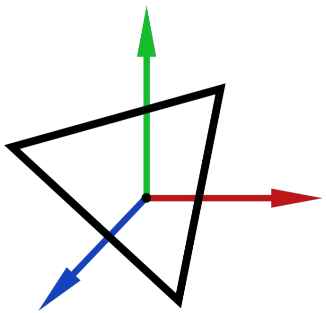


Computer Graphics Seminar

MTAT.03.305

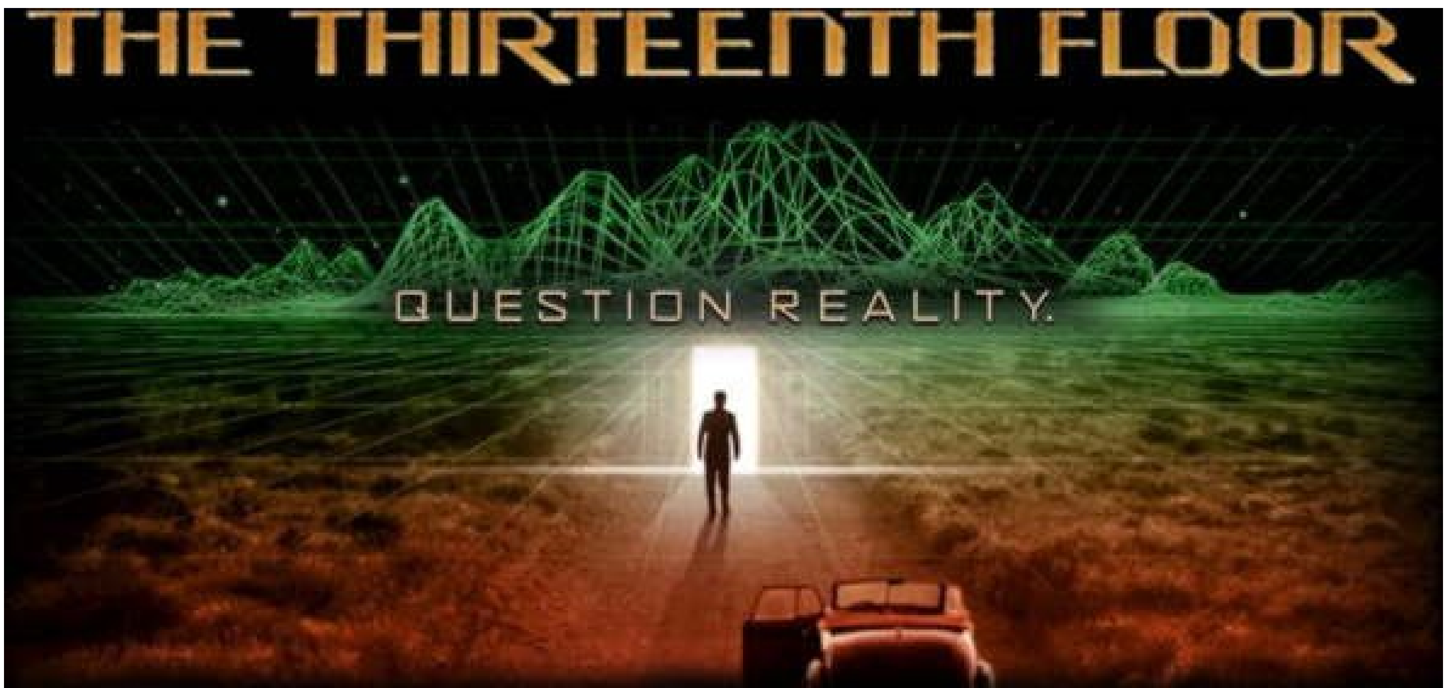
Fall 2020

Raimond Tunnel

UNIVERSITY OF TARTU
Institute of Computer Science

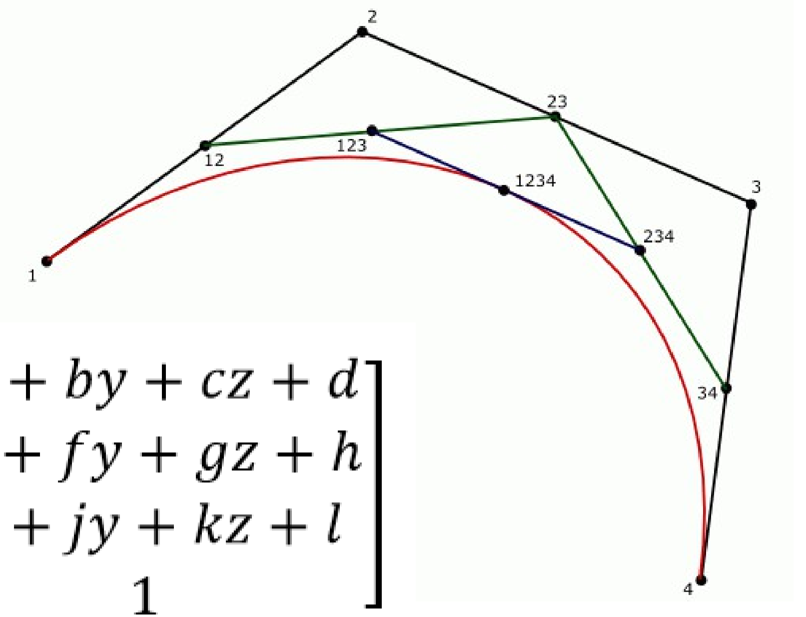
Computer Graphics

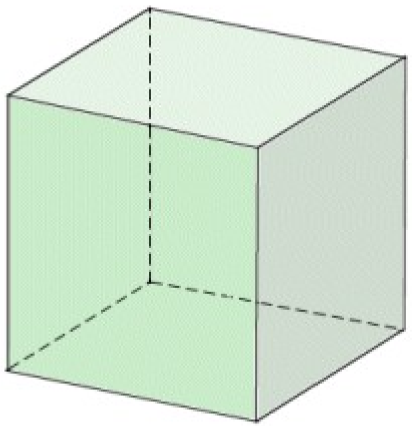
- Graphical illusion via the computer
- Displaying something meaningful (incl art)



Math

- Computers are good at... computing.
- To do computer graphics, we need math for the computer to compute.
- Geometry, algebra, calculus.

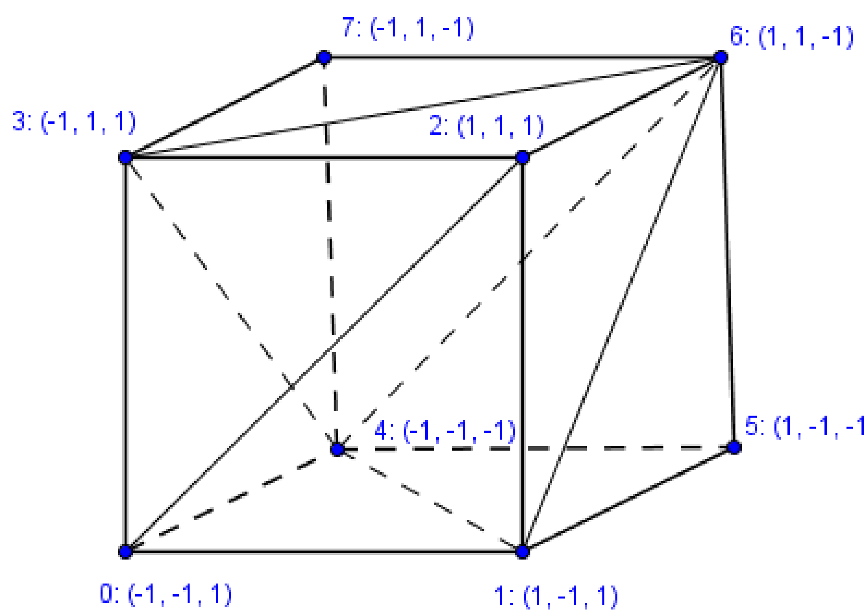




$$\begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} ax + by + cz + d \\ ex + fy + gz + h \\ ix + jy + kz + l \\ 1 \end{bmatrix}$$

Math

- For creating and manipulating 3D objects we use:
  - **Analytic geometry** – math about coordinate systems
  - **Linear algebra** – math about vectors and spaces

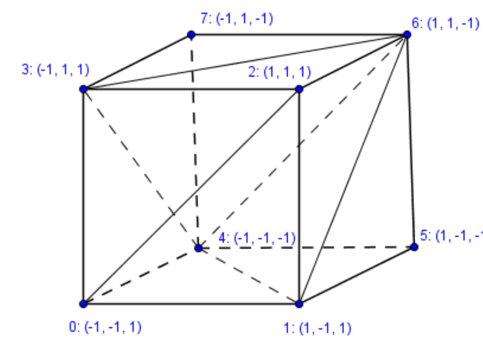


Skills for Computer Graphics

- Mathematical understanding

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} ax + by \\ cx + dy \end{pmatrix}$$

- Geometrical (spatial) thinking



- Programming

```
GLuint vaoHandle;
glGenVertexArrays(1, &vaoHandle);
glBindVertexArray(vaoHandle);
```

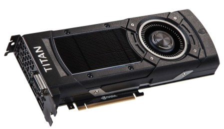
- Visual creativity & aesthetics

The Standard Graphics Pipeline

Data

Define geometry and transformations

OpenGL



Vertex transformations

Apply geometry and transformations

Vertex shader

Culling & Clipping

Rasterization

Fragment shading

Visibility tests & Blending

Point

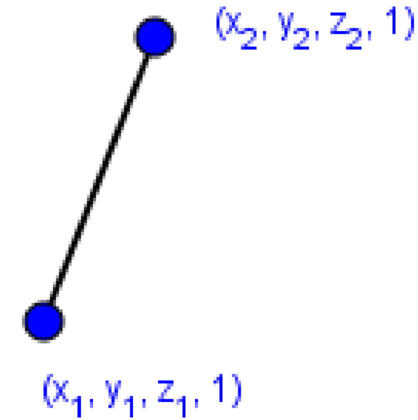
- Simplest geometry primitive
- In homogeneous coordinates:

$$(x, y, z, w),\ w \neq 0$$

- Represents a point $(x/w, y/w, z/w)$
- Usually you can put **w = 1 for points**
- Actual division will be done by GPU later

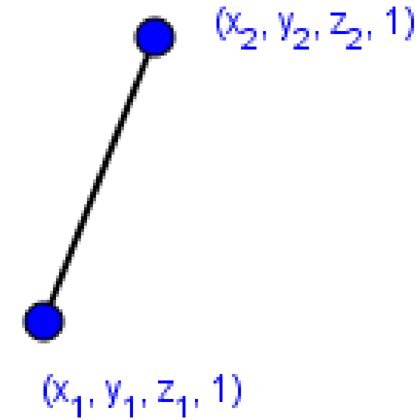
Line (segment)

- Consists of:
  - 2 endpoints
  - *Infinite* number of points between
- Defined by the endpoints
- Interpolated and rasterized in the GPU



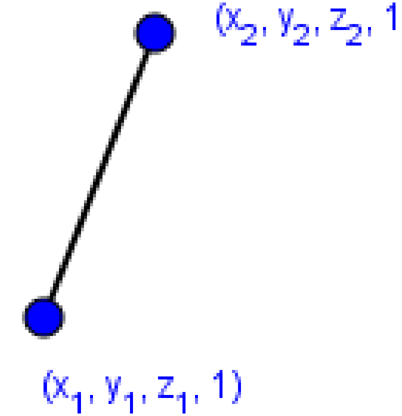
Line (segment)

- Consists of:
  - 2 endpoints
  - *Infinite* number of points between
- Defined by the endpoints
- Interpolated and rasterized in the GPU

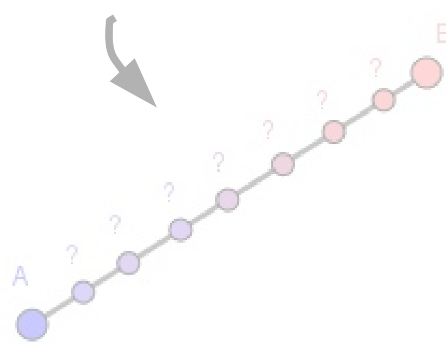


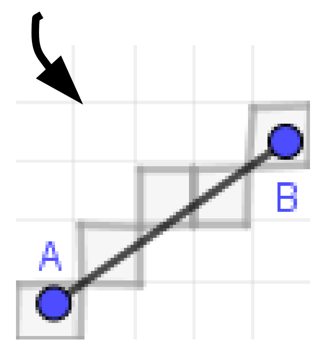
Line (segment)

- Consists of:
  - 2 endpoints
  - *Infinite* number of points between



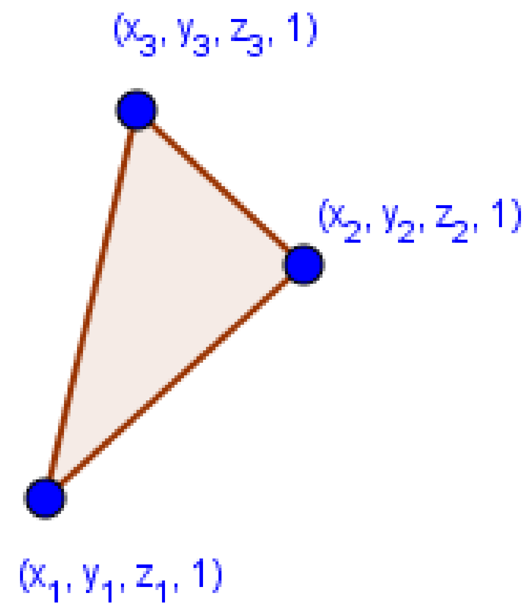
- Defined by the endpoints
- Interpolated and rasterized in the GPU





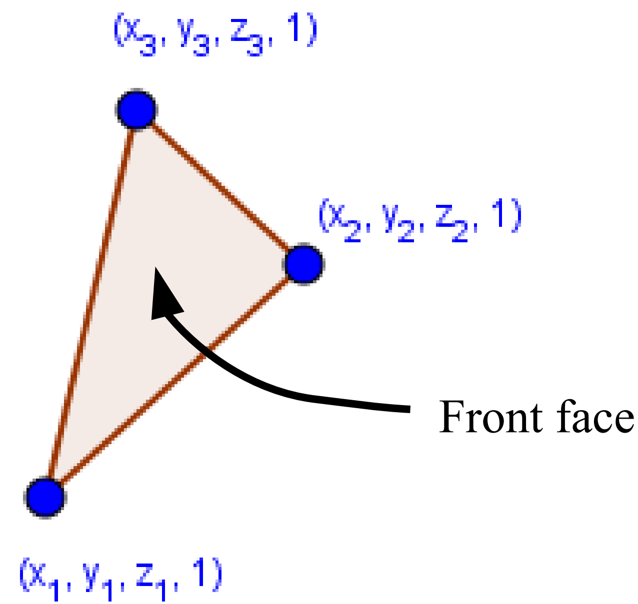
Triangle

- Consists of:
  - 3 points called vertices
  - 3 lines called edges
  - 1 face
- Defined by 3 vertices
- Face interpolated and rasterized in the GPU
- Counter-clockwise order defines the **front face**



Triangle

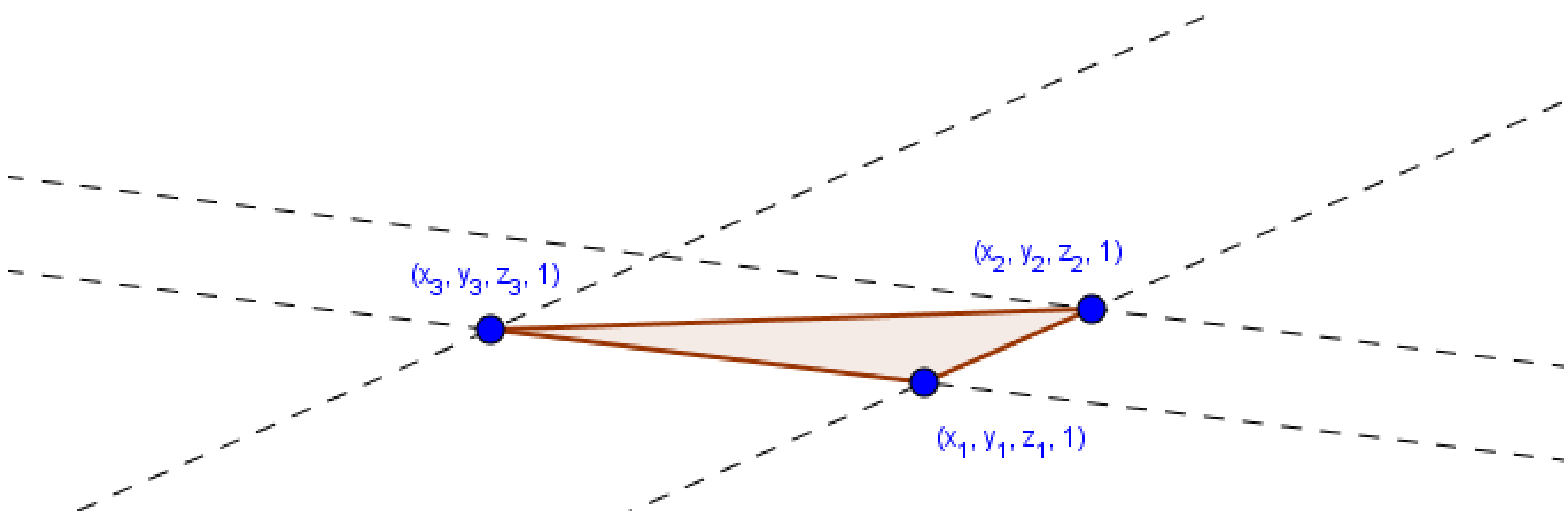
- Consists of:
  - 3 points called vertices
  - 3 lines called edges
  - 1 face



- Defined by 3 vertices
- Face interpolated and rasterized in the GPU
- Counter-clockwise order defines the **front face**

Why triangles?

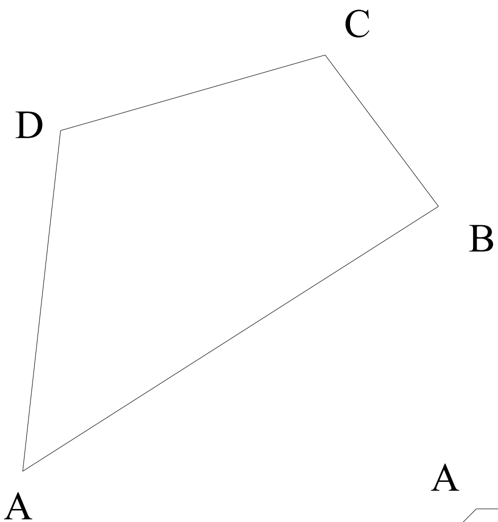
- They are in many ways the simplest polygons
  - 3 different points always form a plane
  - Easy to rasterize (fill the face with pixels)
  - Every other polygon can be converted to triangles

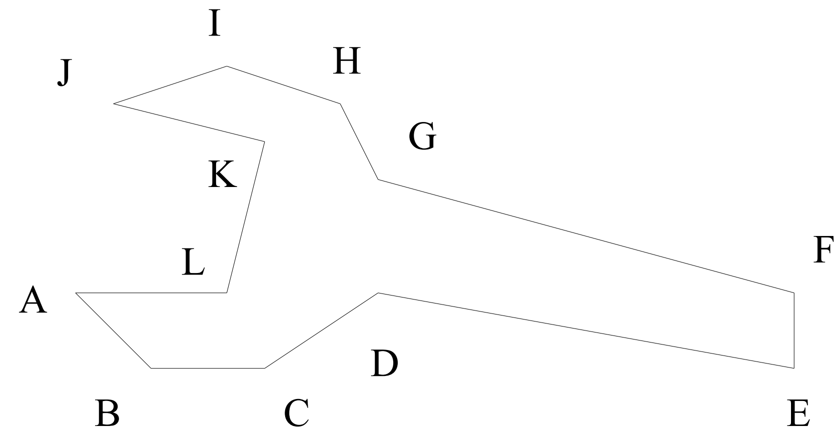


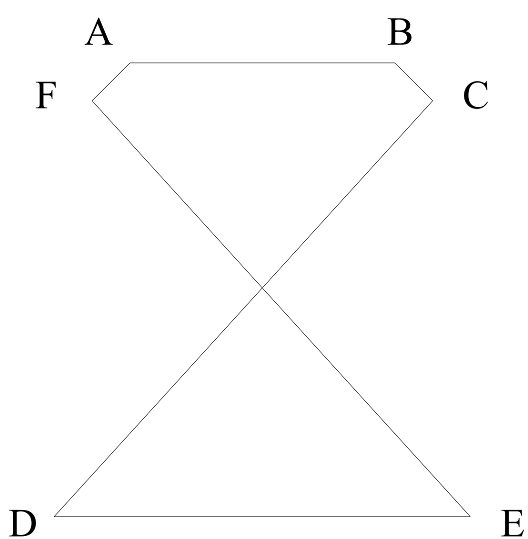
Why triangles?

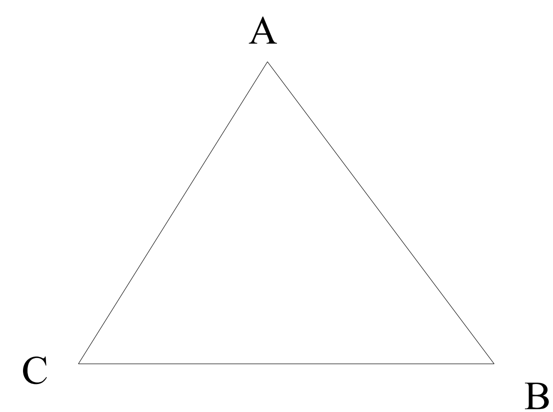
- They are in many ways the simplest polygons
  - 3 different points always form a plane
  - Easy to rasterize (fill the face with pixels)
  - Every other polygon can be converted to triangles
- OpenGL used to support other polygons too
  - Must have been:
    - **Simple** – No edges intersect each other
    - **Convex** – All points between any two inner points are inner points

Examples of polygons









OpenGL < 3.1 primitives

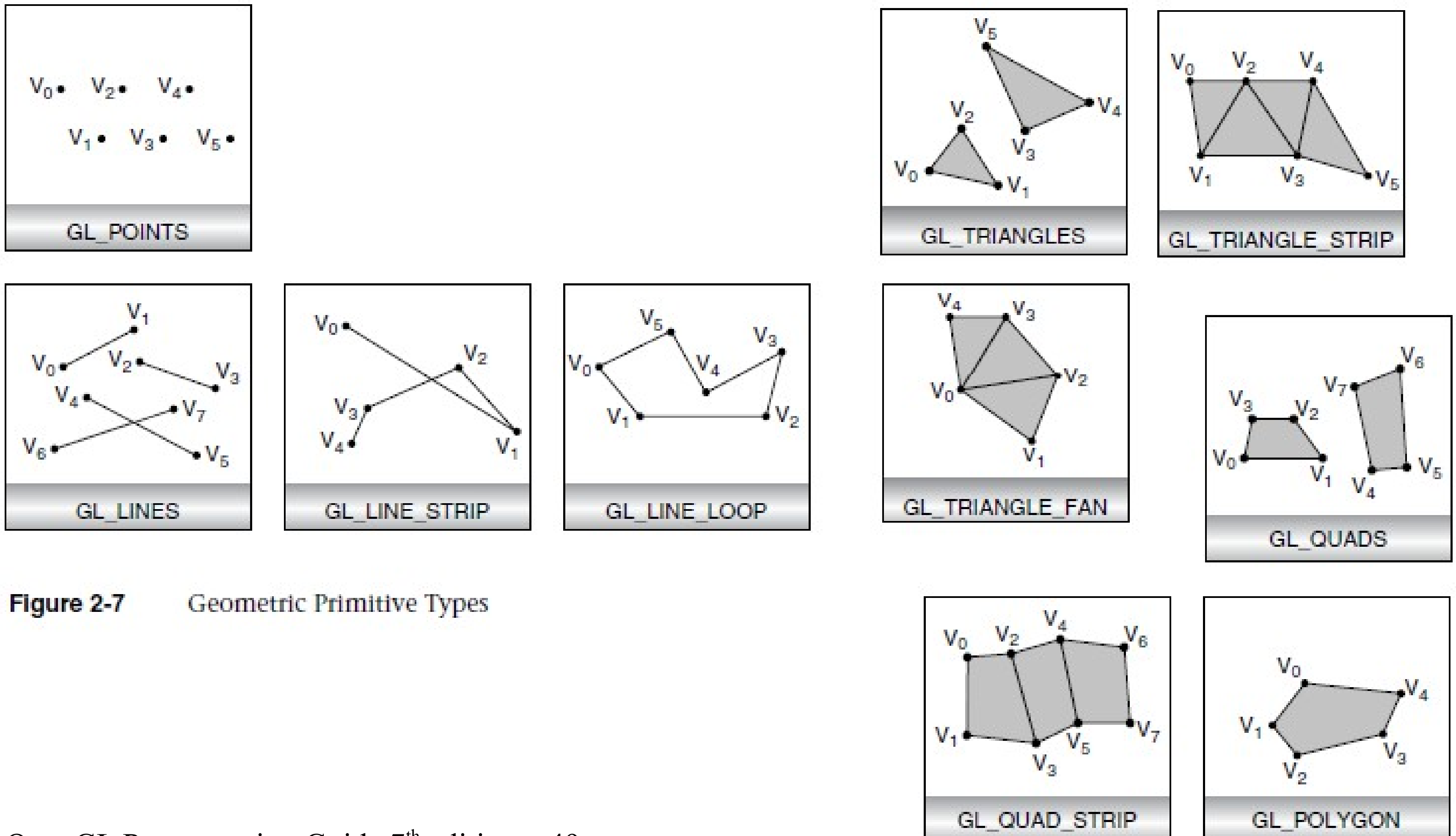


Figure 2-7 Geometric Primitive Types

OpenGL Programming Guide 7th edition, p49

After OpenGL 3.1

Table 3.1 OpenGL Primitive Mode Tokens

Primitive Type	OpenGL Token
Points	GL_POINTS
Lines	GL_LINES
Line Strips	GL_LINE_STRIP
Line Loops	GL_LINE_LOOP
Independent Triangles	GL_TRIANGLES
Triangle Strips	GL_TRIANGLE_STRIP
Triangle Fans	GL_TRIANGLE_FAN

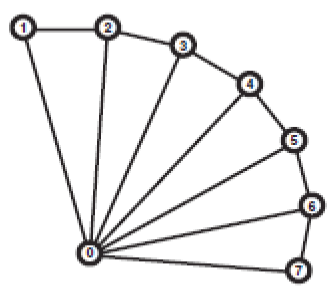


Figure 3.2 Vertex layout for a triangle fan

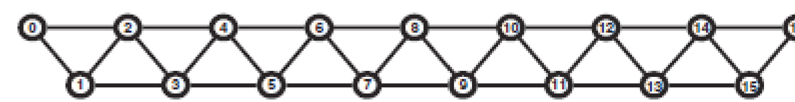
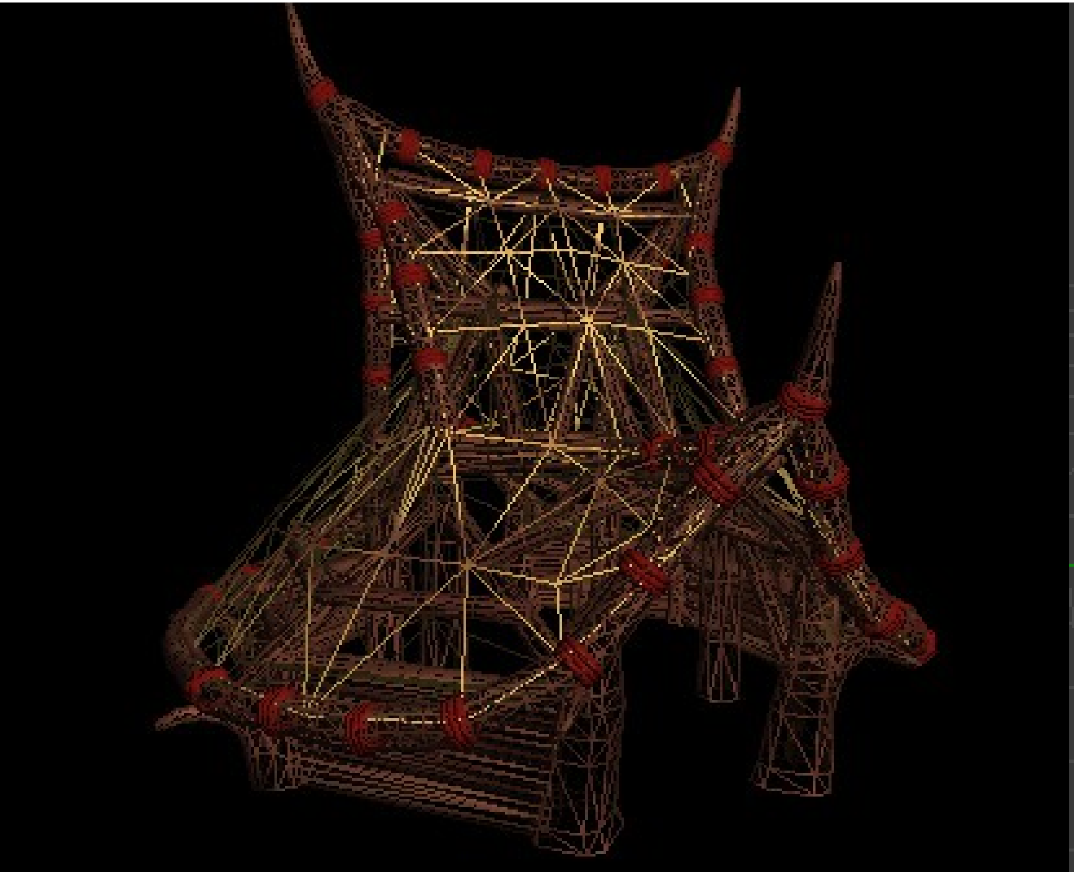


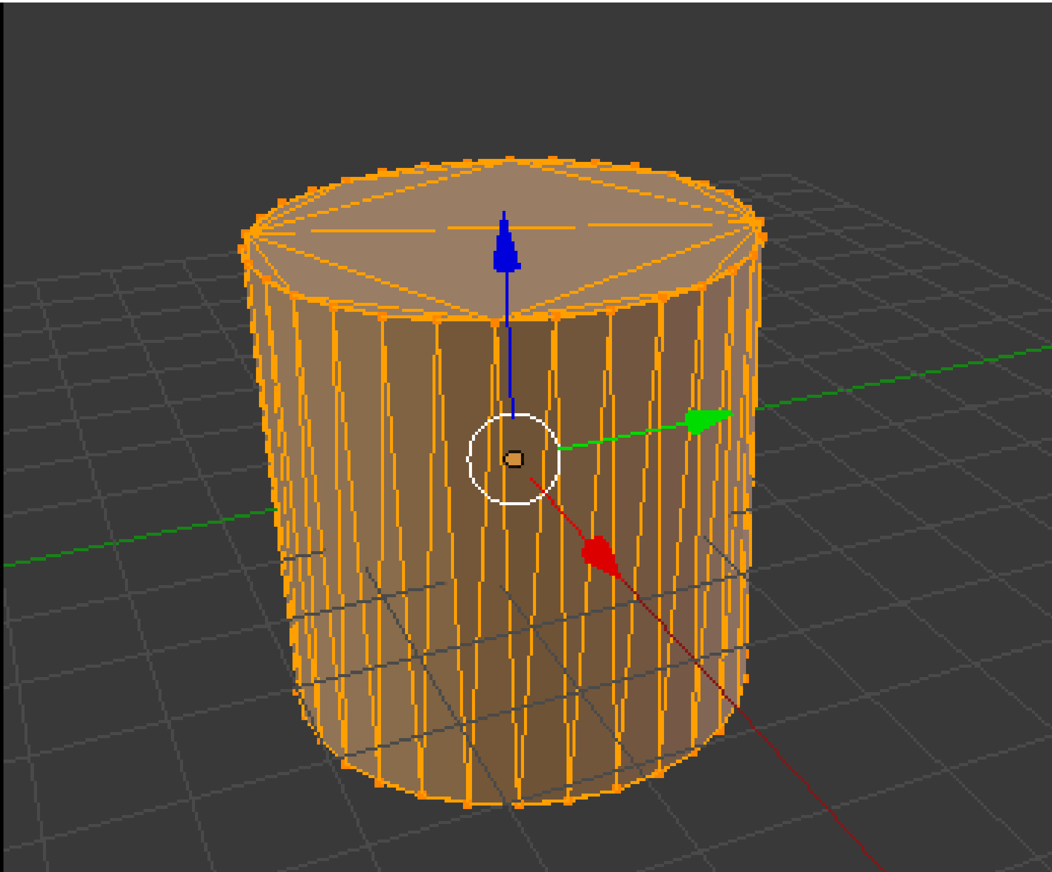
Figure 3.1 Vertex layout for a triangle strip

OpenGL Programming Guide 8th edition, p89-90

In the beginning there were points

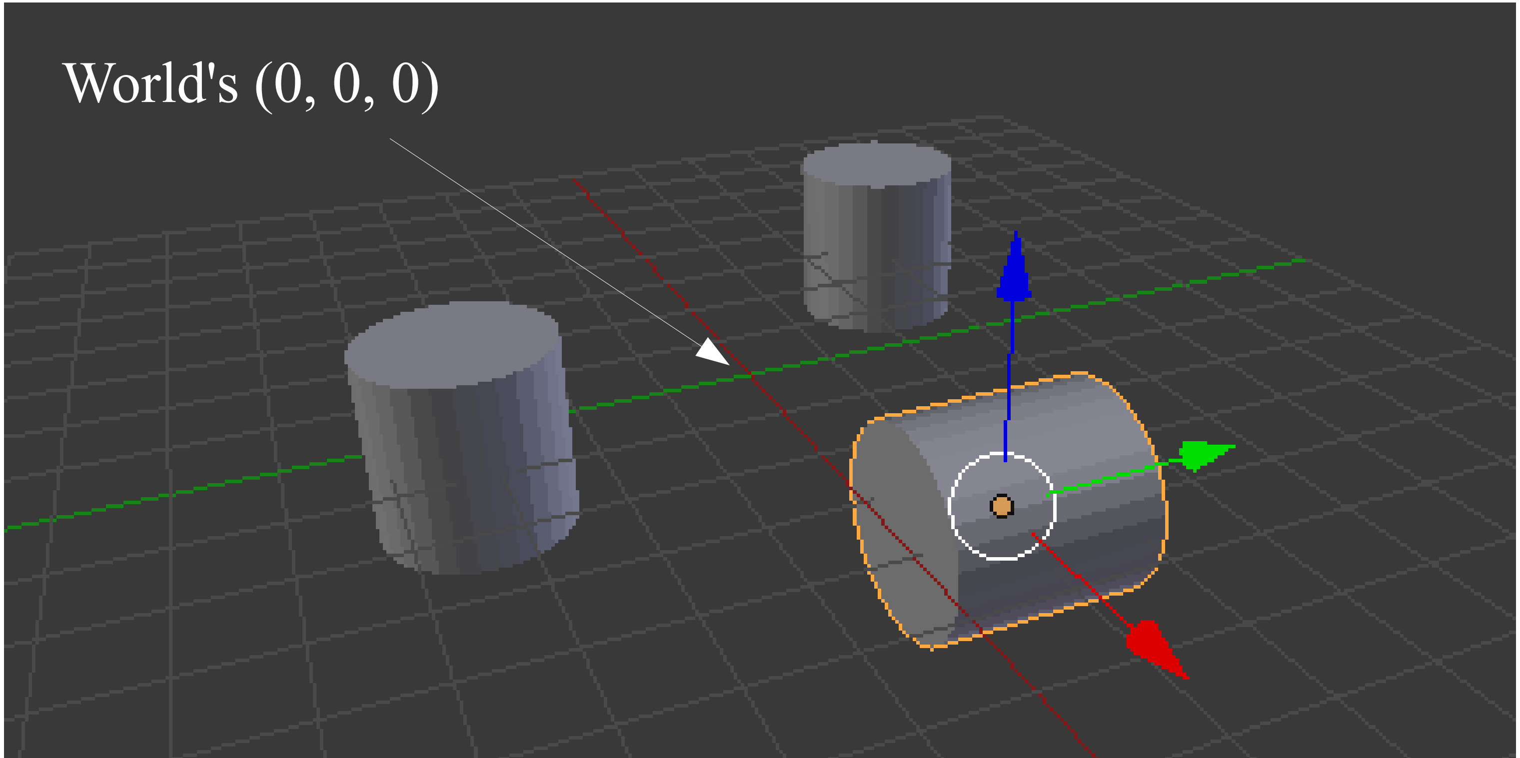
- We can now define our geometric objects!





In the beginning there were points

- We can now define our geometric objects!
- We want to move our objects!



Transformations

- Linear transformations
  - Scaling, reflection
  - Rotation
  - Shearing
- Affine transformations
  - Translation (moving / shifting)
- Projection transformations
  - Perspective
  - Orthographic

Homogeneous coordinates are needed here...

...and for the perspective projection

Transformations

- Every transformation is a function
- As you have learned from algebra, **all linear functions can be represented as matrices**

$$f(v)=\begin{pmatrix} 2\cdot x \\ y \\ z \end{pmatrix}=\begin{pmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}\cdot\begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

$$v \in R^3$$

$$v=\begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

Column-major format

Transformations

- Every transformation is a function
- As you have learned from algebra, **all linear functions can be represented as matrices**

$$f(v)=\begin{pmatrix} 2\cdot x \\ y \\ z \end{pmatrix}=\begin{pmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}\cdot\begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

Linear function, which increases the first coordinate two times.

$$v \in R^3$$

$$v=\begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

Column-major format

Transformations

- Every transformation is a function
- As you have learned from algebra, **all linear functions can be represented as matrices**

$$f(v) = \begin{pmatrix} 2 \cdot x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

Same function as a matrix

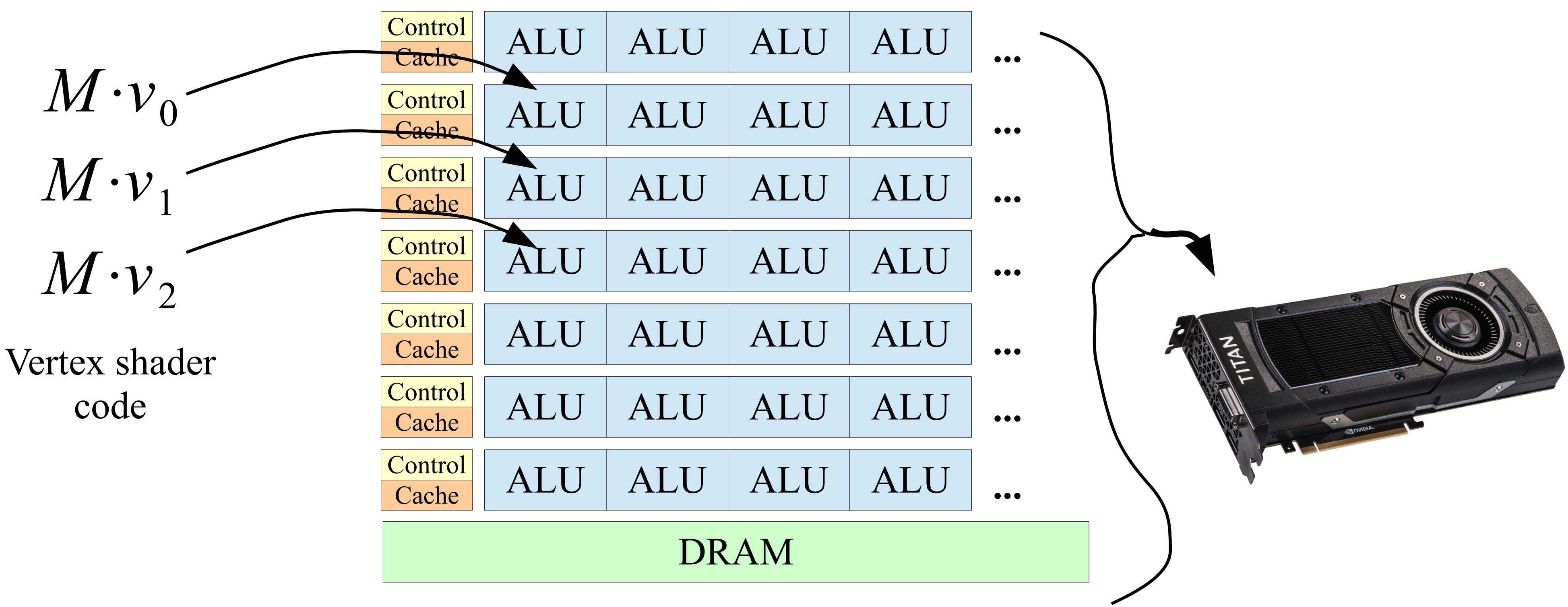
$$v \in R^3$$

$$v = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

Column-major format

Transformations

- GPU-s are built for doing transformations with matrices on points (vertices).



Transformations

- GPU-s are built for doing transformations with matrices on points (vertices).

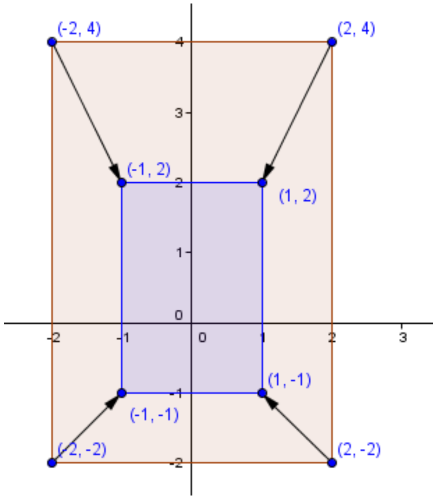
- Linear transformations satisfy:

$$f(a_1 x_1 + \ldots + a_n x_n) = a_1 f(x_1) + \ldots + a_n f(x_n)$$

We will not use homogeneous coordinates at the moment, but they will be back...

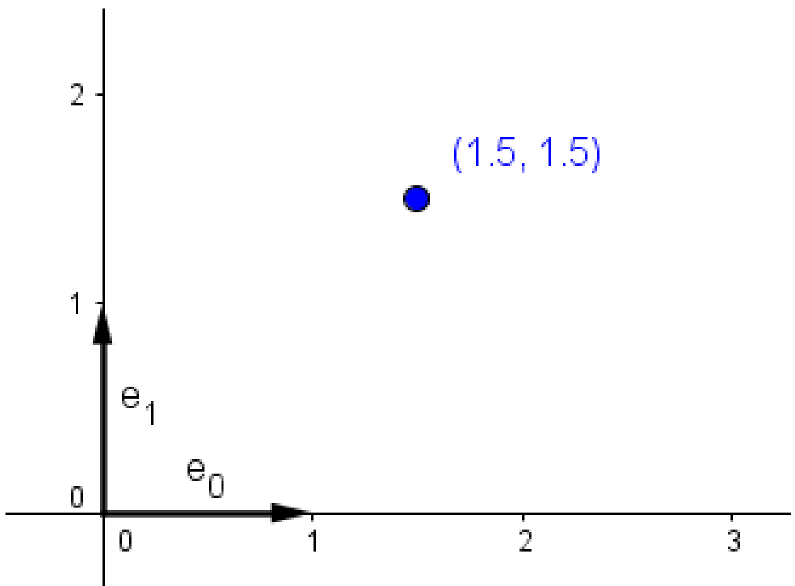
Linear Transformation

Scale

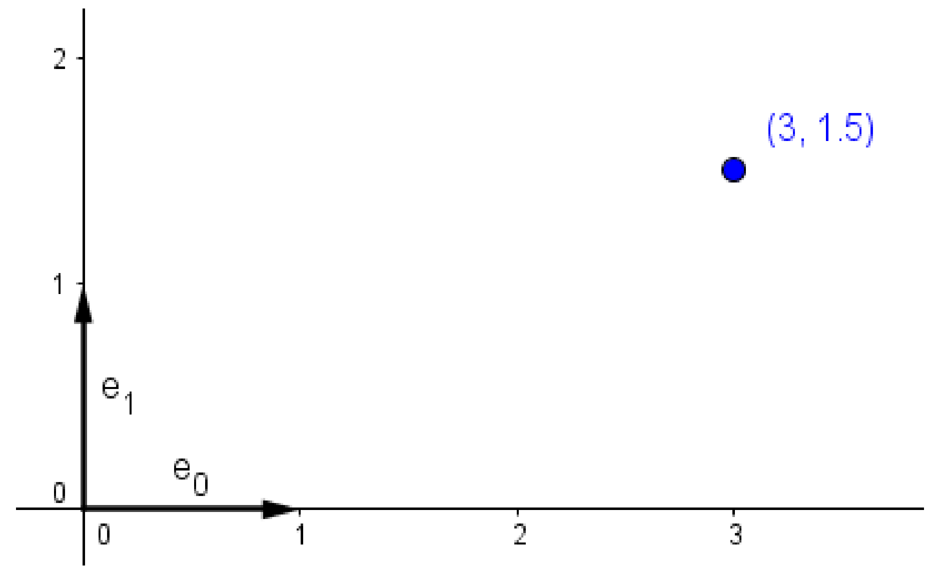


Scaling

- Multiplies the coordinates by a scalar factor.



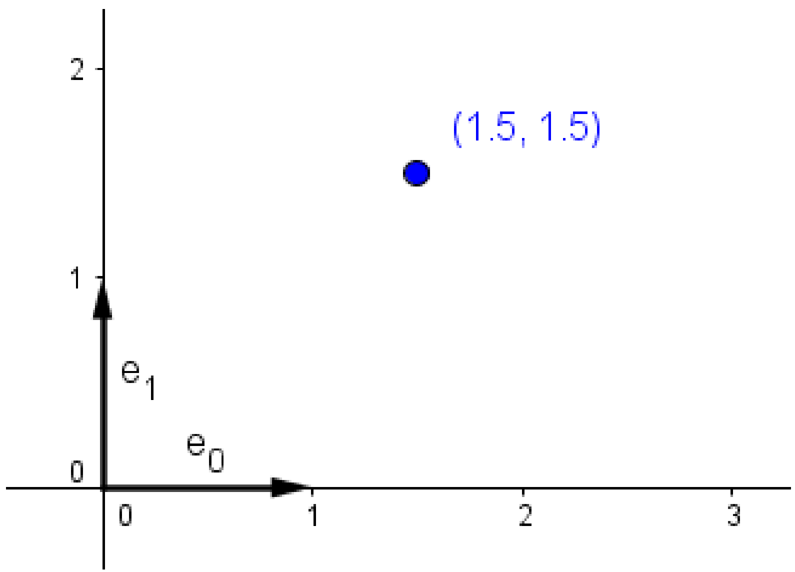
$$\begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix}$$



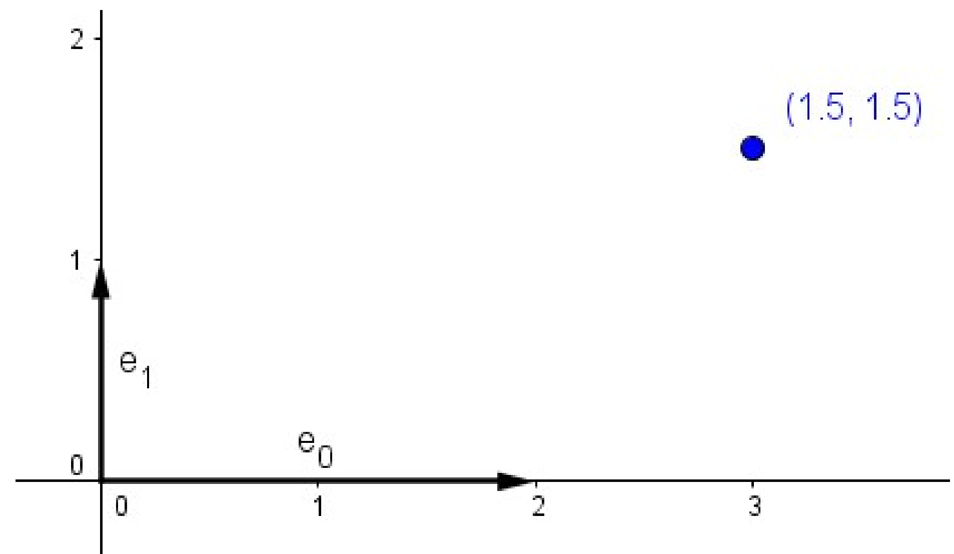
$$\begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1.5 \\ 1.5 \end{pmatrix} = \begin{pmatrix} 3 \\ 1.5 \end{pmatrix}$$

Scaling

- Multiplies the coordinates by a scalar factor.
- Scales the standard basis vectors / axes.



$$\begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 2 \\ 0 \end{pmatrix} = e_0$$



$$\begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} = e_1$$

Scaling

- In general we could scale each axis

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

a_x – x-axis scale factor

a_y – y-axis scale factor

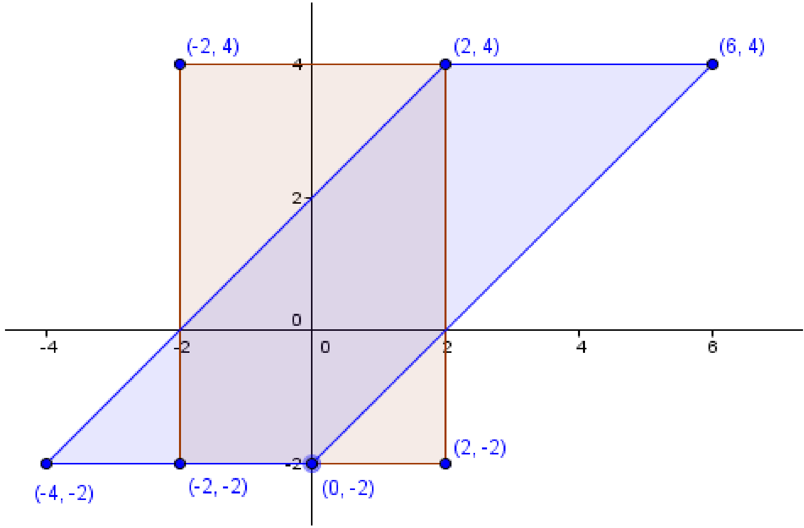
a_z – z-axis scale factor

- If some factor is negative, this matrix will reflect the points from that axis. Thus we get reflection.

What happens to out triangles when an odd number of factors are negative?

Linear Transformation

Shear



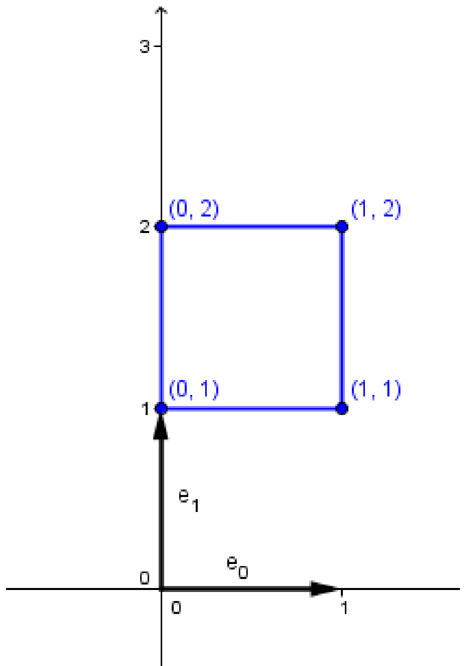
Shearing

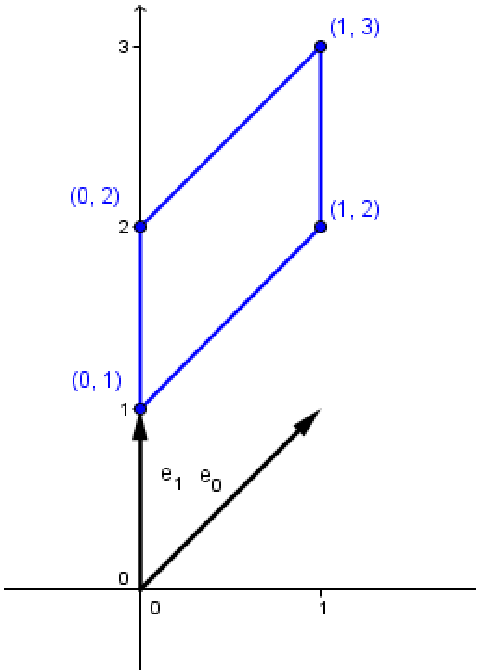
- Remember it for translations later.
- Tilts only one axis.
- Squares become parallelograms.

$$\begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 2 \end{pmatrix} = \begin{pmatrix} 0 \\ 2 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 2 \end{pmatrix} = \begin{pmatrix} 1 \\ 3 \end{pmatrix}$$

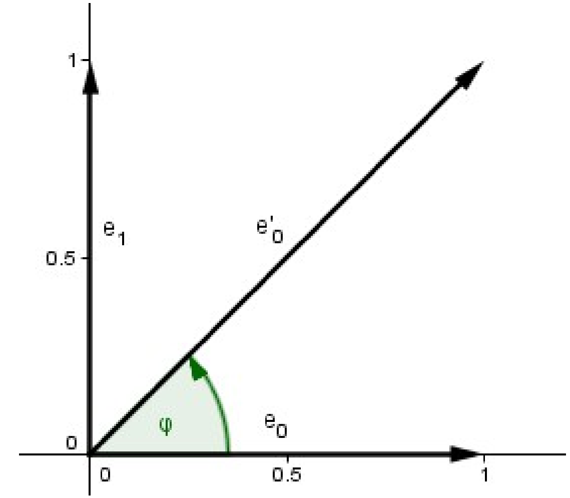




Shearing

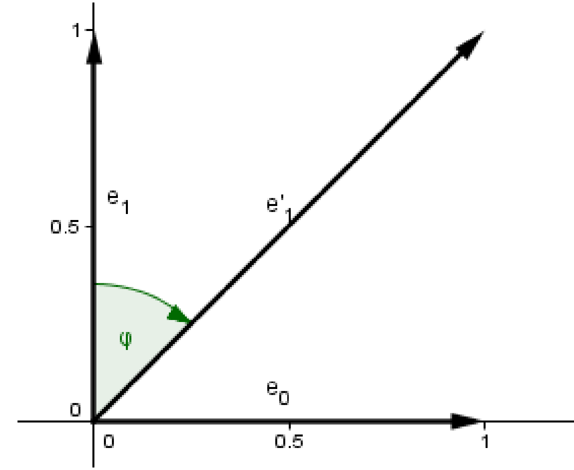
- **Shear-y**, we tilt parallel to y-axis by angle φ counter-clockwise

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



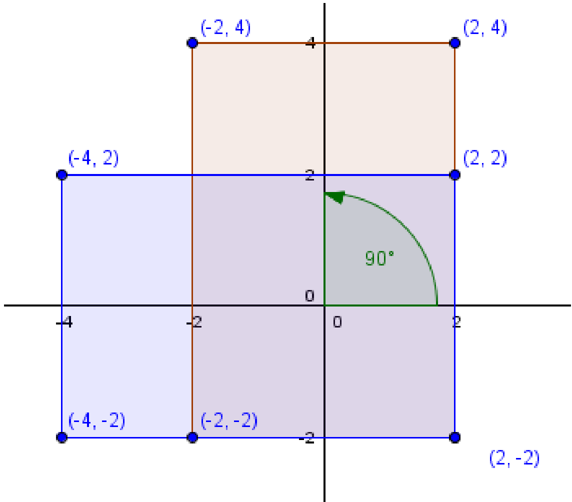
- **Shear-x**, we tilt parallel to x-axis by angle φ clockwise

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



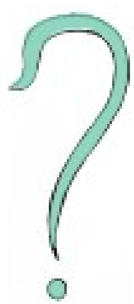
Linear Transformation

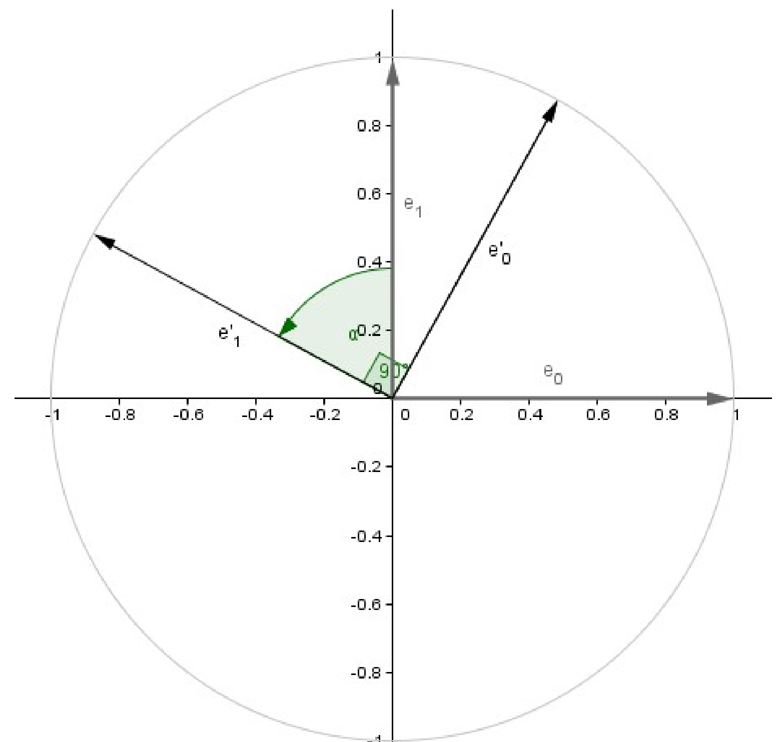
Rotation



Rotation

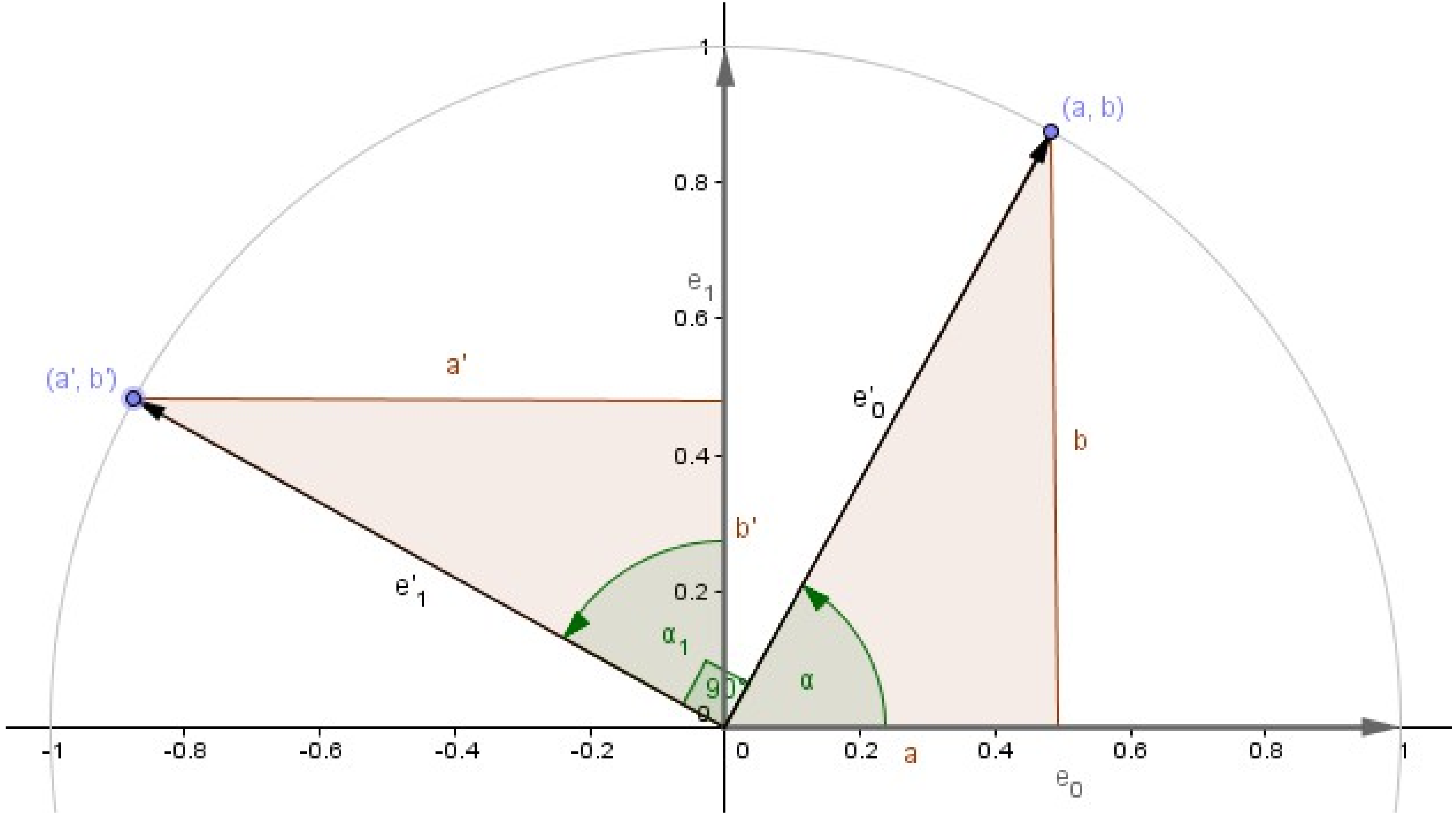
- Shearing moved only one axis
- Also changed the size of the basis vector
- Can we do better?





Did you notice that the columns of the transformation matrix show the coordinates of the new basis vectors?

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

- So if we rotate by α in counter-clockwise order in 2D, the transformation matrix is:

$$\begin{array}{cc} \mathbf{e'}_{0} & \mathbf{e'}_{1} \\ \multicolumn{2}{c}{\begin{pmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{pmatrix}} \end{array}$$

- In 3D we can do rotations in each plane (xy, xz, yz), so there can be 3 different matrices.

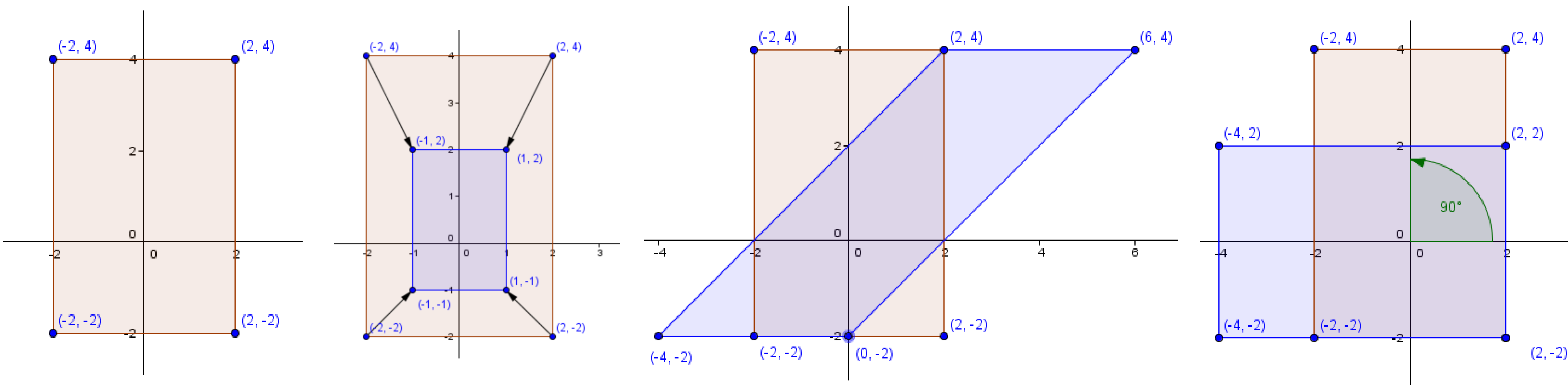
Rotation

- To do a rotation around an arbitrary axis, we can:
  - Rotate that axis to be the x-axis
  - Rotate around the new x-axis $\longleftarrow \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) & 0 \\ 0 & \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$
  - Invert the first rotations (move the old x-axis back)
- OpenGL provides a command for rotating around a given axis.
- Generally quaternions are used for rotations.

Quaternions are elements of a number system that extend the complex numbers...

Do we have everything now?

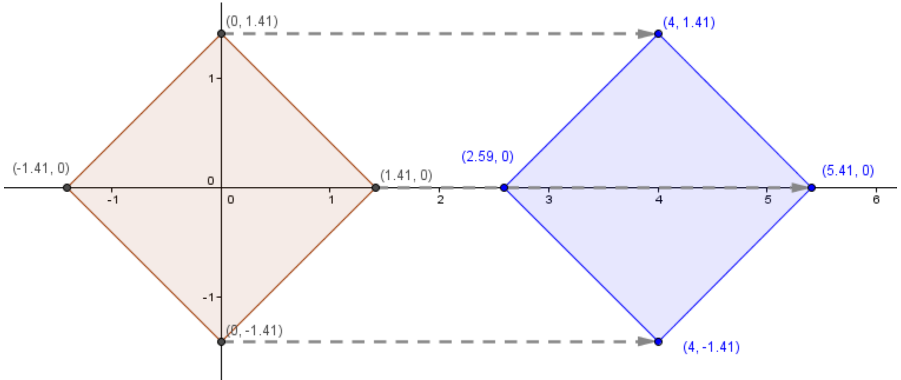
- We can scale, shear and rotate our geometry around the origin...



What if we have an object not centered in the origin?

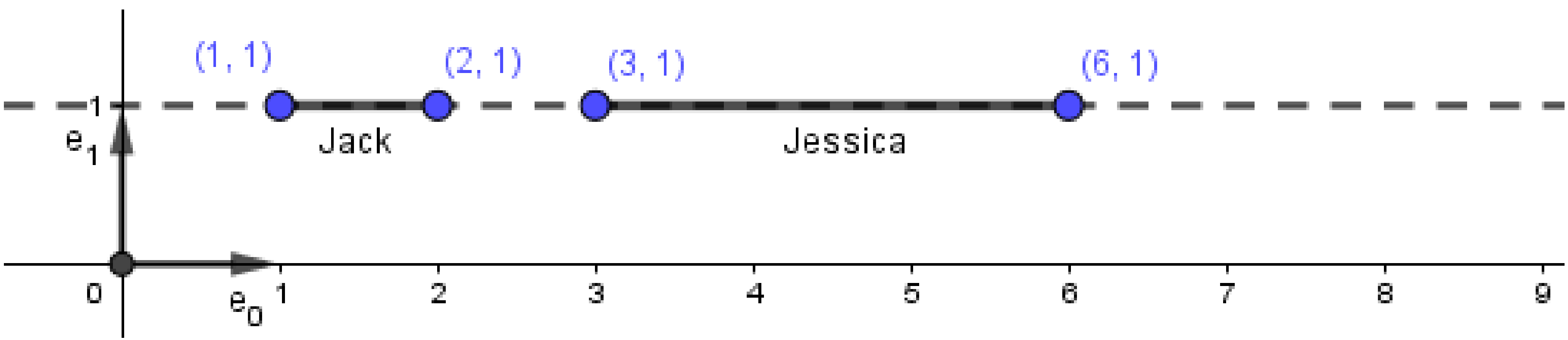
Affine Transformation

Translation



Translation

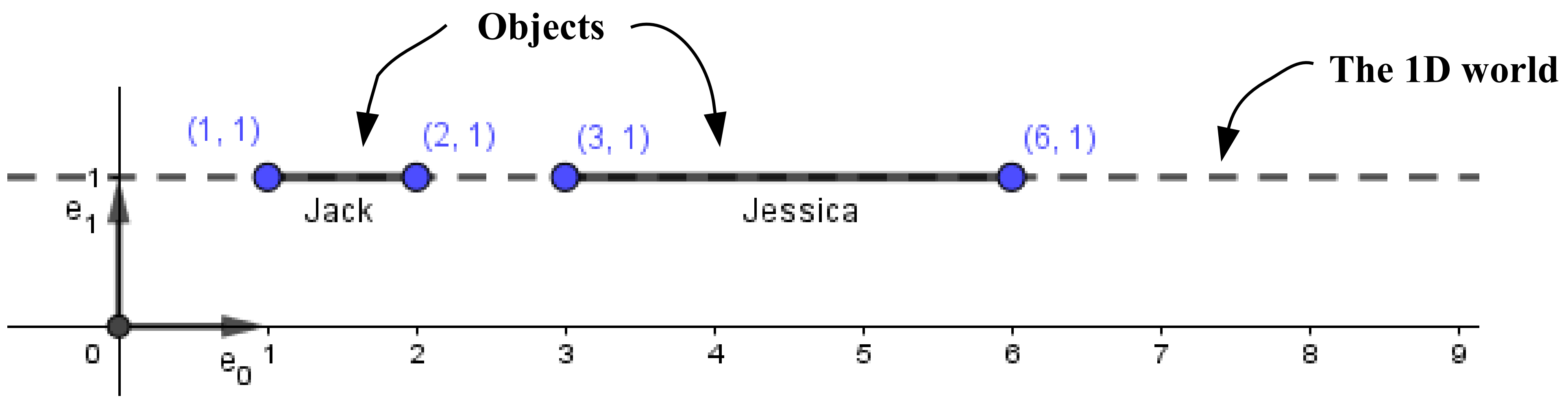
- Imagine that a 1D world is located at y=1 line in 2D space.



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

Translation

- Imagine that a 1D world is located at y=1 line in 2D space.

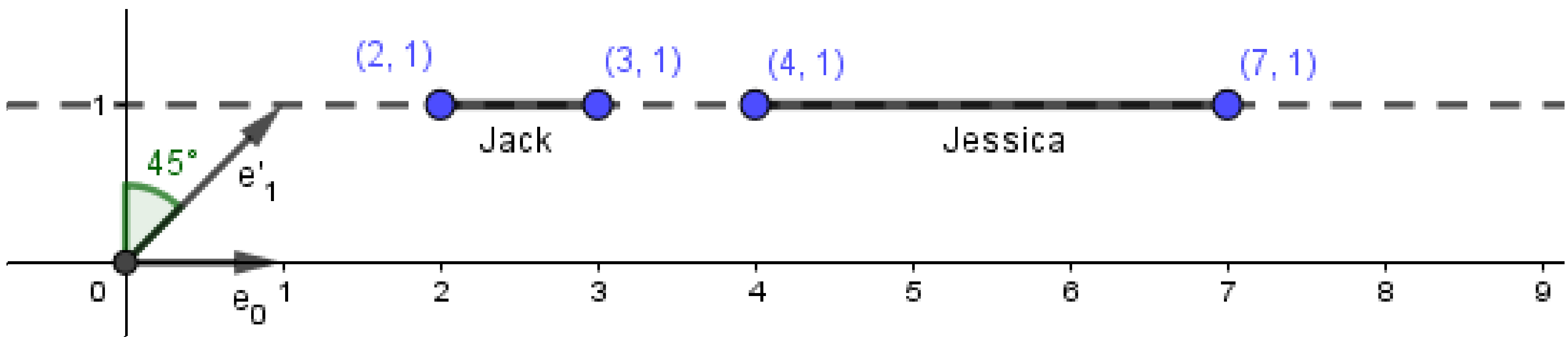


- Notice that all the points are in the form: (x, 1)

Translation

$\tan(45°) = 1$

- Do a shear-x(45°) operation on the 2D world!

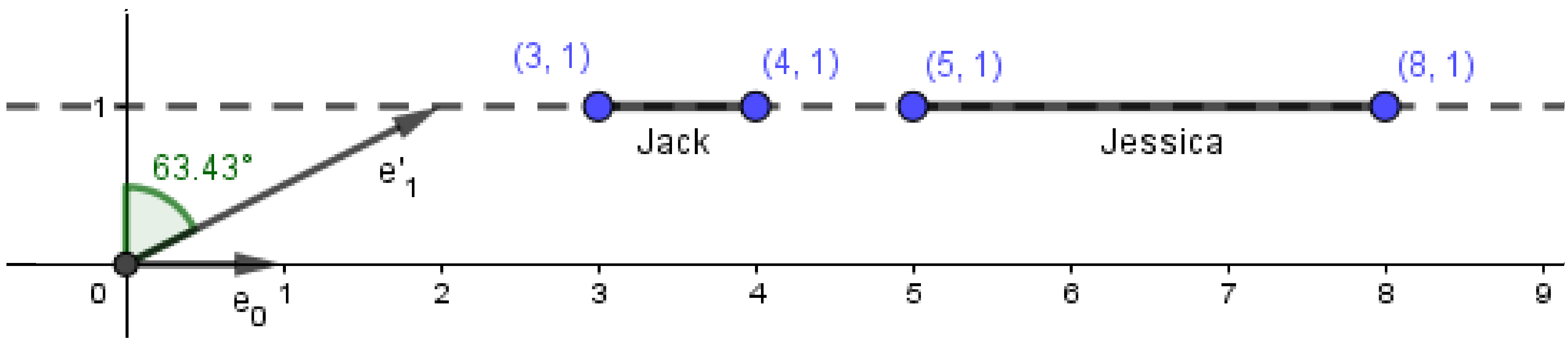


- Everything has now moved 1 unit in x to the right from the original position.

Translation

$\tan(45°) = 1$
$\tan(63.4°) = 2$

- What if we do shear-x(63.4°)?



- We can do translation (movement)!

Translation

- When we represent our points in one dimension higher space, where the extra coordinate is 1, we get to the **homogeneous** space.

$$\begin{pmatrix} 1 & x_t \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ 1 \end{pmatrix} = \begin{pmatrix} x + x_t \\ 1 \end{pmatrix}$$

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

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

Transformations

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

Linear transformations

Translation column

Augmented trasnformation matrix!

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

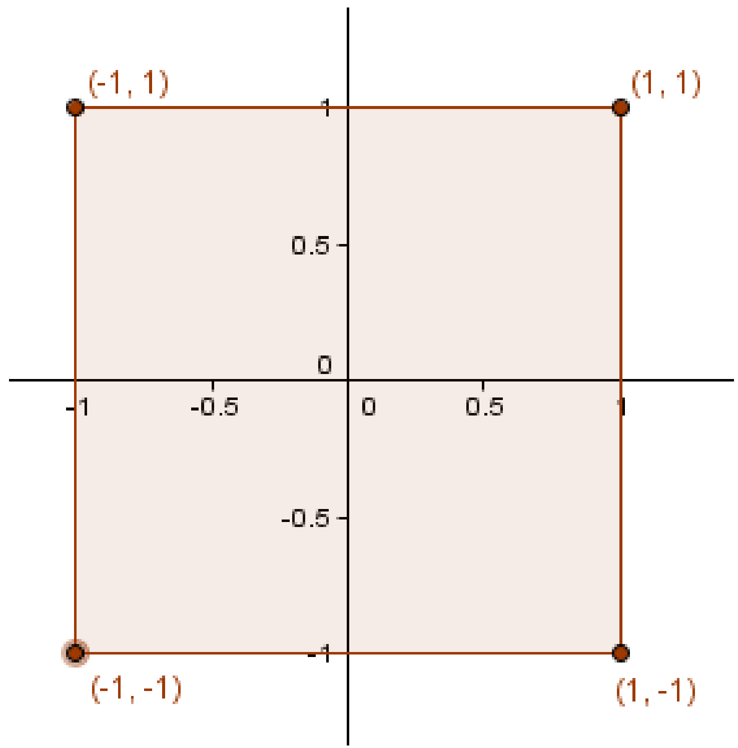
Used for perspective projection...

Multiple transformations

- Everything starts from the origin!
- To apply multiple transformations, just multiply matrices.

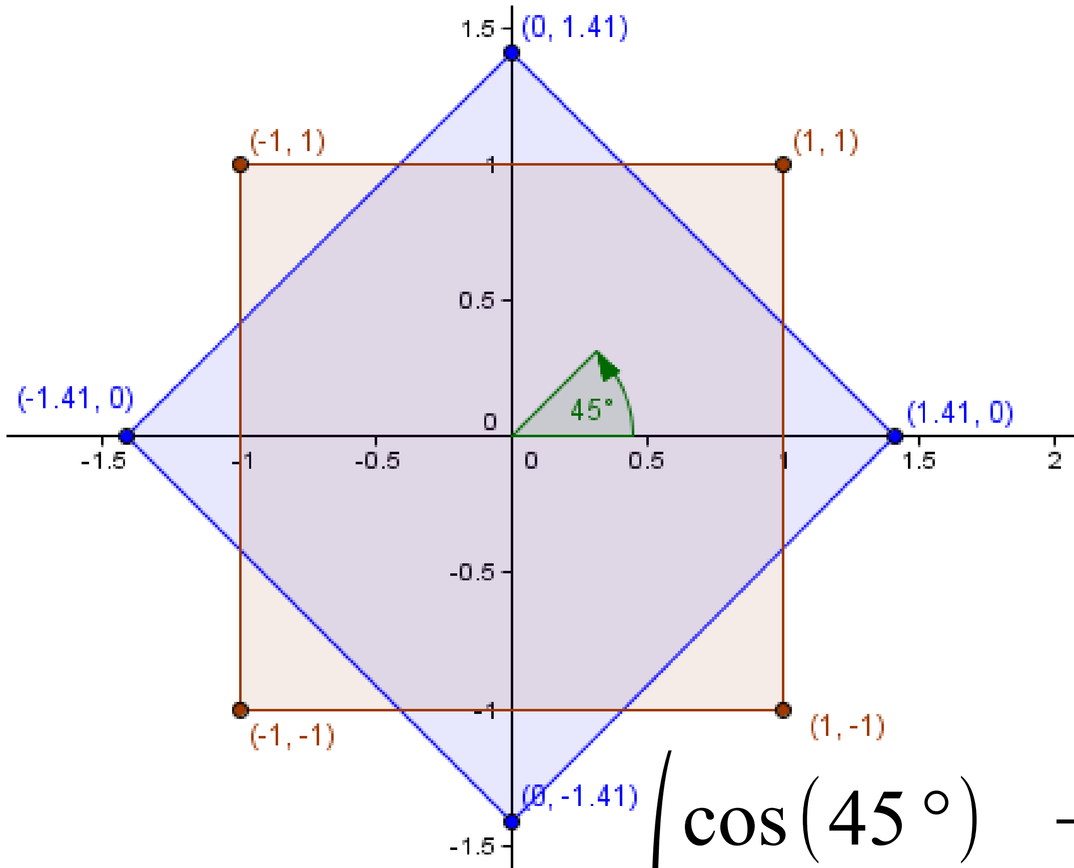


Multiple transformations



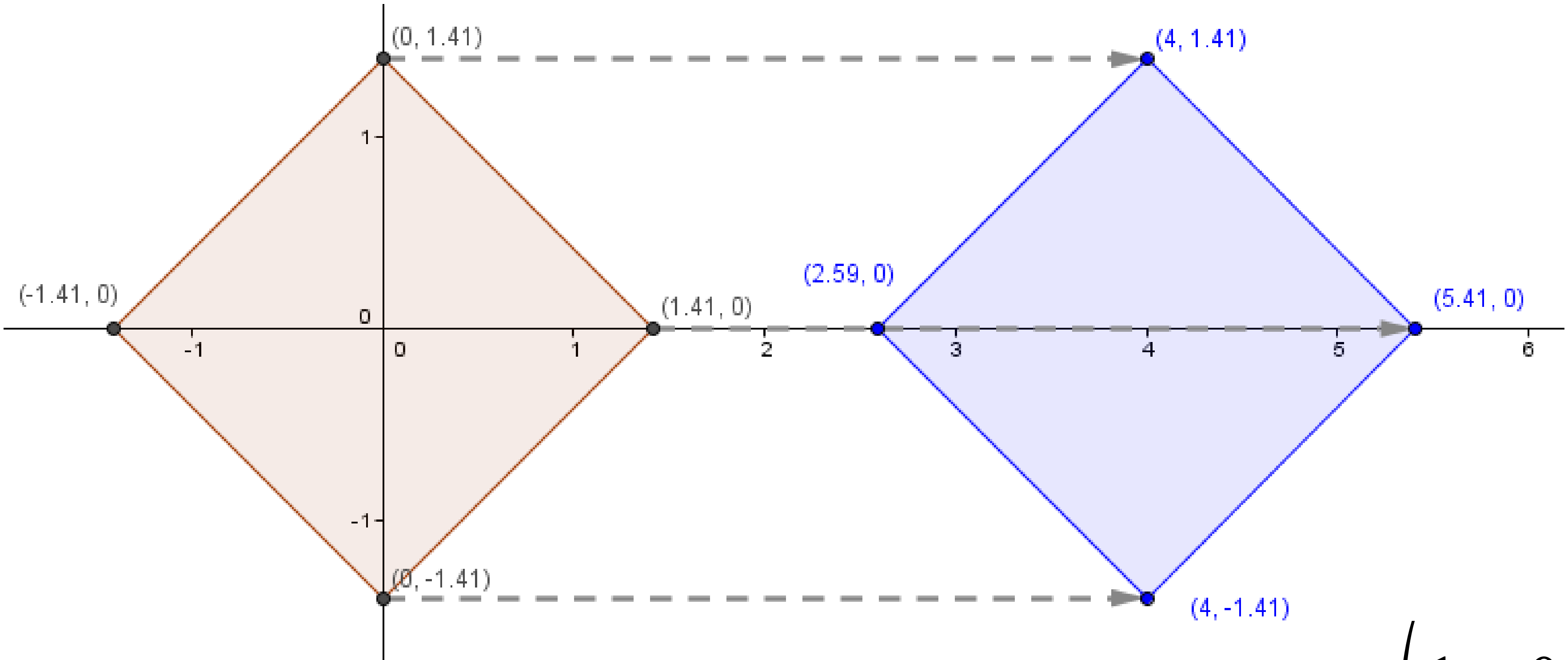
Our initial geometry defined by vertices: (-1, -1), (1, -1), (1, 1), (-1, 1)

Multiple transformations



$$\begin{pmatrix} \cos(45°) & -\sin(45°) & 0 \\ \sin(45°) & \cos(45°) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Multiple transformations



$$\begin{pmatrix} 1 & 0 & 4 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

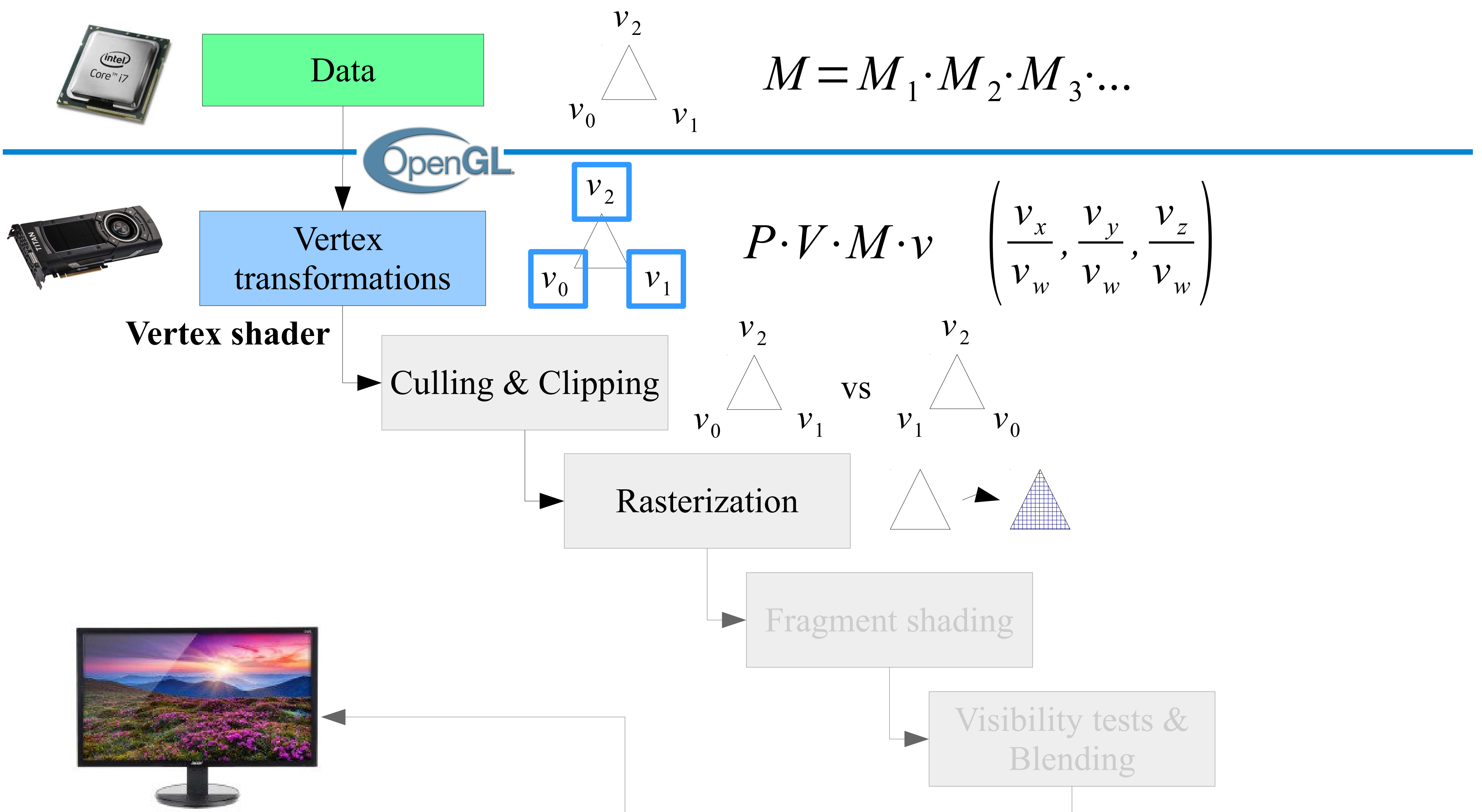
Multiple transformations

- Combine the transformations to a single matrix.

$$\begin{pmatrix} 1 & 0 & 4 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \cos(45°) & -\sin(45°) & 0 \\ \sin(45°) & \cos(45°) & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \cos(45°) & -\sin(45°) & 4 \\ \sin(45°) & \cos(45°) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

- This works for combining different affine transformations, but the result is hard to read...
- Order of transformations / matrices is important!
- http://cgdemos.tume-maailm.pri.ee

Now You Know



Data
v_2 v_0 v_1
$M = M_1 \cdot M_2 \cdot M_3 \cdot \ldots$
Vertex transformations
Vertex shader
$P \cdot V \cdot M \cdot v$
$\left(\frac{v_x}{v_w}, \frac{v_y}{v_w}, \frac{v_z}{v_w} \right)$
Culling & Clipping
vs
Rasterization
Fragment shading
Visibility tests & Blending

Next time...

- What is color in computer graphics?
- How to color our rasterized pixels?
- Light calculations.