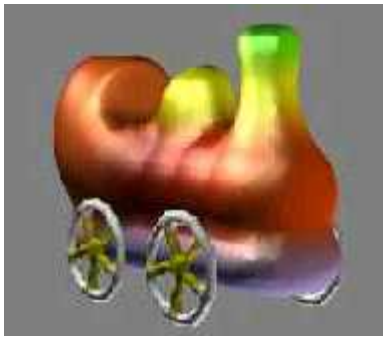


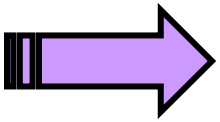


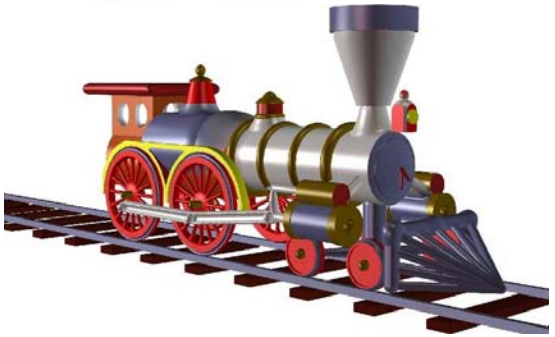


CSC 305 Transformations II

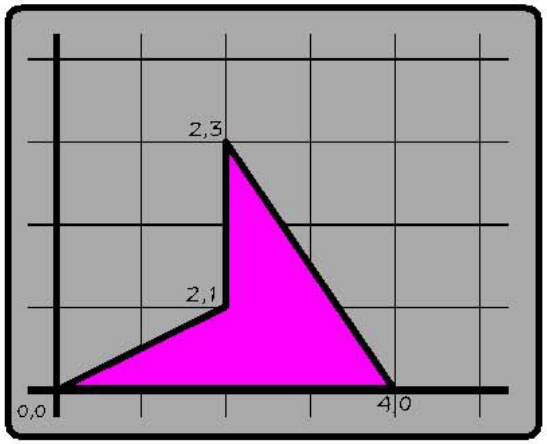
by Brian Wyvill

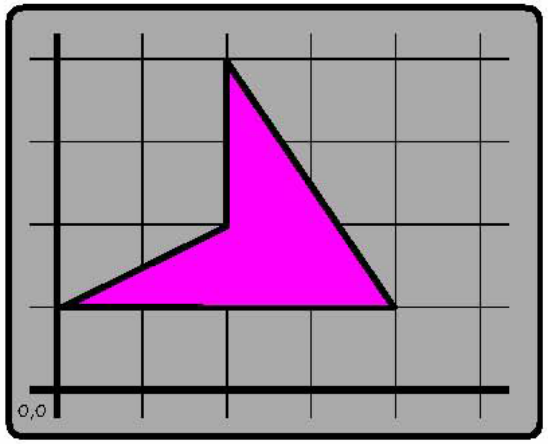
The University of Victoria Graphics Group





Translation





Object Description:
0 0
4 0
2 3
2 1

Object Description:
0 1
4 1
2 4
2 2

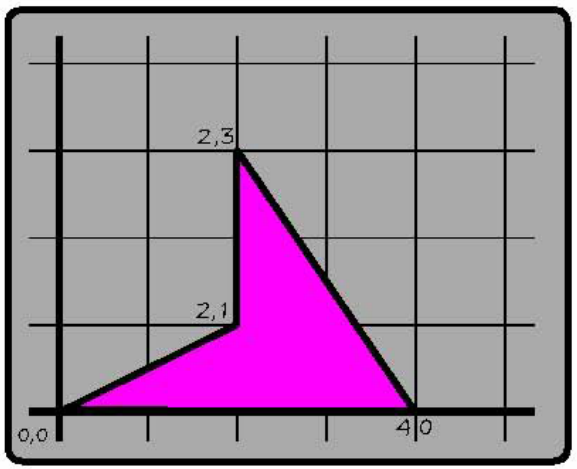
$$x_{new} = x + d_x \qquad y_{new} = y + d_y$$

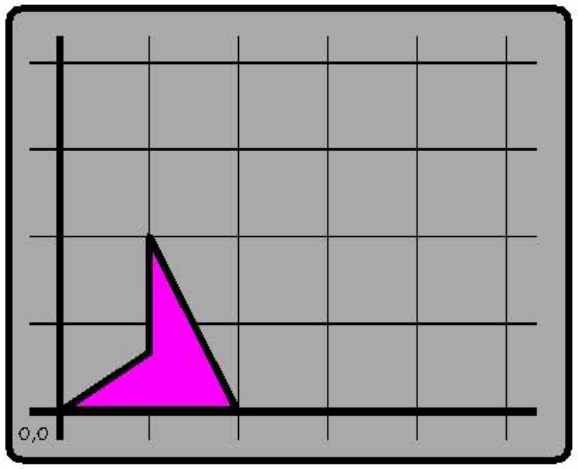
Using Vector Notation

$$P = \begin{vmatrix} x \\ y \end{vmatrix} \qquad P_{new} = \begin{vmatrix} x_{new} \\ Y_{new} \end{vmatrix} \qquad T = \begin{vmatrix} d_x \\ d_y \end{vmatrix} \qquad \text{so} \quad \boxed{P_{new} = P + T}$$



Scaling





Object Description:
0 0
4 0
2 3
2 1

Scale by 0.5 0.6667

Object Description:
0 0
2 0
1 2
1 0.6667

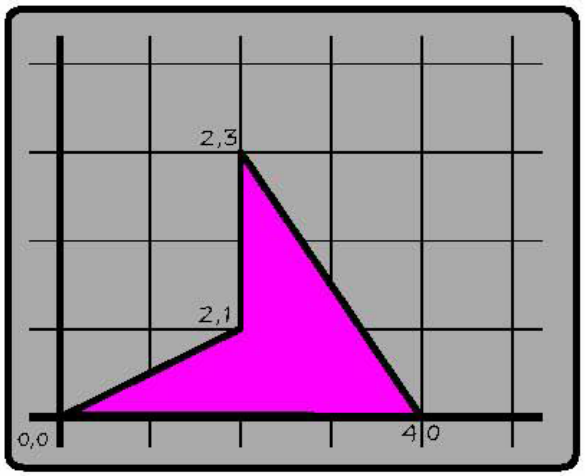
$$x_{new} = x * S_x \qquad y_{new} = y * S_y$$

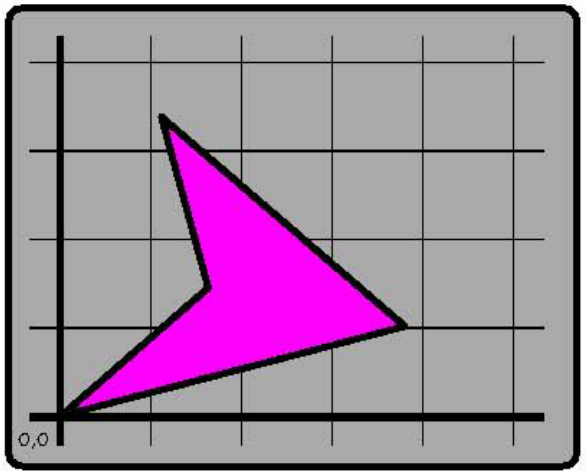
Using Matrices S is the scaling matrix

$$\begin{vmatrix} x_{new} \\ y_{new} \end{vmatrix} = \begin{vmatrix} S_x & 0 \\ 0 & S_y \end{vmatrix} * \begin{vmatrix} x \\ y \end{vmatrix} \quad \text{or} \quad \boxed{P_{new} = S * P}$$



Rotation





Object Description:
0 0
4 0
2 3
2 1

Rotate by 15 degrees →

Object Description:
0 0
3.863703 1.035276
1.155395 3.415416
1.673033 1.483564

$$x_{new} = x * \cos\theta - y * \sin\theta$$
$$y_{new} = x * \sin\theta + y * \cos\theta$$

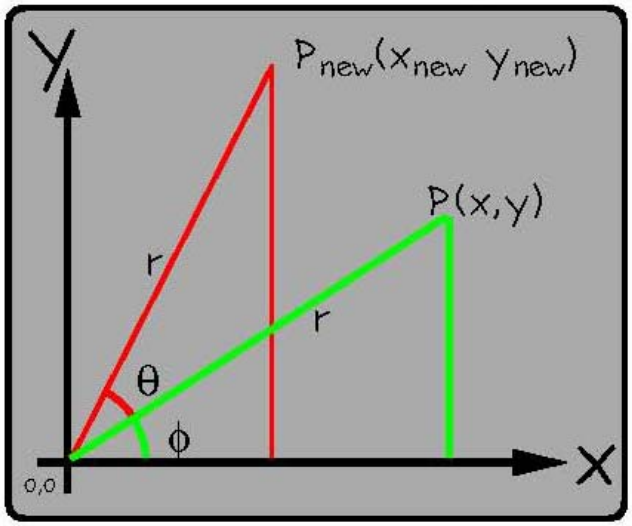
Using Matrices: R is the rotation matrix

$$\begin{vmatrix} x_{new} \\ y_{new} \end{vmatrix} = \begin{vmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{vmatrix} * \begin{vmatrix} x \\ y \end{vmatrix} \quad \text{or} \quad P_{new} = R * P$$



Rotation

Positive angles are measured anti-clockwise from x towards y.



$$x_{new} = x * \cos\theta - y * \sin\theta$$
$$y_{new} = x * \sin\theta + y * \cos\theta \qquad \text{———— (2.1)}$$

From the diagram:

$$x = r\cos\phi, \quad y = r\sin\phi \qquad \text{———— (2.2)}$$
$$x_{new} = r\cos(\theta+\phi) = r\cos\phi\cos\theta - r\sin\phi\sin\theta$$
$$y_{new} = r\sin(\theta+\phi) = r\cos\phi\sin\theta + r\sin\phi\cos\theta$$

Substituting for x and y from 2.2 we get 2.1



Homogeneous Coordinates

We have : $P_{new} = P + T$ Translation

$P_{new} = S * P$ Scaling

$P_{new} = R * P$ Rotation

Note the addition for Translation.

By expressing points as homogeneous coordinates all three affine transformations can be treated the same.

$\begin{vmatrix} x \\ y \end{vmatrix}$ In homogeneous coordinates becomes: $\begin{vmatrix} x \\ y \\ W \end{vmatrix}$

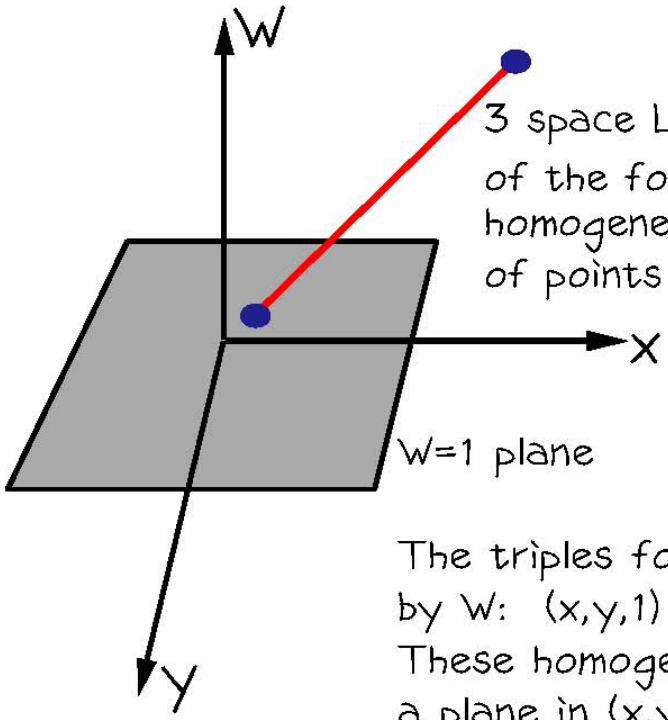
x,y,W represents the same point as x',y',W if one is a multiple of the other.
Thus (2,3,6) is the same as (4,6,12) since
2/6 = 4/12 and 3/6 = 6/12 (W != 0)

In particular: (x,y,W) is the same point as (x/W, y/W, 1) (W != 0)

Dividing the homogeneous point by W gives the Cartesian point (x/W, y/W)
Note the dimension change.



Homogeneous Representation of 2D transforms



3 space Line representing all the triples of the form (tx, ty, tW) t!=0 each 2D homogeneous point represents a line of points in 3 space.

The triples found by dividing by W: (x,y,1) represent points in 2 space. These homogenized points form a plane in (x,y,W) space where W=1



Homogeneous Representation of 2D transforms

continued

Points are now 3 element column vectors so we need 3x3 matrices to represent tranformations. The translation equation becomes:

$$\begin{vmatrix} x_{new} \\ y_{new} \\ 1 \end{vmatrix} = \begin{vmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{vmatrix} * \begin{vmatrix} x \\ y \\ 1 \end{vmatrix}$$

Column Vector post multiplication notation.

Matrices can be transposed to go from one convention to the other:

$$(P * M)^T = M^T * P^T$$



Column Vector vs. Row Vector

Column vectors take less space on the page in book format. However for the assignment premultiplying matrics as they are read from the user is easier than stacking the matrics. Use whichever convention you feel most comfortable with but be consistent.

Row Vector Representation – e.g. rotation around arbitrary point:

$$\begin{vmatrix} x_{new} & y_{new} & 1 \end{vmatrix} = \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ dx & dy & 1 \end{vmatrix} * \begin{vmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{vmatrix} * \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -dx & -dy & 1 \end{vmatrix} * \begin{vmatrix} x & y & 1 \end{vmatrix}$$

Column Vector Representation – e.g. rotation around arbitrary point:

$$\begin{vmatrix} x_{new} \\ y_{new} \\ 1 \end{vmatrix} = \begin{vmatrix} 1 & 0 & -dx \\ 0 & 1 & -dy \\ 0 & 0 & 1 \end{vmatrix} * \begin{vmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{vmatrix} * \begin{vmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{vmatrix} * \begin{vmatrix} x \\ y \\ 1 \end{vmatrix}$$



Homogeneous Representation of 2D transforms

continued

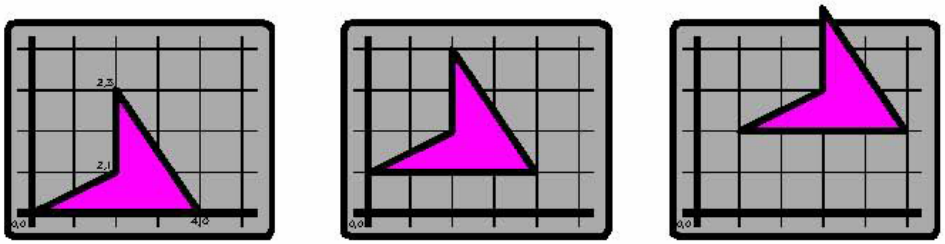
$$\begin{vmatrix} x_{new} \\ y_{new} \\ 1 \end{vmatrix} = \begin{vmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{vmatrix} * \begin{vmatrix} x \\ y \\ 1 \end{vmatrix}$$ Translation

$$\begin{vmatrix} x_{new} \\ y_{new} \\ 1 \end{vmatrix} = \begin{vmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{vmatrix} * \begin{vmatrix} x \\ y \\ 1 \end{vmatrix}$$ Scaling

$$\begin{vmatrix} x_{new} \\ y_{new} \\ 1 \end{vmatrix} = \begin{vmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{vmatrix} * \begin{vmatrix} x \\ y \\ 1 \end{vmatrix}$$ Rotation



Composing Geometric Transformations



E.g. Translate by (0,1) then translate by (1,1)

First translation T0 = $\begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{vmatrix}$

Next translation T1 = $\begin{vmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{vmatrix}$

P' = T0 * P

P'' = T1 * P' so P'' = T1 * T0 * P

Matrix product T1 * T0 is $\begin{vmatrix} 1 & 0 & 1 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{vmatrix}$

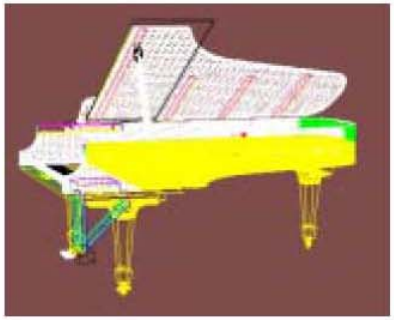


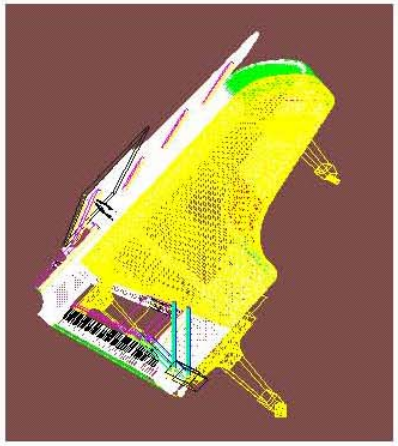
Composing Geometric Transformations

continued

Matrix product also known as concatanation, compounding or composition of matrices. Consider a set of n=1000000 points P representing the following piano mesh. The points are scaled by matrix S and rotated by matrix R and translated by matrix T.

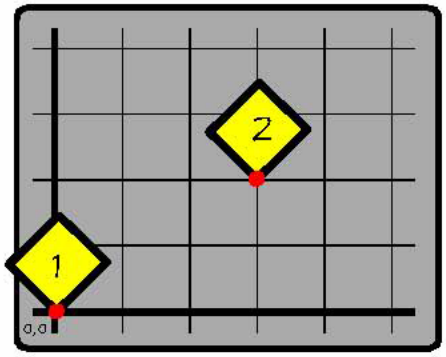
Number of vector matrix multiplies = n*S followed by n*R then n*T = 3n
Alternatively M = R*S*T and then n*M matrix vector calculations = 1n



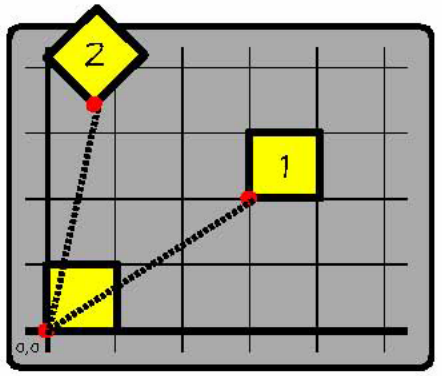




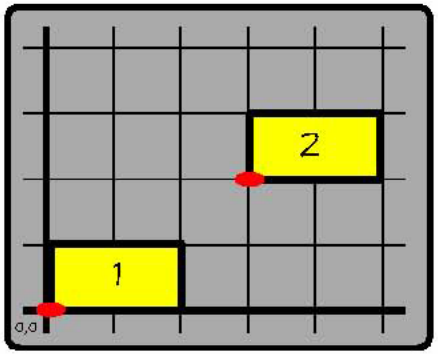
Order of Concatanation



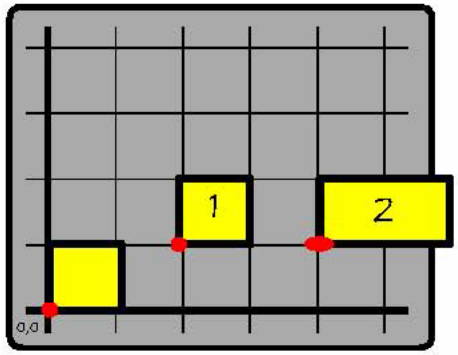
Start with a unit square at the origin.
Rotatate about the origin
translate to 3,2



Translate to 3,2 rotate
Rotate about the origin



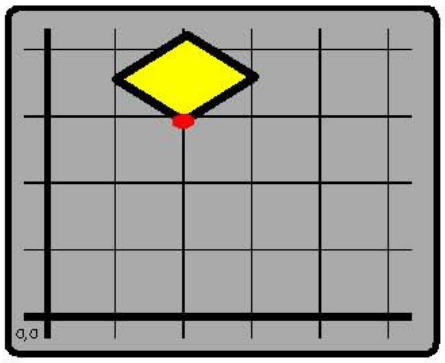
Scale about the origin by 2,1
translate to 3,2



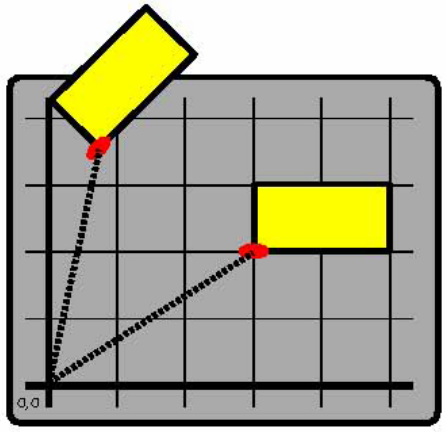
Translate to 2,1
Scale about the origin by 2,1



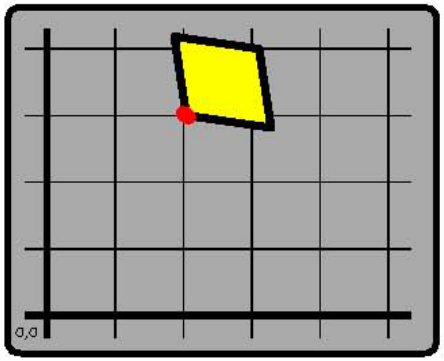
Order of Concatanation



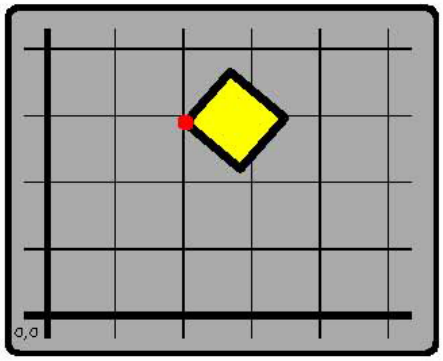
Rotate by 45,
scale by 2,1 translate to 2,3



Scale by 2,1 translate to 2,3
rotate by 45



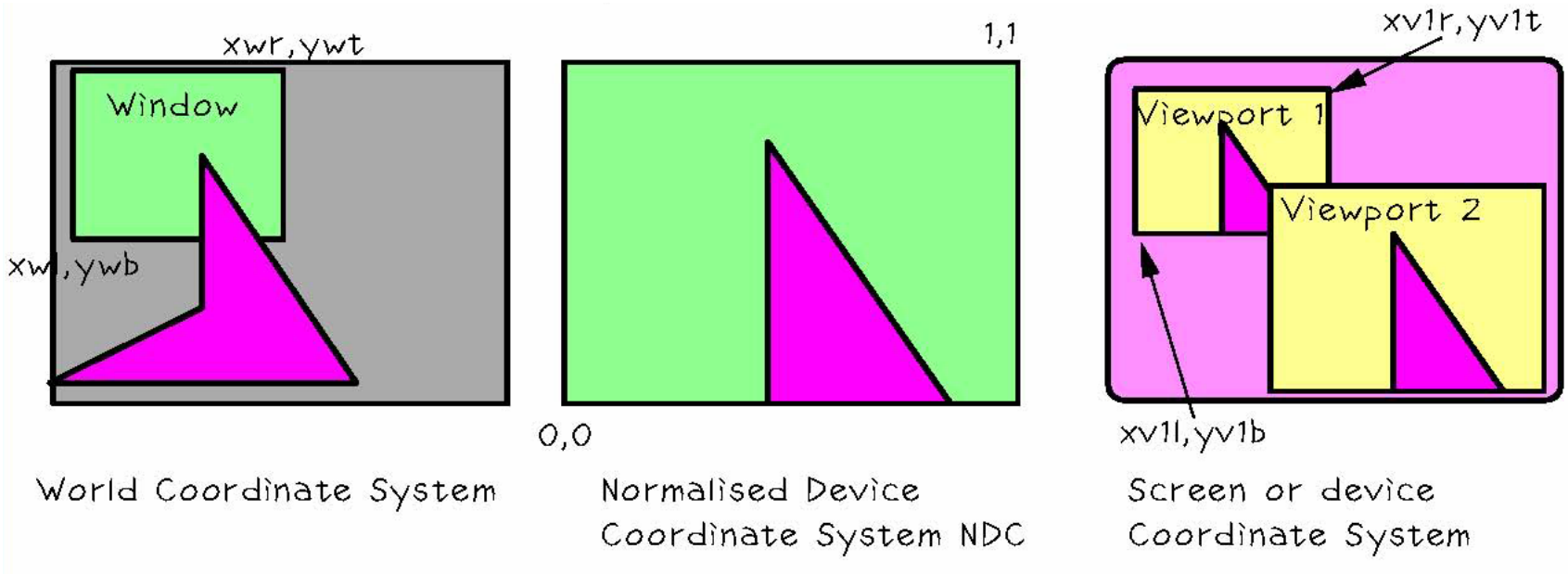
Rotate by 45, scale 2,1,
rotate by -45.
translate to 2,3



The square starts off at 2,3
What transformations are applied
to produce the rotated version above?



Window to Viewport Transformation



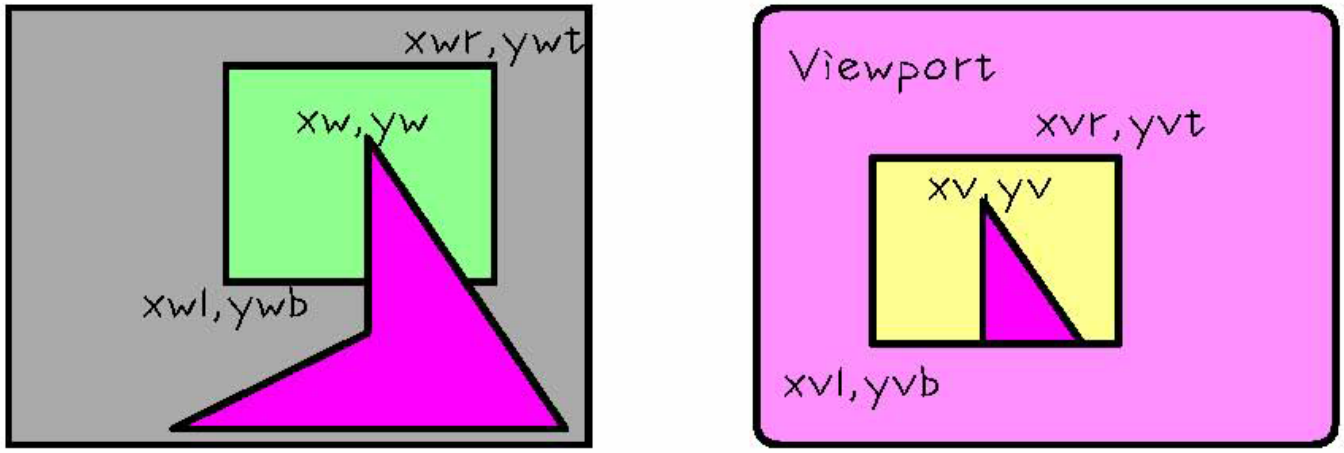
Windows in computer graphics refers to a rectangular area in the users own coordinate system known as the WORLD system. NOT window manager windows (invented much later and misnamed).

Viewport in computer graphics is a window manager window.

Normalised Device Coordinates are useful when displaying on many different devices.



Window to Viewport mapping



Simple mapping keeping scaling in x and y the same

ww = xwr − xwl wh = ywt − ywb arbitrary point in world: xw, yw
vw = xvr − xvl vh = yvt − yvb arbitrary point in viewport: xv, yv
scale = (ww/vw > wh/vh) ? wh/vh : ww/vw

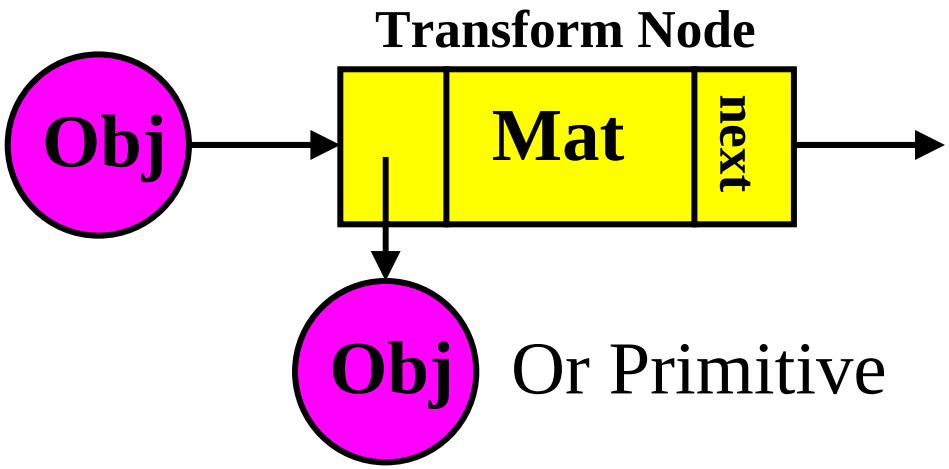
xv = (xw − xwl)*scale + xvl
yv = (yw − ywb)*scale + yvb

$$M = \begin{vmatrix} 1 & 1 & xvl \\ 0 & 1 & yvb \\ 0 & 0 & 1 \end{vmatrix} * \begin{vmatrix} scale & 1 & 0 \\ 0 & scale & 0 \\ 0 & 0 & 1 \end{vmatrix} * \begin{vmatrix} 1 & 1 & -xwl \\ 0 & 1 & -ywb \\ 0 & 0 & 1 \end{vmatrix}$$

$$\begin{vmatrix} xv \\ yv \\ 1 \end{vmatrix} = M * \begin{vmatrix} xw \\ yw \\ 1 \end{vmatrix}$$



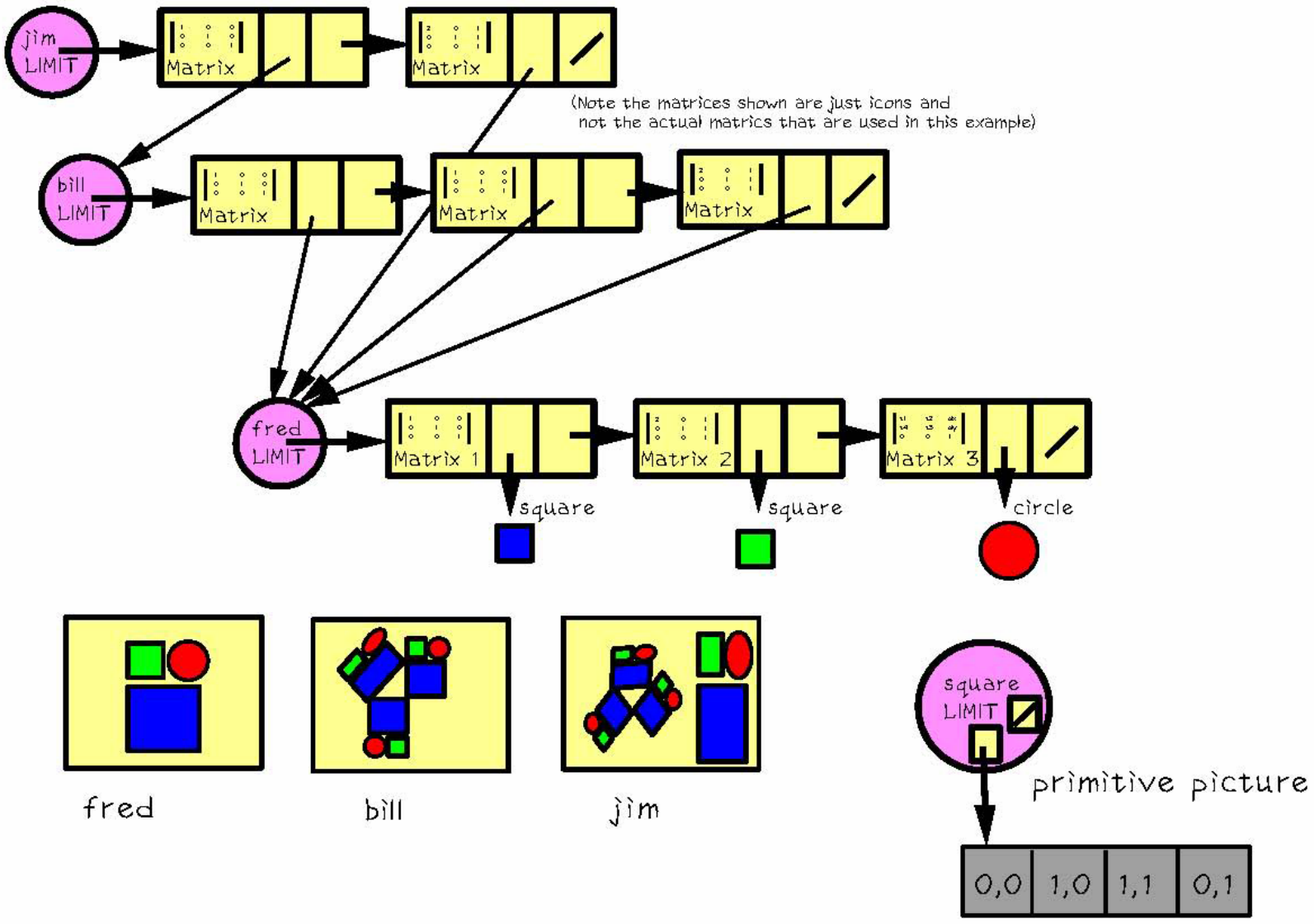
Instancing – The Scene Graph



Traverse structure for visualisation

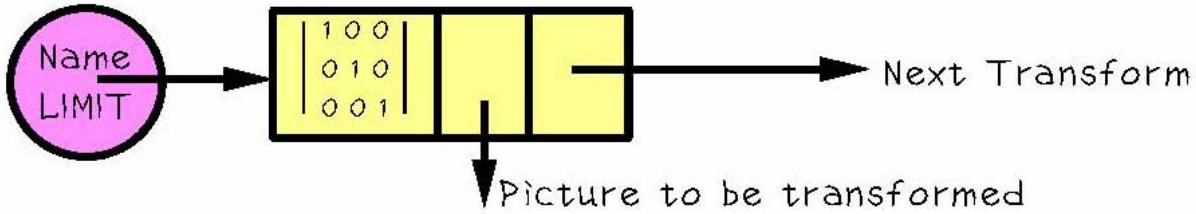


Example

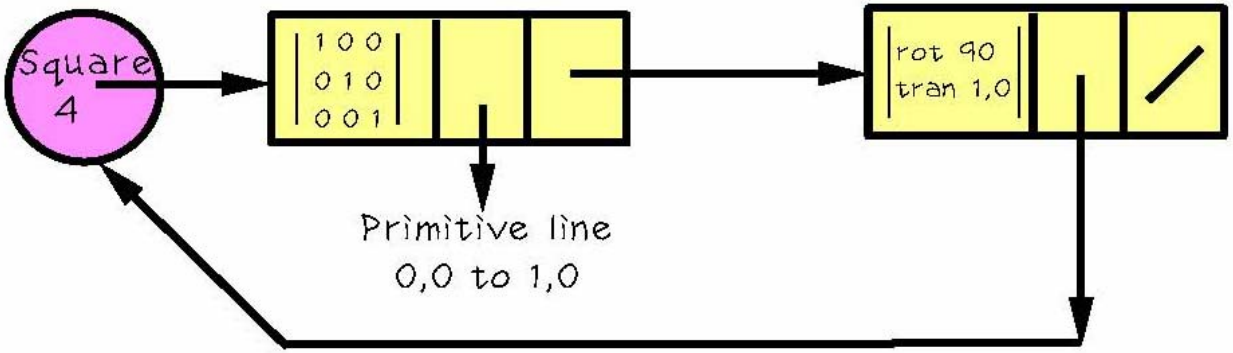

jim
LIMIT
Matrix
Matrix
(Note the matrices shown are just icons and
not the actual matrics that are used in this example)
bill
LIMIT
Matrix
Matrix
Matrix
fred
LIMIT
Matrix 1
Matrix 2
Matrix 3
square
square
circle
fred
bill
jim
square
LIMIT
primitive picture
0,0
1,0
1,1
0,1

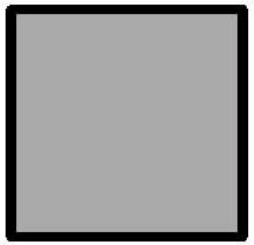
Recursive Hierarchies

Hierarchical Data Structure



E.g. Square



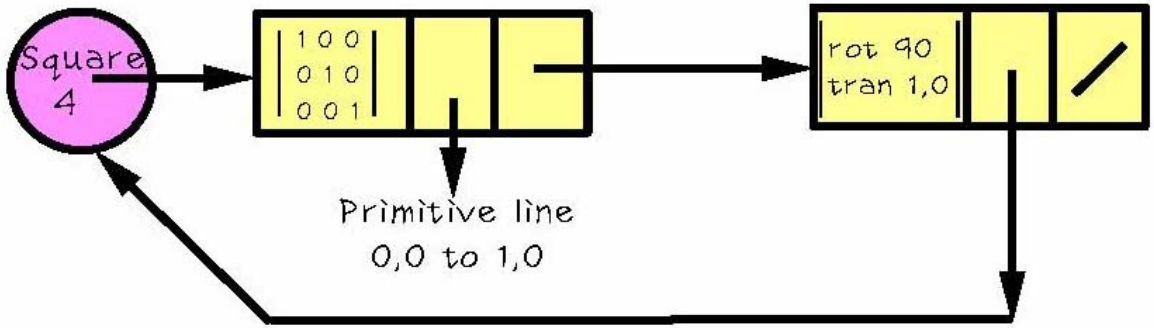


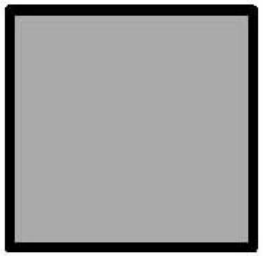


Traversal algorithm

```
(pseudo-code)
// matrix m is set to identity for scaling pass and to a scale and
// translate matrix for the window transform

plot(PICTURE* p, MATRIX m)
begin
  if (p IS  PRIMITIVE) output(p, m);
  else
   begin
     TRANSFORM t;
    t=p.firstTrans;
    if (p.limit>0)
       begin
         p.limit--;
         while (t != NULL)
            begin
                  plot(t.pic, matmult(m, t.m) );
              t=t.next;
            end  // while
          p.limit++;
        end // if
    end
end
```

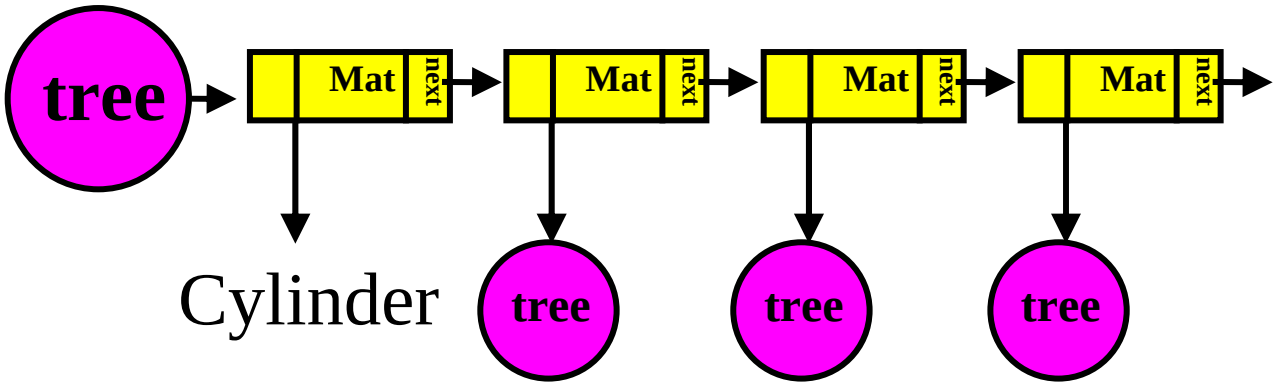


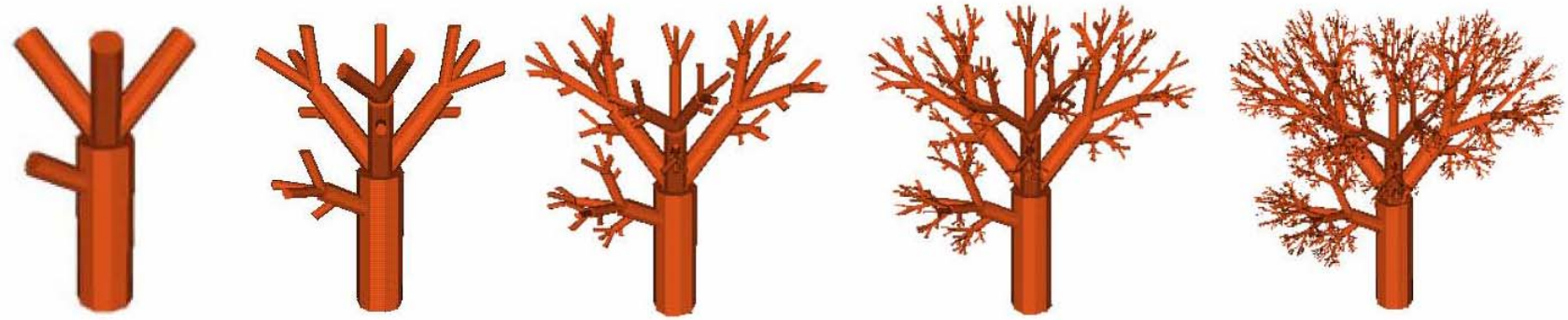


Note that Transform objects are initialised with an identity matrix.



Multiple Recursion





Local Limits

More on the Traversal algorithm

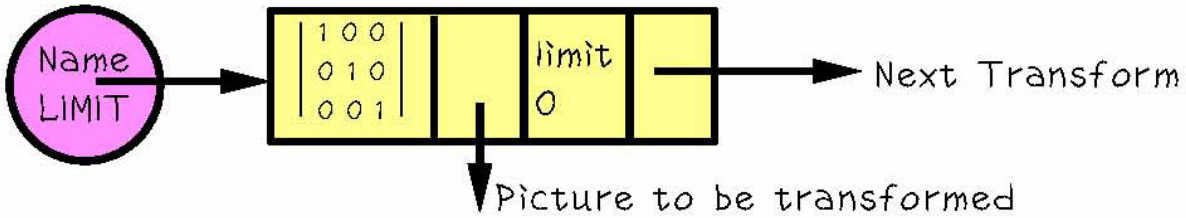
```
plot(PICTURE* p, MATRIX m)
begin
  if (p IS  PRIMITIVE) output(p, m);
  else
   begin
    TRANSFORM *t;
    t=p.firstTrans;
    if (p.limit>0)
       begin
         p.limit--;
         while (t != NULL)
            begin
               if ( p.limit == t.limit) plot(t.pic, postmult(m, t.m) );
              t=t.next;
            end  // while
          p.limit++;
        end // if
   end
end
```



Tree with leaves added but no local limit



Tree with leaves added using a local limit set to zero





Space Filling Curves

The Koch curve:

```
o koch
 lim 2
 a line scale 3 if 0
 a koch  scale 0.333333
 a koch  scale 0.333333  rot z 60  ori 1 0 0
 a koch  scale 0.333333  rot z -60 ori 1.5 0.8660254  0
 a koch  scale 0.333333  ori 2 0 0
 col black
c
o snow
 a koch
 a koch rot z 120 ori 3 0 0
 a koch scale 1 -1 1     rot z 60
c
```



Limit 1

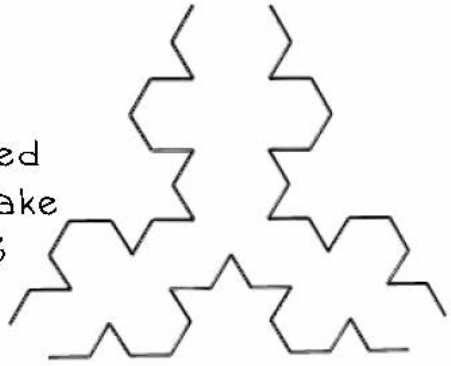


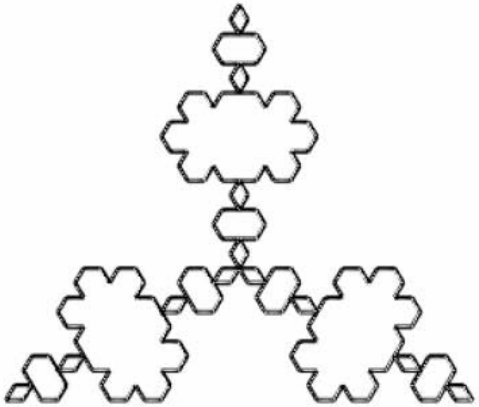
Limit 2



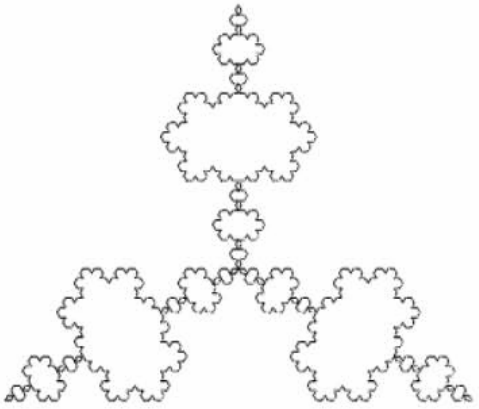
Limit 2

Exploded Snowflake Limit 3





Snowflake Limit 4



Snowflake Limit 5

