Look-ahead Filter

Hypothesis:

look-ahead filter:

Let a be the number a frames to look ahead, the hypothesis:

$v(k+a) = v(k-1).v\Delta(k)^{a+1}$

and $v\Delta(k)$ can be updated:

$v\Delta(k) = slerp(v\Delta(k+a), v\Delta(k), \gamma)$

forward filter

$v\Delta(k) = slerp(p\Delta(k), v\Delta(k-1), \alpha)$

Data Structure

the data structure have to fit both forward filter and look-ahead filter, as look-ahead number a=0, which means forward filter.

there are queues in the filter to buffer history data:

- p_cam_orien queue

  physical camera orientation buffer, include knee points orientation of each frame.
- v_cam_orien queue

  virtual camera orientation buffer
- $v\Delta$ queue

  virtual camera velocity buffer
- frame queue

  frame buffer

Queue	0	1	2	3	4
p_cam_orien_buffer					
v_cam_orien_buffer					
v_cam_velocity_buffer					
frame_buffer					

Queue Handler

let set a = 5, so each queue size would be 5

Initialization:

```
# initialize buffer
v_cam_orien_buffer = deque(maxlen=a)
p_cam_orien_buffer = deque(maxlen=a)
v_cam_velocity_buffer =  deque(maxlen=a)
frame_buffer = deque(deque(maxlen=a))
# initialize camera orientation and velocity
prev_v_cam_orien = Quaternion()
prev_p_cam_orien = Quaternion()
prev_v_cam_velocity = Quaternion()
```

- i = 0

  $p[0] = intergrate(p_{prev}, \omega[0], \Delta t)$

  $p\Delta[0] = p[0].p^*_{prev}$

  $v\Delta[0] = slerp(p\Delta[0], v\Delta_{prev}, alpha)$

  $v[0] = v_{prev}.v\Delta[0]$

  push to buffer:

  $frame[0], v[0], v\Delta[0]$

  $p_{prev} = p[0]$

  $v_{prev} = v[0]$

  $v\Delta_{prev} = v\Delta[0]$

Queue	0	1	2	3	4
p_cam_orien_buffer	$p[0]$				
v_cam_orien_buffer	$v[0]$				
v_cam_velocity_buffer	$v\Delta[0]$				
frame_buffer	frame[0]				

- i = 1
- i = 2
- i = 3

  ...

- i = 4

  $p[4] = intergrate(p_{prev}, \omega[4], \Delta t)$

  $p\Delta[4] = p[4].p^*_{prev}$

  $v\Delta[4] = slerp(p\Delta[4], v\Delta_{prev}, alpha)$

  $v[4] = v_{prev}.v\Delta[4]$

  push to buffer:

  $frame[4], v[4], v\Delta[4]$

  $p_{prev} = p[4]$

  $v_{prev} = v[4]$

  $v\Delta_{prev} = v\Delta[4]$

Queue	0	1	2	3	4
p_cam_orien_buffer	$p[0]$	$p[1]$	$p[2]$	$p[3]$	$p[4]$
v_cam_orien_buffer	$v[0]$	$v[1]$	$v[2]$	$v[3]$	$v[4]$
v_cam_velocity_buffer	$v\Delta[0]$	$v\Delta[1]$	$v\Delta[2]$	$v\Delta[3]$	$v\Delta[4]$
frame_buffer	frame[0]	frame[1]	frame[2]	frame[3]	frame[4]

- i = 5

$p[5] = intergrate(p_{prev}, \omega[5], \Delta t)$

$p\Delta[5] = p[5].p^*_{prev}$

$v\Delta[5] = slerp(p\Delta[5], v\Delta_{prev}, \alpha)$

$v[5] = v_{prev}.v\Delta[5]$

if (queue.size() >= a)

{

$v\Delta[0] = slerp(v\Delta[0], v\Delta[5], \gamma)$

$v[1] = v[0].v\Delta[0]$

trans_matrices[1] = cal_trans_matrices(v[1],p[1])

}

push to buffer:

$frame[5], v[5], v\Delta[5]$

$p_{prev} = p[5]$

$v_{prev} = v[5]$

$v\Delta_{prev} = v\Delta[5]$

Queue	0	1	2	3	4
p_cam_orien_buffer	p[1]	p [2]	p [3]	p [4]	p [5]
v_cam_orien_buffer	v[1]	v [2]	v[3]	v[4]	v[5]
v_cam_velocity_buffer	$v\Delta[1]$	$v\Delta[2]$	$v\Delta[3]$	$v\Delta[4]$	$v\Delta[5]$
frame_buffer	frame[1]	frame[2]	frame[3]	frame[4]	frame[5]

Pseudo code:

```
# integrate physycal camera orientation
p_cam_orien = integrate_p_cam_orien(prev_p_cam_orien,angular_vecloty, timestamp)
# calculate physical camera velocity
p_cam_velocity = p_cam_orien*prev_p_cam_orien.conjugate
# virtual camera velocity calculation in foward direction
current_v_cam_velocity= slerp(p_cam_velocity,prev_v_cam_velocity,alpha)
```

```
# virtual camera velocity calculation in look-ahead direction
if len(v_cam_velocity_buffer)>= a:

    foward_v_cam_velocity = v_cam_velocity_buffer[0]
    foward_v_cam_velocity =
slerp(current_v_cam_velocity,foward_v_cam_velocity,gamma)

    # virtual camera orientation integration
    v_cam_orien_buffer[1] = v_cam_orien_buffer[0]* forward_v_cam_velocity
    trans_matrices =
cal_trans_matrices(v_cam_orien_buffer[1],p_cam_orien_buffer[1])

# push orientation and velocity to buffer
p_cam_orien_buffer.push_back(p_cam_orien)
v_cam_orien_buffer.push_back(v_cam_orien)
v_cam_velocity_buffer.push_back(current_v_cam_velocity)

# store current to previous orientation and velocity
prev_v_cam_orien = v_cam_orien
prev_p_cam_orien = p_cam_orien
prev_v_cam_velocity = current_v_cam_velocity
```