CPSC 314
2021W2 Notepack

Instructor: Dinesh K. Pai

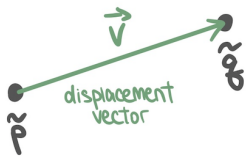
Difference in Notations, this course vs textbook		
	CPSC 314	Textbook
Point	$\widetilde{\boldsymbol{p}}$	$\widetilde{\boldsymbol{p}}$
Vector	$\vec{\boldsymbol{v}}$	$\vec{\boldsymbol{v}}$
Column Matrix	$\bar{\mathsf{v}}$	$\boldsymbol{v}$(bold)
Row Matrix	$\underline{\mathsf{v}}$	$\boldsymbol{v}^T$ (bold)
Basis	$\underline{\vec{\boldsymbol{b}}}$	$\vec{\boldsymbol{b}}^T$ (bold)

1 Points, Vectors and Coordinates

Points $\neq$ Vectors $\neq$ Arrays

A **point** is a real location in space:

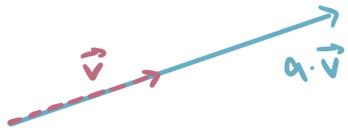
A **vector** is an algebraic object. A canonical example is the *displacement vector* from one point to another.



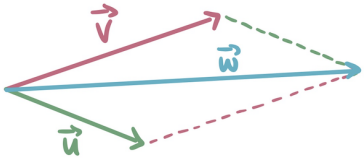
With vectors, we can:

1. Multiply with a scalar:

$$\text{a} \times \vec{\boldsymbol{v}} \rightarrow \text{a new vector}$$



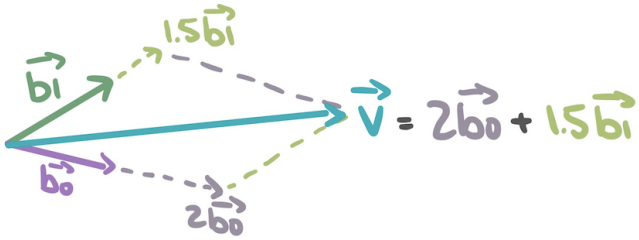
2. Add 2 vectors together



$$\vec{u} + \vec{v} = \vec{w}$$

1.1 Basis

A basis is an independent set of vectors that can produce any vector in a given space by linear combination. The size of the basis is the dimension of the space.



1.2 Coordinates of a Vector in a Basis $\underline{\vec{b}}$

We can write any vector $\vec{v}$ as:

$$\vec{v} = v_0 \times \vec{b}_0 + v_1 \times \vec{b}_1$$

$$\vec{v} \xrightarrow{\text{in basis } \underline{\vec{b}}} \begin{pmatrix} v_0 \\ v_1 \end{pmatrix} = \overline{\mathsf{v}}$$

NOTATION

$$\overline{\mathsf{v}} = \begin{pmatrix} v_0 \\ v_1 \end{pmatrix}$$

column matrix

$$\underline{\mathsf{v}} = \begin{pmatrix} v_0 & v_1 \end{pmatrix}$$

row matrix

Then, the basis can be written as:

$$\underline{\vec{\boldsymbol{b}}} = \begin{pmatrix} \vec{\boldsymbol{b}}_0 & \vec{\boldsymbol{b}}_1 \end{pmatrix}$$

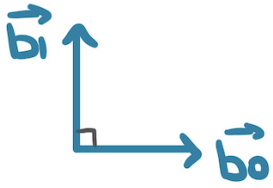
We can then write this very compactly

$$\vec{\boldsymbol{v}} = v_0 \times \vec{\boldsymbol{b}}_0 + v_1 \times \vec{\boldsymbol{b}}_1 = \begin{pmatrix} \vec{\boldsymbol{b}}_0 & \vec{\boldsymbol{b}}_1 \end{pmatrix} \begin{pmatrix} v_0 \\ v_1 \end{pmatrix} = \underline{\vec{\boldsymbol{b}}}\, \overline{\mathsf{v}}$$

, where $\vec{\boldsymbol{v}}$ is a real vector, $\underline{\vec{\boldsymbol{b}}}$ is a basis, and $\overline{\mathsf{v}}$ is the column matrix of coordinates of $\vec{\boldsymbol{v}}$ in basis $\underline{\vec{\boldsymbol{b}}}$.

1.3 Orthonormal Basis

Basis vectors are of unit length, and perpendicular to each other.



Computing the components of a vector is easier in an orthonormal basis.
Ex. To calculate the component along $\vec{\boldsymbol{b}}_0$, compute a dot product:

$$\vec{\boldsymbol{v}} \cdot \vec{\boldsymbol{b}}_0 = 1.1 \times \vec{\boldsymbol{b}}_0 \cdot \vec{\boldsymbol{b}}_0 + 0.9 \times \vec{\boldsymbol{b}}_0 \cdot \vec{\boldsymbol{b}}_1 = 1.1 \times 1 + 0.9 \times 0$$

1.4 Change of Basis

Suppose we have 2 basis, $\underline{\vec{\boldsymbol{a}}}$ and $\underline{\vec{\boldsymbol{b}}}$. Given that $\vec{\boldsymbol{v}} = \underline{\vec{\boldsymbol{b}}}\, \overline{\mathsf{v}}_b$, we want to find its coordinates in $\underline{\vec{\boldsymbol{a}}}$ instead (i.e., $\overline{\mathsf{v}}_a$).

- Remember that both coordinates represent the same physical vector:
  $\vec{\boldsymbol{v}} = \underline{\vec{\boldsymbol{b}}}\, \overline{\mathsf{v}}_b = \underline{\vec{\boldsymbol{a}}}\, \overline{\mathsf{v}}_a$

- Express $\underline{\vec{\boldsymbol{b}}} = \begin{pmatrix} \vec{\boldsymbol{b}}_0 & \vec{\boldsymbol{b}}_1 \end{pmatrix}$ in $\underline{\vec{\boldsymbol{a}}}$:
  $\vec{\boldsymbol{b}}_0 = \underline{\vec{\boldsymbol{a}}} \begin{pmatrix} l_{00} \\ l_{10} \end{pmatrix}$ and $\vec{\boldsymbol{b}}_1 = \underline{\vec{\boldsymbol{a}}} \begin{pmatrix} l_{01} \\ l_{11} \end{pmatrix}$
- The exchange from basis $\underline{\vec{\boldsymbol{a}}}$ to basis $\underline{\vec{\boldsymbol{b}}}$ is then:
  $\underline{\vec{\boldsymbol{b}}} = \underline{\vec{\boldsymbol{a}}} \begin{pmatrix} l_{00} & l_{01} \\ l_{10} & l_{11} \end{pmatrix} = \underline{\vec{\boldsymbol{a}}}\ \underline{\mathsf{L}}$
- Can finally express: $\underline{\vec{\boldsymbol{a}}}\ \underline{\mathsf{L}}\ \overline{\mathsf{v}_\mathsf{b}} = \underline{\vec{\boldsymbol{a}}}\ \overline{\mathsf{v}_\mathsf{a}}$

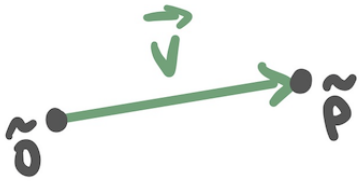
Therefore, the desired transformation of coordinates is

$$\boxed{\underline{\mathsf{L}}\ \overline{\mathsf{v}}_b = \overline{\mathsf{v}}_a}$$

Tip: Think of a basis as a type of currency. Going from one basis to another is similar to converting a monetary amount from CAD to USD.

1.5 Representing Points (and Vectors)

Given a vector $\vec{\boldsymbol{v}}$ and a basis $\underline{\vec{\boldsymbol{b}}}$, we have seen that we can obtain its coordinates in this basis. We can also say that $\vec{\boldsymbol{v}}$ represents the displacement from the origin $\widetilde{\mathbf{0}}$ (a special point) to another point $\widetilde{\boldsymbol{p}}$.



We can extend the + operation to allow the addition of a point and a vector, which will give us another point:

$$\widetilde{\boldsymbol{p}} = \vec{\boldsymbol{v}} + \widetilde{\mathbf{0}} = \vec{\boldsymbol{b}}_0 v_0 + \vec{\boldsymbol{b}}_1 v_1 + \widetilde{\mathbf{0}} = \begin{pmatrix} \vec{\boldsymbol{b}}_0 & \vec{\boldsymbol{b}}_1 & \widetilde{\mathbf{0}} \end{pmatrix} \begin{pmatrix} v_0 \\ v_1 \\ 1 \end{pmatrix} = \underline{\widetilde{\boldsymbol{b}}}\ \overline{\mathsf{p}}_m$$

...where $\underline{\widetilde{\boldsymbol{b}}} = \begin{pmatrix} \vec{\boldsymbol{b}}_0 & \vec{\boldsymbol{b}}_1 & \widetilde{\mathbf{0}} \end{pmatrix}$ is called a **coordinate frame** and $\overline{\mathsf{p}}_m = \begin{pmatrix} v_0 \\ v_1 \\ 1 \end{pmatrix}$ is the **homogenous coordinates** of a point (in 2D).

Take-home question 1: How can we represent a vector using a coordinate frame?

Take-home question 2: What are the columns of $\underline{\widetilde{\boldsymbol{b}}}$ in 3D space?

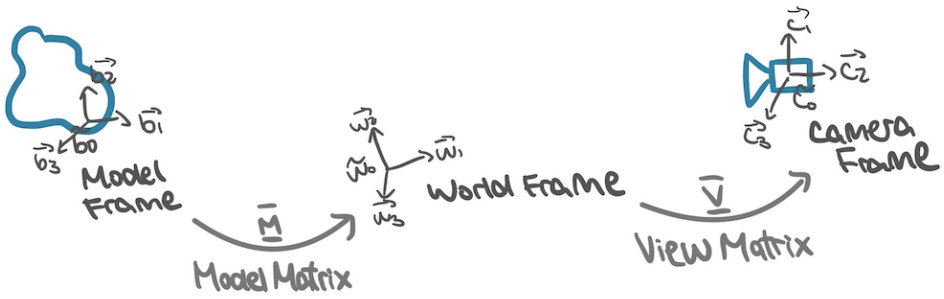
1.6 Three Important Frames

The three important frames are:

1. Model frame $\underline{\widetilde{\boldsymbol{b}}}$ (1 per object)
2. World frame $\underline{\widetilde{\boldsymbol{w}}}$
3. Camera frame $\underline{\widetilde{\boldsymbol{c}}}$ (1 per camera)

To go from one frame to another, we use transformation matrices:

1. Model matrix $\underline{\overline{\mathsf{M}}}$
2. View matrix $\underline{\overline{\mathsf{V}}}$



The transformation matrices can be inverted or combined with each other for different operations. For instance, the ModelView matrix is the concatenation of the Model and View matrices.

Ex. Suppose we have a point $\widetilde{\boldsymbol{p}}$ whose coordinates in its model frame are $\overline{\mathsf{p}}_m$. How can we obtain its coordinates in the world frame?

$\longrightarrow$ Answer: $\overline{\mathsf{p}}_w = \underline{\overline{\mathsf{M}}}\ \overline{\mathsf{p}}_m$

Recall that $\widetilde{\boldsymbol{p}} = \underline{\widetilde{\boldsymbol{b}}}\ \overline{\mathsf{p}}_m = \underline{\widetilde{\boldsymbol{w}}}\ \overline{\mathsf{p}}_w$. From the expression in the example above, we can multiply both sides by the world frame and obtain:

$$\underline{\widetilde{\boldsymbol{w}}}\ \overline{\mathsf{p}}_w = \underline{\widetilde{\boldsymbol{w}}}\ \underline{\overline{\mathsf{M}}}\ \overline{\mathsf{p}}_m = \widetilde{\boldsymbol{p}}$$

From this modified expression, we can see that $\underline{\widetilde{\boldsymbol{w}}}\ \underline{\overline{\mathsf{M}}} = \underline{\widetilde{\boldsymbol{b}}}$. We now have a method of converting 1 frame to another.

1.7 Anatomy of a Transformation Matrix

1.7.1 The Model Matrix $\underline{\overline{\mathsf{M}}}$

In 3D space, $\underline{\widetilde{\boldsymbol{b}}} = \begin{pmatrix} \vec{\boldsymbol{b}}_1 & \vec{\boldsymbol{b}}_2 & \vec{\boldsymbol{b}}_3 & \widetilde{\boldsymbol{b}}_0 \end{pmatrix}$ where $\vec{\boldsymbol{b}}_1, \vec{\boldsymbol{b}}_2, \vec{\boldsymbol{b}}_3$ are the unit vectors that form the basis of the model frame and $\widetilde{\boldsymbol{b}}_{\mathbf{0}}$ is the origin of the model frame.

Since we know that $\underline{\widetilde{\boldsymbol{w}}}\ \underline{\mathsf{M}} = \underline{\widetilde{\boldsymbol{b}}}$, we can see that the first 3 columns of the Model matrix must be the coordinates of $\vec{\boldsymbol{b}}_1, \vec{\boldsymbol{b}}_2, \vec{\boldsymbol{b}}_3$ expressed in the world frame $\underline{\widetilde{\boldsymbol{w}}}$ (respectively). Similarly, the last column of the Model matrix must be the coordinates of the origin of the model frame expressed in the world frame.

The Model matrix has the form of:

$$\underline{\mathsf{M}} = \begin{pmatrix} a & d & g & j \\ b & e & h & k \\ c & f & i & l \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Take-home question 3: Think about why the values in the last row are (0 0 0 1).

1.7.2 The View $\underline{\mathsf{V}}$ and Camera $\underline{\mathsf{C}}$ Matrices

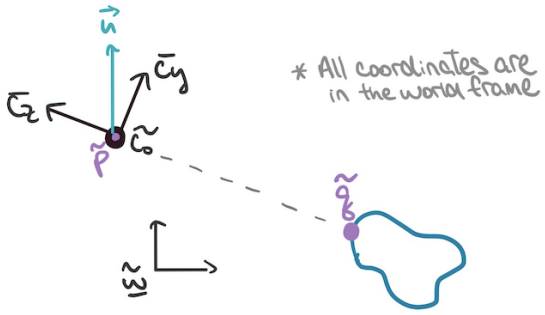
Note: In Three.js, we can use the lookAt() function to find the View matrix, which is the inverse of the Camera matrix.

The Camera matrix has the following form:

$$\underline{\mathsf{C}} = \begin{pmatrix} \bar{\mathsf{c}}_x & \bar{\mathsf{c}}_y & \bar{\mathsf{c}}_z & \bar{\mathsf{c}}_0 \end{pmatrix}$$

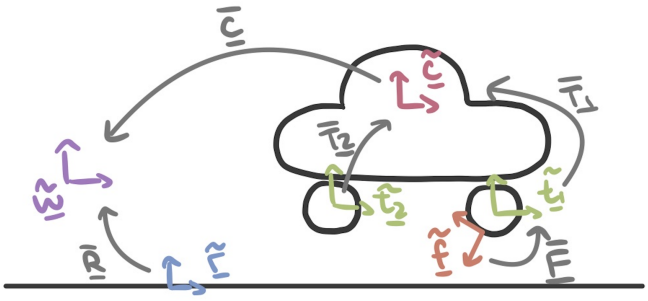
To find each column in $\underline{\mathsf{C}}$:

1. The origin of the camera frame ($\bar{\mathsf{c}}_0$) is its coordinates in the world frame: $\bar{\mathsf{c}}_0 = \bar{\mathsf{p}}_w$
2. By convention, $\vec{\boldsymbol{c}}_z$ points away from the direction it is looking at. Thus, $\bar{\mathsf{c}}_z = normalize(\bar{\mathsf{p}} - \bar{\mathsf{q}})$
3. Pick a (user defined) vector $\vec{\boldsymbol{u}}$ to be the "up vector".
4. Pick $\vec{\boldsymbol{c}}_x$ to be perpendicular to the $\vec{\boldsymbol{u}}$ and $\vec{\boldsymbol{c}}_z$: $\bar{\mathsf{c}}_x = normalize(\bar{\mathsf{u}} \times \bar{\mathsf{c}}_z)$
5. Calculate $\vec{\boldsymbol{c}}_y$: $\bar{\mathsf{c}}_y = \bar{\mathsf{c}}_z \times \bar{\mathsf{c}}_x$



1.8 Scene Graphs and Hierarchies of Transformations

In Assignment 1, we placed several objects in the scene, such as an armadillo and a sphere. However, they are not connected in any way. In many cases, we want objects to move together, as shown in the following example:

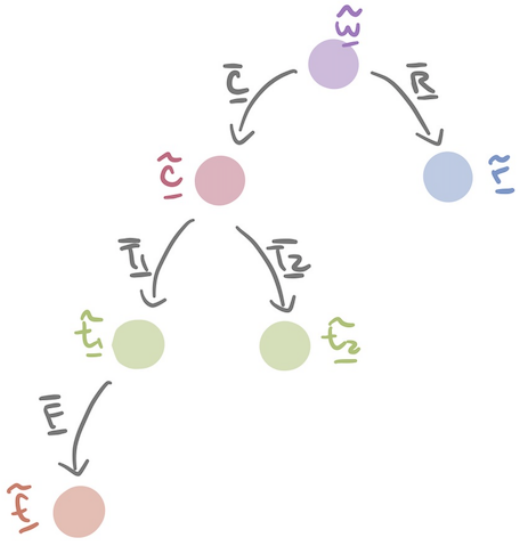


In this scene, a car is moving on the road. The tires of the car are moving along with the car, but they are also rotating. On one of the tires, there is a fly that is rotating with the tire.

Ex. Suppose we have a vertex on the fly with coordinates $\overline{\mathsf{p}}_f$ in the fly frame $\widetilde{\boldsymbol{f}}$. How can we find its coordinates $\overline{\mathsf{p}}_w$ in the world frame $\widetilde{\boldsymbol{w}}$ and what is the model matrix of the fly?

$\longrightarrow$ Answer: $\overline{\mathsf{p}}_w = \overline{\mathsf{C}}\ \overline{\mathsf{T}_1}\ \overline{\mathsf{F}}\ \overline{\mathsf{p}}_f$, where $\overline{\mathsf{C}}\ \overline{\mathsf{T}_1}\ \overline{\mathsf{F}}$ is the model matrix of the fly.

A scene graph is a data structure to represent the hierarchy of objects in the scene:



In three.js, the nodes in the graph above are called Object3D. Each Object3D stores its transformation to its parent `.matrix` and its transformation to the world frame `.matrixWorld`. In three.js, the 'world' is also called a scene. We can add an edge to the scene graph by calling `Object3D.add()`.

Take-home question 4: Is this matrixWorld the same as the model matrix?

1.9 Interpreting Chains of Transformations (Optional)

In the car example, we can express any point $\widetilde{\boldsymbol{p}}$ in any of the frames:

$$\widetilde{\boldsymbol{p}} = \underline{\widetilde{\boldsymbol{w}}}\ \overline{\mathsf{p}}_w = \underline{\widetilde{\boldsymbol{c}}}\ \overline{\mathsf{p}}_c = \underline{\widetilde{\boldsymbol{t}}}\ \overline{\mathsf{p}}_t = \underline{\widetilde{\boldsymbol{f}}}\ \overline{\mathsf{p}}_f = ...$$

We have also seen that we can concatenate transformation matrices together:

$$\widetilde{\boldsymbol{p}} = \underline{\widetilde{\boldsymbol{w}}}\ \overline{\mathsf{p}}_w = \underline{\widetilde{\boldsymbol{w}}}\ \underline{\overline{\mathsf{C}}}\ \underline{\overline{\mathsf{T}_1}}\ \underline{\overline{\mathsf{F}}}\ \overline{\mathsf{p}}_f$$

There are two ways of interpreting this:

1. Transformation of coordinates from one frame to another:

$$\widetilde{\boldsymbol{p}} = \underline{\widetilde{\boldsymbol{w}}}\ (\underline{\overline{\mathsf{C}}}(\underline{\overline{\mathsf{T}_1}}(\underline{\overline{\mathsf{F}}}\ \overline{\mathsf{p}}_f))) = \underline{\widetilde{\boldsymbol{w}}}\ \underline{\overline{\mathsf{C}}}\ \underline{\overline{\mathsf{T}_1}}\ \overline{\mathsf{p}}_t = \underline{\widetilde{\boldsymbol{w}}}\ \underline{\overline{\mathsf{C}}}\ \overline{\mathsf{p}}_c = \underline{\widetilde{\boldsymbol{w}}}\ \overline{\mathsf{p}}_w$$

2. Moving any point (or vector) in a frame:

$$\widetilde{\boldsymbol{p}} = (((\underline{\widetilde{\boldsymbol{w}}}\ \underline{\overline{\mathsf{C}}})\underline{\overline{\mathsf{T}_1}})\underline{\overline{\mathsf{F}}})\ \overline{\mathsf{p}}_f = \underline{\widetilde{\boldsymbol{c}}}\ \underline{\overline{\mathsf{T}_1}}\ \underline{\overline{\mathsf{F}}}\ \overline{\mathsf{p}}_f = \underline{\widetilde{\boldsymbol{t_1}}}\ \underline{\overline{\mathsf{F}}}\ \overline{\mathsf{p}}_f = \underline{\widetilde{\boldsymbol{f}}}\ \overline{\mathsf{p}}_f$$

1.10 Transformation Sequences and Matrices

It is very important to pay attention to the order in which matrices are multiplied, because matrix multiplication is not commutative. We often apply sequences of transformations, with the order specified by the user or the problem. For example, if a car is first translated and then rotated, then the translation transformation must be applied before the rotation.

However, sometimes (e.g., when defining a Object3D), parts of a transformation matrix maybe specified separately. E.g., the position of the frame's origin can be set by setting an Object3D's `.position` property, and the rotation maybe set by changing the `.rotation` property. In this case, the order is not the order in which these properties were set, but we use a standard pattern for a series of simultaneous transformations:

$$\underline{\overline{\mathsf{M}}} = \underline{\overline{\mathsf{T}}}\ \underline{\overline{\mathsf{R}}}\ \underline{\overline{\mathsf{S}}}$$

...where $\underline{\overline{\mathsf{T}}}$ is the translation matrix, $\underline{\overline{\mathsf{R}}}$ is the rotation matrix, and $\underline{\overline{\mathsf{S}}}$ is the scaling matrix. By default, the `matrixAutoUpdate` propery is set to true, and Three.js will recalculate the matrix, in the same order, when any part is changed.

See `https://threejs.org/docs/#manual/en/introduction/Matrix-transformations`.

1.10.1 Translation

Translations are affine transformations. The translation matrix has the form:

$$\underline{\overline{\mathsf{T}}} = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

...where t_x, t_y and t_z are the units of translation in the x, y and z directions, respectively.

Take-home question 5: Why are translations considered affine transformations?

1.10.2 Scaling

Scaling operations are linear transformations. The scaling matrix has the form:

$$\underline{\overline{\mathsf{S}}} = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

...where s_x, s_y and s_z represent the amount of scaling in the x, y and z directions, respectively.

1.10.3 Rotation

Rotation operations are also linear transformations. The rotation matrix has the form:

$$\underline{\overline{\mathsf{R}}} = \begin{pmatrix} a & d & g & 0 \\ b & e & h & 0 \\ c & f & i & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

An important property of rotation matrices is that the operation preserves dot products between vectors:

$$u = Ru$$

$$||u|| = ||v|| = ||Ru||$$

$$||u|| = \sqrt{u_x^2 + u_y^2 + u_z^2} = \sqrt{u^T u}$$

$$u^T u = v^T R^T R v = v^T v$$

From the equation above, we can see that $R^T R = I$.

The $3x3$ matrix in the top-left corner of the rotation matrix depends on the angle, axis, and direction of rotation. For example, for a rotation $+\theta$, the $3x3$ matrices are:

$$Rot_x : \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{pmatrix}$$

$$Rot_y : \begin{pmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{pmatrix}$$

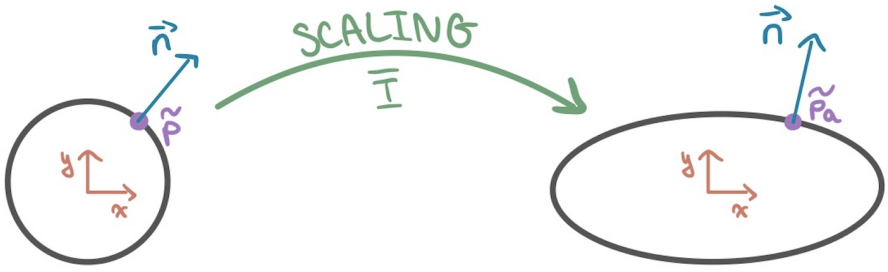
$$Rot_y : \begin{pmatrix} 0 & 0 & 1 \\ \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \end{pmatrix}$$

Take-home question 6: How would you write the $3x3$ matrix for a rotation in the $-z$ direction?

1.11 Normals

Remember: *Normals aren't normal.*

For example, take a point $\widetilde{\boldsymbol{p}}$ (with coordinates $\overline{\mathsf{p}}$) on a sphere. The normal at this point is denoted by $\vec{\boldsymbol{n}}$ (with coordinates $\overline{\mathsf{n}}$). Then, we scale the sphere along the x-axis using the matrix $\overline{\underline{\mathsf{T}}} = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ and we obtain an ellipse. We will denote the newly transformed point as $\widetilde{\boldsymbol{p}}_a$ (with coordinates $\overline{\mathsf{p}}_a$) and the normal as $\vec{\boldsymbol{n}}_a$ (with coordinates $\overline{\mathsf{n}}_a$), as shown in the diagram below.



To transform the coordinates of the original point to the new point, we can directly apply the transformation matrix $\overline{\underline{\mathsf{T}}}$:

$$\overline{\mathsf{p}}_a = \overline{\underline{\mathsf{T}}}\,\overline{\mathsf{p}}$$

However, we cannot directly apply $\overline{\underline{\mathsf{T}}}$ to transform $\overline{\mathsf{n}}$ into $\overline{\mathsf{n}}_a$. Why?

Intuitively, for a circle with its origin at its center, we can think of the coordinates $\overline{\mathsf{p}}$ as being a multiple of $\overline{\mathsf{n}}$ times its radius r:

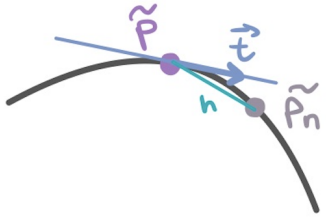
$$\overline{\mathsf{p}} = r\overline{\mathsf{n}} = \begin{pmatrix} rn_1 \\ rn_2 \\ rn_3 \end{pmatrix}$$

If we look at the diagram above, the point's x-coordinate will grow larger when we apply scaling in the x-direction using $\overline{\underline{\mathsf{T}}}$. If we do the same operation to the normal, we can expect the normal's x-coordinate to be scaled in the same way. However, we can see from the diagram that as the sphere becomes an ellipse, the normal's x-coordinate should actually become smaller as the surface stretches out. The normal's coordinates is no longer an integer multiple of the point's coordinates (not colinear anymore). Thus, we can conclude that normals do not transform the same way as points do.

To transform normals, we must first look at how tangents transform:

Take a point $\widetilde{\boldsymbol{p}}$ with tangent $\vec{\boldsymbol{t}}$ and a nearby point $\widetilde{\boldsymbol{p}}_n$. We can draw a line between them with length h. We can define the tangent as:

$$\vec{t} = \lim_{h \to 0} \frac{\widetilde{\boldsymbol{p}}_h - \widetilde{\boldsymbol{p}}}{h}$$



Since we know that points can be transformed by $\overline{\underline{\mathsf{T}}}$, we can say that the difference between two points can also be transformed by $\overline{\underline{\mathsf{T}}}$:

$$\overline{\underline{\mathsf{T}}}\,\vec{t} = \lim_{h \to 0} \frac{\overline{\underline{\mathsf{T}}}\,\widetilde{\boldsymbol{p}}_h - \overline{\underline{\mathsf{T}}}\,\widetilde{\boldsymbol{p}}}{h}$$

Therefore, we can conclude that tangents *do* transform the same way as points.

Going back to normals, we can define $\vec{\boldsymbol{n}}$ as the vector that satisfies:

$$\vec{\boldsymbol{n}} \cdot \vec{\boldsymbol{t}} = 0$$

$$\overline{\mathsf{n}}^T \overline{\mathsf{t}} = 0$$

After applying T, we want $\bar{\mathsf{n}}_a$ to satisfy:

$$\bar{\mathsf{n}}_a^T \bar{\mathsf{t}}_a = 0$$

...where $\bar{\mathsf{t}}_a = \overline{\underline{\mathsf{T}}}\bar{\mathsf{t}}$. We can write the equation above as:

$$(\bar{\mathsf{n}}_a^T \overline{\underline{\mathsf{T}}})\ \bar{\mathsf{t}} = 0$$

This tells us that:

$$\bar{\mathsf{n}}_a^T \overline{\underline{\mathsf{T}}} = \bar{\mathsf{n}}^T$$

$$\overline{\underline{\mathsf{T}}}^T \bar{\mathsf{n}}_a = \bar{\mathsf{n}}$$

$$\boxed{\bar{\mathsf{n}}_a = \overline{\underline{\mathsf{T}}}^{-T}\ \bar{\mathsf{n}}}$$

More specifically, since normals are vectors, the inverse transpose operation is only done on the upper left 3x3 matrix of $\overline{\underline{\mathsf{T}}}$:

$$T = \left(\begin{array}{c|c} A & 0 \\ \hline 0 & 1 \end{array}\right)$$

$$\boxed{\bar{\mathsf{n}}_a = \left(\begin{array}{c|c} A^{-T} & 0 \\ \hline 0 & 1 \end{array}\right) \bar{\mathsf{n}}}$$

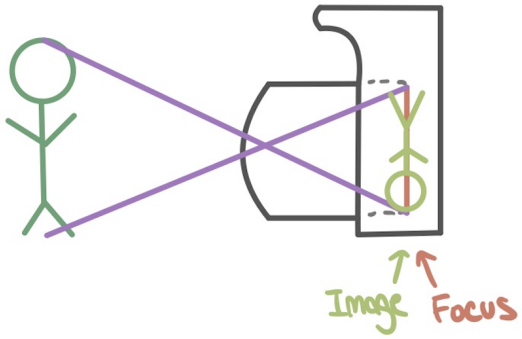
If $\overline{\underline{\mathsf{T}}}$ were the modelViewMatrix, then A^{-T} is the normalMatrix (supplied by Three.js).

2 Projection

2.1 Camera Models

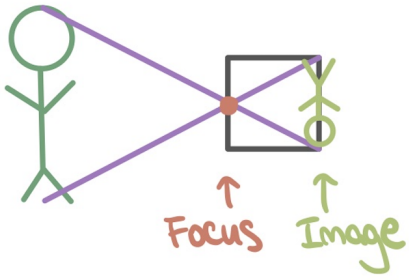
1. *Real Camera*

   A real camera has a finite depth of field that it can focus on, due to physical constraints.



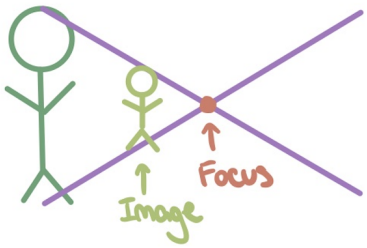
2. *Pinhole Camera*

   A pinhole camera has theoretically an 'infinite' depth of field. However, the image on the image plane is reversed.



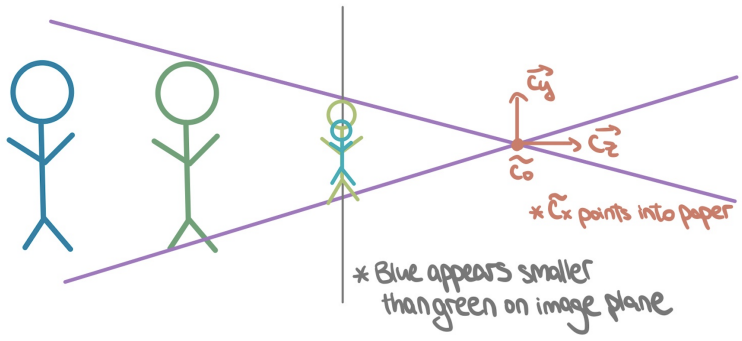
3. *Graphics Camera*

   A graphics camera is like a pinhole camera, but the image plane is placed in front of the focal point. As a result, the image has the same orientation as the original scene.



2.2 The Camera Frame

Foreshortening describes the effect that an object farther away would appear smaller to the human eye, as shown in the diagram below:

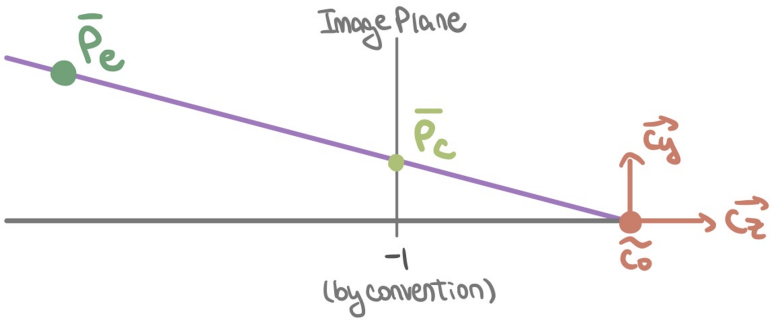


In graphics, to achieve the foreshortening effect so that the scene looks realistic, we can use perspective projection.

Take-home question 7: What is the difference between perspective and orthographic projection? What would the scene above look like in orthographic projection?

2.3 Perspective Projection - Part 1

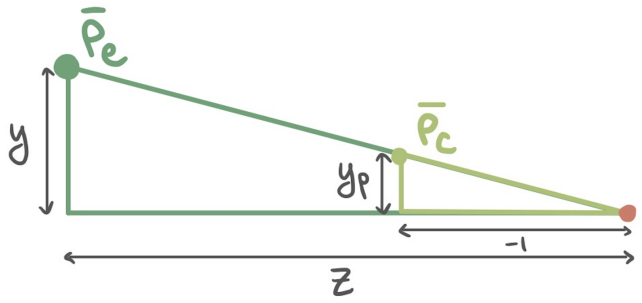
Suppose we have a point $\widetilde{\boldsymbol{p}}$ that is being captured by the graphics camera. Its coordinates in the camera frame are denoted by $\overline{\mathsf{p}}_e$ (also called the *eye coordinates*), and its coordinates after being projected onto the image plane frame are denoted by $\overline{\mathsf{p}}_c$ (that will eventually become the *clip coordinates*). For simplicity, we will initially assume that the near plane is at $n = -1$:



$$\overline{\mathsf{p}}_e = \begin{pmatrix} y \\ z \\ 1 \end{pmatrix} \qquad \overline{\mathsf{p}}_c = \begin{pmatrix} y_p \\ -1 \\ 1 \end{pmatrix}$$

Note: In this example, we only have the y and z axes for simplification. The same principle can be applied to the x axis for perspective projection.

To find y_p, we can use similar triangles:



$$\frac{y}{y_p} = \frac{z}{-1} \longrightarrow y_p = \frac{y}{-z}$$

We want to find a transformation matrix $\overline{\underline{\mathsf{P}}}$ that takes coordinates from the camera frame into the image plane frame. This transformation is linear if we use homogeneous coordinates:

$$\begin{pmatrix} y \\ z \\ 1 \end{pmatrix} \xrightarrow{\overline{\underline{\mathsf{P}}}} \begin{pmatrix} y/-z \\ -1 \\ 1 \end{pmatrix}$$

Homogeneous coordinates of points have an additional feature that is very useful, that exploits the last coordinate (that we have set to 1 so far). We can multiply all coordinates by an arbitrary constant w ($w \neq 0$), and consider the result to represent the same point! We can always recover the standard form by dividing all coordinates by zero.

$$\begin{pmatrix} y \\ z \\ 1 \end{pmatrix} \equiv \begin{pmatrix} wy \\ wz \\ w \end{pmatrix}$$

Then, we can rewrite the coordinates of the point in the image plane in the perspective projection equation by multiplying it with $-z$:

$$\begin{pmatrix} y \\ z \\ 1 \end{pmatrix} \xrightarrow{\overline{\underline{\mathsf{P}}}} \begin{pmatrix} y/-z \\ -1 \\ 1 \end{pmatrix} \equiv \begin{pmatrix} y \\ z \\ -z \end{pmatrix}$$

The transformation matrix of this perspective projection is:

$$\overline{\mathsf{p}}_c = \overline{\underline{\mathsf{P}}}\ \overline{\mathsf{p}}_e$$

$$\begin{pmatrix} y \\ z \\ -z \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} y \\ z \\ 1 \end{pmatrix}$$

There is still an issue with this transformation: The depth information is lost due to the matrix being singular (the last column hasb only 0's). Physically, all 3D points have been squashed on to the 2D image plane. To retain all of the 3D information, we can slightly modify $\overline{\underline{\mathsf{P}}}$ and obtain a new transformation $\overline{\underline{\mathsf{P}}}_u$ in an operation that is sometimes called "unhinging":

$$\overline{\mathsf{p}}'_c = \overline{\underline{\mathsf{P}}}_u\ \overline{\mathsf{p}}_e$$

$$\begin{pmatrix} y \\ 1 \\ -z \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} y \\ z \\ 1 \end{pmatrix}$$

We obtained this simple and elegant matrix since we assumed that the near plane is at $n = -1$. In the more general case of an arbitrary n, the unhinging matrix becomes

$$\overline{\underline{\mathsf{P}}}_u = \begin{pmatrix} -n & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{pmatrix}.$$

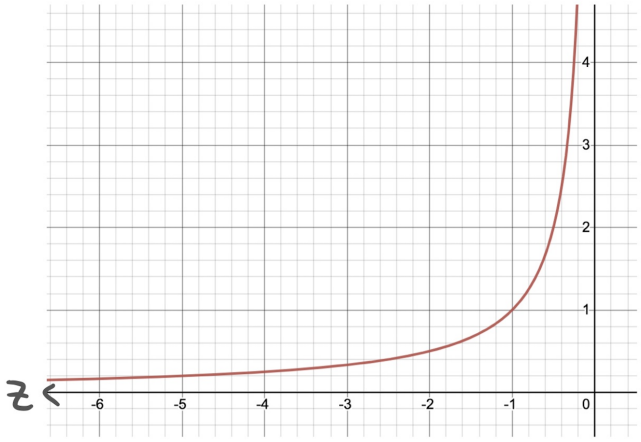
Finally, we want to perform **perspective divide** on the clip coordinates with the arbitrary constant $w = -z$ that was used in a previous step to obtain :

$$\overline{\mathsf{p}}_u = \overline{\mathsf{p}}'_c / w = \begin{pmatrix} -ny/-z \\ 1/-z \\ 1 \end{pmatrix}$$

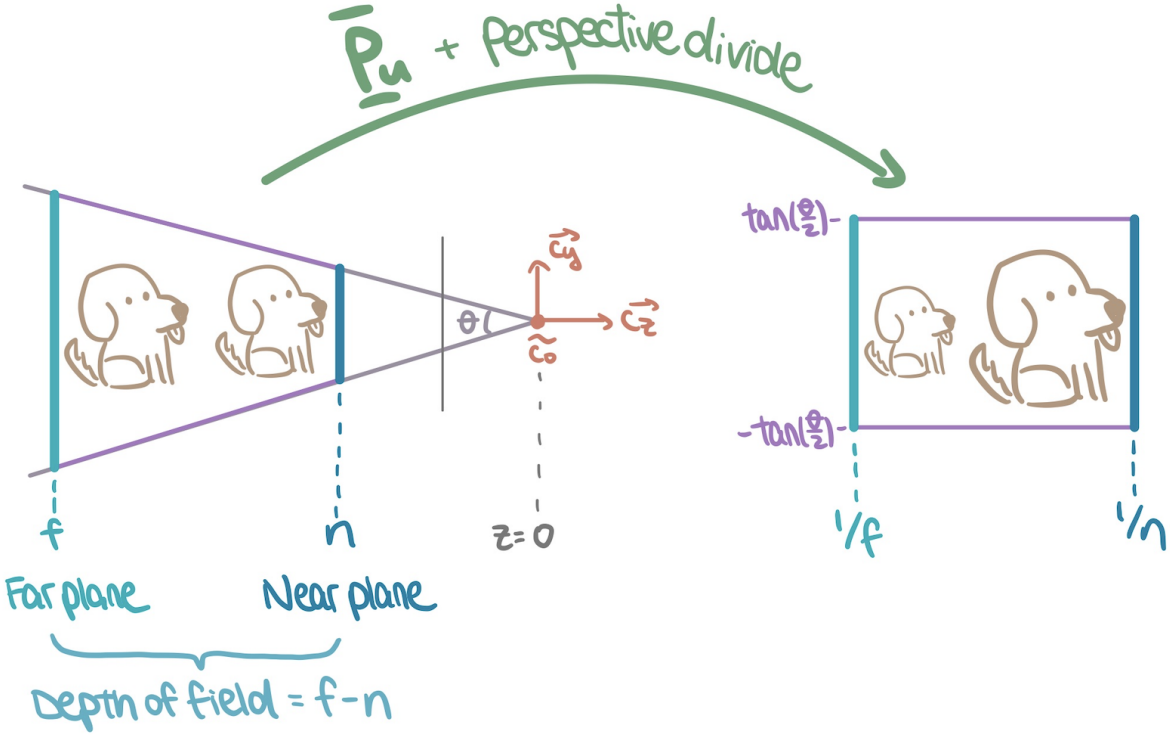
Now, the second entry of $\overline{\mathsf{p}}_u$ contains a *depth-like* value (monotonic function with depth), so depth information is now preserved. We can plot its relationship with depth as:

2.3.1 The View Frustum

A *frustum* is a truncated pyramid, which is the shape of the region that a perspective camera can capture. From the side, when only viewing the y and z axes (2D case), this region has a trapezoid



shape bounded by the far plane and the near plane. We only generate images for objects that are found within this region. After performing perspective projection on this region using $\underline{\overline{P}}_u$ and after performing perspective divide, the view frustum region becomes a parallel-sided box:



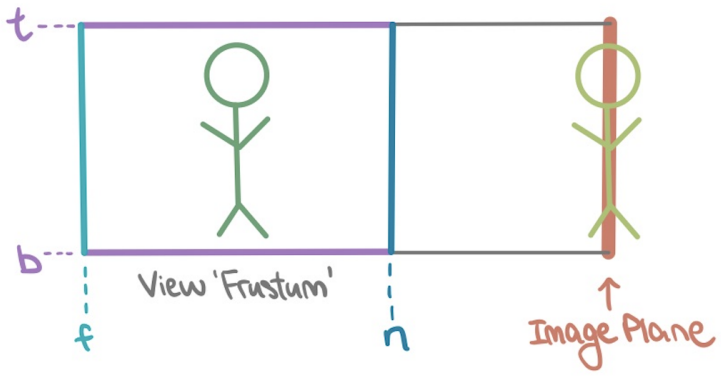
We can once again use the principle of similar triangles to verify that after perspective projection and perspective divide, all of the points situated at the top of the view frustum will have a constant value of $y = tan(\frac{\theta}{2})$. The same argument can be made for the bottom plane as well as for the left/right planes in the 3D case.

2.3.2 Side Note

In the next step of the pipeline, we will need to convert $\overline{\mathsf{p}}_u$ to Normalized Device Coordinates (NDC). We will first use the orthographic projection case to demonstrate how this is done, then come back to perspective projection in Section 2.5.

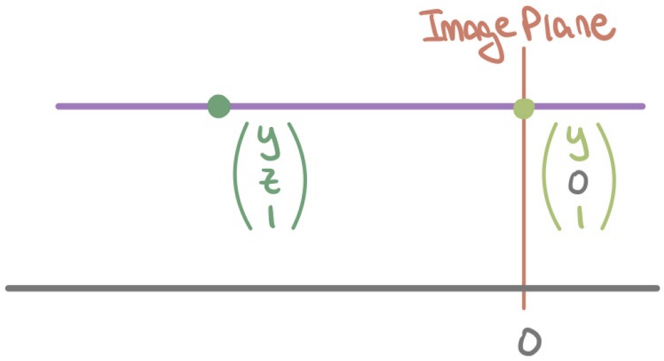
2.4 Orthographic Projection

In orthographic projection, we want to capture all light rays parallel to the z-axis on an image plane. Its view 'frustum' is simply a box. Since light rays are parallel to each other, if we take an object and change its depth within the view 'frustum', its apparent size on the image will not change (no foreshortening).



Suppose we have a 2D point $\widetilde{\boldsymbol{p}}$ with homogeneous coordinates $\begin{pmatrix} y \\ z \\ 1 \end{pmatrix}$ being captured by an orthographic camera.

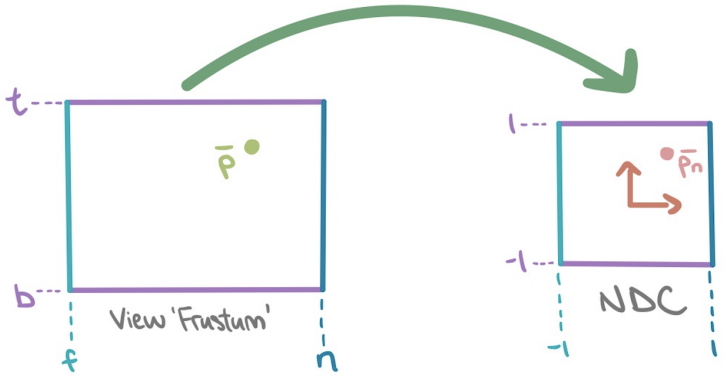
The point's coordinates on the image plane will have the same y-value, but its z-value will be 0, since the image plane is situated at $z = 0$ in the case of orthographic projection :



The projection matrix $\overline{\mathsf{Q}}$ that will take the point's original coordinates to its coordinates on the image plane will have the form:

$$\begin{pmatrix} y \\ z \\ 1 \end{pmatrix} \longrightarrow \begin{pmatrix} y \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} y \\ z \\ 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Since we only want to focus on a portion of the entire scene at a time, we define the limits of the view 'frustum' using f, n, t, and b. Also, we want to view this portion of the scene inside the limits of the 'frustum' on a standard sized screen, with **Normalized Device Coordinates (NDC)** ranging between $(-1, 1)$.



Suppose there is a point in the view frustum with coordinates $\overline{\mathsf{p}}$. The point's coordinates in NDC is $\overline{\mathsf{p}}_n$. We now want to find the transformation matrix that takes $\overline{\mathsf{p}} \longrightarrow \overline{\mathsf{p}}_n$.

First, consider the case where the point is at the center of the view frustum, then $\overline{\mathsf{p}} = \begin{pmatrix} \frac{t+b}{2} \\ \frac{n+f}{2} \\ 1 \end{pmatrix}$ (by inspection) and $\overline{\mathsf{p}}_n = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$. The transformation matrix $\overline{\underline{\mathsf{T}}}$ in this case performs a simple translation:

$$\overline{\underline{\mathsf{T}}} = \begin{pmatrix} 1 & 0 & \frac{-(t+b)}{2} \\ 0 & 1 & \frac{-(n+f)}{2} \\ 0 & 0 & 1 \end{pmatrix}$$

We can also view this translation transformation as *changing the origin of the camera to be in the center of the projected image.*

Then, we also want to account for scaling. Since NDC ranges from $(-1, 1)$, the final image has height 2. The scaling transformation matrix $\overline{\underline{\mathsf{S}}}$ is:

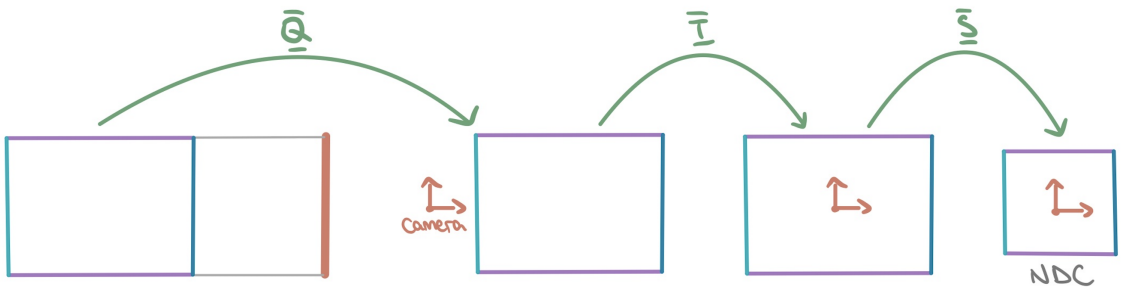
$$\underline{\overline{\mathsf{S}}} = \begin{pmatrix} \frac{2}{t-b} & 0 & 0 \\ 0 & \frac{2}{n-f} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Combining $\underline{\overline{\mathsf{T}}}$ and $\underline{\overline{\mathsf{S}}}$, we obtain the transformation from the view frustum to NDC:

$$\overline{\mathsf{p}}_n = \underline{\overline{\mathsf{S}}}\ \underline{\overline{\mathsf{T}}}\ \overline{\mathsf{p}}$$

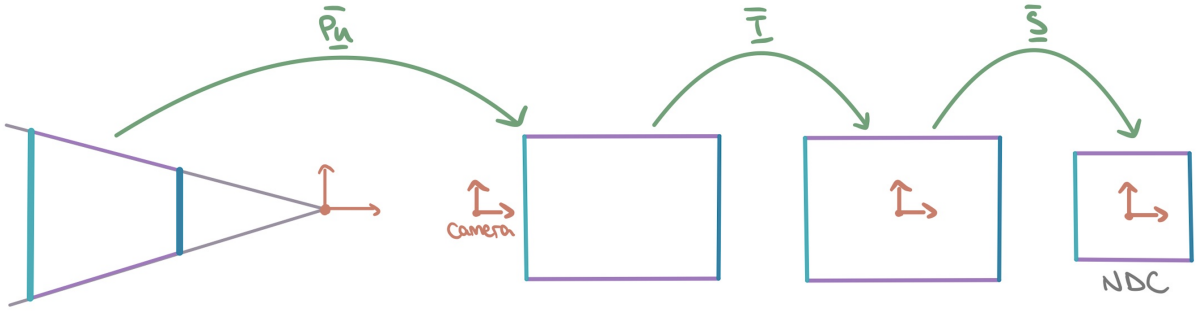
$$\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{2}{t-b} & 0 & \frac{-(t+b)}{t-b} \\ 0 & \frac{2}{n-f} & \frac{-(n+f)}{n-f} \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{t+b}{2} \\ \frac{n+f}{2} \\ 1 \end{pmatrix}$$

The chain of transformations can be visualized as:



2.5 Perspective Projection - Part 2

Continuing from Section 2.3, we now want to convert $\overline{\mathsf{p}}_u$ to NDC. The same transformations used in Orthographic Projection applies here and the chain of transformations can be visualized as:



...and the full perspective transform matrix $\underline{\overline{\mathsf{P}}}_f$ is:

$$\underline{\overline{\mathsf{P}}}_f = \underline{\overline{\mathsf{S}}}\ \underline{\overline{\mathsf{T}}}\ \underline{\overline{\mathsf{P}}}_u$$

However, please note that if n and f are the positions of the near and far planes of the perspective camera, the near plane of the orthographic box are mapped to $-1/n$ and $-1/f$, respectively. Substituting $n \to -1/n$ and $f \to -1/f$ in the formulas of the previous section, we get

$$\underline{\overline{\mathsf{T}}} = \begin{pmatrix} 1 & 0 & \frac{-(t+b)}{2} \\ 0 & 1 & \frac{n+f}{2nf} \\ 0 & 0 & 1 \end{pmatrix}, \quad \underline{\overline{\mathsf{S}}} = \begin{pmatrix} \frac{2}{t-b} & 0 & 0 \\ 0 & \frac{2nf}{n-f} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

and,

$$\underline{\overline{\mathsf{P}}}_f = \underline{\overline{\mathsf{S}}}\ \underline{\overline{\mathsf{T}}}\ \underline{\overline{\mathsf{P}}}_u = \begin{pmatrix} -\frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & -\frac{n+f}{n-f} & \frac{2nf}{n-f} \\ 0 & -1 & 0 \end{pmatrix}.$$