

HIERARCHICAL TRANSFORMATIONS

A Practical Introduction

Christopher Peters

HPCViz, KTH Royal Institute of Technology, Sweden

chpeters@kth.se

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

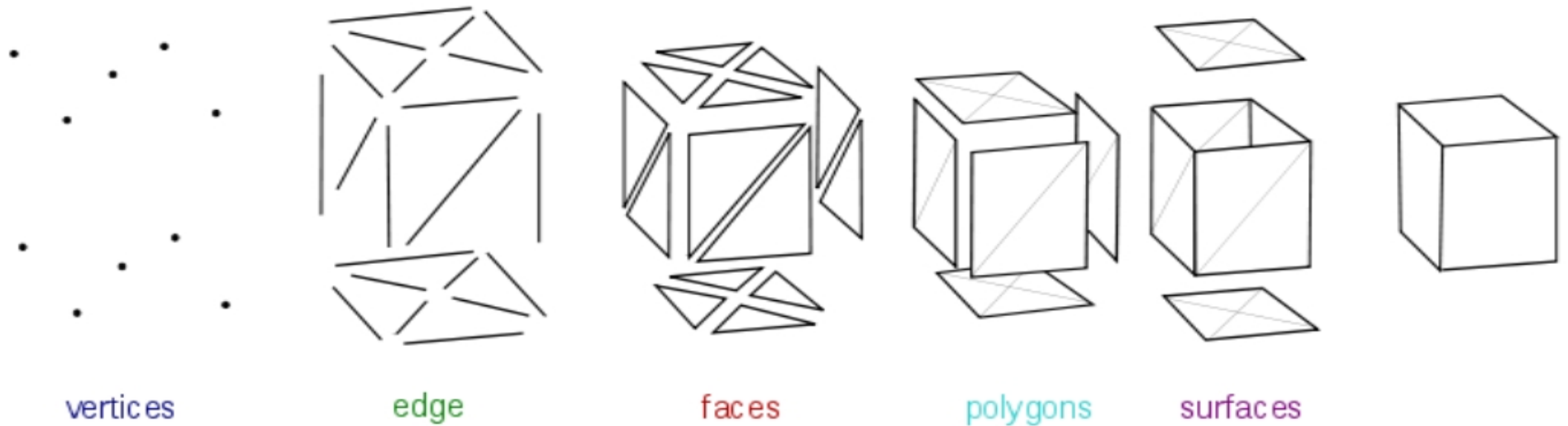
Transformations

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



Geometric primitives

(a brief introduction)



Graphical objects are composed of primitives

- More about geometry in subsequent lectures

Transformations

Recall *translation* from previous lecture:

- Translate a point **p** along a vector **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:

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

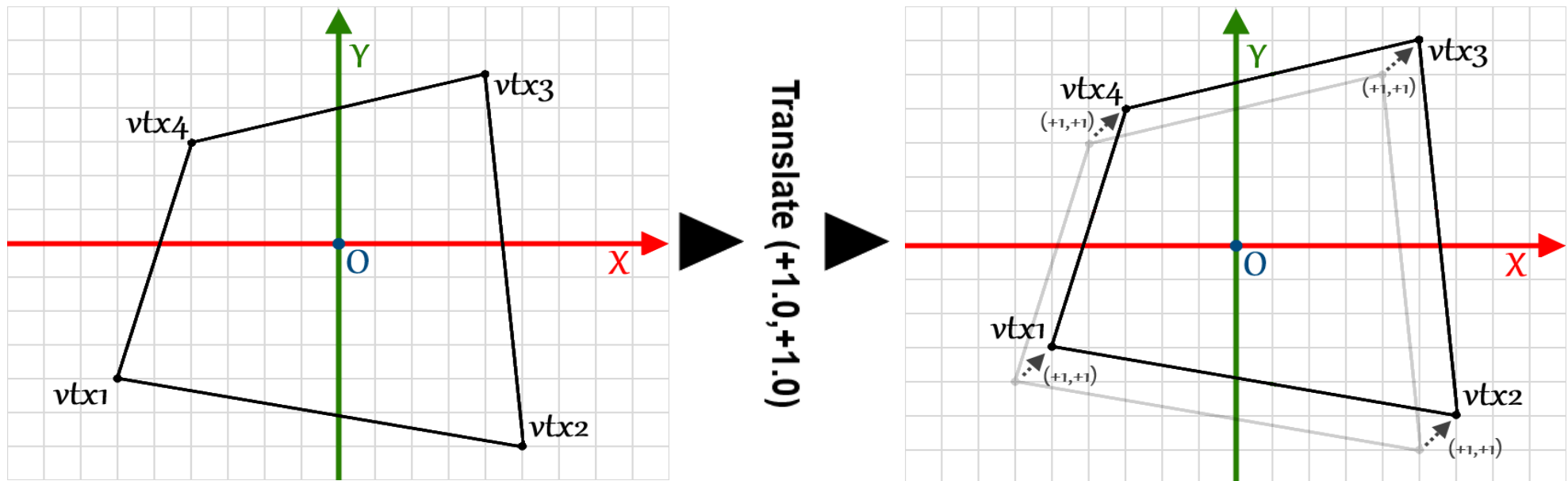
Translating an object

Translation operation takes place on a point

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

How to translate that?

Translate each of its vertices



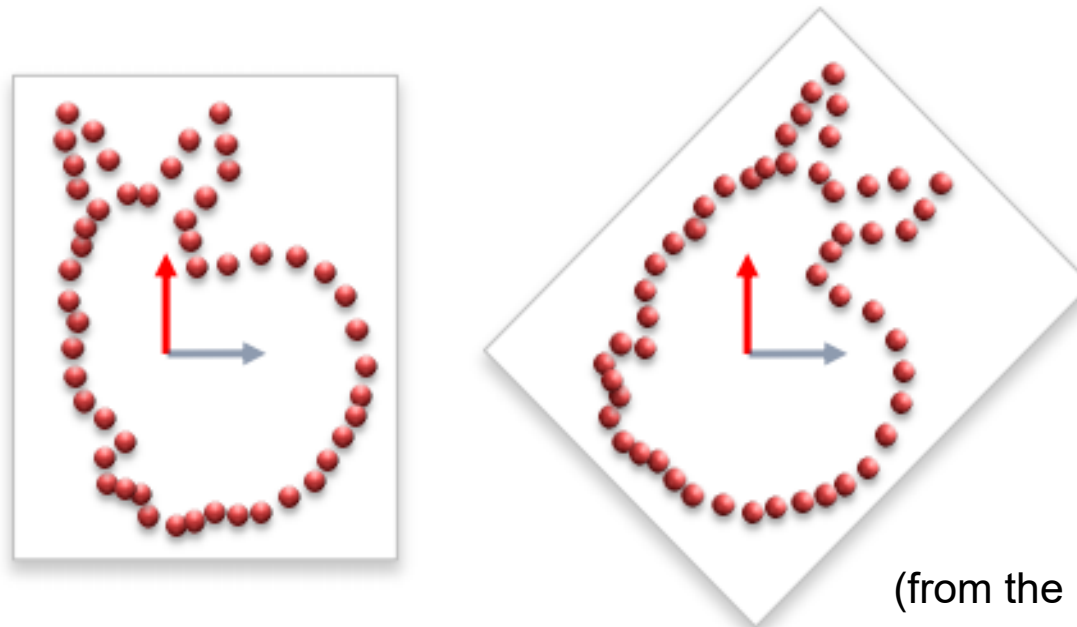
Rotating an object

Rotation operation takes place on a point

How to rotate a object?

The same procedure applies:

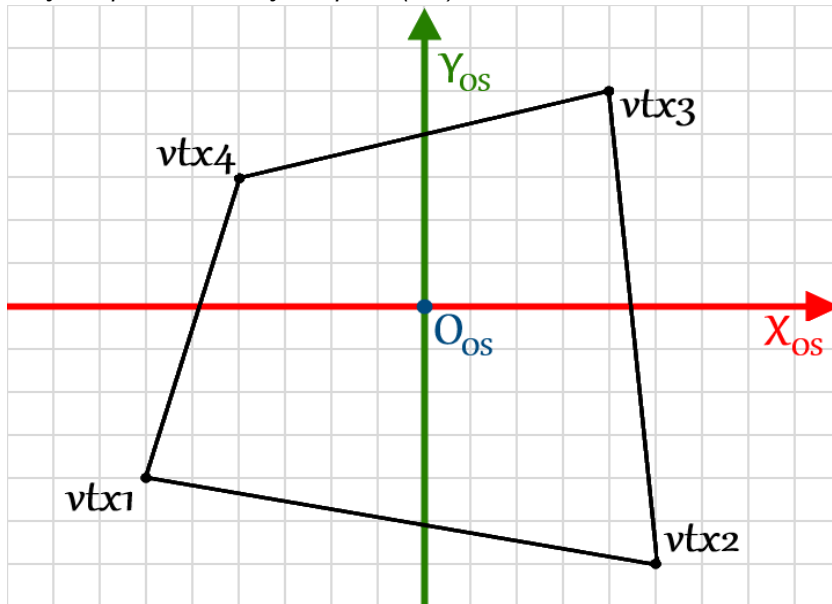
Rotate each vertex that comprises the object



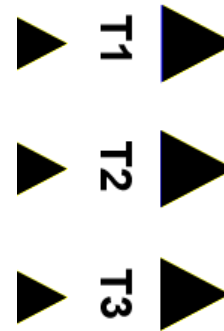
(from the previous lecture)

World space

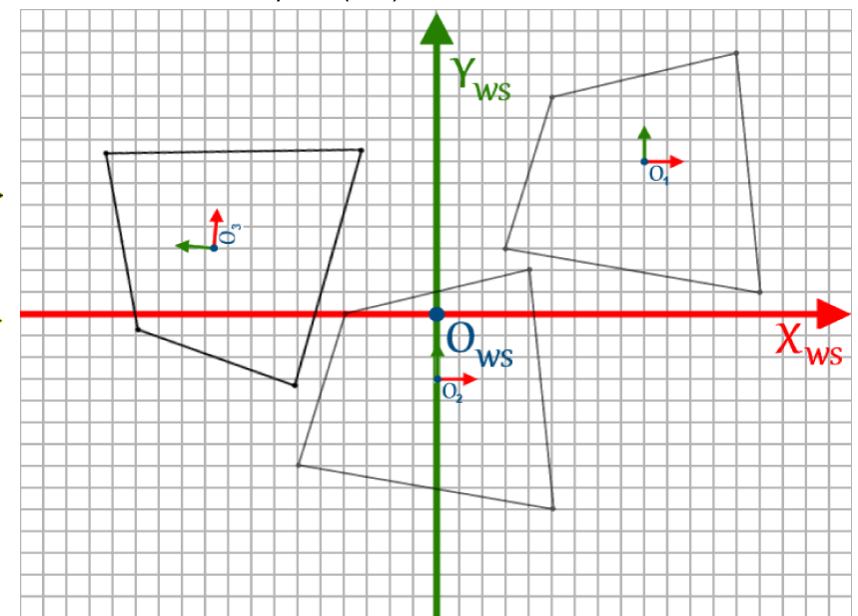
Object specified in Object space (OS)



Transforms



Positioned in world space (WS) via transform



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

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

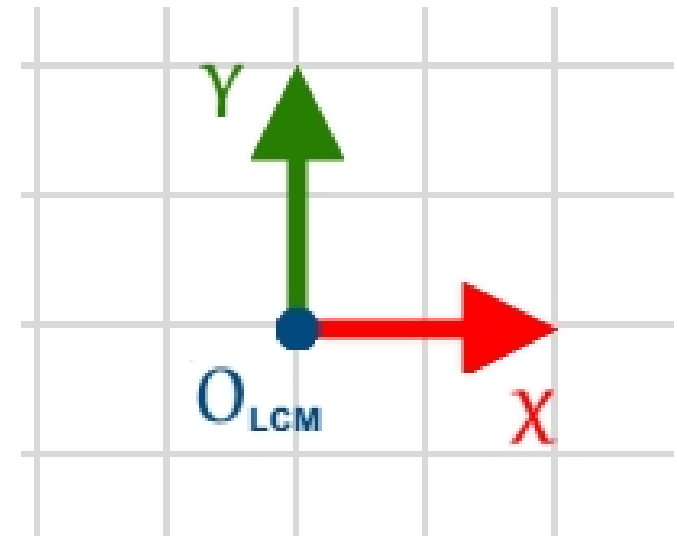
$$\mathbf{M} \cdot \mathbf{x} = \begin{pmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix}$$

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?

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



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

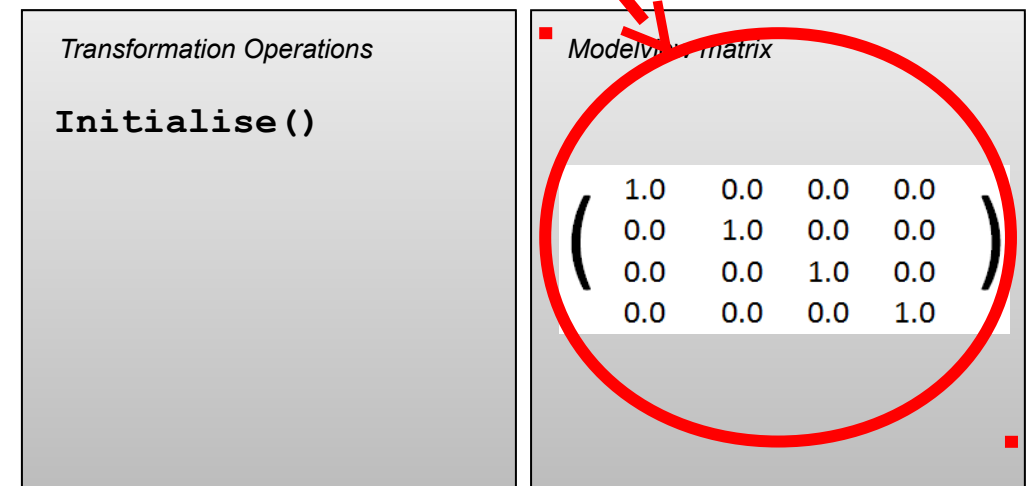
$$\begin{pmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{pmatrix}$$

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

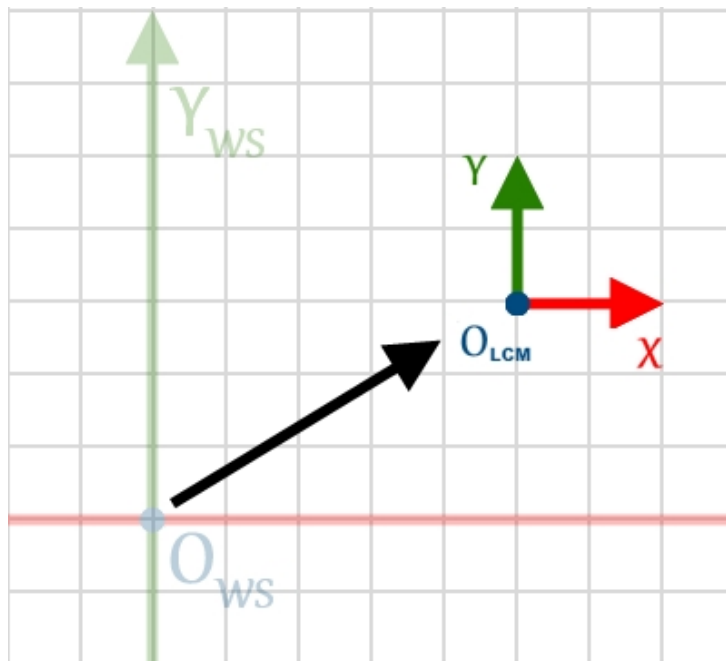
Identity matrix



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)`

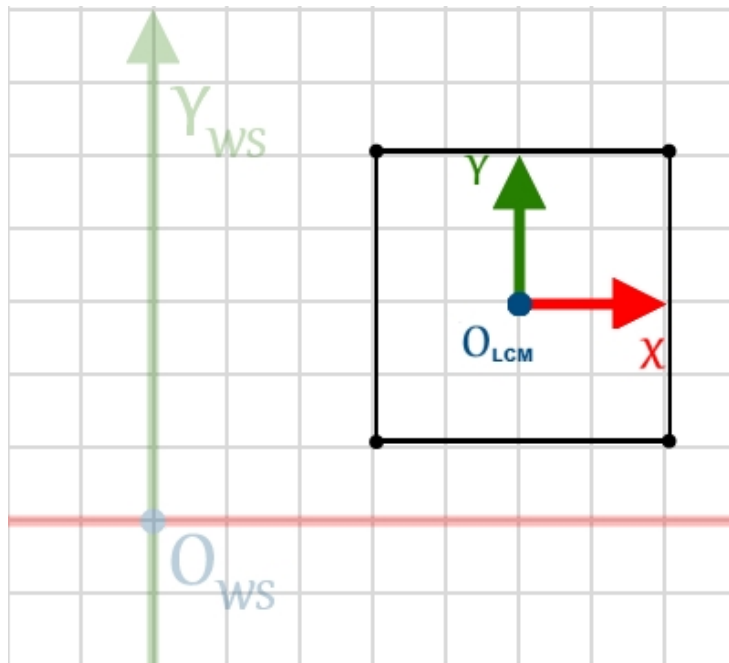
Modelview matrix

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

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()
```

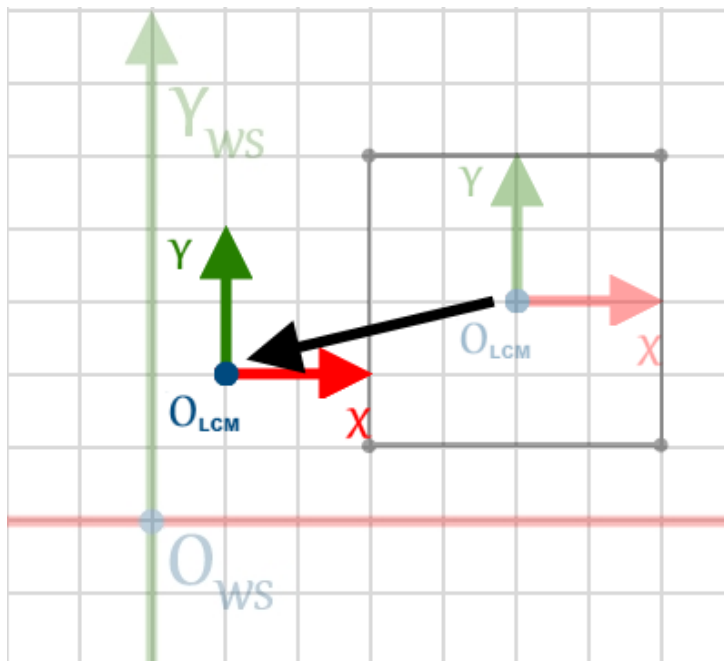
Modelview matrix

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

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)
```

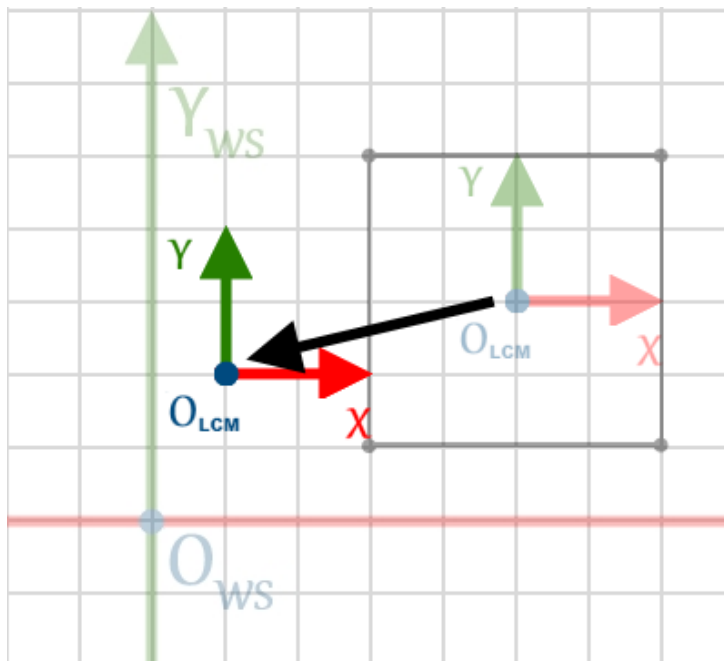
Modelview matrix

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

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)
```

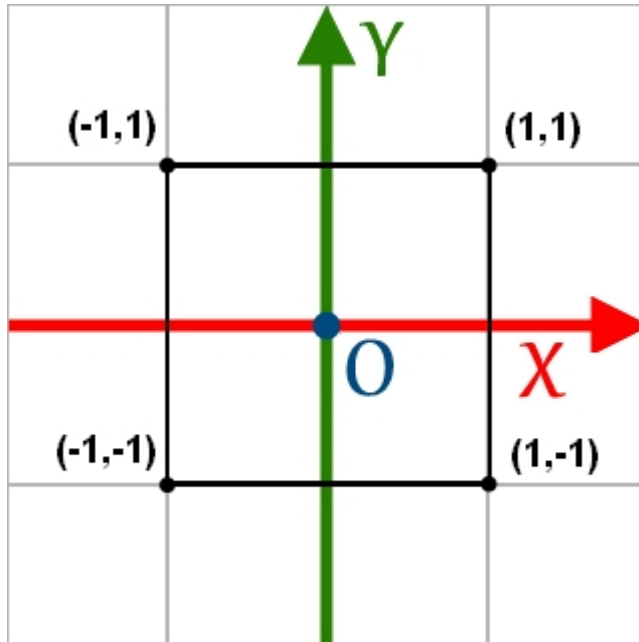
Modelview matrix

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

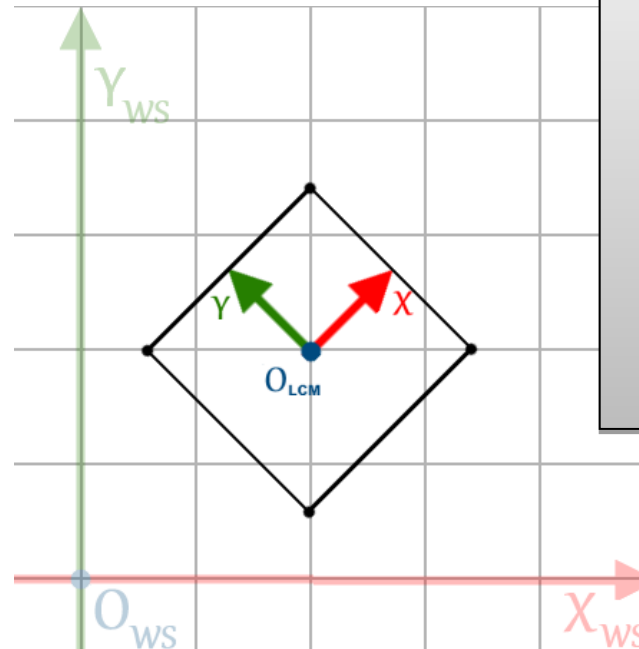
Result

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()
```

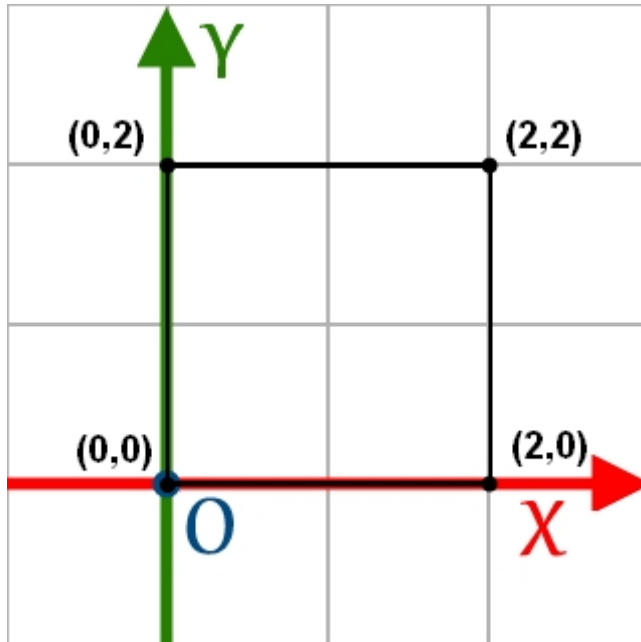
Rotations also occur about the origin of the object

- 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()
```

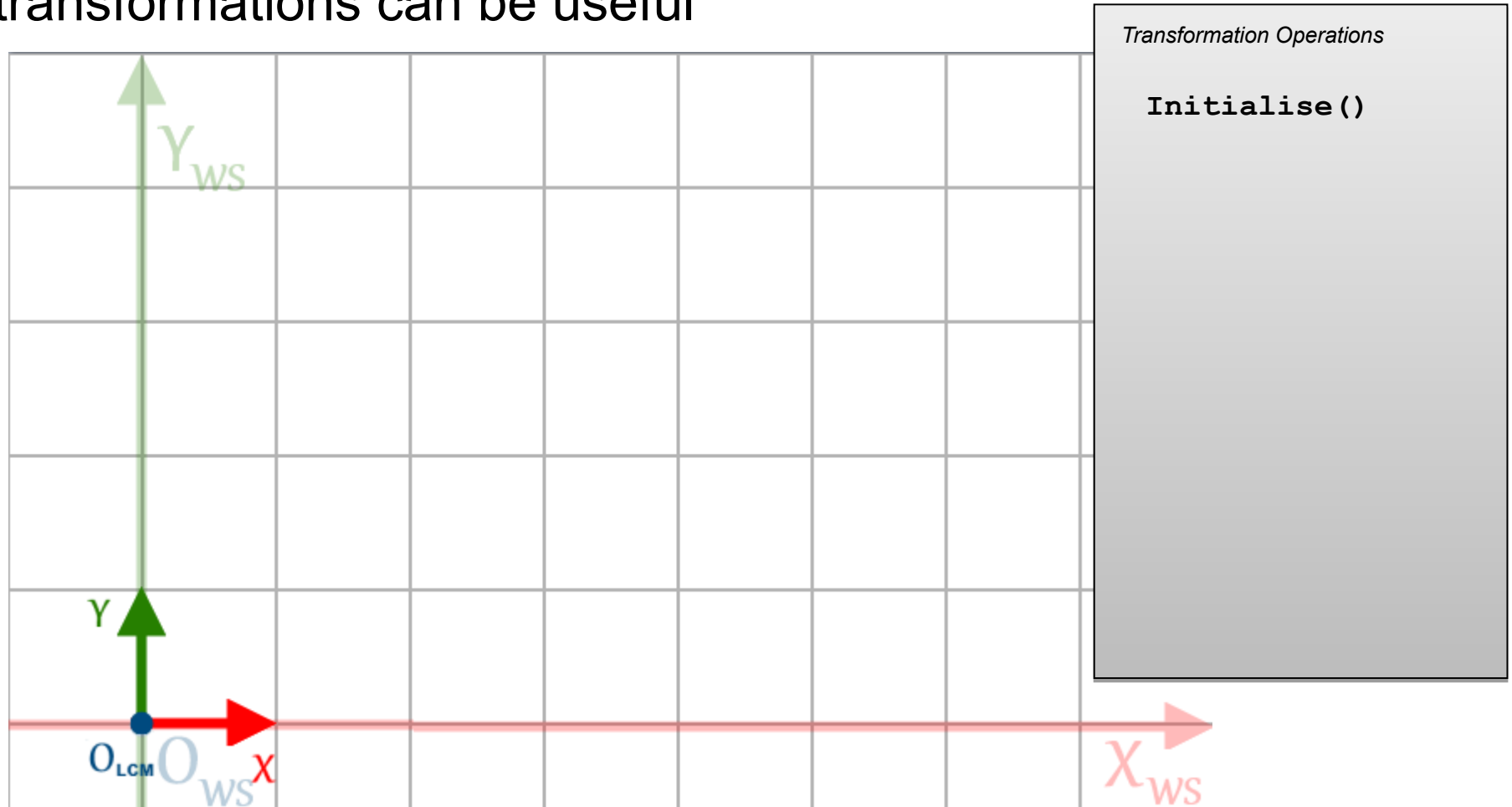
Rotations also occur about the origin of the object

- Default *axis of rotation*

Notice that the transformation is the exact same

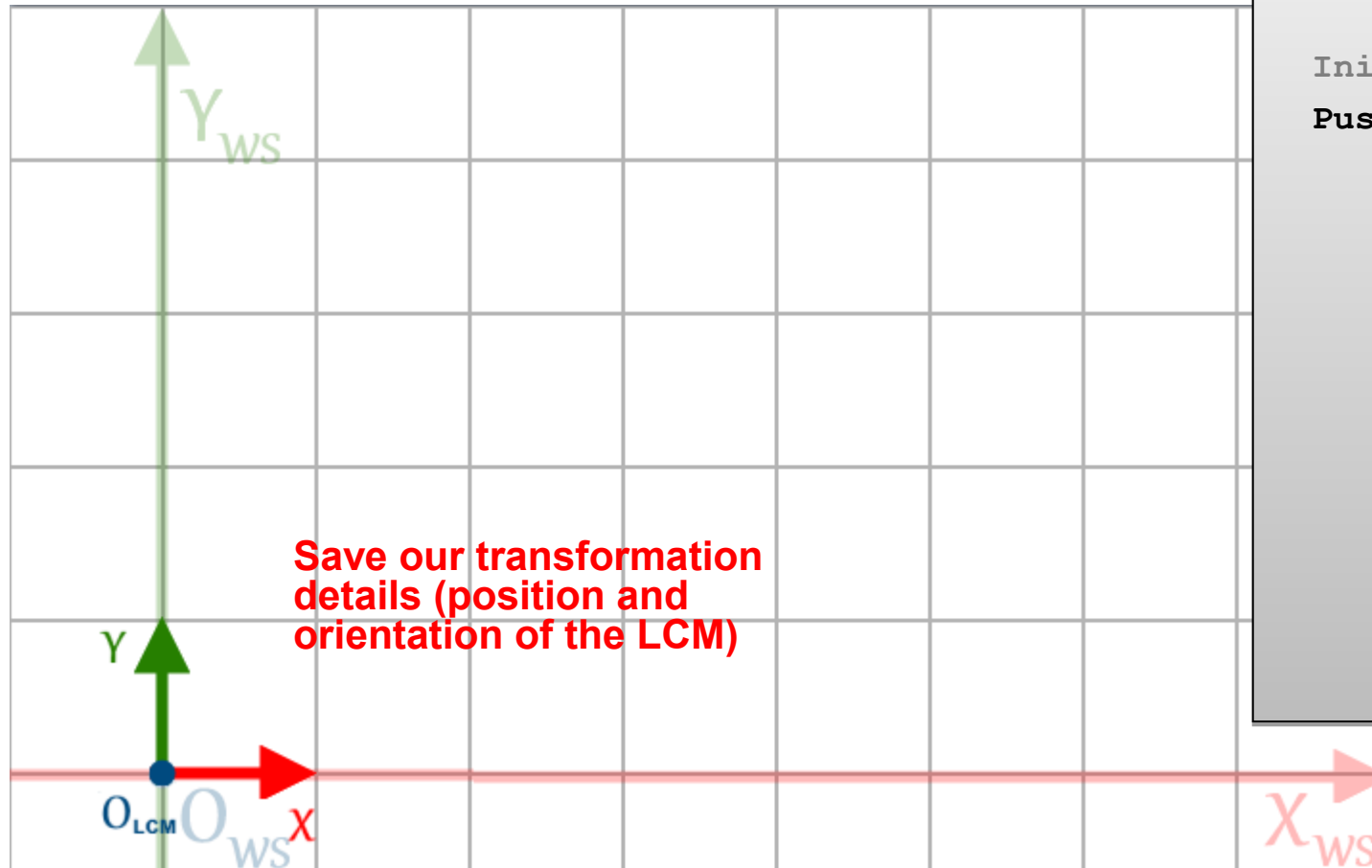
Saving and loading transformations

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



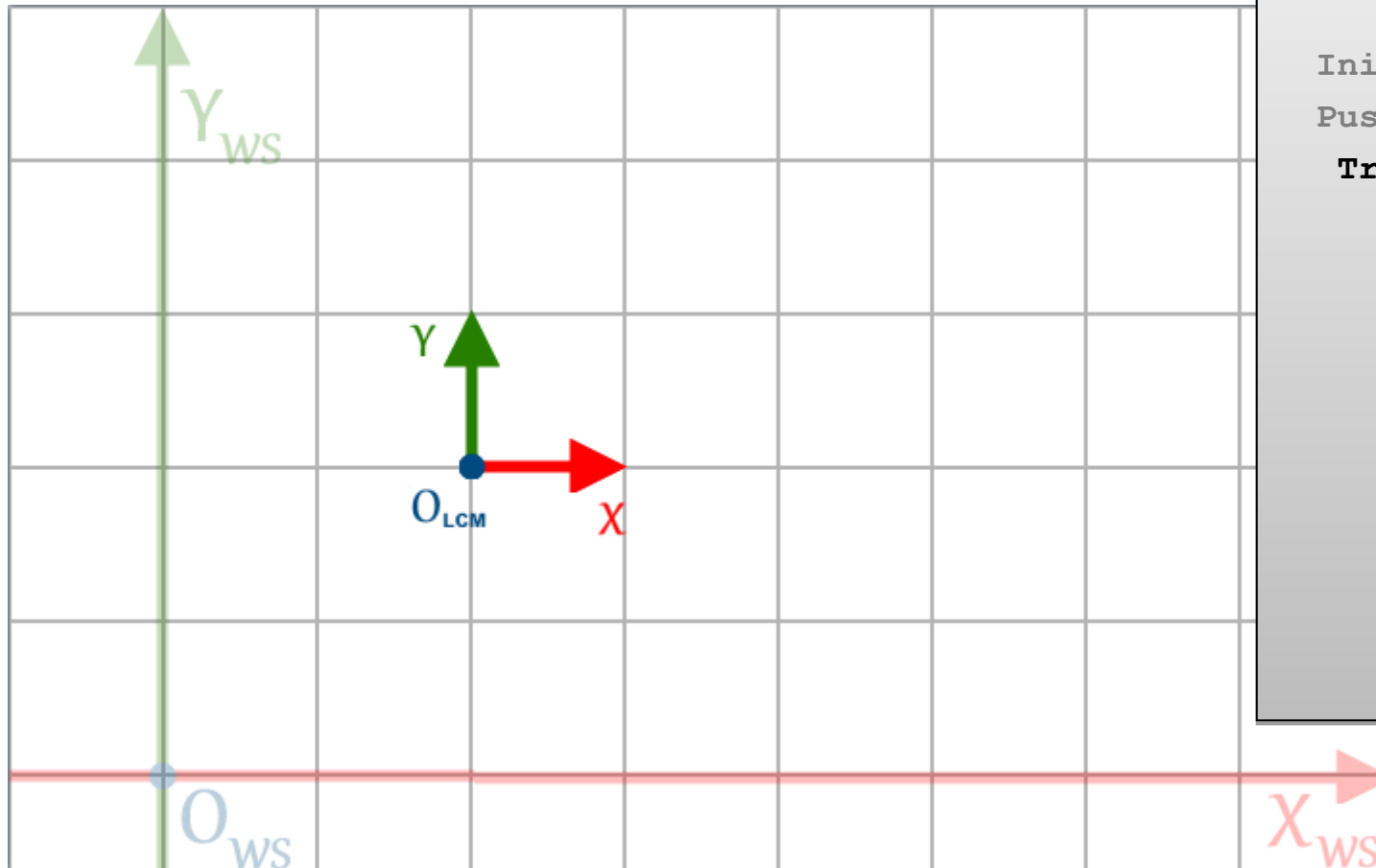
Saving and loading transformations

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



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

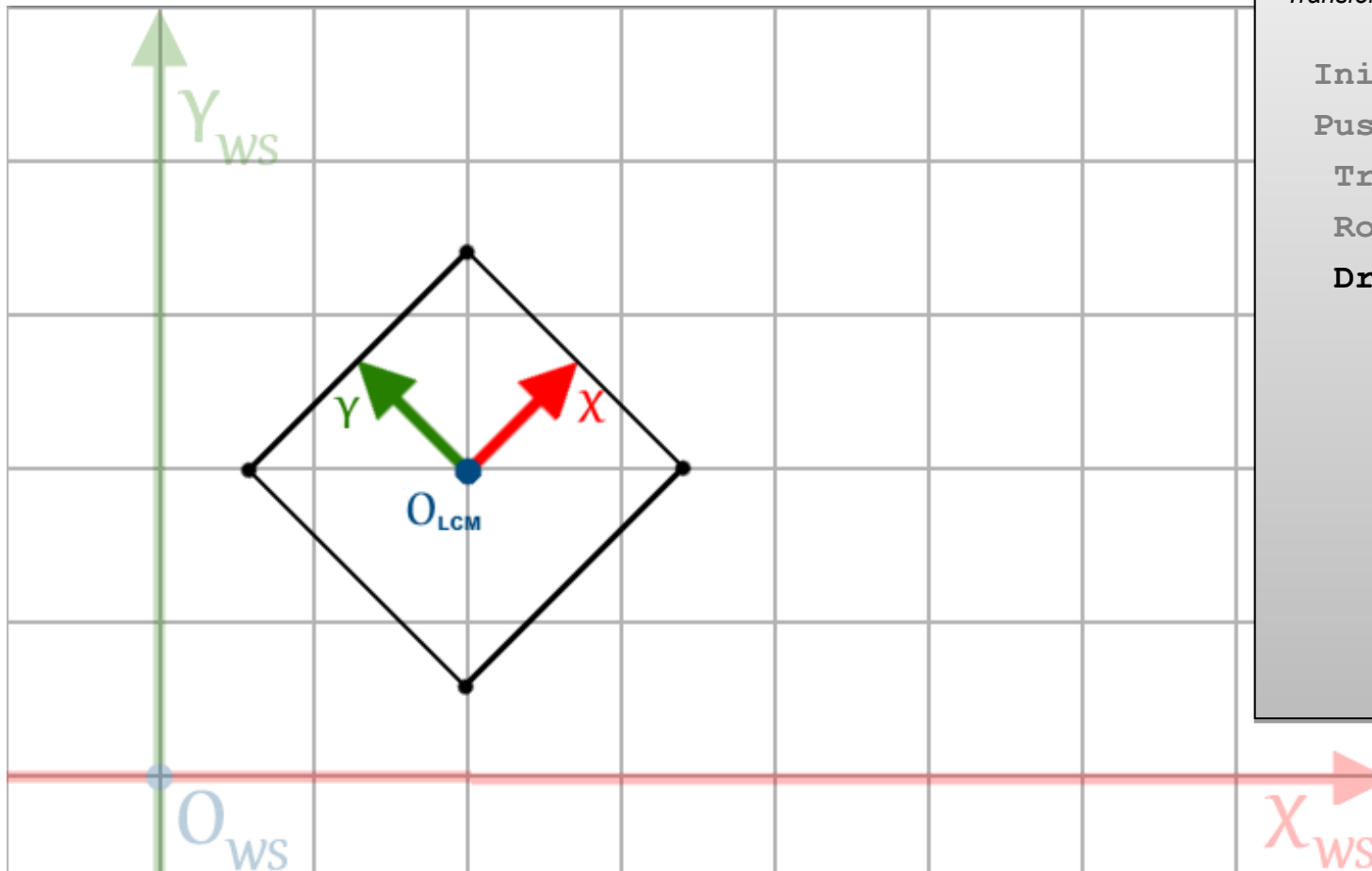


Transformation Operations

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

Saving and loading transformations

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

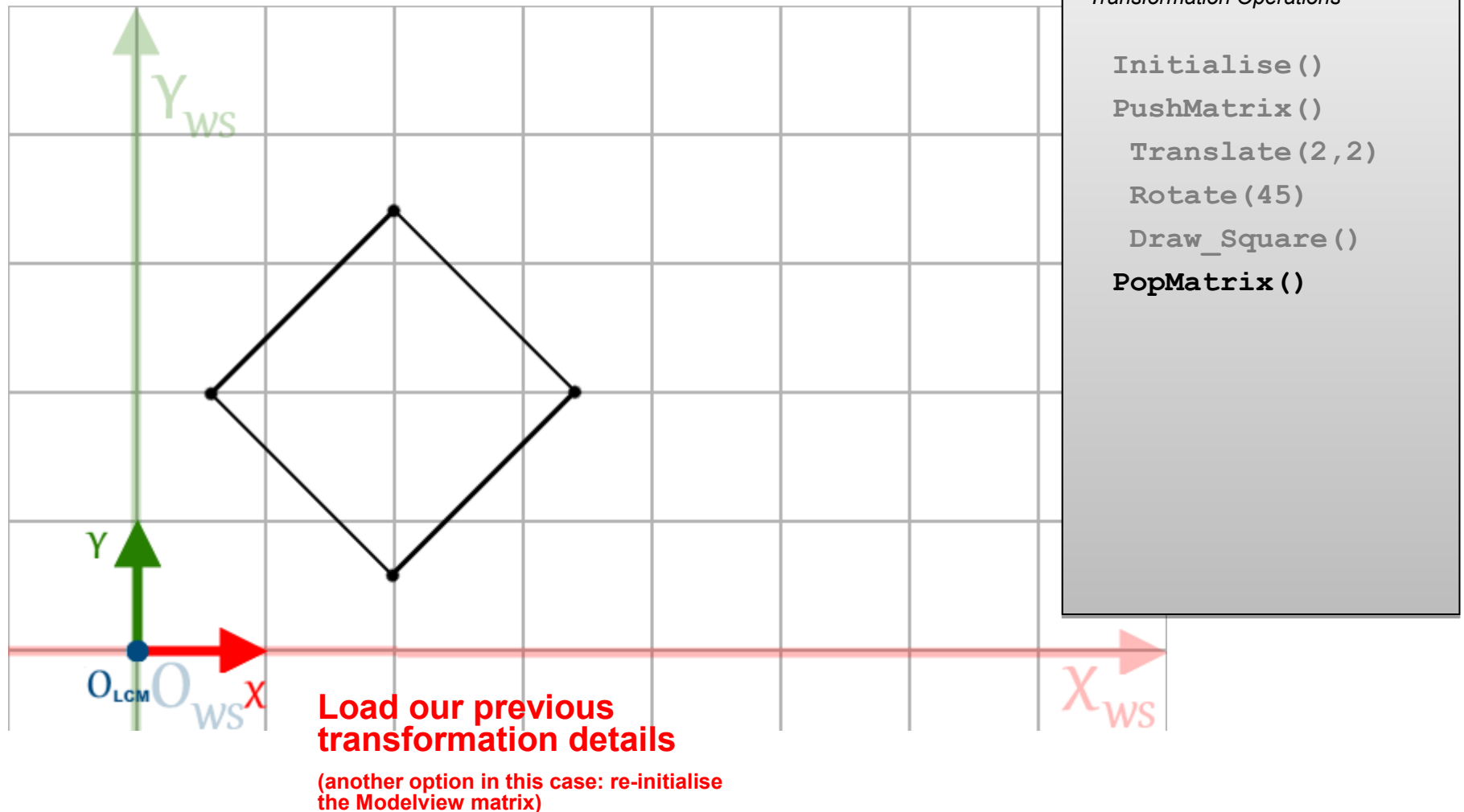


Transformation Operations

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

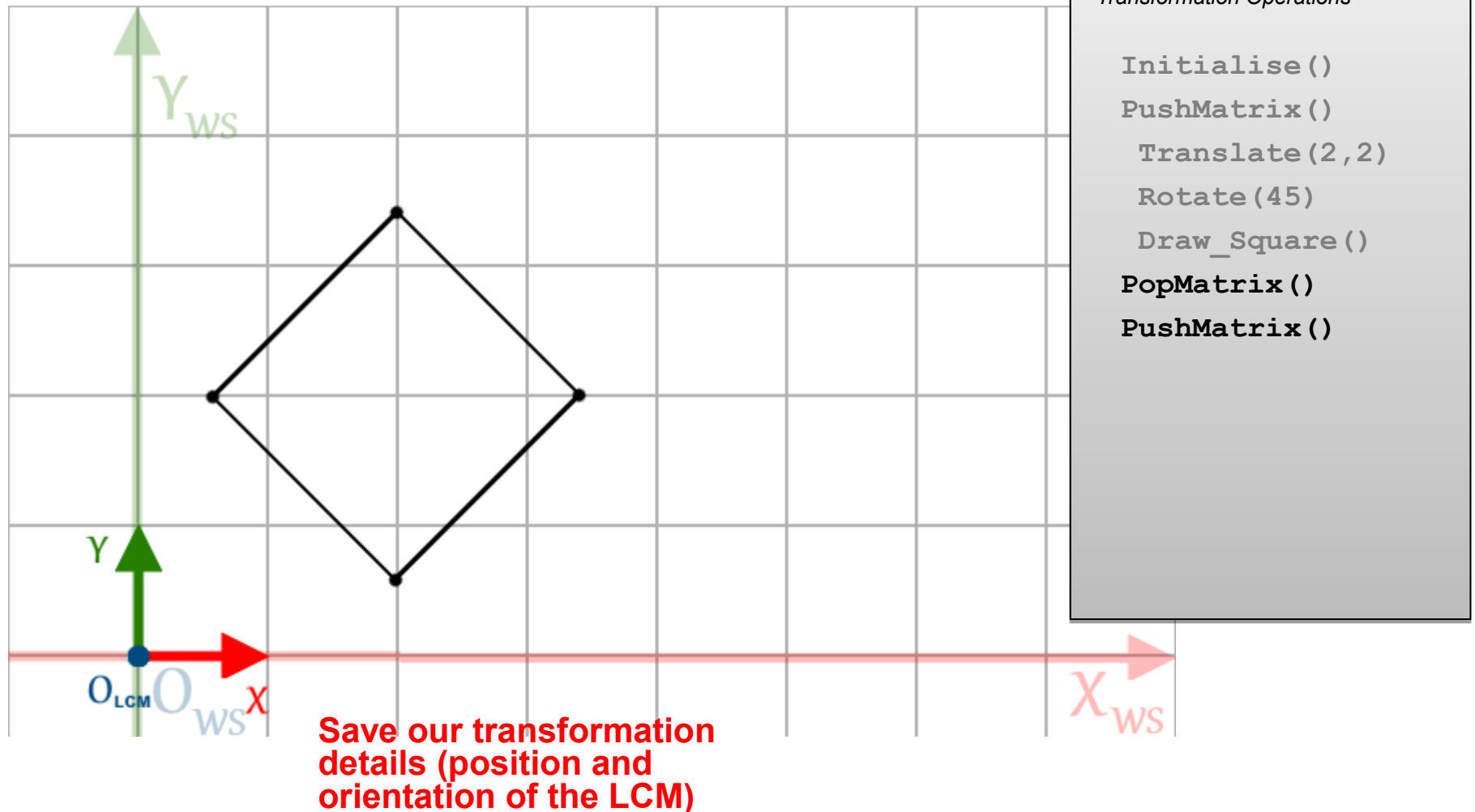
Saving and loading transformations

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



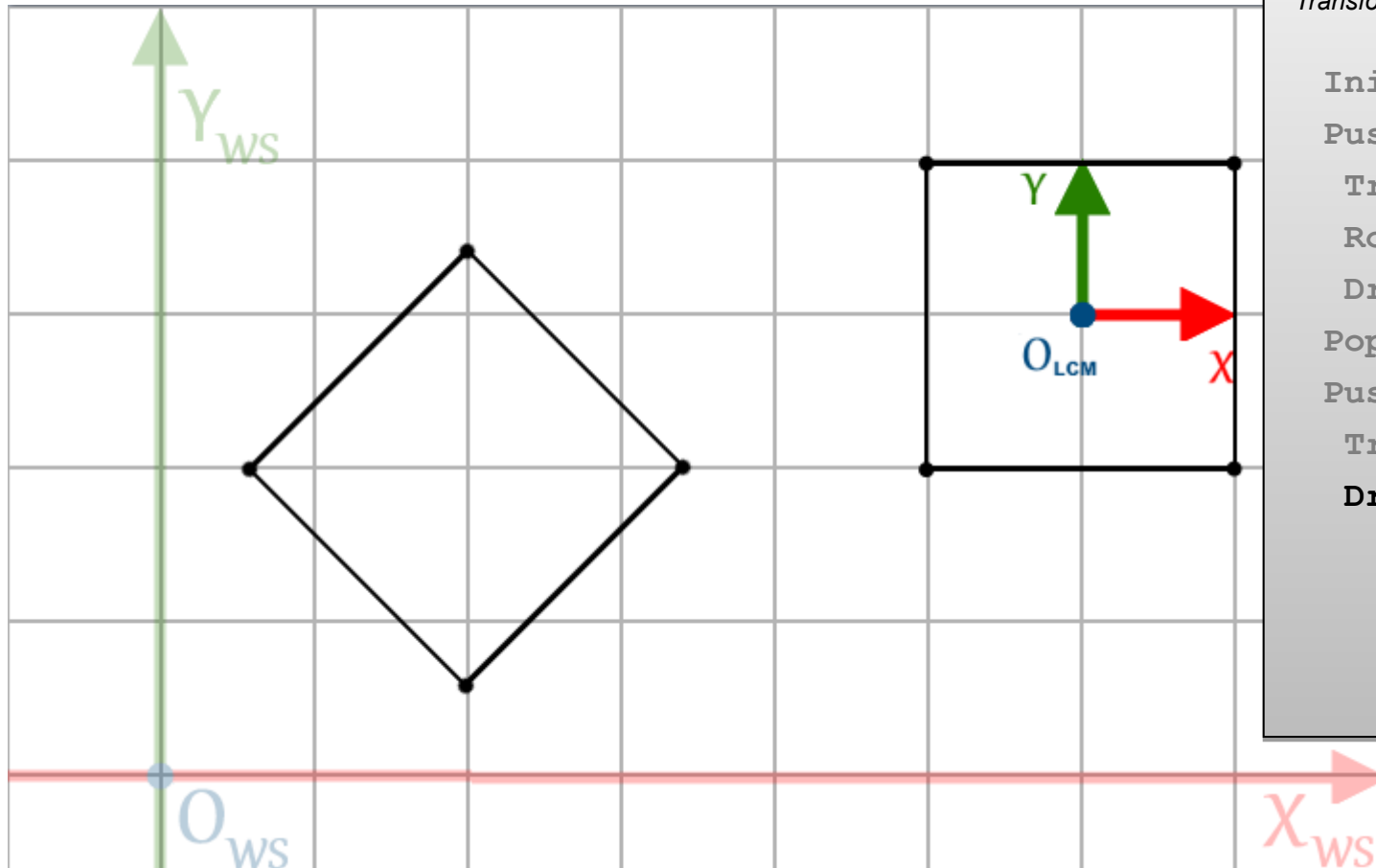
Saving and loading transformations

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



Saving and loading transformations

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



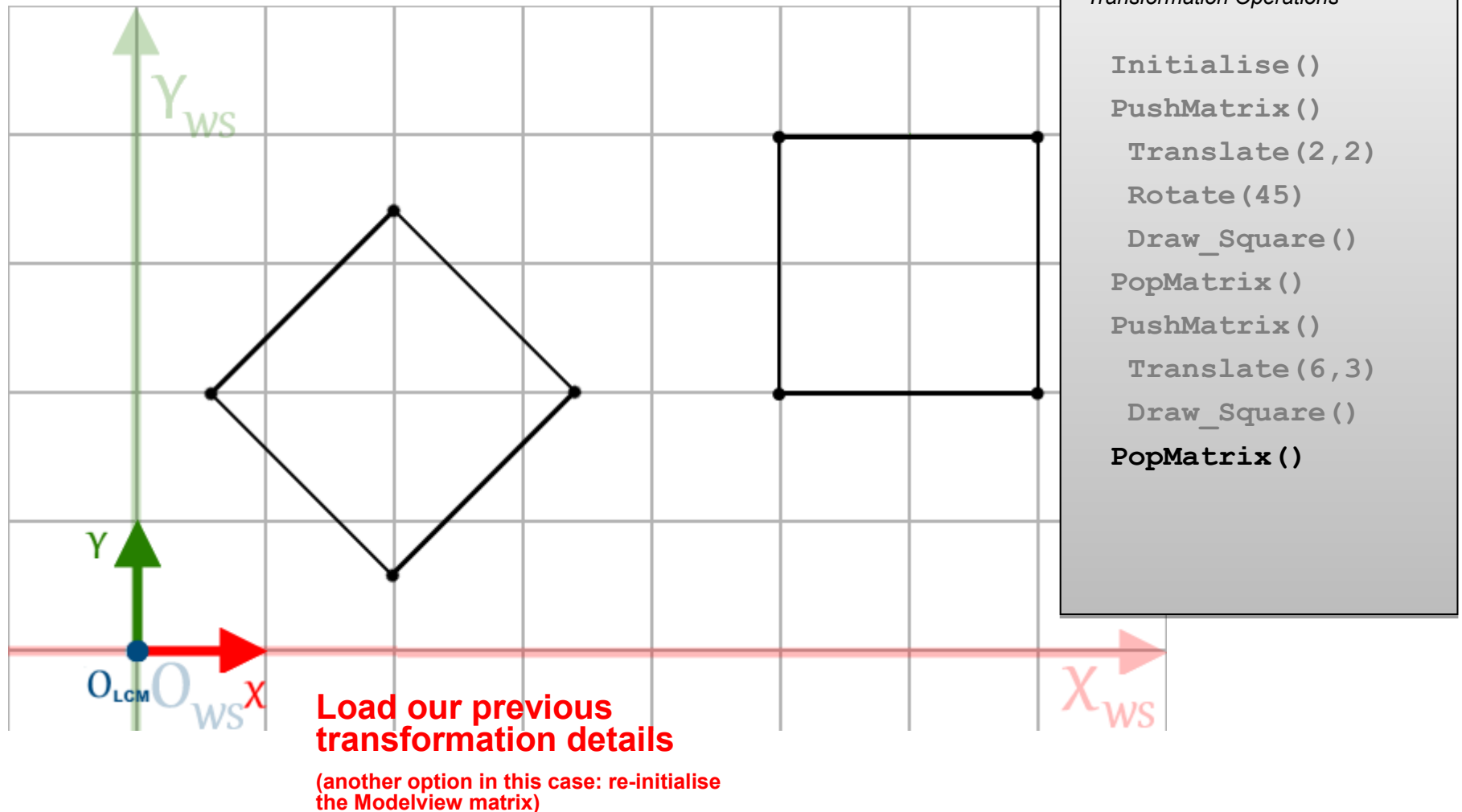
Transformation Operations

```

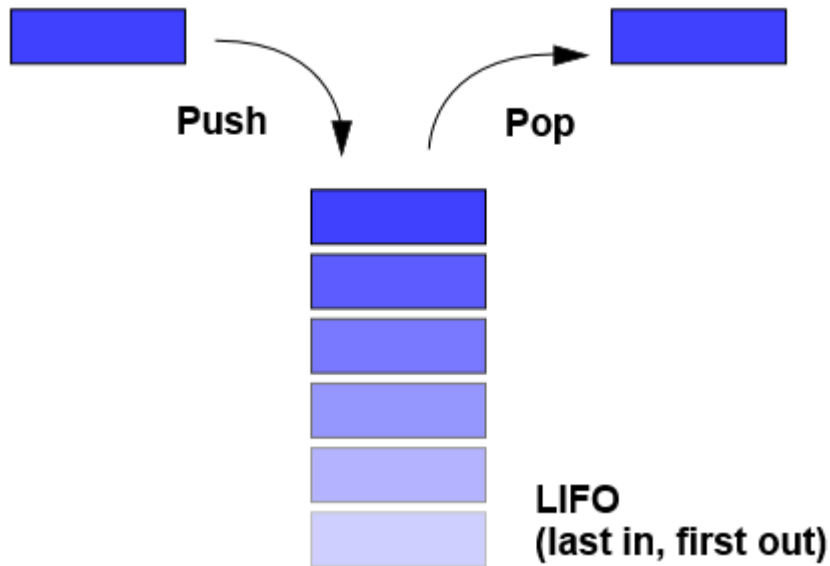
Initialise()
PushMatrix()
  Translate(2,2)
  Rotate(45)
  Draw_Square()
PopMatrix()
PushMatrix()
  Translate(6,3)
  Draw_Square()
  
```

Saving and loading transformations

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



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)

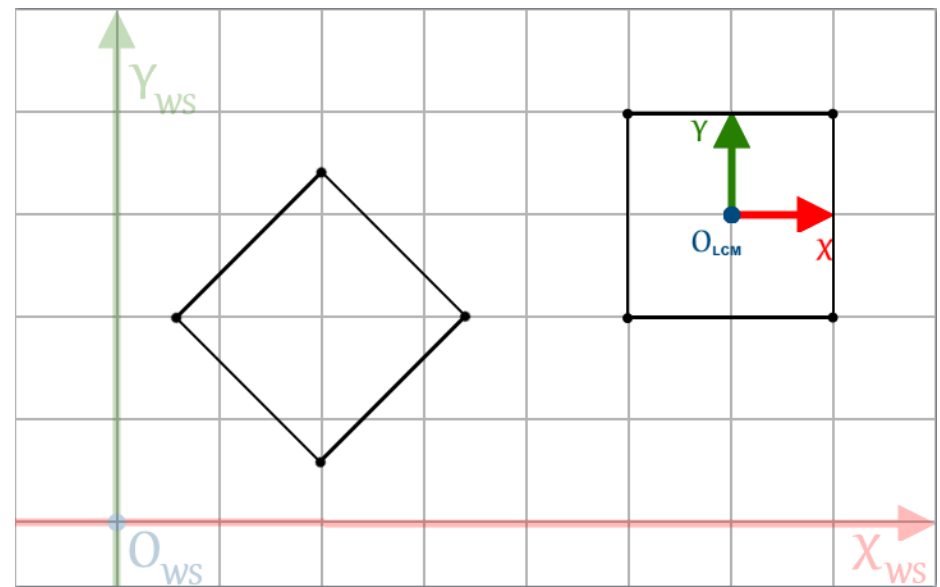
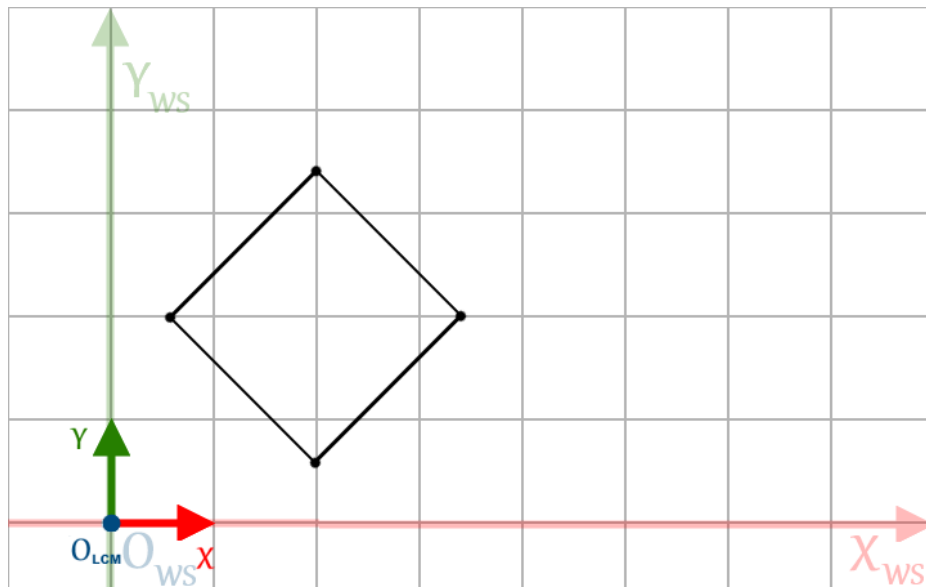
Hand configuration depends the elbow configuration, depends on shoulder configuration, and so on...

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



Hierarchies

It's slightly different in a hierarchy

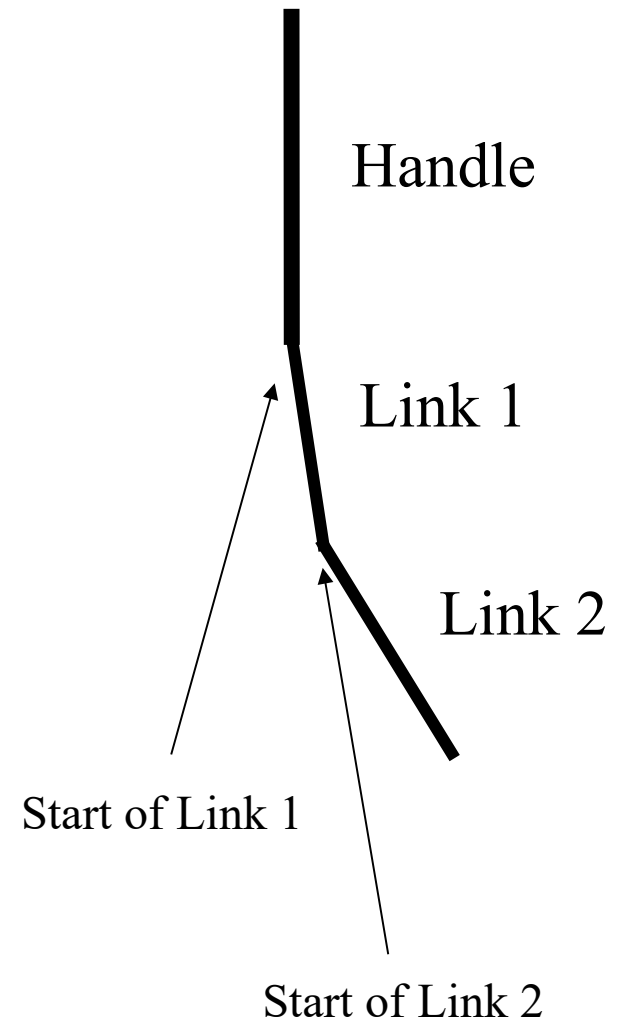
- Objects depend on others (a parent object) for their configurations (position and orientation)
- These objects need to be placed relative to their parent objects' coordinates, rather than in world-space

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



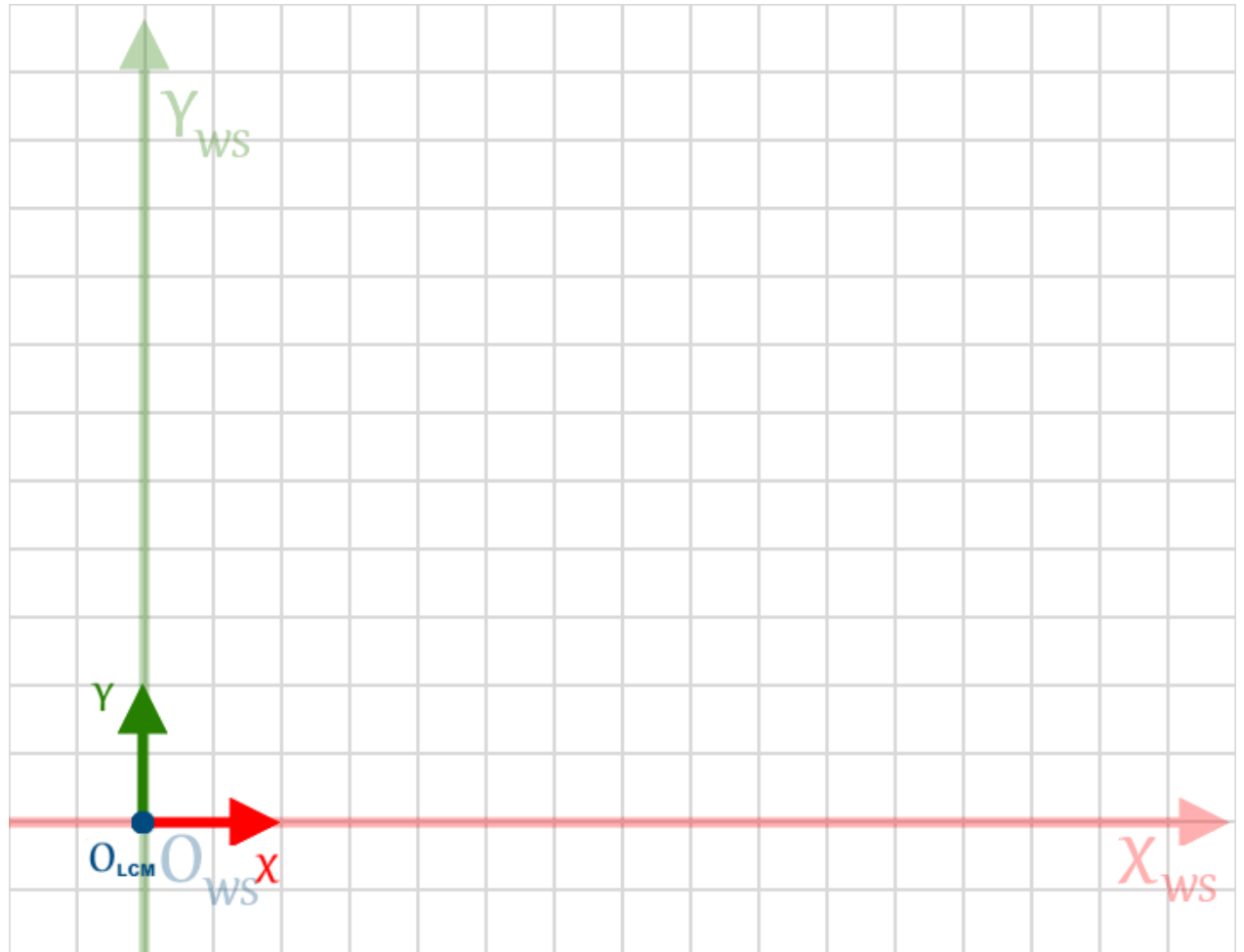
Step by step

- 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)
```

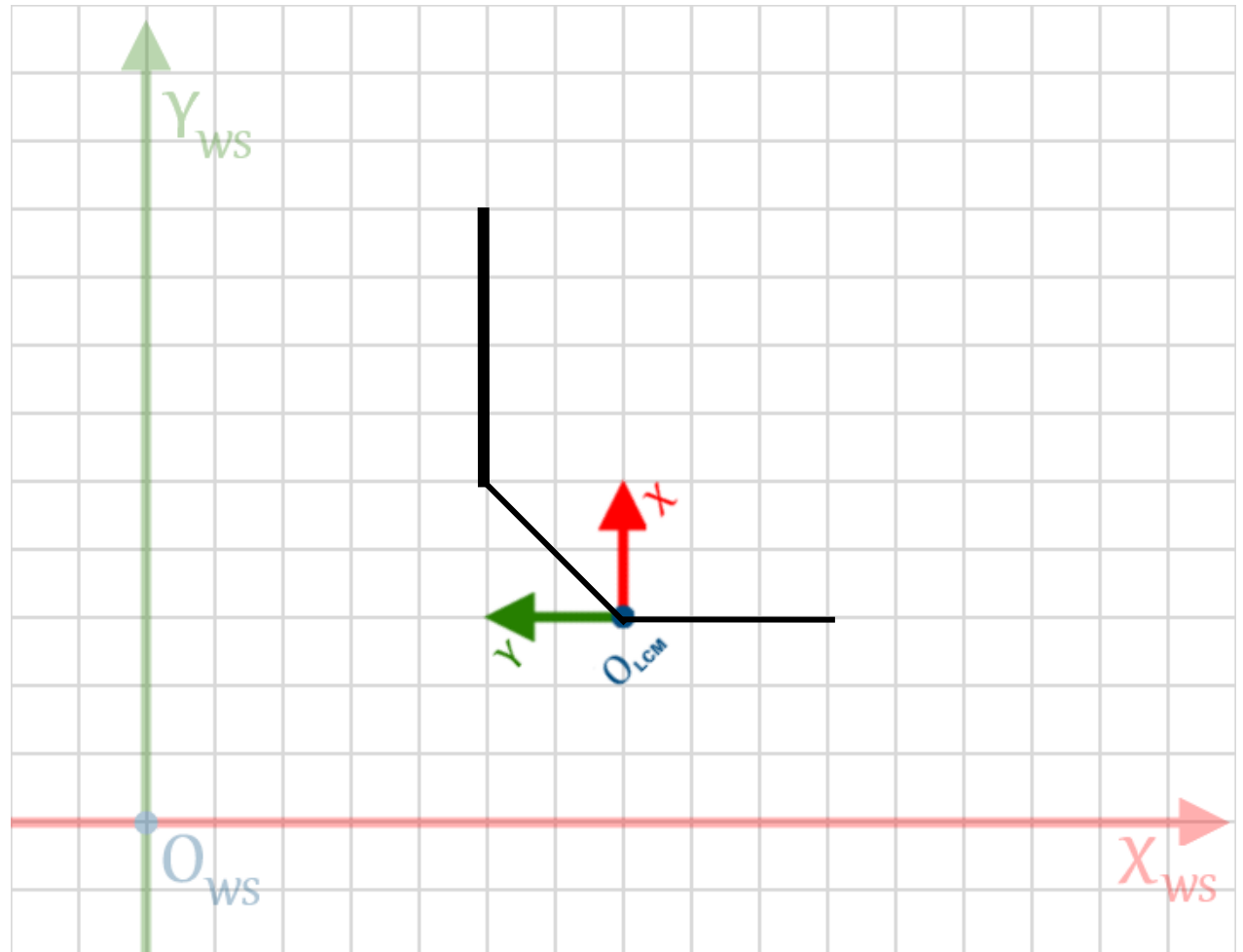


Step by step

- 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()
```

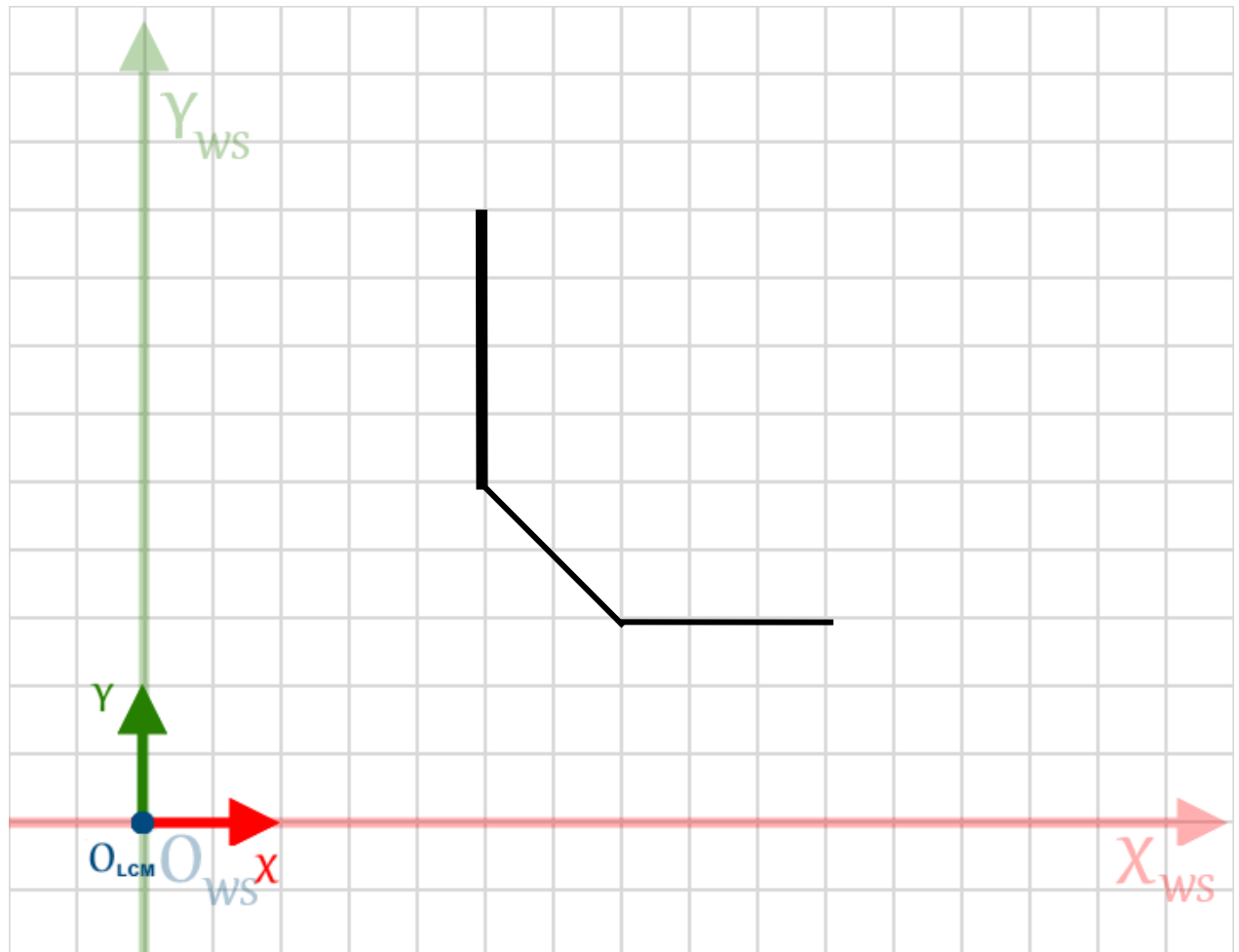


Step by step

- In more detail:

Transformation Operations

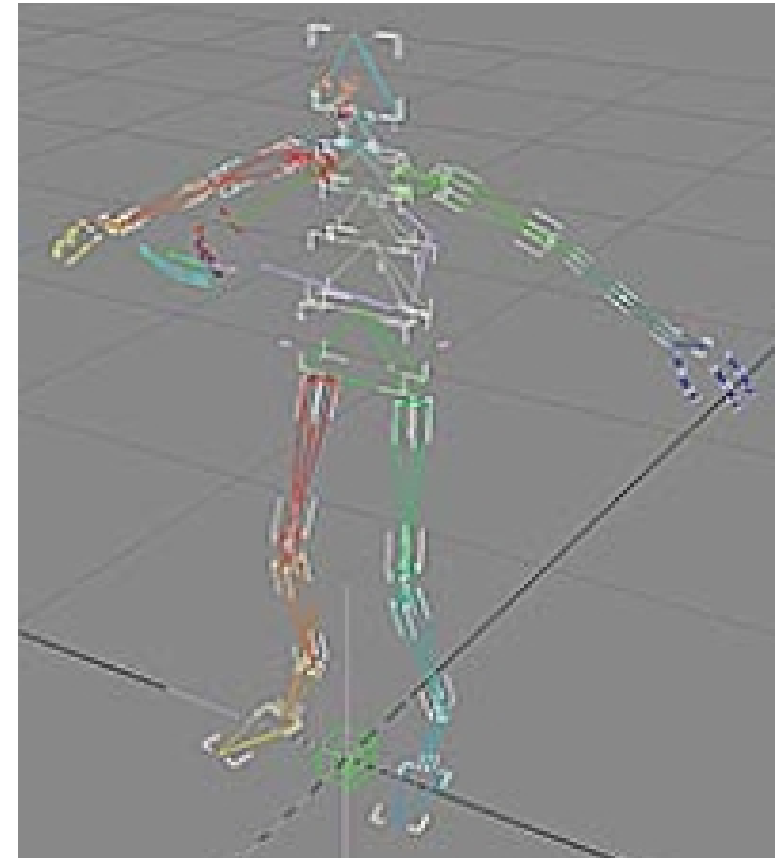
```
Initialise()  
PushMatrix()  
  Translate(Handle_pos)  
  DrawHandle()  
  Translate(Link1_trans)  
  Rotate(Link1_ang)  
  Draw_Link1()  
  Translate(Link2_trans)  
  Rotate(Link2_ang)  
  Draw_Link2()  
PopMatrix()
```



Humanoid Hierarchy

Use *glPushMatrix()* and *glPopMatrix()* to add/remove transforms from the stack

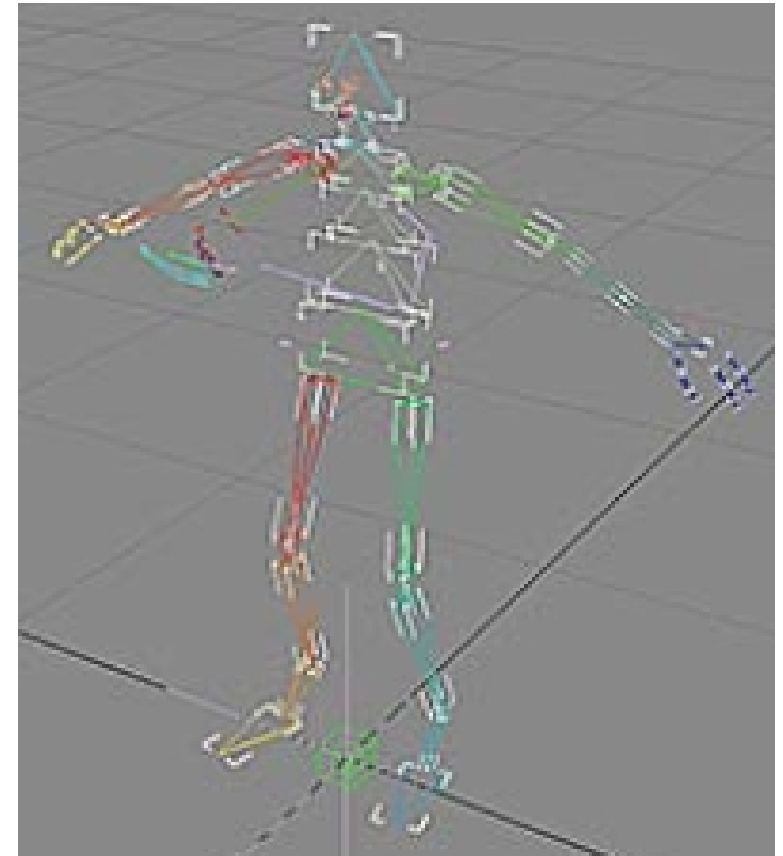
```
glPushMatrix();  
  glTranslate(position of hand);  
  Draw_palm_of_hand();  
glPopMatrix();
```



Humanoid Hierarchy

Use *glPushMatrix()* and *glPopMatrix()* to add/remove transforms from the stack

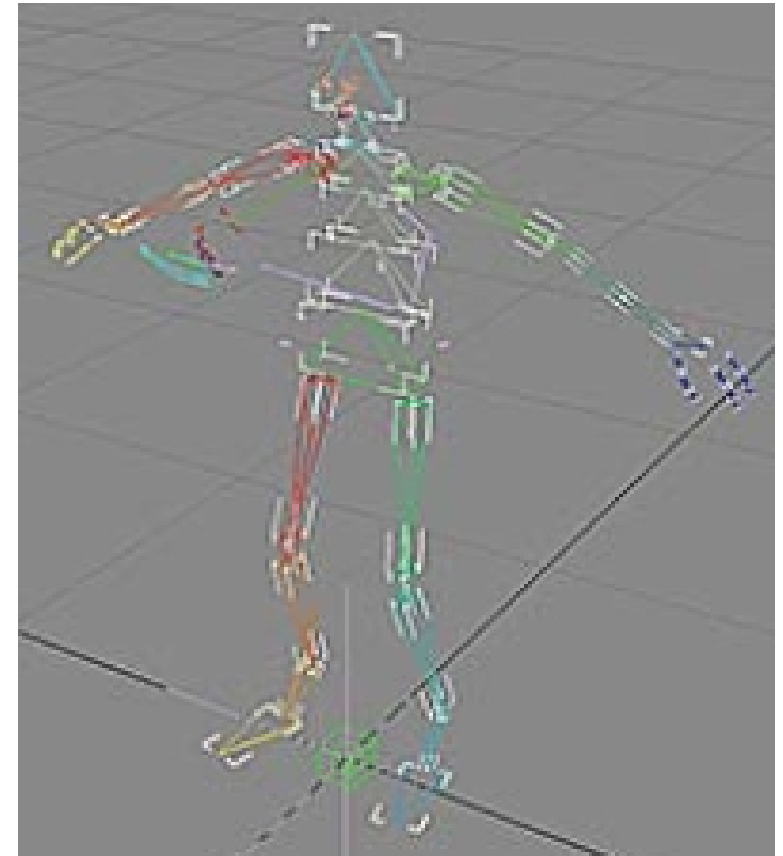
```
glPushMatrix();  
glTranslate(position of hand);  
Draw_palm_of_hand();  
glPushMatrix();  
    glTranslate(position of finger 1);  
    glRotatef(finger 1 rotation);  
    Draw_finger_1();  
glPopMatrix();  
glPopMatrix();
```



Humanoid Hierarchy

Use *glPushMatrix()* and *glPopMatrix()* to add/remove transforms from the stack

```
glPushMatrix();  
glTranslate(position of hand);  
Draw_palm_of_hand();  
glPushMatrix();  
    glTranslate(position of finger 1);  
    glRotatef(finger 1 rotation);  
    Draw_finger_1();  
glPopMatrix();  
glPushMatrix();  
    glTranslate(position of finger 2);  
    glRotatef(finger 2 rotation);  
    Draw_finger_2();  
glPopMatrix();  
glPopMatrix();
```



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