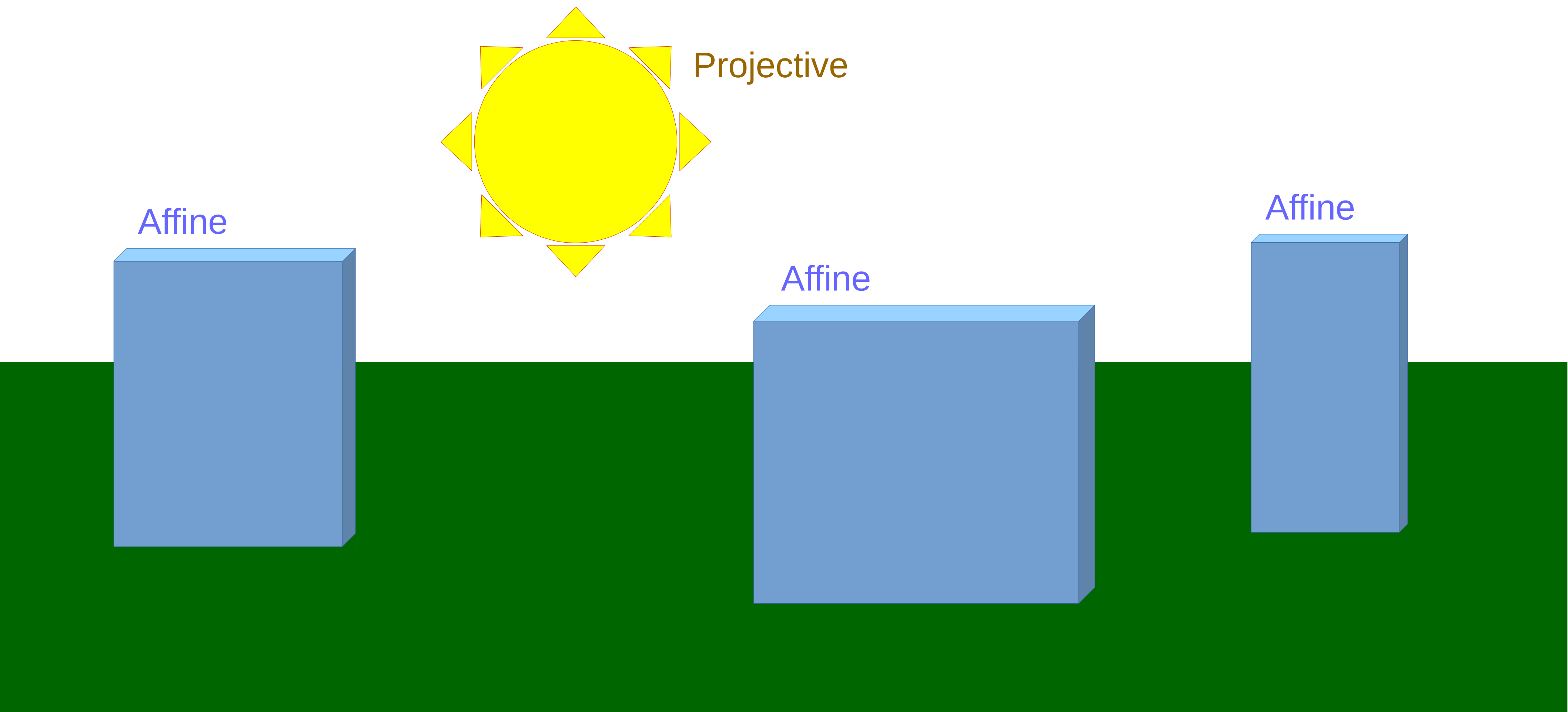
Cartesian	Homogeneous
(4, 2)	(4, 2, 1)
(40, 20)	(4, 2, 0.1)
(400, 200)	(4, 2, 0.01)
(4000, 2000)	(4, 2, 0.001)
(? , ?)	(4, 2, 0)

Should be somewhere on the same line,

but further than any other point, right?

Projective space expands affine space

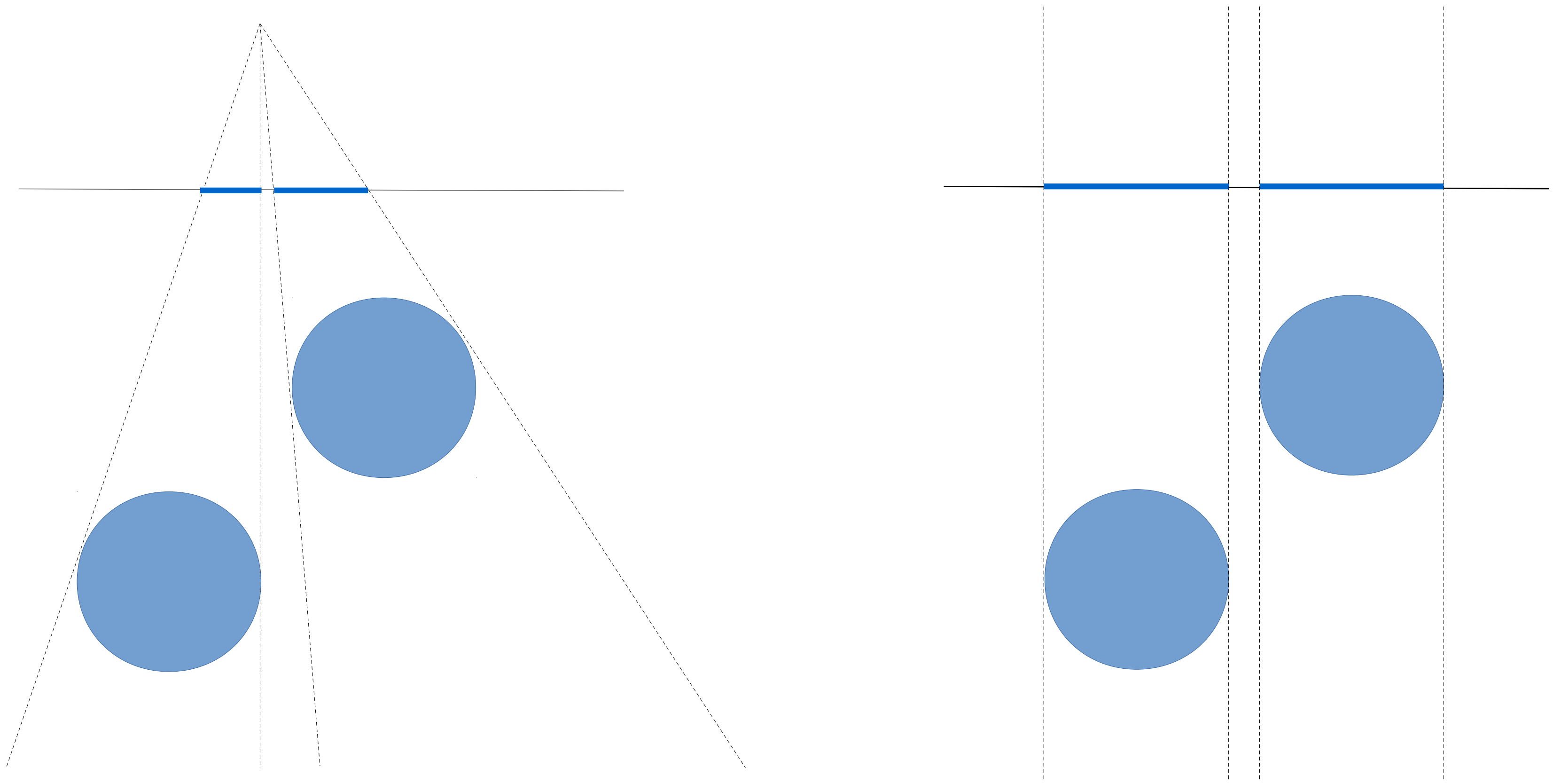
- (X, Y, Z, 0) = (X, Y, Z, 0) – don't translate to Euclidean space
- infinitely "far" from any Cartesian point
- in a way represents direction in Euclidean space



Benefits of projective space

1. Central and parallel projections are the same

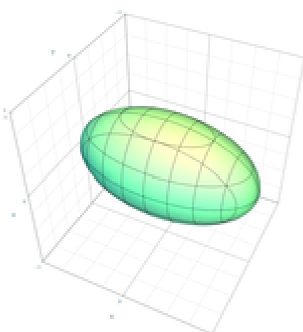
$$(x_p, y_p, z_p, 0) - (x_a, y_a, z_a, 1) = (x_p, y_p, z_p, 0)$$



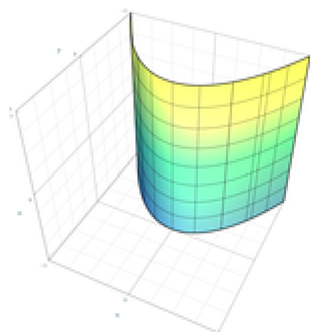
Benefits of projective space

2. All the surfaces described by an equation of degree *n* are the same

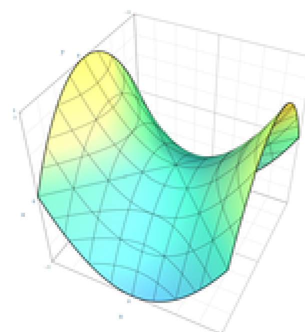
There are 17 different quadrics:



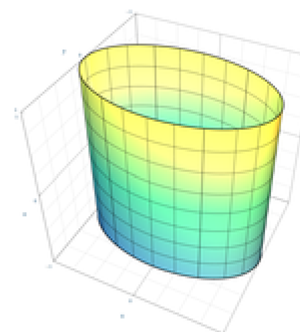
$$\frac{x^2}{a^2}+\frac{y^2}{b^2}+\frac{z^2}{c^2}=1$$



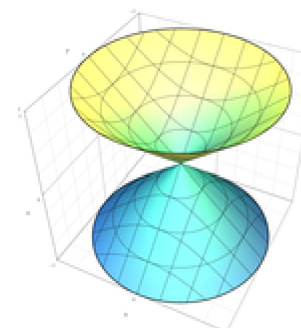
$$x^2+2ay=0$$



$$\frac{x^2}{a^2}-\frac{y^2}{b^2}-z=0$$



$$\frac{x^2}{a^2}+\frac{y^2}{b^2}=1$$



$$\frac{x^2}{a^2}+\frac{y^2}{a^2}-\frac{z^2}{b^2}=0$$

...and 12 more

$x^2 + 2\,a\,y = 0$; $\qquad x^2 / w^2 + 2\,a\,y / w = 0$; $\qquad x^2 + 2\,a\,y\,w = 0$

$$Q(X)=\sum_{ij} a_{ij} X_i X_j = 0$$

Benefits of projective space

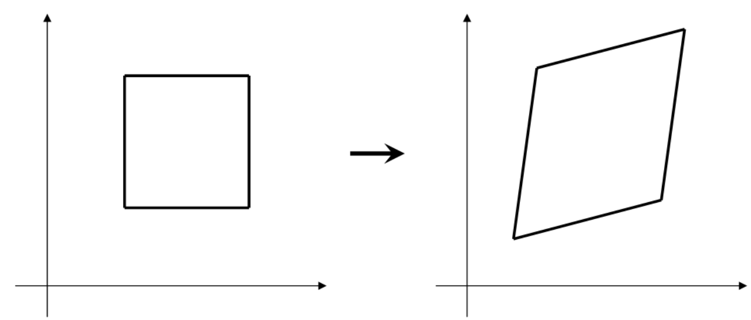
3. Common geometrical transformations form the logical structure...

Translation:
$x' = x + A$
$y' = y + B$

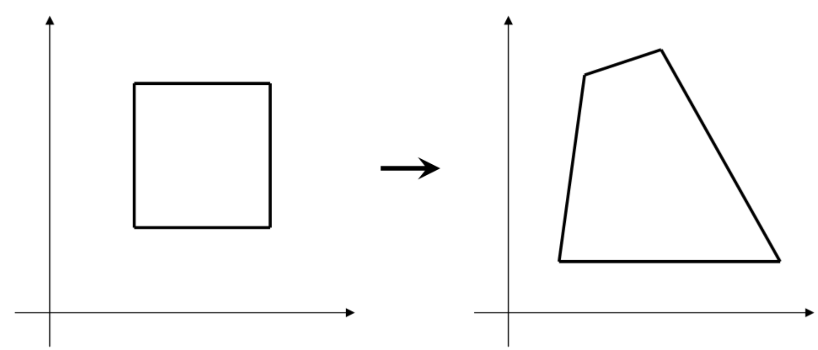
Rotation:
$x' = \sin(r)\, x + \cos(r)\, y$
$y' = -\cos(r)\, x + \sin(r)\, y$

Scaling:
$x' = Ax$
$y' = By$

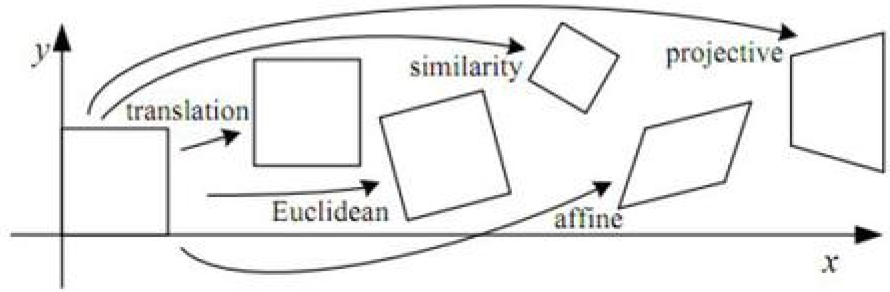
Affine transformation:
$x' = Ax + By + C$
$y' = Dx + Ey + F$

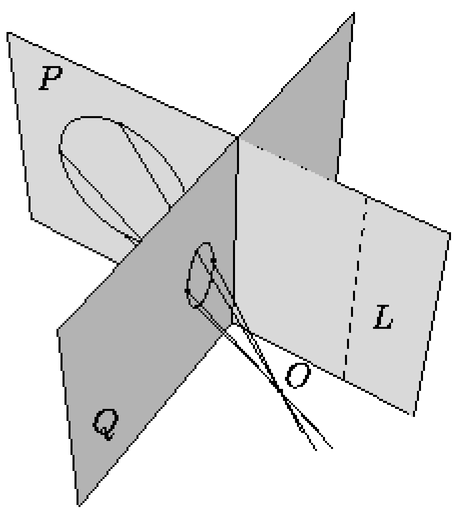


Projective transformation:
$x' = (Ax + By + C)/(ax + by + c)$
$y' = (Dx + Ey + F)/(ax + by + c)$



Projective transformation





$$x' = \frac{Ax+By+C}{ax+by+c}$$

$$y' = \frac{Dx+Ey+F}{ax+by+c}$$

Projective transformation is a matrix multiplication

$$\begin{bmatrix} A & D & a \\ B & E & b \\ C & F & c \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} Ax + By + Cw \\ Dx + Ey + Fw \\ ax + by + cw \end{bmatrix}$$

$$x' = \frac{Ax+By+C}{ax+by+c}$$

$$y' = \frac{Dx+Ey+F}{ax+by+c}$$

$$w' = 1$$

- Affine: a = 0, b = 0, c = 1
  - Scale:

    A = x-scale, E = y-scale
  - Rotation:

    A, E = sin(r); B, -D = cos(r)
  - Translation:

    A = 1, B = 0, C = x-translation

    D = 0, E = 1, F = y-translation

Benefits of projective space

3. All projective transformations form the logical structure
(that is a square matrices over multiplication group)

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ A & B & 1 \end{bmatrix} \qquad \begin{bmatrix} sin(r) & -cos(r) & 0 \\ cos(r) & sin(r) & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} A & 0 & 0 \\ 0 & B & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} A & D & 0 \\ B & E & 0 \\ C & F & 1 \end{bmatrix}$$

$$\begin{bmatrix} A & D & a \\ B & E & b \\ C & F & c \end{bmatrix}$$

Nevermind the “group”.
Properties are important, not the name

1) For all a, b in G, the result of the operation, a • b, is also in G.

   Composability

2) For all a, b and c in G, (a • b) • c = a • (b • c).

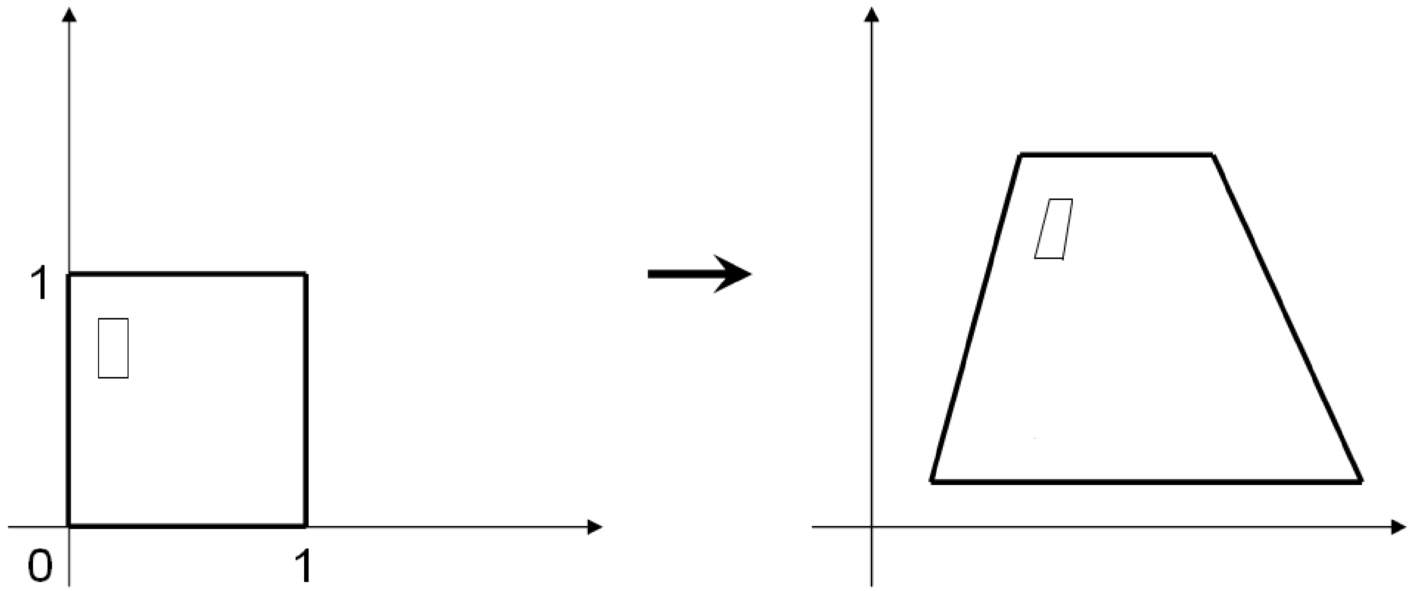
   Parallelability

3) There exists an element e in G such that, for every element a in G, the equation e • a = a • e = a holds.

4) For each a in G, there exists an element b in G, commonly denoted a^{-1}, such that a • b = b • a = e, where e is the identity element.

   Invertability

Ex. 1. Build a matrix from a 4 points transformation



$x'_i = (A\ x_i + B\ y_i + C) / (a\ x_i + b\ y_i + c)$

$y'_i = (D\ x_i + E\ y_i + F\ ) / (a\ x_i + b\ y_i + c)$

$i = 1..4$

(x'_i, y'_i) – translated affine coordinates

(x_i, y_i) – singular cube “corners”: (0, 0), (0, 1), (1, 1), (1, 0).

Let’s use SymPy

http://live.sympy.org/

```
from sympy import *

aa, bb, cc = symbols('aa bb cc')        # A  B  C
dd, ee, ff = symbols('dd ee ff')        # D  E  F
a, b, c = symbols('a b c')              #  a  b  c

x0, x1, x2, x3 = symbols('x0 x1 x2 x3')
y0, y1, y2, y3 = symbols('y0 y1 y2 y3')

x_0, y_0 = (0, 0)
x_1, y_1 = (0, 1)
x_2, y_2 = (1, 0)
x_3, y_3 = (1, 1)

...
```

Linear system

$$x' = \frac{Ax+By+C}{ax+by+c}$$

$$Ax + By + C = x'(ax + by + c)$$

$$y' = \frac{Dx+Ey+F}{ax+by+c}$$

$$Dx + Ey + F = y'(ax + by + c)$$

```
linsolve([aa*x_0 + bb*y_0 + cc - x0*a*x_0 - x0*b*y_0 - x0*c,
          dd*x_0 + ee*y_0 + ff - y0*a*x_0 - y0*b*y_0 - y0*c,
          aa*x_1 + bb*y_1 + cc - x1*a*x_1 - x1*b*y_1 - x1*c,
          dd*x_1 + ee*y_1 + ff - y1*a*x_1 - y1*b*y_1 - y1*c,
          aa*x_2 + bb*y_2 + cc - x2*a*x_2 - x2*b*y_2 - x2*c,
          dd*x_2 + ee*y_2 + ff - y2*a*x_2 - y2*b*y_2 - y2*c,
          aa*x_3 + bb*y_3 + cc - x3*a*x_3 - x3*b*y_3 - x3*c,
          dd*x_3 + ee*y_3 + ff - y3*a*x_3 - y3*b*y_3 - y3*c],
         (aa, bb, cc, dd, ee, ff, a, b, c))
```

A class of solutions

```
aa =  -c*(x0*x1*y2 - x0*x1*y3 + x0*x3*y1 - x0*x3*y2 - x1*x2*y0 + x1*x2*y3
+ x2*x3*y0 - x2*x3*y1)/(x1*y2 - x1*y3 - x2*y1 + x2*y3 + x3*y1 - x3*y2),

bb =  -c*(x1*((x2 - x3)*(y0 - y1 - y2 + y3) - (y2 - y3)*(x0 - x1 - x2 +
x3)) + (x0 - x1)*((x1 - x3)*(y2 - y3) - (x2 - x3)*(y1 - y3)))/((x1 -
x3)*(y2 - y3) - (x2 - x3)*(y1 - y3)),

cc =  c*x0,

dd =  -c*(x0*y1*y2 - x0*y2*y3 - x1*y0*y3 + x1*y2*y3 - x2*y0*y1 + x2*y0*y3
+ x3*y0*y1 - x3*y1*y2)/(x1*y2 - x1*y3 - x2*y1 + x2*y3 + x3*y1 - x3*y2),

ee =  -c*(y1*((x2 - x3)*(y0 - y1 - y2 + y3) - (y2 - y3)*(x0 - x1 - x2 +
x3)) + (y0 - y1)*((x1 - x3)*(y2 - y3) - (x2 - x3)*(y1 - y3)))/((x1 -
x3)*(y2 - y3) - (x2 - x3)*(y1 - y3)),

ff =  c*y0,

a =  -c*(x0*y1 - x0*y3 - x1*y0 + x1*y2 - x2*y1 + x2*y3 + x3*y0 - x3*y2)/
(x1*y2 - x1*y3 - x2*y1 + x2*y3 + x3*y1 - x3*y2),

b =  -c*((x2 - x3)*(y0 - y1 - y2 + y3) - (y2 - y3)*(x0 - x1 - x2 + x3))/
((x1 - x3)*(y2 - y3) - (x2 - x3)*(y1 - y3)),

c = c
```

Handpicked solution

```
linsolve([aa*x_0 + bb*y_0 + cc - x0*a*x_0 - x0*b*y_0 - x0*c,
          dd*x_0 + ee*y_0 + ff - y0*a*x_0 - y0*b*y_0 - y0*c,
          aa*x_1 + bb*y_1 + cc - x1*a*x_1 - x1*b*y_1 - x1*c,
          dd*x_1 + ee*y_1 + ff - y1*a*x_1 - y1*b*y_1 - y1*c,
          aa*x_2 + bb*y_2 + cc - x2*a*x_2 - x2*b*y_2 - x2*c,
          dd*x_2 + ee*y_2 + ff - y2*a*x_2 - y2*b*y_2 - y2*c,
          aa*x_3 + bb*y_3 + cc - x3*a*x_3 - x3*b*y_3 - x3*c,
          dd*x_3 + ee*y_3 + ff - y3*a*x_3 - y3*b*y_3 - y3*c,
          c - 1], (aa, bb, cc, dd, ee, ff, a, b, c))
```

```
aa = (-x0*x1*y2 + x0*x1*y3 - x0*x3*y1 + x0*x3*y2 + x1*x2*y0 - x1*x2*y3 - x2*x3*y0 +
x2*x3*y1)/(x1*y2 - x1*y3 - x2*y1 + x2*y3 + x3*y1 - x3*y2),
bb = (x1*((x2 - x3)*(-y0 + y1 + y2 - y3) + (y2 - y3)*(x0 - x1 - x2 + x3)) + (-x0 +
x1)*((x1 - x3)*(y2 - y3) - (x2 - x3)*(y1 - y3)))/((x1 - x3)*(y2 - y3) - (x2 - x3)*(y1
- y3)),
cc = x0,
dd = (-x0*y1*y2 + x0*y2*y3 + x1*y0*y3 - x1*y2*y3 + x2*y0*y1 - x2*y0*y3 - x3*y0*y1 +
x3*y1*y2)/(x1*y2 - x1*y3 - x2*y1 + x2*y3 + x3*y1 - x3*y2),
ee = (y1*((x2 - x3)*(-y0 + y1 + y2 - y3) + (y2 - y3)*(x0 - x1 - x2 + x3)) + (-y0 +
y1)*((x1 - x3)*(y2 - y3) - (x2 - x3)*(y1 - y3)))/((x1 - x3)*(y2 - y3) - (x2 - x3)*(y1
- y3)),
ff = y0,
a = (-x0*y1 + x0*y3 + x1*y0 - x1*y2 + x2*y1 - x2*y3 - x3*y0 + x3*y2)/(x1*y2 - x1*y3 -
x2*y1 + x2*y3 + x3*y1 - x3*y2),
b = (-(x2 - x3)*(y0 - y1 - y2 + y3) + (y2 - y3)*(x0 - x1 - x2 + x3))/((x1 - x3)*(y2 -
y3) - (x2 - x3)*(y1 - y3)),
c = 1
```

Decomposition

```
linsolve([cc - x0*c,
          ff - y0*c,
          bb + cc - x1*b - x1*c,
          ee + ff - y1*b - y1*c,
          aa + cc - x2*a - x2*c,
          dd + ff - y2*a - y2*c,
          aa + bb + cc - x3*a - x3*b - x3*c,
          dd + ee + ff - y3*a - y3*b - y3*c,
          c - 1],
          (aa, bb, cc, dd, ee, ff, a, b, c))
```

```
linsolve([cc - x0*c,
          ff - y0*c,
           c - 1], (cc, ff, c))

# cc, ff, c =  x0, y0, 1


linsolve([bb + x0 - x1*b - x1,
          ee + y0 - y1*b - y1,
          aa + x0 - x2*a - x2,
          dd + y0 - y2*a - y2], (aa, bb, dd, ee))

# aa = a*x2 - x0 + x2
# bb = b*x1 - x0 + x1
# dd = a*y2 - y0 + y2
# ee = b*y1 - y0 + y1
```

Simplification

```
linsolve([a*x2 - x0 + x2 + b*x1 - x0 + x1 + 1 - x3*a - x3*b - x3,
          a*y2 - y0 + y2 + b*y1 - y0 + y1 + y0 - y3*a - y3*b - y3], (a, b))

# a = (-2*x0*y1 + 2*x0*y3 + x1*y0 - x1*y2 + x2*y1 - x2*y3 - x3*y0 + x3*y2 + y1 - y3)/
(x1*y2 - x1*y3 - x2*y1 + x2*y3 + x3*y1 - x3*y2)

# b = ((x2 - x3)*(-y0 + y1 + y2 - y3) - (y2 - y3)*(-2*x0 + x1 + x2 - x3 + 1))/((x1 -
x3)*(y2 - y3) - (x2 - x3)*(y1 - y3))
```

```
expand(((x1 - x3)*(y2 - y3) - (x2 - x3)*(y1 - y3)))

# x1*y2 - x1*y3 - x2*y1 + x2*y3 + x3*y1 - x3*y2
```

```
factor(-2*x0*y1 + 2*x0*y3 + x1*y0 - x1*y2 + x2*y1 - x2*y3 - x3*y0 + x3*y2 + y1 - y3)

simplify(-2*x0*y1 + 2*x0*y3 + x1*y0 - x1*y2 + x2*y1 - x2*y3 - x3*y0 + x3*y2 + y1 -
y3)

collect(-2*x0*y1 + 2*x0*y3 + x1*y0 - x1*y2 + x2*y1 - x2*y3 - x3*y0 + x3*y2 + y1 - y3,
        (x0, x1, x2, x3))

# x0*(-2*y1 + 2*y3) + x1*(y0 - y2) + x2*(y1 - y3) + x3*(-y0 + y2) + y1 - y3
# -2*x0*(y1 - y3) + x1*(y0 - y2) + x2*(y1 - y3) - x3*(y0 - y2) + y1 - y3
# (y1 - y3)*(x2 - 2*x0) + (y0 - y2)*(x1 - x3) + y1 - y3
# (y1 - y3)*(x2 - 2*x0 + 1) + (y0 - y2)*(x1 - x3)
```

Cheap solution

```
#
#    SOLUTION
#
# d = 1 / ((x1 - x3)*(y2 - y3) - (x2 - x3)*(y1 - y3))
#
# a = ((y1 - y3)*(x2 - 2*x0 + 1) + (y0 - y2)*(x1 - x3)) * d
# b = ((x2 - x3)*(-y0 + y1 + y2 - y3) - (y2 - y3)*(-2*x0 + x1 + x2 - x3 + 1)) * d
# c = 1
# aa = a*x2 - x0 + x2
# bb = b*x1 - x0 + x1
# cc = x0
# dd = a*y2 - y0 + y2
# ee = b*y1 - y0 + y1
# ff = y0
```

1 division
12 multiplications
32 additions/subtractions
It’s cheap.

Ex. 2. How to determine if a point “hits” transformed singular cube?

Ex. 2.1. How many inverse transformation matrices are there?

Ex. 2.2. What is the performance benefit of not using inversion?

Ex. 2.3. How to determine if a point “hits” transformed triangle?

Ex. 2.3.1. How come it’s not the same as Ex. 2?

Ex. 2.3.2. What is the performance of Ex 2.3 comparing to Ex.2?

Do you see now that knowing this stuff helps write better code?