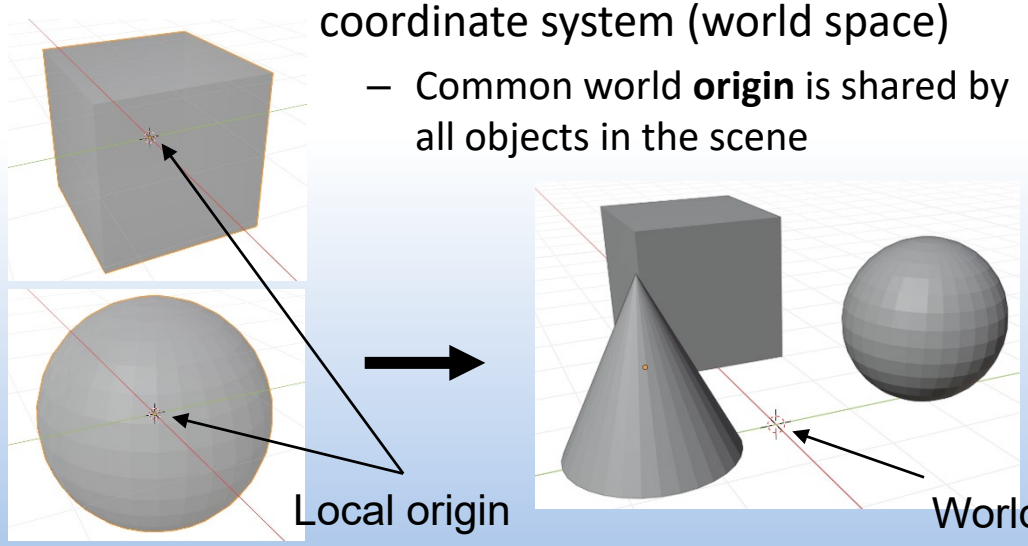
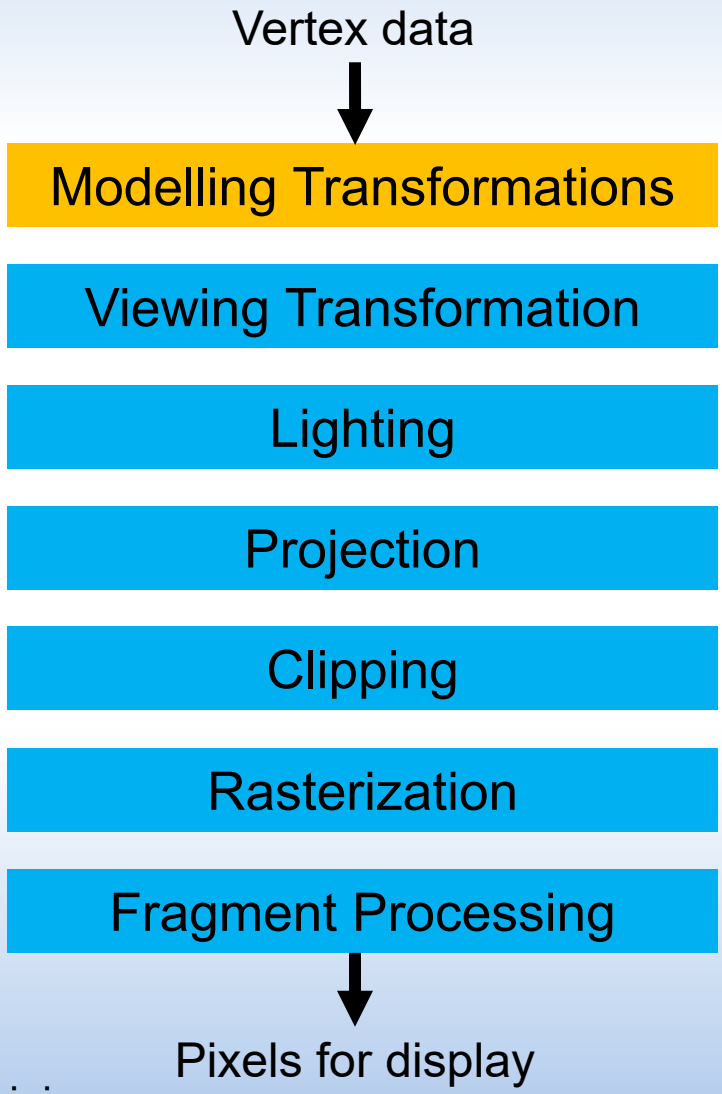
Transformations

The Computer Graphics Pipeline

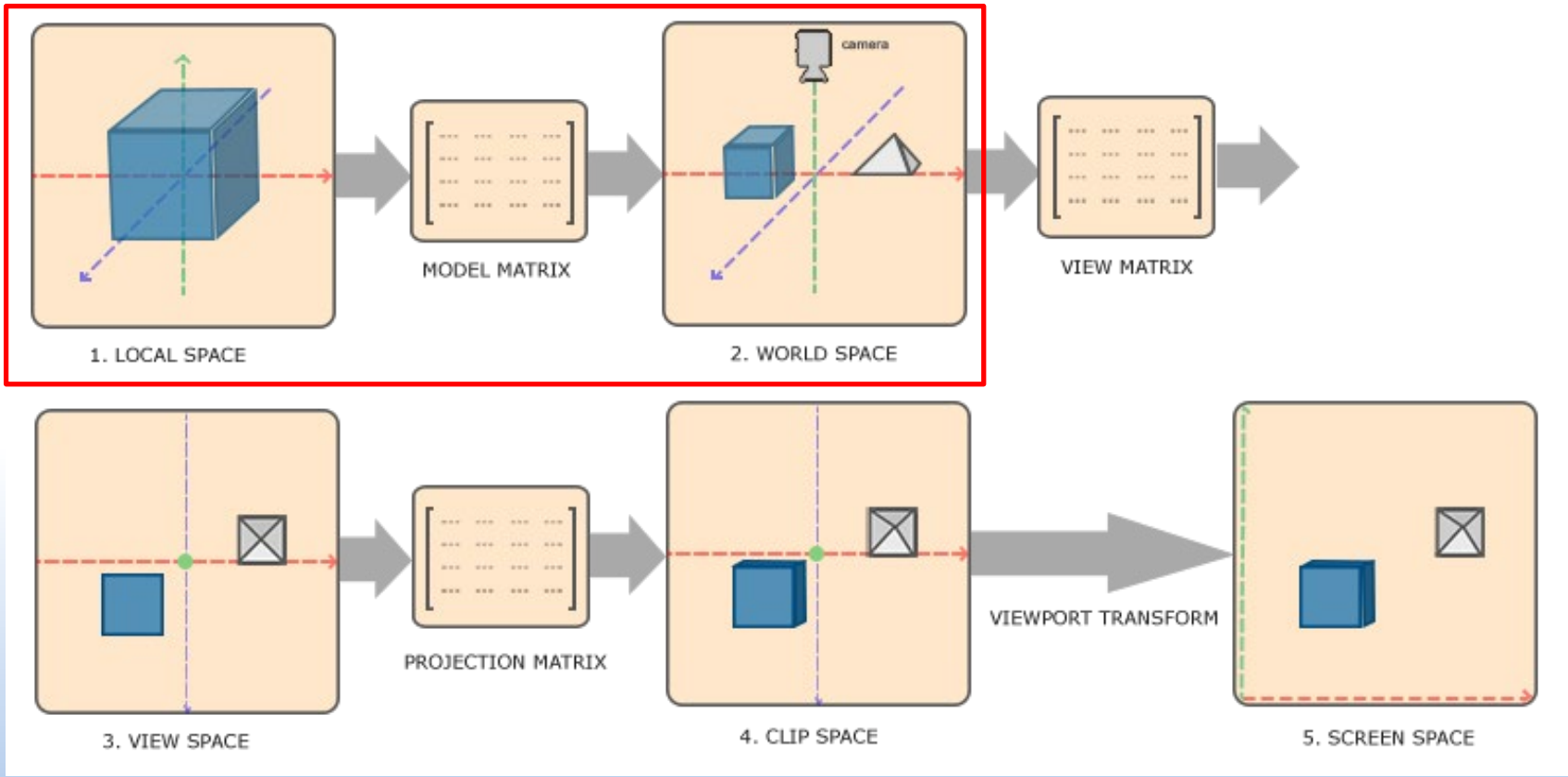
- Modelling transformations
  - **Model space** (or local space) to **world space**
    - Models typically defined in their own coordinate system
      - Each have their own local **origin**
    - Places objects in a common coordinate system (world space)
      - Common world **origin** is shared by all objects in the scene





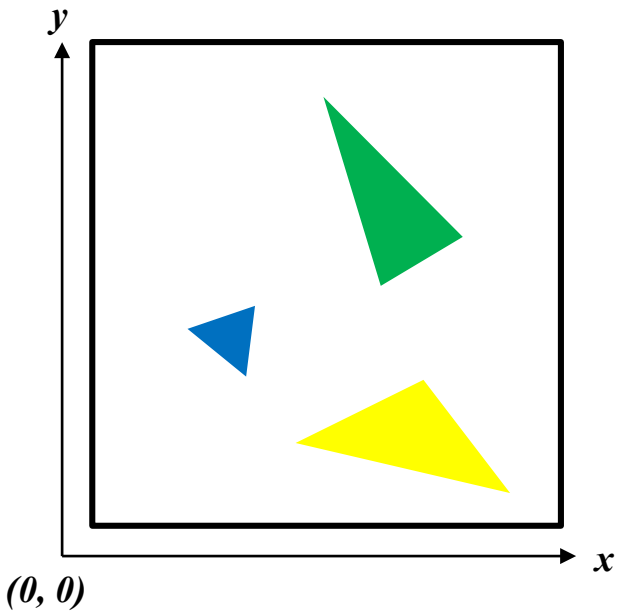
The Computer Graphics Pipeline

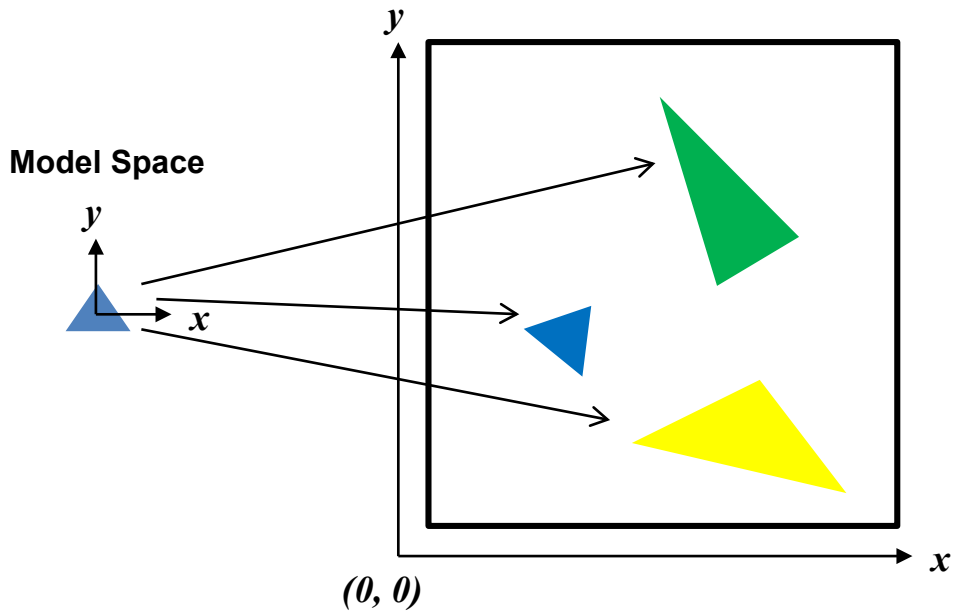
- Coordinate systems
  - Transformation matrices



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

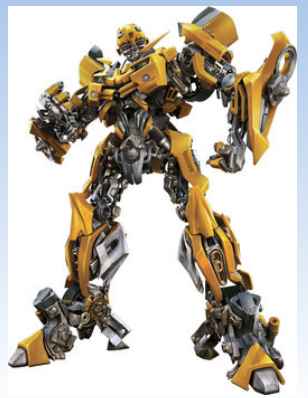
What we want…



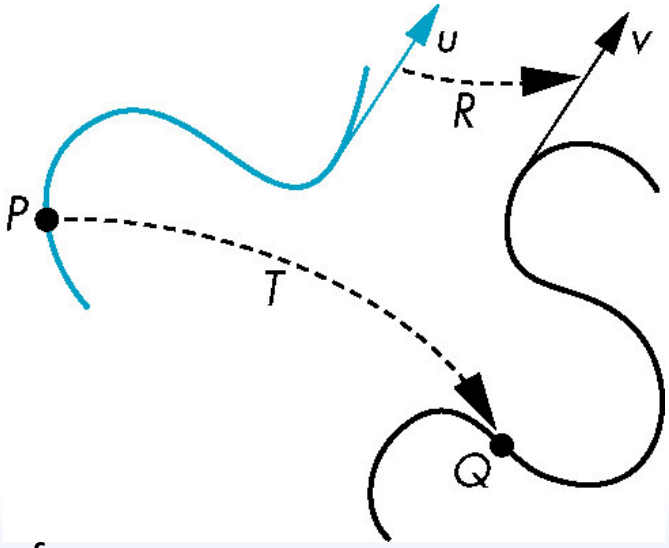


World Space
(Common to all objects, not just confined with the display area)

Transformations



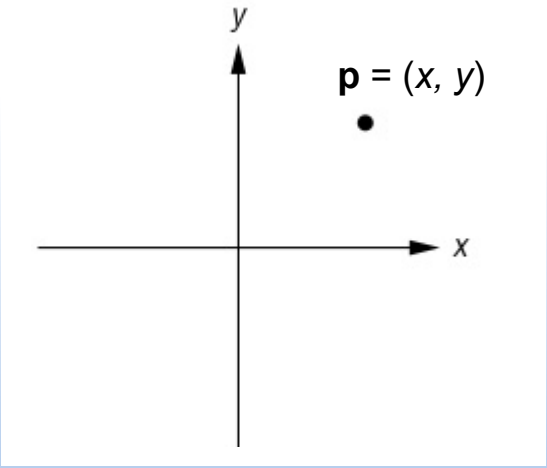
- What are transformations?
  - ➢ Process of mapping a point to another location
  - ➢ Modelling transformation
    - Operations applied to geometric description of an object
      - Change position, orientation, size
    - Affine transformations
      - Lines are preserved
        - » Allows us to transform endpoints of line segment
        - » Draw line segment between transformed endpoints

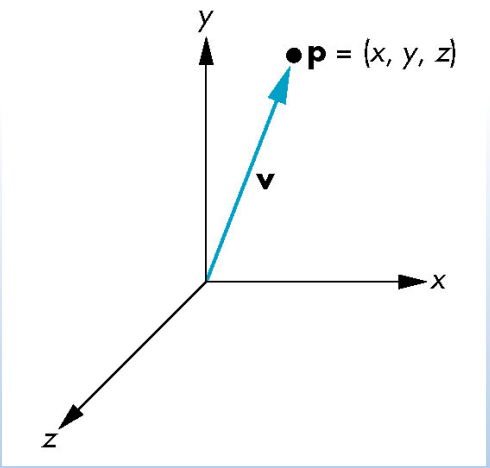


https://en.wikipedia.org/wiki/File:BumblebeeHIRES.jpg

Points and Vectors

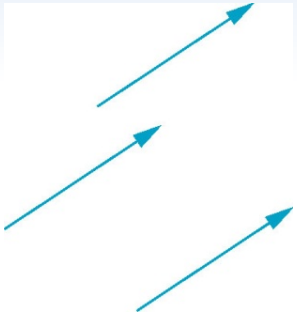
- Points
  - A location in space
    - Defined by coordinates in a space
  - Not sufficient in themselves
    - How do find the distance between points?
    - How do find the direction from one point to another?

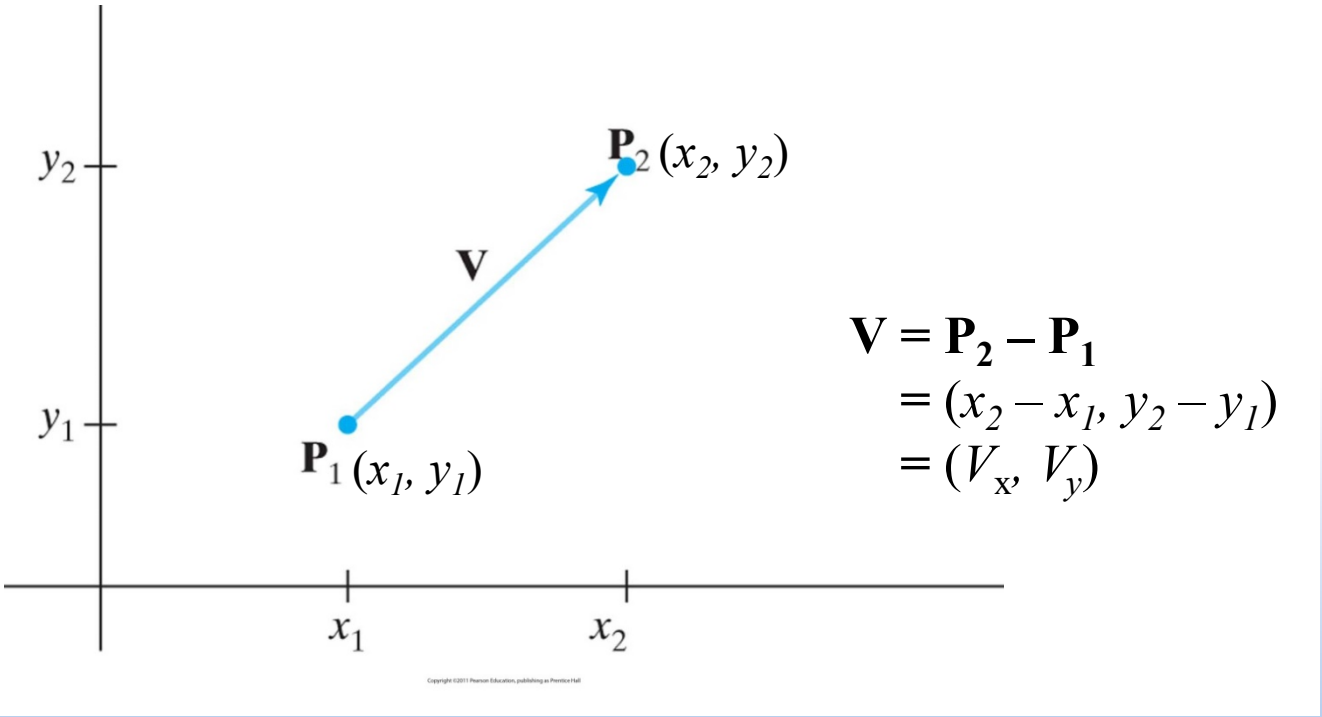




Vectors

- Vectors
  - Quantity with **magnitude** (length) and **direction**
  - No fixed position in space





$$\mathbf{V} = \mathbf{P_2} - \mathbf{P_1}$$
$$= (x_2 - x_1,\ y_2 - y_1)$$
$$= (V_x,\ V_y)$$

Vectors

- **IMPORTANT NOTE**
  - The term "**vector**" in mathematics are not the same as its use in programming
    - In programming, vectors typically refer to sequential containers for storing data, e.g., an array
    - Can use a programming vector to represent an actual mathematical vector, but often not the case
      - Can represent colour, coordinates, etc.
- OpenGL Mathematics (GLM) library
  - Classes and functions designed and implemented with the same naming conventions and functionalities as GLSL

Vectors

- Vector datatype (GLM and GLSL)
  - Dimensions 2, 3 or 4
  - Typically used for coordinates, vectors, colours
    - GLM

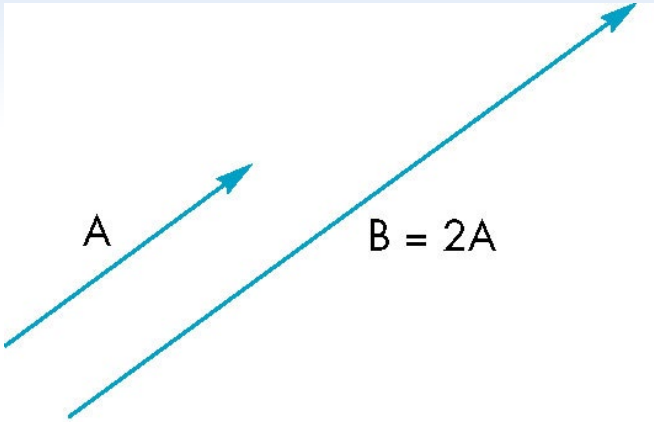
```
#include <glm/glm.hpp>     // include the GLM library
using namespace glm;       // to avoid having to use glm::
glm::ivec2 vectorA;        // integer vector
glm::uvec3 vectorB;        // unsigned integer vector
glm::vec4 vectorC;         // floating point vector
```

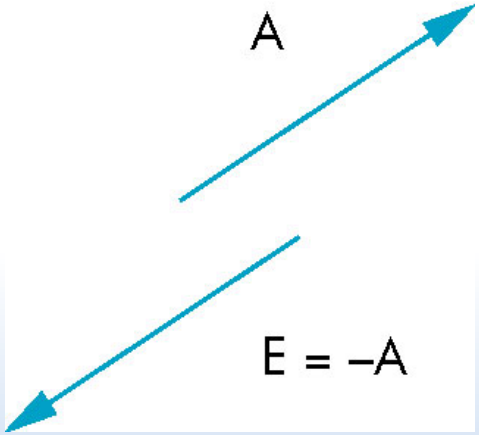
    - GLSL

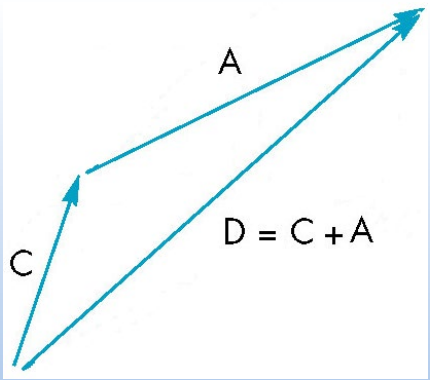
```
ivec2 vectorA;              // integer vector
uvec3 vectorB;             // unsigned integer vector
vec4 vectorC;              // floating point vector
```

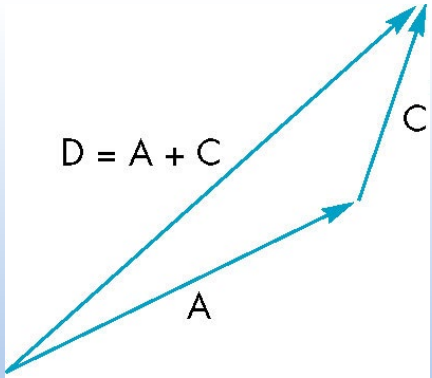
Vectors

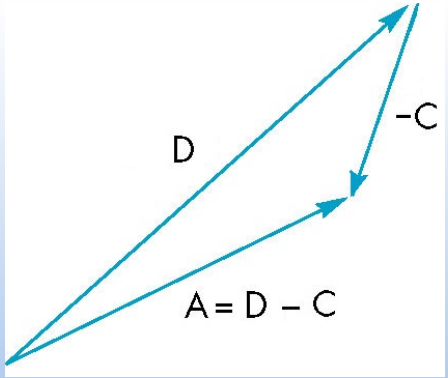
- Vector operations
  - ➢ Multiply by scalar
    - Alters magnitude
  - ➢ Inverse/negate
    - Same magnitude, opposite direction
  - ➢ Addition/subtraction
    - Combine directed line segments using head-to-tail rule











Vectors

- GLM

```
glm::vec3 A(1.0f, 2.0f, 3.0f);                  // initialise
glm::vec3 B = glm::vec3(1.0f, 2.0f, 3.0f);   // assign
glm::vec3 C = -A + B;          // negation and addition
glm::vec3 D = 2 * C;           // scalar multiplication
glm::vec4 E(D, 1.0f);          // vec4 from vec3 and a scalar
glm::vec3 F = glm::vec3(E);    // vec3 from vec4
cout << D.x << " " << D.y << endl;    // accessing components
```

- GLSL

```
vec3 A = vec3(1.0f, 2.0f, 3.0f);      // initialise
vec3 B = -A;                   // negation
vec3 C = A + B;                // addition
vec3 D = 2 * C;                // scalar multiplication
vec4 E = vec4(D, 1.0f);        // vec4 from vec3 and a scalar
vec3 F = E.xyz;                // vec3 from vec4
vec4 G = E.zxxw;               // “swizzling”, swap order
```

Vectors

- Vector operations
  - Magnitude of a vector

```
length(vectorA);
```

$$|\vec{v}| = \sqrt{{v_1}^2 + {v_2}^2 + \cdots + {v_n}^2}$$

  - Normalisation
    - Obtain a vector in the same direction, but of unit vector (i.e. vector of length = 1)

```
normalize(vectorA);
```

$$\hat{\vec{v}} = \frac{\vec{v}}{|\vec{v}|}, \vec{v} \neq 0$$

  - Distance between two points
    - GLM only

```
glm::distance(pointA, pointB);
```

    - Essentially

```
length(pointA - pointB);
```

$$dist = |A - B|$$

Vectors

- Vector operations
  - Dot product (or scalar product)

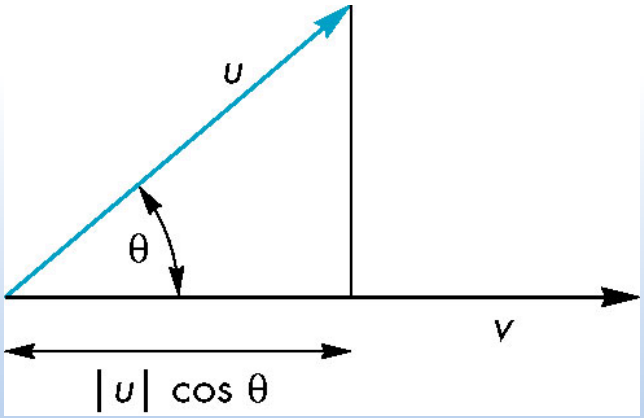
```
dot(vectorA, vectorB);
```

$$\vec{u} \cdot \vec{v} = \sum_{i=1}^{n} u_i v_i = u_1 v_1 + u_2 v_2 + \cdots + u_n v_n$$

- For example
  - $(u_x, u_y, u_z) \bullet (v_x, v_y, v_z) = u_x v_x + u_y v_y + u_z v_z$

$$\vec{u} \cdot \vec{v} = |\vec{u}||\vec{v}| \cos\theta$$

$$\theta = \cos^{-1}\left(\frac{\vec{u} \cdot \vec{v}}{|\vec{u}||\vec{v}|}\right)$$

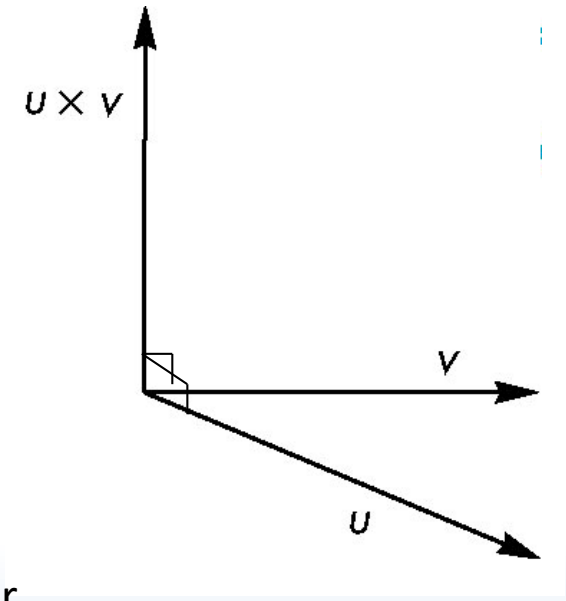


Vectors

- Vector operations
  - Cross product
    - 3D vectors only
    - Cross product results in a vector perpendicular to the plane containing the input vectors

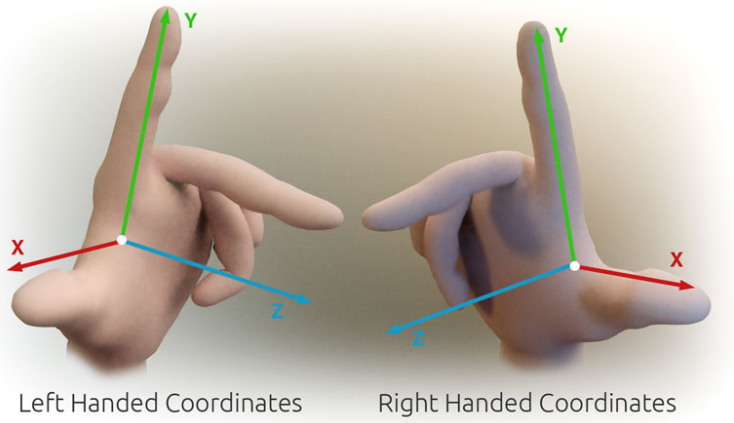
```
cross(vectorA, vectorB);
```

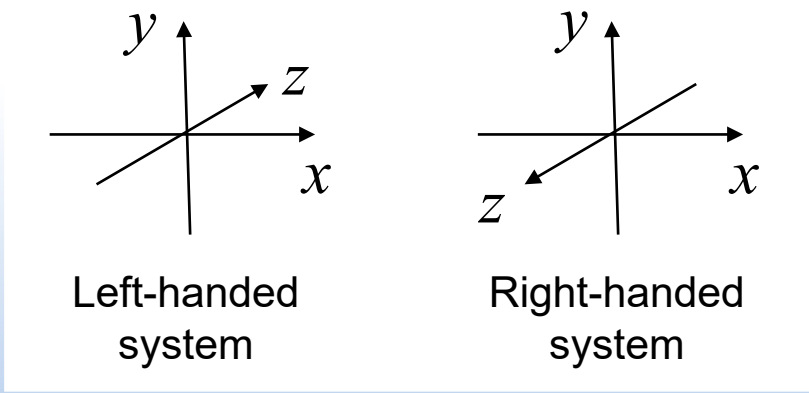
    - Direction based on “handedness” of the coordinate system
      - Thumb points in direction of resulting vector



Handedness of a 3D Coordinate System

- "Handedness" of a 3D system
  - ➢ Determined by
    - Thumb: +*x*-axis
    - Index finger: +*y*-axis
    - Middle finger: +*z*-axis
  - ➢ OpenGL
    - Right-handed system
  - ➢ DirectX
    - Left-handed system





https://upload.wikimedia.org/wikipedia/commons/b/b2/3D_Cartesian_Coodinate_Handedness.jpg

Matrices

- Matrices
  - ➢ Rectangular grid of numbers arranged in rows and columns
    - Dimensions = rows x columns
  - ➢ Square matrices
    - Same number of rows and columns
    - Diagonal elements
      - Elements where row and column index the same
    - **Identity** matrix
      - Diagonal elements = 1, all others 0
      - In some ways, what 1 is for scalars
        - » $\mathbf{M M^{-1} = M^{-1} M = I}$
        - » **MI = IM = M**

$$\mathrm{M}_{3\times3} = \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix}$$

$$\mathrm{I} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Matrices

- Matrix addition
  - ➢ Component-wise addition
    - Example

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} + \begin{bmatrix} 2 & 3 \\ 4 & 5 \end{bmatrix} = \begin{bmatrix} 3 & 5 \\ 7 & 9 \end{bmatrix}$$

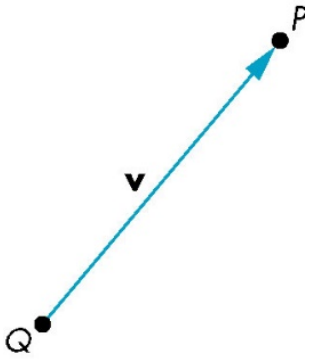
- Scalar multiplication
  - ➢ Component-wise multiplication with a scalar
    - Example

$$2 \cdot \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 2 \times 1 & 2 \times 2 \\ 2 \times 3 & 2 \times 4 \end{bmatrix} = \begin{bmatrix} 2 & 4 \\ 6 & 8 \end{bmatrix}$$

Point-vector addition

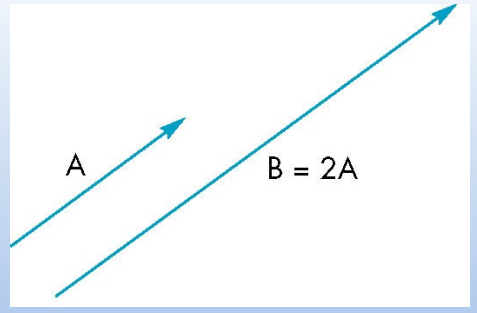
$$P = Q + \boldsymbol{v}$$

$$\begin{bmatrix} P_x \\ P_y \end{bmatrix} = \begin{bmatrix} Q_x \\ Q_y \end{bmatrix} + \begin{bmatrix} v_x \\ v_y \end{bmatrix}$$



Scaling a vector

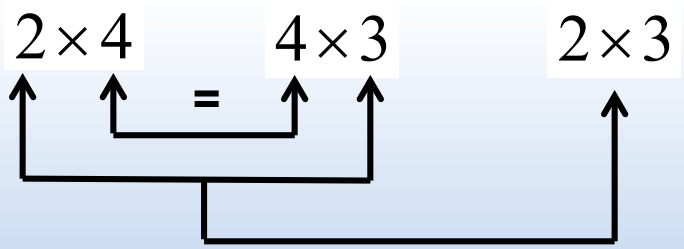
$$B = 2 \cdot A = \begin{bmatrix} 2A_x \\ 2A_y \end{bmatrix}$$



Matrices

- Matrix multiplication
  - Can only multiply matrix A with matrix B if
    - The number of **columns** in A **is equal** to the number of **rows** in B
    - The result is a matrix with the number of **rows** in A x the number of **columns** in B

$$\begin{bmatrix} ? & ? & ? & ? \\ ? & ? & ? & ? \end{bmatrix} \begin{bmatrix} ? & ? & ? \\ ? & ? & ? \\ ? & ? & ? \\ ? & ? & ? \end{bmatrix} = \begin{bmatrix} ? & ? & ? \\ ? & ? & ? \end{bmatrix}$$



Matrices

- Matrix multiplication
  - Generalisation of the vector dot product
    - Sums of the products of the elements in the row vectors of A with the corresponding elements in the column vectors of B

$$\mathrm{AB} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}$$

$$\mathrm{AB} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{bmatrix}$$

$$\mathrm{AB} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{bmatrix}$$

$$\mathrm{AB} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{bmatrix}$$

$$\mathrm{AB} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{bmatrix}$$

Matrices

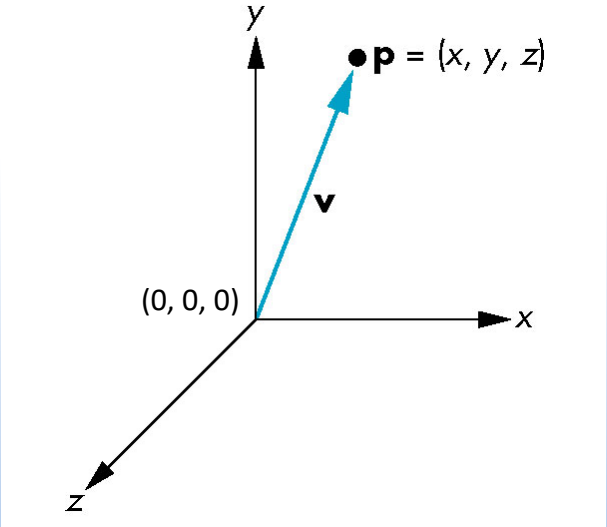
- GLM and GLSL
  - Matrices defined in column-major order
    - `mat`*`n`*`x`*`m`* : a matrix with n columns and m rows
    - `mat`*`n`* : a matrix with n columns and n rows
      - n and m can be 2, 3, or 4
  - For example

```
mat3 identityMatrix = mat3(1.0f);

mat3 theMatrix = mat3(0.0f, 1.0f, 2.0f, 3.0f, 4.0f, 5.0f,
       6.0f, 7.0f, 8.0f);
theMatrix[1] = vec3(10.0f, 11.0f, 12.0f);
theMatrix[2][0] = 13.0f;
```

$$\begin{bmatrix} 0 & 3 & 6 \\ 1 & 4 & 7 \\ 2 & 5 & 8 \end{bmatrix} \rightarrow \begin{bmatrix} 0 & 10 & 13 \\ 1 & 11 & 7 \\ 2 & 12 & 8 \end{bmatrix}$$

Homogenous Coordinates

- **Homogeneous coordinates**
  - Key to all computer graphics systems
    - Potential **confusion** between representation of a **vector** and a **point**
    - Homogeneous coordinates avoids this difficulty by using **another dimension** to represent points and vectors



$$\mathbf{p} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

$$\mathbf{v} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Homogenous Coordinates

- Points and vectors
  - Both represented by (x, y, z)
  - Add a 4th coordinate, (x, y, z, w)
    - **w = 1 (or non-zero)** for **points:** $\mathbf{P} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$
    - In vertex shader

```
gl_Position = vec4(aPosition, 1.0f);
```

    - **w = 0** for **vectors:** $\mathbf{V} = \begin{bmatrix} x \\ y \\ z \\ 0 \end{bmatrix}$

Homogenous Coordinates

- Points
  - To recover original position
    - $(x \quad y \quad z \quad w) \rightarrow \left(\frac{x}{w} \quad \frac{y}{w} \quad \frac{z}{w}\right)$
- Matrices
  - All standard transformations can be implemented with matrix multiplications using 3 x 3 or 4 x 4 matrices

These refer to the same point:

$$\begin{bmatrix}\mathbf{1}\\ \mathbf{2}\\ \mathbf{3}\\ \mathbf{1}\end{bmatrix} \quad \begin{bmatrix}\mathbf{2}\\ \mathbf{4}\\ \mathbf{6}\\ \mathbf{2}\end{bmatrix} \quad \begin{bmatrix}\mathbf{5}\\ \mathbf{10}\\ \mathbf{15}\\ \mathbf{5}\end{bmatrix}$$

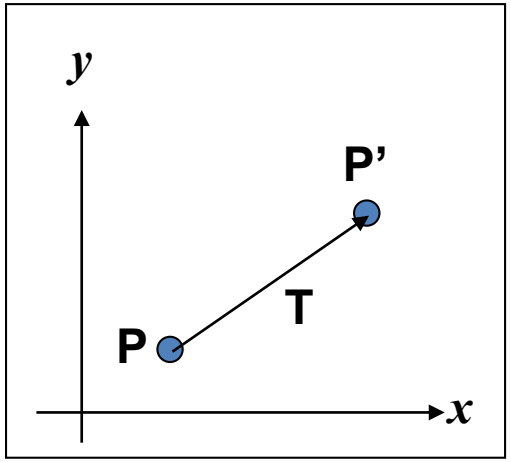
$$\mathrm{M}_{3\times3} = \begin{bmatrix} m_{11} & m_{12} & 0 \\ m_{21} & m_{22} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

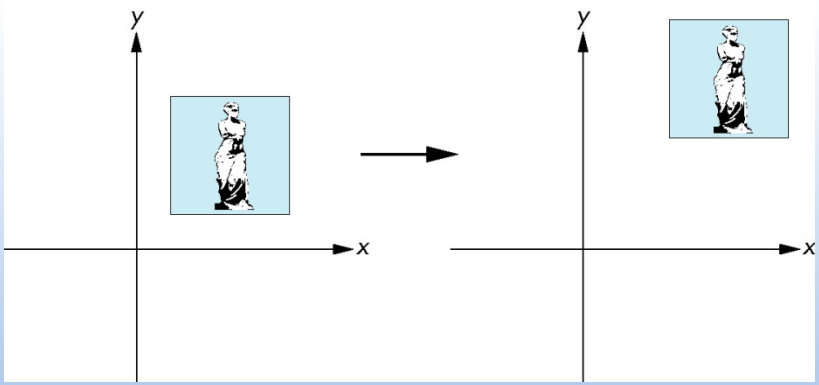
$$\mathrm{M}_{4\times4} = \begin{bmatrix} m_{11} & m_{12} & 0 & 0 \\ m_{21} & m_{22} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathrm{M}_{4\times4} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & 0 \\ m_{21} & m_{22} & m_{23} & 0 \\ m_{31} & m_{32} & m_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2D Transformations

- Translation
  - Move a point to a new location
    - Move original point along a straight-line to its new location
  - An object is defined by multiple coordinate positions
    - Relocate all coordinate positions by same displacement along parallel paths

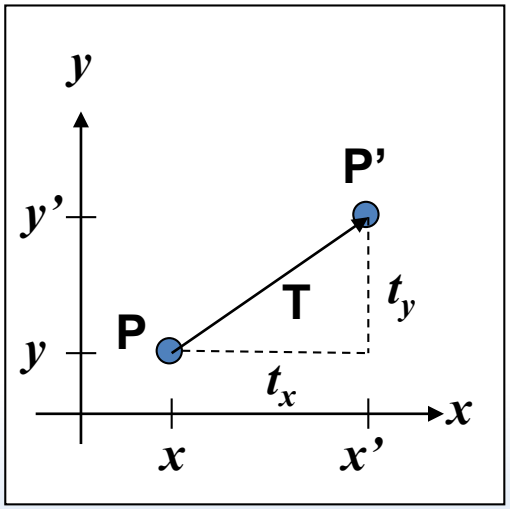




2D Transformations

- Translation
  - Translate original coordinates (*x, y*) by translational distances t_x and t_y to obtain new coordinate position (*x', y'*)

$$x' = x + t_x \quad y' = y + t_y$$



  - Equation in matrix form

$$P' = P + T$$

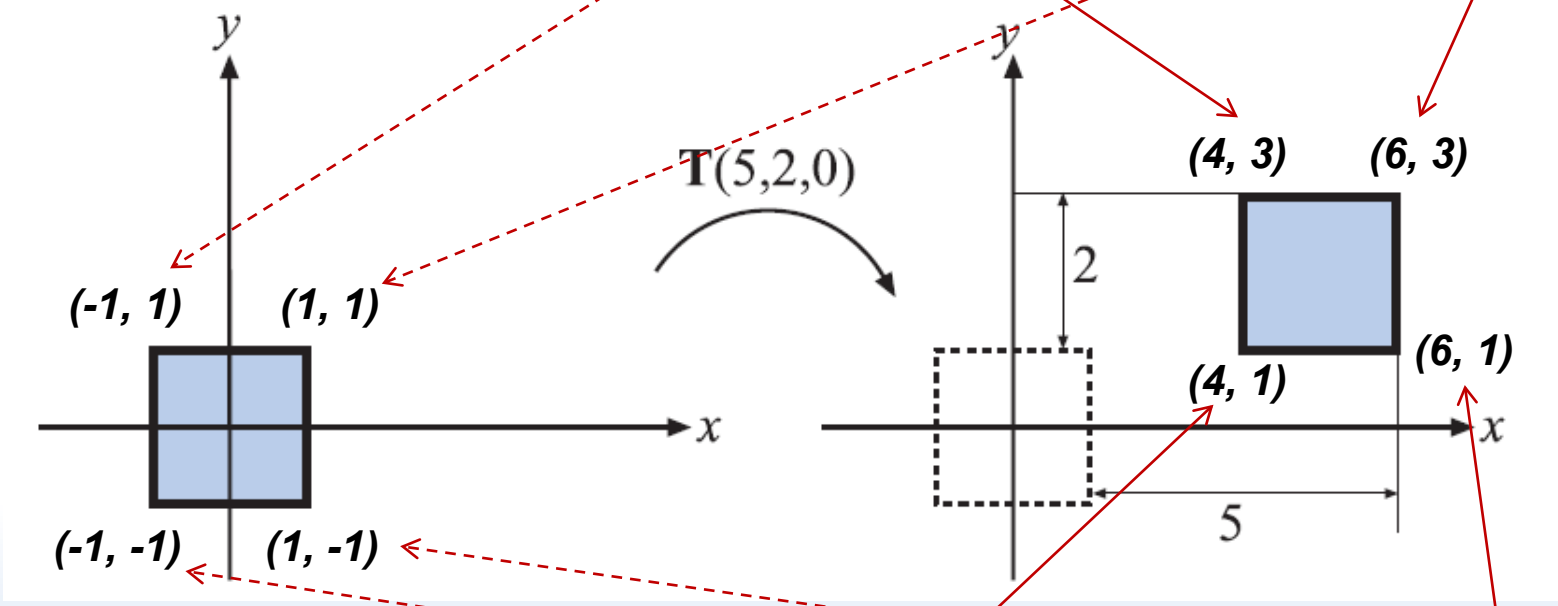
$$P' = \begin{bmatrix} x' \\ y' \end{bmatrix} \quad P = \begin{bmatrix} x \\ y \end{bmatrix} \quad T = \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

P' – New position
P – Original position
T – Translational vector

2D Transformations

$$P_1' = \begin{bmatrix} -1 \\ 1 \end{bmatrix} + \begin{bmatrix} 5 \\ 2 \end{bmatrix} = \begin{bmatrix} 4 \\ 3 \end{bmatrix}$$

$$P_4' = \begin{bmatrix} 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 5 \\ 2 \end{bmatrix} = \begin{bmatrix} 6 \\ 3 \end{bmatrix}$$



$$P_2' = \begin{bmatrix} -1 \\ -1 \end{bmatrix} + \begin{bmatrix} 5 \\ 2 \end{bmatrix} = \begin{bmatrix} 4 \\ 1 \end{bmatrix}$$

$$P_3' = \begin{bmatrix} 1 \\ -1 \end{bmatrix} + \begin{bmatrix} 5 \\ 2 \end{bmatrix} = \begin{bmatrix} 6 \\ 1 \end{bmatrix}$$

2D Transformations

- Scaling
  - Alter the size of an object
    - Expand or contract along each axis
  - Simple scaling operation
    - Multiply original object positions (*x, y*), by scaling factor s_x and s_y to produce scaled coordinates *(x', y')*

$$x' = x \times s_x \quad y' = y \times s_y$$

    - Equation in matrix form

$$P' = S\,P$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

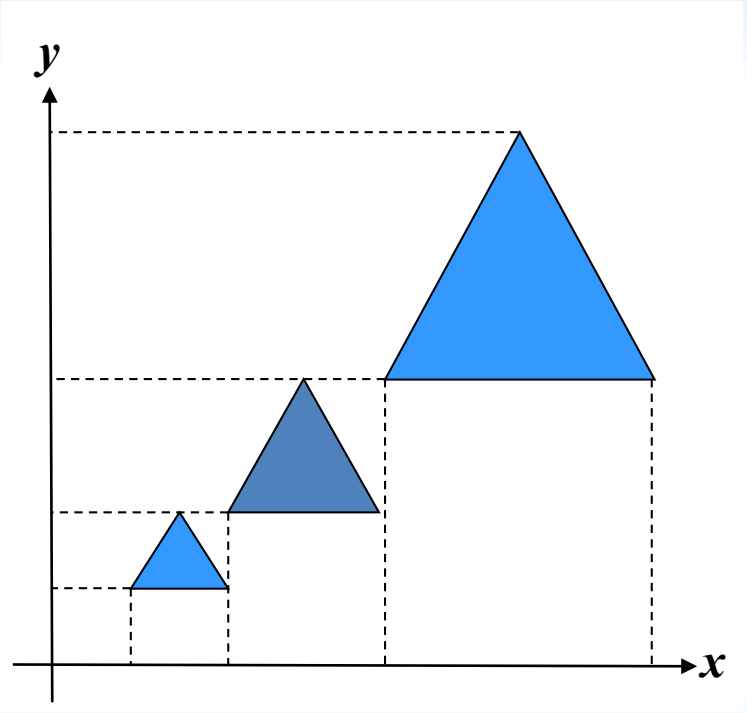
P' – Scaled coordinates
P – Original position
S – Scaling matrix

2D Transformations

- Scaling
  - ➢ Scaling factors
    - s = 1, no change in size
    - 0 < s < 1, reduces size
    - s > 1, increases size
    - Uniform scaling maintains relative object proportions
      - Equal scaling factors
      - As opposed to differential scaling (unequal scaling factors)
    - s < 0, negative values not only resize the object but **reflect** it about the respective coordinate axes

2D Transformations

- Scaling
  - Objects transformed using simple scaling both scaled and repositioned
    - Scaling factors with absolute values less than 1 move objects closer to coordinate origin
    - Values greater than 1 move objects farther



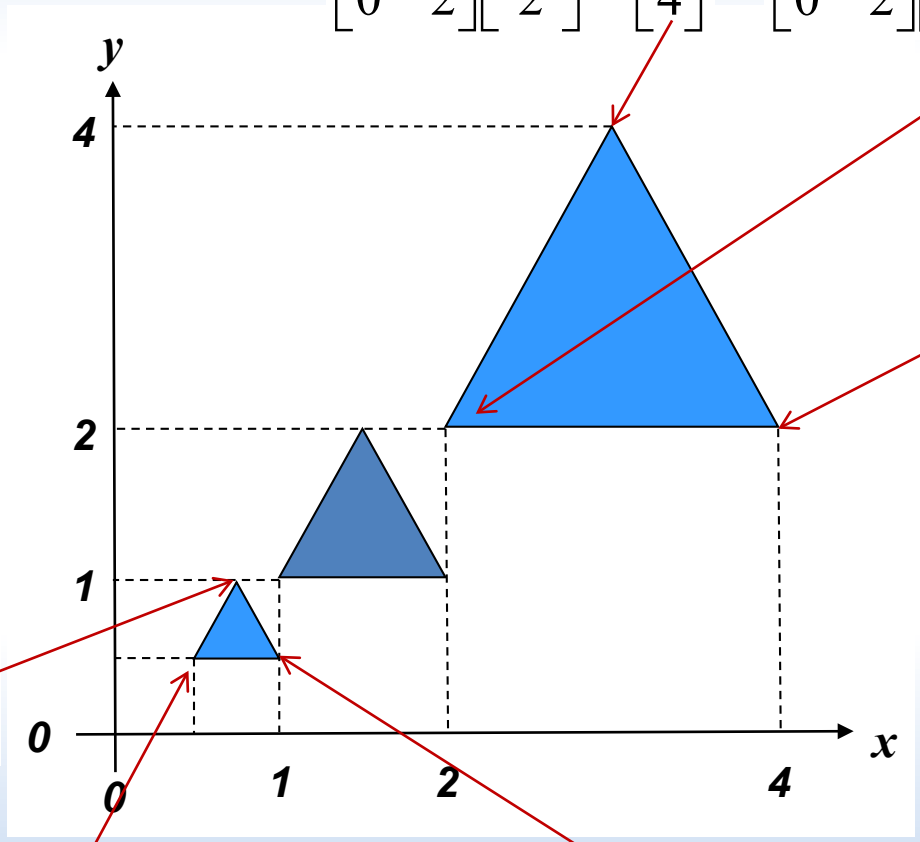
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

2D Transformations

$$\begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}\begin{bmatrix} 1.5 \\ 2 \end{bmatrix} = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$$

$$\begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}\begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$$

$$\begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}\begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 4 \\ 2 \end{bmatrix}$$



$$\begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}\begin{bmatrix} 1.5 \\ 2 \end{bmatrix} = \begin{bmatrix} 0.75 \\ 1 \end{bmatrix}$$

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

$$\begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}\begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0.5 \end{bmatrix}$$

2D Transformations

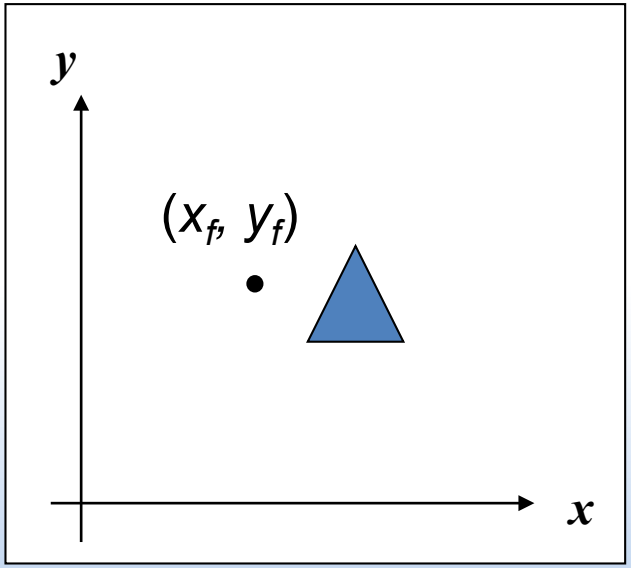
- Scaling
  - ➢ Can control the location of scaled object
    - By choosing a position, called the fixed point that is to remain unchanged after the scaling
    - Often the centre of the object

$$x' - x_f = (x - x_f)s_x$$

$$y' - y_f = (y - y_f)s_y$$

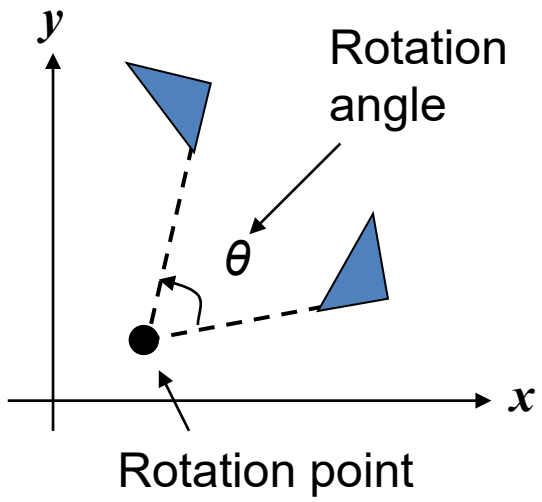
$$x' = x \times s_x + x_f(1 - s_x)$$

$$y' = y \times s_y + y_f(1 - s_y)$$



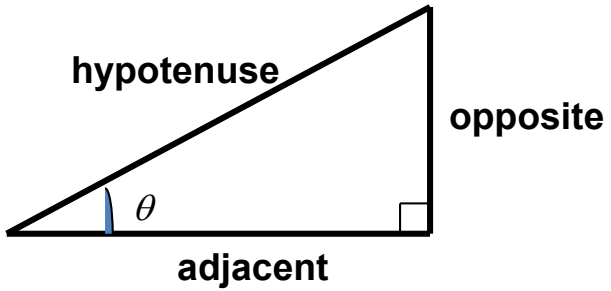
2D Transformations

- 2D rotation
  - In 2D, rotation about an axis perpendicular to *xy* plane (parallel to *z*-axis)
    - Rotation about a pivot point
    - Positive angle θ defines counter clockwise rotation
  - For objects
    - Every point rotated through same angle
    - Rigid-body, every point moved without deformation



2D Transformations

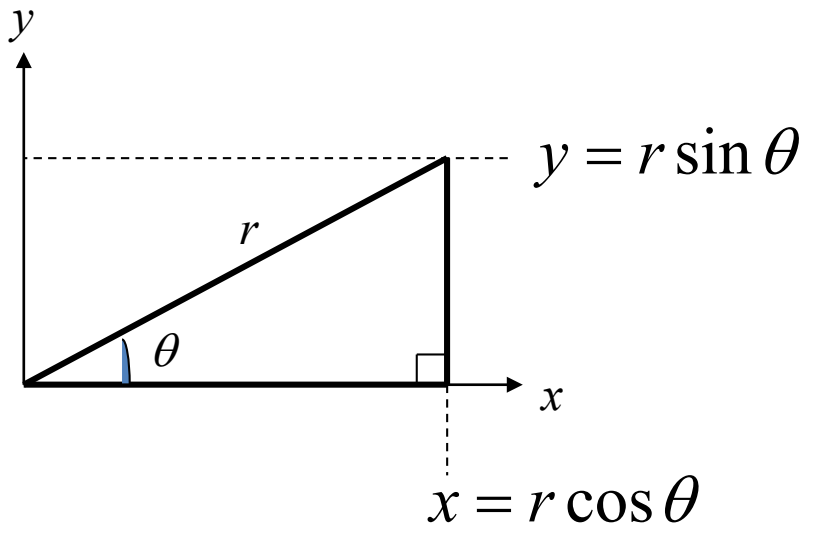
- 2D rotation
  - Trigonometry



$$\sin\theta = \frac{\textbf{opposite}}{\textbf{hypotenuse}}$$

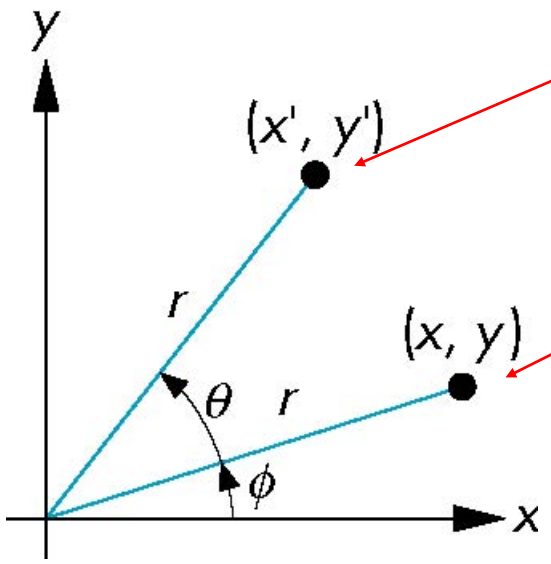
$$\cos\theta = \frac{\textbf{adjacent}}{\textbf{hypotenuse}}$$

$$\tan\theta = \frac{\textbf{opposite}}{\textbf{adjacent}}$$



2D Transformations

- 2D rotation
  - Rotation about origin



$$x' = r\cos(\phi+\theta) = r\cos\phi\cos\theta - r\sin\phi\sin\theta$$
$$y' = r\sin(\phi+\theta) = r\cos\phi\sin\theta + r\sin\phi\cos\theta$$

$$x = r\cos\phi$$
$$y = r\sin\phi$$

$$\boxed{\begin{aligned} x' &= x\cos\theta - y\sin\theta \\ y' &= x\sin\theta + y\cos\theta \end{aligned}}$$

2D Transformations

- 2D rotation
  - Rotation about origin
    - Equation in matrix form

$$P' = R\,P$$

P – Original position
P' – Rotated coordinates
R – Rotation matrix

    - Rotation matrix

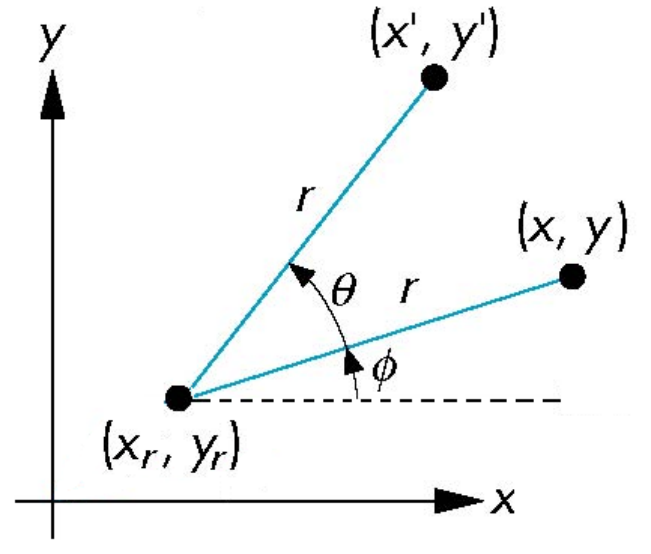
$$R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

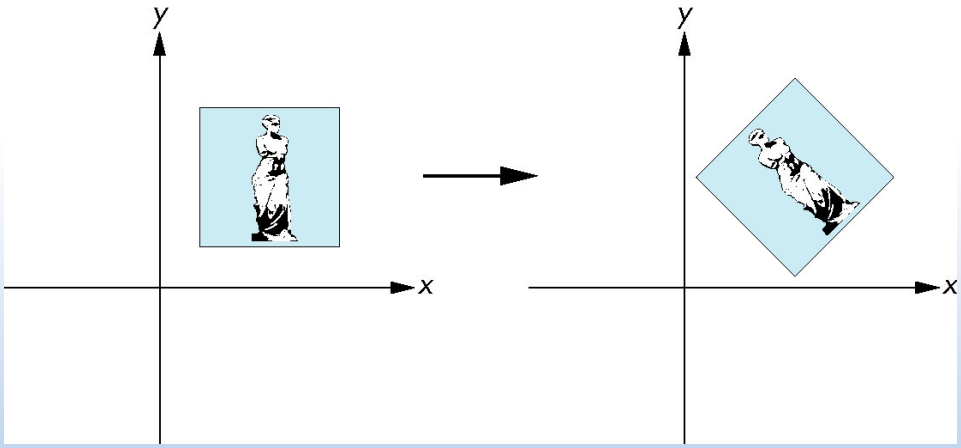
  - What about rotation about an arbitrary pivot point?

2D Transformations

- 2D rotation
  - Rotation about an arbitrary pivot point
    - Have to include an additive term in the matrix form



$$x' = x_r + (x - x_r)\cos\theta - (x - x_r)\sin\theta$$
$$y' = y_r + (y - y_r)\sin\theta + (y - y_r)\cos\theta$$



2D Transformations

- The basic 2D transformations (translation, scaling and rotation)
  - Can be expressed in the general form:

$$P' = M_1 P + M_2$$

  - To produce a sequence of transformations, could calculate transformed coordinates one step at a time
- **Homogenous coordinates**
  - Multiplication and translational terms can be combined into a single 3 x 3 matrix
    - Then **all transformations** can be expressed as **matrix multiplications**

2D Transformations

- 2D translation matrix
  - $P' = T\,P$

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

- 2D scaling matrix
  - $P' = S\,P$

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

- 2D rotation matrix
  - $P' = R\,P$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

2D Transformations

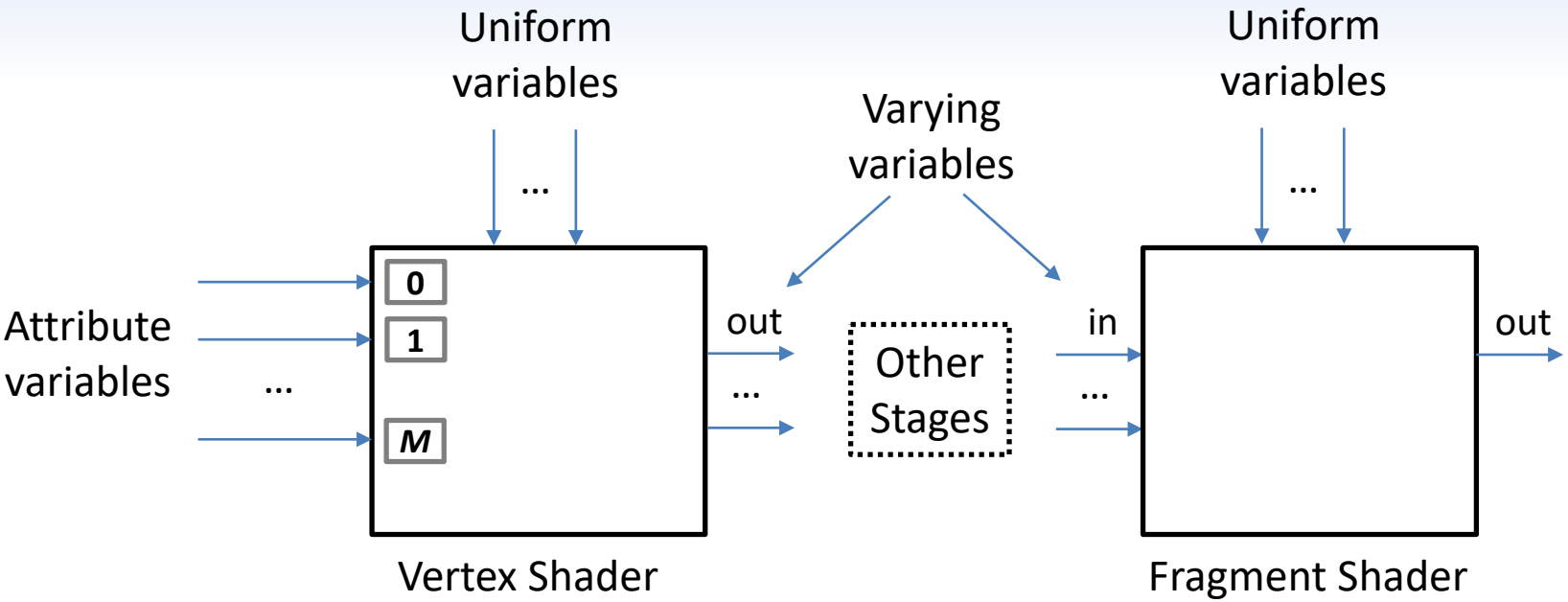
- GLM

```
#include <glm/glm.hpp>
#include <glm/gtx/transform.hpp>
glm::mat4 translationMatrix =
       glm::translate(glm::vec3(1.0f, 1.0f, 0.0f));
glm::mat4 scalingMatrix =
       glm::scale(glm::vec3(2.0f, 2.0f, 1.0f));
glm::mat4 rotationMatrix =
       glm::rotate(angleInRadians,
       glm::vec3(0.0f, 0.0f, 1.0f));

angleInRadians = glm::radians(angleInDegrees);
angleInDegrees = glm::degrees(angleInRadians);
```

Modelling Transformations

```
#version 330 core
layout(location = 0) in vec3 aPosition;
layout(location = 1) in vec3 aColor;

uniform mat4 uModelMatrix;

out vec3 vColor;

void main()
{
   gl_Position = uModelMatrix * vec4(aPosition, 1.0f);

   vColor = aColor;
}
```

Shaders


Uniform variables
...
Varying variables
Uniform variables
...
Attribute variables
...
0
1
M
out
...
Other Stages
in
...
out
Vertex Shader
Fragment Shader

Transformations

- Transforming the vertices of an object
  - Set uniform matrix before rendering an object
    - Declare variable for object's matrix

```
glm::mat4 gModelMatrix;
```

    - Initialise to identity

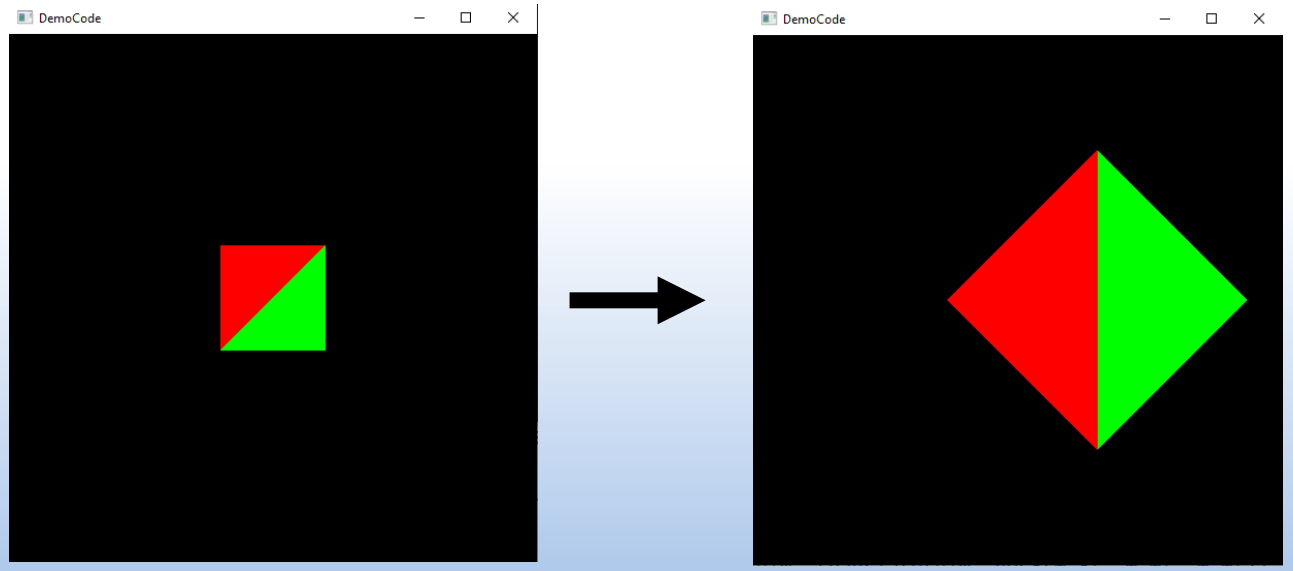
```
gModelMatrix = glm::mat4(1.0f);
```

    - Apply transformations to the model matrix
    - Set uniform variable before rendering

```
gShader.use();
gShader.setUniform("uModelMatrix", gModelMatrix);
glBindVertexArray(gVAO);
glDrawArrays(GL_TRIANGLES, 0, 6);
```

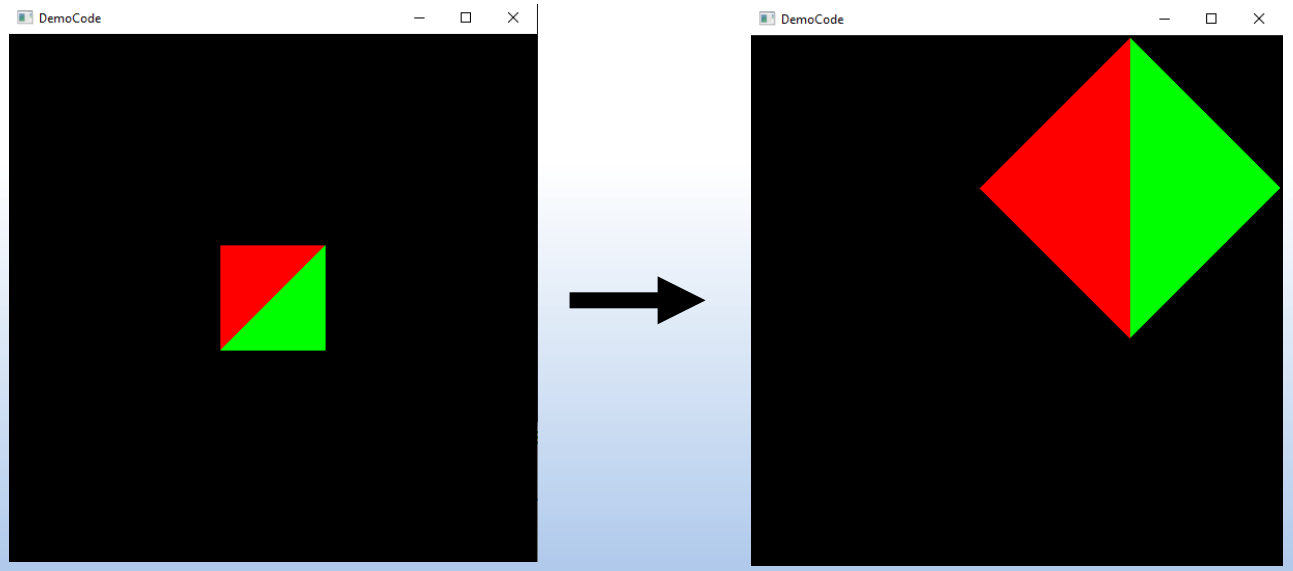
Transformations

```
gModelMatrix *=
    glm::translate(glm::vec3(0.3f, 0.0f, 0.0f));
gModelMatrix *=
    glm::rotate(glm::radians(45.0f),
    glm::vec3(0.0f, 0.0f, 1.0f));
gModelMatrix *=
    glm::scale(glm::vec3(2.0f, 2.0f, 1.0f));
```



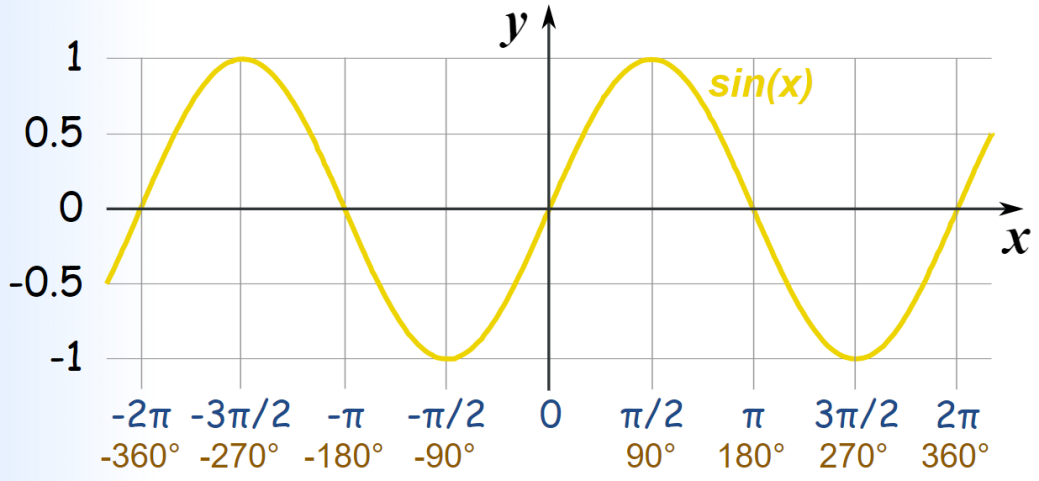
Transformations

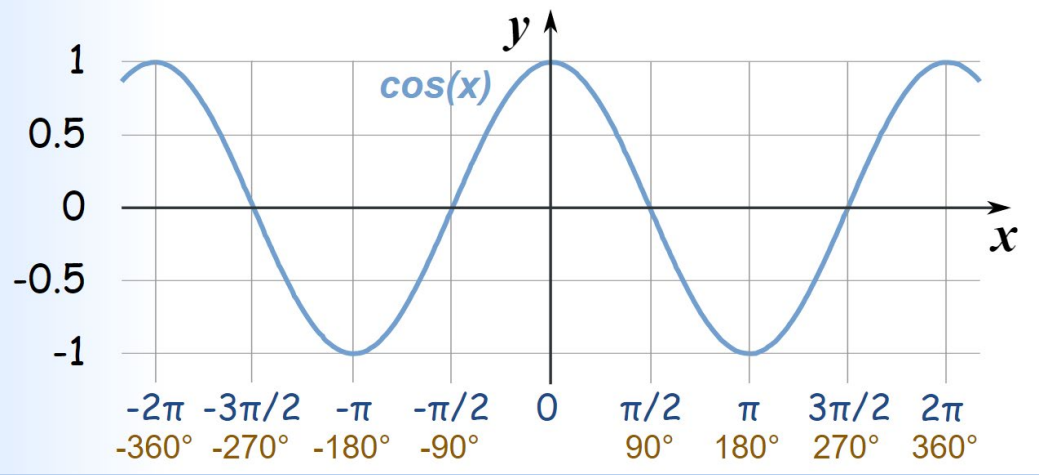
```
gModelMatrix *=
    glm::rotate(glm::radians(45.0f),
    glm::vec3(0.0f, 0.0f, 1.0f));
gModelMatrix *=
    glm::scale(glm::vec3(2.0f, 2.0f, 1.0f));
gModelMatrix *=
    glm::translate(glm::vec3(0.3f, 0.0f, 0.0f));
```



Sine and Cosine

π = 180 degrees or 3.14159265359 radians





https://www.mathsisfun.com/algebra/trig-sin-cos-tan-graphs.html

Transformations

- Sequence of transformations
  - ➢ Composite transformation matrix
    - Often referred to as **concatenation**, or composition, of matrices
    - Many positions in the scene typically transformed by the same sequence, more efficient to form a single transformation matrix
    - Matrix multiplication is **associate**, but **not commutative**

$P' = M_2 M_1 P$

$= M_2 (M_1 P)$

$= (M_2 M_1) P$

$= M P$

$$\begin{bmatrix} 1 & 0 & t_{2x} \\ 0 & 1 & t_{2y} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & t_{1x} \\ 0 & 1 & t_{1y} \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_{1x} + t_{2x} \\ 0 & 1 & t_{1y} + t_{2y} \\ 0 & 0 & 1 \end{bmatrix}$$

Transformations

- Matrix multiplication
  - Properties
    - **MI = IM = M**
      - Multiplying by identity matrix leaves points unchanged
    - **$MM^{-1} = M^{-1}M = I$**
      - Inverse 2D transformation matrices
        - » For transformations in the opposite direction
        - » Translation, scaling, rotation

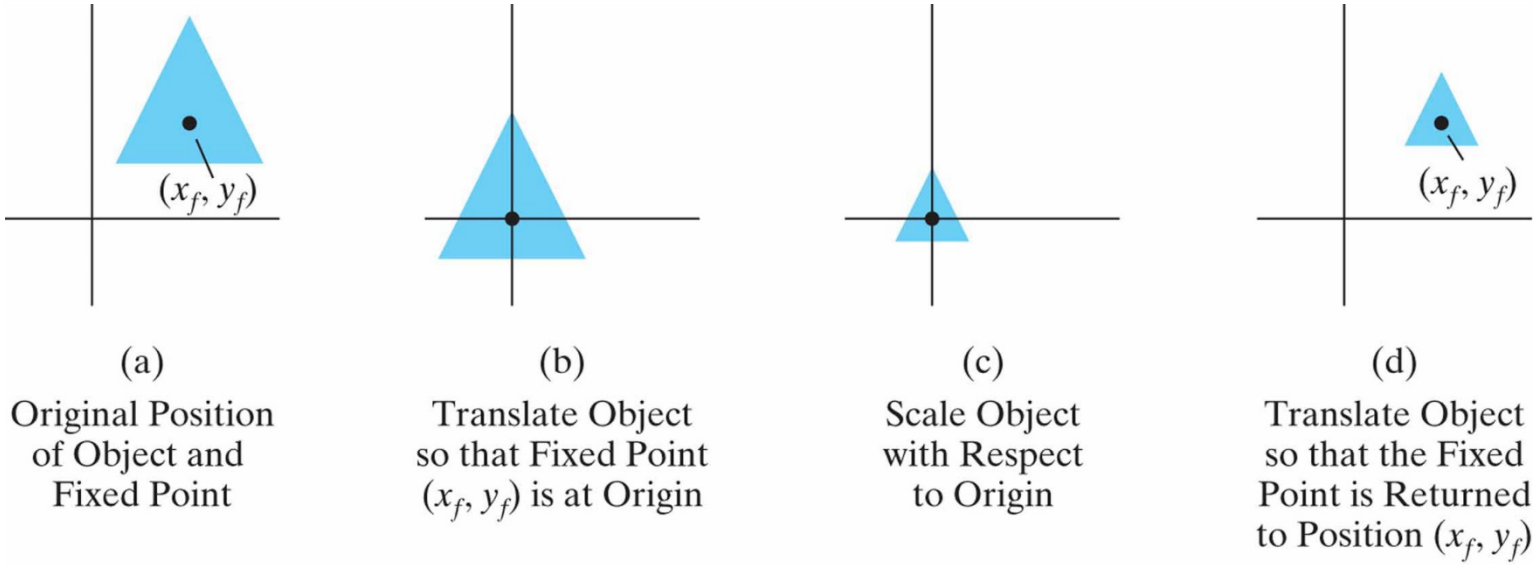
$$T^{-1} = \begin{bmatrix} 1 & 0 & -t_x \\ 0 & 1 & -t_y \\ 0 & 0 & 1 \end{bmatrix}$$

$$S^{-1} = \begin{bmatrix} \frac{1}{s_x} & 0 & 0 \\ 0 & \frac{1}{s_y} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R^{-1} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Transformations

- General 2D fixed-point scaling

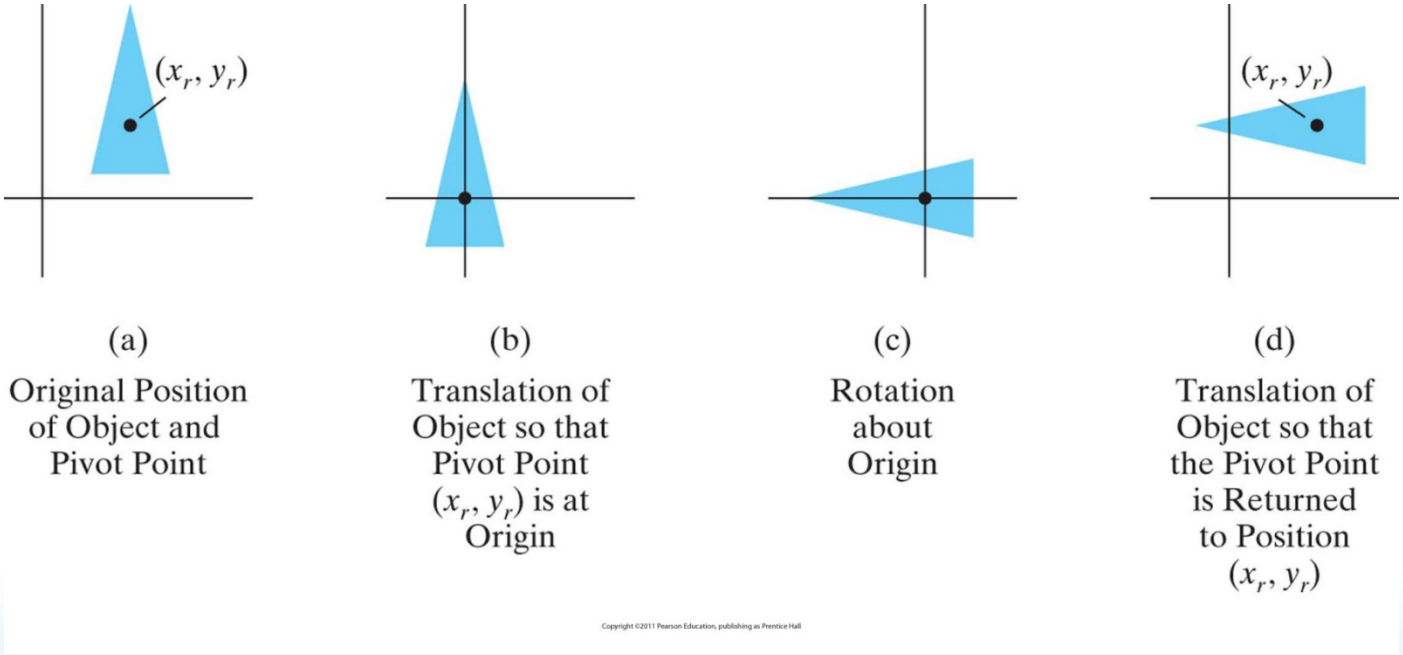


$$\begin{bmatrix} 1 & 0 & x_r \\ 0 & 1 & y_r \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_r \\ 0 & 1 & -y_r \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & x_r(1-s_x) \\ 0 & s_y & y_r(1-s_y) \\ 0 & 0 & 1 \end{bmatrix}$$

$$T(x_f, y_f)S(s_x, s_y)T^{-1}(x_f, y_f) = R(x_f, y_f, s_x, s_y)$$

Transformations

- General 2D pivot-point rotation

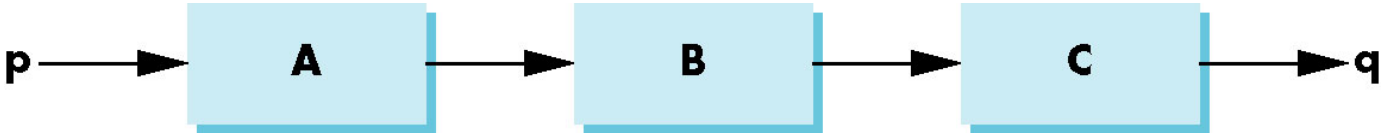


$$\begin{bmatrix} 1 & 0 & x_r \\ 0 & 1 & y_r \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_r \\ 0 & 1 & -y_r \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & x_r(1-\cos\theta)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

$$T(x_r, y_r) R(\theta) T^{-1}(x_r, y_r) = R(x_r, y_r, \theta)$$

Transformations

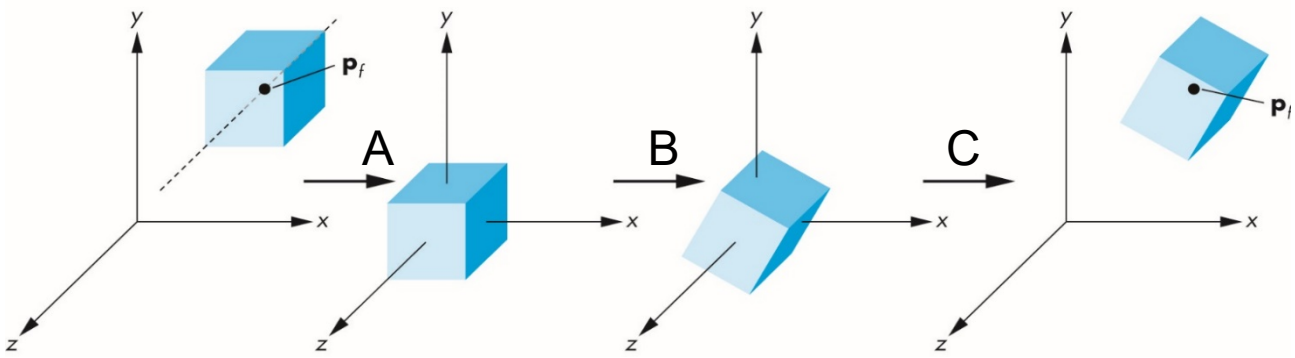
- **Concatenation** of transformations
  - Multiplying together sequences of transformations



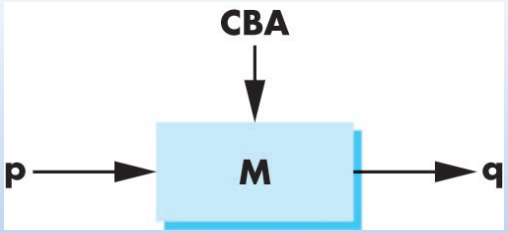
  - **Order of transformations**
    - Matrix multiplication is **associative**, but **not commutative**
    - q = CBAp = (C (B (A p)))
      - Note the order
        - » Post-multiplication, point on the right
        - » Multiplication from right-to-left

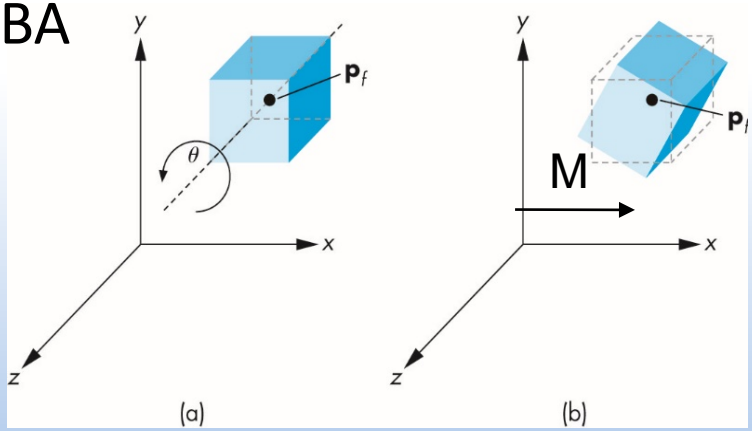
Transformations

- Concatenation of transformations
  - If same transformation applied to many vertices



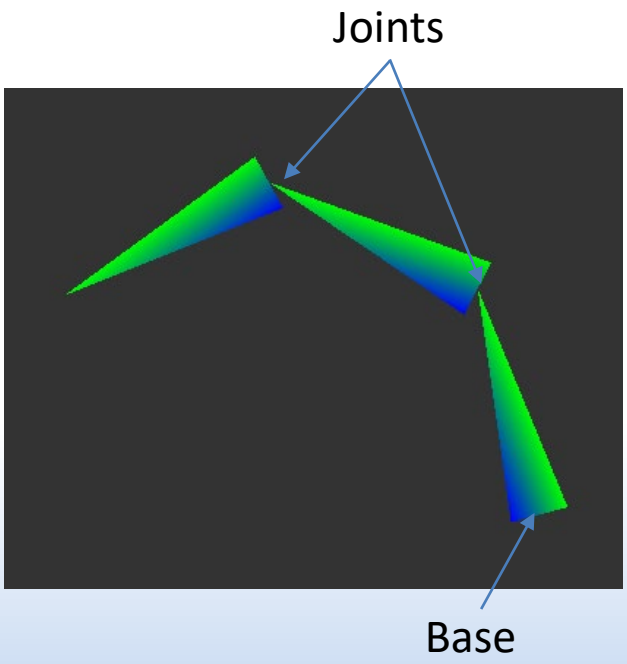
  - More efficient to form M = CBA
    - q = Mp



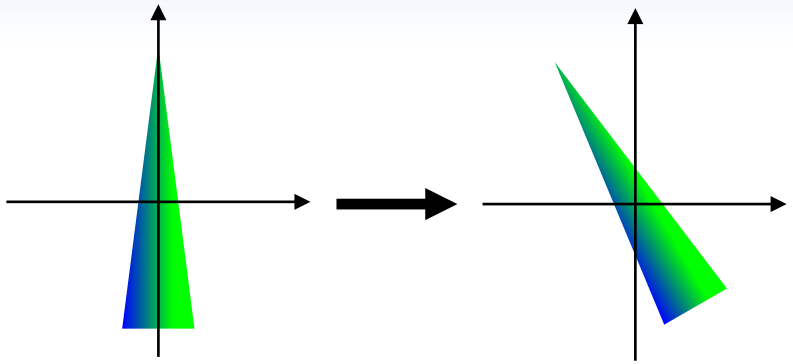


2D Transformations

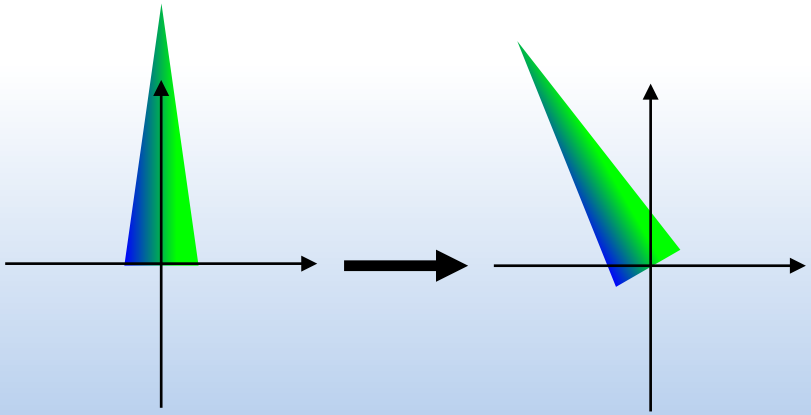
- Hierarchical models
  - E.g., a robot arm



If rotated directly

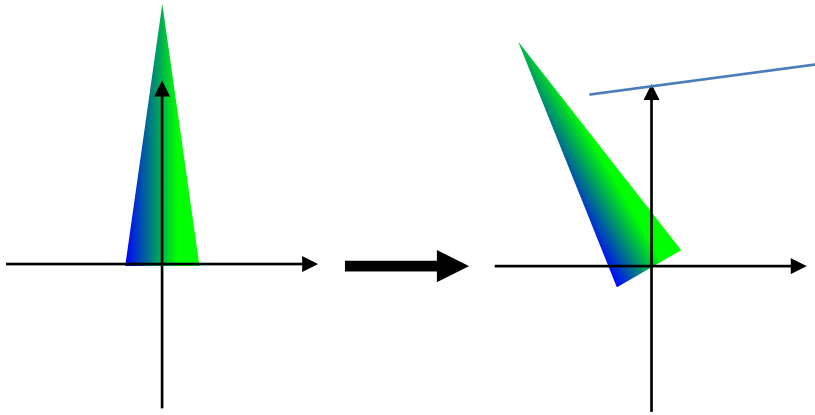


1. Transform pivot point of base to origin before rotating

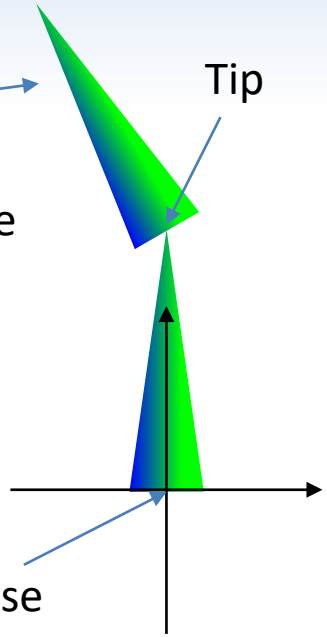


2D Transformations

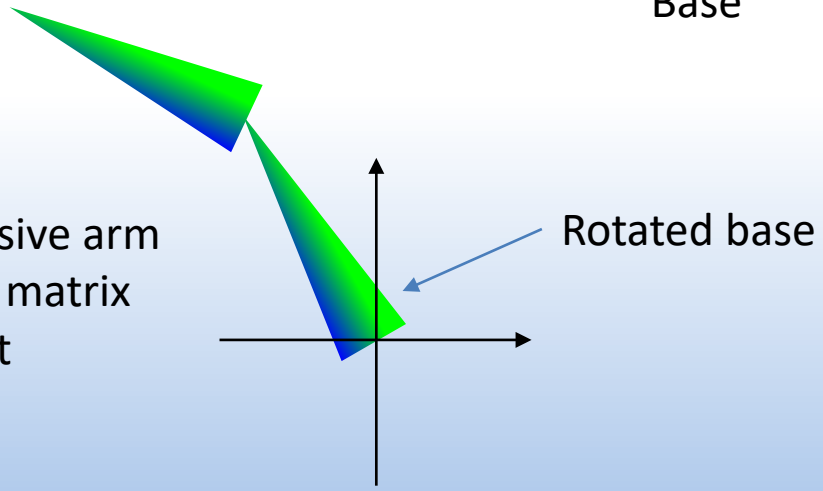
2. Transform successive arm segment's pivot point to the origin before rotating



3. Transform successive arm segment's pivot point to the tip of previous segment

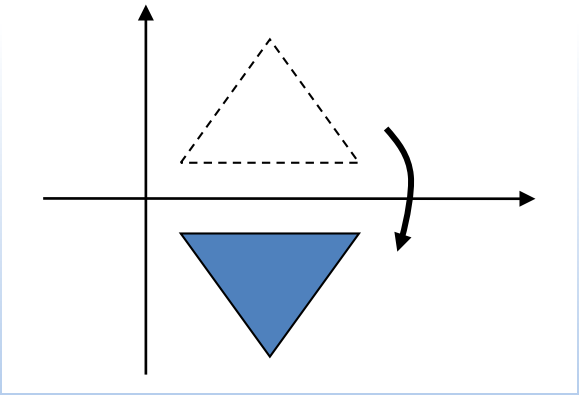


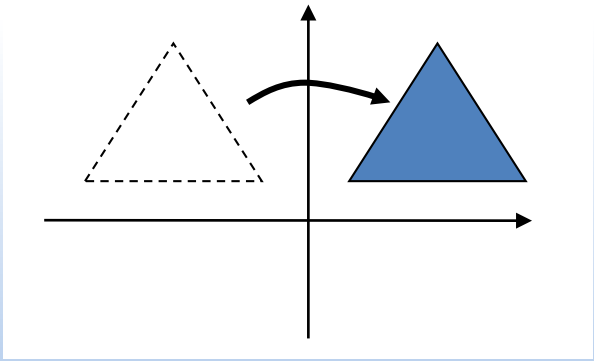
4. Transform successive arm segments by model matrix of previous segment



2D Transformations

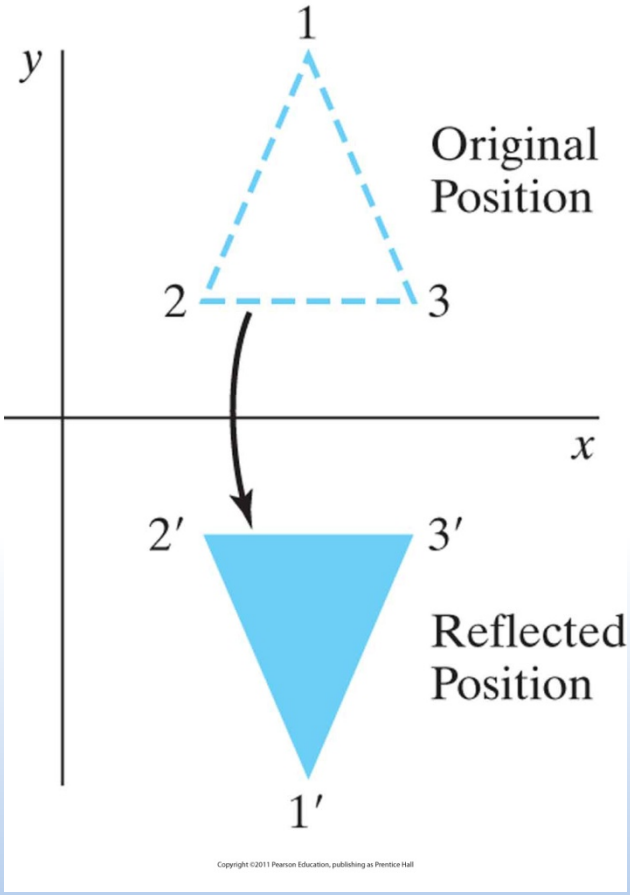
- 2D reflection
  - Transformation that produces a mirror image
    - About a line (or a plane in 3D)
  - Transformation matrix
    - Similar to scaling matrix, but with negative scaling factors
  - Reflections about any line
    - Achieved by combining with other transformations



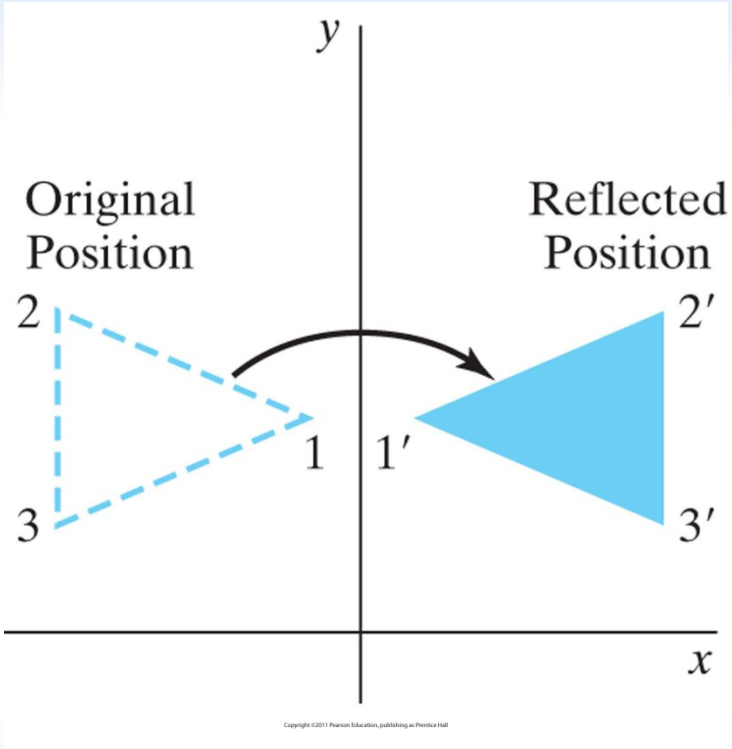


2D Transformations

- 2D reflection



$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



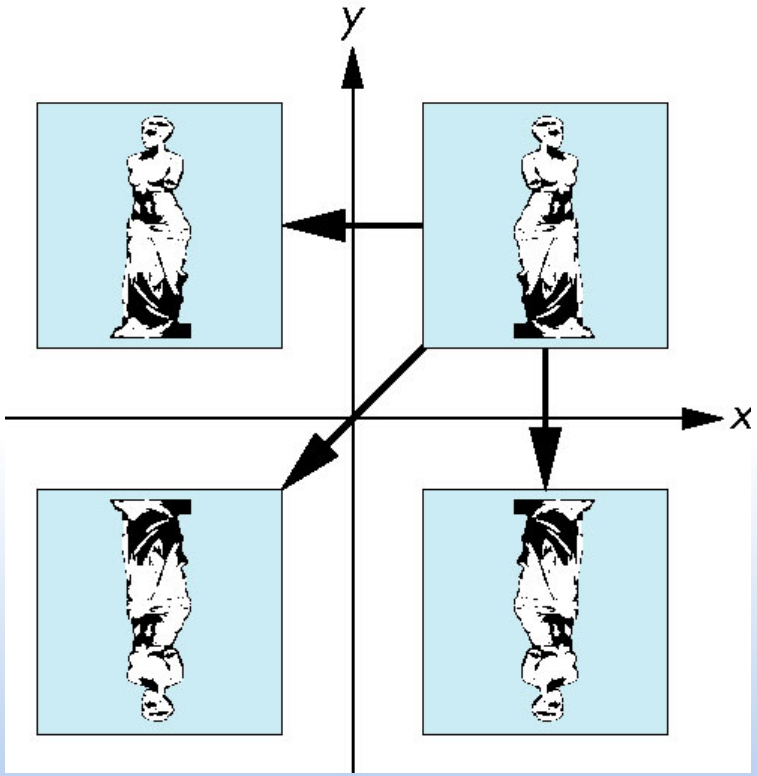
$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

2D Transformations

- 2D reflection
  - Using negative scaling factors

$s_x = -1\ s_y = 1$

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



original

$s_x = -1\ s_y = -1$

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$s_x = 1\ s_y = -1$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

References

- Among others, material sourced from
  - Hearn, Baker & Carithers, “Computer Graphics with OpenGL”, Prentice-Hall
  - Angel & Shreiner, “Interactive Computer Graphics: A Top-Down Approach with OpenGL”, Addison Wesley
  - Akenine-Moller, Haines & Hoffman, “Real Time Rendering”, A.K. Peters
  - Joey de Vries, “Learn OpenGL,” https://learnopengl.com/
  - https://www.khronos.org/opengl/wiki/
  - http://en.wikipedia.org/wiki/