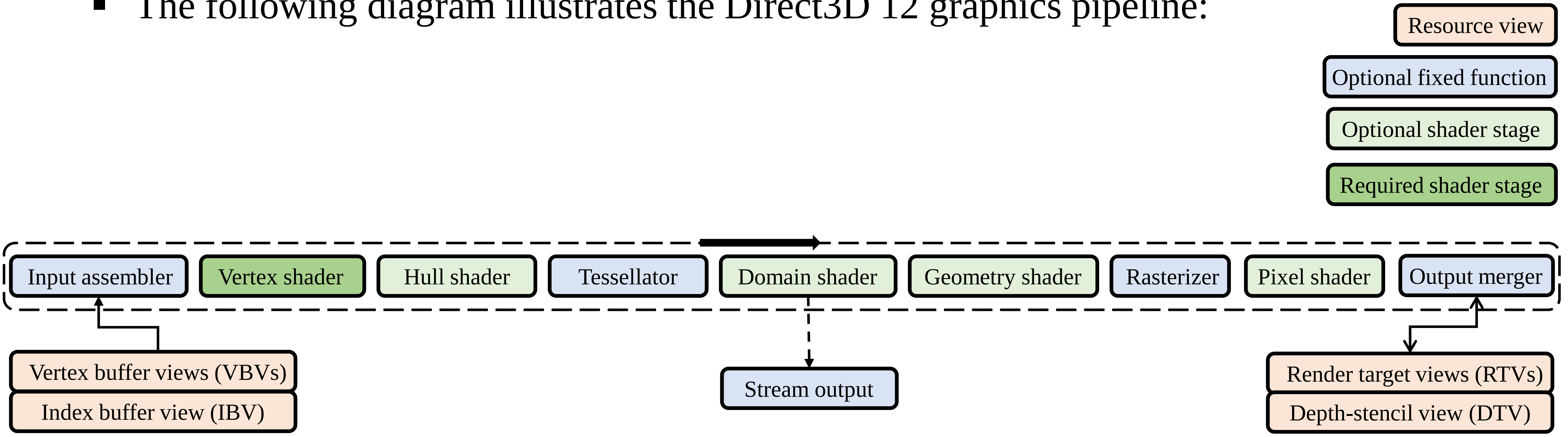
4. Input Assembler & Vertex Processing

Prof. HyeongYeop Kang
siamiz@khu.ac.kr
YouTube: HKang IIIXR LAB
IIIXR LAB

Direct3D 12 graphics Pipeline

In the GPU, rendering is done in a pipeline architecture, where the output of one stage is taken as the input for the next stage.

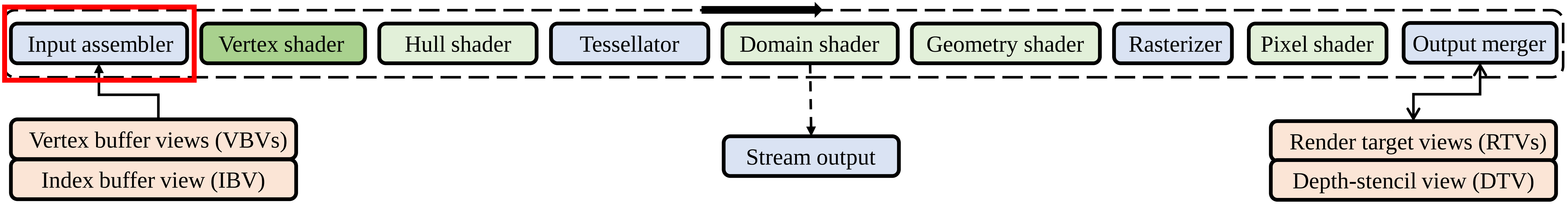
- A shader in the rendering pipeline is a synonym of a program.
- In contrast, Input Assembler, Tessellator, Rasterizer, and Output-Merger stages are hard-wired stages that perform fixed functions.
- The following diagram illustrates the Direct3D 12 graphics pipeline:



Input-Assembler Stage

Input-Assembler (IA) stage produces primitives or patches.

- IA stage reads vertex information from the user-specified buffers (e.g. vertex buffer and index buffer).
- Then, IA stage assemble the data into primitives.
  - Primitives are basic shapes such as triangles, lines, and points.
  - Through the IASetPrimitiveTopology method, users can set the primitive (topology) type.



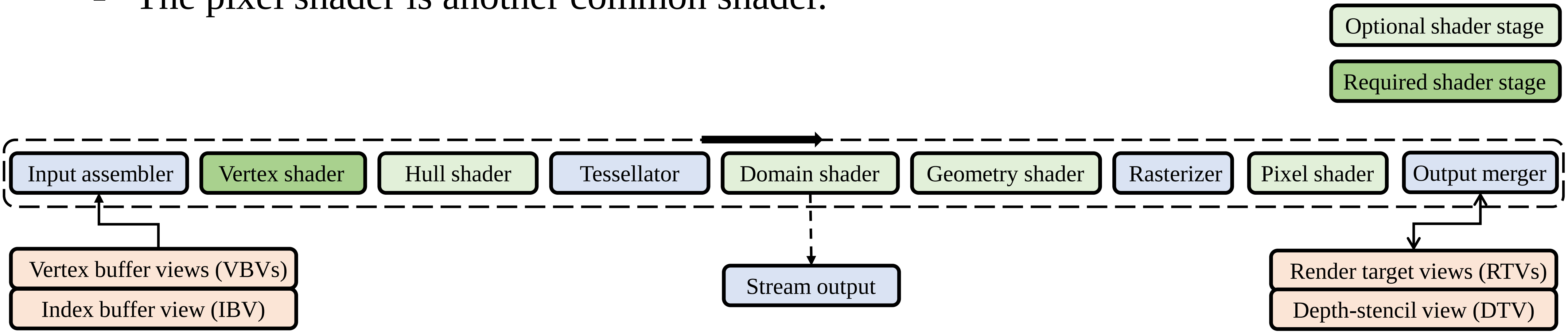
Vertex Shader

In graphics pipeline, certain sections are defined to be programmable.

- We call these programmable sections shaders.
- Although they can perform floating-point operations really fast, they must be designed according to some program guidelines (such as input and output structure).

There are five types of shaders:

- Among them, the vertex shader is a required shader stage.
- The pixel shader is another common shader.



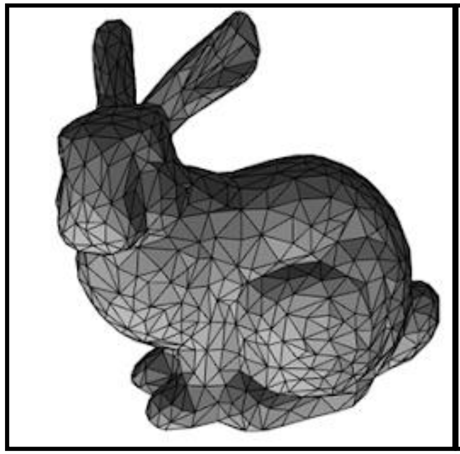
Vertex Shader

The vertex specifications are taken as the input for Vertex Shader in each frame.

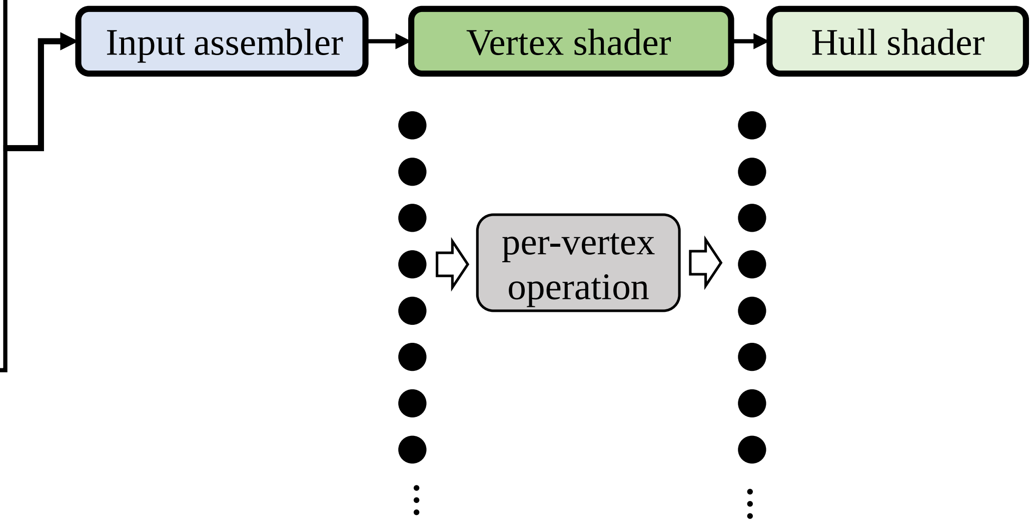
- Vertex specifications are provided by vertex and index buffers through the input-assembler.

- Vertex shaders are performed once for every vertex.

- Vertex shaders are mostly used for computing final positions of input vertices.



```
# OBJ file format with ext .obj
# vertex count = 2503
# face count = 4968
v -3.4101800e-003 1.3031957e-001 2.1754370e-002
v -8.1719160e-002 1.5250145e-001 2.9656090e-002
v -3.0543480e-002 1.2477885e-001 1.0983400e-003
v -2.4901590e-002 1.1211138e-001 3.7560240e-002
v -1.8405680e-002 1.7843055e-001 -2.4219580e-002
v 1.9067940e-002 1.2144925e-001 3.1968440e-002
v 6.0412000e-003 1.2494359e-001 3.2652890e-002
v -1.3469030e-002 1.6299355e-001 -1.2000020e-002
v -3.4393240e-002 1.7236688e-001 -9.8213000e-004
```

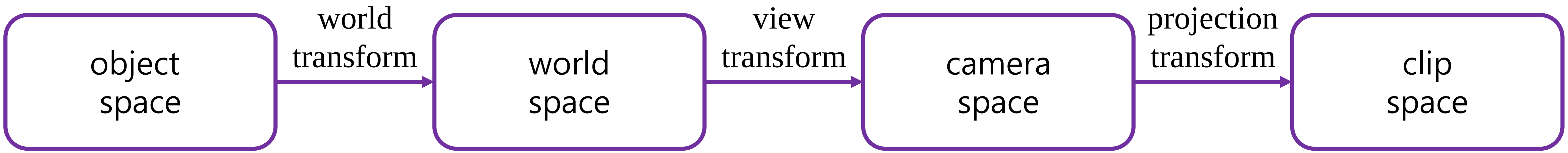


Vertex Shader

There are four major spaces and three transforms:

- Object space:
  - This is the coordinate system defined by an object's point of view.
  - The axes are rotated with the object.
  - This is also called model space.

- World space:
  - This is the coordinate system defining a 3D world (or universe).
  - 3D world includes multiple objects, characters, and cameras.



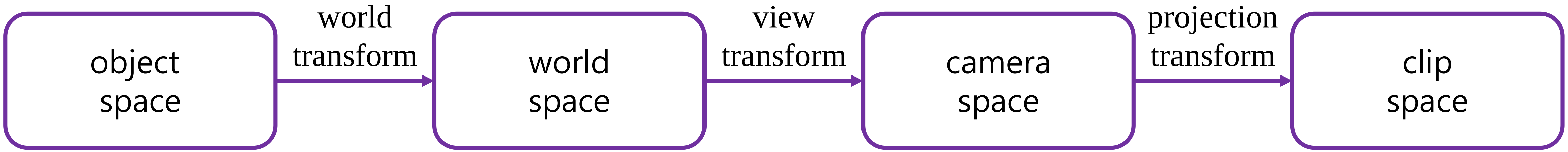
Spaces and transforms for the vertex shader.

Vertex Shader

There are four major spaces and three transforms:

- Camera space:
  - This is the coordinate system defined by an camera’s point of view.
  - The world space coordinate system is recalculated relative to the target camera.
  - In some application, this is also called eye space.

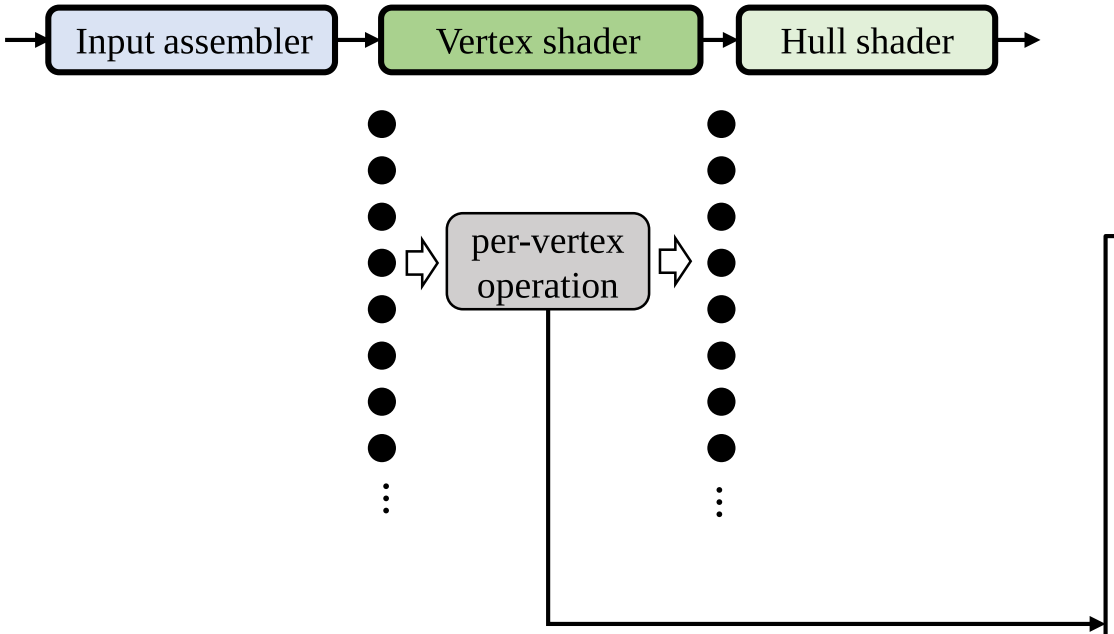
- Clip space:
  - This is the coordinate system defined by performing a projection transform to the points in the camera space.



Spaces and transforms for the vertex shader.

Vertex Shader

The vertex specifications are taken as the input for Vertex Shader in each frame.



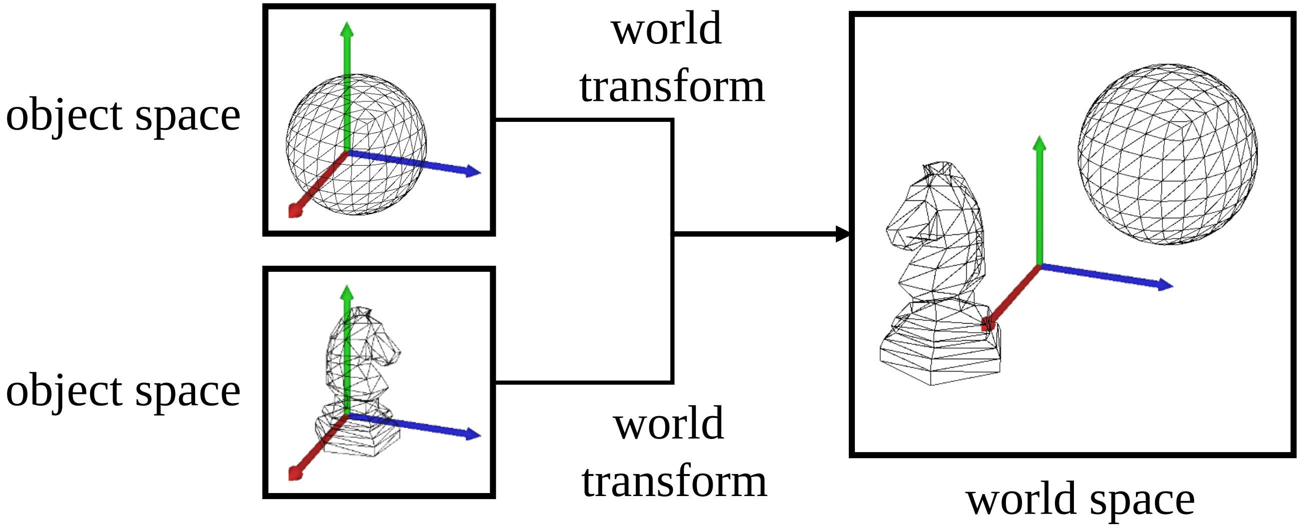

```
VS_OUTPUT main (VS_INPUT input)
{
   VS_OUTPUT Output;
   float4 pos = float4(input.vPos, 1.0f);

   pos = mul(pos, mWorld);
   pos = mul(pos, mView);
   pos = mul(pos, mProjection);
   Output.Position = pos;

   Output.Color = float4(input.vColor, 1.00f);
   return Output;
}
```

World Transform

The role of the world transform is to assemble all objects defined in their own object spaces into a single environment named the world space.

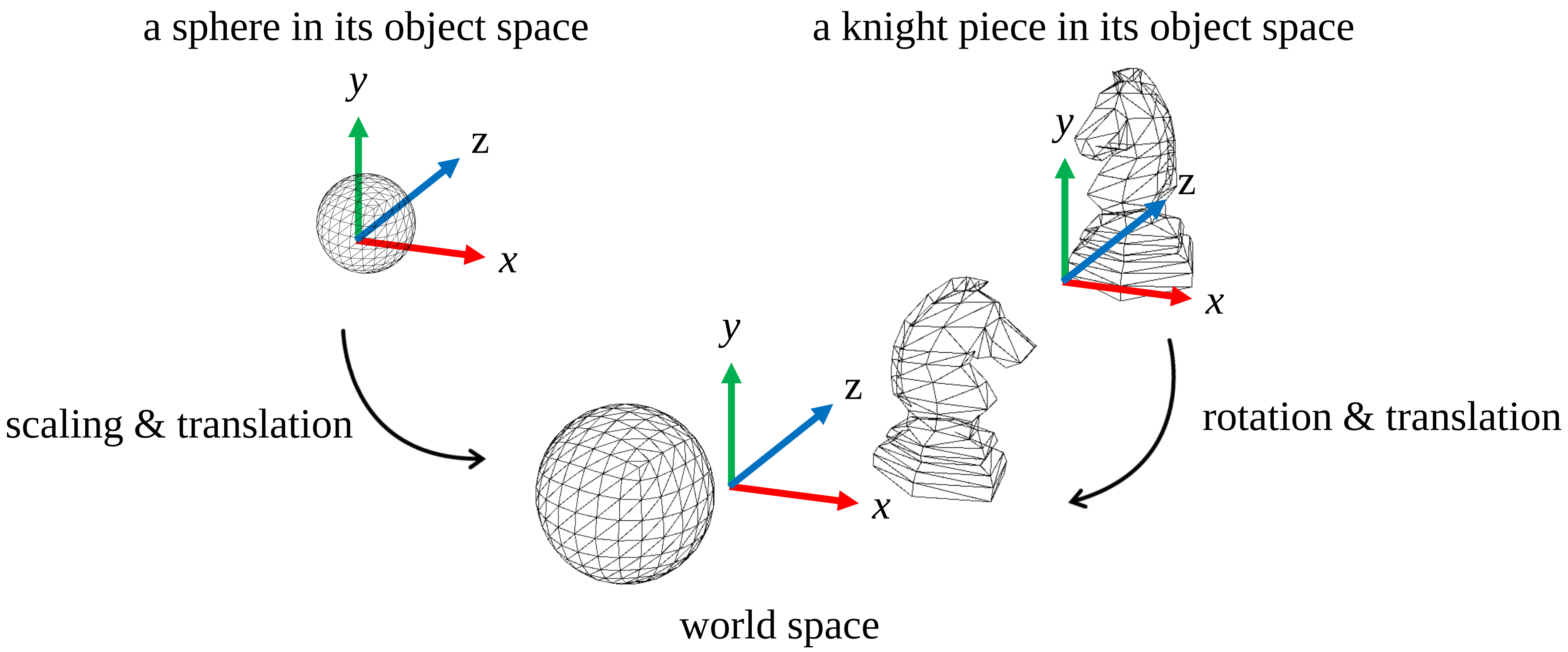


- The world matrix composed of affine transforms is denoted by $[\boldsymbol{L}|\boldsymbol{t}]$, where $\boldsymbol{L}$ is the 'combined' linear transform and $\boldsymbol{t}$ is the 'combined' translation.

World Transform

Object space vs. world space

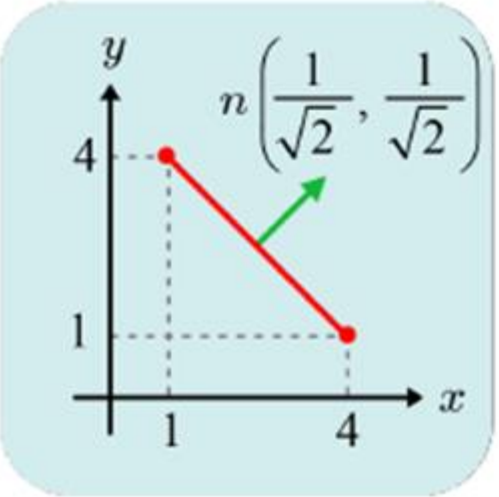
- The coordinate system used for creating an object is named object space.
- The object space for a model typically has no relationship to that of another model.
- The world transform ‘assembles’ all models into a single coordinate system called world space.



World Transform

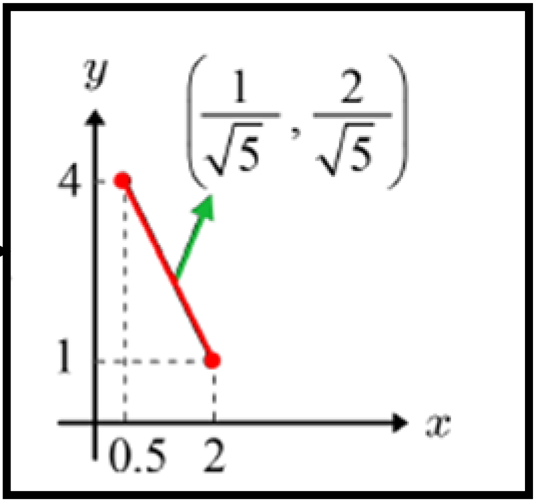
We learned how the vertex positions in the vertex array were transformed by world transform. How about vertex normal?

- If the world transform is in [$\boldsymbol{L}|t$], the normal is affected only by $\boldsymbol{L}$, not by t.
- If $\boldsymbol{L}$ includes *a non-uniform* scaling, it cannot be applied to surface normal.
  - Let $\boldsymbol{L}$ be $\begin{pmatrix} \frac{1}{2} & 0 \\ 0 & 1 \end{pmatrix}$.
  - The triangle's normal scaled by $\boldsymbol{L}$ is no longer orthogonal to the triangle scaled by $\boldsymbol{L}$.



$$\boldsymbol{L}_n = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix} \begin{pmatrix} \frac{1}{2} & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} \frac{1}{2\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix}$$

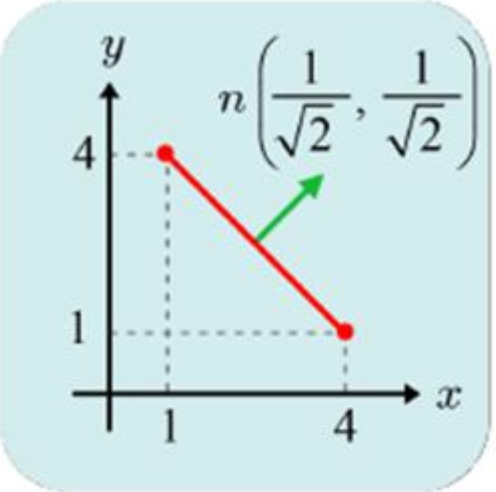
$$\widetilde{\boldsymbol{L}_n} = \begin{pmatrix} \frac{1}{\sqrt{5}} & \frac{2}{\sqrt{5}} \end{pmatrix}$$



World Transform

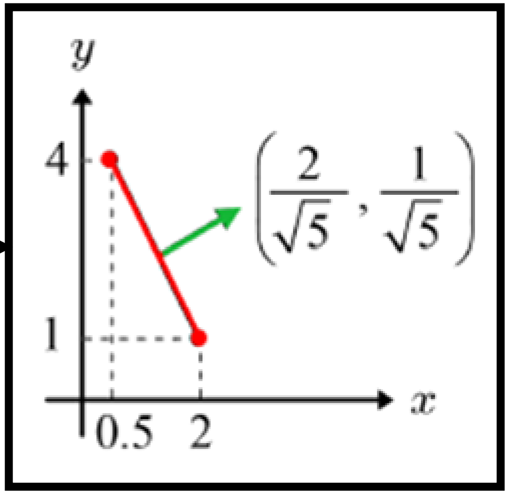
Instead, we have to use inverse transpose of $\boldsymbol{L}$, which is $(\boldsymbol{L}^{-1})^T$. It is simply denoted by $\boldsymbol{L}^{-T}$.

- If $\boldsymbol{L}$ does not contain any non-uniform scaling, n can be transformed by $\boldsymbol{L}$.
- However, $\boldsymbol{L}_n$ and $\boldsymbol{L}_n^{-T}$ have the same direction even though their magnitudes may be different.
- Therefore, we can transform n consistently by $\boldsymbol{L}^{-T}$ regardless of whether $\boldsymbol{L}$ contains a non-uniform scaling or not.
- Note that the normal transformed by $\boldsymbol{L}^{-T}$ will be finally normalized.



$$\boldsymbol{L}_n = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix} \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} \sqrt{2} & \frac{1}{\sqrt{2}} \end{pmatrix}$$

$$\widetilde{\boldsymbol{L}_n} = \begin{pmatrix} \frac{2}{\sqrt{5}} & \frac{1}{\sqrt{5}} \end{pmatrix}$$



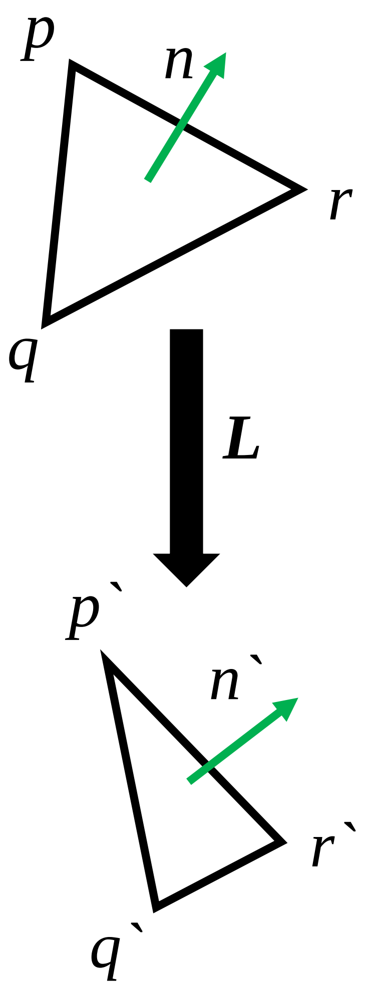
World Transform

Consider the triangle (p, q, r) and its normal n.

- Since the n is orthogonal to the vector connecting p and q, the dot product of two orthogonal vectors is zero:

$$(q - p)n^T = 0$$

where n, q, and p represent row vectors.



- A linear transform $\boldsymbol{L}$ transforms p and q to p` and $q$`, respectively.
  - $p\boldsymbol{L} = p$` and $q\boldsymbol{L} = q$`.
  - Therefore, $(q`\boldsymbol{L}^{-1} - p`\boldsymbol{L}^{-1})n^T = (q` - p`)\,\boldsymbol{L}^{-1}n^T = 0$

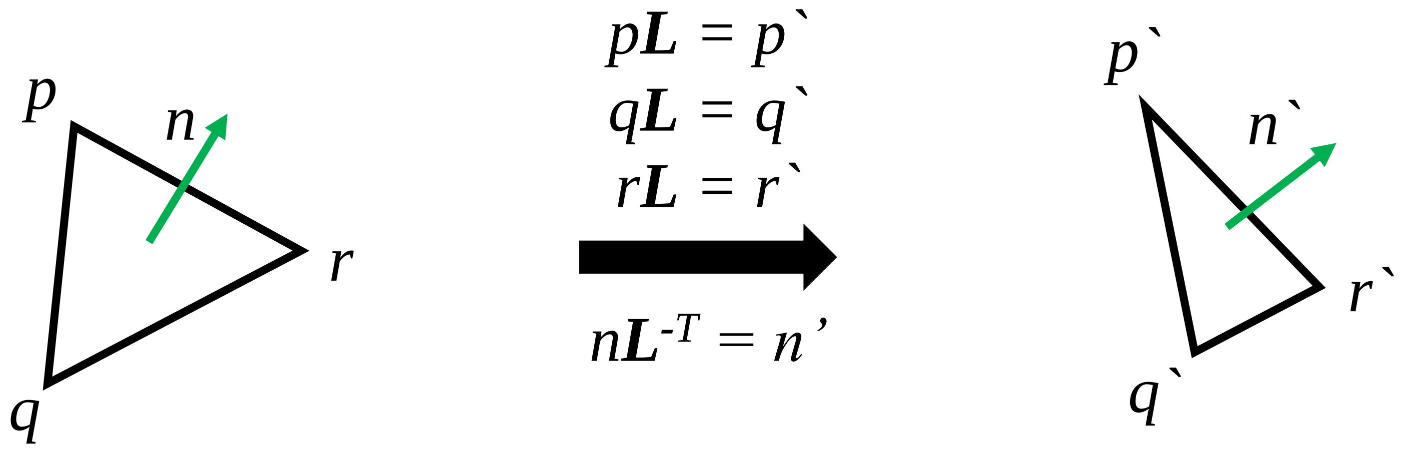
World Transform

Consider the triangle (p, q, r) and its normal n.

- Let’s obtain the transpose of $(q` – p`)\ \boldsymbol{L}^{-1}n^T$:

$$n\boldsymbol{L}^{-T}(q` – p`)^T = 0$$

- Now, we can see the vector $n\boldsymbol{L}^{-T}$ is orthogonal to $(q` – p`)$, and therefore it can be taken as the normal vector $n`$ of the transformed triangle.



View Transform

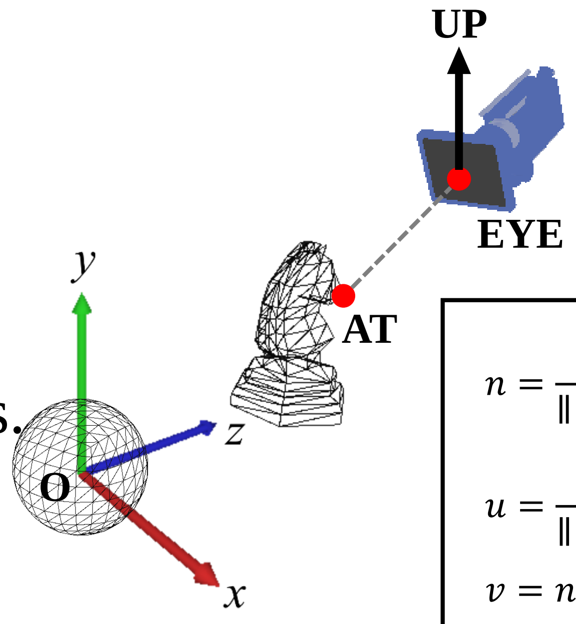
Since the camera is also an object, it is defined in the world space.

Camera pose specification in the world space is as follows:

- **EYE**: camera position
- **AT**: a reference point toward which the camera is aimed
- **UP**: view up vector that describes where the top of the camera is pointing.
  (In most cases, **UP** is set to the vertical axis, *y*-axis, of the world space.)

The camera space, $\begin{pmatrix} u \\ v \\ n \\ \mathbf{EYE} \end{pmatrix}$, can be created.

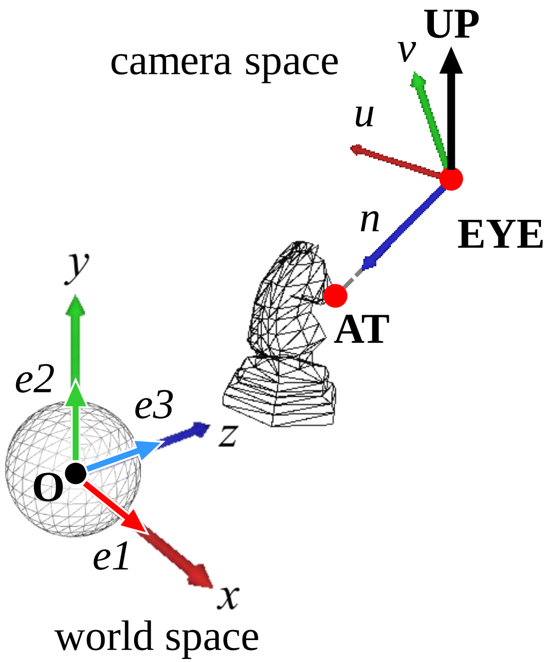
- Note that $\begin{pmatrix} u \\ v \\ n \end{pmatrix}$ is an orthonormal basis.



$$n = \frac{\mathbf{AT} - \mathbf{EYE}}{\| \mathbf{AT} - \mathbf{EYE} \|}$$

$$u = \frac{\mathbf{UP} \times n}{\| \mathbf{UP} \times n \|}$$

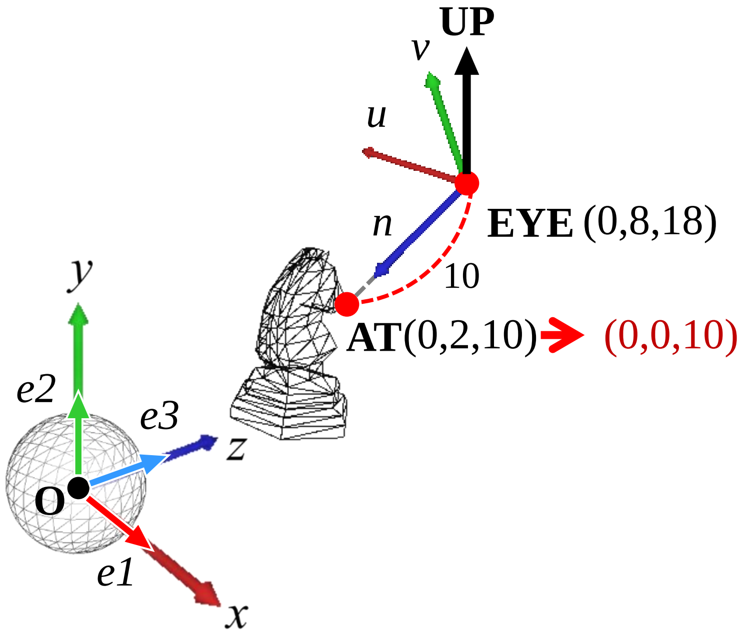
$$v = n \times u$$



View Transform

A point is given different coordinates in distinct spaces.

- The camera space $\begin{pmatrix} u \\ v \\ n \\ \mathbf{EYE} \end{pmatrix}$ and the world space $\begin{pmatrix} e_1 \\ e_2 \\ e_3 \\ \mathbf{O} \end{pmatrix}$.
- The mouth of the knight piece has the coordinates (0, 2, 10) in the world space.
- The mouth of the knight piece has the coordinates (0,0,10) in the camera space.



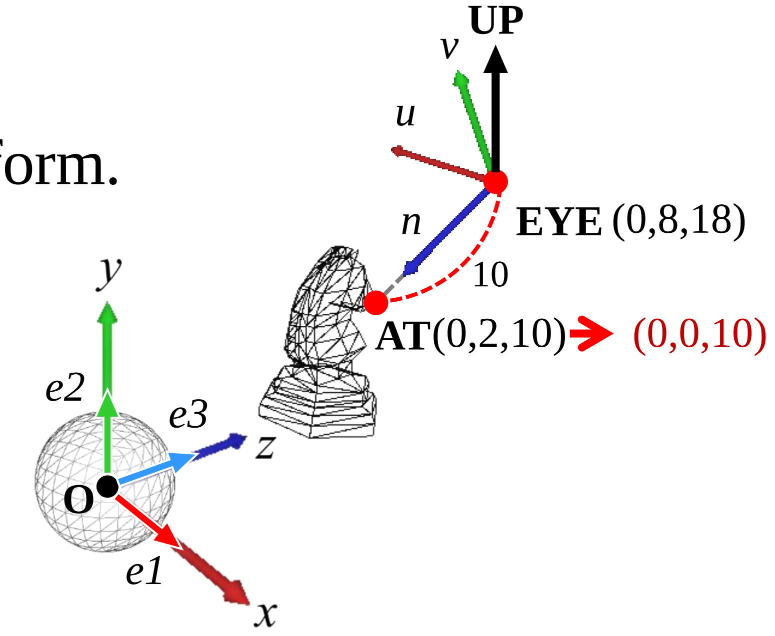
View Transform

A point is given different coordinates in distinct spaces.

- If all the world-space objects can be newly defined in terms of the camera space in the manner of the knight's mouth end, it becomes much easier to develop the rendering algorithms.

- In general, it is called space change.

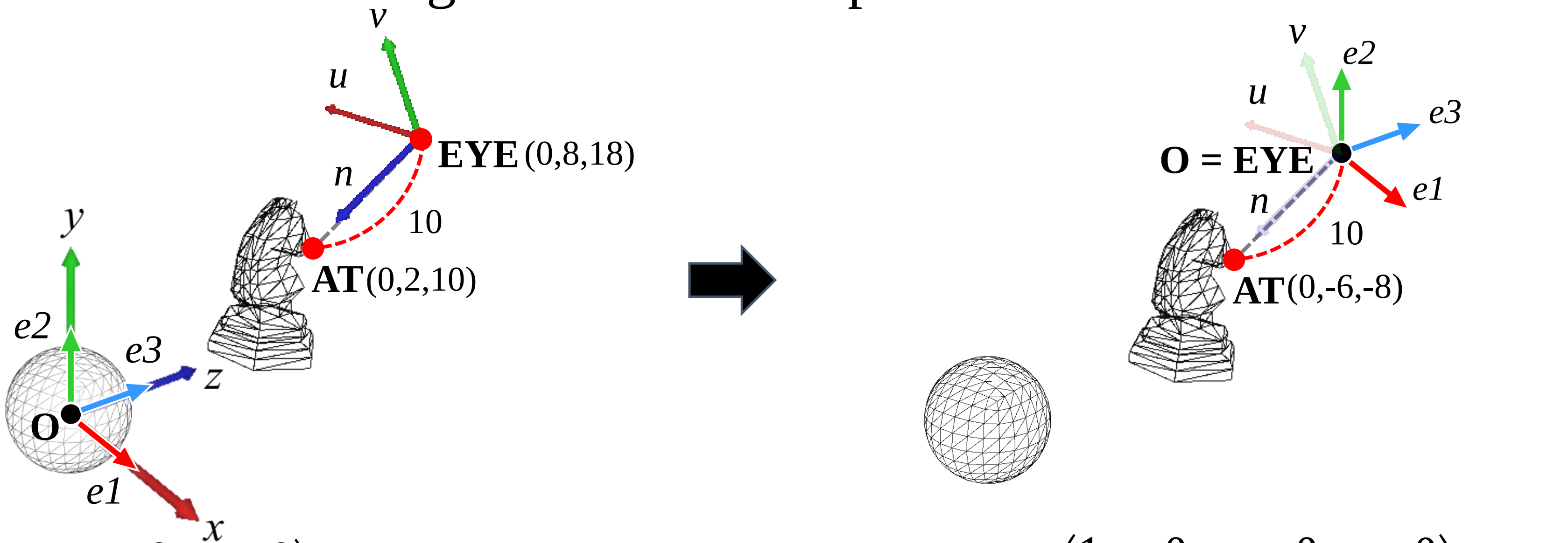
- The space change from $\begin{pmatrix} e_1 \\ e_2 \\ e_3 \\ \mathbf{O} \end{pmatrix}$ to $\begin{pmatrix} u \\ v \\ n \\ \mathbf{EYE} \end{pmatrix}$ is the view transform.



View Transform

The space change can be intuitively described by the process of superimposing the camera space, $(u, v, n, \mathbf{EYE})^T$, onto the world space, $(e_1, e_2, e_3, \mathbf{O})^T$.

- First, **EYE** is translated to the origin of the world space.



$$T = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -\mathbf{EYE}_x & -\mathbf{EYE}_y & -\mathbf{EYE}_z & 1 \end{pmatrix}$$

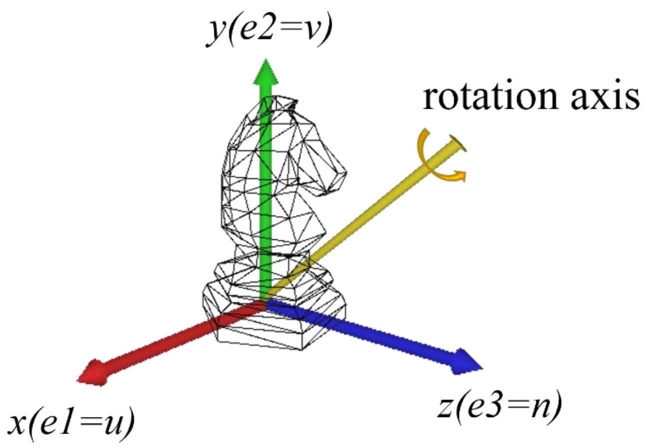
$$(0, 2, 10, 1)\, T = (0, 2, 10, 1) \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -8 & -18 & 1 \end{pmatrix} = (0, -6, -8, 1)$$

- The world space and the camera space now share the origin, due to translation.

View Transform

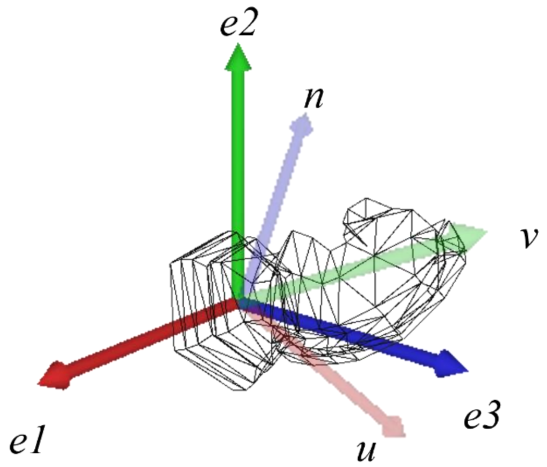
We then need a rotation R that transforms $\{u, v, n\}^T$ into $\{e_1, e_2, e_3\}^T$. It's called basis change.

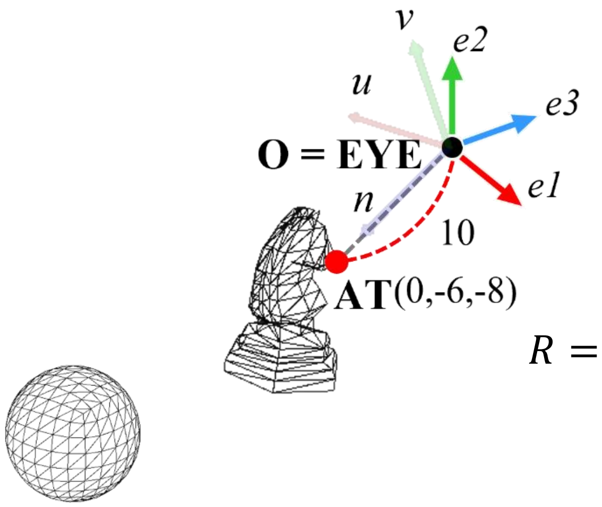
- As we learned before, u, v, and n fill the rows of the rotation matrix.



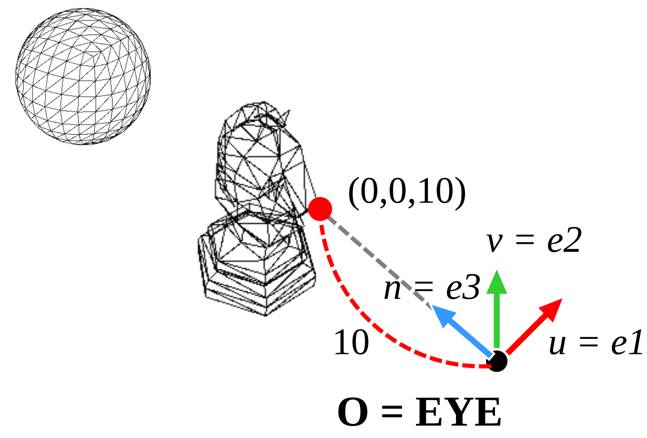
$$R = \begin{pmatrix} \boxed{u_x \quad u_y \quad u_z} \\ \boxed{v_x \quad v_y \quad v_z} \\ \boxed{n_x \quad n_y \quad n_z} \end{pmatrix}$$

$$R^{-1} = \begin{pmatrix} u_x & v_x & n_x \\ u_y & v_y & n_y \\ u_z & v_z & n_z \end{pmatrix}$$





$$R = (x\ y\ z\ 1)\begin{pmatrix} u_x & v_x & n_x & 0 \\ u_y & v_y & n_y & 0 \\ u_z & v_z & n_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = e_1$$



View Transform

The view matrix

- M_{view} is applied to all objects in the world space to transform them into the camera space.

$$M_{view} = TR$$

$$= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -\mathbf{EYE}_x & -\mathbf{EYE}_y & -\mathbf{EYE}_z & 1 \end{pmatrix} \begin{pmatrix} u_x & v_x & n_x & 0 \\ u_y & v_y & n_y & 0 \\ u_z & v_z & n_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} u_x & v_x & n_x & 0 \\ u_y & v_y & n_y & 0 \\ u_z & v_z & n_z & 0 \\ -u \cdot \mathbf{EYE} & -v \cdot \mathbf{EYE} & -n \cdot \mathbf{EYE} & 1 \end{pmatrix}$$