

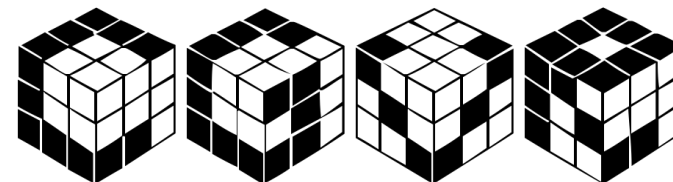
Computer Graphics

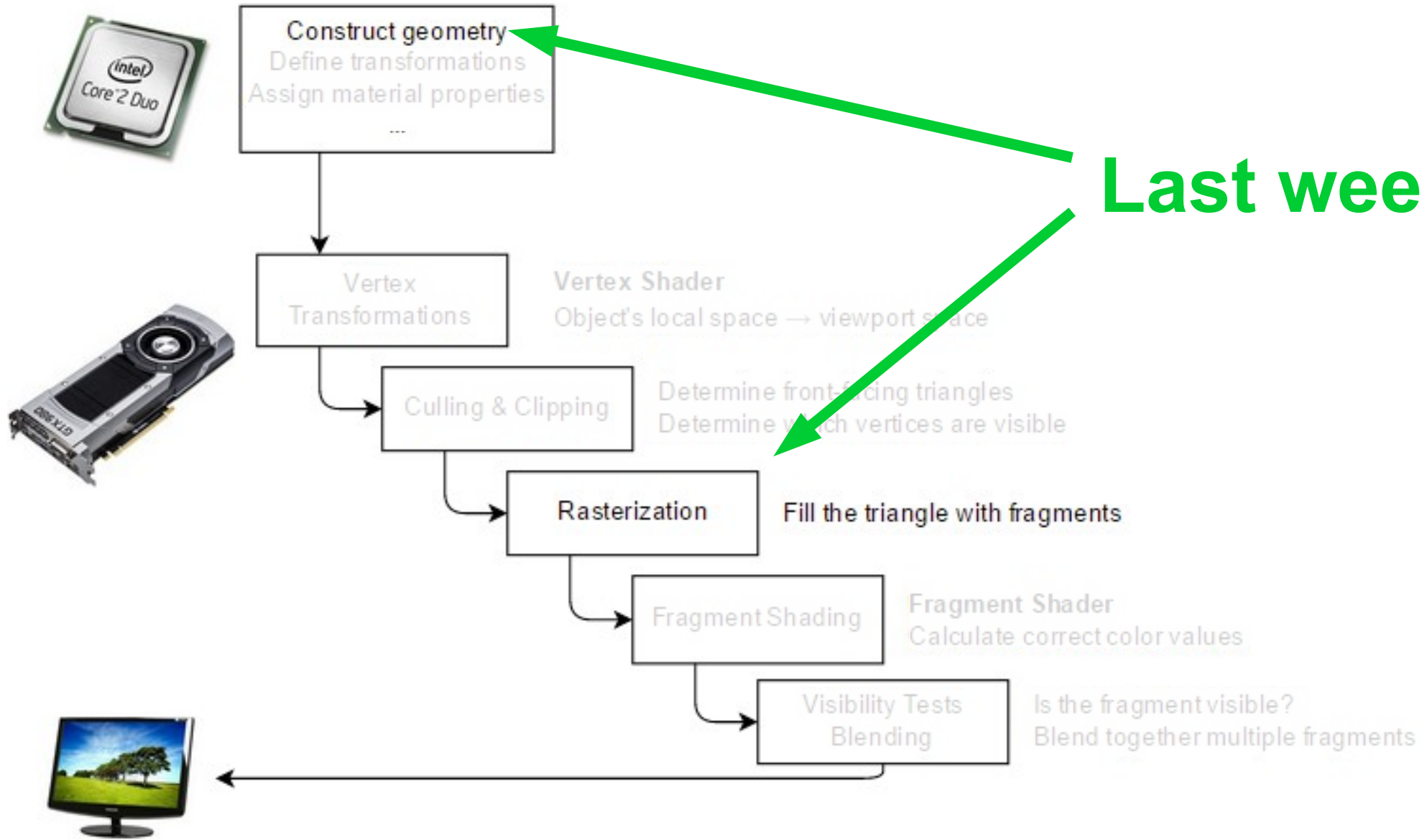
MTAT.03.015

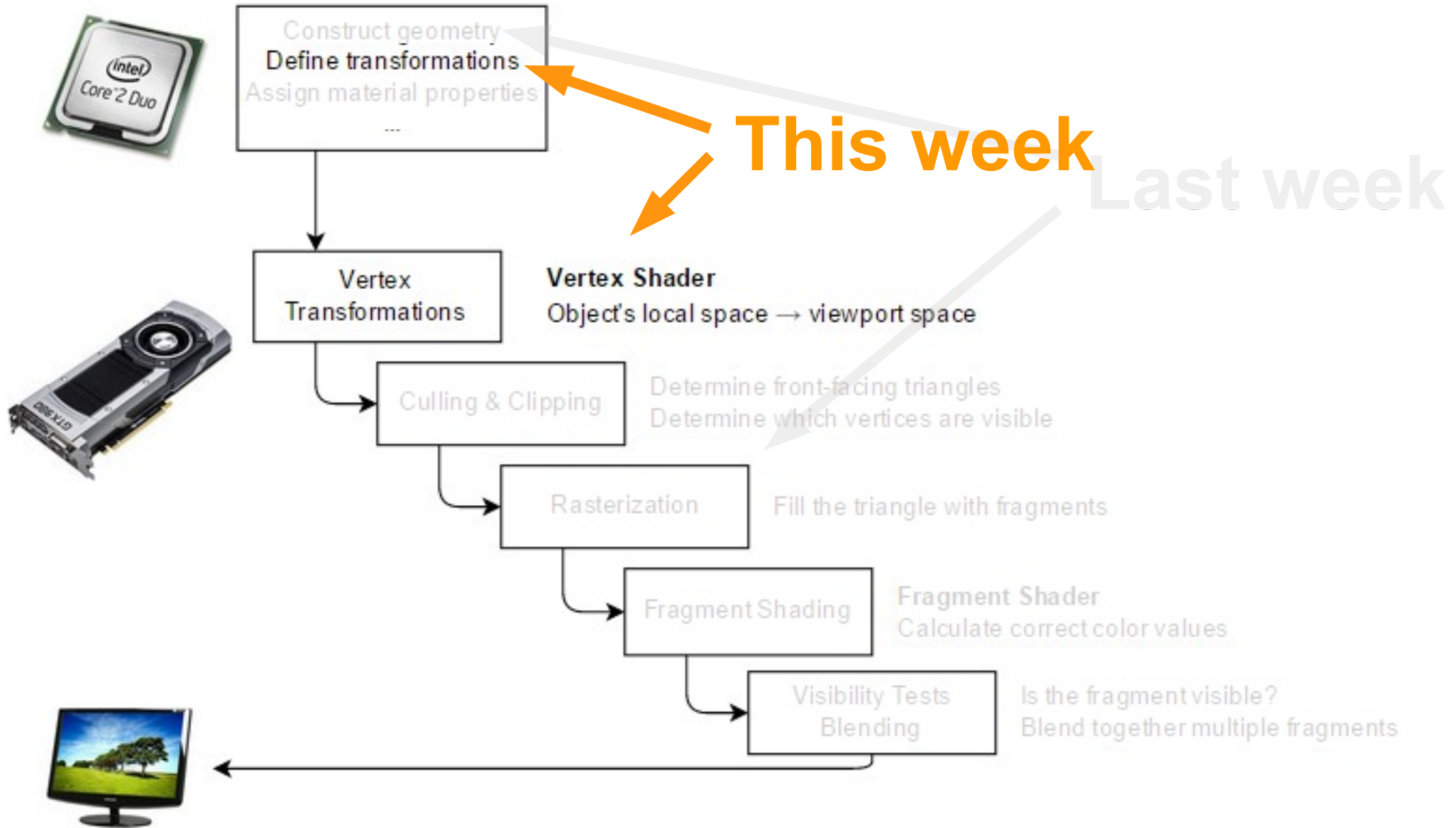
Raimond Tunnel



UNIVERSITY OF TARTU
Institute of Computer Science







Transformations

- Watch the Computerphile video, try to find out:
 - 1) Why are we using matrices?



The True Power of the Matrix (Transformations in Graphics) – Computerphile

Transformations

- Watch the Computerphile video, try to find out:
 - 1) Why are we using matrices?
 - 2) Where do the homogeneous coordinates come in?



The True Power of the Matrix (Transformations in Graphics) – Computerphile

Linear Transformations

- Also called *linear mapping*, *linear function*.

Linear Transformations

- Also called *linear mapping*, *linear function*.
- Transforms a vector space V into a vector space W , while preserving addition and scalar multiplication.

Linear Transformations

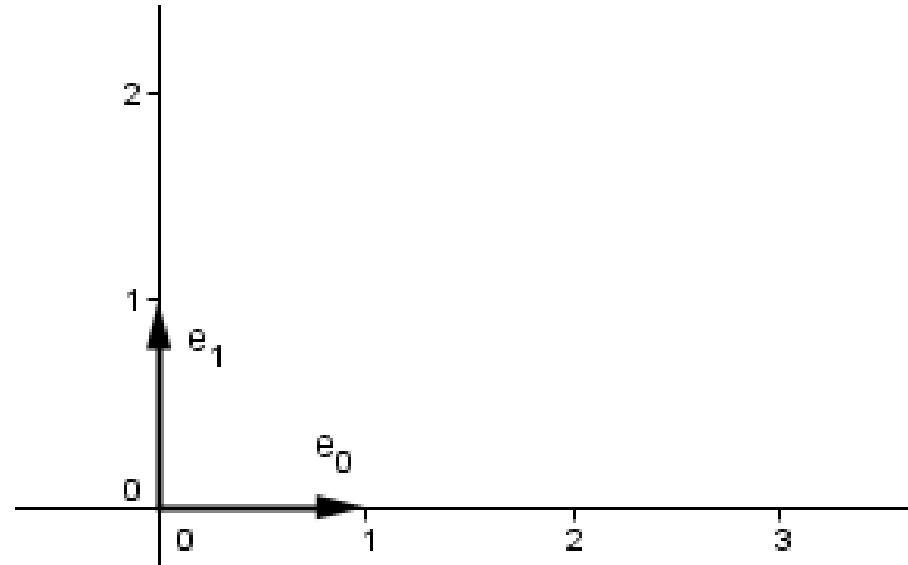
- Also called *linear mapping*, *linear function*
- Transforms a vector space V into a vector space W , while preserving addition and scalar multiplication
- Satisfies: $f(\alpha \cdot v + \beta \cdot u) = \alpha \cdot f(v) + \beta \cdot f(u)$

Linear Transformations

- Also called *linear mapping*, *linear function*
- Transforms a vector space V into a vector space W , while preserving addition and scalar multiplication
- Satisfies: $f(\alpha \cdot v + \beta \cdot u) = \alpha \cdot f(v) + \beta \cdot f(u)$
- In 3D: $\alpha, \beta \in \mathbb{R} \quad u, v \in \mathbb{R}^3$

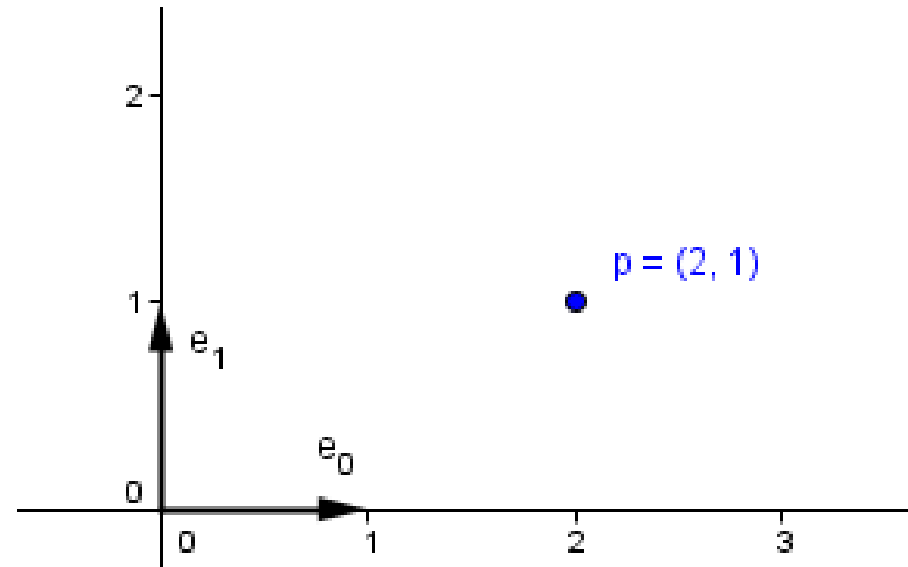
Linear Transformations

- Take our vector space of points



Linear Transformations

- Take our vector space of points
- Take for example a point $p = (2, 1)$



Linear Transformations

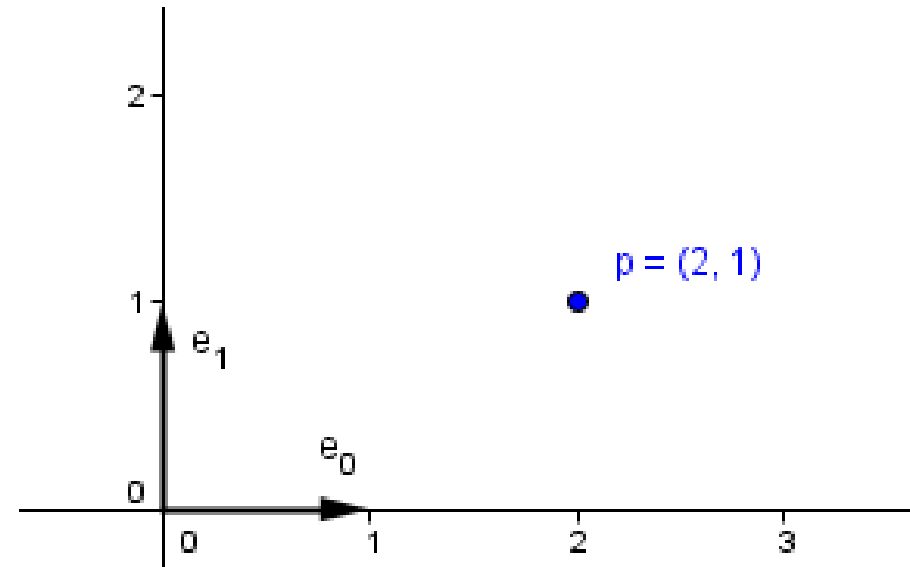
- Take our vector space of points
- Take for example a point $p = (2, 1)$
- Try mappings:

$$1) f(p) = (p_x, p_y)$$

$$2) f(p) = (2 \cdot p_x, p_y)$$

$$3) f(p) = (p_x, 2 \cdot p_y)$$

$$4) f(p) = (2 \cdot p_x, 2 \cdot p_y)$$



Linear Transformations

- From Algebra you know that all linear **transformations** can be represented **as matrices**.

Linear transformation \rightarrow Matrix

Linear Transformations

- From Algebra you know that all linear **transformations** can be represented as **matrices**.
- **Every matrix** also gives you a **linear transformation**.

Linear transformation \rightarrow Matrix

Linear transformation \leftarrow Matrix

Linear Transformations

- What would be the matrices for the linear transformations we just saw?

$$f(p) = \begin{pmatrix} ? & ? \\ ? & ? \end{pmatrix} \cdot \begin{pmatrix} p_x \\ p_y \end{pmatrix}$$



$$f(p) = (p_x, p_y)$$

$$f(p) = (p_x, 2 \cdot p_y)$$

$$f(p) = (2 \cdot p_x, p_y)$$

$$f(p) = (2 \cdot p_x, 2 \cdot p_y)$$

Scale

- Stretches or shrinks the space

2D

$$\begin{pmatrix} a_x & 0 \\ 0 & a_y \end{pmatrix}$$

a_x – x-axis scale factor

a_y – y-axis scale factor

3D

$$\begin{pmatrix} a_x & 0 & 0 \\ 0 & a_y & 0 \\ 0 & 0 & a_z \end{pmatrix}$$

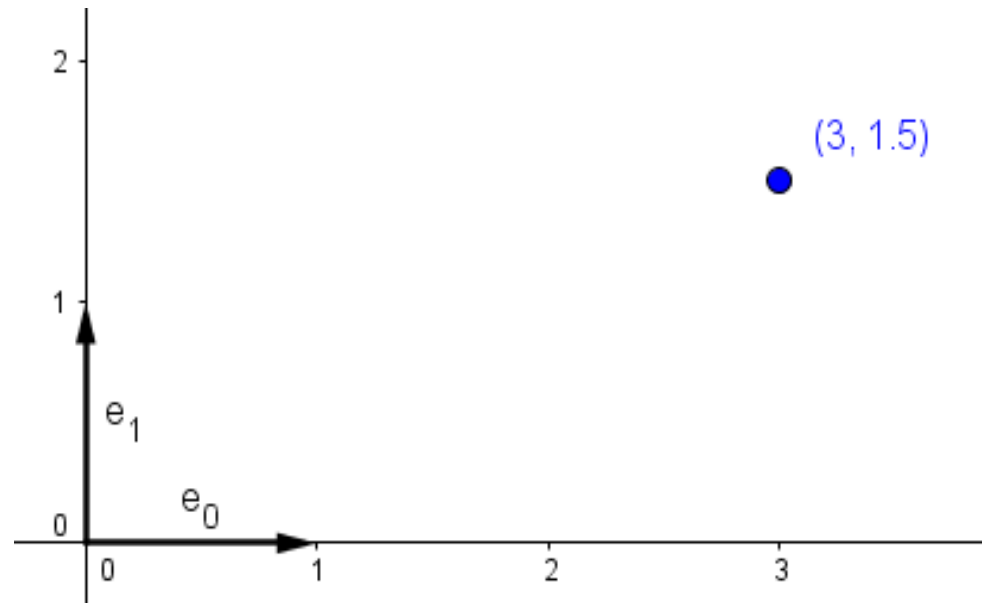
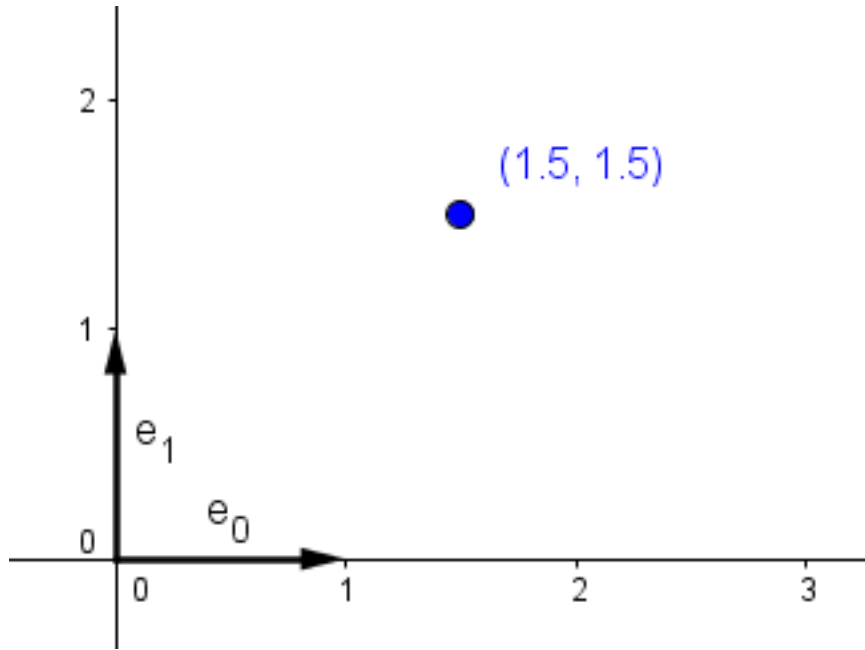
a_x – x-axis scale factor

a_y – y-axis scale factor

a_z – z-axis scale factor

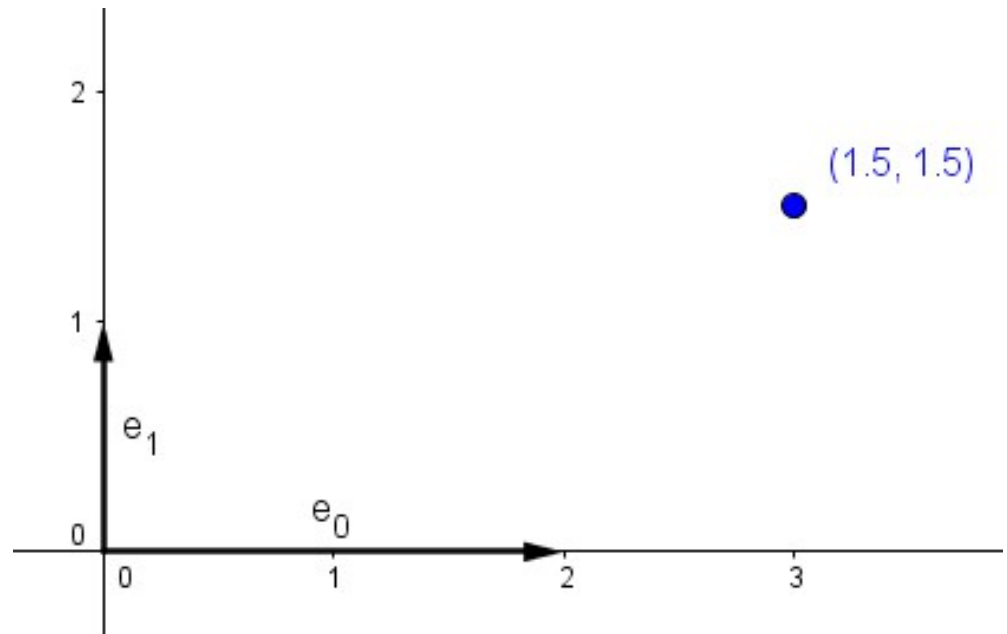
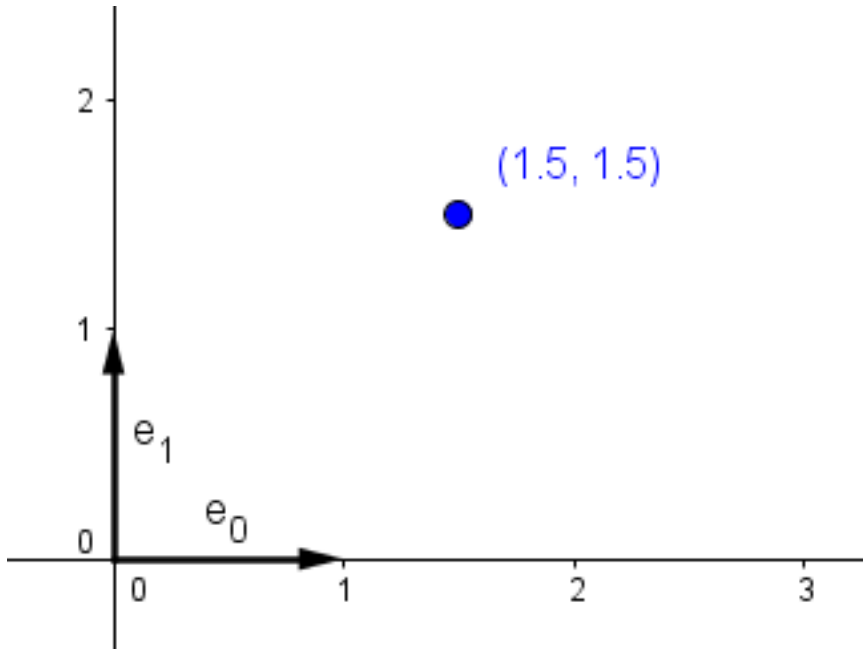
Scale

- Transformations can be easily understood, if we see what they do with the standard basis

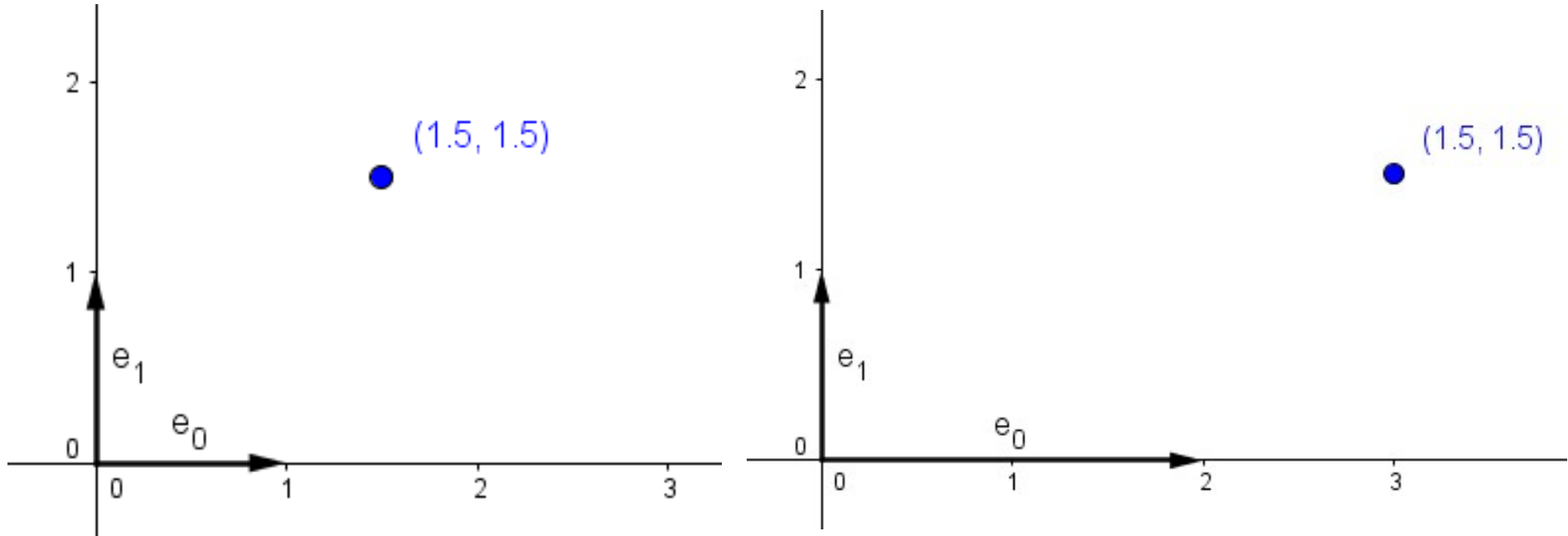


Scale

- Transformations can be easily understood, if we see what they do with the standard basis



Scale

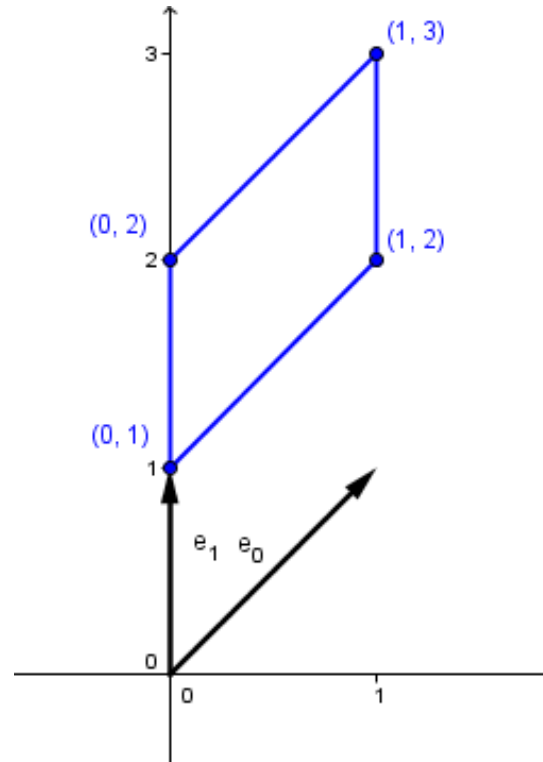
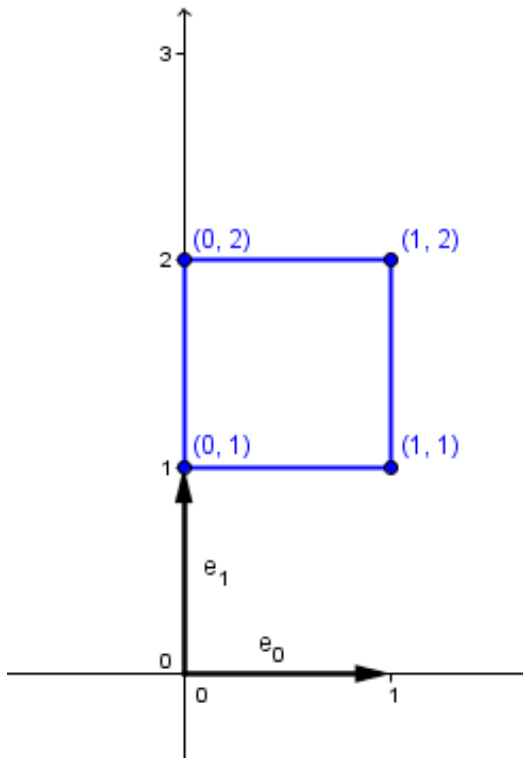


- One can read the transformed standard basis from the columns of the transformation matrix!



Shear

- Shear-x, shear-y
- Tilts one of the axes parallel to other(s)



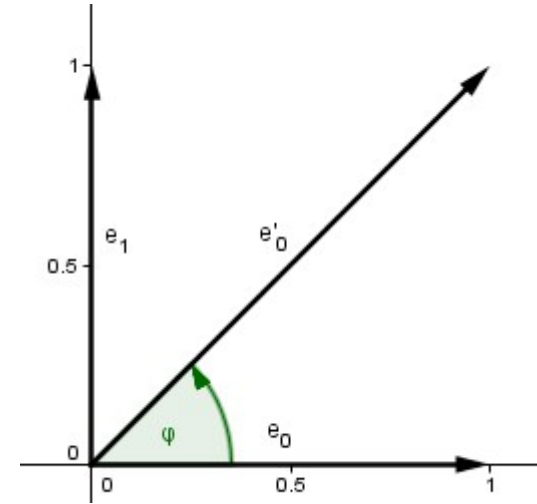
Shear-x or shear-y?
Matrix?



Shear

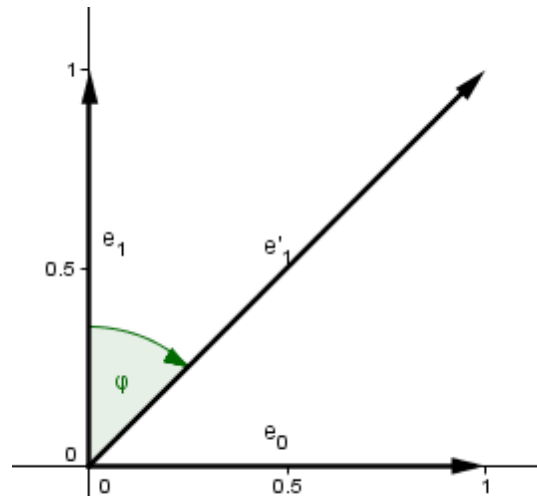
- Shear-y, we tilt the x basis parallel to y by angle φ counter-clockwise

$$\begin{pmatrix} 1 & 0 \\ \tan(\varphi) & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x \\ y + \tan(\varphi) \cdot x \end{pmatrix}$$



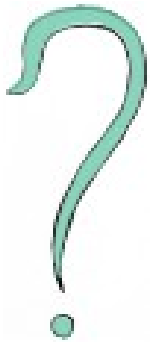
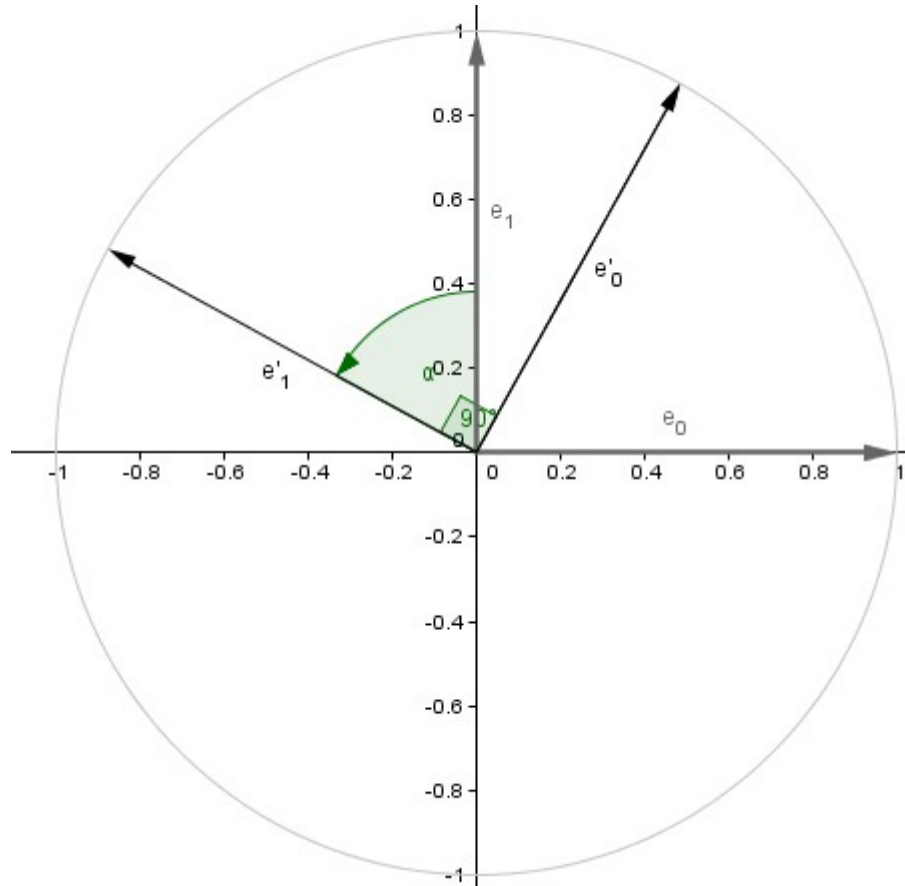
- Shear-x, we tilt the y basis parallel to x by angle φ clockwise

$$\begin{pmatrix} 1 & \tan(\varphi) \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x + \tan(\varphi) \cdot y \\ y \end{pmatrix}$$



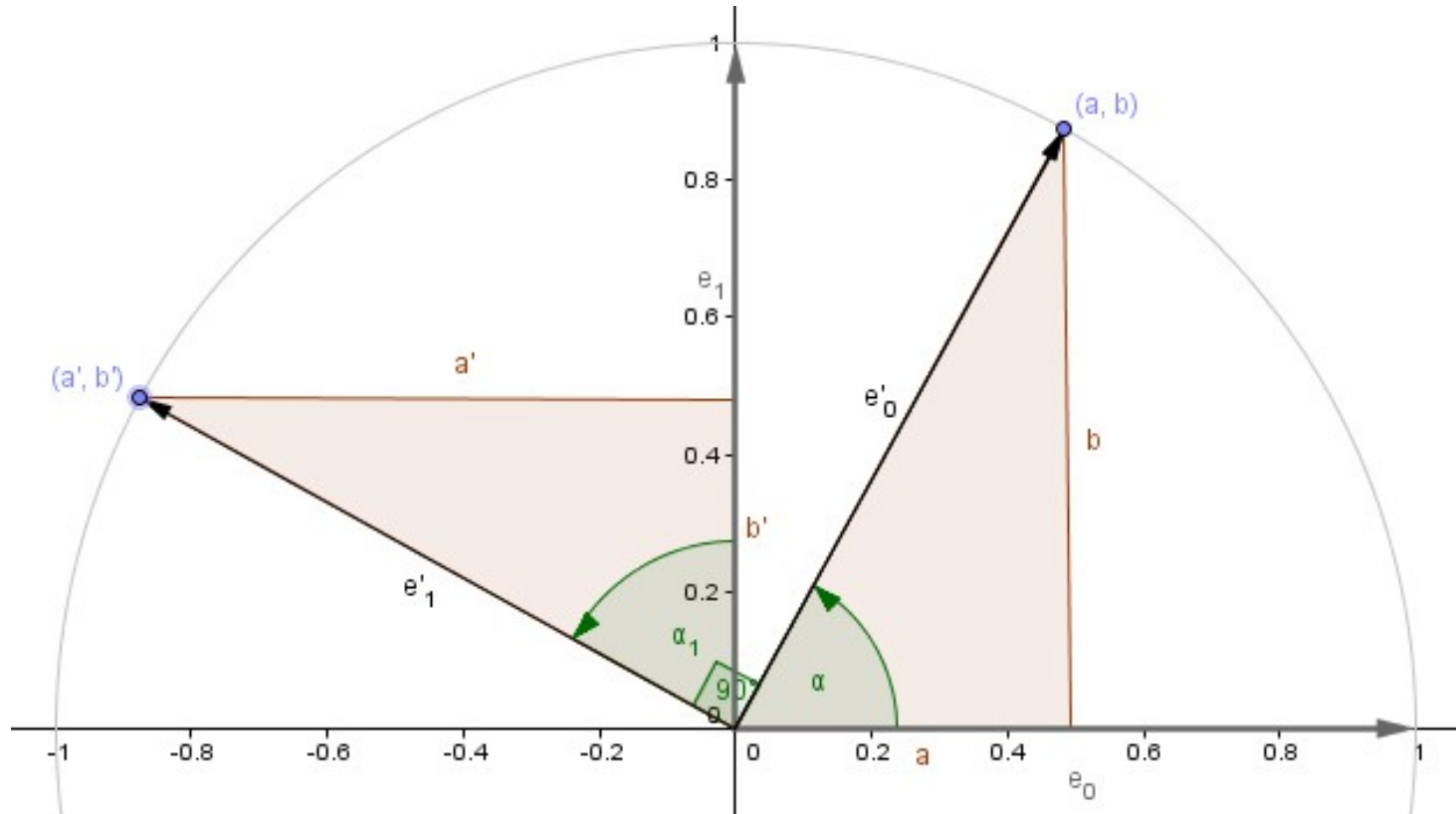
Rotation

- We want to keep the basis vectors on the unit-circle.



Can you derive the matrix?

Rotation



$$e'_0 = (a, b) = (\cos(\alpha), \sin(\alpha))$$

$$e'_1 = (-a', b') = (-\sin(\alpha), \cos(\alpha))$$

$$\cos(\alpha) = \frac{a}{\|e'_0\|} = \frac{a}{1} = a$$

Rotation

- Rotates around the z axis by the angle α

$$2D \quad \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{pmatrix} \quad \alpha - \text{Positive angle to rotate by}$$

Rotation

- Rotates around the z axis by the angle α

2D

$$\begin{pmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{pmatrix}$$

α – Positive angle to rotate by

3D

- Similar matrices that rotate around each main axis.

Rotation

- Rotates around the z axis by the angle α

2D

$$\begin{pmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{pmatrix}$$

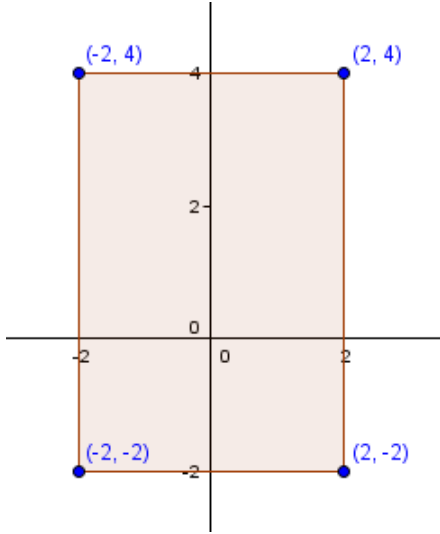
α – Positive angle to rotate by

3D

- Similar matrices that rotate around each main axis.
- What about rotation around an arbitrary axis?

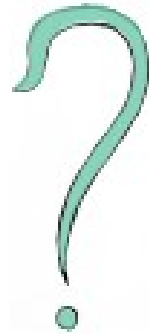
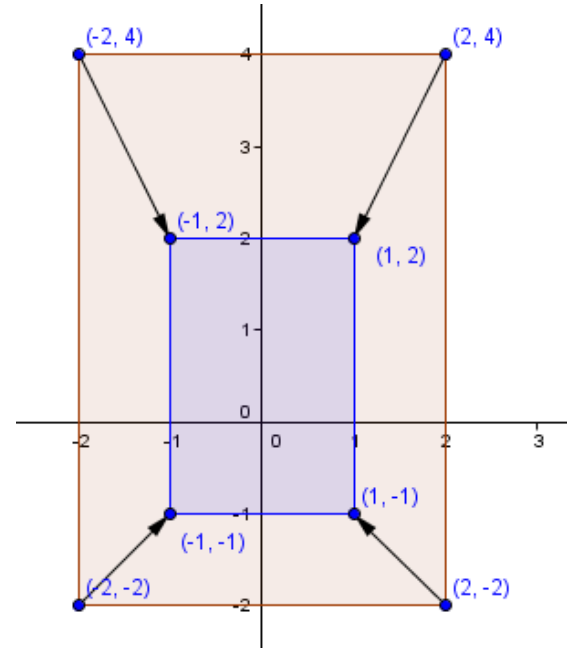
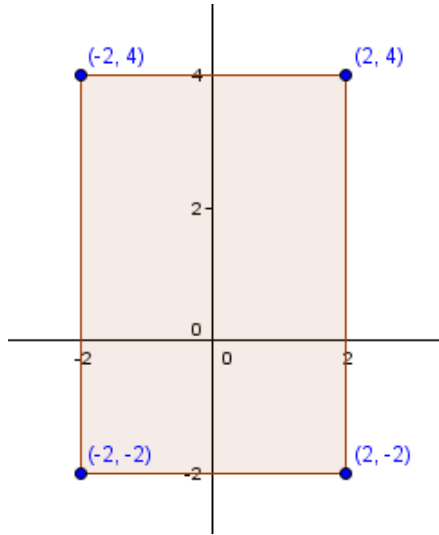


Linear Transformations

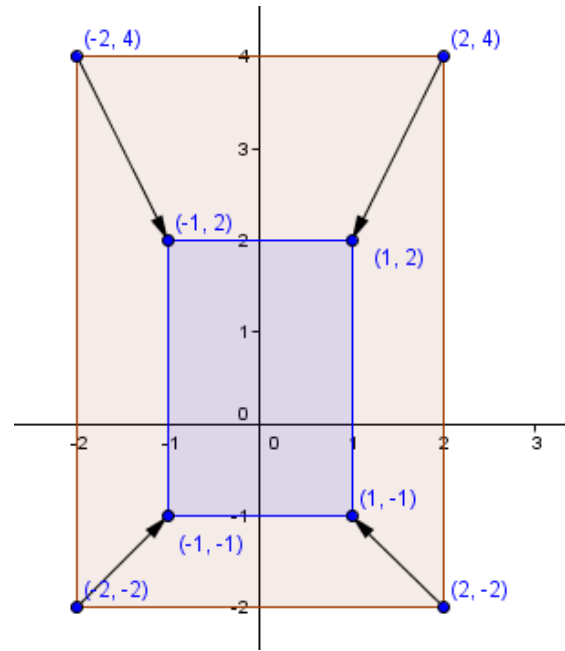
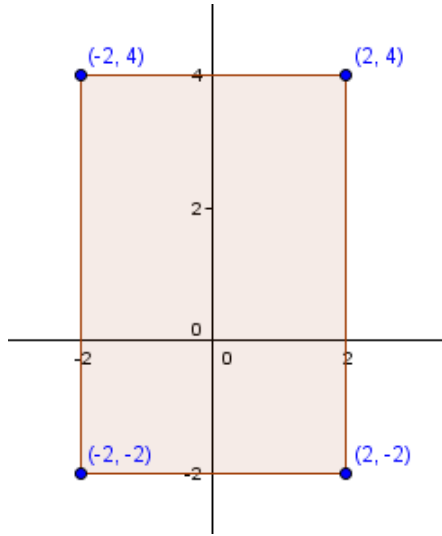


Defined geometry

Linear Transformations

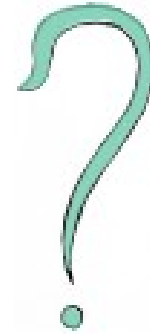
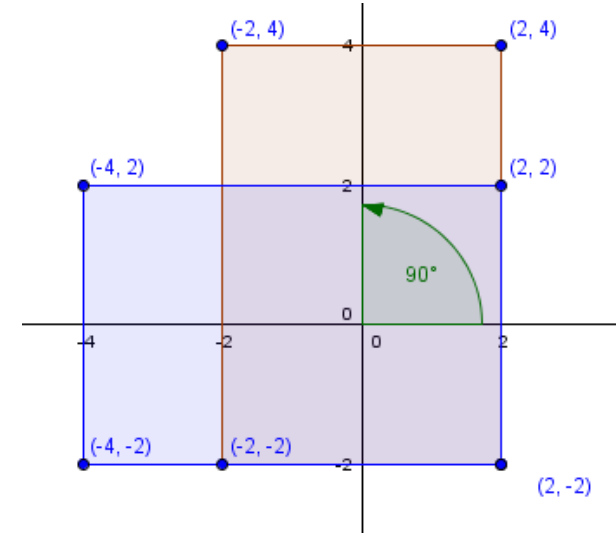
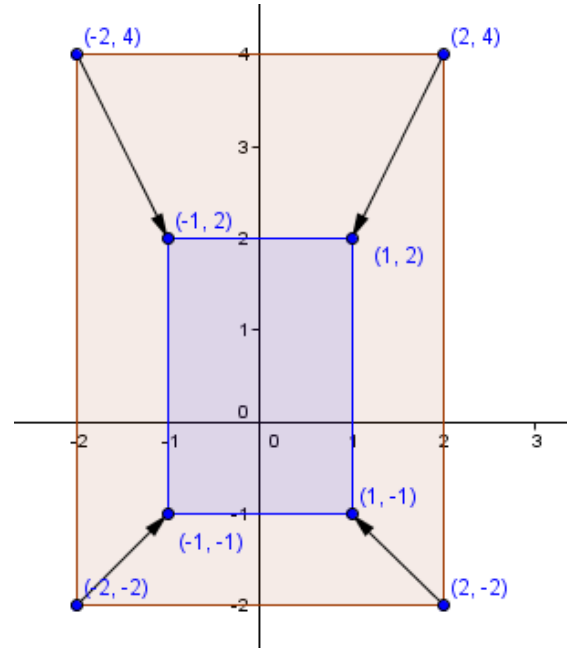
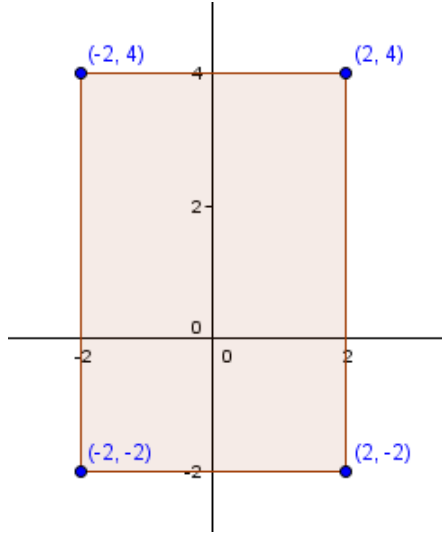


Linear Transformations

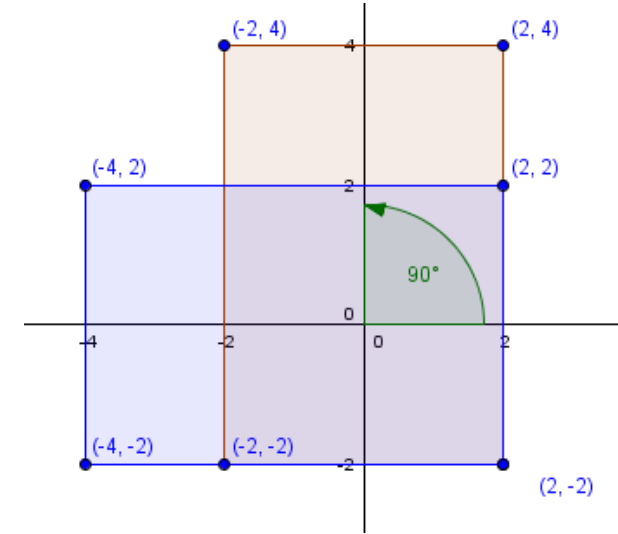
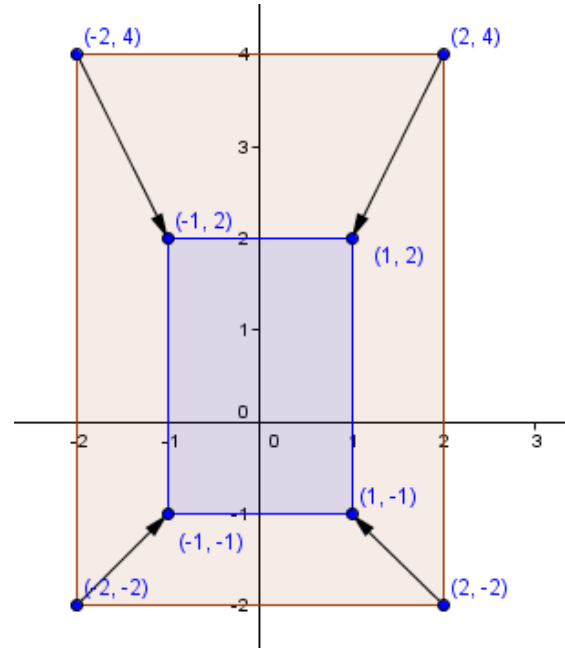
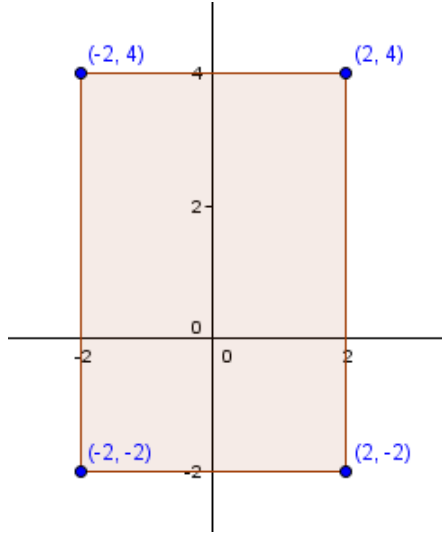


Scale

Linear Transformations

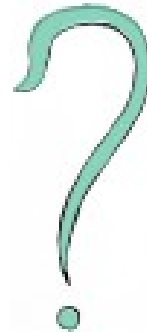
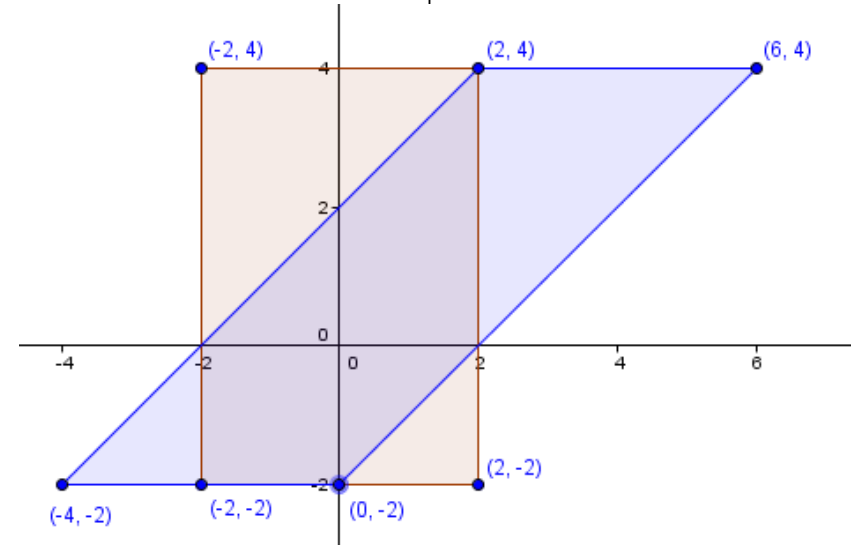
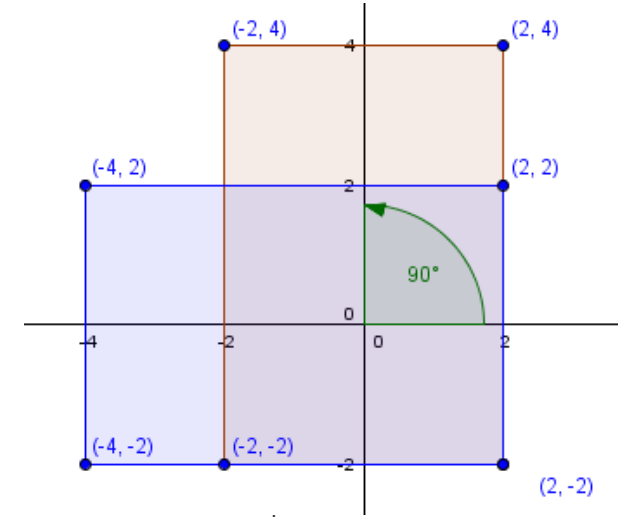
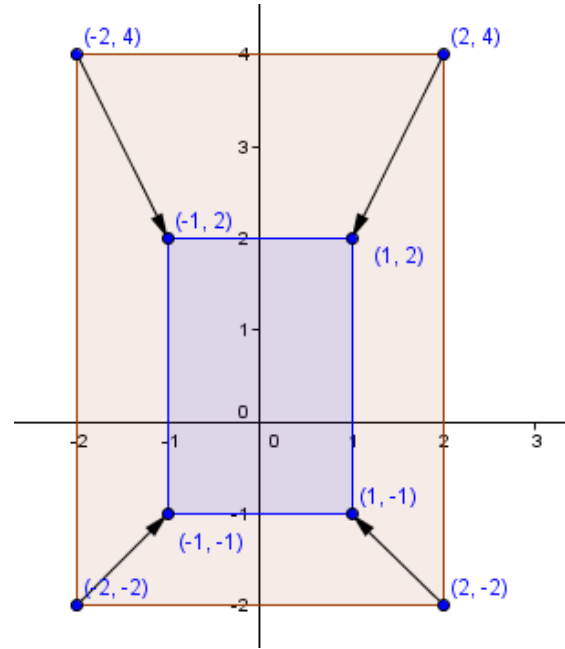
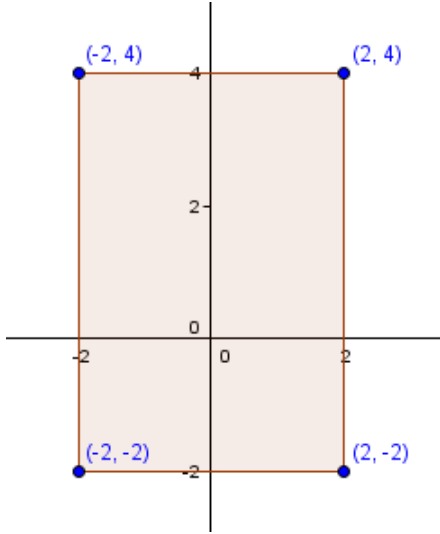


Linear Transformations

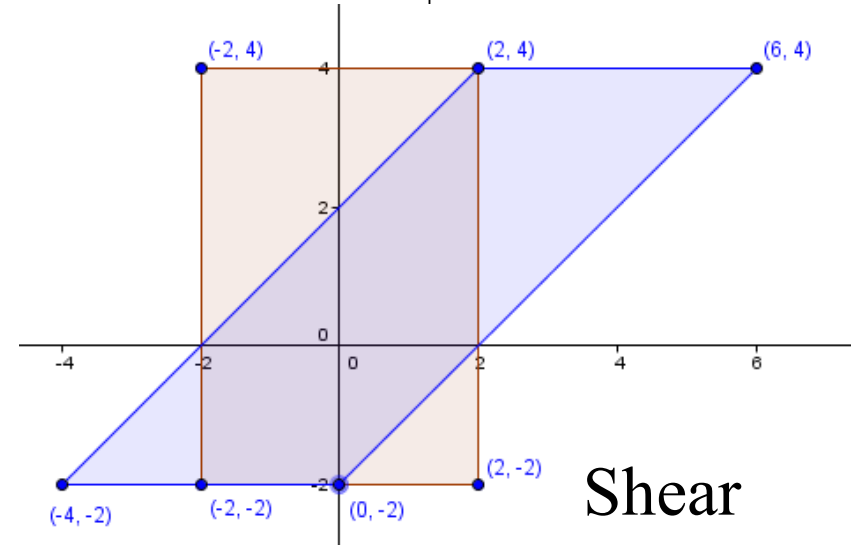
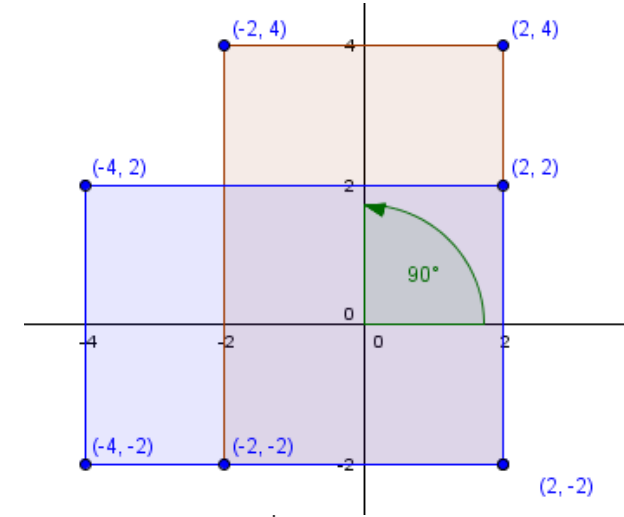
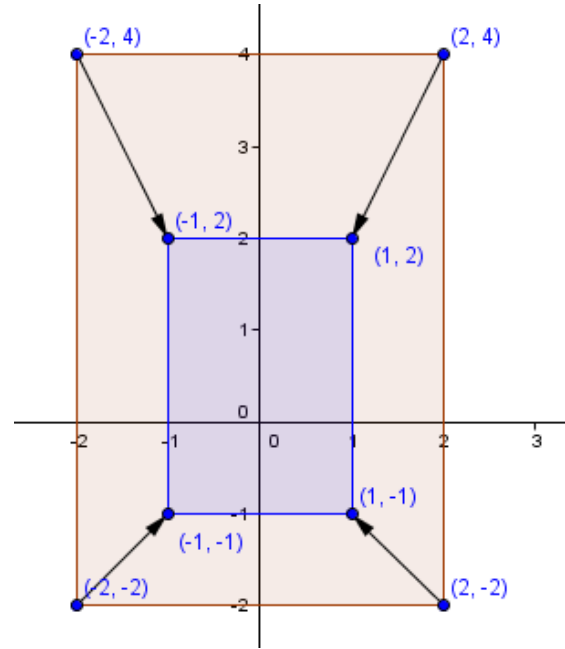
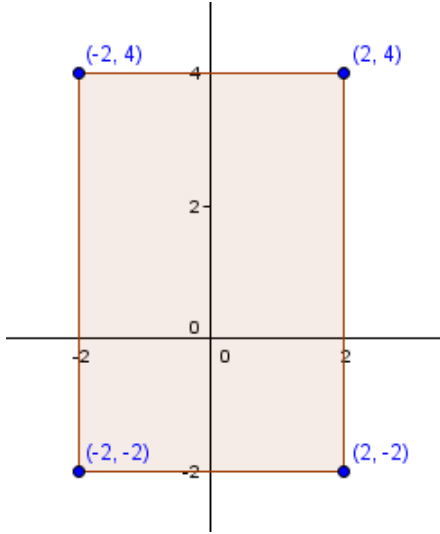


Rotation

Linear Transformations

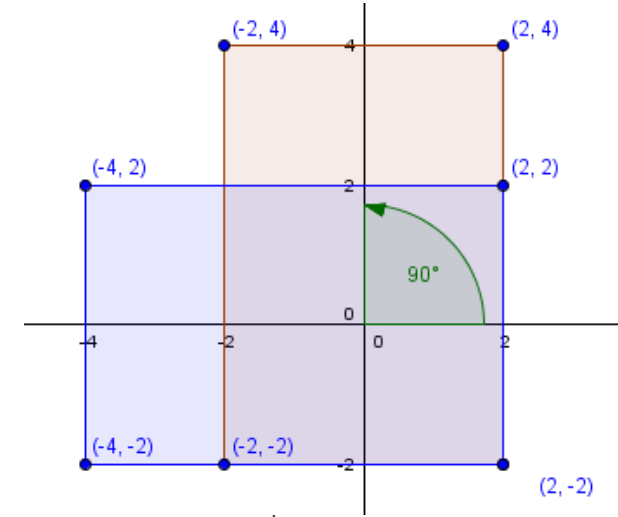
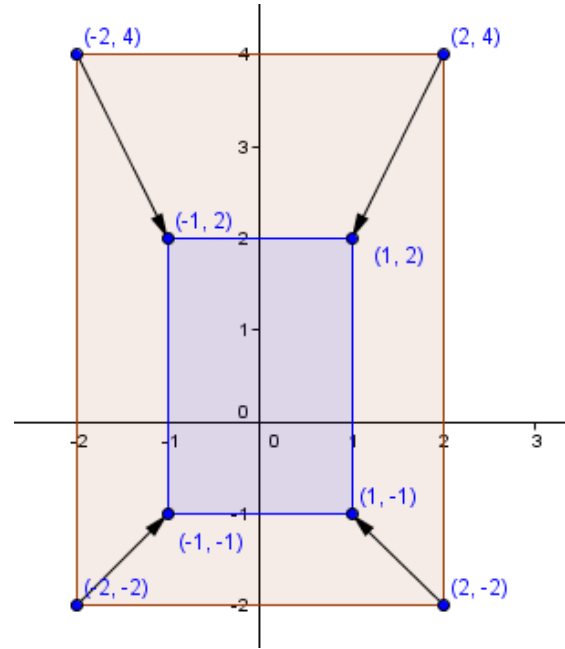
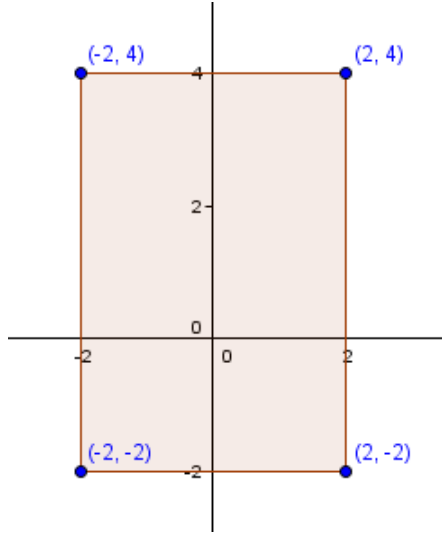


Linear Transformations

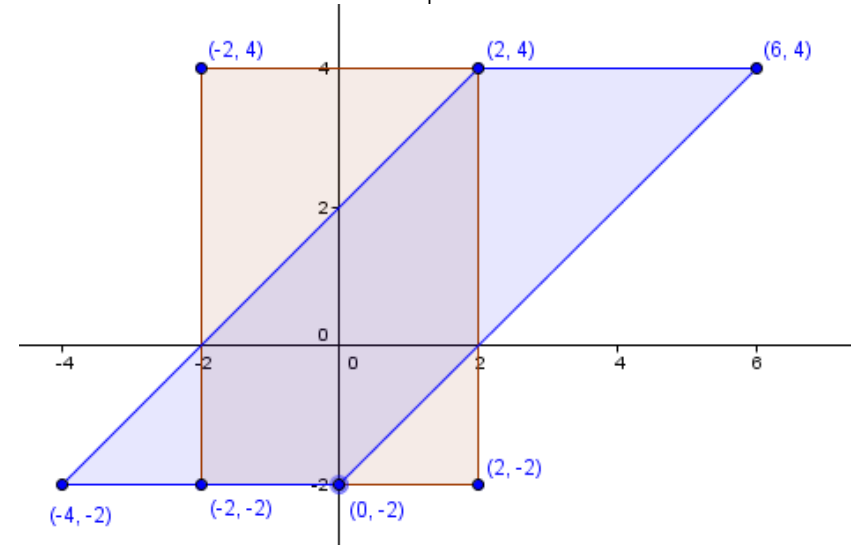


Shear

Linear Transformations



- Will these be enough?

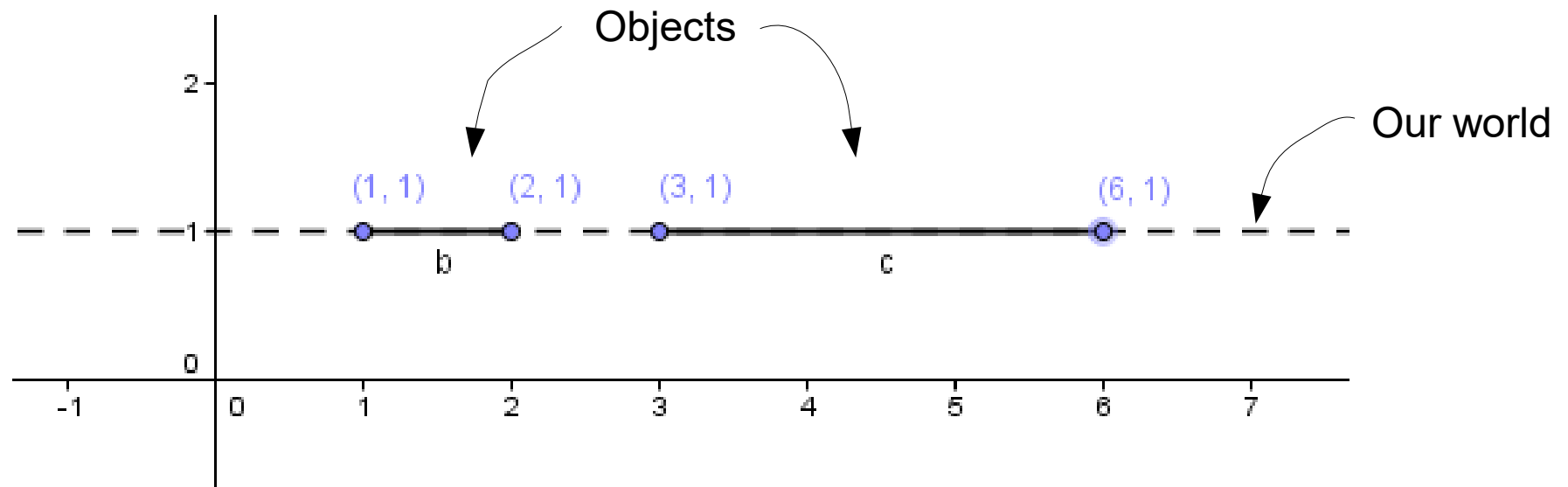


Translation

- Imagine a 1D world located at $y=1$ line in 2D.

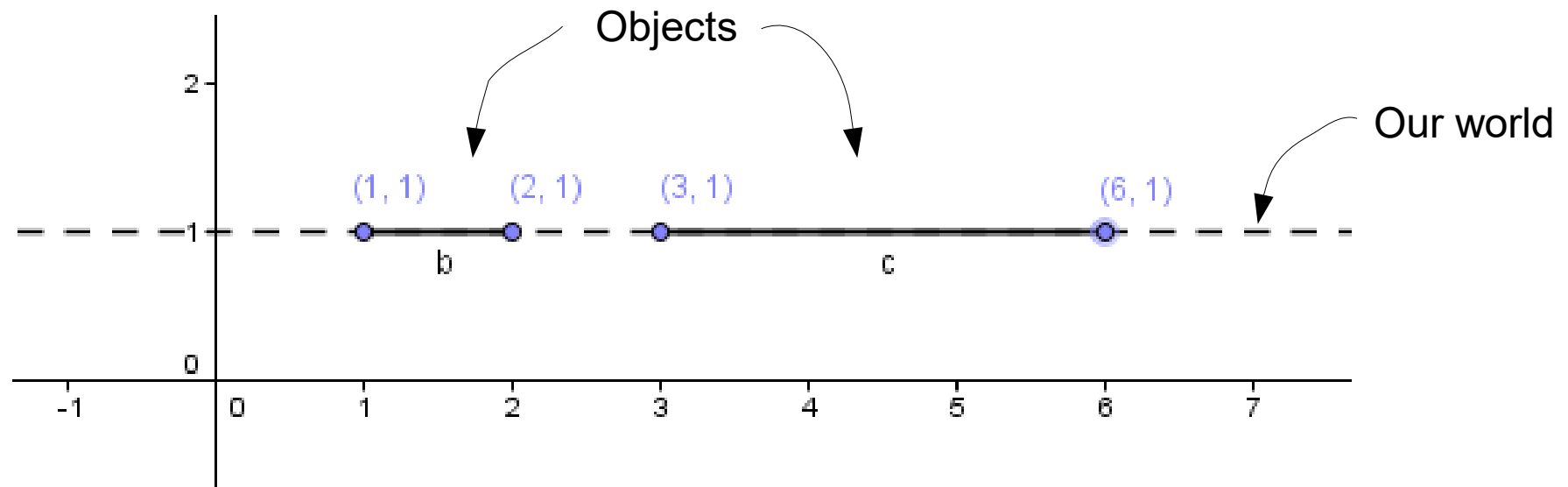
Translation

- Imagine a 1D world located at $y=1$ line in 2D.



Translation

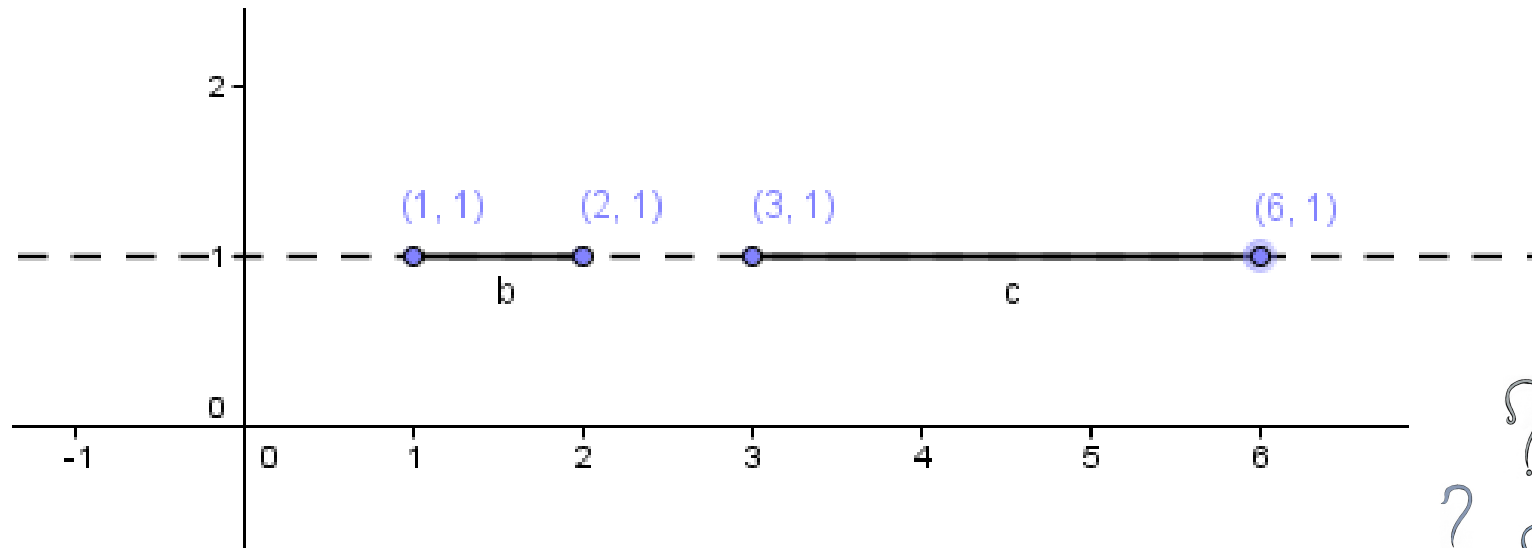
- Imagine a 1D world located at $y=1$ line in 2D.



- Notice that all the points are in the form: $(x, 1)$

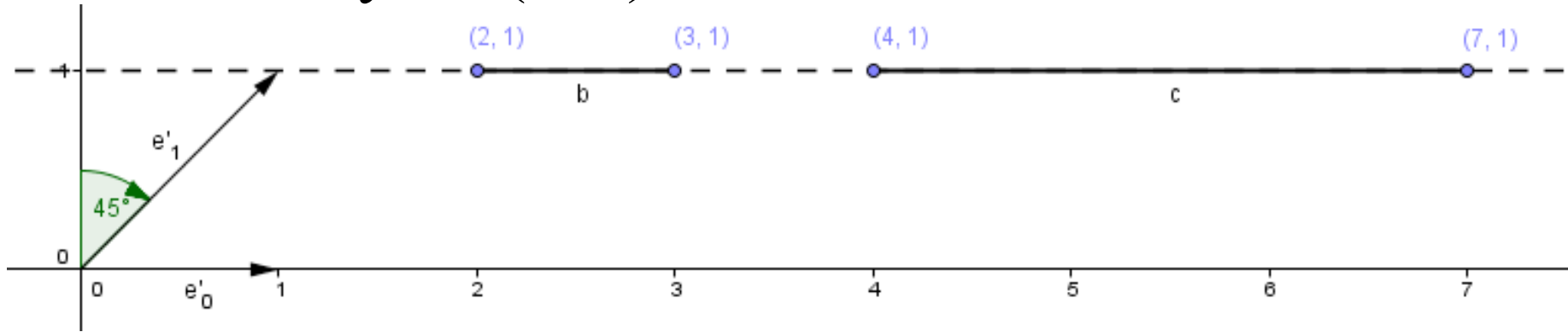
Translation

- How to transform the 2D space so that stuff in the 1D hyperplane $y=1$ moves an equal amount?

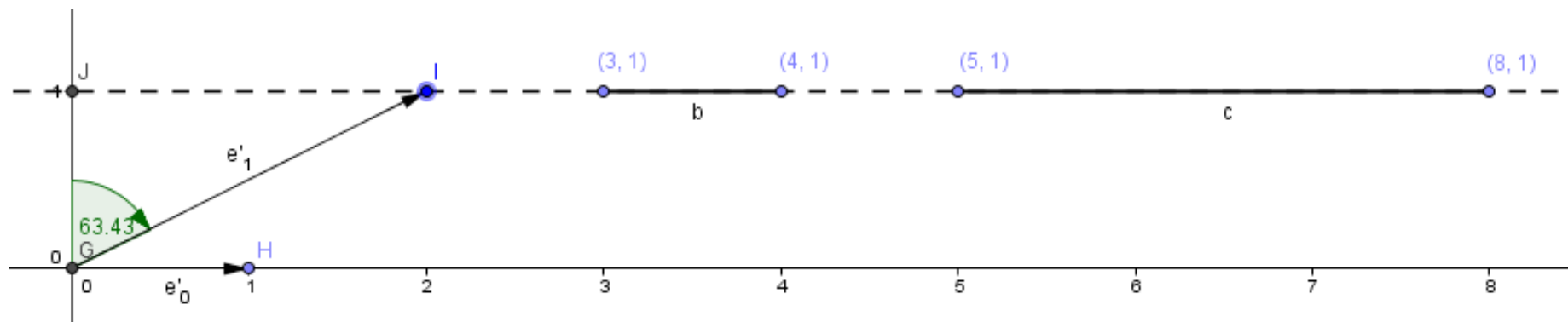


Translation

- Shear-x by $\tan(45^\circ) = 1$



- Shear-x with $\tan(63.4^\circ) = 2$



Translation

- Affine transformation in the current space, linear shear transformation in 1 dimension higher space.

$$\text{2D Shear-xy} \begin{pmatrix} 1 & 0 & x_t \\ 0 & 1 & y_t \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} x + x_t \\ y + y_t \\ 1 \end{pmatrix}$$

$$\text{1D Shear-x} \begin{pmatrix} 1 & x_t \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ 1 \end{pmatrix} = \begin{pmatrix} x + x_t \\ 1 \end{pmatrix}$$

3D Shear-xyz

$$\begin{pmatrix} 1 & 0 & 0 & x_t \\ 0 & 1 & 0 & y_t \\ 0 & 0 & 1 & z_t \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x + x_t \\ y + y_t \\ z + z_t \\ 1 \end{pmatrix}$$

Transformations

- This together gives us a **very good toolset** to transform our geometry as we wish.

Affine
transformation

$$\begin{pmatrix} a & b & c & x_t \\ d & e & f & y_t \\ g & h & i & z_t \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} ax + by + cz + x_t \\ dx + ey + fz + y_t \\ gx + hy + iz + z_t \\ 1 \end{pmatrix}$$



Transformations

- This together gives us a **very good toolset** to transform our geometry as we wish.

Linear transformations

Affine
transformation

$$\begin{pmatrix} a & b & c & x_t \\ d & e & f & y_t \\ g & h & i & z_t \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} ax + by + cz + x_t \\ dx + ey + fz + y_t \\ gx + hy + iz + z_t \\ 1 \end{pmatrix}$$

Transformations

- This together gives us a **very good toolset** to transform our geometry as we wish.

Linear transformations

Translation column

Affine transformation

$$\begin{pmatrix} a & b & c & x_t \\ d & e & f & y_t \\ g & h & i & z_t \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} ax + by + cz + x_t \\ dx + ey + fz + y_t \\ gx + hy + iz + z_t \\ 1 \end{pmatrix}$$

Transformations

- This together gives us a **very good toolset** to transform our geometry as we wish.

Linear transformations

Translation column

Affine transformation

$$\begin{pmatrix} a & b & c & x_t \\ d & e & f & y_t \\ g & h & i & z_t \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} ax + by + cz + x_t \\ dx + ey + fz + y_t \\ gx + hy + iz + z_t \\ 1 \end{pmatrix}$$

Used for perspective projection...

Transformations

- This together gives us a **very good toolset** to transform our geometry as we wish.

Linear transformations

Translation column

Affine transformation

$$\begin{pmatrix} a & b & c & x_t \\ d & e & f & y_t \\ g & h & i & z_t \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} ax + by + cz + x_t \\ dx + ey + fz + y_t \\ gx + hy + iz + z_t \\ 1 \end{pmatrix}$$

Used for perspective projection...

Must be 1

Multiple Transformations

- How can we apply multiple transformations?

$$A \cdot (B \cdot (C \cdot v))$$

- Is it the same as?

$$B \cdot (A \cdot (C \cdot v))$$



Transformations

- In some graphics libraries you assign the **position / translation, rotation and scale** individually.

Transformations

- In some graphics libraries you assign the **position / translation, rotation and scale** individually.

```
object.position.set(2.7, 1.2, 0);  
object.scale.set(2.4, 0.1, 0.4);  
object.rotation.set(0, toRad(180), 0);
```


Transformations

- In some graphics libraries you assign the **position / translation, rotation and scale** individually.
- To the GPU the object transformations are sent as a matrix (*model matrix*).

Transformations

- In some graphics libraries you assign the **position / translation, rotation and scale** individually.
- To the GPU the object transformations are sent as a matrix (*model matrix*).

projectionMatrix · viewMatrix · **modelMatrix** · *v*

$$P \cdot V \cdot \mathbf{M} \cdot v$$

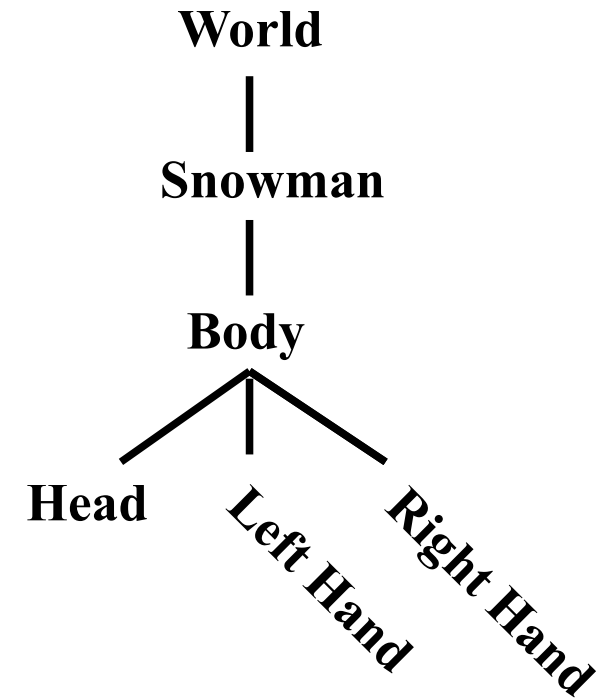
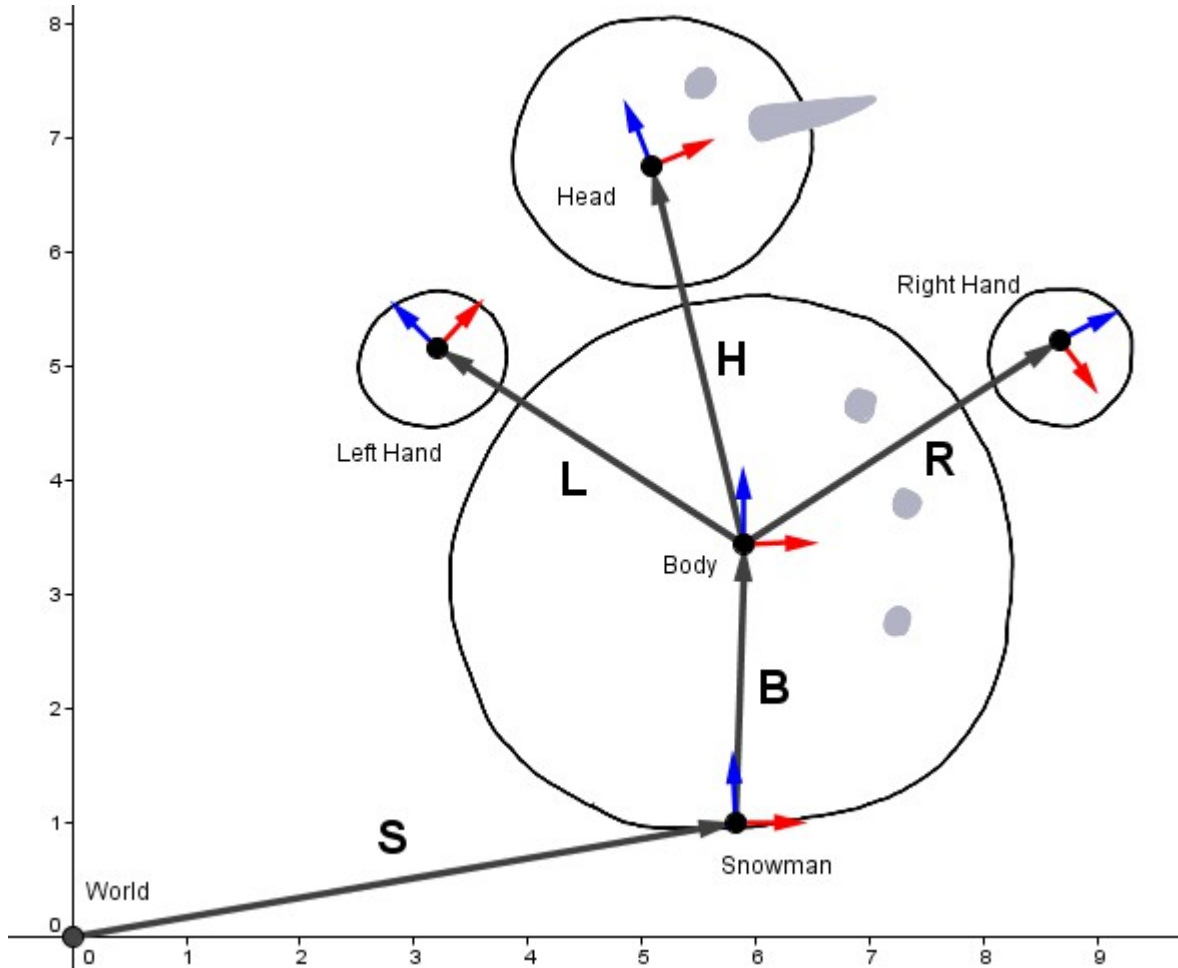
Transformations

- In some graphics libraries you assign the **position / translation, rotation and scale** individually.
- To the GPU the object transformations are sent as a matrix (*model matrix*).
- **Questions about transformations?**

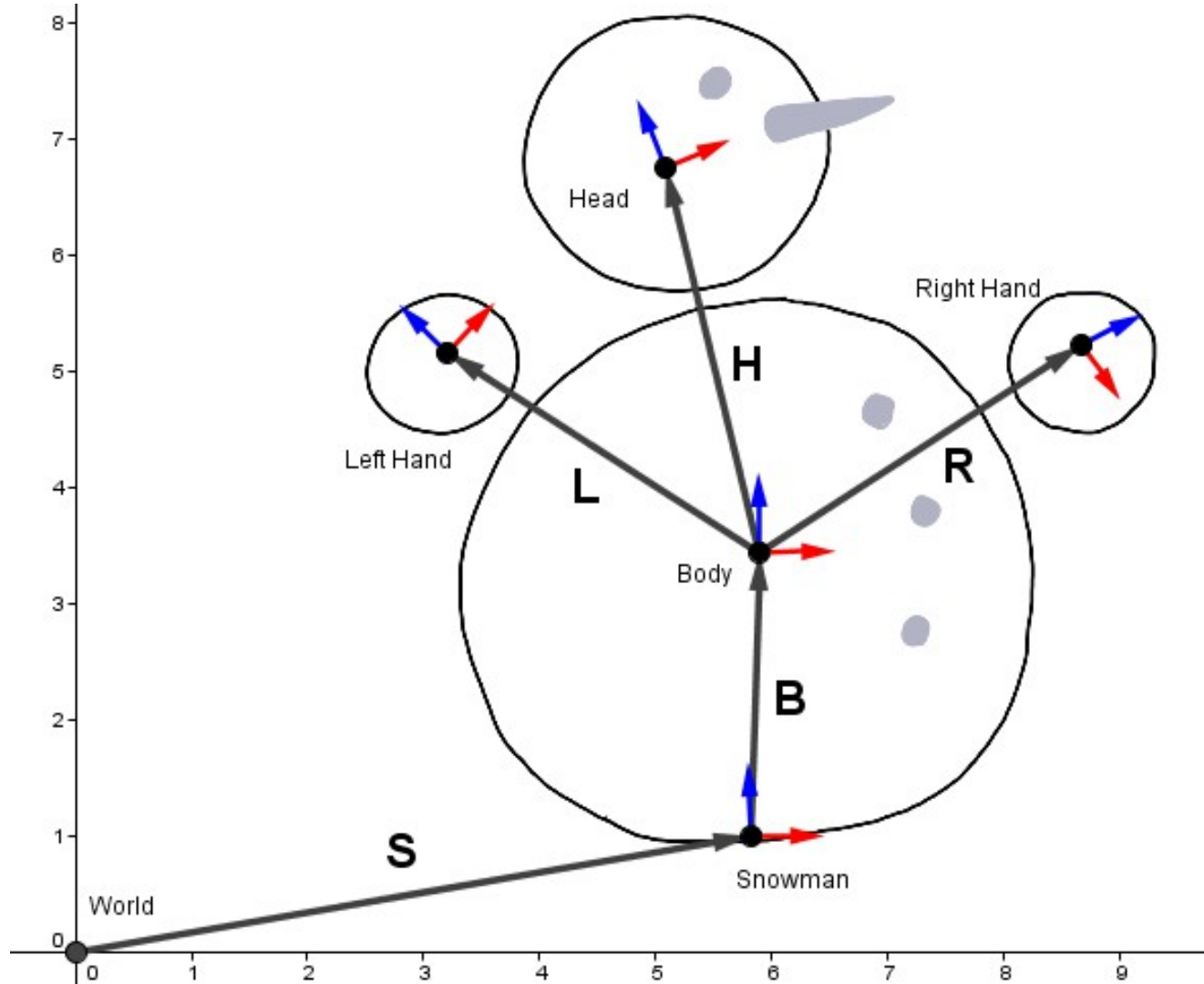


Scene Graph

- Dependency between (parts of) objects.



Scene Graph



Body

$$S \cdot B \cdot v$$

Head

$$S \cdot B \cdot H \cdot v$$

Left hand

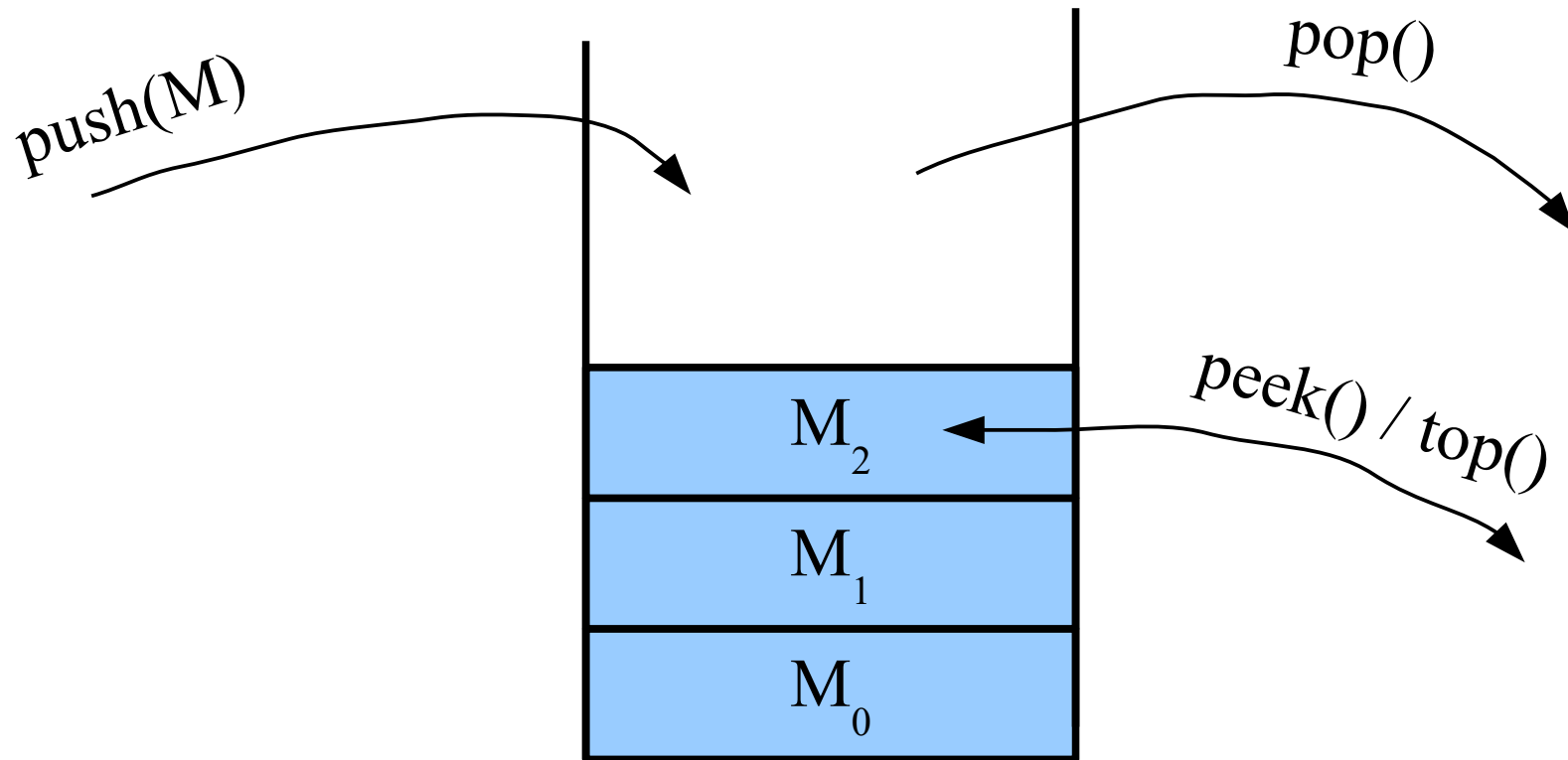
$$S \cdot B \cdot L \cdot v$$

Right hand

$$S \cdot B \cdot R \cdot v$$

Matrix Stack

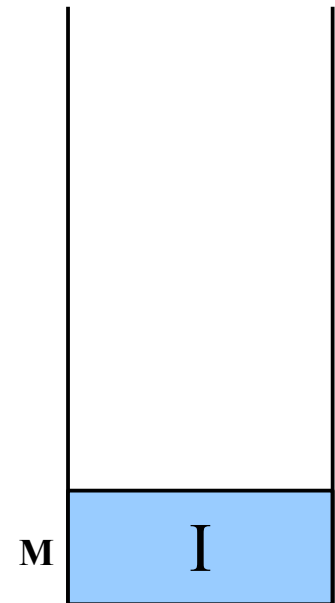
- Stack can be used to save and load matrices (intermediary states)



Matrix Stack

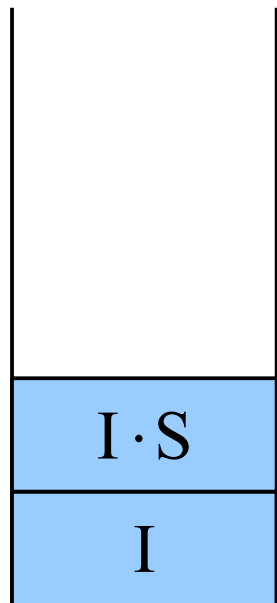
- Stack can be used to save and load matrices (intermediary states)
- *Current state* is in the **top of the stack**

1) $M = \text{Identity}$, $\text{push}(M)$



Matrix Stack

- Stack can be used to save and load matrices (intermediary states)
- *Current state* is in the **top of the stack**

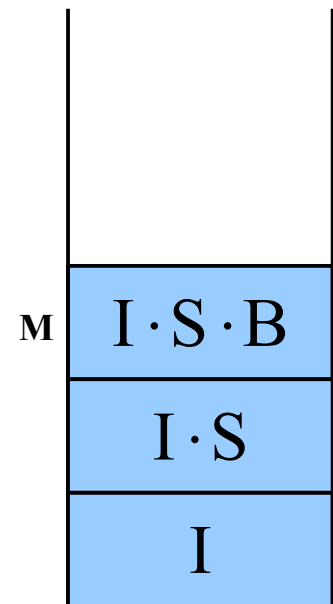


1) $M = \text{Identity}$, $\text{push}(M)$

2) $M \neq S$, $\text{push}(M)$ **Move to Snowman's space**

Matrix Stack

- Stack can be used to save and load matrices (intermediary states)
- Current state* is in the **top of the stack**



1) $M = \text{Identity}$, $\text{push}(M)$

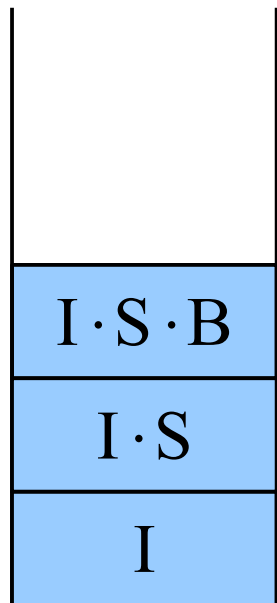
2) $M *= S$, $\text{push}(M)$

3) $M *= B$, $\text{push}(M)$

Move to Body's space

Matrix Stack

- Stack can be used to save and load matrices (intermediary states)
- *Current state* is in the **top of the stack**



1) $M = \text{Identity}$, $\text{push}(M)$

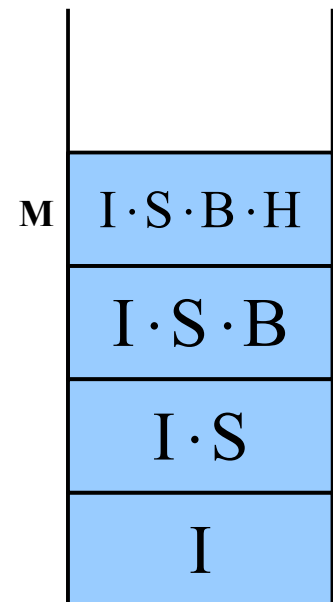
2) $M *= S$, $\text{push}(M)$

3) $M *= H$, $\text{push}(M)$

4) *Draw Body vertices*

Matrix Stack

- Stack can be used to save and load matrices (intermediary states)
- Current state* is in the **top of the stack**



1) $M = \text{Identity}$, $\text{push}(M)$

2) $M *= S$, $\text{push}(M)$

3) $M *= B$, $\text{push}(M)$

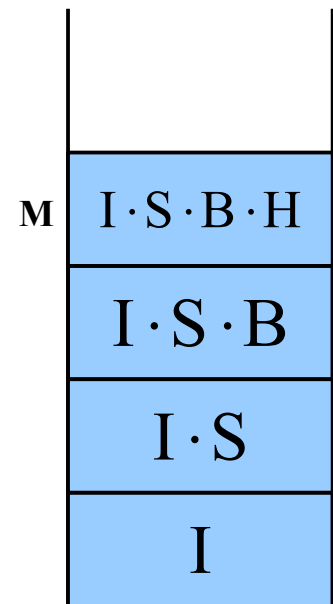
4) *Draw Body vertices*

5) $M *= H$, $\text{push}(M)$

Move to Head's space

Matrix Stack

- Stack can be used to save and load matrices (intermediary states)
- Current state* is in the **top of the stack**



2) $M *= S$, push(M)

3) $M *= B$, push(M)

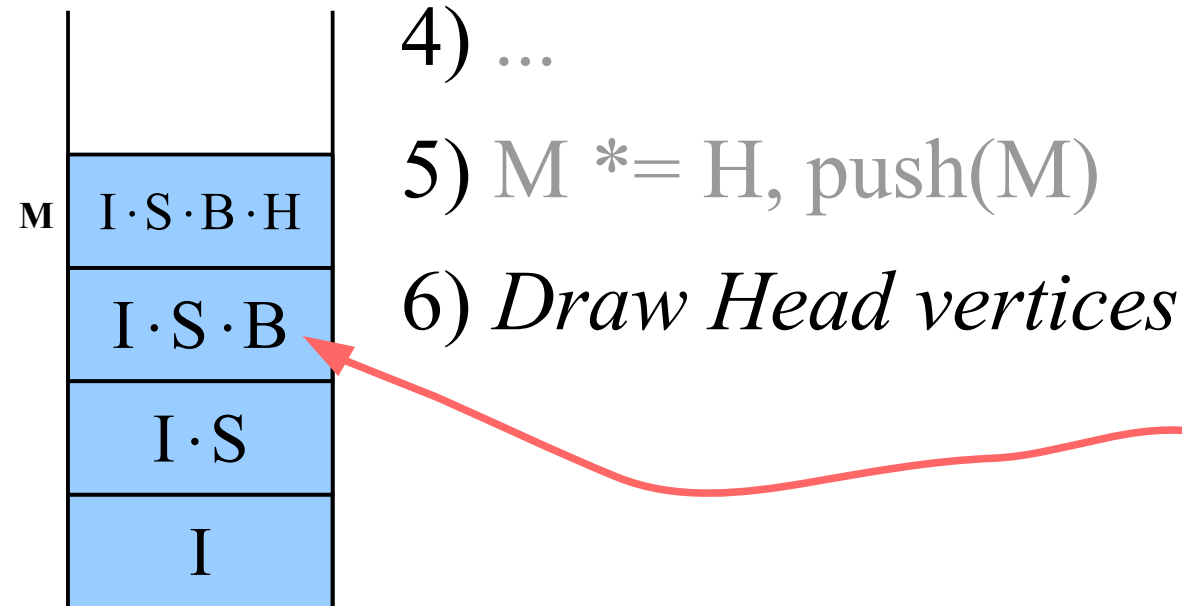
4) *Draw Body vertices*

5) $M *= H$, push(M)

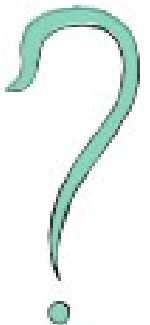
6) *Draw Head vertices*

Matrix Stack

- Stack can be used to save and load matrices (intermediary states)
- *Current state* is in the **top of the stack**

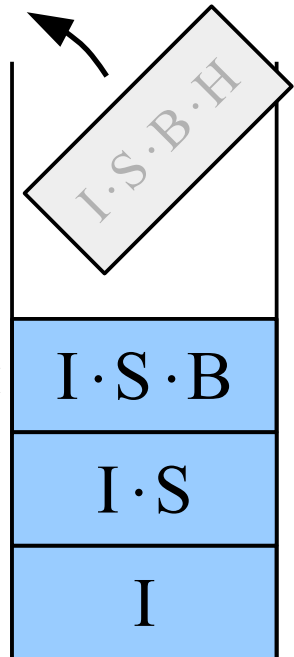


We now want to get back to the
Body's space



Matrix Stack

- Stack can be used to save and load matrices (intermediary states)
- Current state* is in the **top of the stack**



4) ...

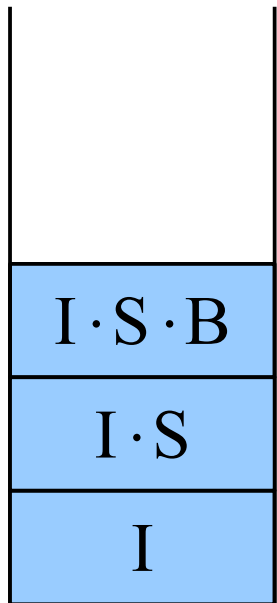
5) $M \ast = H$, `push(M)`

6) *Draw Head vertices*

7) `pop()`, $M = \text{top}()$

Matrix Stack

- Stack can be used to save and load matrices (intermediary states)
- *Current state* is in the **top of the stack**



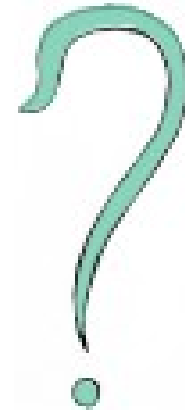
4) ...

5) $M *= H$, $\text{push}(M)$

6) *Draw Head vertices*

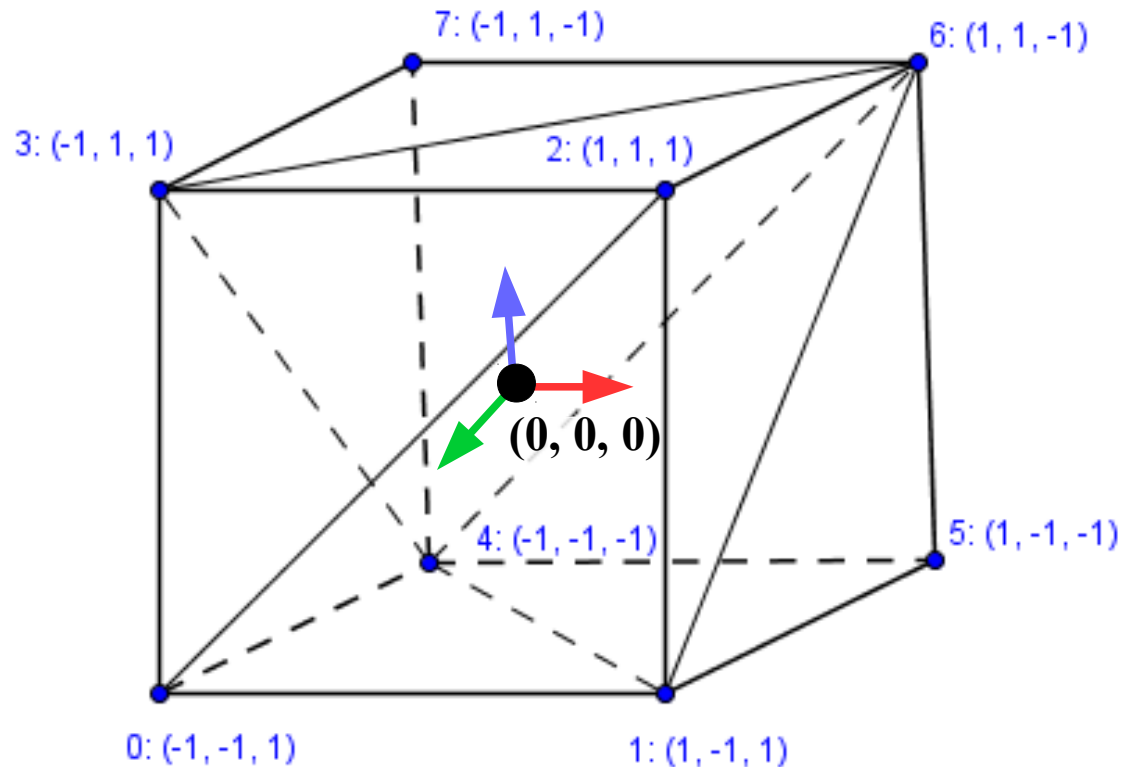
7) $\text{pop}()$, $M = \text{top}()$

8) ...?



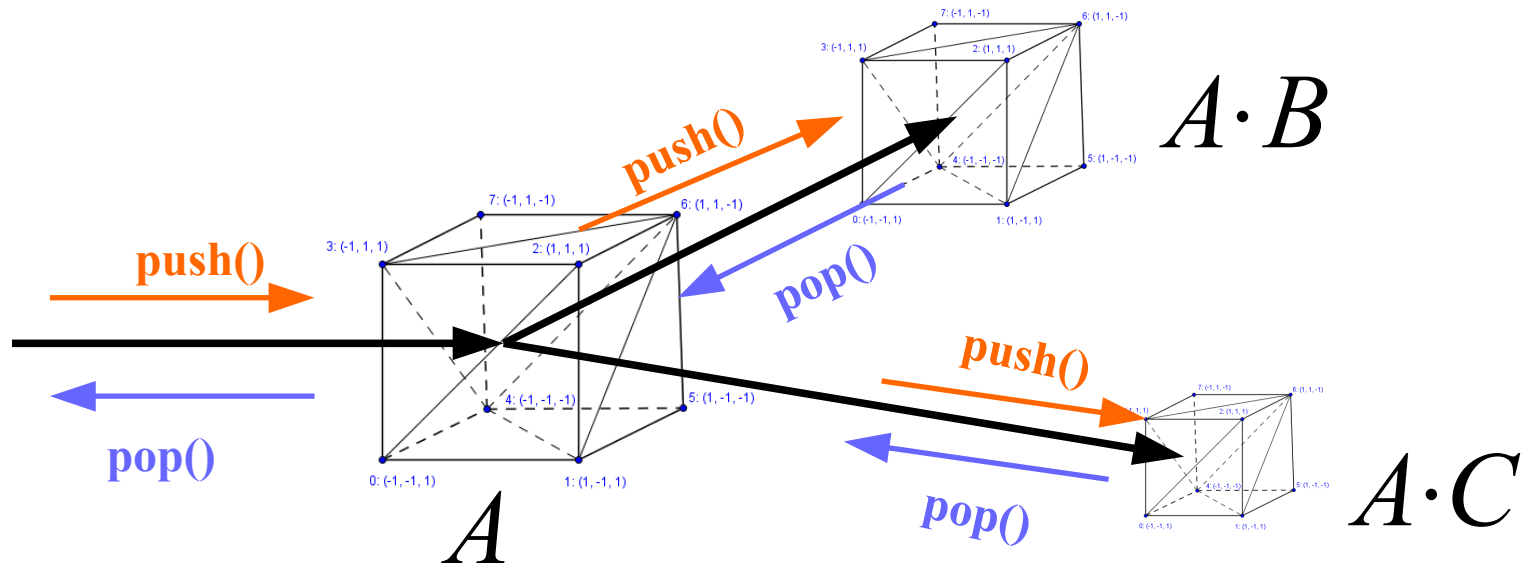
Matrix Stack

- Each (part of an) **object** is modelled in its own **local coordinate space**.



Matrix Stack

- Each (part of an) **object** is modelled in its own **local coordinate space**.
- When we traverse the scene graph, important intermediary states are saved / loaded.



Matrix Stack

- Each (part of an) **object** is modelled in its own **local coordinate space**.
- When we traverse the scene graph, important intermediary states can be saved / loaded.
- No need to recalculate same matrix multiplications many times or find inverse transformations.

$$M = A \cdot B \cdot D \cdot D^{-1} = A \cdot B$$

vs

$$\text{stack.pop()}, \quad M = \text{stack.top}()$$

Matrix Stack

- Each (part of an) **object** is modelled in its own **local coordinate space**.
- When we traverse the scene graph, important intermediary states can be saved / loaded.
- No need to recalculate same matrix multiplications many times or find inverse transformations.
- Questions about the matrix stack?



What new did you find out today?

What more would you like to know?

Next time:

Frames of reference, projections