Advanced Animation Programming

GPR-450
Daniel S. Buckstein

Animating Rotations & Transforms
Week 4

License

Animating Rotations & Transforms

- Rotation matrices

- Frenet-Serret frames & applications

- Animating transformations

Rotation Matrices

- "RUDE" coordinate frame model:

$$T_{4\times4} = \begin{bmatrix} \hat{r} & \hat{u} & \hat{d} & \vec{e} \\ & & & \\ & \vec{0} & & 1 \end{bmatrix}$$

$\hat{r}$: Relative "right" vector

$\hat{u}$: Relative "up" vector

$\hat{d}$: Relative "direction" vector

$\vec{e}$: Relative "center" vector ('e' possibly from 'Einstein' or 'Euclid')

Rotation Matrices

- It is possible to construct the rotation part of a matrix for *any coordinate frame* given only two things:
- Position of object
- Target location (think of this as a "look-at" point)

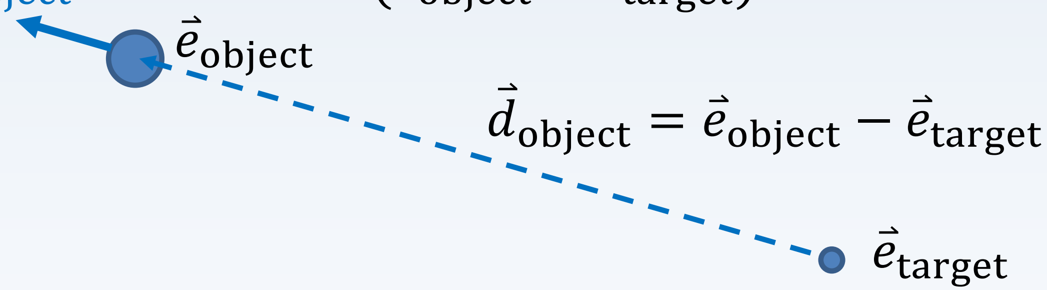
Position: $\vec{e}_{\text{object}}$

Target: $\vec{e}_{\text{target}}$

Rotation Matrices

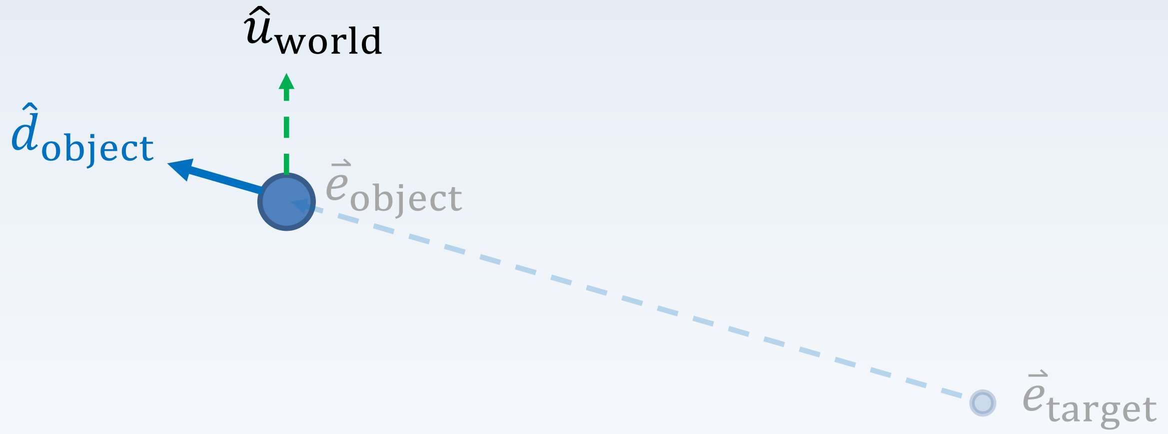
- In a ***right-handed system***, the *direction basis* is the vector *from the target to the object's position*:
- Since it is a basis vector… normalize it!!!

$$\hat{d}_{\text{object}} = \text{normalize}\left(\vec{e}_{\text{object}} - \vec{e}_{\text{target}}\right)$$



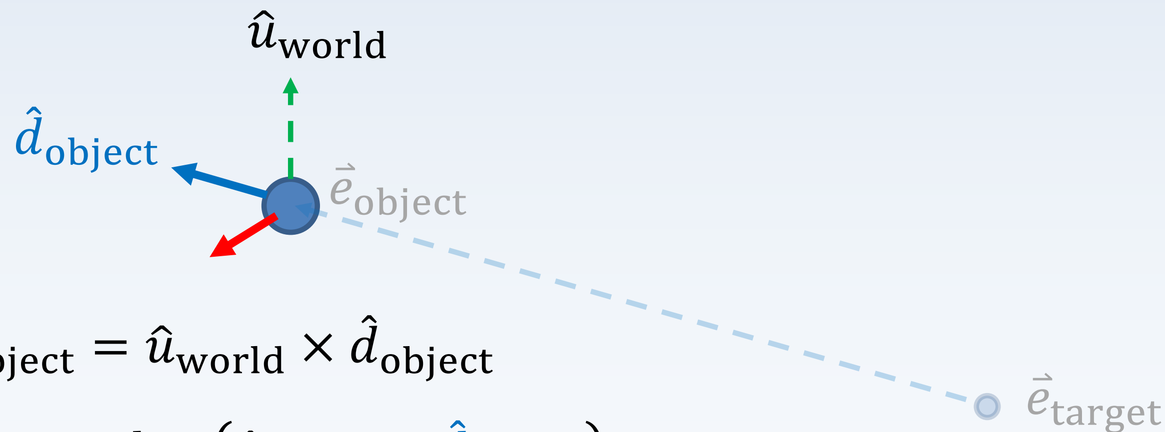
Rotation Matrices

- We find the *right* basis vector next
- First pick a "world up" vector
- (i.e. the "default" up basis vector)



Rotation Matrices

- We find the *right* basis vector next
- The *right basis* is the *cross product* of the *world up* and the *direction basis*:
  - Don't forget to normalize again…



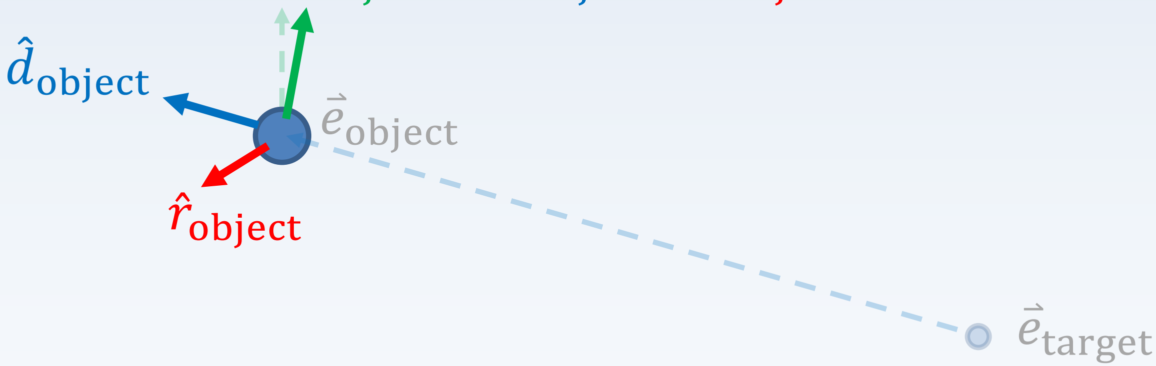
$$\vec{r}_{\text{object}} = \hat{u}_{\text{world}} \times \hat{d}_{\text{object}}$$

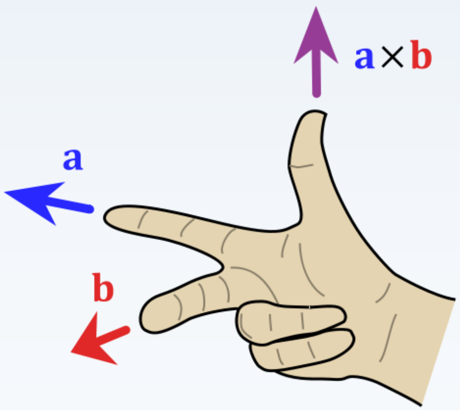
$$\hat{r}_{\text{object}} = \text{normalize}\left(\hat{u}_{\text{world}} \times \hat{d}_{\text{object}}\right)$$

Rotation Matrices

- Finally, the actual *up basis* is the cross product of the *direction* and *right* basis vectors!!!
  - (normalize is optional this time, both of the inputs are already normalized!)

$$\hat{u}_{\text{object}} = \hat{d}_{\text{object}} \times \hat{r}_{\text{object}}$$

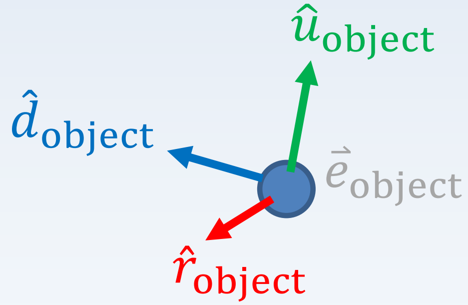




The right-hand rule: very useful for calculating relative directions!

Rotation Matrices

- These three basis vectors are the *columns* in a rotation matrix!
- They represent the *basis* for a coordinate frame relative to the parent frame!

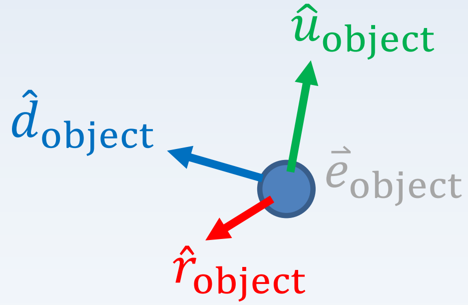


$$
\begin{aligned}
R &= [\hat{e}_0 \quad \hat{e}_1 \quad \hat{e}_2] \\
&\equiv [\,\hat{r} \quad \hat{u} \quad \hat{d}\,]
\end{aligned}
$$

Rotation Matrices

- "RUDE" coordinate frame model:

$$T_{4\times4} = \begin{bmatrix} \hat{r} & \hat{u} & \hat{d} & \vec{e} \\ & \vec{0} & & 1 \end{bmatrix}$$



Rotation Matrices

- Right-handed, ‘d’ is actually negative relative to the target:
- Think of it as “RUNE” instead (N is “negative direction”, could also be “normal”)

Rotation Matrices

- Always remember: ***everything is relative***.
- ***There is no absolute "up"***
- ***There is only "what you call up" and everything relative to it***
- Relative right, up and direction are called "***basis vectors***" (denoted by $\hat{e}_{\text{index}}$)

$$R = \begin{bmatrix} \hat{e}_0 & \hat{e}_1 & \hat{e}_2 \end{bmatrix}$$

Rotation Matrices

- Can be used to perform the same transformations as the formula:

$$\vec{v}' = R\vec{v} + \vec{t}$$

- Tangent basis coordinate frame model:

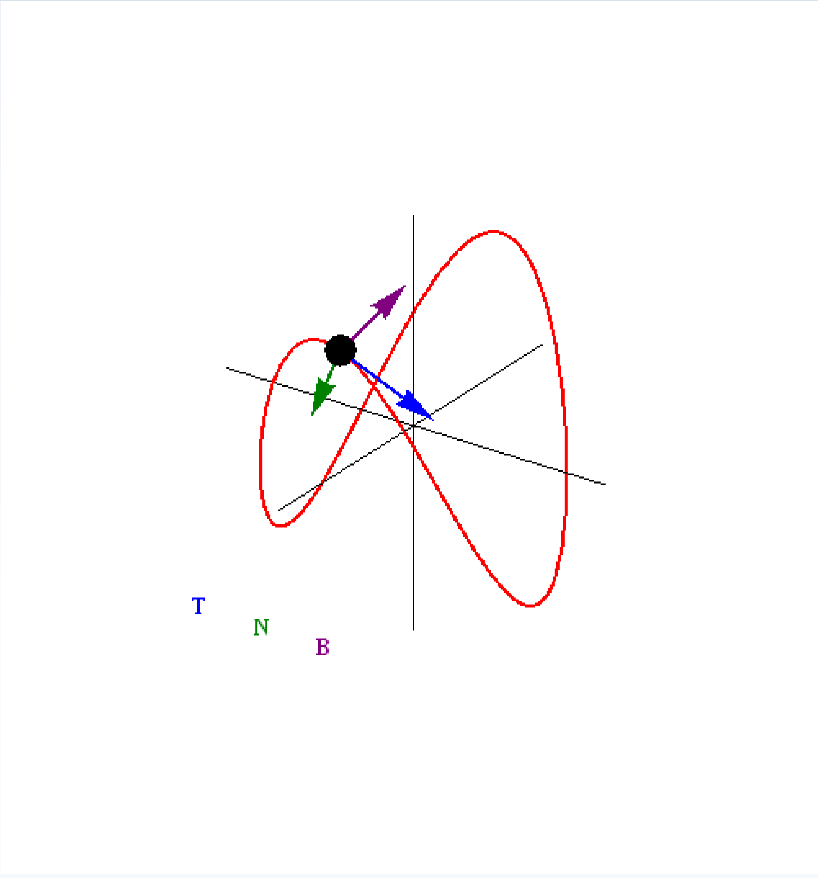
$$T_{4\times 4} = \begin{bmatrix} \hat{e}_0 & \hat{e}_1 & \hat{e}_2 & \vec{p} \\ & & & \\ & \vec{0} & & 1 \end{bmatrix}$$

Frenet-Serret Frames

- Application of the above: determining the orientation of an object on a curve!
- Basis vectors are:
- **Tangent**: rate of change along curve
- **Normal**: vector perpendicular to curve
- **Binormal***: vector perpendicular to normal and curve, required to have a full matrix

Frenet-Serret Frames

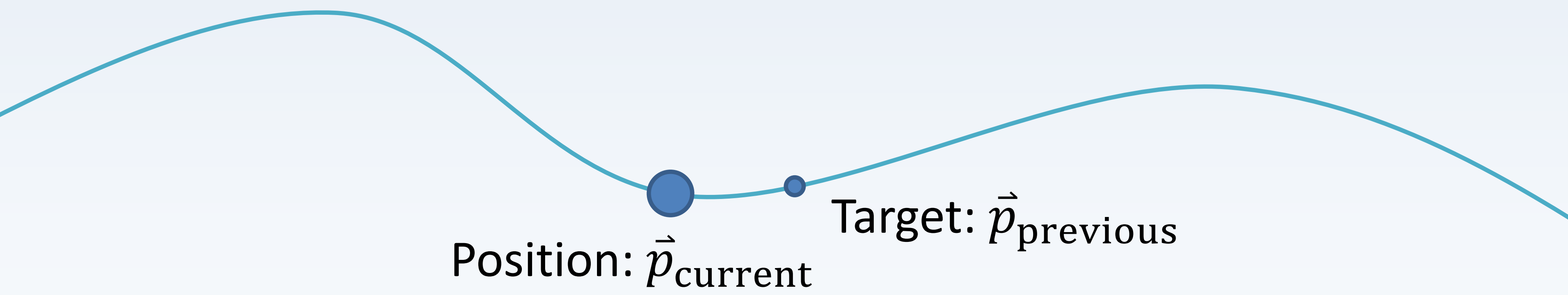
- Describes the full basis of an object on a curve
- A.K.A. “TNB frame”
- Ultimately, which way is “forward” (direction), “up” and “right” when comparing against curves
- *Continuous method* is calculus-heavy…



https://en.wikipedia.org/wiki/Frenet%E2%80%93Serret_formulas

Frenet-Serret Frames

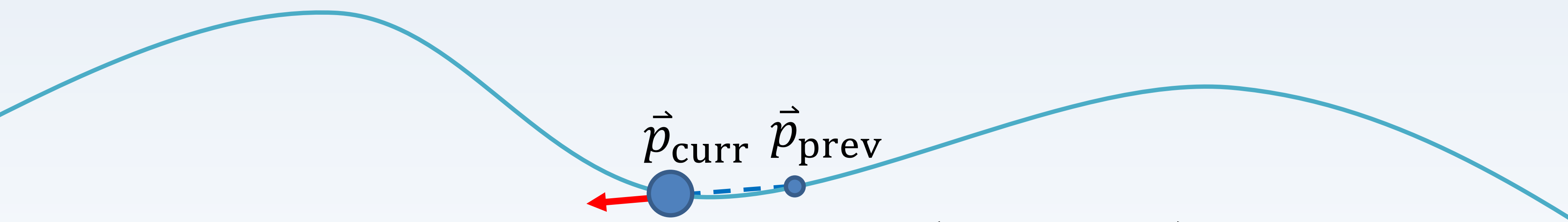
- *Discrete method*: calculate Frenet-Serret frame using slightly-modified "look-at" algorithm from before!
- Easy way: use the previous sample on the curve to calculate the "tangent"



Frenet-Serret Frames

- Normalize difference to get *tangent*

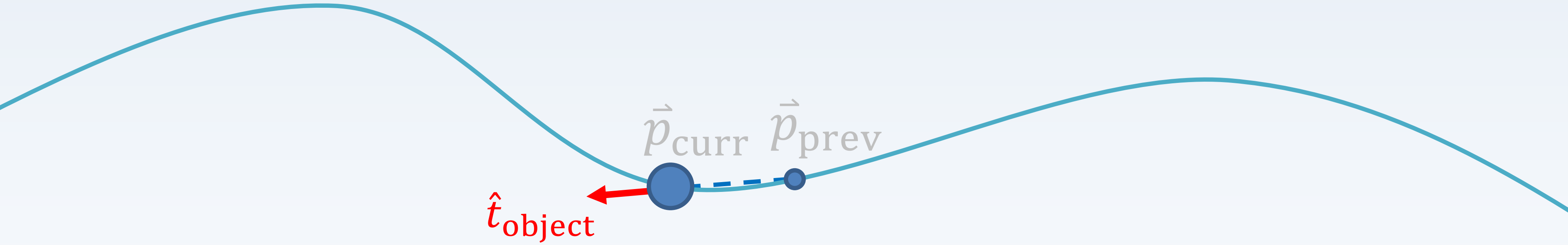
- …how are the other vectors calculated?



$$\hat{t}_{\text{object}} = \text{normalize}\left(\vec{p}_{\text{curr}} - \vec{p}_{\text{prev}}\right)$$

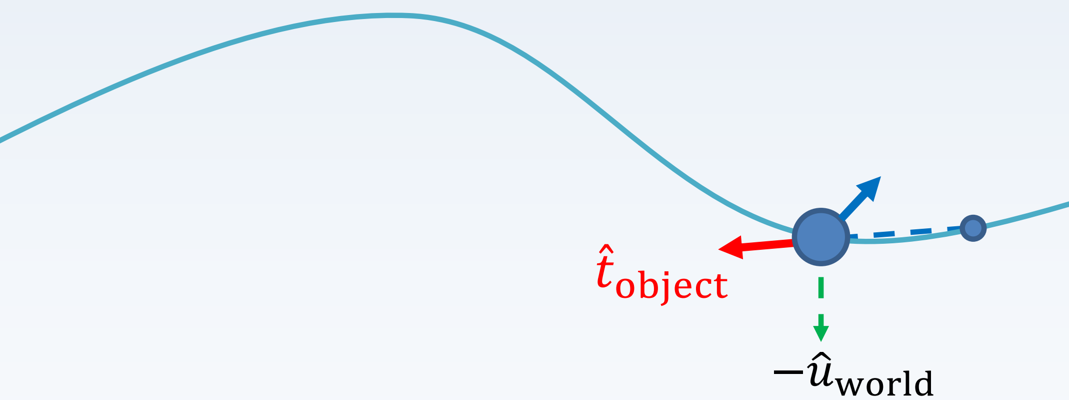
Frenet-Serret Frames

- *Frenet-Serret frames are on *curves*, which is why the basis order is TNB
  - Bi*normal* = 2nd *normal*
- 3D surfaces have the basis order TBN
  - Bi*tangent* = 2nd *tangent*



Frenet-Serret Frames

- Earlier we calculated the *third basis* first, followed by the *first basis*.
- Here we calculated the *first basis* then *third*.
- Thus, to get the correct results, we use the "world *down*" vector to calculate binormal:



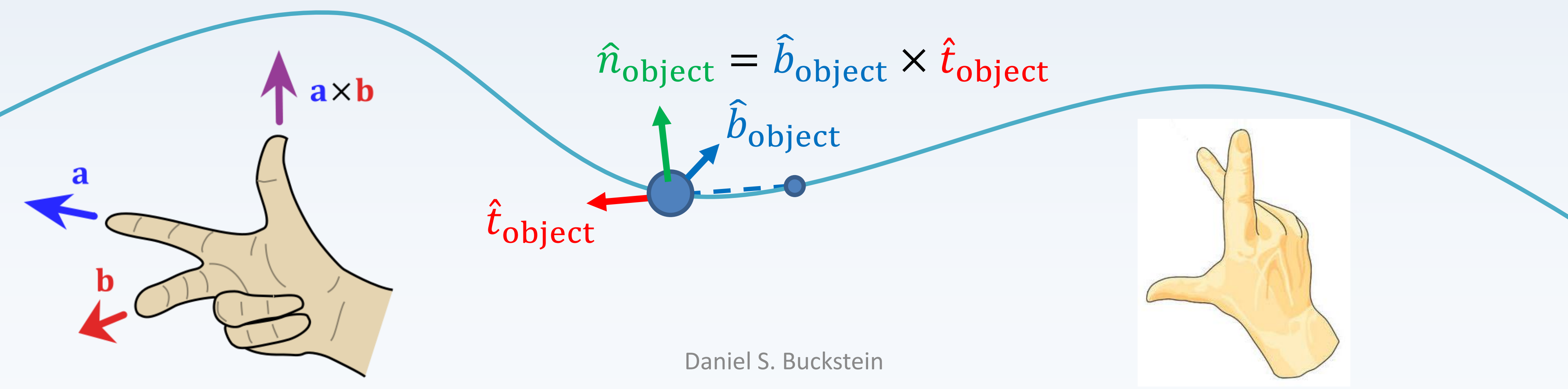
$$\vec{b}_{\text{object}} = -\hat{u}_{\text{world}} \times \hat{t}_{\text{object}}$$
$$= \hat{t}_{\text{object}} \times \hat{u}_{\text{world}}$$
$$\hat{b}_{\text{object}} = \text{normalize}\left(\hat{t}_{\text{object}} \times \hat{u}_{\text{world}}\right)$$

Frenet-Serret Frames

- The remaining basis (2nd) is calculated as *3rd cross 1st*, or

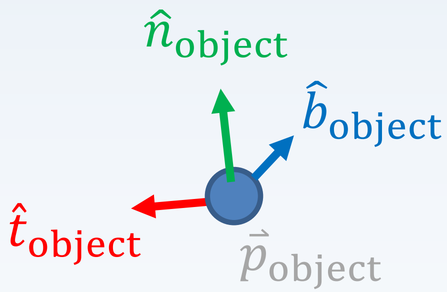
$$\hat{e}_1 = \hat{e}_2 \times \hat{e}_0$$

- This is exactly what we did in previously, but the 1st and 3rd bases had different directions!

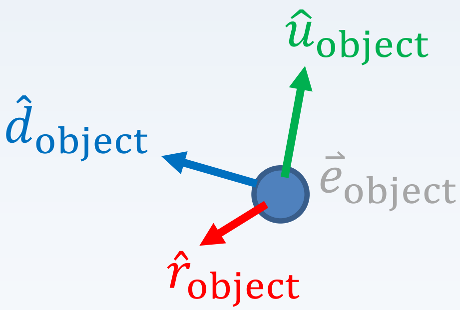


Frenet-Serret Frames

- So we have two different representations of *relatively* the same thing:
- Two different rotations that satisfy the right-hand rule!
- Their basis vectors are expressed differently.



$$
\begin{aligned}
R &= [\hat{e}_0 \quad \hat{e}_1 \quad \hat{e}_2] \\
&\equiv [\,\hat{r} \quad \hat{u} \quad \hat{d}\,] \\
&\equiv [\,\hat{t} \quad \hat{n} \quad \hat{b}\,]
\end{aligned}
$$



Frenet-Serret Frames

- As seen in the previous two examples, ***basis vectors are just basis vectors***…
- …how you use them gives them context!
- However, the general rule is as follows (and ***calculation order matters***):

$$\hat{e}_0 = \hat{e}_1 \times \hat{e}_2$$
$$\hat{e}_1 = \hat{e}_2 \times \hat{e}_0$$
$$\hat{e}_2 = \hat{e}_0 \times \hat{e}_1$$

Frenet-Serret Frames

- Corollary: (what a coincidence)

IF

$$\hat{e}_0 = i$$
$$\hat{e}_1 = j$$
$$\hat{e}_2 = k$$

THEN

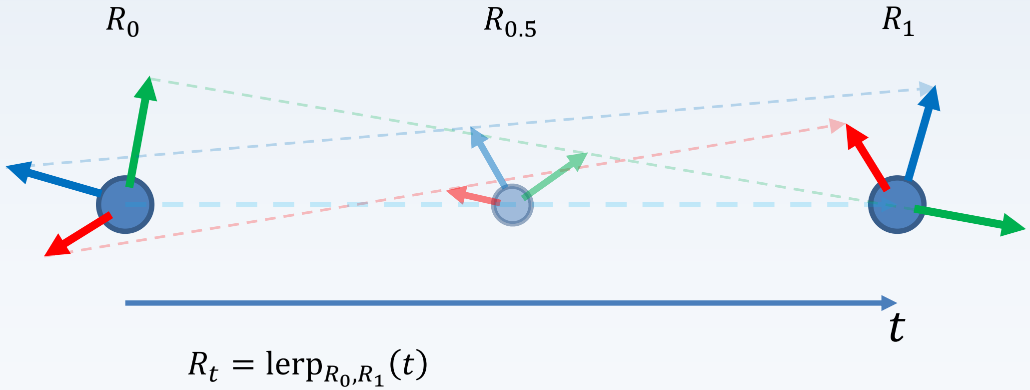
$$\hat{e}_0 = i = \hat{e}_1 \times \hat{e}_2 = jk$$
$$\hat{e}_1 = j = \hat{e}_2 \times \hat{e}_0 = ki$$
$$\hat{e}_2 = k = \hat{e}_0 \times \hat{e}_1 = ij$$

Animating Transformations

- Rotation matrices have issues ☹
- Linearly interpolating between two matrices will result in the basis vectors becoming *un-normalized*… why???
- Because interpolating the matrix means…
- …interpolating the basis vectors
- The result is NLERP without the N… ***no arc***!

Animating Transformations

- Linearly interpolating rotation matrices:
- This has the same effect as applying *scale* simultaneously…
- Here's a graphical example (over time):

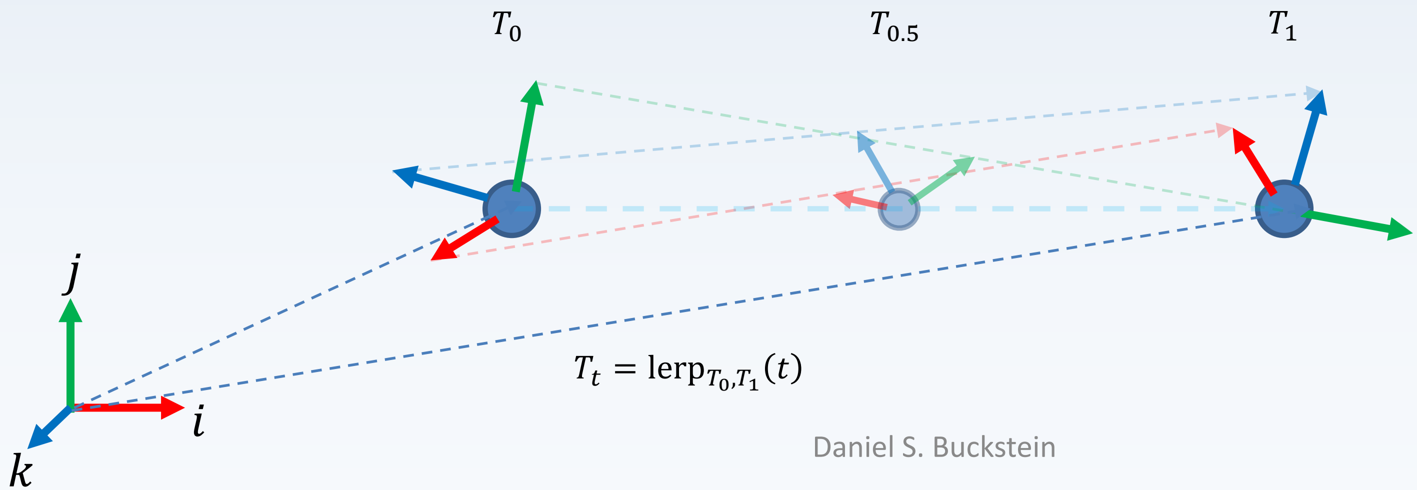


Animating Transformations

- Same effect when dealing with homogeneous transformations (rotation + position):
- Problem persists because there is still a rotation matrix involved →

$$T_0 = \begin{bmatrix} R_0 & \vec{p}_0 \\ 0 & 1 \end{bmatrix}$$

$$T_1 = \begin{bmatrix} R_1 & \vec{p}_1 \\ 0 & 1 \end{bmatrix}$$



Animating Transformations

- To interpolate a rotation + translation combo, the interpolation algorithm must select the appropriate method for each.
- E.g. LERP for position → vector ***LERP***
- E.g. LERP for rotation → quaternion ***SLERP***

Animating Transformations

- The algorithm (made of math):

$$T_t = \begin{bmatrix} R_t & \vec{p}_t \\ 0 & 1 \end{bmatrix}$$

$$R_t = \text{convert}\left(\text{slerp}_{\hat{q}_0,\hat{q}_1}(t)\right)$$

$$\vec{p}_t = \text{lerp}_{\vec{p}_0,\vec{p}_1}(t)$$

- More on this method soon… ☺

The end.

- Questions? Comments? Concerns?