HIERARCHICAL TRANSFORMATIONS

A Practical Introduction

Christopher Peters

HPCViz, KTH Royal Institute of Technology, Sweden

chpeters@kth.se

https://www.kth.se/profile/chpeters/

Transformations

Many objects are composed of hierarchies
Transformations enable us to compose hierarchies

Atlas, Boston Dynamics

ROYAL INSTITUTE OF TECHNOLOGY

Transformations

Positioning geometric objects in the virtual world is an operation fundamental for scene composition and computer animation

Scenes are composed of:

- Viewer/camera
- Objects and shapes (composed of geometric primitives)
- Other (textures, lighting, …)

In this lecture, we will consider only rotation and translation transformations

- There are others too: Shear, squash, stretch…

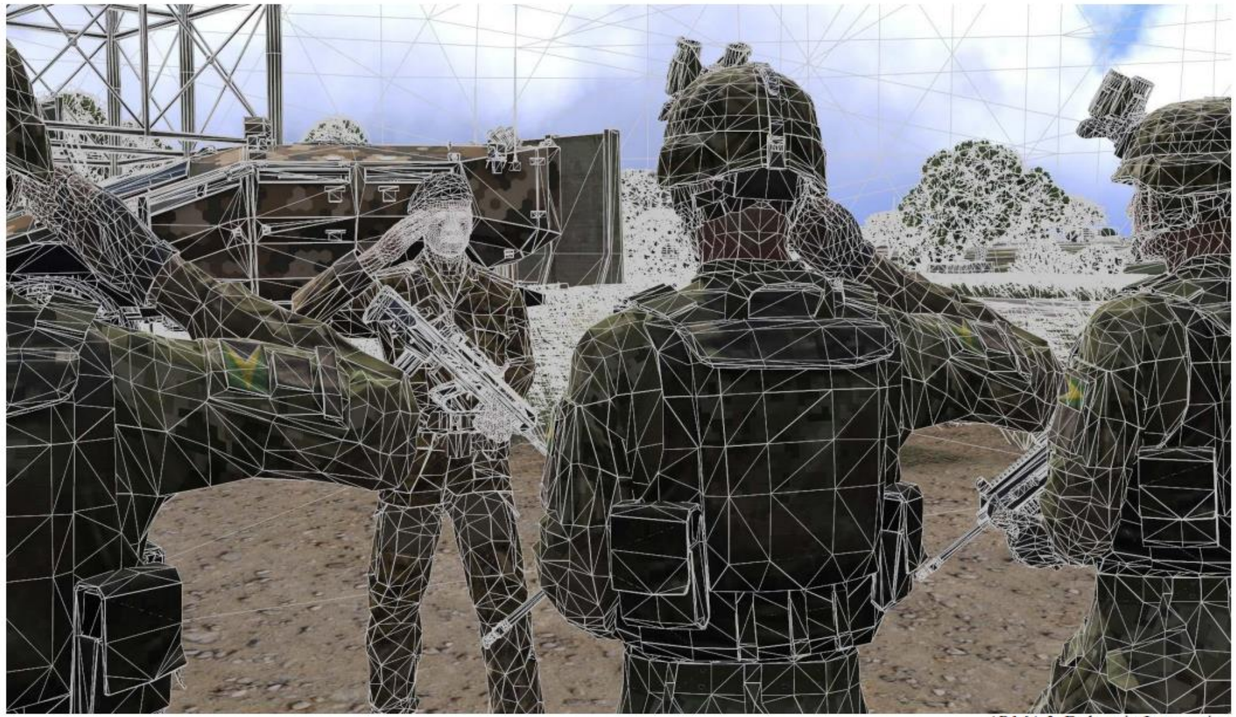
Scene composition

ARMA 3, Bohemia Interactive

A photorealistic scene (circa 2013)

Scene composition

A photorealistic scene (circa 2013)

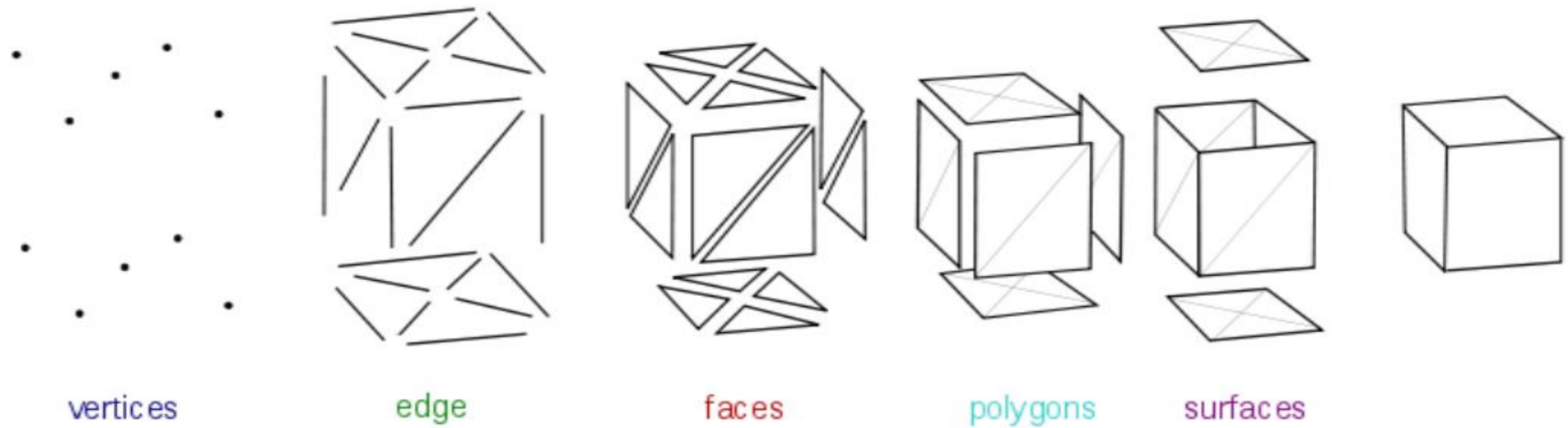


ARMA 3, Bohemia Interactive

Underlying representation (geometry: white)

Geometric primitives

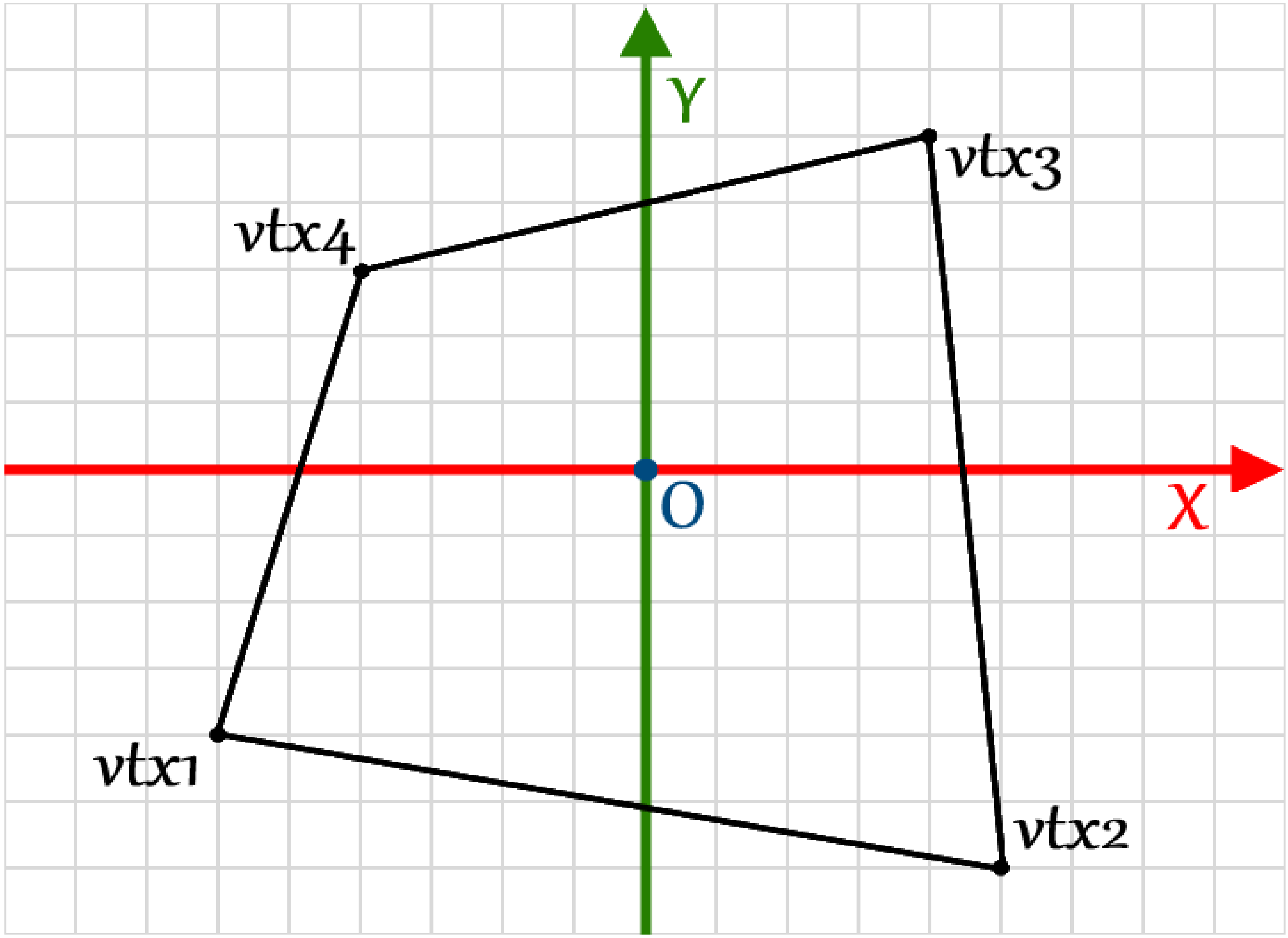
(a brief introduction)



Graphical objects are composed of primitives

- More about geometry in subsequent lectures

Vertices



Vertices:

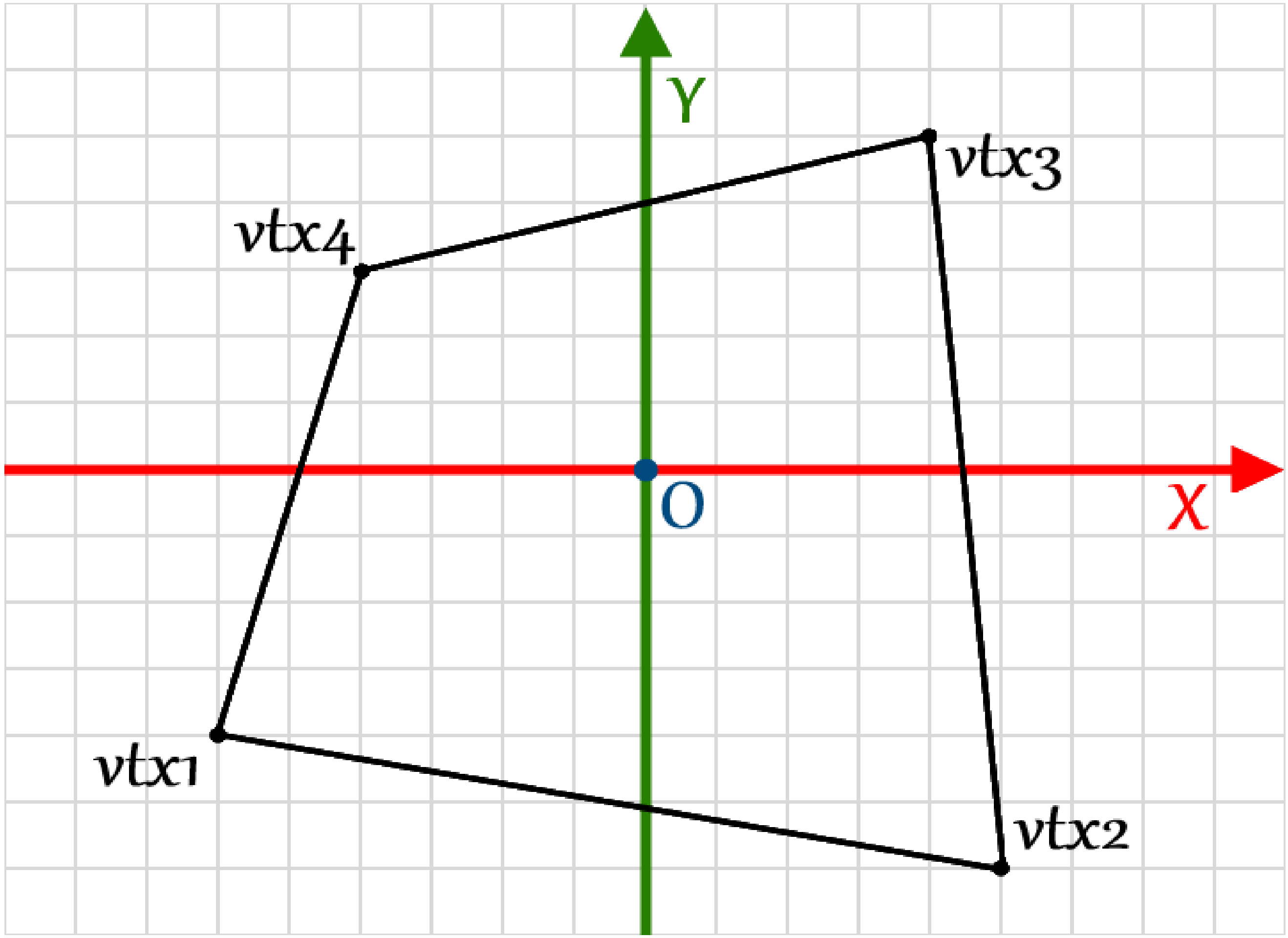
vtx1 (-6.0,-4.0),

vtx2 (5.0, -6.0),

vtx3 (4.0, 5.0),

vtx4 (-4.0, 3.0)

Vertices



Vertices:

vtx1 (-6.0,-4.0),

vtx2 (5.0, -6.0),

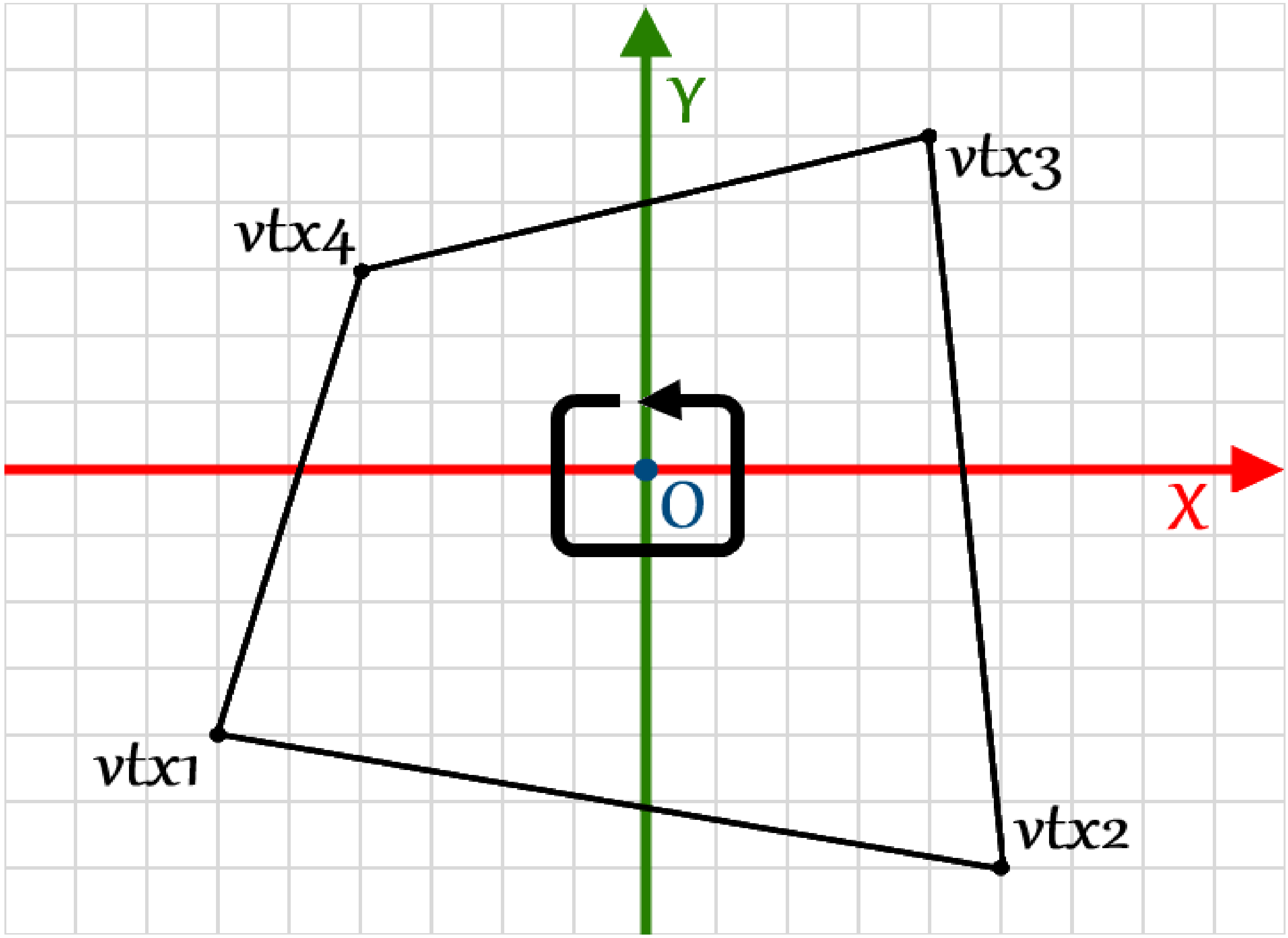
vtx3 (4.0, 5.0),

vtx4 (-4.0, 3.0)

Q: Why this ordering?

Hint: do cross-product on vectors defined by two edges incident to any vertex

Vertices



Vertices:

vtx1 (-6.0,-4.0),

vtx2 (5.0, -6.0),

vtx3 (4.0, 5.0),

vtx4 (-4.0, 3.0)

Right-hand rule

Winding order of the vertices

ROYAL INSTITUTE OF TECHNOLOGY

Transformations

Recall *translation* from previous lecture:

- Translate a point $\mathbf{p}$ along a vector $\mathbf{t}$
- General case:

$$\mathbf{p}' = \mathbf{p} + \mathbf{t}$$

- 2D:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} = \begin{bmatrix} x + t_x \\ y + t_y \end{bmatrix}$$

- 3D:

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} = \begin{bmatrix} x + t_x \\ y + t_y \\ z + t_z \end{bmatrix}$$

Translating an object

Translation operation takes place on a point
But a geometric object (*mesh*) is a collection of vertices
How to translate that?

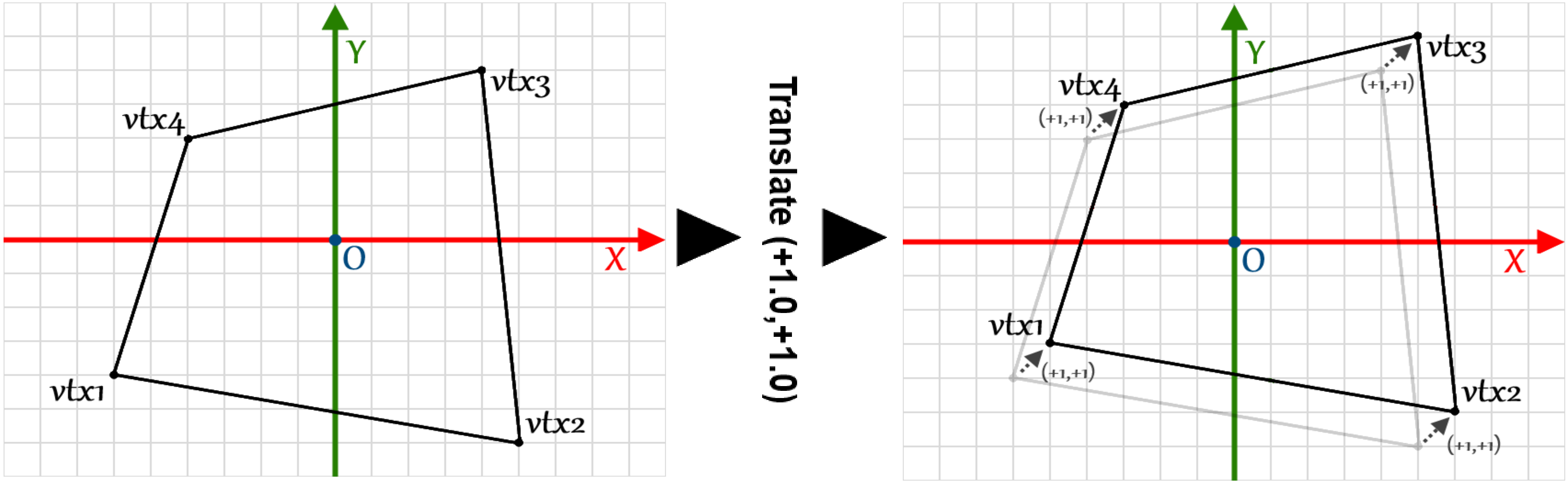
Translating an object

Translation operation takes place on a point

But a geometric object (*mesh*) is a collection of vertices

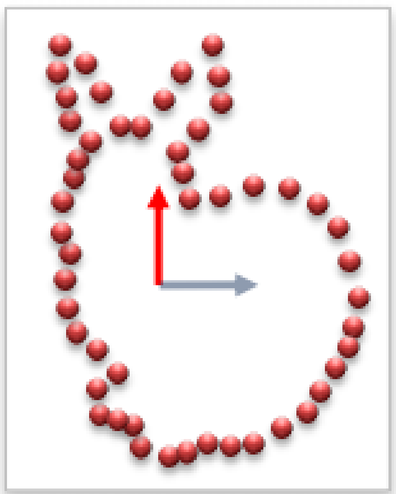
How to translate that?

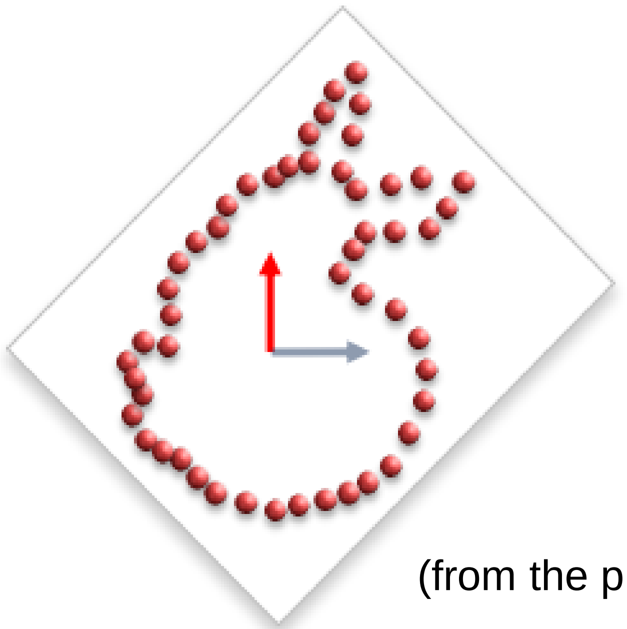
Translate each of its vertices



Rotating an object

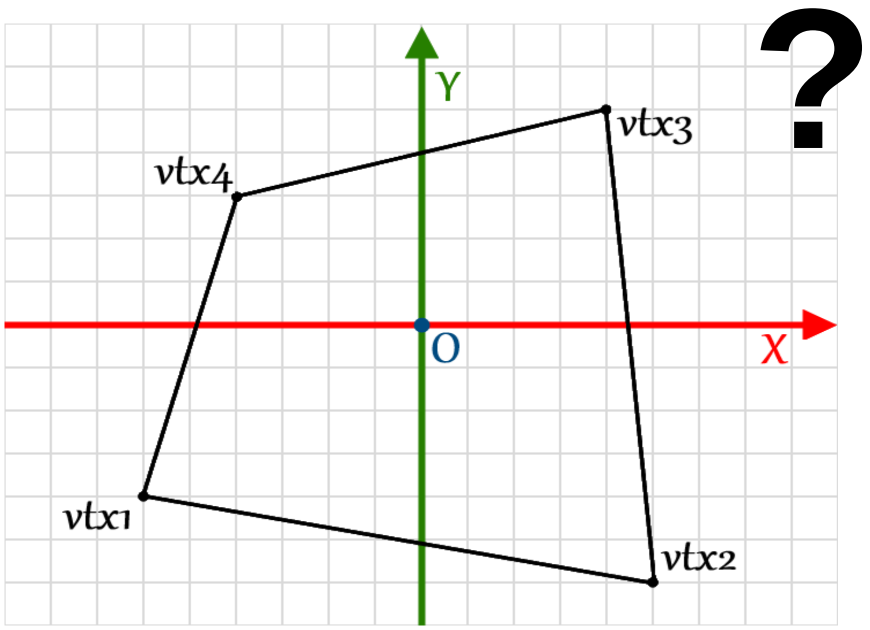
Rotation operation takes place on a point
How to rotate a object?
The same procedure applies:
Rotate each vertex that comprises the object





(from the previous lecture)

Coordinate spaces

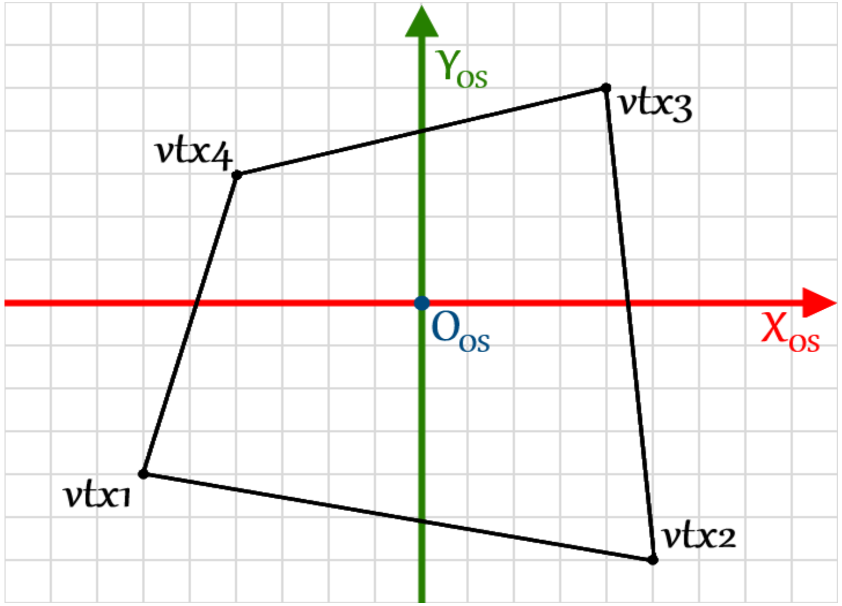


What are the coordinates of an object?

– Answer: It depends on the *coordinate space*

Coordinate spaces

Object specified in Object space (OS)



What are the coordinates of an object?

- Answer: It depends on the *coordinate space*

The vertices of an object are usually specified in its own local coordinate space

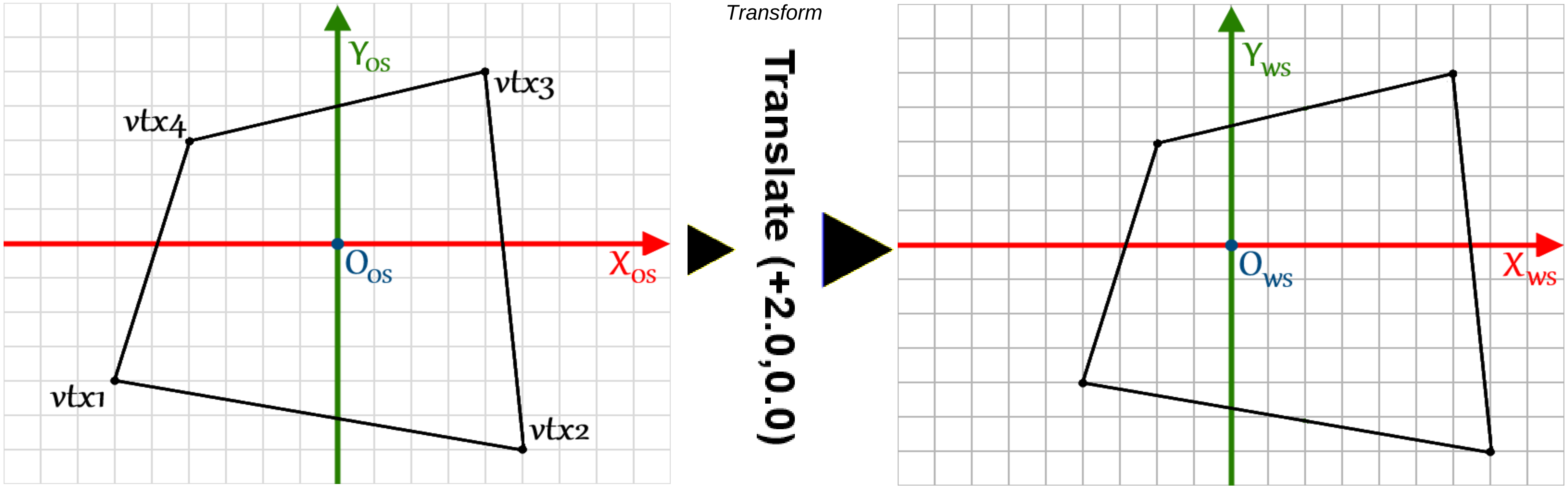
- **Object space (OS)**
- Origin often located near the *centroid* of the object

ROYAL INSTITUTE OF TECHNOLOGY

World space

Object specified in Object space (OS)

Positioned in world space (WS) via transform



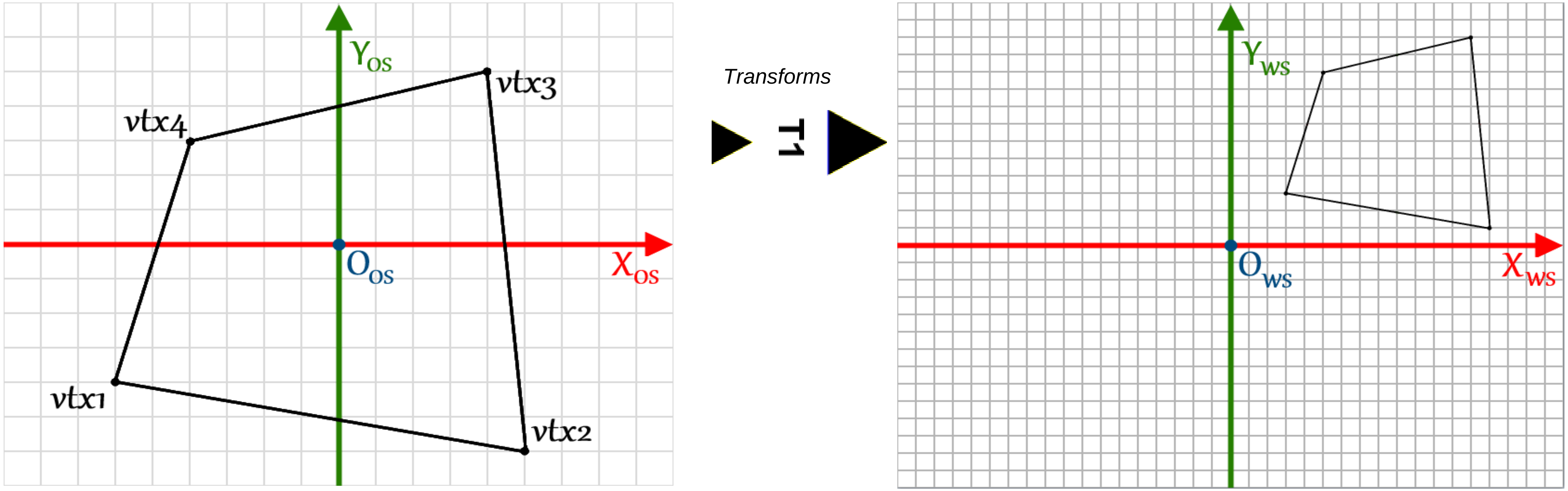
An instance of an object is positioned in the world using a transformation

- **World space (WS)**
- In this case, the transformation **`Translate(`**$\mathbf{t_x}$**`,`**$\mathbf{t_y}$**`)`**
- *Displacement* of $\mathbf{t_x}$ units along the x-axis and $\mathbf{t_y}$ units along the y-axis

ROYAL INSTITUTE OF TECHNOLOGY

World space

Object specified in Object space (OS) | *Positioned in world space (WS) via transform*

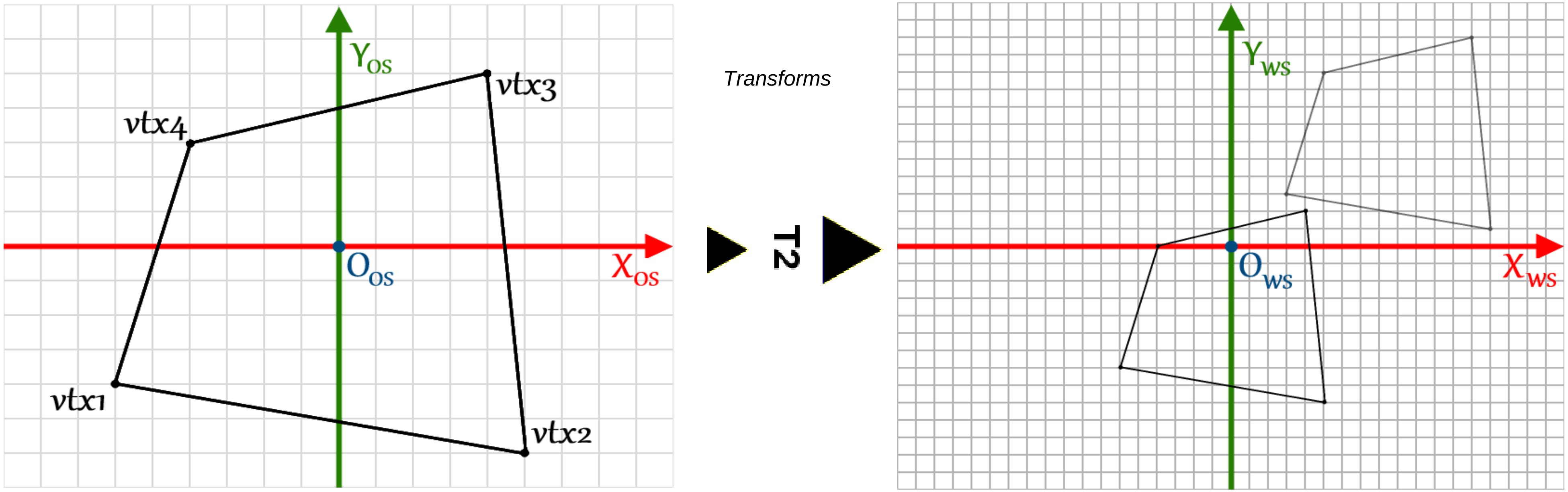


Multiple instances of the same object can be positioned in the world via individual transformations

World space

Object specified in Object space (OS)

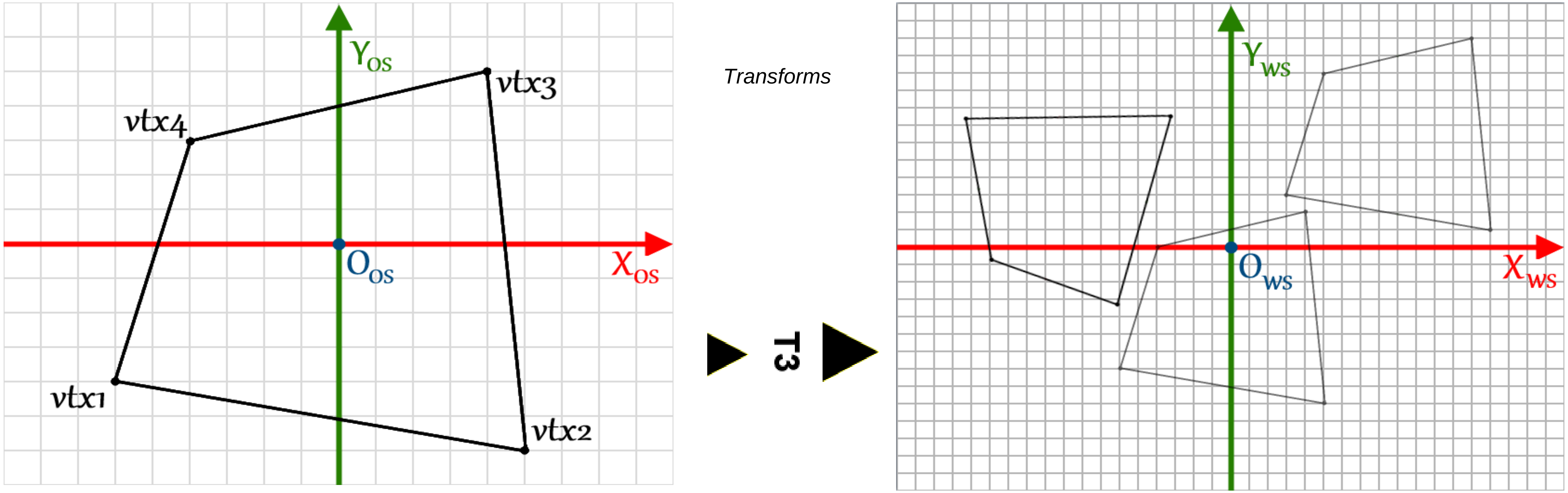
Positioned in world space (WS) via transform



Multiple instances of the same object can be positioned in the world via individual transformations

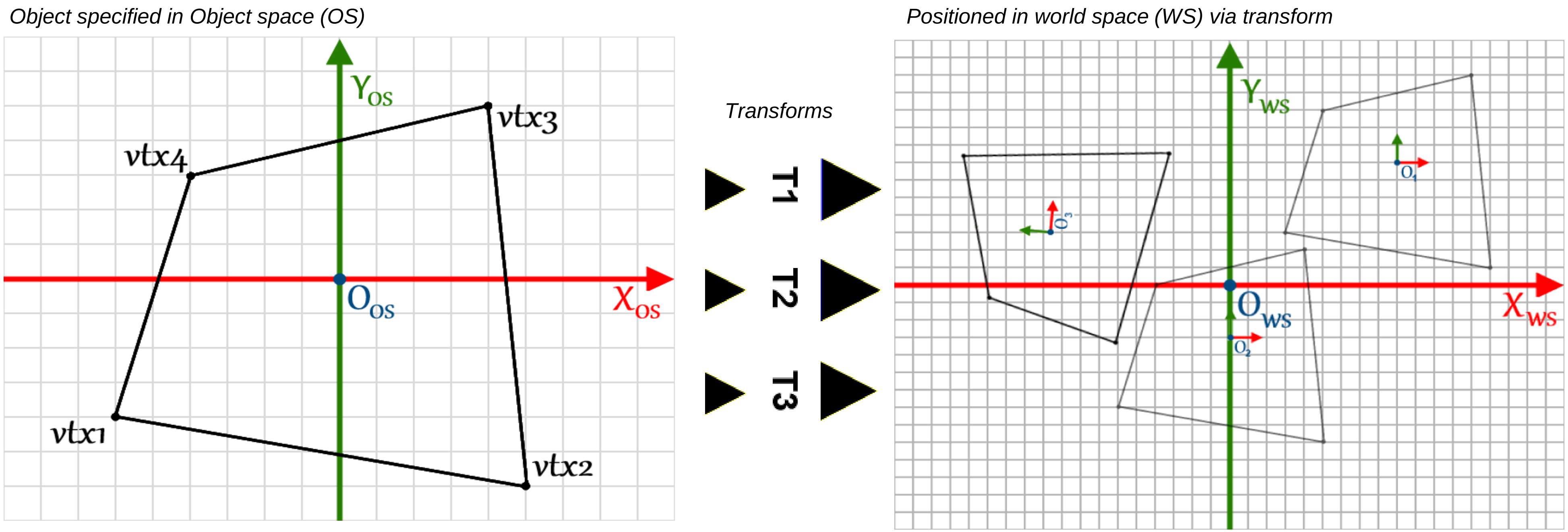
World space

Object specified in Object space (OS) *Positioned in world space (WS) via transform*



Multiple instances of the same object can be positioned in the world via individual transformations

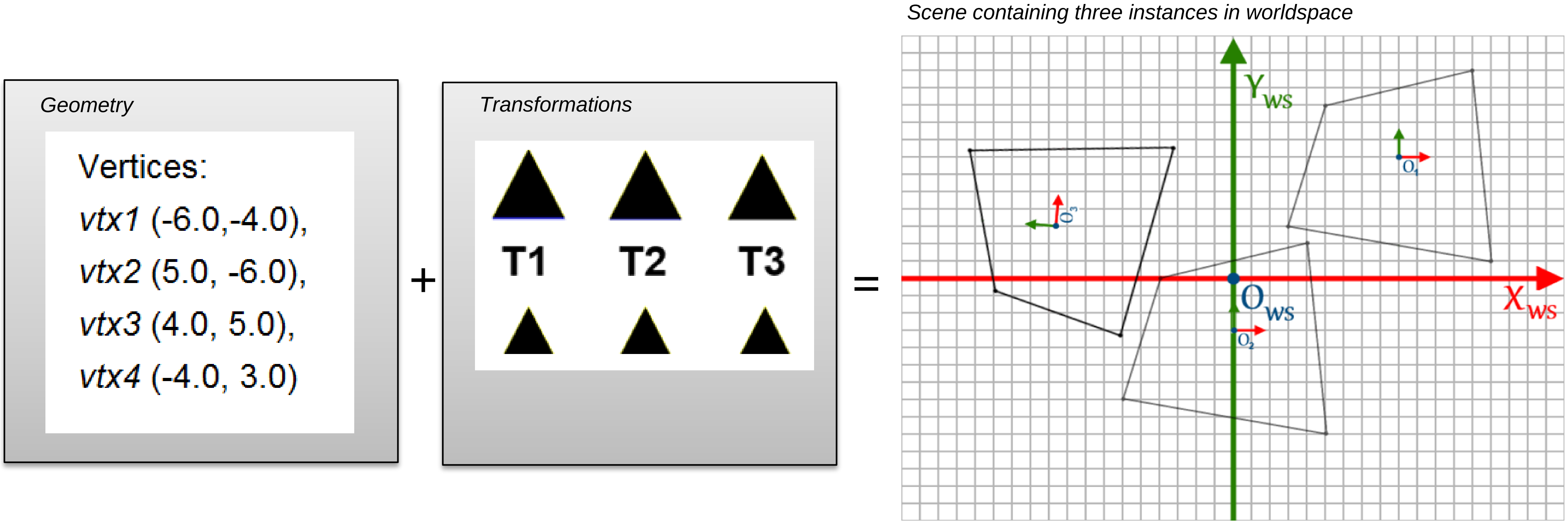
World space



Multiple instances of the same object can be positioned in the world via individual transformations

- Objects positioned according to their respective object space origins
- More on this later

Geometry and transformations



Geometry is usually stored separately from respective transformations

- Objects definitions versus object instances
- Memory savings

Representation

Recall: Transformations are represented as 4x4 *matrices*
From the last lecture:

Translation

$$\mathbf{T}(t_x, t_y, t_z) = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotation around x-axis

$$\mathbf{R}_x(\phi) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi & 0 \\ 0 & \sin\phi & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotation around y-axis

$$\mathbf{R}_y(\phi) = \begin{pmatrix} \cos\phi & 0 & \sin\phi & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\phi & 0 & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotation around z-axis

$$\mathbf{R}_z(\phi) = \begin{pmatrix} \cos\phi & -\sin\phi & 0 & 0 \\ \sin\phi & \cos\phi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\mathbf{M} \cdot \mathbf{x} = \begin{pmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix}$$

Local Coordinate Marker

Nothing is displayed on the screen until you draw an object
Transformation matrices are stored in memory
How do we keep track of positioning information?

Local Coordinate Marker

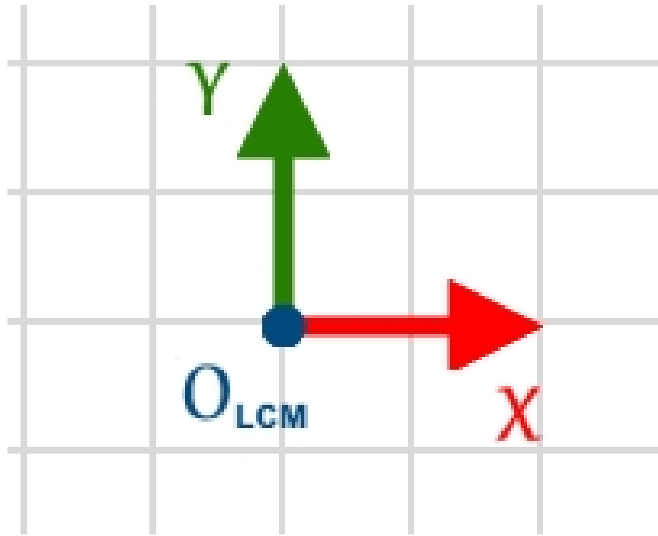
Nothing is displayed on the screen until you draw an object

Transformation matrices are stored in memory

How do we keep track of positioning information?

One answer: Local Coordinate Marker (LCM)

- A special coordinate system that we track via pen and graph paper or mentally
- The LCM represents a transformation matrix
- But in a manner more intuitive to humans



Local Coordinate Marker

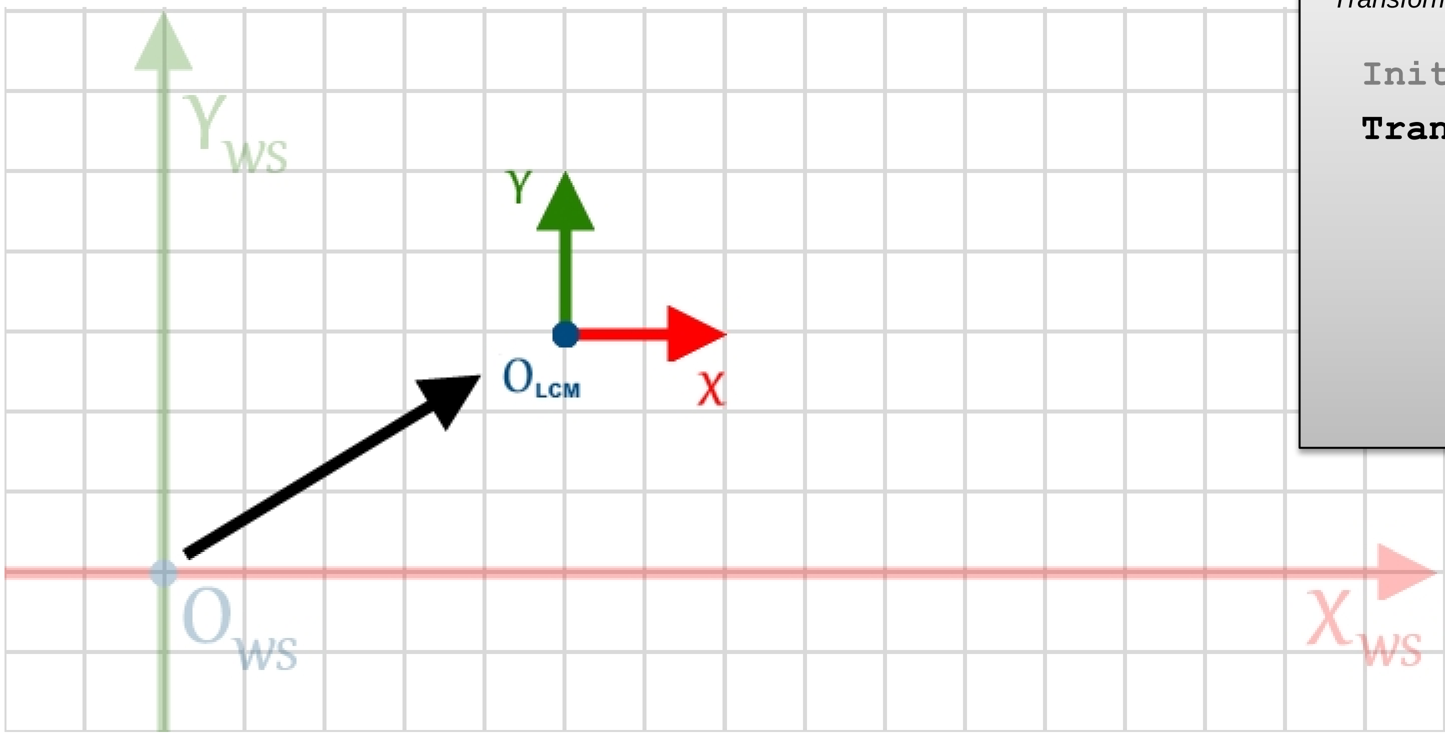
Transformation Operations

```
Initialise()
```

LCM begins at the worldspace origin

Its basis vectors match those of the WS basis

Local Coordinate Marker

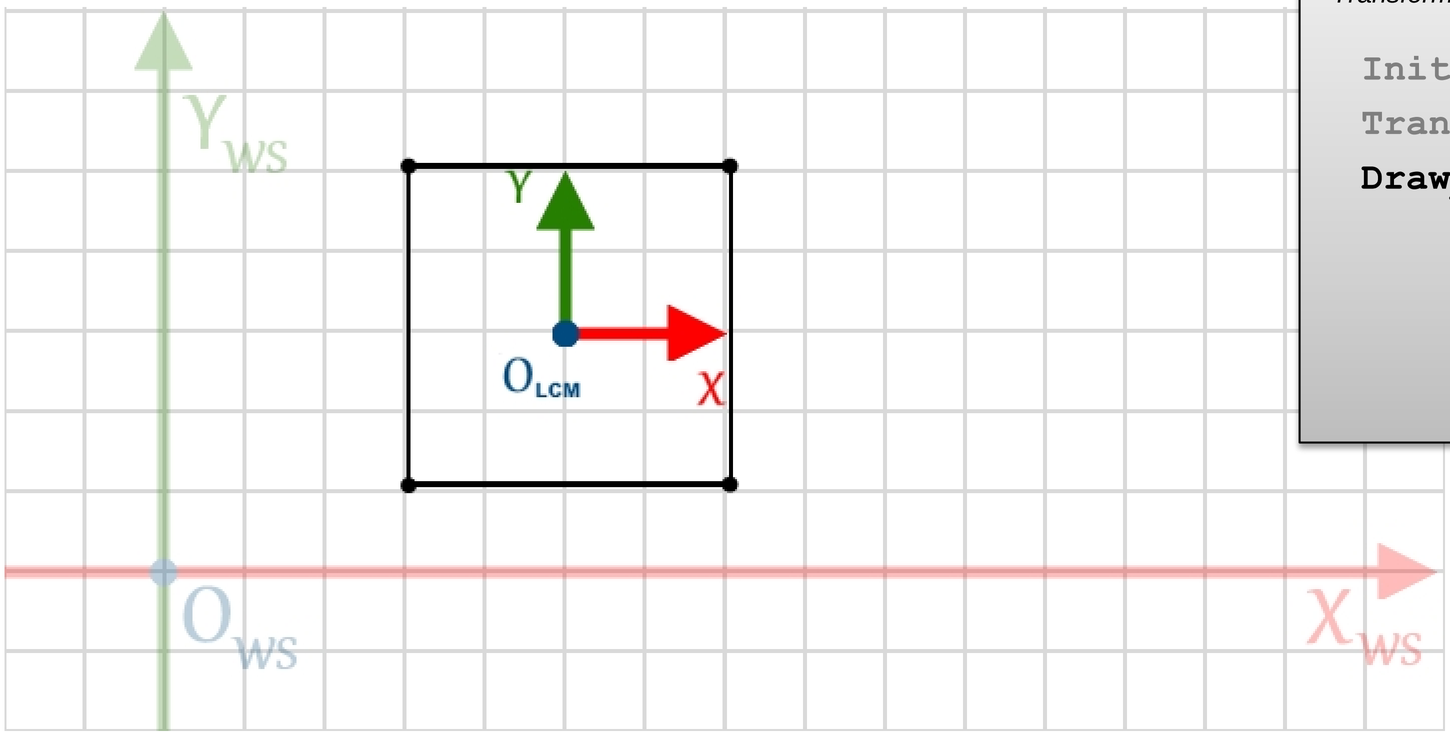


Transformation Operations

```
Initialise()
Translate(5,3)
```

We keep a track of the marker as we conduct various positioning operations

Local Coordinate Marker



Transformation Operations

```
Initialise()
Translate(5,3)
Draw_Square()
```

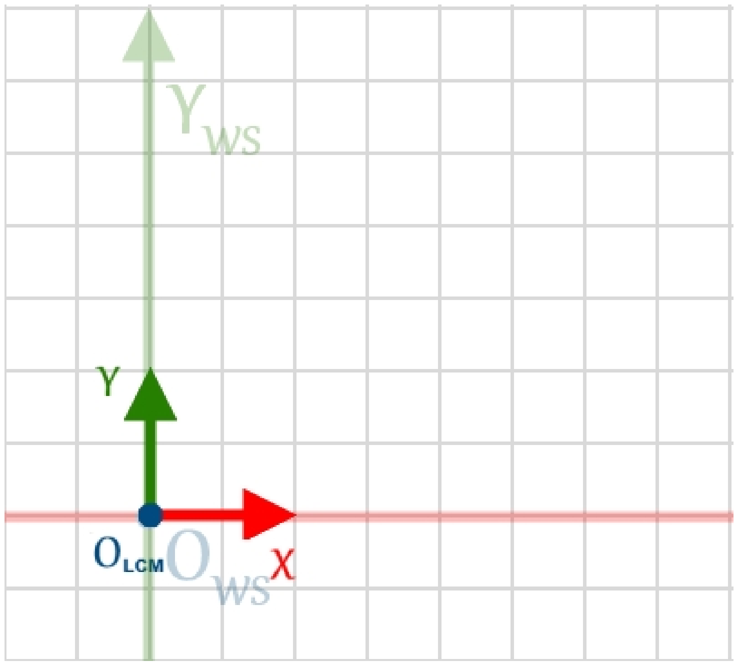
Until we draw the object

- Note: the LCM is not drawn on the screen!
- (unless you decide to add some code to do so…)

Practical transformations

The LCM represents a special transformation matrix

- *Modelview matrix*
- When a geometric object is drawn, it is placed according to the transform defined in the Modelview matrix



Transformation Operations

```
Initialise()
```

Modelview matrix

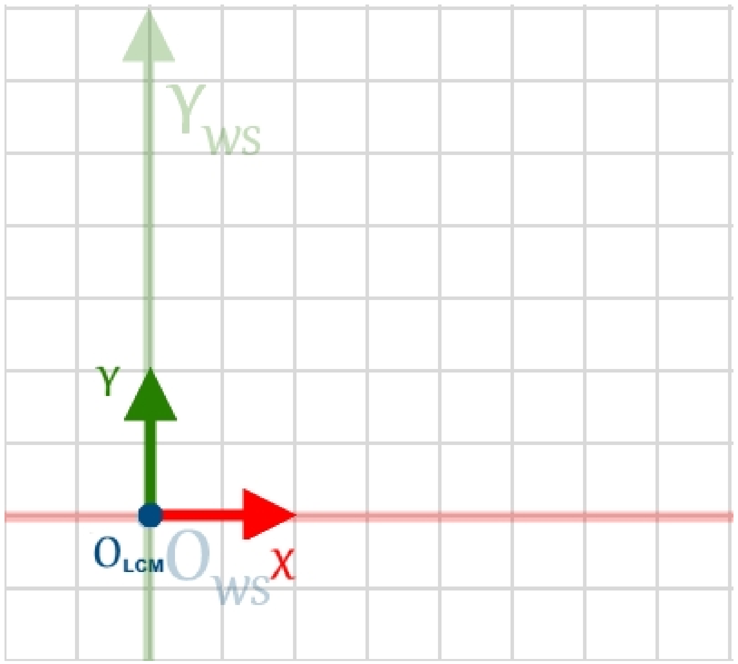
$$\begin{pmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{pmatrix}$$

Practical transformations

The LCM represents a special transformation matrix

- *Modelview matrix*
- When a geometric object is drawn, it is placed according to the transform defined in the Modelview matrix

Identity matrix



Transformation Operations

```
Initialise()
```

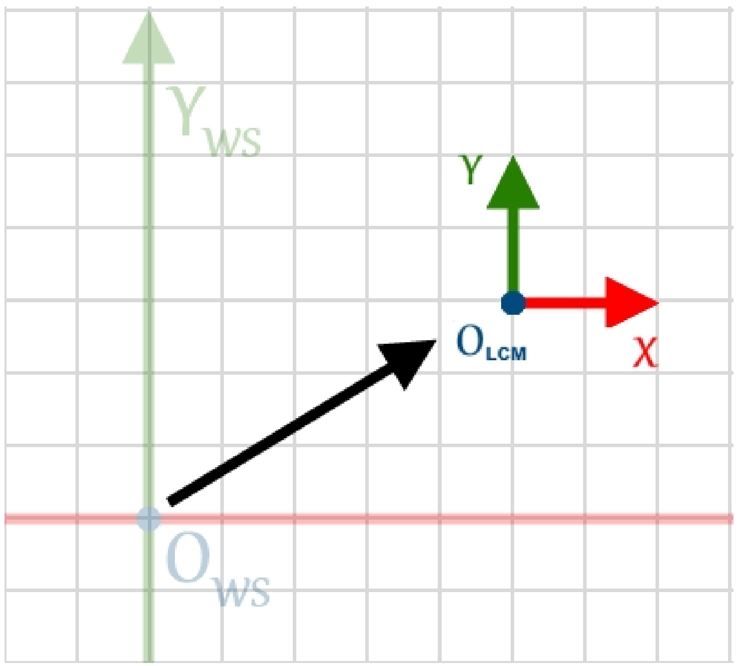
Modelview matrix

$$\begin{pmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{pmatrix}$$

Practical transformations

The LCM represents a special transformation matrix

- *Modelview matrix*
- When a geometric object is drawn, it is placed according to the transform defined in the Modelview matrix



Transformation Operations

```
Initialise()
Translate(5,3)
```

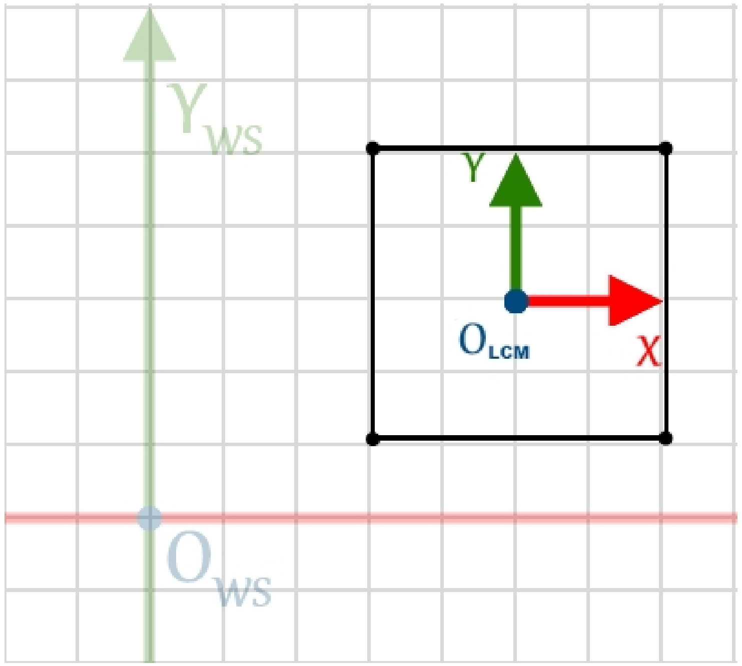
Modelview matrix

$$\begin{pmatrix} 1.0 & 0.0 & 0.0 & 5.0 \\ 0.0 & 1.0 & 0.0 & 3.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{pmatrix}$$

Practical transformations

The LCM represents a special transformation matrix

- *Modelview matrix*
- When a geometric object is drawn, it is placed according to the transform defined in the Modelview matrix



Transformation Operations

```
Initialise()
Translate(5,3)
Draw_Square()
```

Modelview matrix

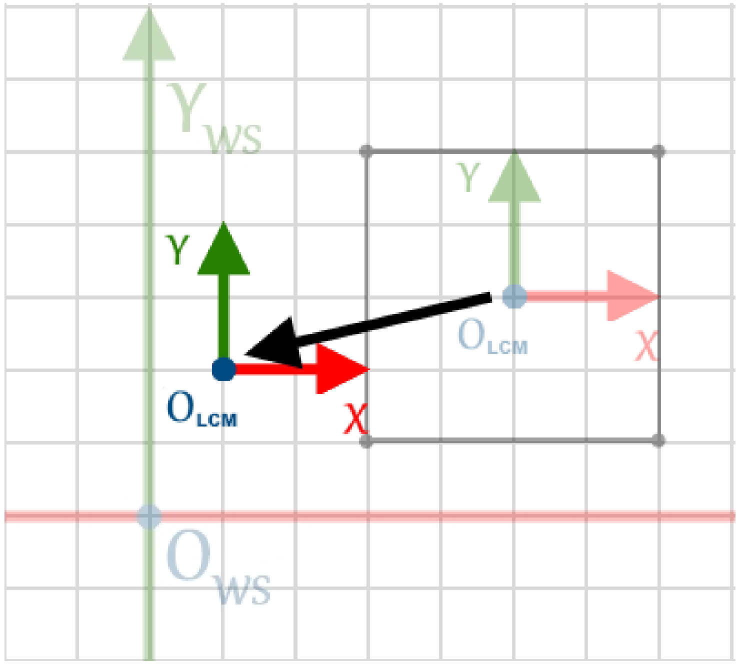
$$\begin{pmatrix} 1.0 & 0.0 & 0.0 & 5.0 \\ 0.0 & 1.0 & 0.0 & 3.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{pmatrix}$$

ROYAL INSTITUTE OF TECHNOLOGY

Practical transformations

The LCM represents a special transformation matrix

- *Modelview matrix*
- When a geometric object is drawn, it is placed according to the transform defined in the Modelview matrix
- Translations and rotations concatenate into the current state of the Modelview matrix



Transformation Operations

```
Initialise()
Translate(5,3)
Draw_Square()
Translate(-4,-1)
```

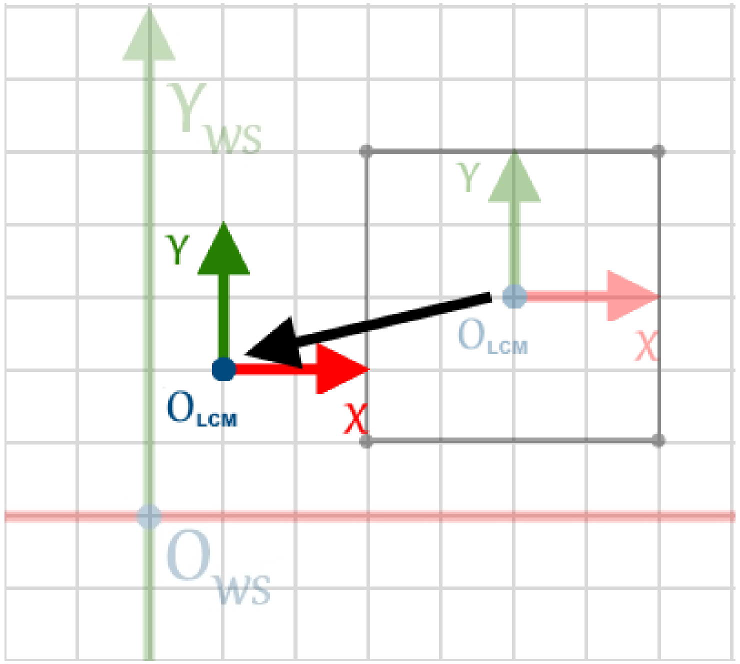
Modelview matrix

$$\begin{pmatrix} 1.0 & 0.0 & 0.0 & 1.0 \\ 0.0 & 1.0 & 0.0 & 2.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{pmatrix}$$

Practical transformations

The LCM represents a special transformation matrix

- *Modelview matrix*
- When a geometric object is drawn, it is placed according to the transform defined in the Modelview matrix
- Translations and rotations concatenate into the current state of the Modelview matrix



Transformation Operations

```
Initialise()
Translate(5,3)
Draw_Square()
Translate(-4,-1)
```

Modelview matrix

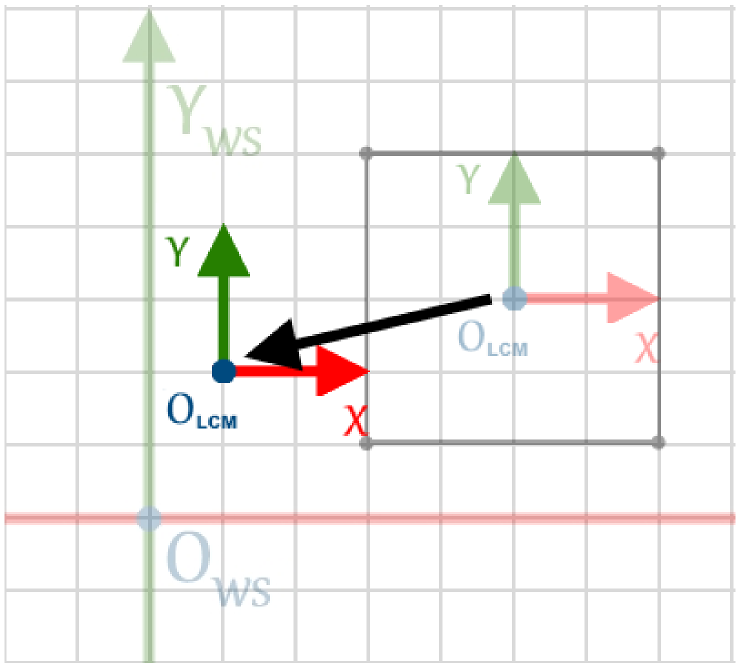
$$\begin{pmatrix} 1.0 & 0.0 & 0.0 & 1.0 \\ 0.0 & 1.0 & 0.0 & 2.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{pmatrix}$$

Displacements

Practical transformations

The LCM represents a special transformation matrix

- *Modelview matrix*
- When a geometric object is drawn, it is placed according to the transform defined in the Modelview matrix
- Translations and rotations concatenate into the current state of the Modelview matrix



Transformation Operations

```
Initialise()
Translate(5,3)
Draw_Square()
Translate(-4,-1)
```

Modelview matrix

$$\begin{pmatrix} 1.0 & 0.0 & 0.0 & 1.0 \\ 0.0 & 1.0 & 0.0 & 2.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{pmatrix}$$

Result

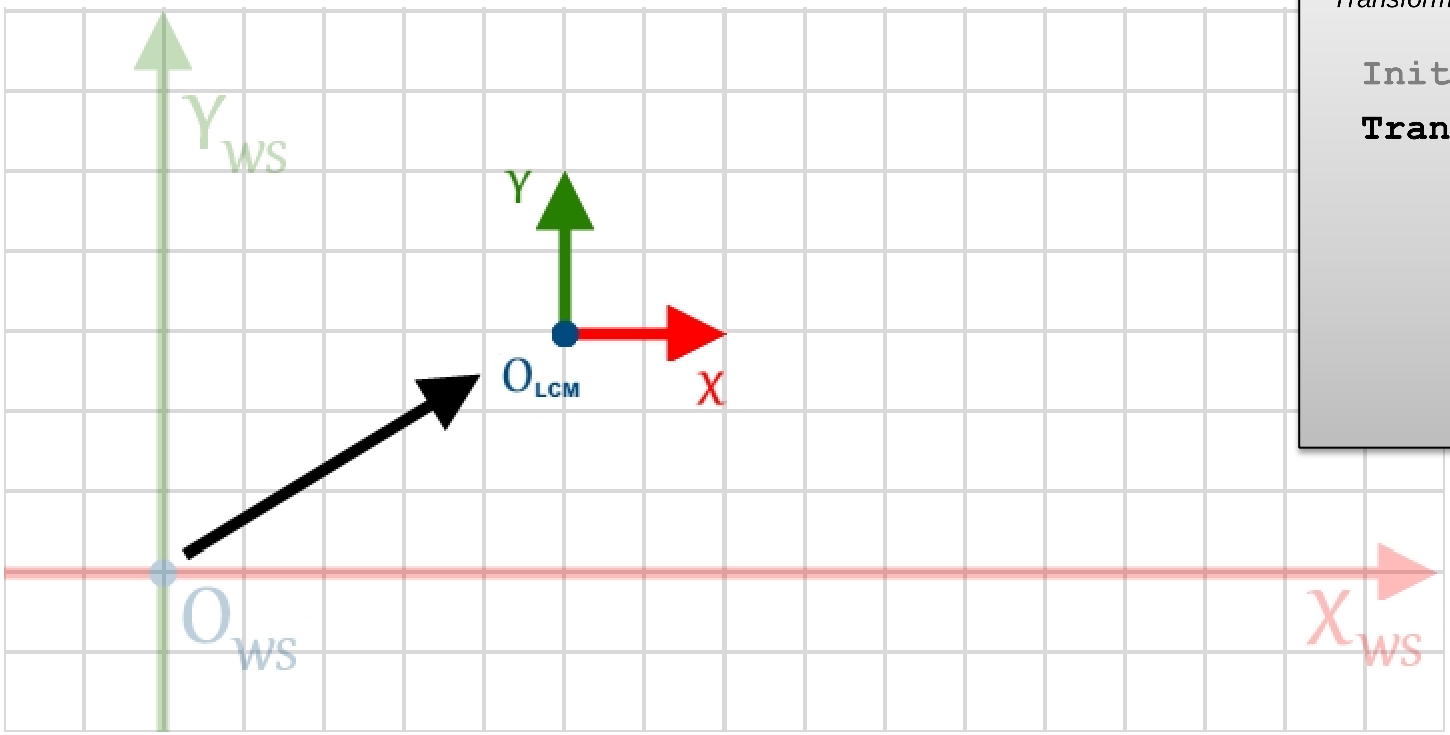
Rotations and translations

Transformation Operations

`Initialise()`

Let's add in some rotations to the mix

Rotations and translations

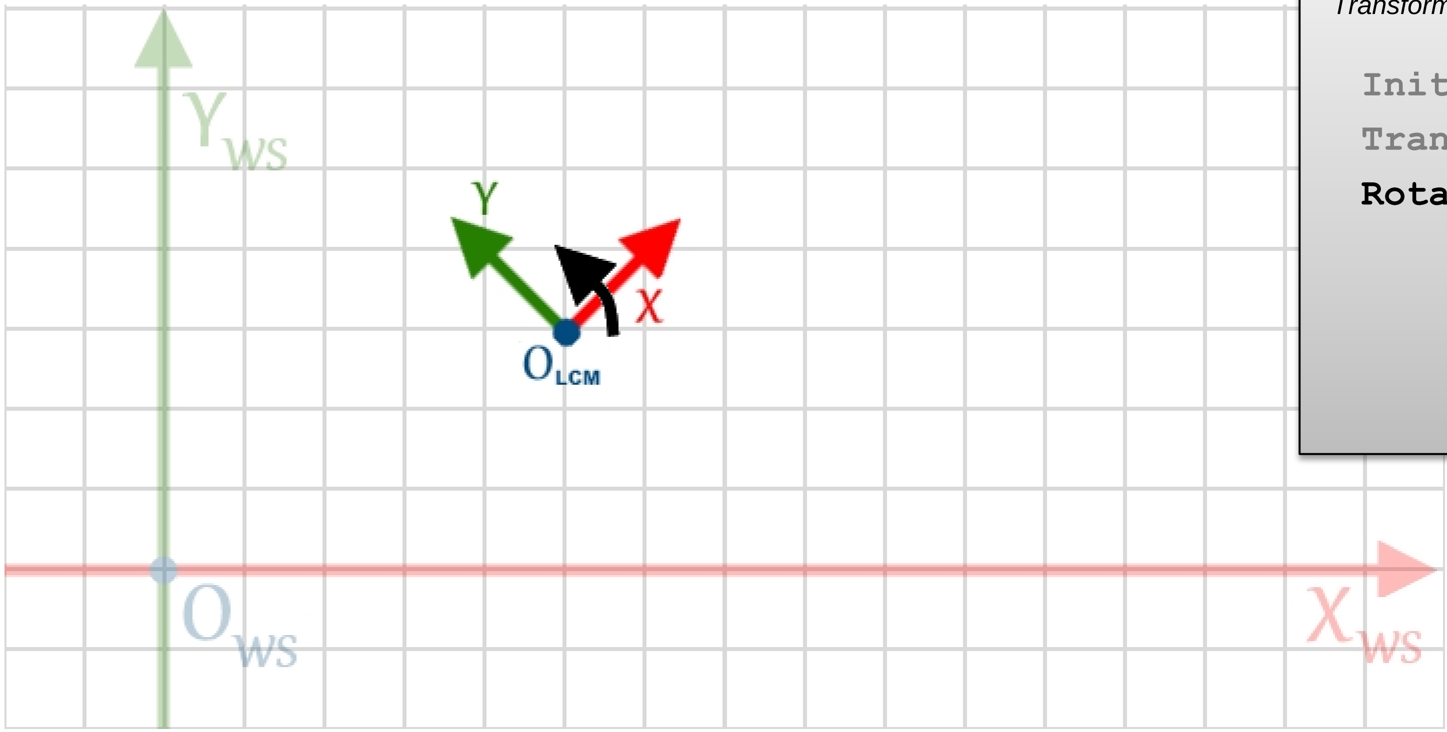


Transformation Operations

```
Initialise()
Translate(5,3)
```

Let's add in some rotations to the mix

Rotations and translations



Transformation Operations

```
Initialise()
Translate(5,3)
Rotate(45)
```

Let's add in some rotations to the mix

Rotations and translations

Transformation Operations

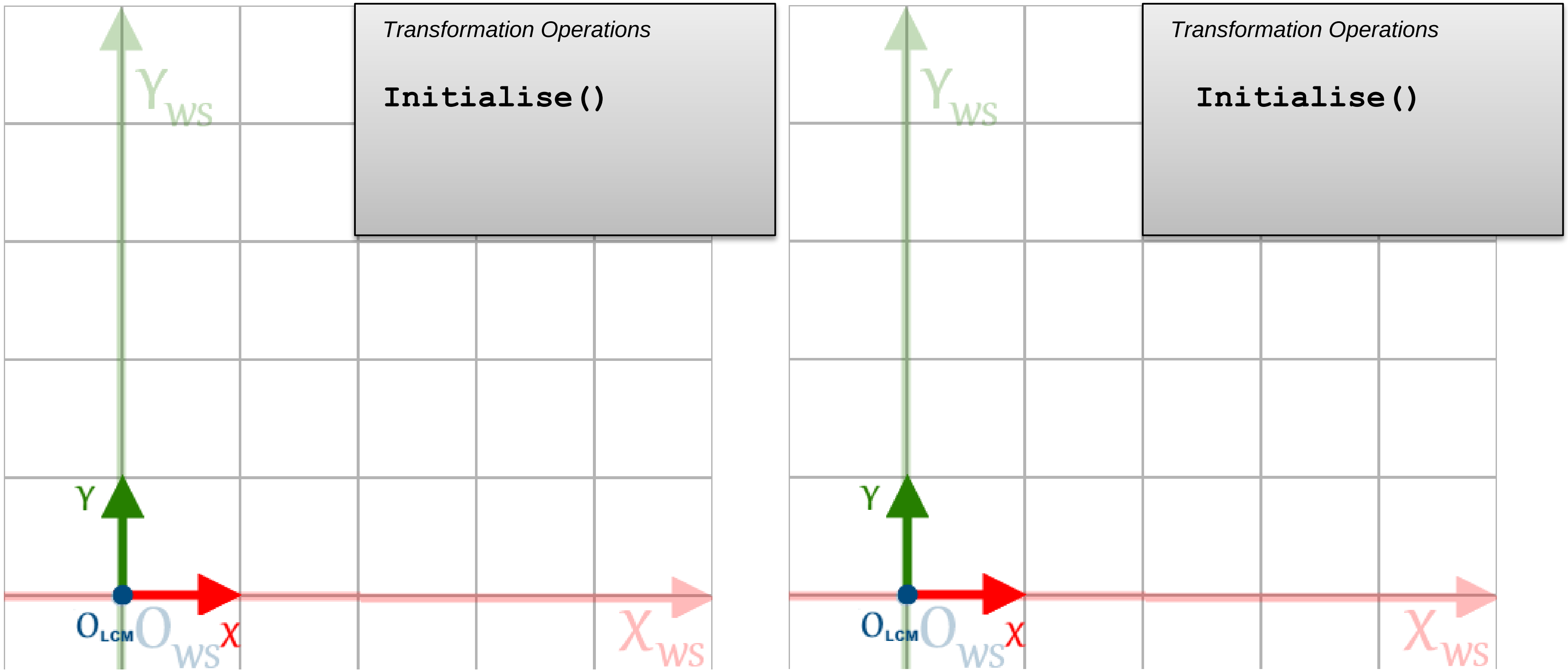
```
Initialise()
Translate(5,3)
Rotate(45)
Translate(2,0)
```

Let's add in some rotations to the mix

Notice how the final translation of (2,0) takes place **with respect to the LCM coordinate system**

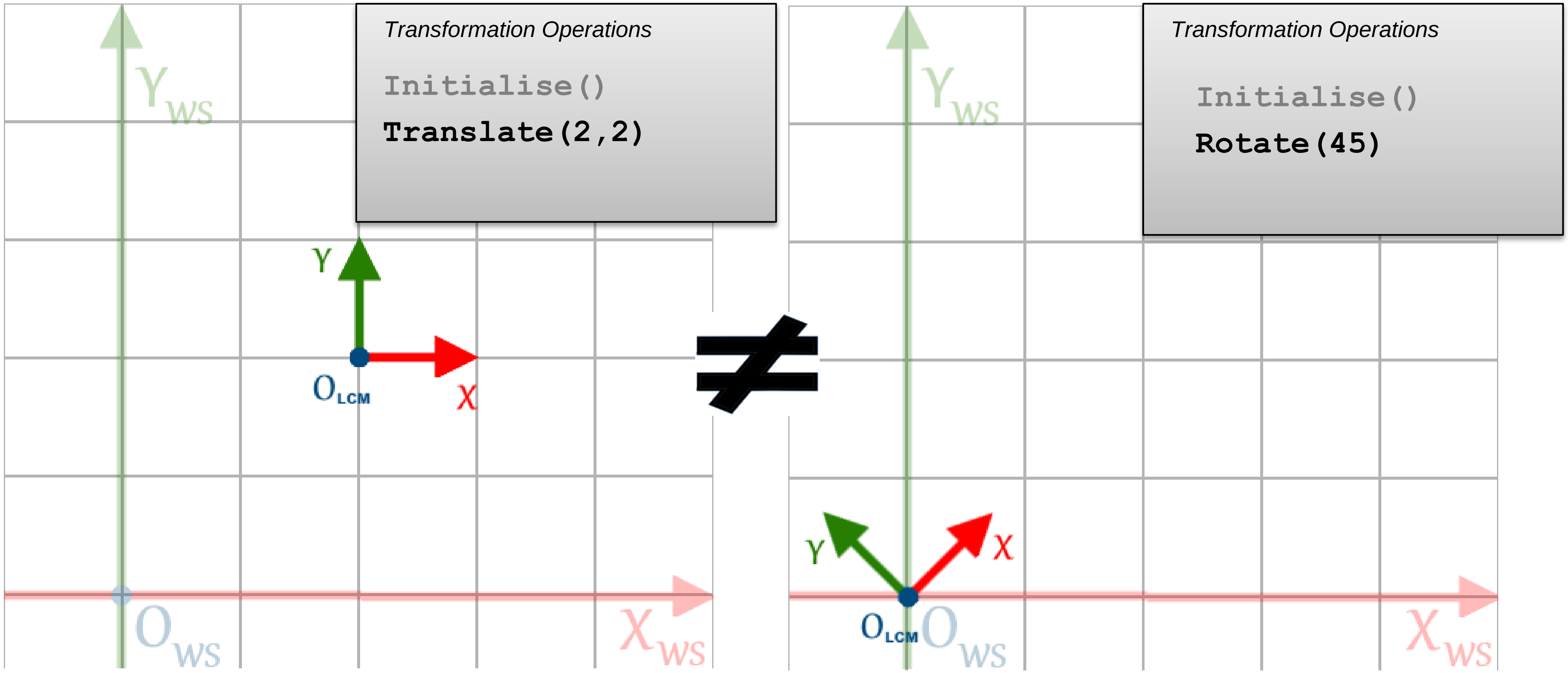
- Not the WS axes

Order matters



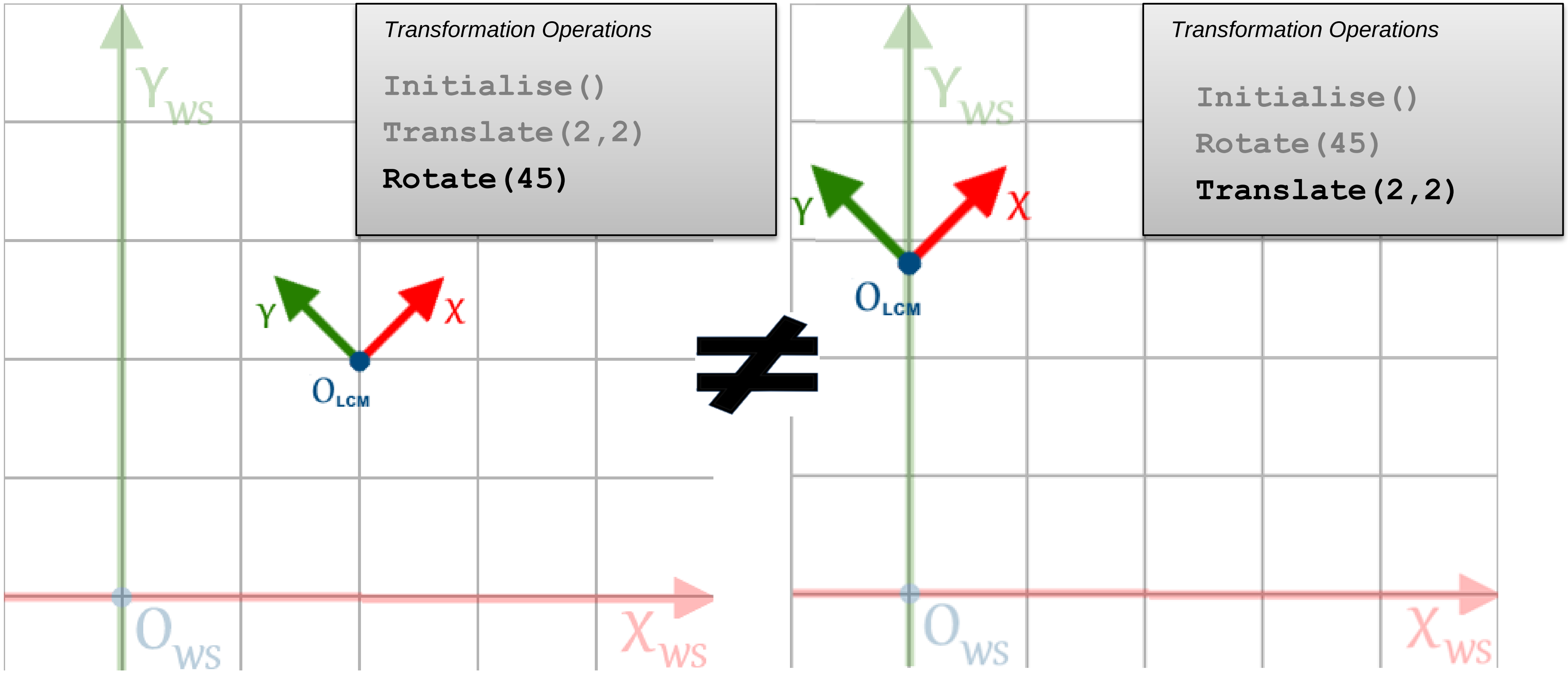
Translation and rotation operations are non-commutative

Order matters



Translation and rotation operations are non-commutative

Order matters

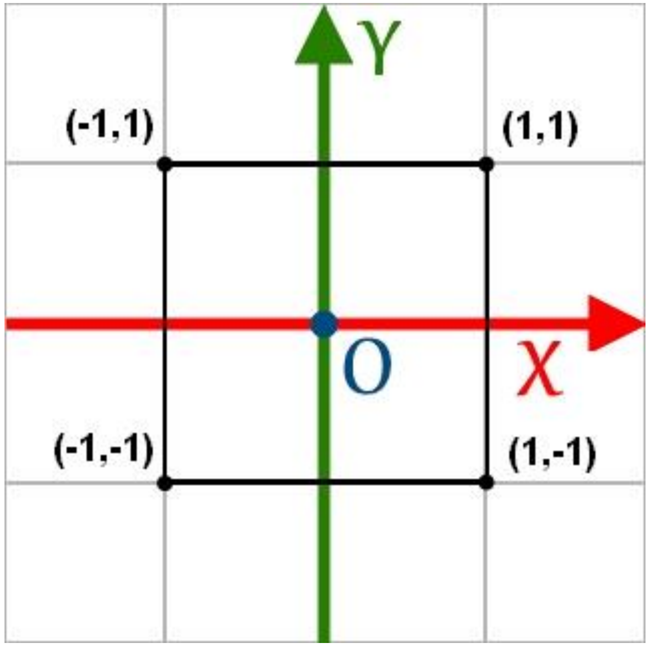


Translation and rotation operations are non-commutative

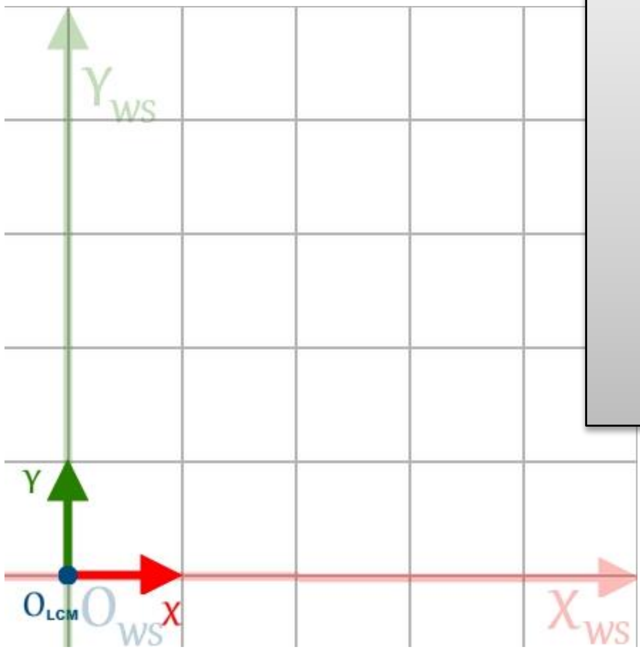
See matrices from last lecture

Object space revisited

Square1 specified in Object space (OS)



Positioning in world space (WS) via transform



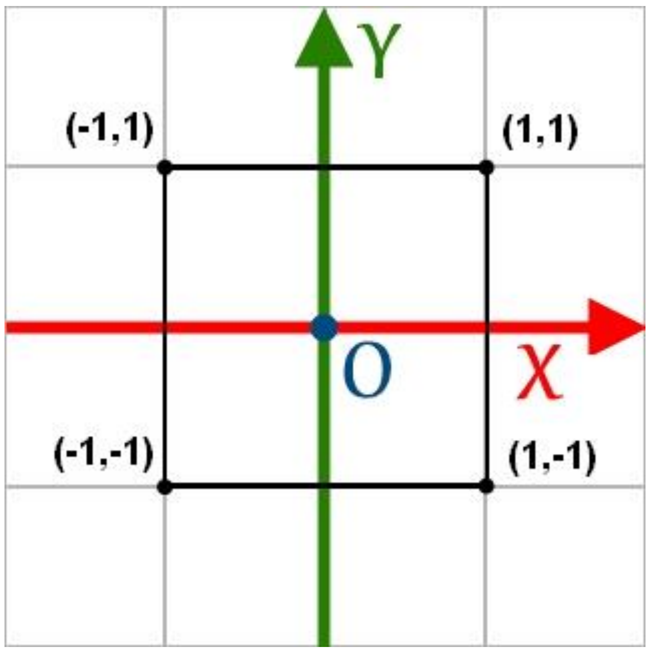
Transformation Operations

```
Initialise()
```

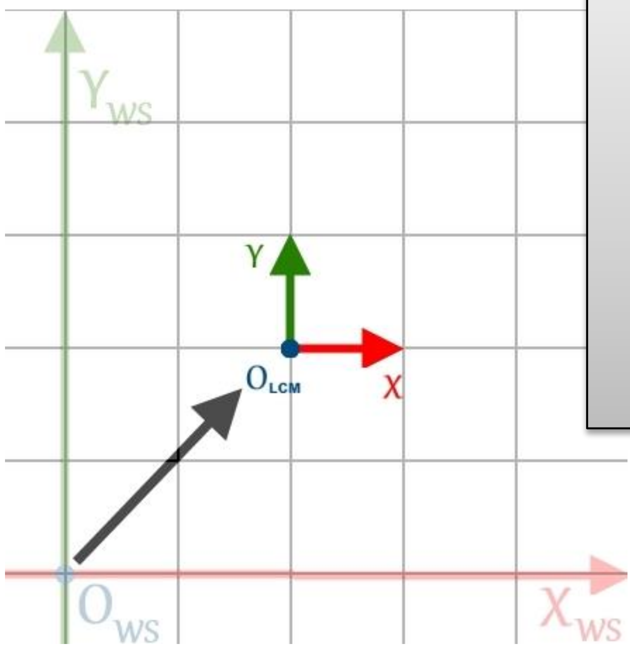
Example 1: Objects are placed in world space according to their corresponding origin in object space

Object space revisited

Square1 specified in Object space (OS)



Positioning in world space (WS) via transform



Transformation Operations

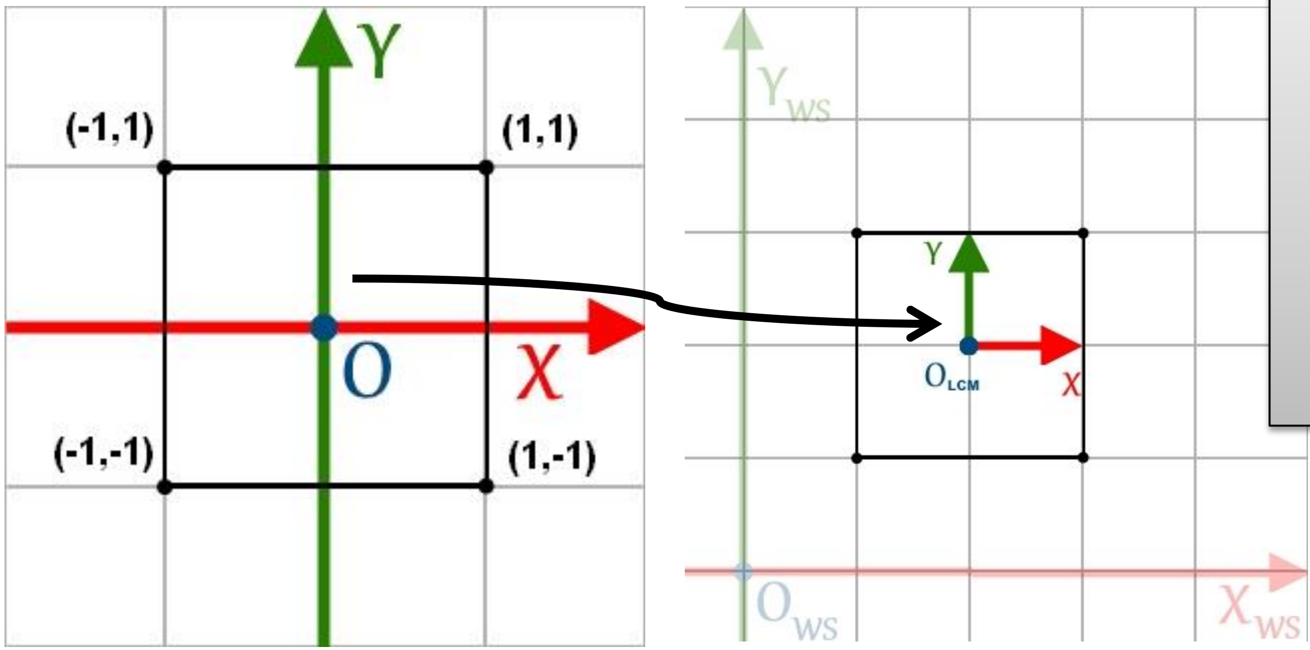
```
Initialise()
Translate(2,2)
```

Example 1: Objects are placed in world space according to their corresponding origin in object space

Object space revisited

Square1 specified in Object space (OS)

Positioning in world space (WS) via transform



Transformation Operations

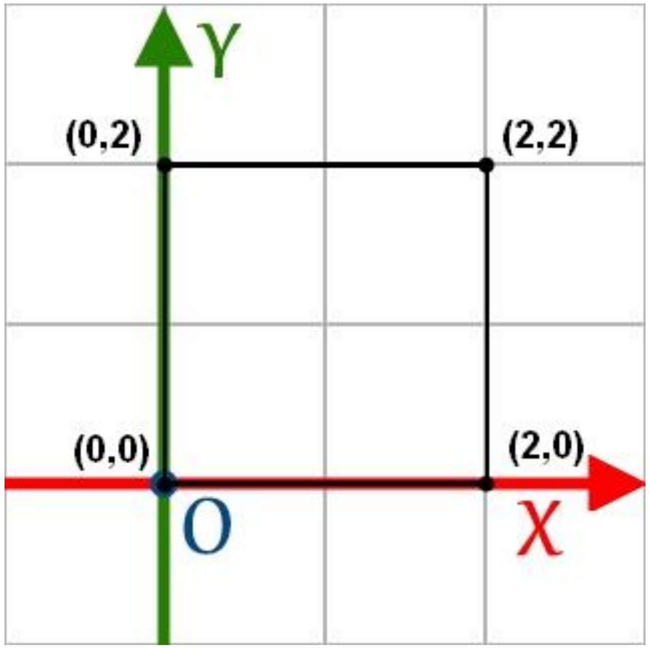
```
Initialise()
Translate(2,2)
Draw_Square1()
```

Example 1: Objects are placed in world space according to their corresponding origin in object space

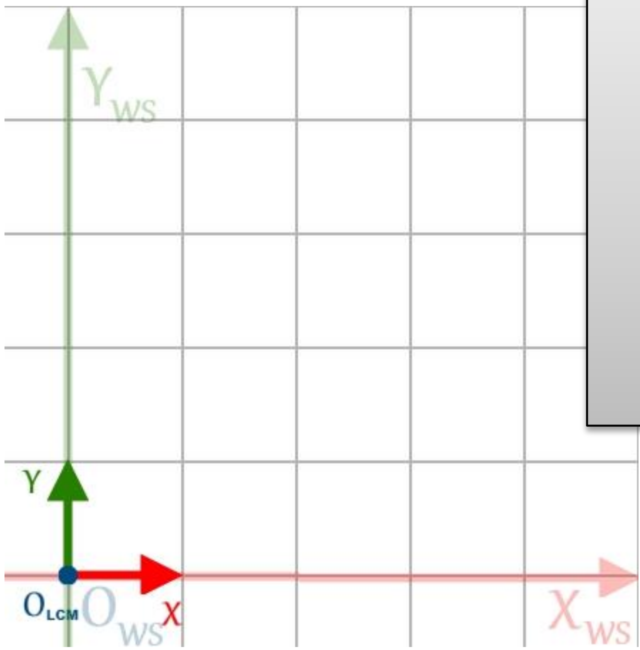
i.e. Object space origin is mapped onto the LCM

Object space revisited

Square2 specified in Object space (OS)



Positioning in world space (WS) via transform



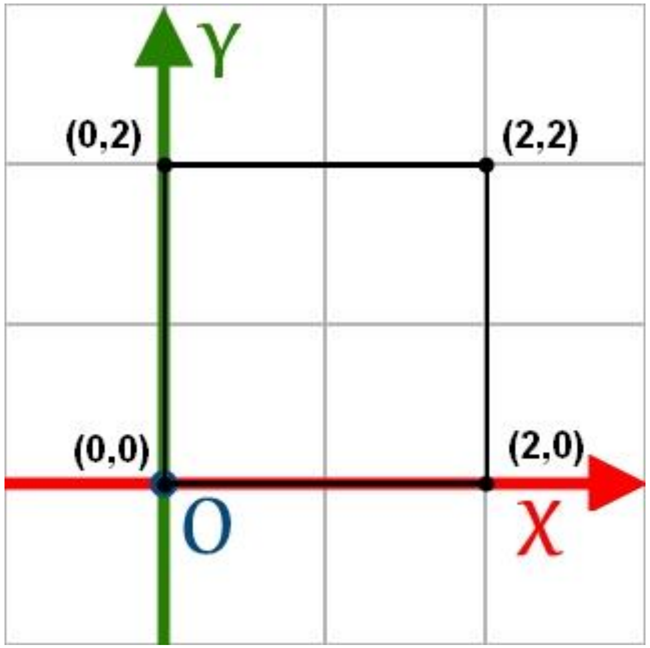
Transformation Operations

```
Initialise()
```

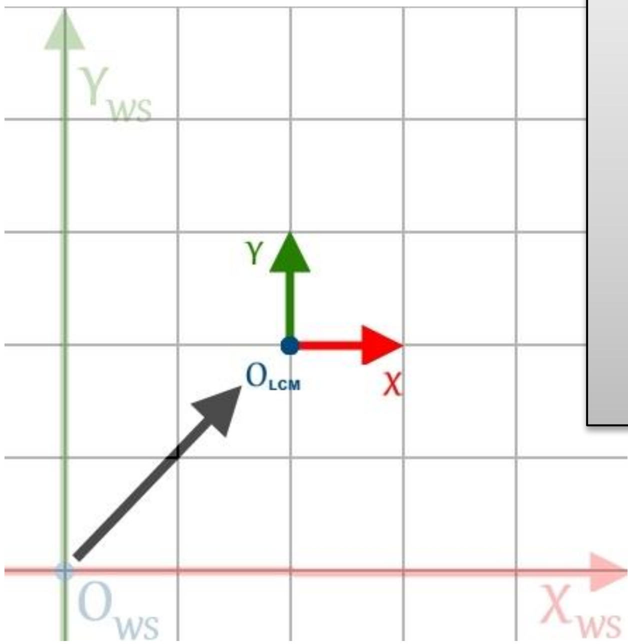
Example 2: Objects are placed in world space according to their corresponding origin in object space

Object space revisited

Square2 specified in Object space (OS)



Positioning in world space (WS) via transform



Transformation Operations

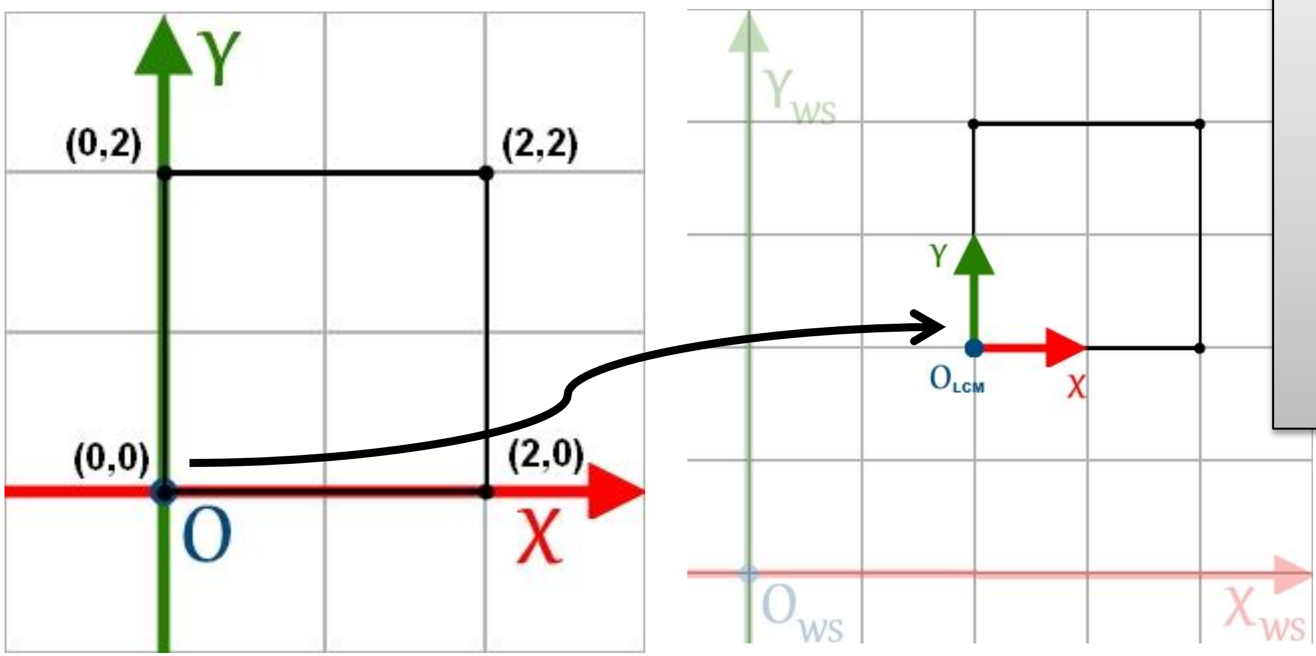
```
Initialise()
Translate(2,2)
```

Example 2: Objects are placed in world space according to their corresponding origin in object space

Object space revisited

Square2 specified in Object space (OS)

Positioning in world space (WS) via transform



Transformation Operations

```
Initialise()
Translate(2,2)
Draw_Square2()
```

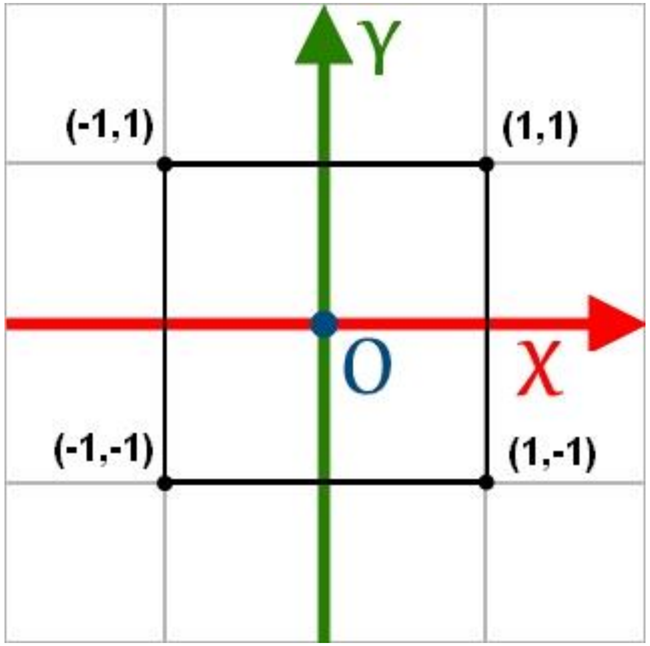
Example 2: Objects are placed in world space according to their corresponding origin in object space

i.e. Object space origin is mapped onto the LCM

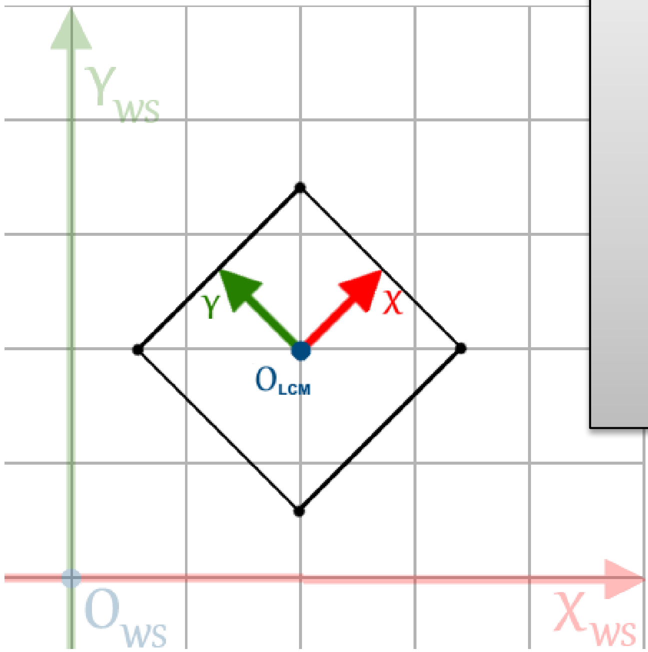
Notice here that the LCM (transformation) is the exact same as in example 1

Object space revisited

Square1 specified in Object space (OS)



Positioning in world space (WS) via transform



Transformation Operations

```
Initialise()
Translate(2,2)
Rotate(45)
Draw_Square1()
```

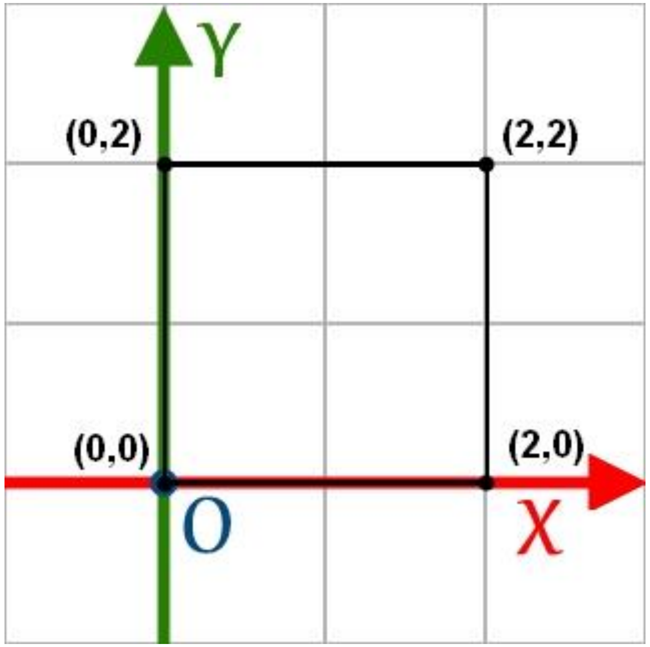
Rotations also occur about the origin of the object

- Default *axis of rotation*

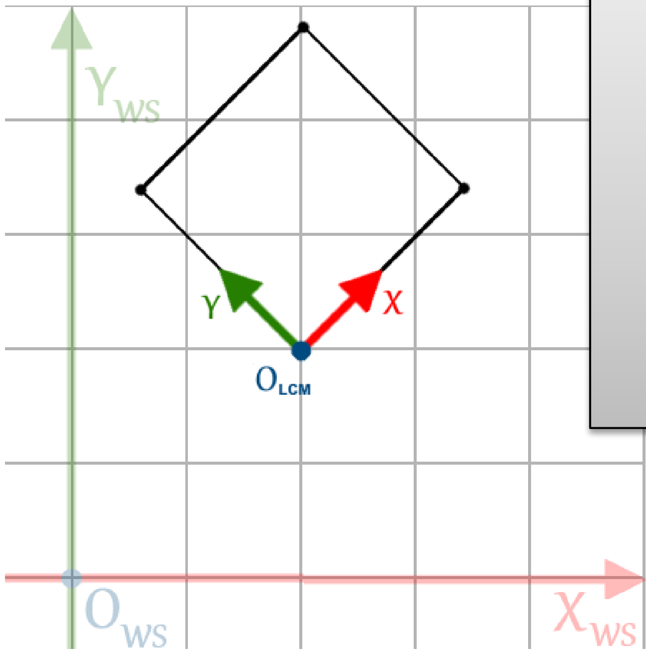
Notice that the transformation is the exact same

Object space revisited

Square2 specified in Object space (OS)



Positioning in world space (WS) via transform



Transformation Operations

```
Initialise()
Translate(2,2)
Rotate(45)
Draw_Square2()
```

Rotations also occur about the origin of the object

- Default *axis of rotation*

Notice that the transformation is the exact same

Saving and loading transformations

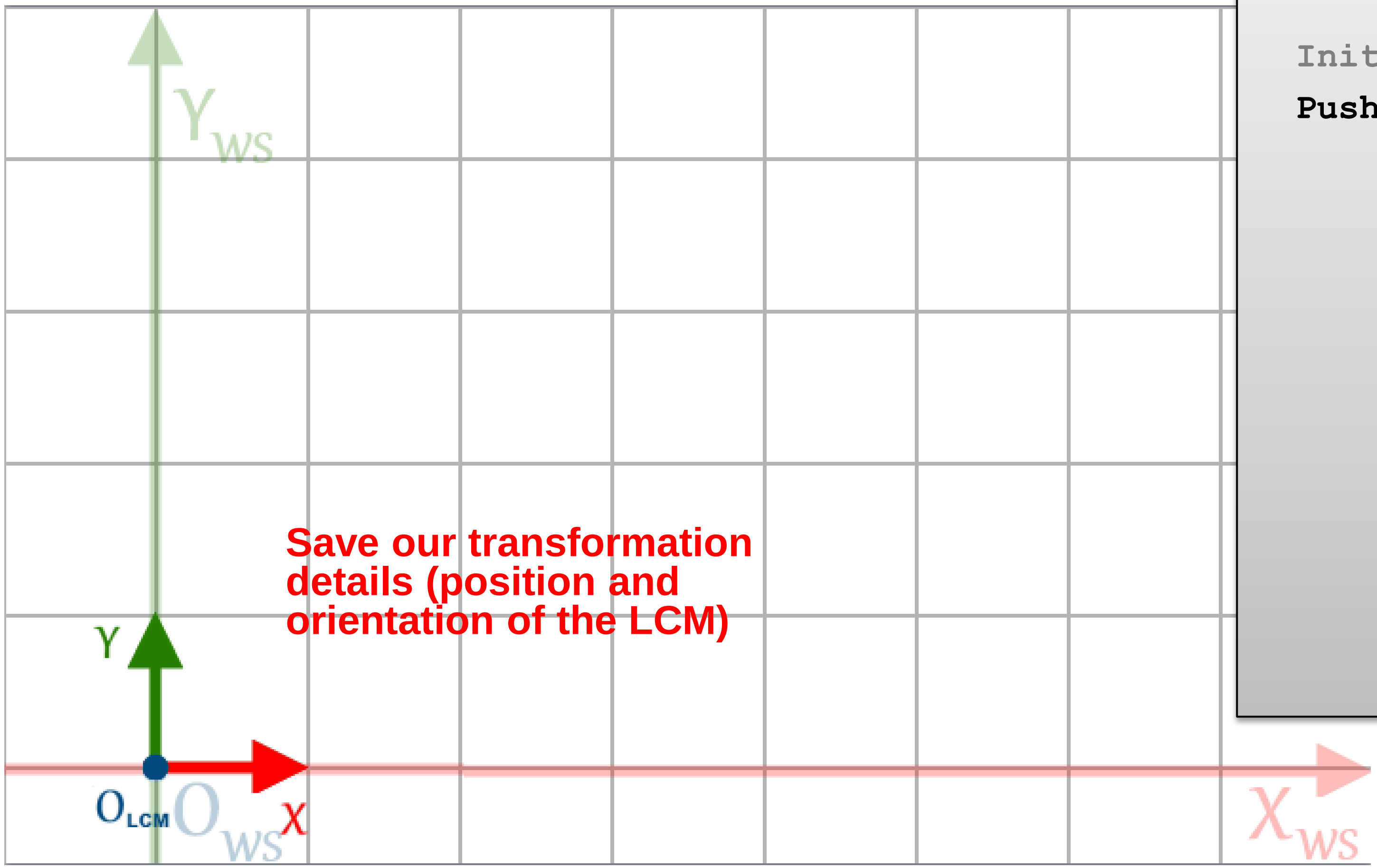
When positioning multiple objects, saving and loading transformations can be useful

Transformation Operations

```
Initialise()
```

Saving and loading transformations

When positioning multiple objects, saving and loading transformations can be useful



Transformation Operations

```
Initialise()
PushMatrix()
```

Saving and loading transformations

When positioning multiple objects, saving and loading transformations can be useful

Transformation Operations

```
Initialise()
PushMatrix()
  Translate(2,2)
```

Saving and loading transformations

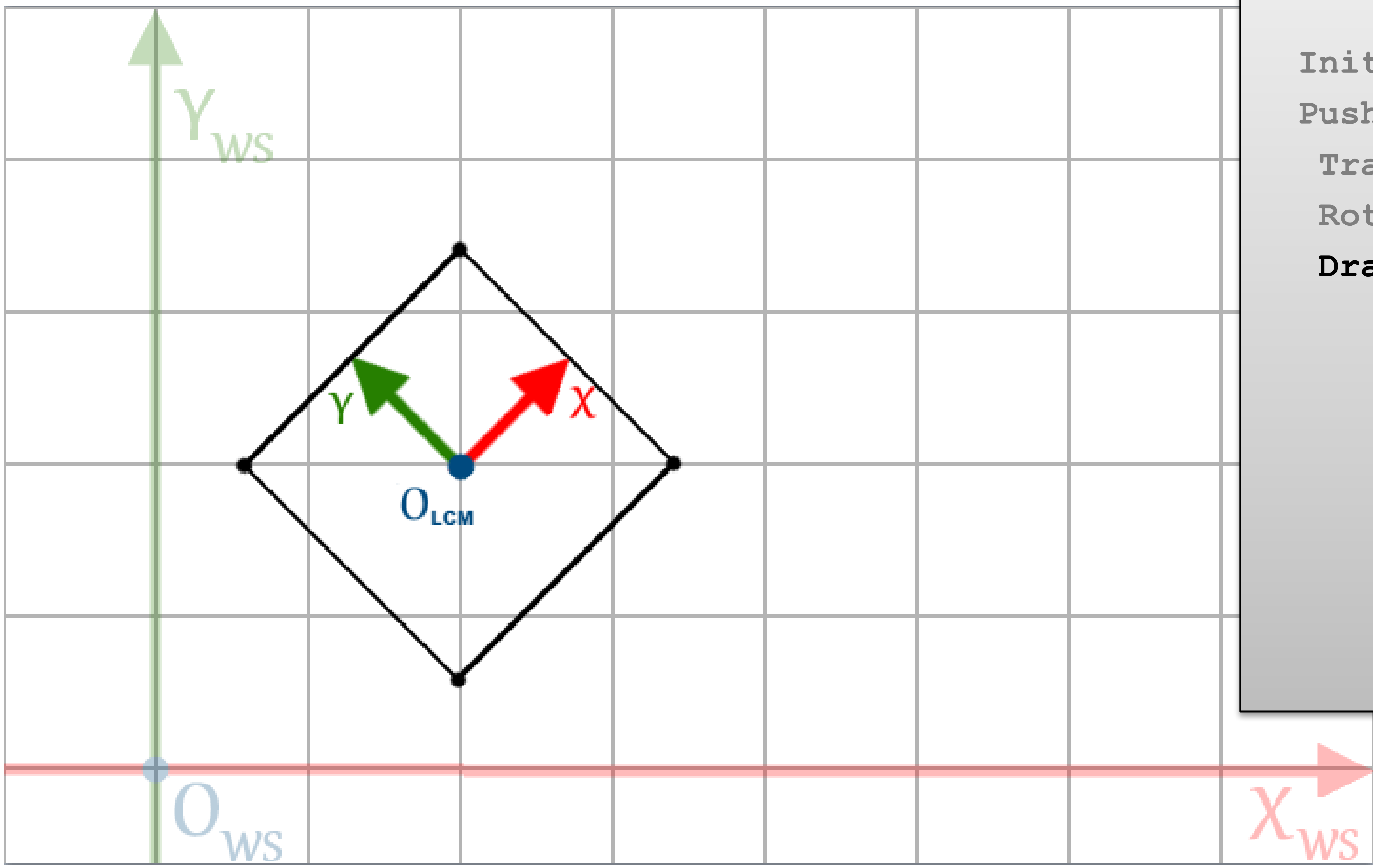
When positioning multiple objects, saving and loading transformations can be useful

Transformation Operations

```
Initialise()
PushMatrix()
  Translate(2,2)
  Rotate(45)
```

Saving and loading transformations

When positioning multiple objects, saving and loading transformations can be useful

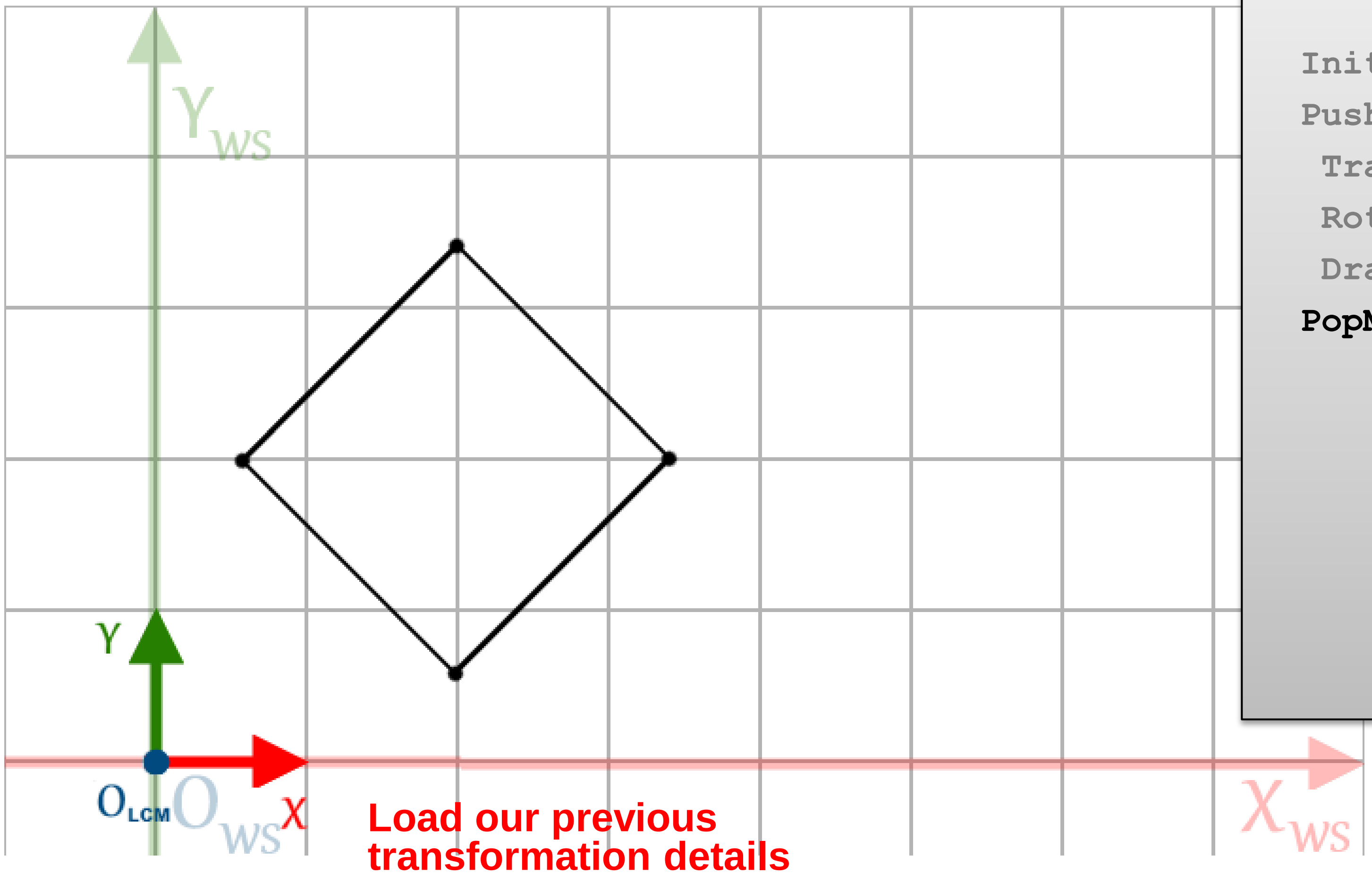


Transformation Operations

```
Initialise()
PushMatrix()
  Translate(2,2)
  Rotate(45)
  Draw_Square()
```

Saving and loading transformations

When positioning multiple objects, saving and loading transformations can be useful



Transformation Operations

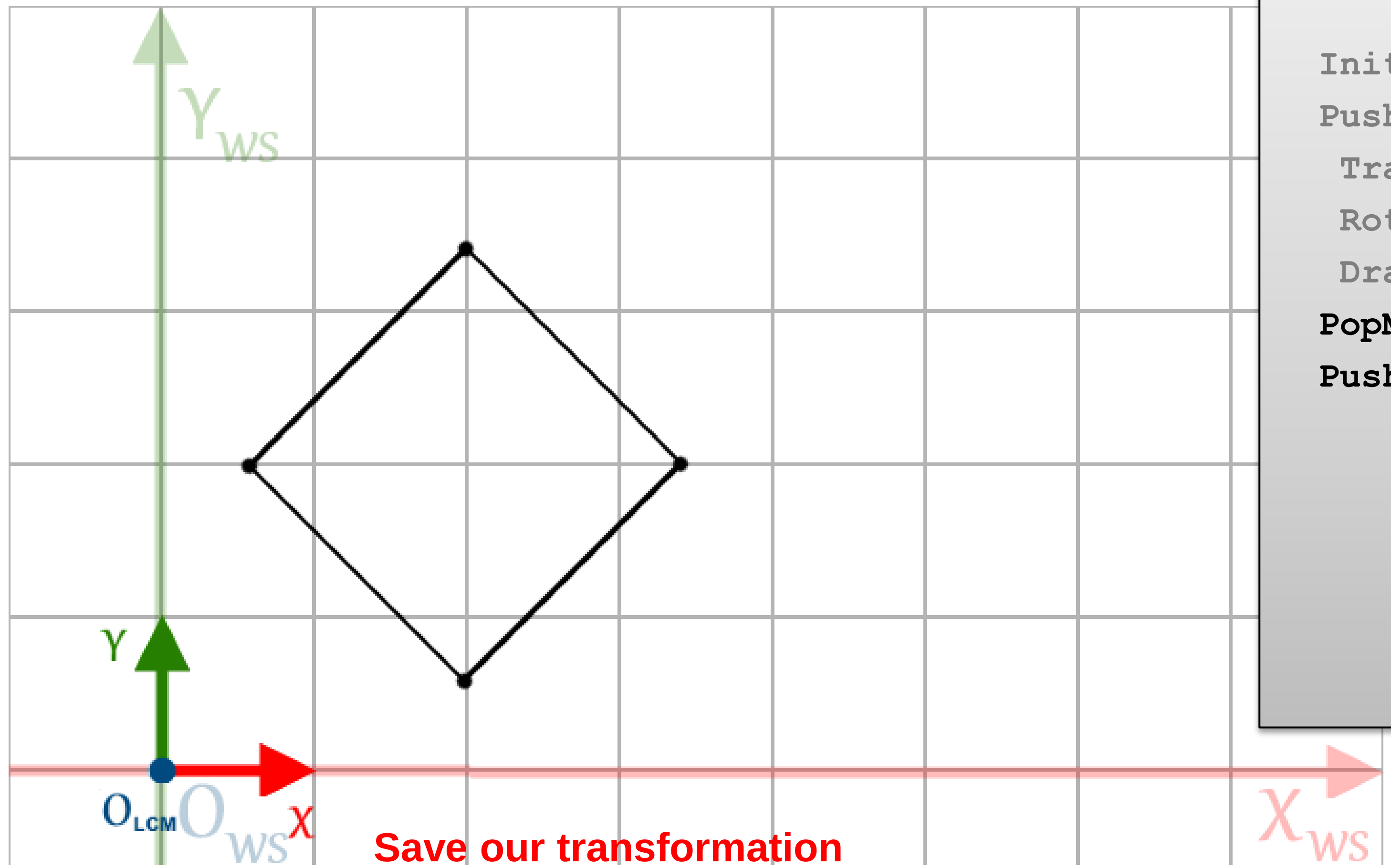
```
Initialise()
PushMatrix()
  Translate(2,2)
  Rotate(45)
  Draw_Square()
PopMatrix()
```

Load our previous transformation details

(another option in this case: re-initialise the Modelview matrix)

Saving and loading transformations

When positioning multiple objects, saving and loading transformations can be useful



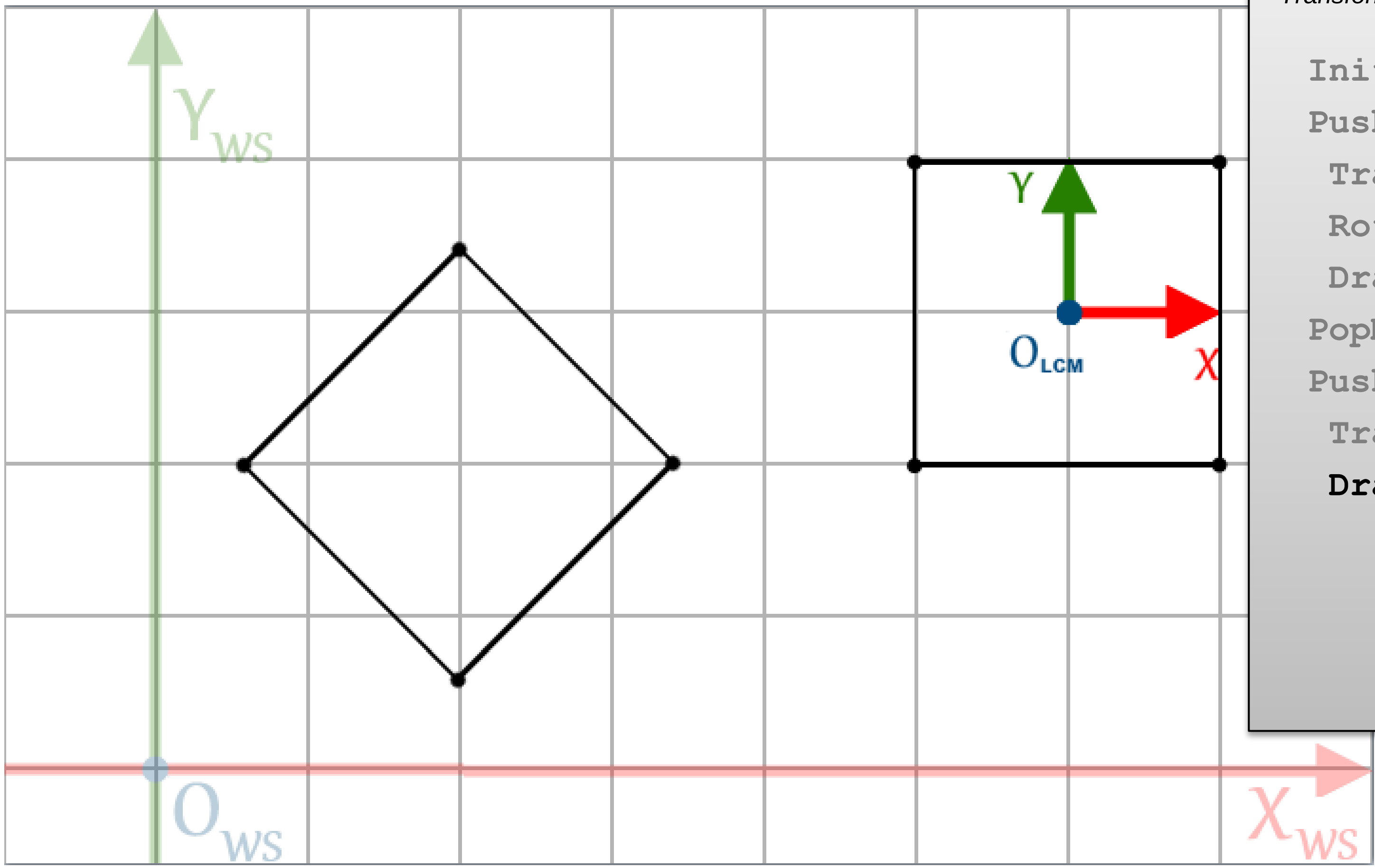
Transformation Operations

```
Initialise()
PushMatrix()
  Translate(2,2)
  Rotate(45)
  Draw_Square()
PopMatrix()
PushMatrix()
```

Save our transformation details (position and orientation of the LCM)

Saving and loading transformations

When positioning multiple objects, saving and loading transformations can be useful

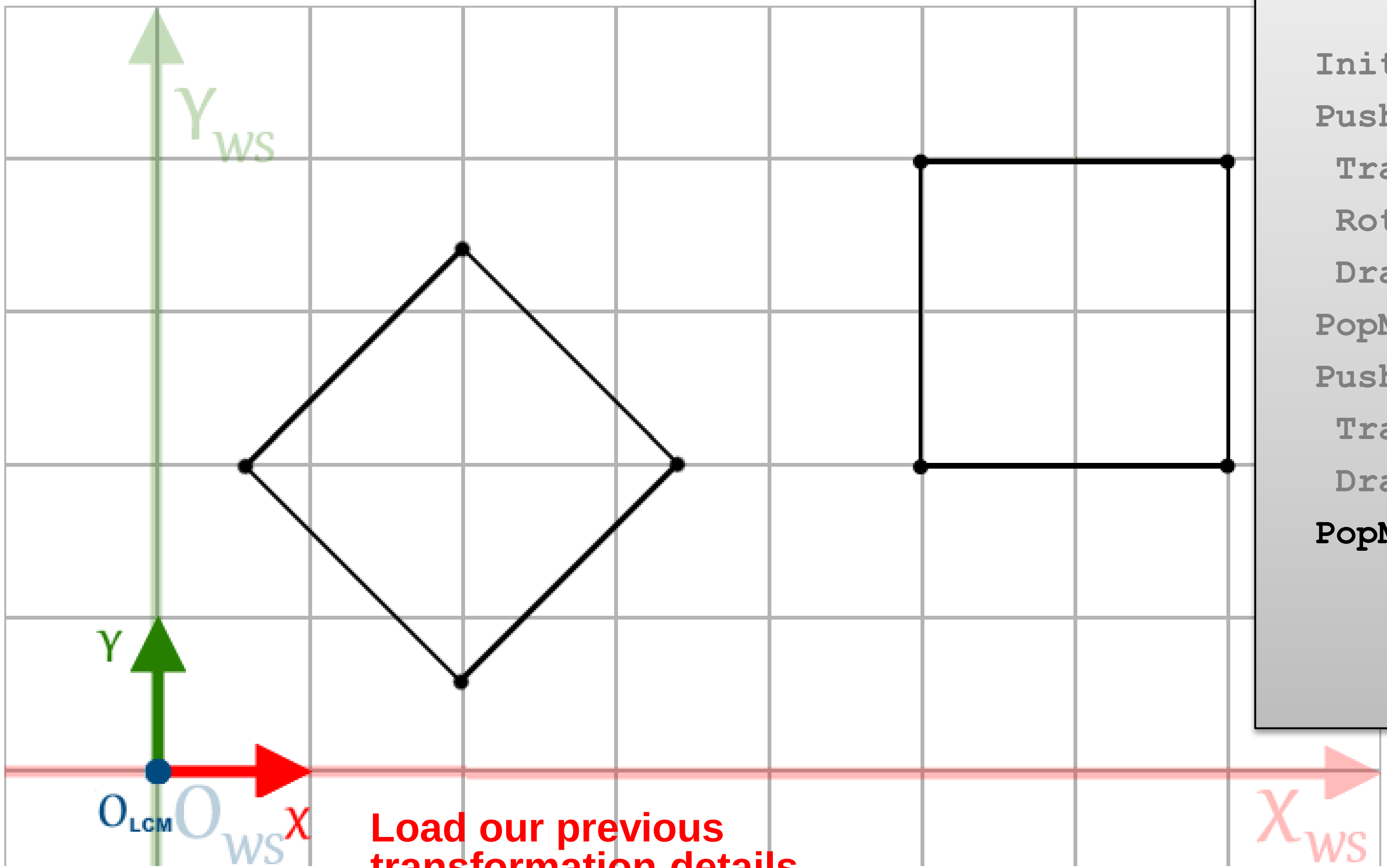


Transformation Operations

```
Initialise()
PushMatrix()
  Translate(2,2)
  Rotate(45)
  Draw_Square()
PopMatrix()
PushMatrix()
  Translate(6,3)
  Draw_Square()
```

Saving and loading transformations

When positioning multiple objects, saving and loading transformations can be useful



Transformation Operations

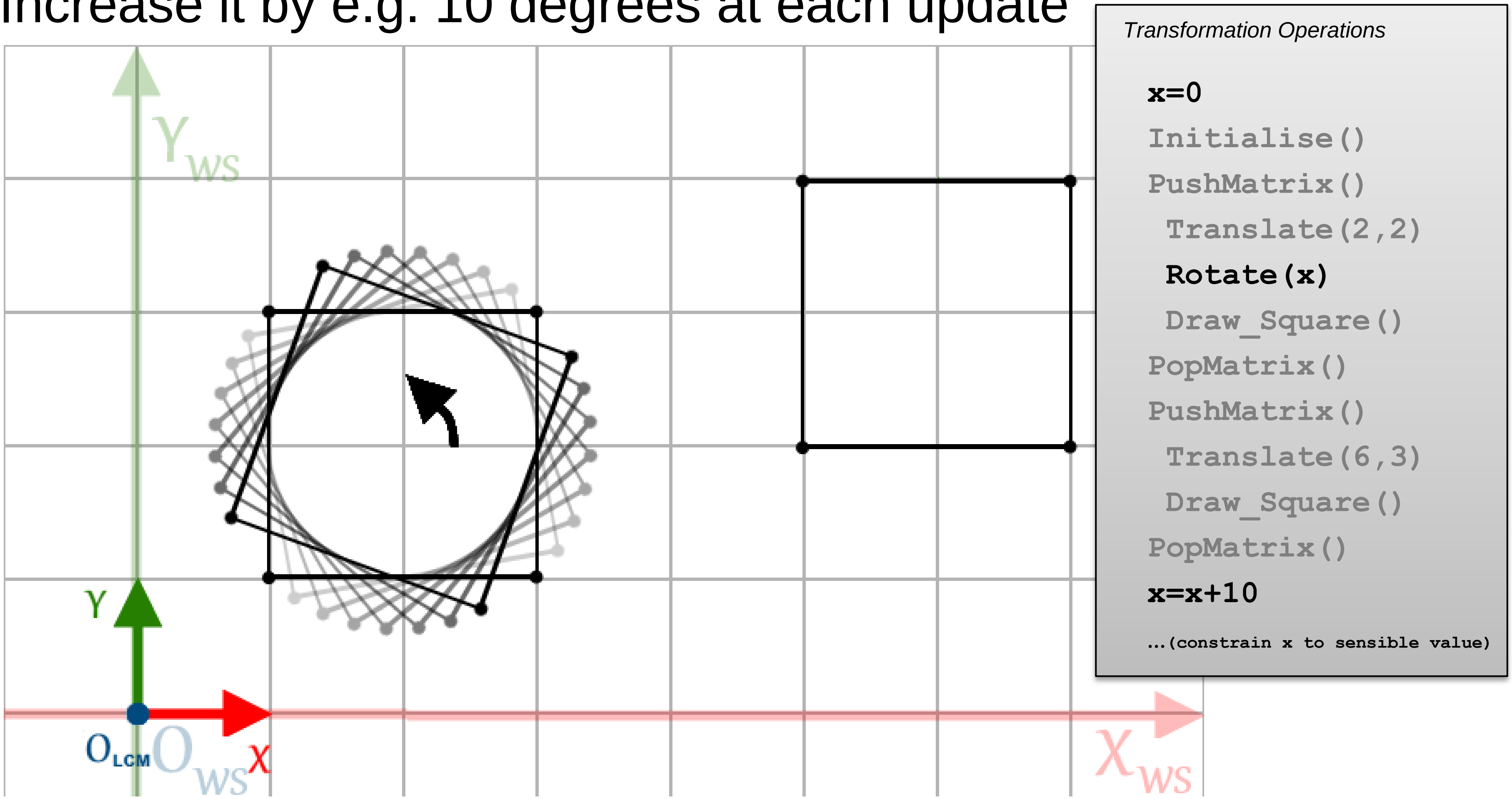
```
Initialise()
PushMatrix()
  Translate(2,2)
  Rotate(45)
  Draw_Square()
PopMatrix()
PushMatrix()
  Translate(6,3)
  Draw_Square()
PopMatrix()
```

Load our previous transformation details

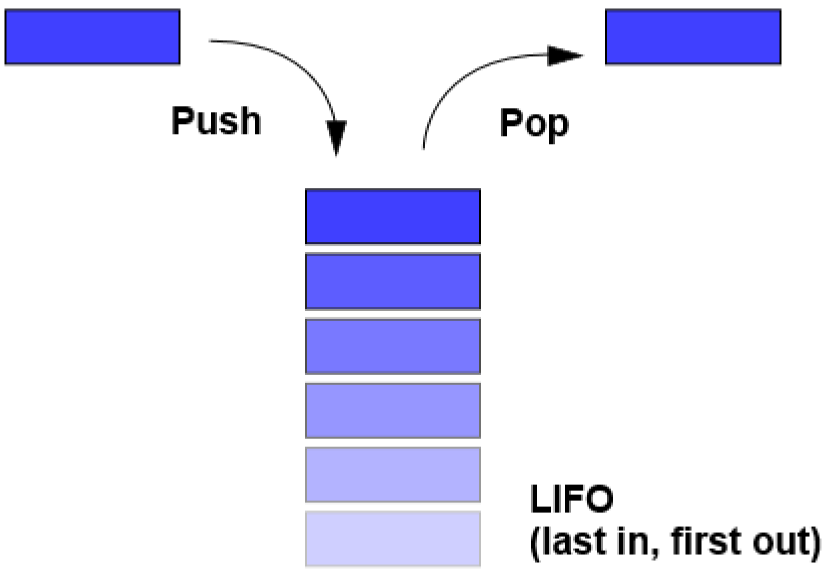
(another option in this case: re-initialise the Modelview matrix)

Adding some animation

Enter a variable angle for the first rotate
Increase it by e.g. 10 degrees at each update



The stack



Transformations are saved on and loaded from a *stack* data structure

Saving a matrix = *push* operation

Loading a matrix = *pop* operation

LIFO (last in, first out)

- Push on to the top of the stack
- Pop off the top of the stack

Operations summary

`Initialise()`

Initialise an identity transformation

Identity matrix (look for functions with similar names to **`LoadIdentity()`**)

`Translate(`t_x`,`t_y`)`

Matrix multiplication

`Rotate(degrees)`

Usually also specify an axis of rotation

In our examples, assume it is (0,0,1)

Rotations around the z axis i.e. in the XY plane

`PushMatrix()`

– Save the current Modelview matrix state on stack

`PopMatrix()`

– Load a previous Modelview matrix state from stack

Introducing hierarchies

A tree of separate objects that move relative to each other

- The positions and orientations of objects further down the tree are dependent on those higher up
- Parent and child objects
- Transformations applied to parents are also applied down the hierarchy to their children

Examples:

1. The human arm (and body)

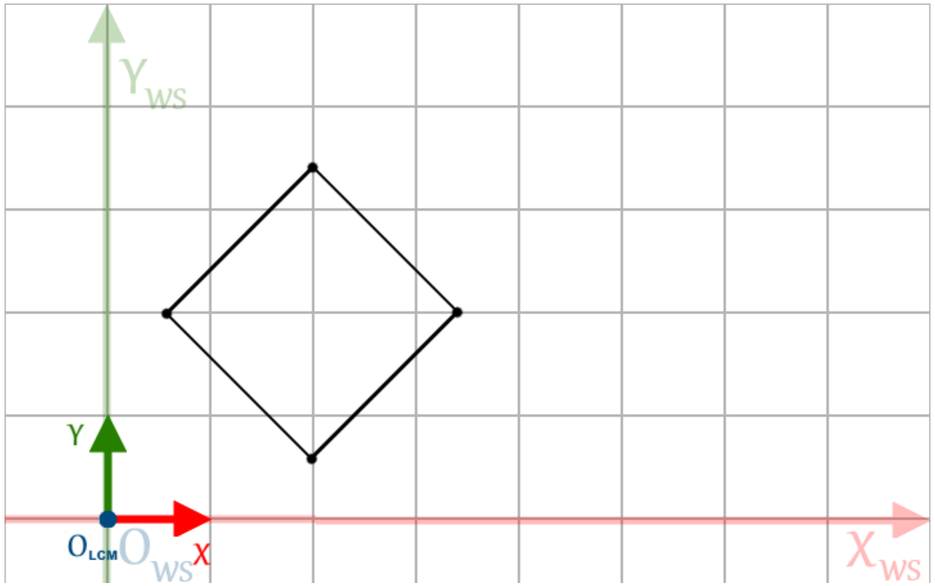
   Hand configuration depends the elbow configuration, depends on shoulder configuration, and so on…

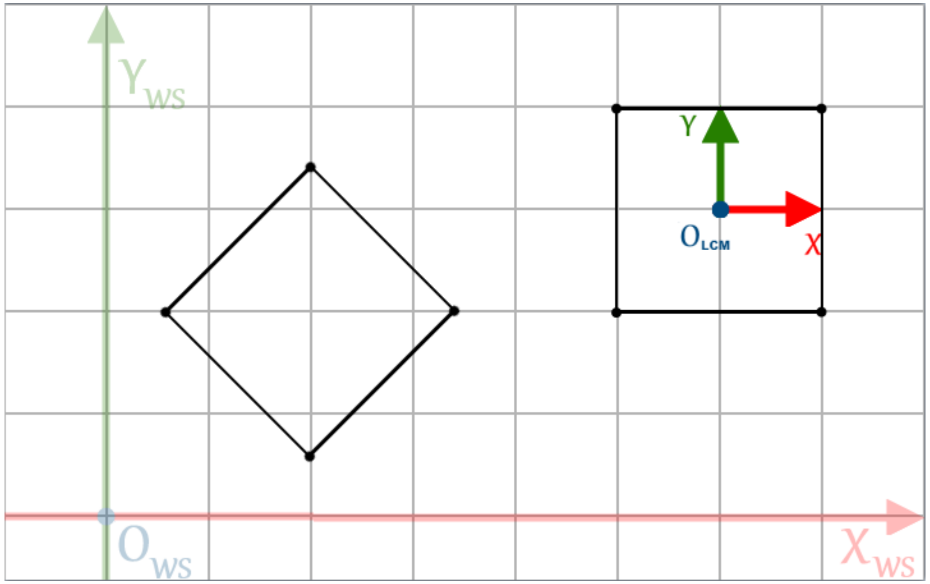
2. The Solar system

   Solar bodies rotate about their own axes as well as orbiting around the Sun (moons around planets, planets around the Sun)

Hierarchies

- You have already learned the basic operations necessary for hierarchical transformations
- Recall: up to now, the LCM has been moved back to the world-space origin before placing each object





ROYAL INSTITUTE OF TECHNOLOGY

Hierarchies

It’s slightly different in a hierarchy

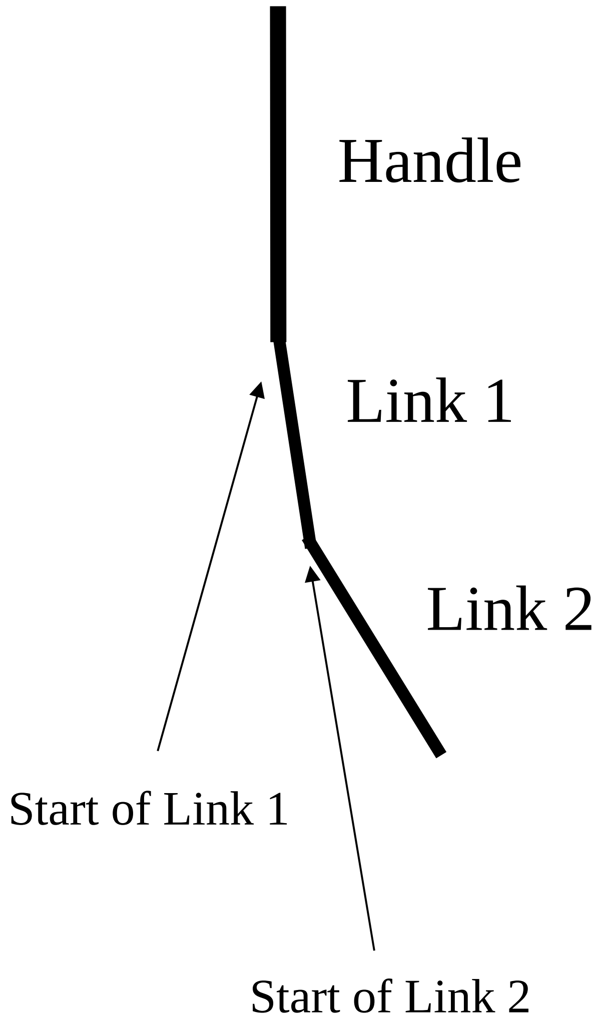
- Objects depend on others (a parent object) for their configurations (position and orientation)
- These objects need to be placed relative to their parent objects’ coordinates, rather than in world-space

In practice, this involves the use of nested **`PushMatrix()`** and **`PopMatrix()`** operations

- Especially when there are multiple *branches*

Simple chain example

- Three components
  - A handle
  - Two links
- In order to define a simple connected chain:
  - Translate the handle location and draw it
  - Translate to the first link and draw it
  - Translate to the second link and draw it
- Note: we do not translate back to the world-space origin after drawing each component
  - i.e. translations are relative to the respective parent objects

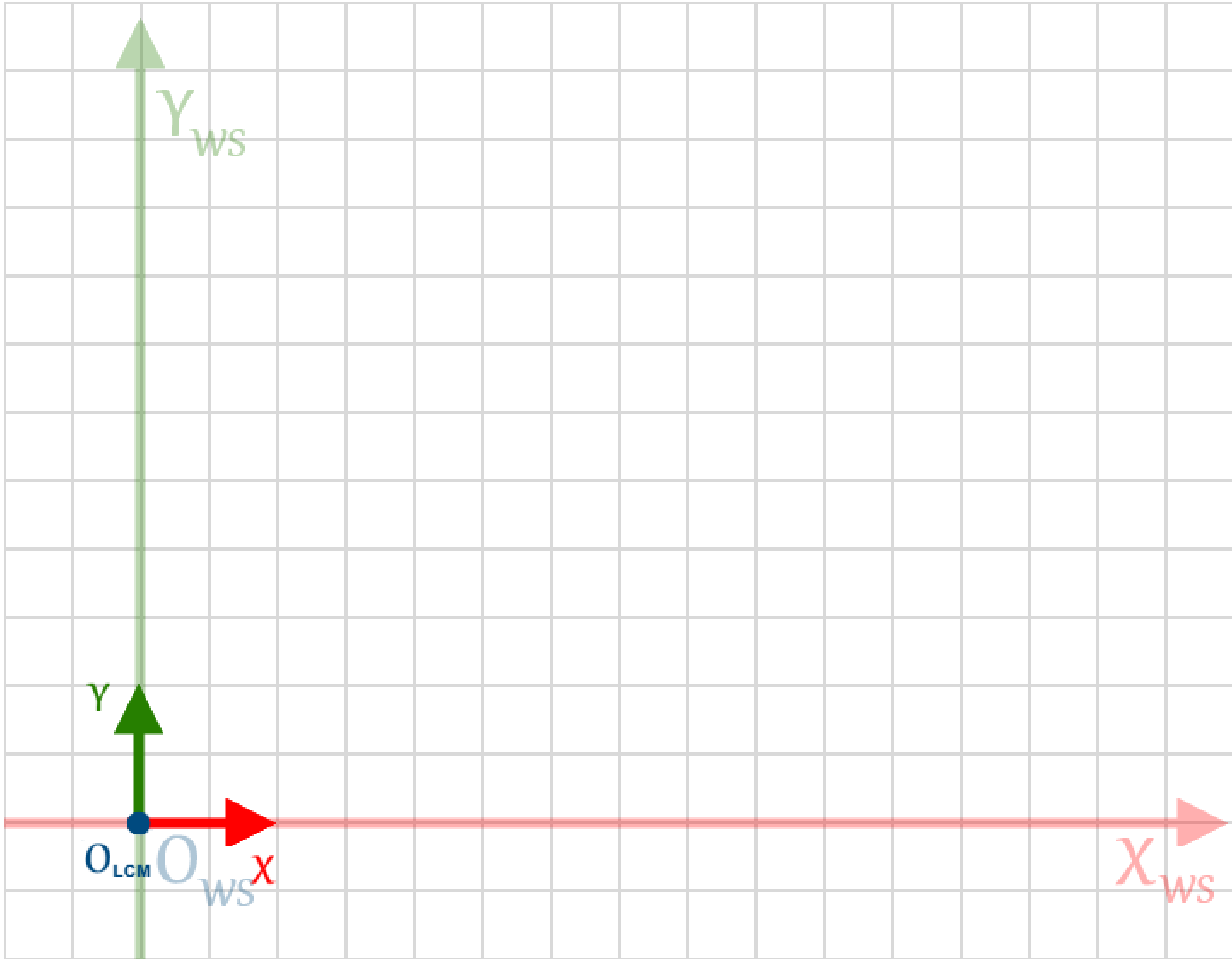


Step by step

- In more detail:

Transformation Operations

```
Initialise()
PushMatrix()
```



Step by step

- In more detail:

Transformation Operations

```
Initialise()
PushMatrix()
  Translate(Handle_pos)
  DrawHandle()
```

Step by step

- In more detail:

Transformation Operations

```
Initialise()
PushMatrix()
  Translate(Handle_pos)
  DrawHandle()
  Translate(Link1_trans)
```

Step by step

- In more detail:

Transformation Operations

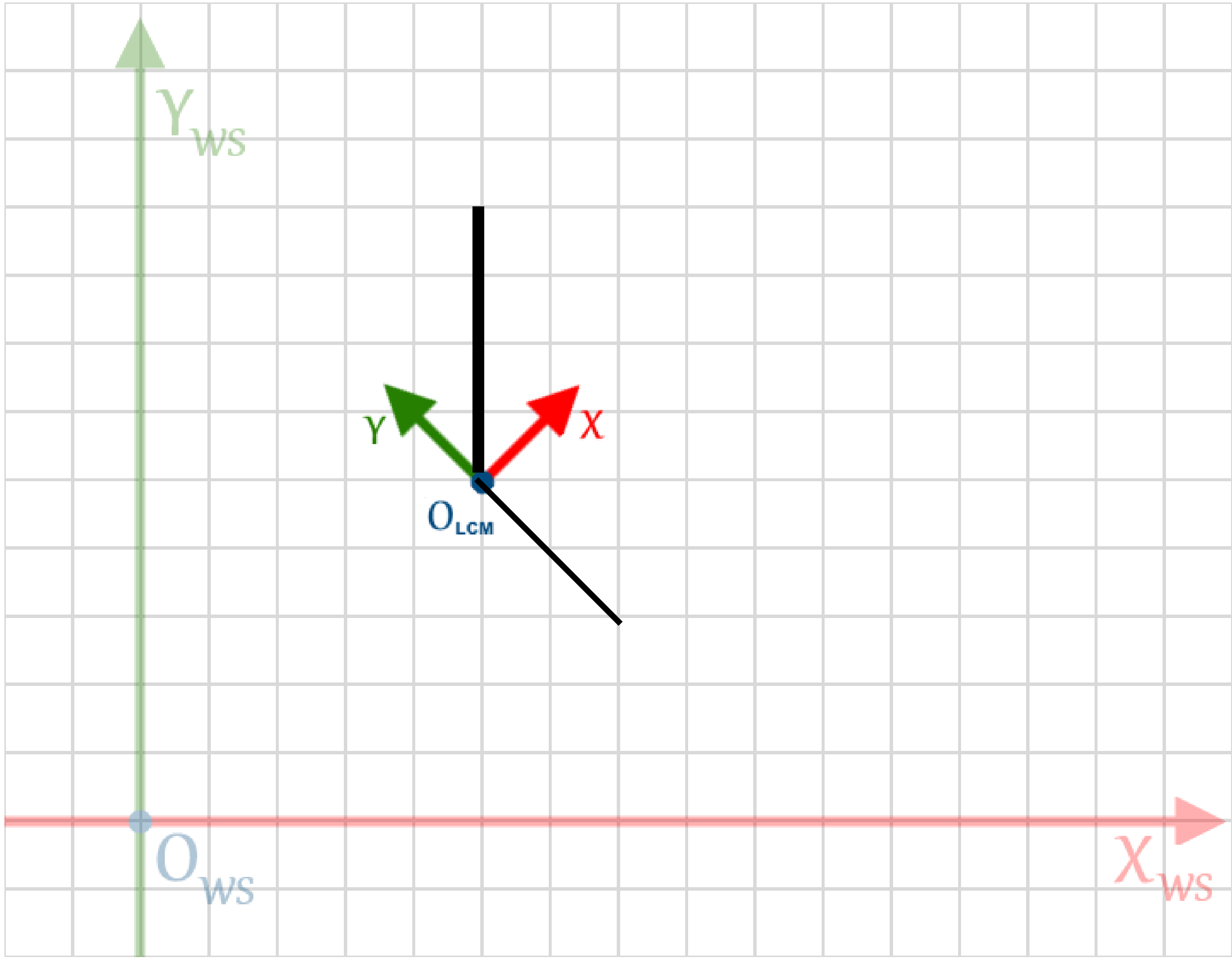
```
Initialise()
PushMatrix()
 Translate(Handle_pos)
 DrawHandle()
 Translate(Link1_trans)
 Rotate(Link1_ang)
```

Step by step

- In more detail:

Transformation Operations

```
Initialise()
PushMatrix()
  Translate(Handle_pos)
  DrawHandle()
  Translate(Link1_trans)
  Rotate(Link1_ang)
  Draw_Link1()
```



Step by step

- In more detail:

Transformation Operations

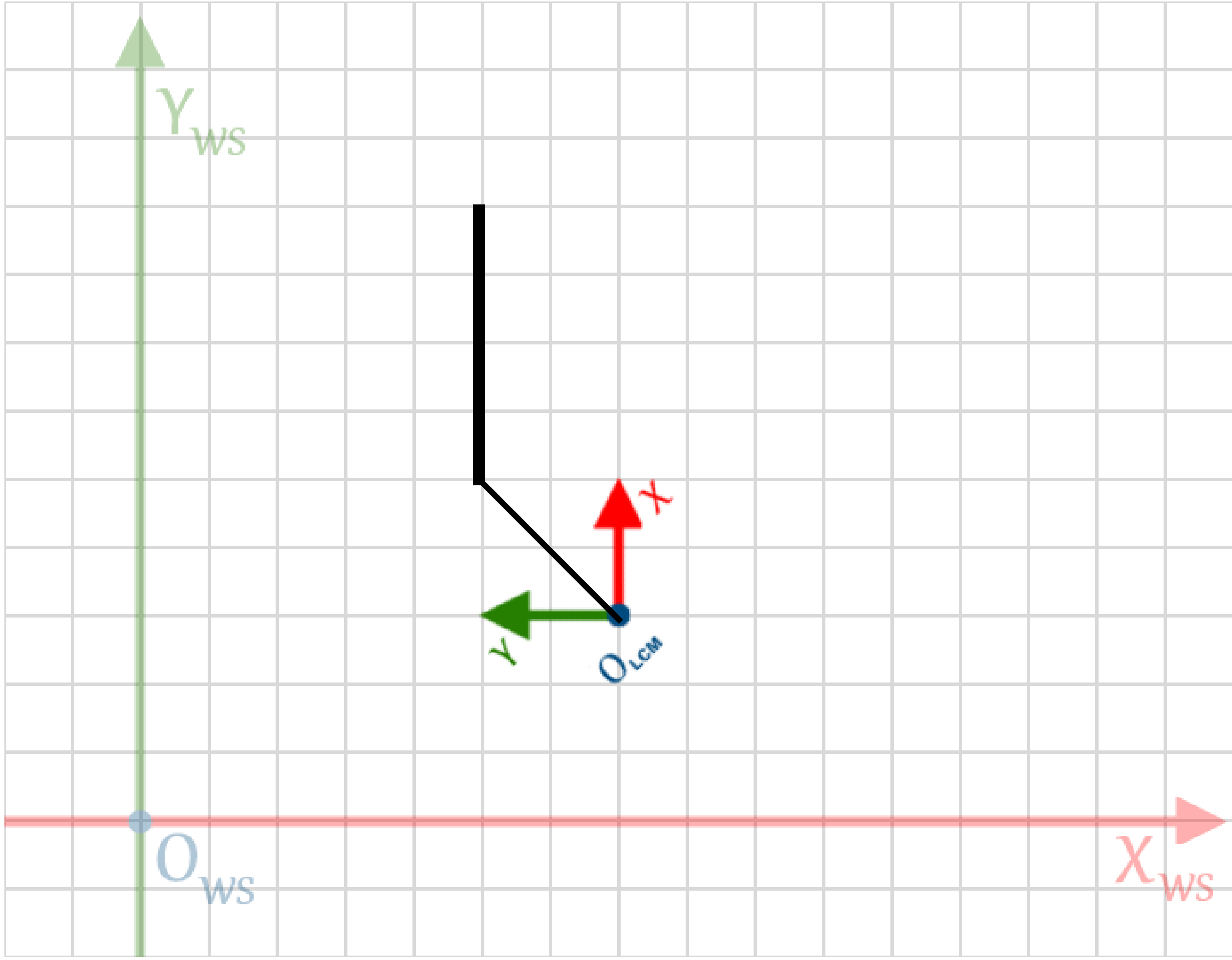
```
Initialise()
PushMatrix()
 Translate(Handle_pos)
 DrawHandle()
 Translate(Link1_trans)
 Rotate(Link1_ang)
 Draw_Link1()
 Translate(Link2_trans)
```

Step by step

- In more detail:

Transformation Operations

```
Initialise()
PushMatrix()
 Translate(Handle_pos)
 DrawHandle()
 Translate(Link1_trans)
 Rotate(Link1_ang)
 Draw_Link1()
 Translate(Link2_trans)
 Rotate(Link2_ang)
```

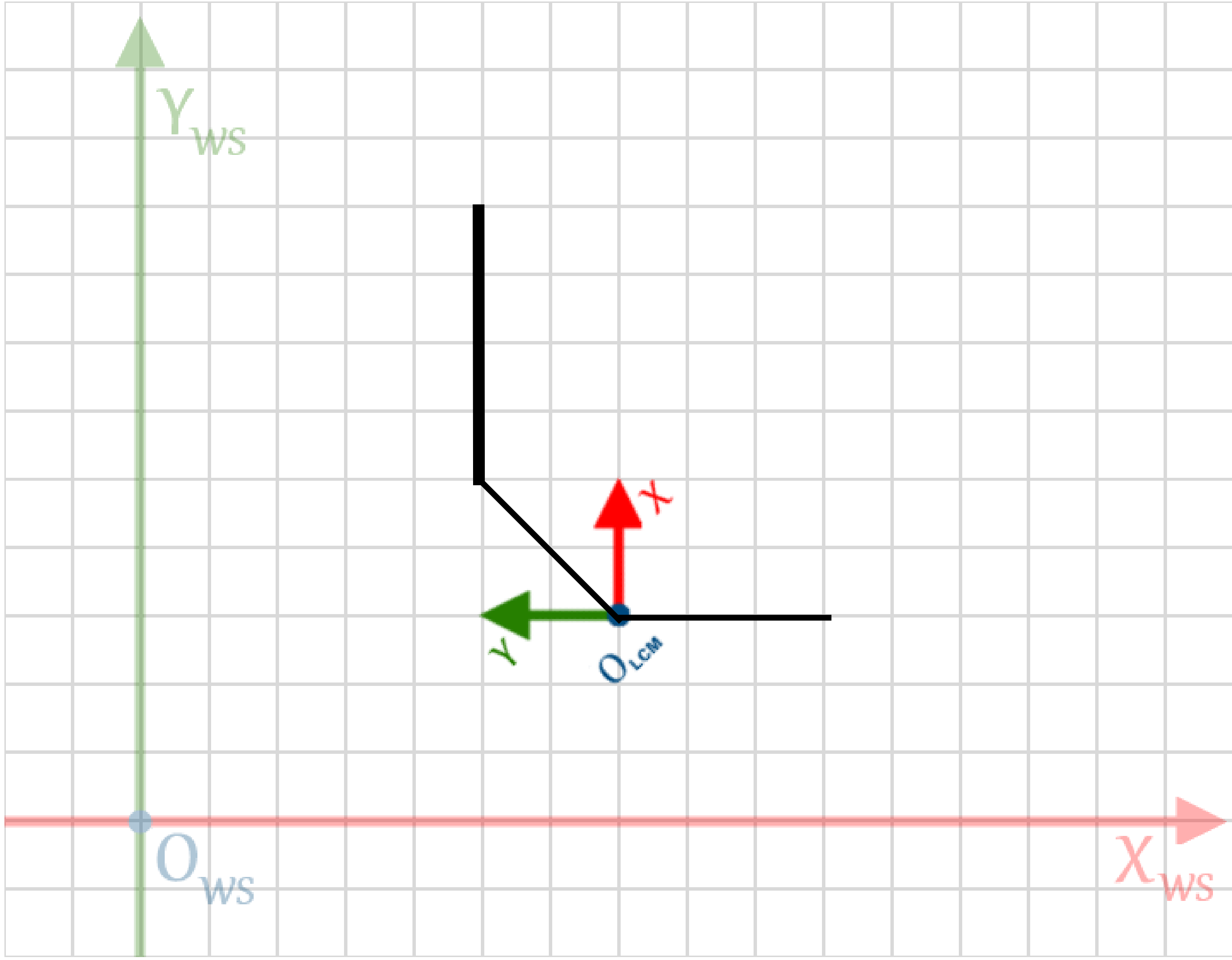


Step by step

- In more detail:

Transformation Operations

```
Initialise()
PushMatrix()
 Translate(Handle_pos)
 DrawHandle()
 Translate(Link1_trans)
 Rotate(Link1_ang)
 Draw_Link1()
 Translate(Link2_trans)
 Rotate(Link2_ang)
 Draw_Link2()
```

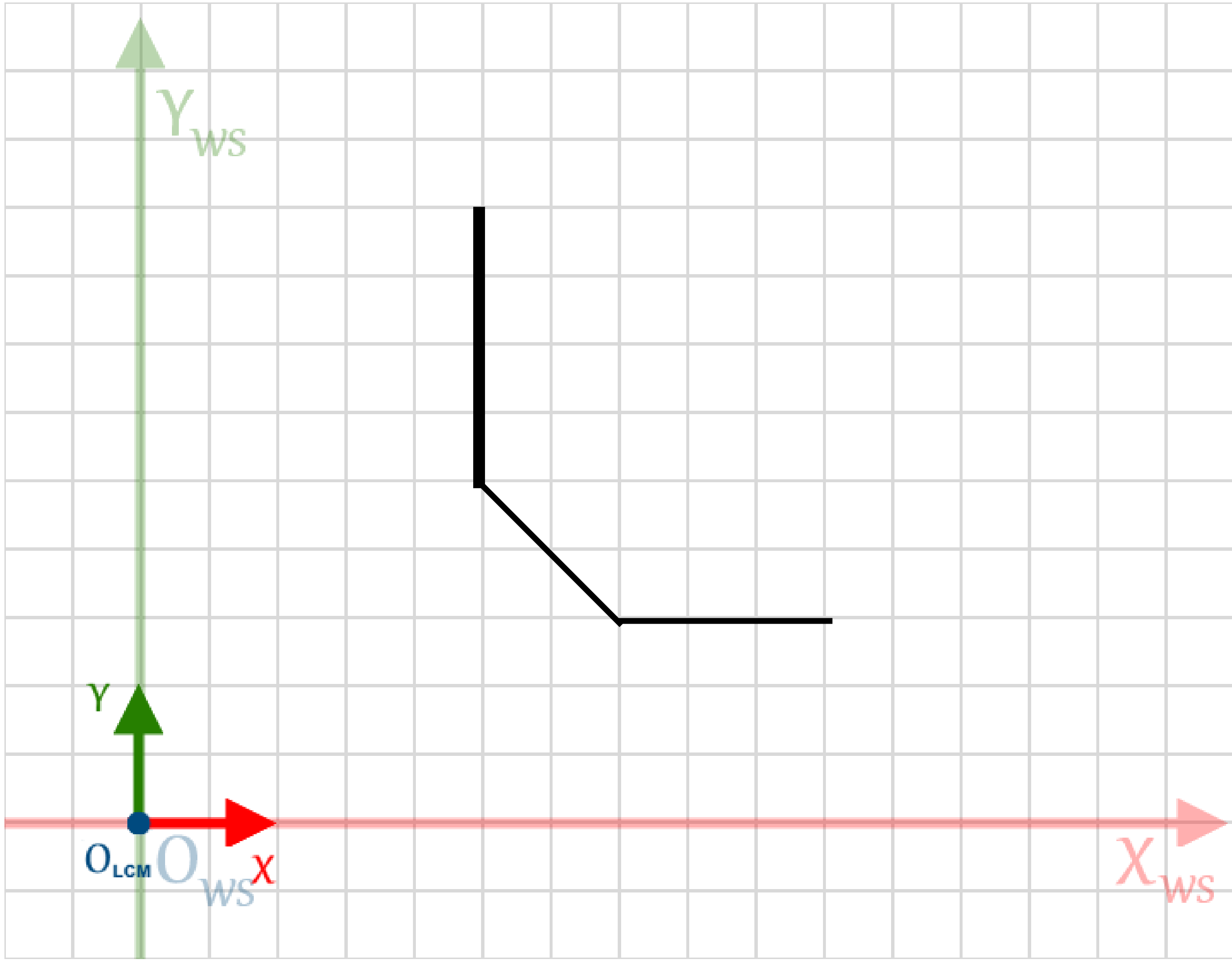


Step by step

- In more detail:

Transformation Operations

```
Initialise()
PushMatrix()
  Translate(Handle_pos)
  DrawHandle()
  Translate(Link1_trans)
  Rotate(Link1_ang)
  Draw_Link1()
  Translate(Link2_trans)
  Rotate(Link2_ang)
  Draw_Link2()
PopMatrix()
```

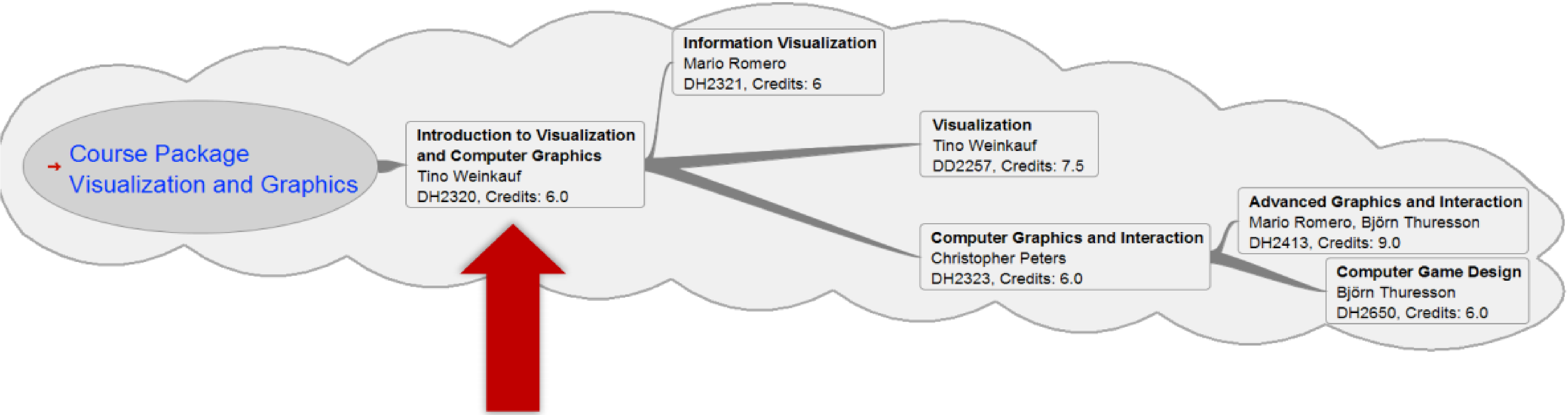


Putting it into Practice

https://processing.org/

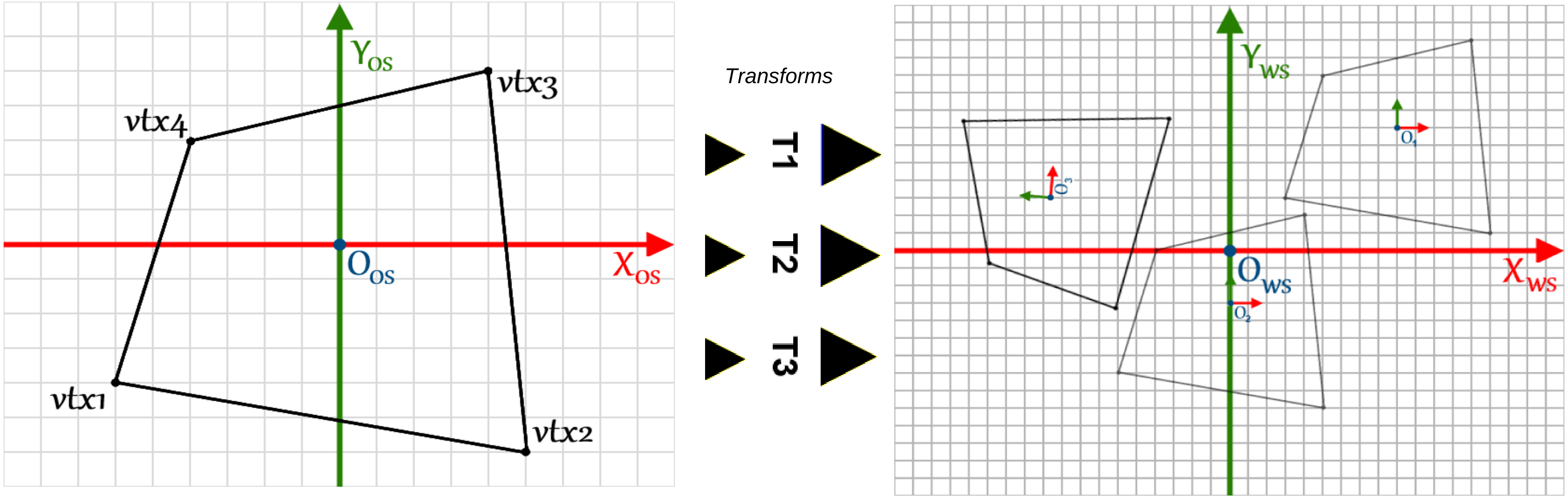
"...a flexible software sketchbook and a language for learning how to code within the context of visual arts"

- Good for a foray into transformations without the complexity of an IDE
- *OpenGL*-based: similar (but less sophisticated) functionality to the framework that you will use in the course
- Straight forward mapping from operations we covered in this lecture to graphics programming functions

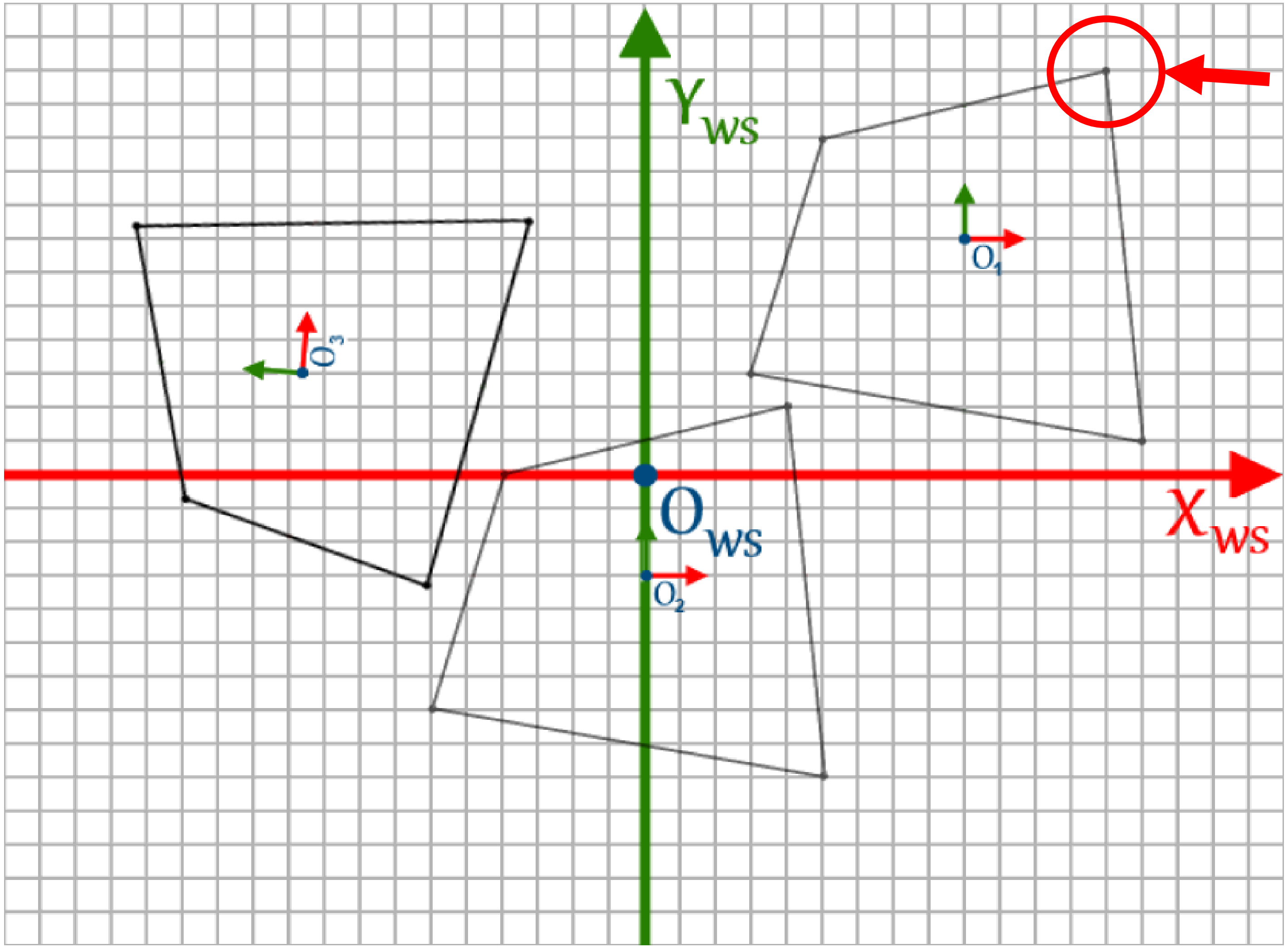

Course Package
Visualization and Graphics
Introduction to Visualization and Computer Graphics
Tino Weinkauf
DH2320, Credits: 6.0
Information Visualization
Mario Romero
DH2321, Credits: 6
Visualization
Tino Weinkauf
DD2257, Credits: 7.5
Computer Graphics and Interaction
Christopher Peters
DH2323, Credits: 6.0
Advanced Graphics and Interaction
Mario Romero, Björn Thuresson
DH2413, Credits: 9.0
Computer Game Design
Björn Thuresson
DH2650, Credits: 6.0

Something for the break

Object specified in Object space (OS) *Positioned in world space (WS) via transform*

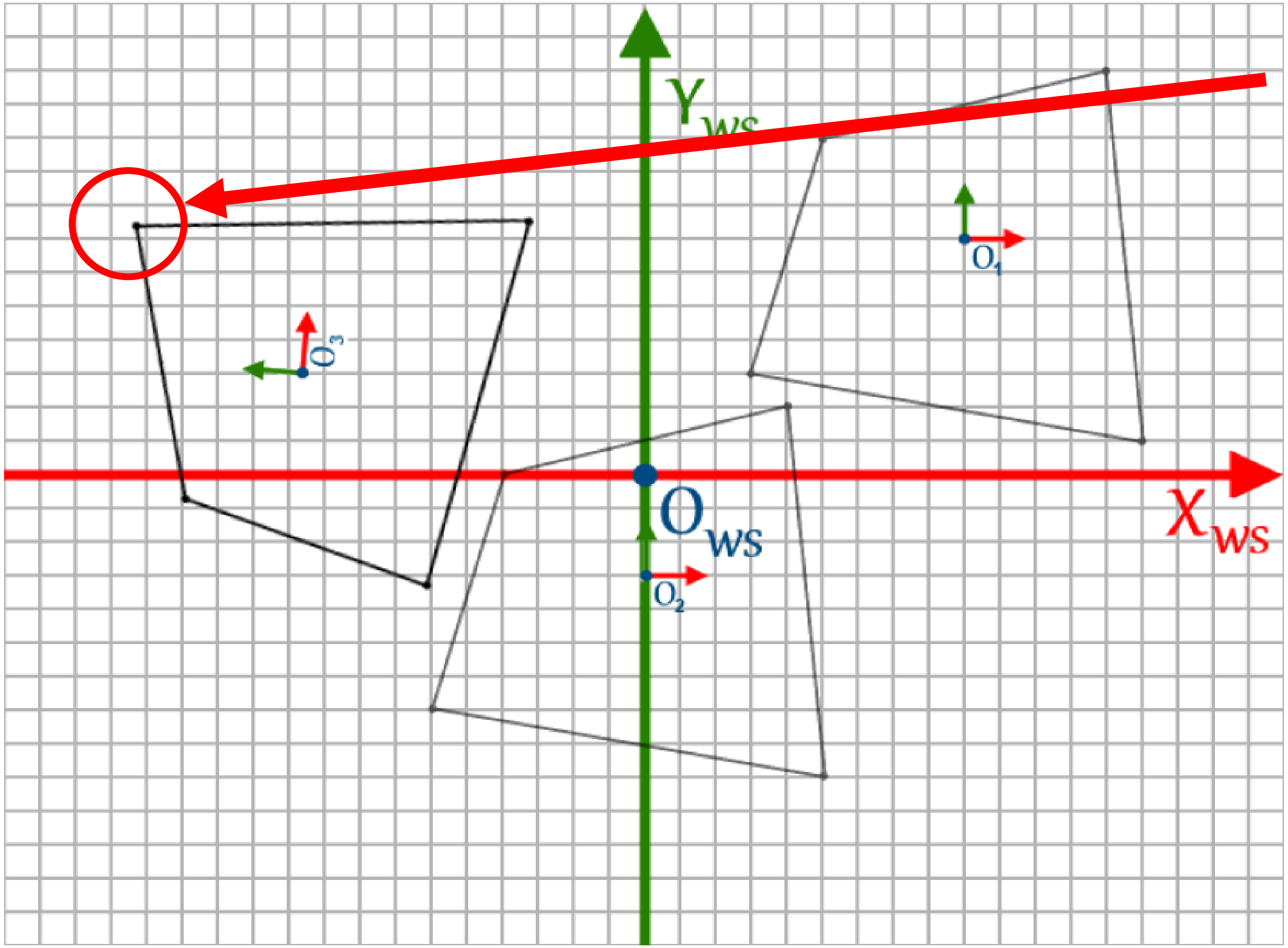


Q: What is the transformation ***T1***?



Vertex position:
OS (4.0,5.0)
WS (13.0,12.0)

Q: What is the transformation ***T3***?



Vertex position:
OS (4.0,5.0)
WS (-14.3,7.4)