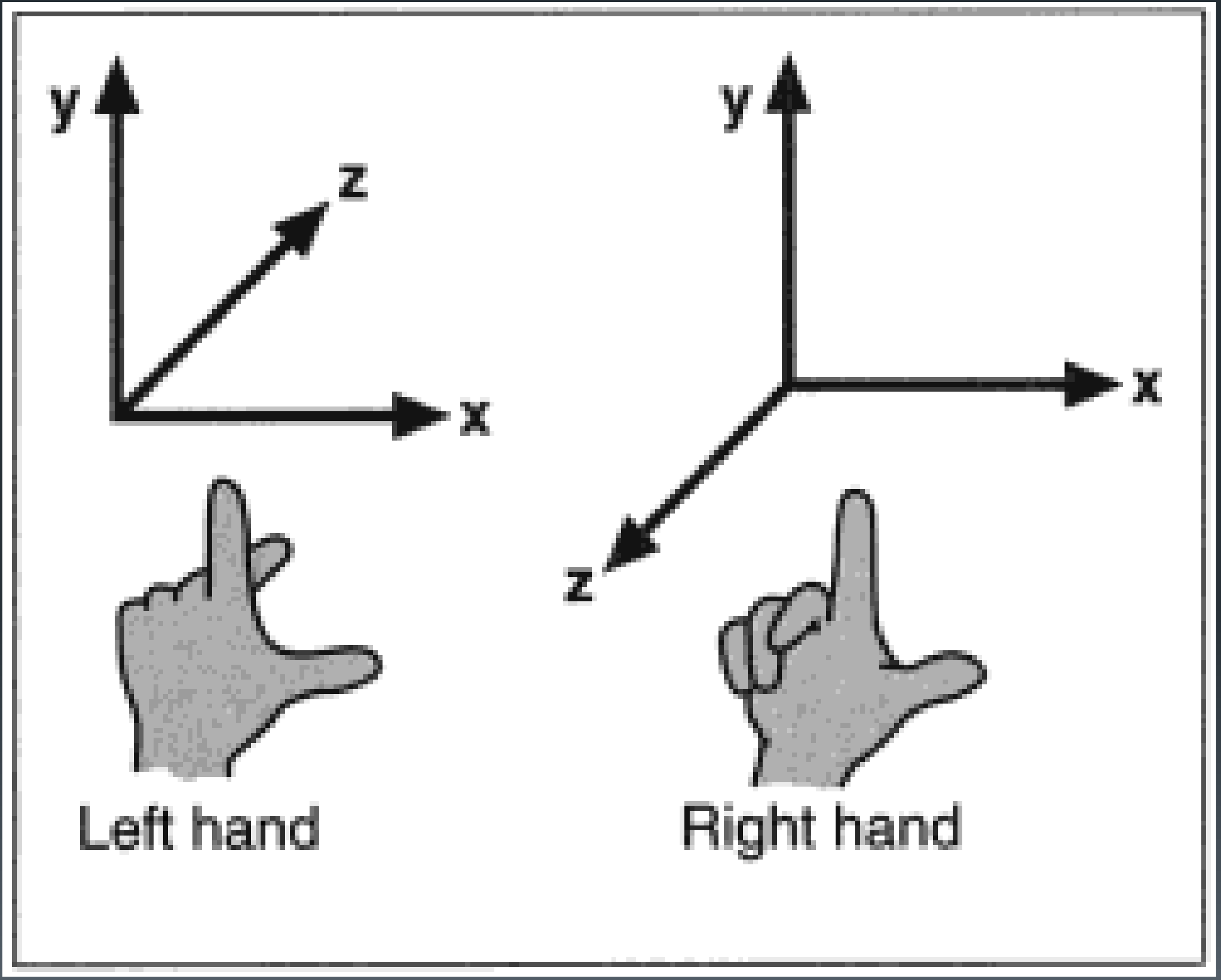
3D Mathematics

Co-ordinate systems, 3D primitives and affine transformations

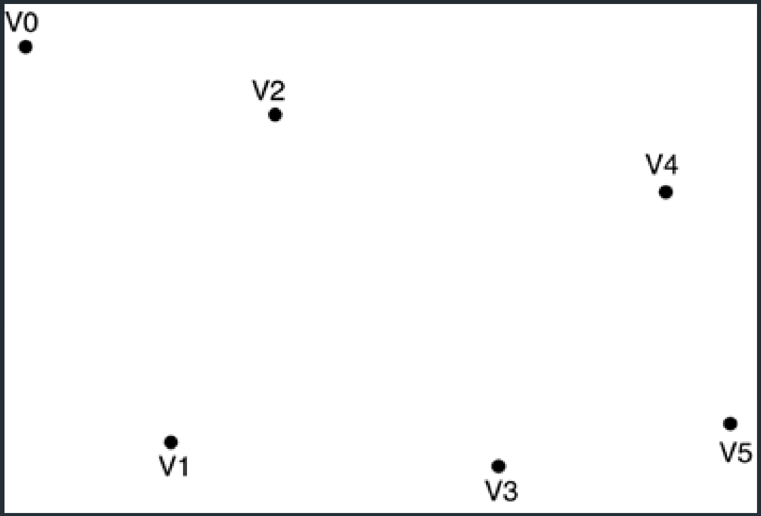
Coordinate Systems



Primitive Types and Topologies

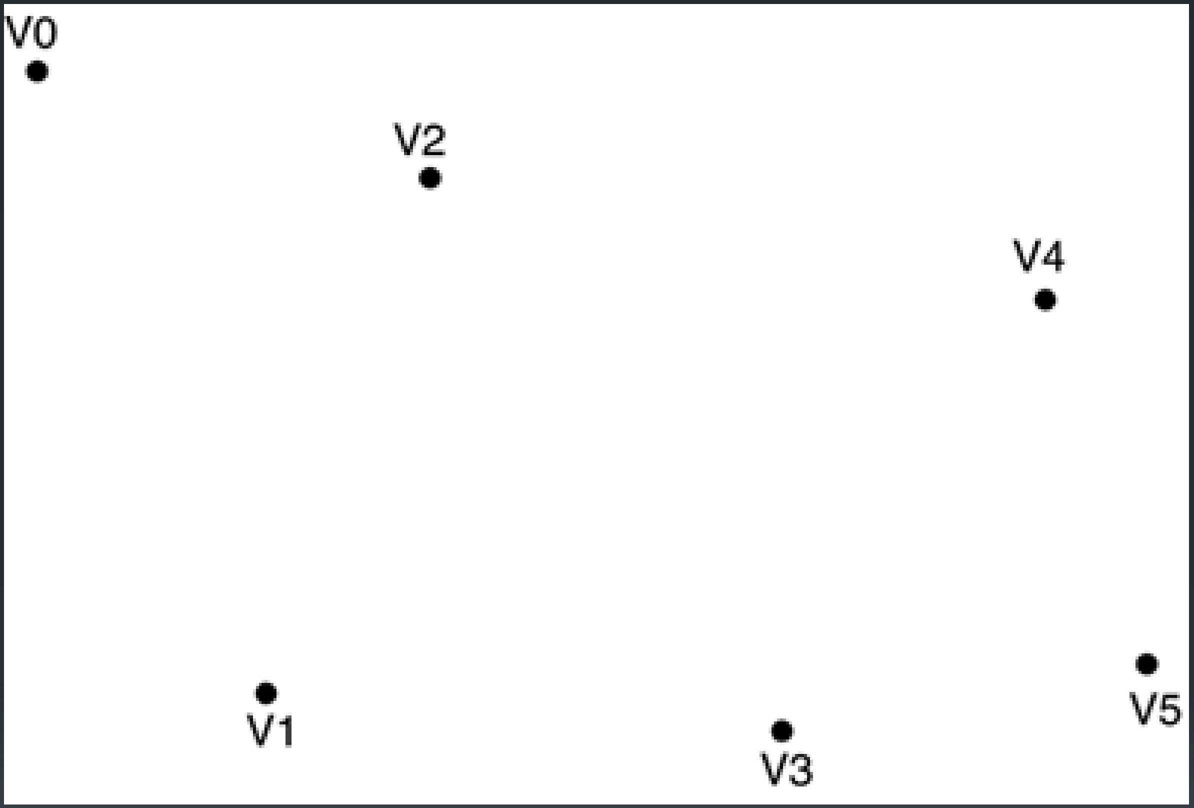
Primitives

Primitive Types and Topologies



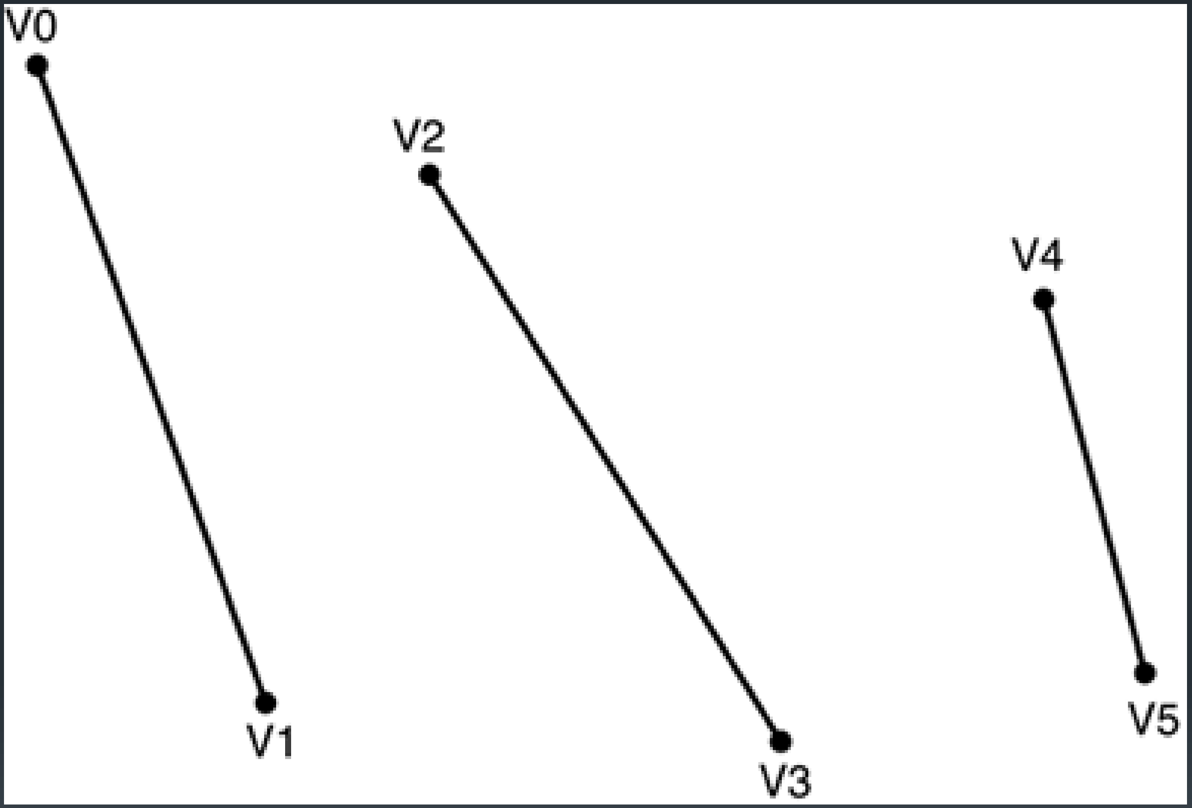
- A primitive is the most basic type of 3D object.
- Each primitive is defined by a set of vertices.
- The type of primitive is determined by the method of connection used to connect the vertices.
- This method of connection is referred to as the **primitive topology**.

Primitive Topologies: Point List



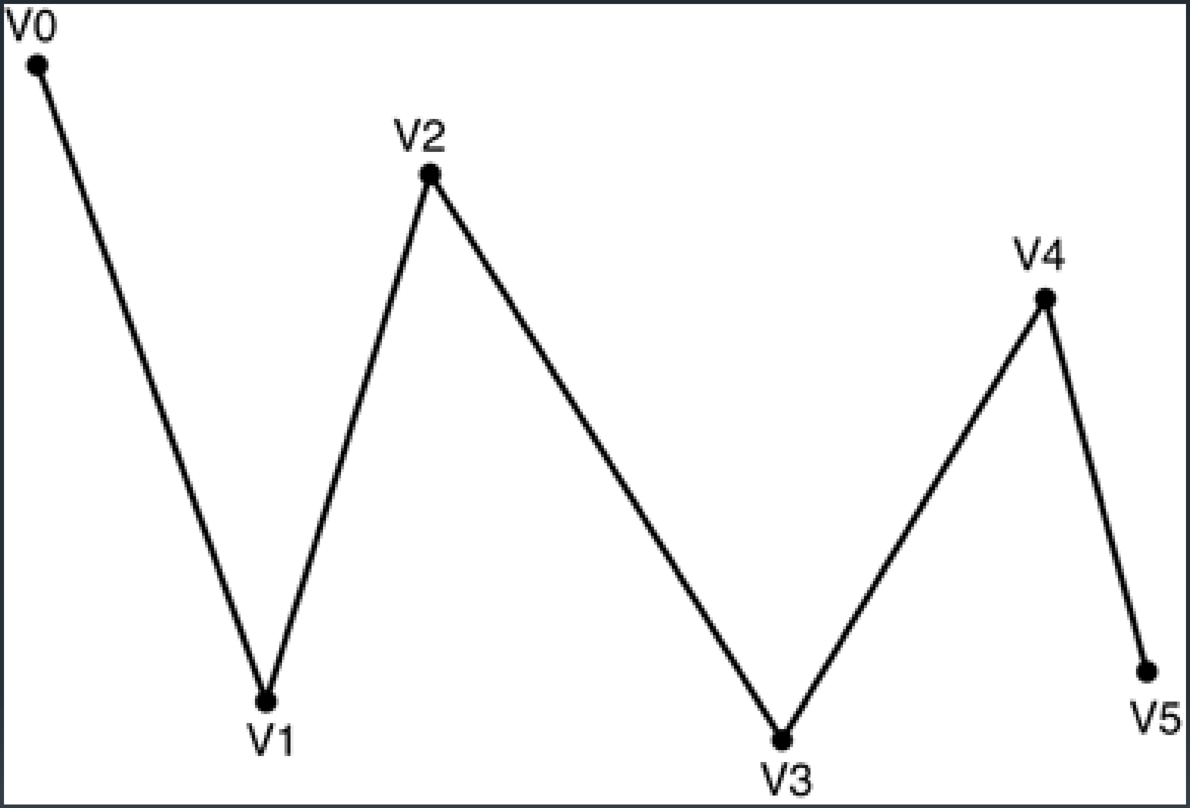
- A point is created for each vertex in the vertex set.

Primitive Topologies: Line List



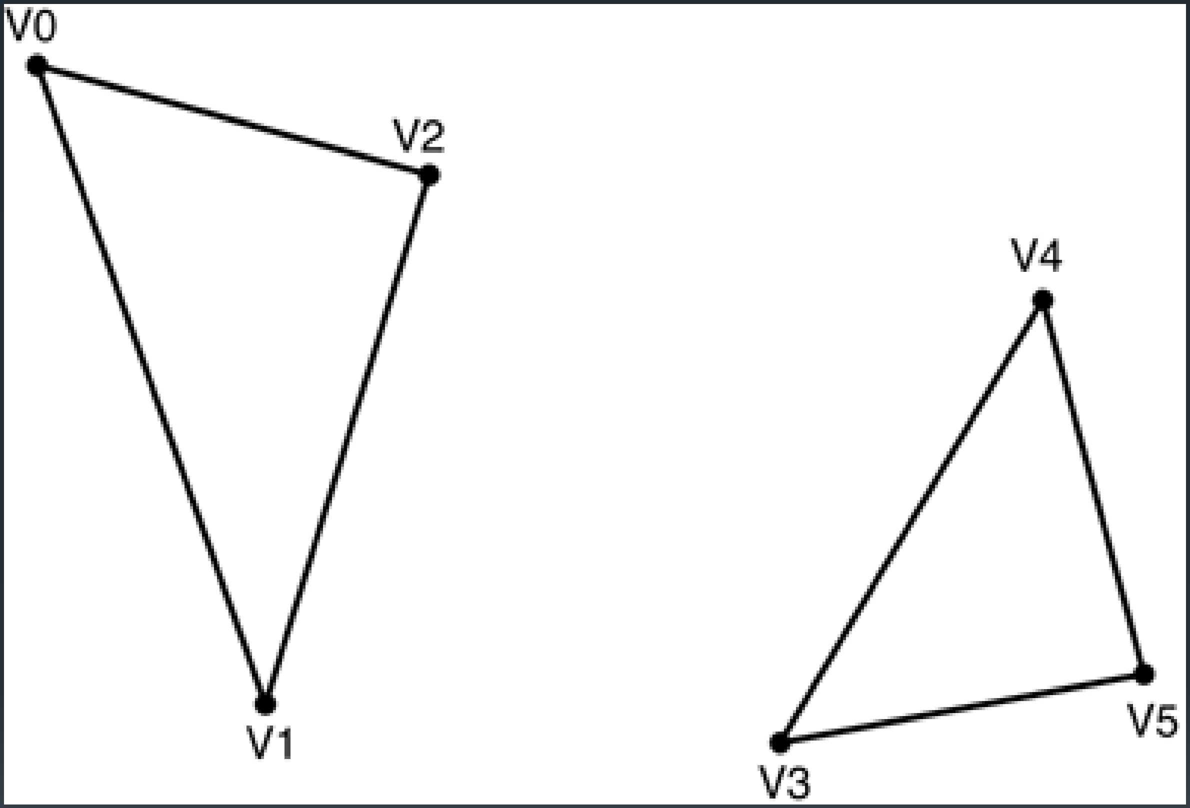
- A line is created for each vertex pair in the vertex set.
- **NOTE:** the order of vertices in the vertex set matters!

Primitive Topologies: Line Strip



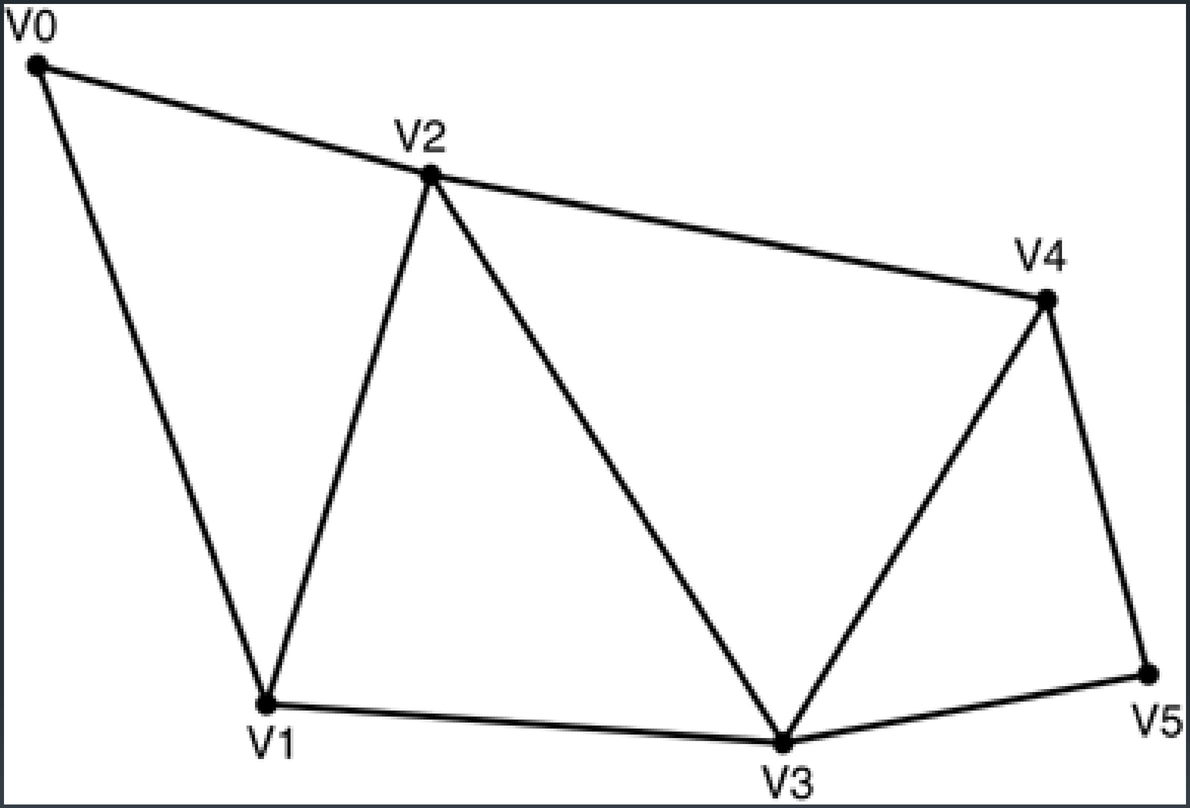
- A line is created between each vertex and the subsequent vertex in the vertex set.
- Creates one continuous line.

Primitive Topologies: Triangle List



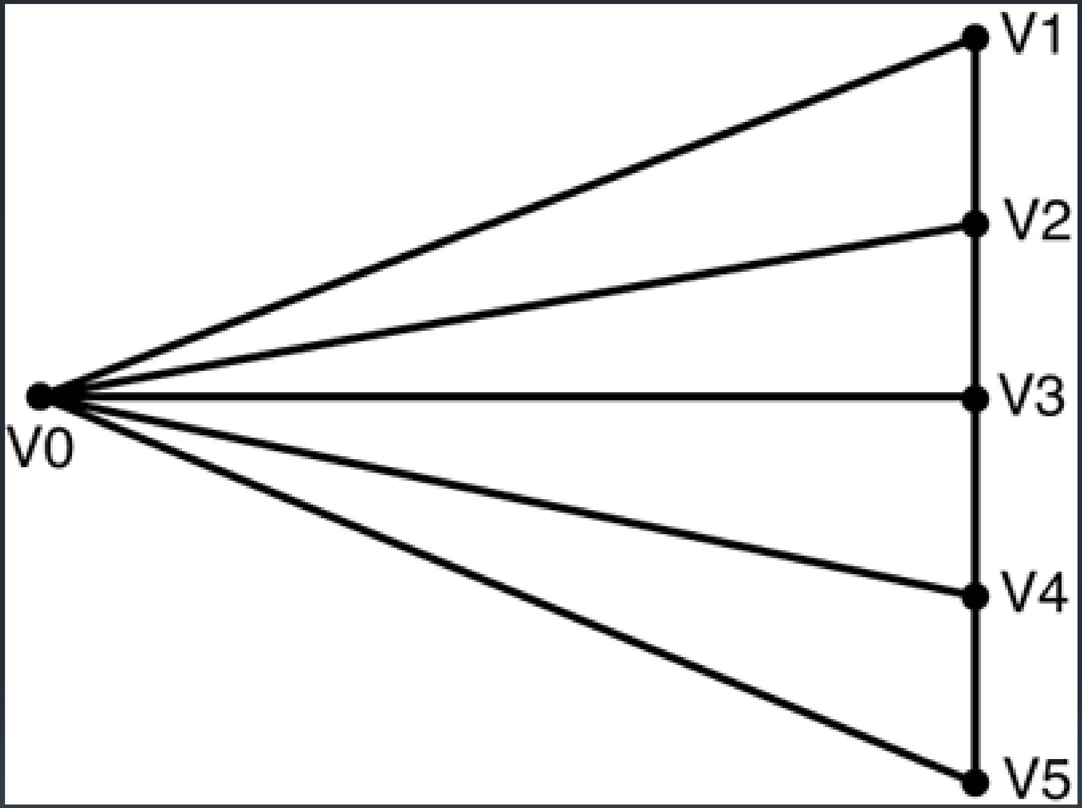
- A triangle is create for each vertex triplet in the vertex set.

Primitive Topologies: Triangle Strip



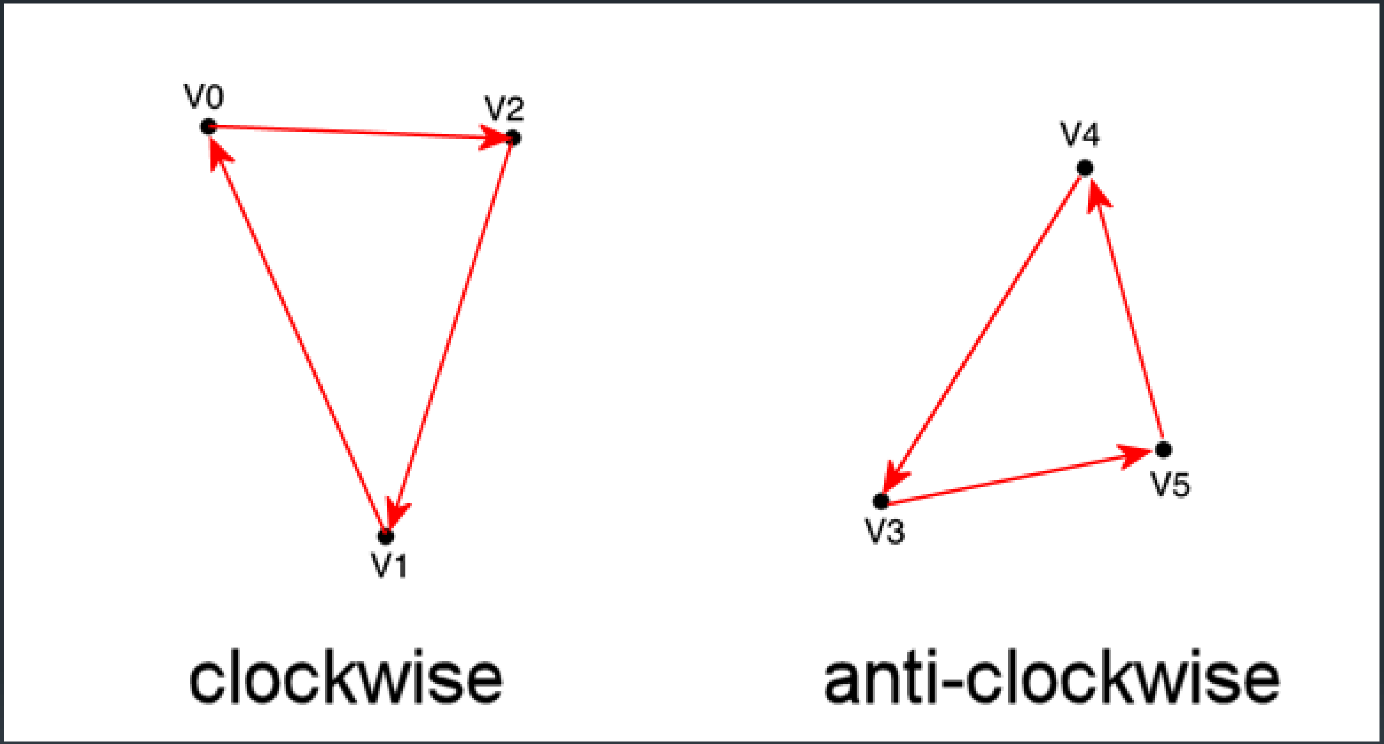
- A triangle is created for the first triplet in the vertex set.
- For each subsequent vertex in the vertex set, a triangle is created using that vertex and the previous two vertices in the vertex set.

Primitive Topologies: Triangle Fan



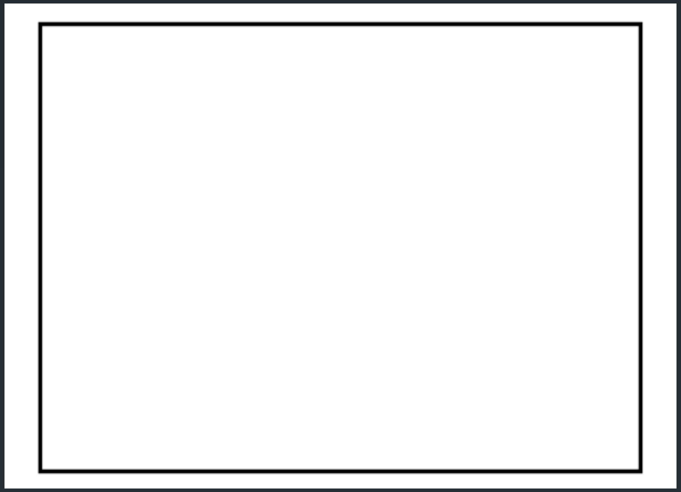
- A number of triangles are created around the first vertex in a vertex set.
- Not commonly used.

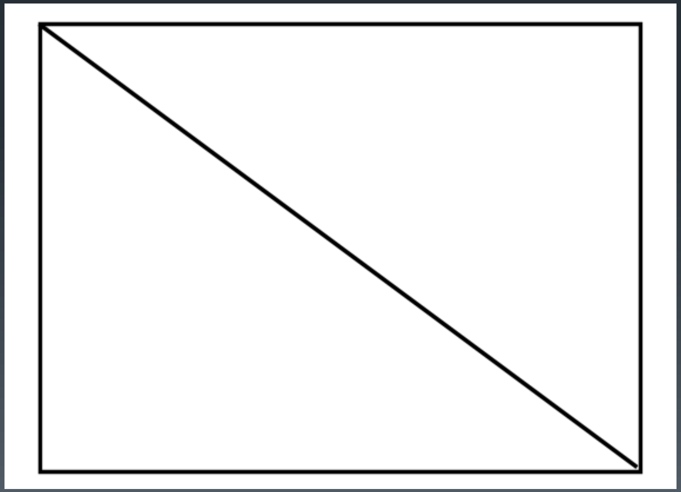
Triangle Winding



- The order in which the vertices of a triangle are specified control the winding direction.
- This is important as it defines the direction of the normal to the triangle. The use of this normal is discussed in a later section.

Triangle Based Rendering





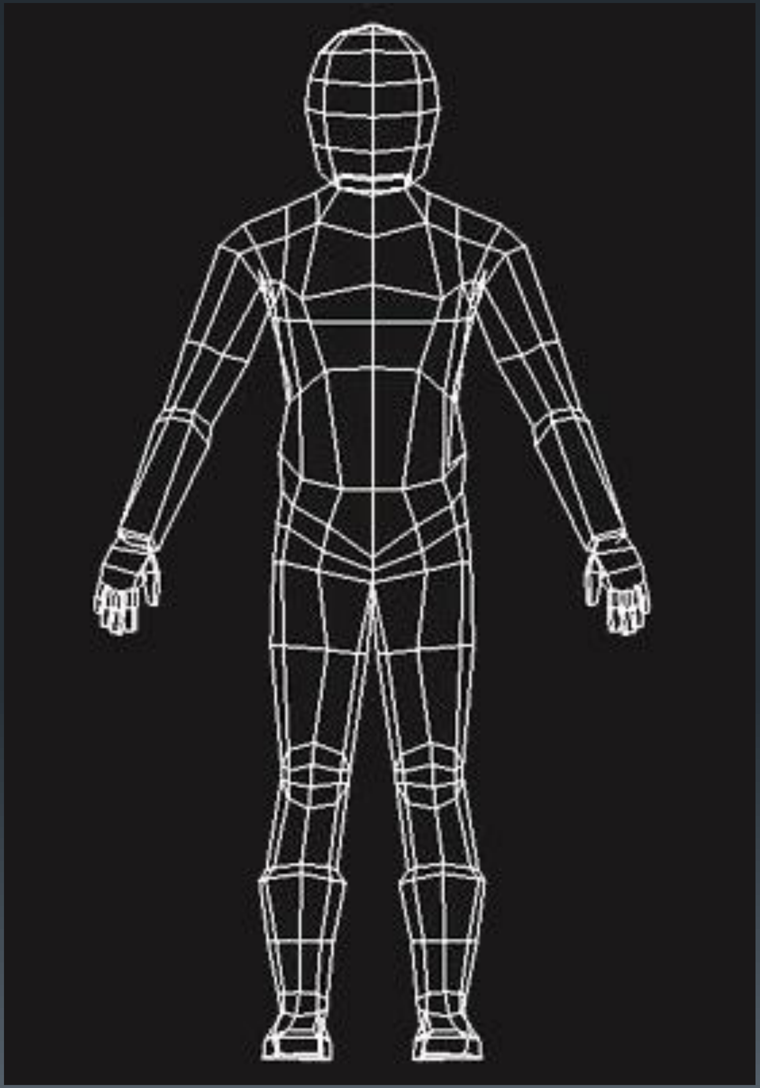
WHY?

- All the vertices in a triangle are co-planar meaning that a triangle is a planar shape.
- This means that the triangle is the simplest primitive that creates a plane.
- We can approximate any other polygon by the use of triangles.

Constructing and Positioning 3D Objects

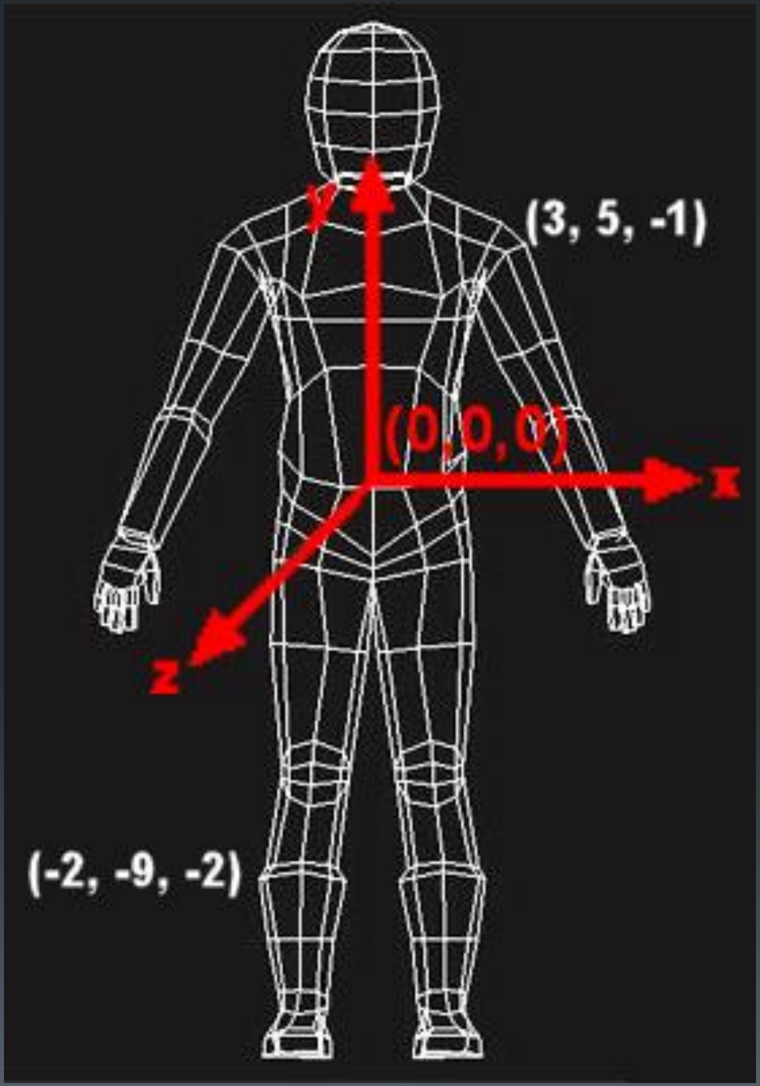
3D Objects

Constructing 3D Objects



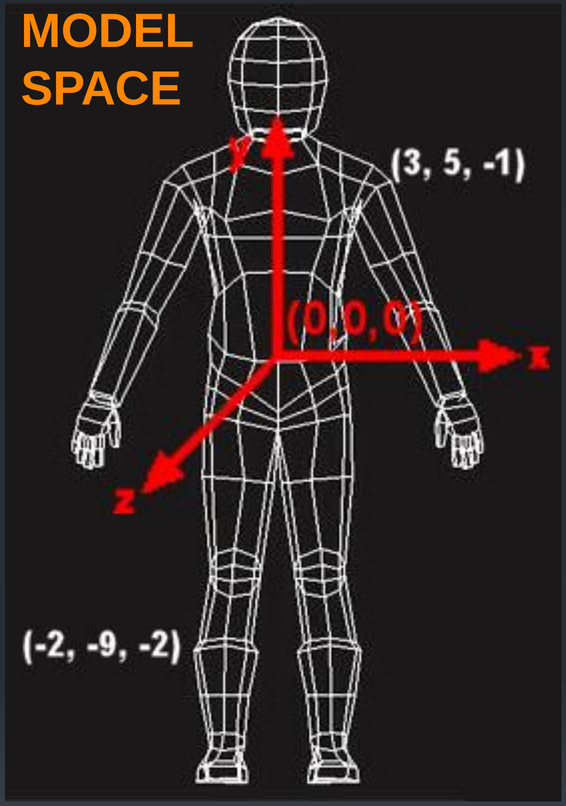
- All complex 3D objects are made up of a collection of triangle primitives.
- This collection of primitives is referred to as a triangle mesh.
- Each primitive is made up of three vertices. Meaning that a 3D mesh is simply a large vertex set.

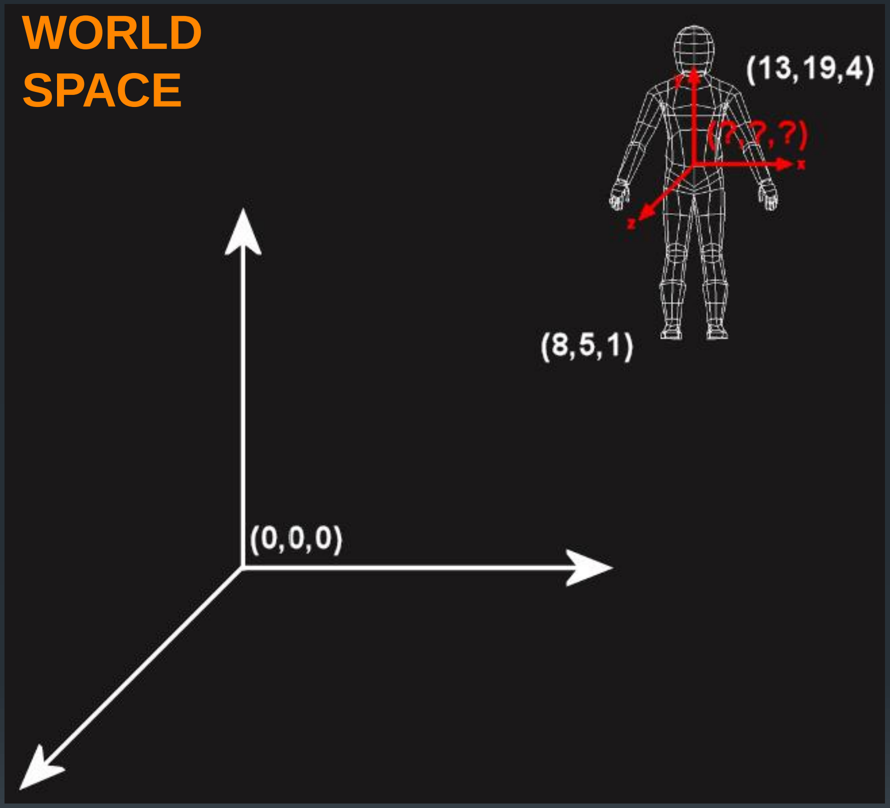
3D Objects: Model Space



- To correctly position all the vertices we first need a frame of reference.
- Each model has its own coordinate system called a **SPACE**.
- All vertices defined in the model are defined according to that space and so that space is known as **MODEL SPACE.**

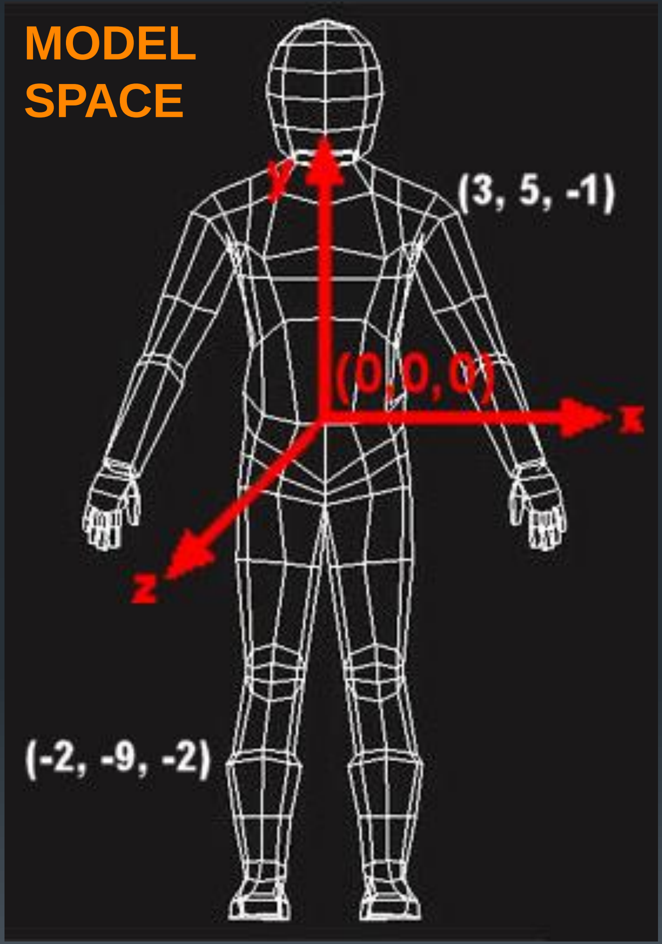
3D Objects: World Space

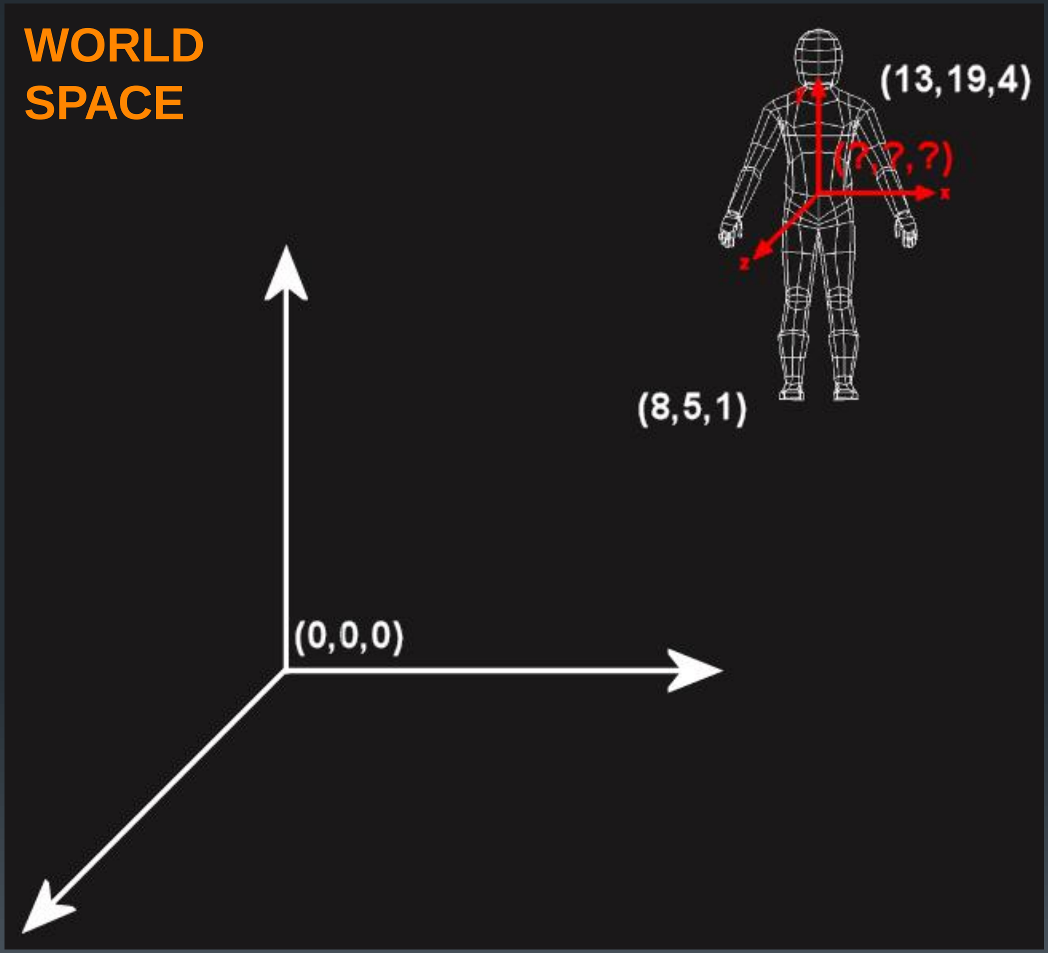




- Our scene also has its own coordinate system known as **WORLD SPACE** and every object has a position in the scene relative to that coordinate system.

3D Objects: World Space





- How do we position our model in the scene?
- What if the model was rotated?

Positioning 3D Objects

- To deal with rotation and positioning of objects, we need to think of in terms of positioning an objects coordinate system and not positioning the object.

- Since the object is defined around its coordinate system, if we transform the coordinate system then every point defined around that system will automatically be adjusted.

Translation, Rotation and Scaling

Affine Transformations

Linear Transforms

- A linear transform is one that preserves vector addition and scalar multiplication:

$$f(x) + f(y) = f(x + y)$$
$$kf(x) = f(kx)$$

- Scaling and rotation are linear transforms but translation is not since:

$$t(x) = x + t$$
$$t(x + y) = x + y + t$$
$$t(x) + t(y) = x + t + y + t$$

- Linear transforms can be represented by a 3x3 matrix, but translation cannot so we have to make use of homogenous coordinates and 4x4 matrices.

Homogenous Coordinates

- Homogenous coordinates for an R^n space are defined in an R^{n+1} space. So for 3 dimensions we use 4 dimensional coordinates: (x,y,z,w).

- Homogenous means "same type" and in this case it refers to the fact that we can define both points and vectors using the same notation.

- The **w** element denotes a point if set to 1 or a vector if set to 0.

Point : w=1 : (x,y,z,1)
Direction Vector : w=0 : (x,y,z,0)

- As we are now working in 4D we need a 4x4 matrix to represent our transforms.

Affine Transformations

- To deal with the non-linearity of the translation transform we make use of affine transformations.

- An affine transformation is one that performs a linear transform followed by a translation. It also preserves parallelism of lines and the colinearity (all points will still lie on the same line after the transform) of points.

Affine Transformations

- Affine transforms are represented by 4x4 matrices using homogenous coordinates.
- An affine transform may also be any sequence of concatenations of individual affine transforms.
- To apply an affine transformation you will multiply all points that need to be transformed by the transformation matrix.
- A rigid body transform is one that preserves distances between points transformed and handedness (i.e. never swaps left and right)

Translation

- Translation is the change of location and is represented by the translation matrix **T** which translates an entity by the vector **t** **where t is (t_x, t_y, t_z).**

$$T(t) = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Change on X axis

Change on Y axis

Change on Z axis

$$Translation = T(t)\,p$$

- Translation is a rigid body transform

Scaling

- Scaling is used to enlarge or shrink an object using the scaling factors sx, sy, sz, which affect the scaling of the object in those three axes.

$$S(s) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Scaling = S(s)p$$

Scale on X axis

Scale on Y axis

Scale on Z axis

Shear

- Shearing or Skewing is used to tilt/skew geometry, there are six basic shearing matrices: $H_{xy}(s)$, $H_{xz}(s)$, $H_{yx}(s)$, $H_{yz}(s)$, $H_{zx}(s)$, $H_{zy}(s)$

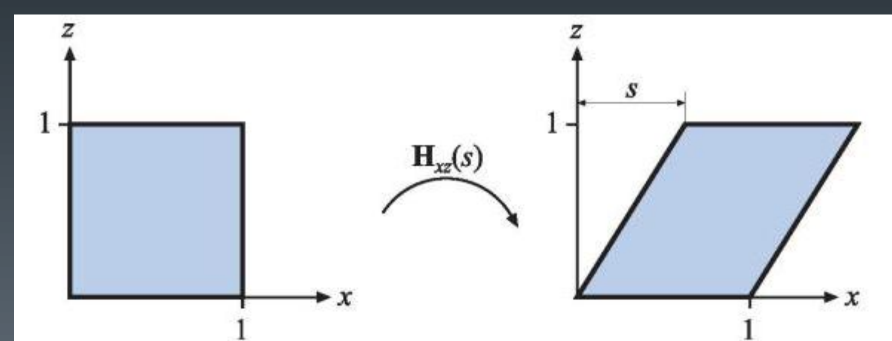
first subscript defines row

Second subscript defines column

$$H_{xz}(s) = \begin{bmatrix} 1 & 0 & s & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Shearing = H_{xz}(s)\,p$$

$H^{-1}_{xz}(s) = H_{xz}(-s)$



Rotation

- The rotation transform rotates a vector by a given angle around a given axis passing through the origin.

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

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

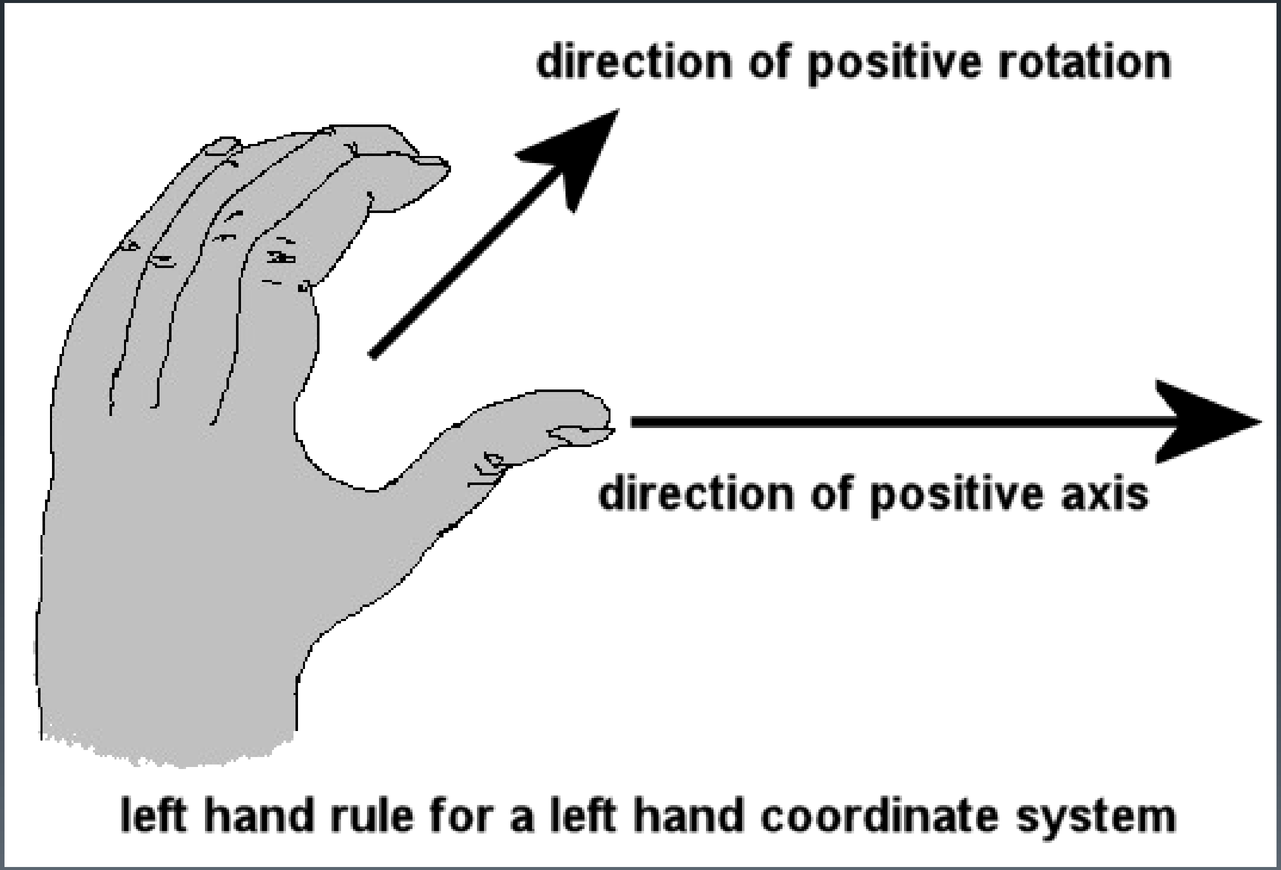
$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Rotation = R_{x/y/z}(\theta)\,p$$

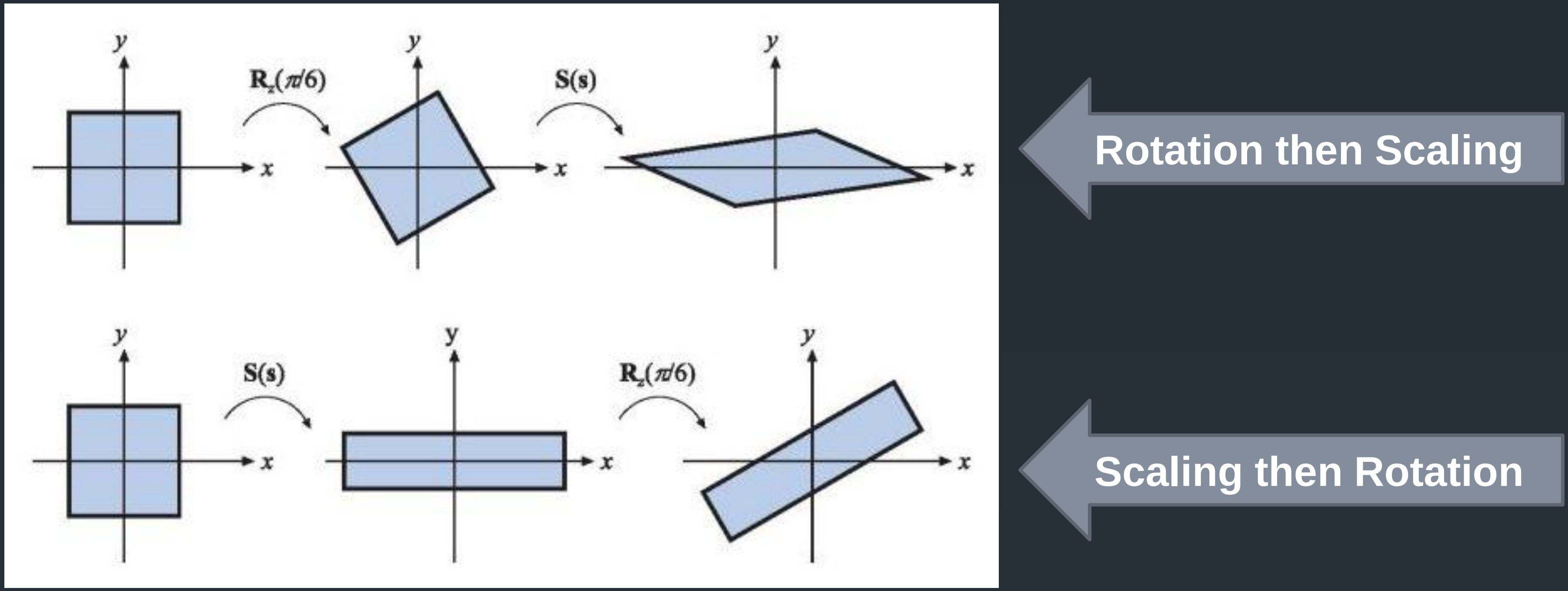
NOTE: Angles are in Radians!!

Direction of Rotation

- The direction of rotation can be calculated using a left hand or right hand rule depending on the handedness of the coordinate system in use (left/right).



Concatenations of Transforms



- Since matrix multiplication is non-commutative, the order of concatenation of affine transformation matrices is very important.

Concatenations of Transforms

- Since matrix multiplication is non-commutative, the order of concatenation of affine transformation matrices is very important.

- For example, if we want to scale then rotate then translate the complete transform will be:
  - **C = T R S**
- With the order being right to left, so for each point **p** in an object:
  - **C = (T (R (Sp)))**
- It is important to note that matrix multiplication is associative:
  - **C = TRSp = (TR)(Sp)**

D3DX10's Transforms

- You will notice that the matrix is multiplied with the point to be transformed, and not the point multiplied by the transformation matrix (which would be the intuitive approach)
- Transform of point p by matrix T = Tp
- Since this is not intuitive, the D3DX math library defines the transformation matrices as transposes due to the property that: $Tp = p^T T^T$

$$\begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} = \begin{bmatrix} p_x & p_y & p_z & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ t_x & t_y & t_z & 1 \end{bmatrix}$$

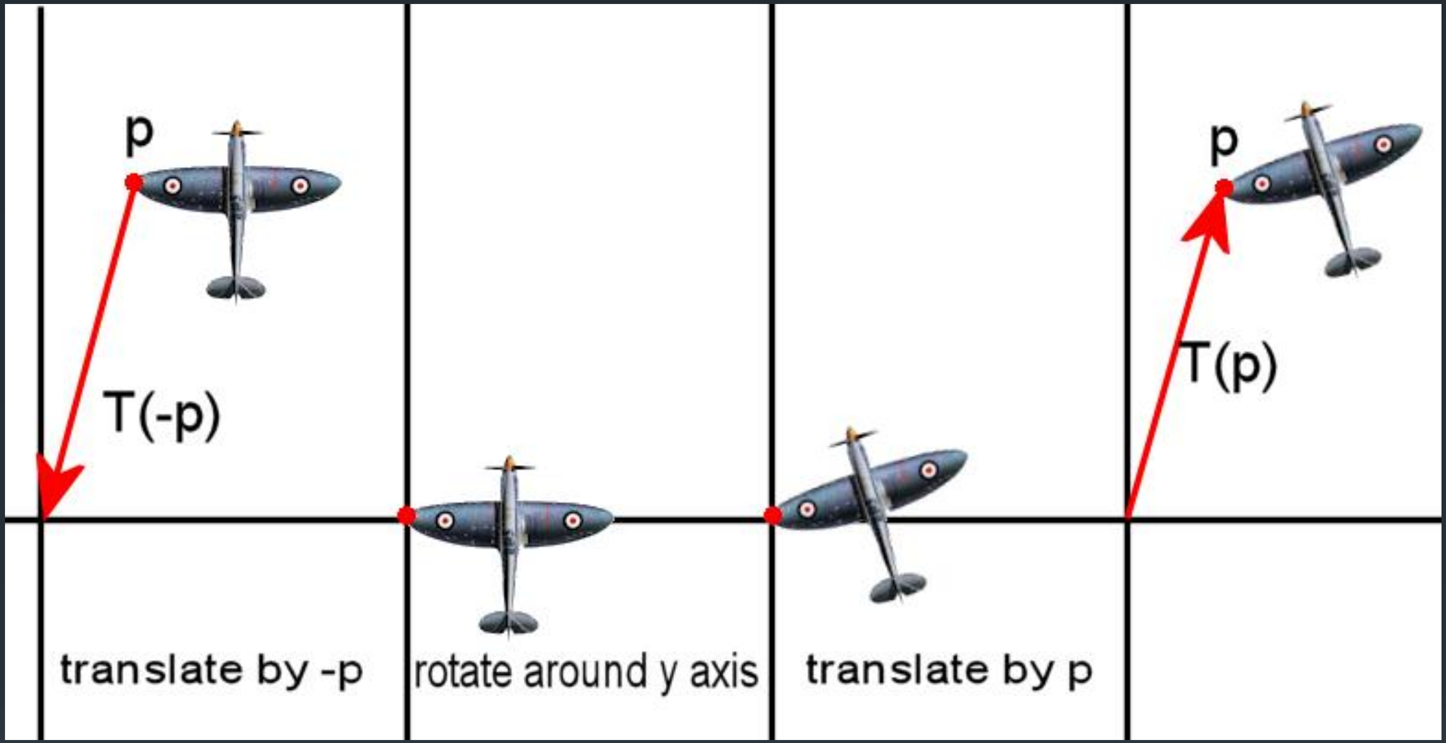
The Normal Transform

- A single affine transformation matrix can be used to transform lines, points, polygons and other geometry but cannot always be used to transform the surface normal.
- Translations and rotations do not affect the normal but scaling & shearing does!
- Uniform scaling simple affects the normal’s length so the surface normal needs to be normalized (made into a unit vector again)
- Non-uniform scaling changes the direction of the normal and so it needs to be recalculated.

Rotation around a point, rotation around an arbitrary axis, Transform concatenation examples.

Practical Examples of Affine Transformations

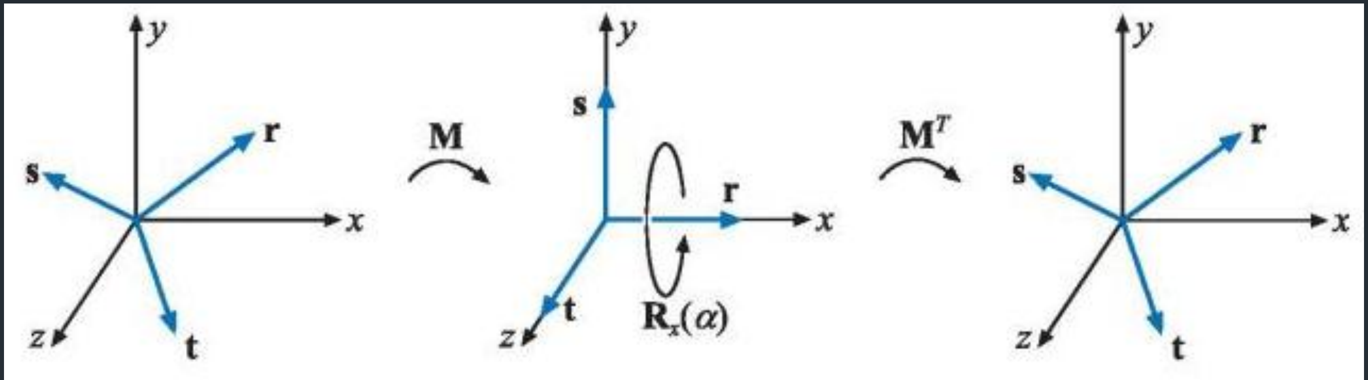
Rotation about a Fixed Point



- The achieve rotation around a specific point, first translate the object so that point is now at the origin, then perform the required rotation and translate the object back to its original position.

The complete transform X is : $X = T(p)R_y(\theta)T(-p)$

Rotation about an Arbitrary Axis



$$s = (0, -rz, ry) \text{ if } rx = \min(|rx|, |ry|, |rz|)$$

$$s = (-rz, 0, rx) \text{ if } ry = \min(|rx|, |ry|, |rz|)$$

$$s = (-ry, rx, 0) \text{ if } rz = \min(|rx|, |ry|, |rz|)$$

$$s = s / \|s\|$$

$$t = r \times s$$

$$M = \begin{bmatrix} r \\ s \\ t \end{bmatrix}$$

$$X = M^T R_x(\theta) M$$

- Make Sure that **r** is a unit vector
- Find two more unit length axes (**s,t**) to form an orthonormal system
- To find **s**, set the smallest value to zero and swap the remaining elements and negate the first one, then normalize **s**.
- To find **t**: **r** cross product **s**
- Create matrix M, this matrix transforms vector **r** to the x axis, **s** to the y axis and **t** to the z axis.
- Rotate around **x** (technically **r**) and then reverse coordinate system transform

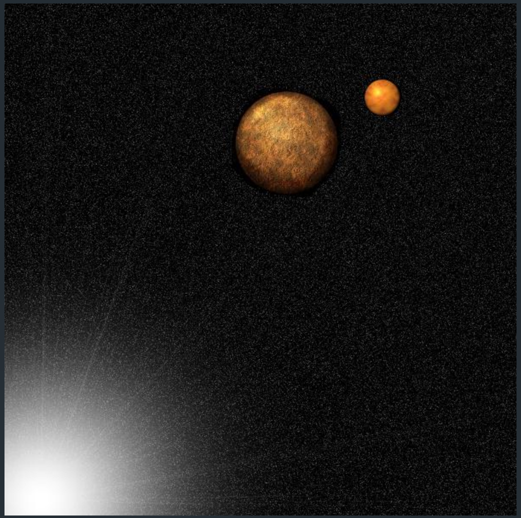
Transform Examples

- We have an object centered around the point (1,0,0) in object space. This object is positioned at (4,3,-1) in world space.

- After we position the object, we want to rotate the object 45 degrees to the left around the Y axis. What does the final transform look like?

$$\mathbf{FT = T(5,3,-1)\ \ R_y(-\pi/2)\ \ T(-5,-3,1)}$$

Space Example



- We have a planet P at position p orbiting (rotating around its own center as well) a sun S centered at the origin (s) at a distance of 10 units. The speed of rotation is 0.1 Radians per frame.
- Planet P has a moon M at position m orbiting planet P at a distance of 2 units. The moon orbits the planet at a speed of 0.01 Radians per frame.
- What are the transforms necessary to perform the necessary adjustments at each frame?