

CSE 190: Virtual Reality Technologies

LECTURE #18: OPTICAL TRACKING

VR Content Presentations

Ran Tao: Virtual Virtual Reality

- <https://youtu.be/KOYXeqAAuxU>

Andrew Hwang: Reality Hacker VR

- <https://www.youtube.com/watch?v=r3pXEtpLDLI&t=4s>

Mou Sun: Skydiving 360

- <https://www.youtube.com/watch?v=a54H2V06dqw>

Clark Wu: 360 Google Spotlight Rain Or Shine

- <https://www.youtube.com/watch?v=QXF7uGfopnY>

Garrett Brush

- https://www.youtube.com/watch?v=e5b-l__or0g
- <https://docs.google.com/presentation/d/1pnB3XnurBCXxfk2PteOja-STU0YjoErBKMG0YQB8UbY/edit?usp=sharing>

Pedro Coutin Portuondo: Stunning 360 Paramotor Flight

- https://www.youtube.com/watch?v=pHxzmc7_5Vg

Announcements

Final Project

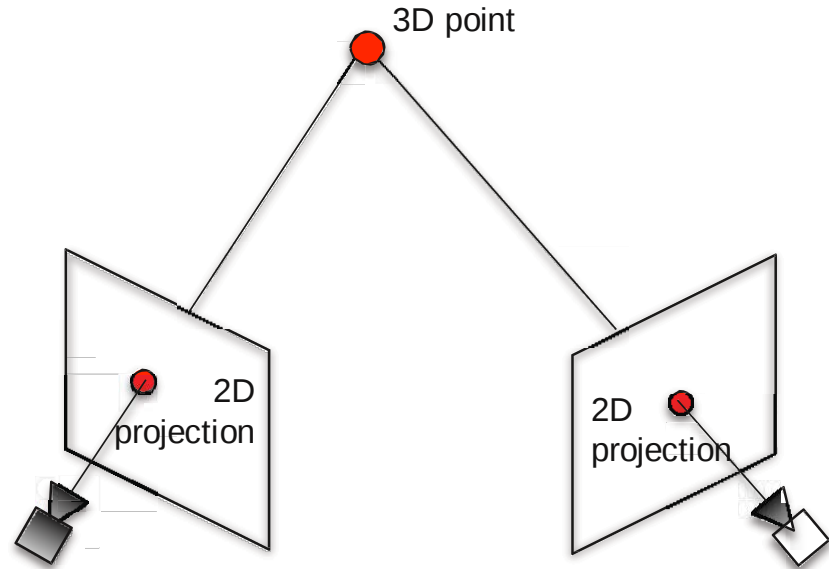
- Blog #2 due Monday night
- Presentations next Tuesday 3-6pm
 - Slide shows 3-4pm: Google Slides
 - Even hour teams 4-5pm
 - Odd hour teams 5-6pm

TA/tutor evaluations

CAPE

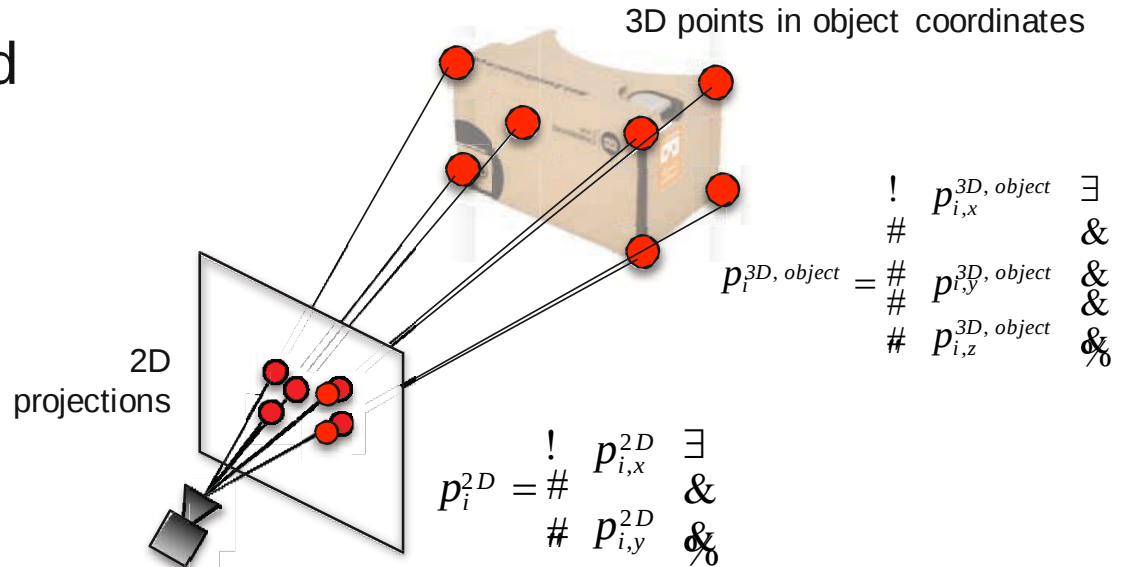
Positional Tracking - Optical

- for tracking individual 3D points, multi-camera setups usually use triangulation
- this does not give us the pose (rotation & translation) of camera or object yet



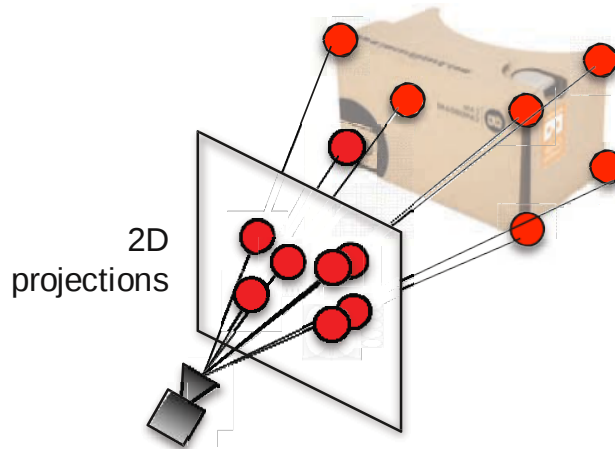
Positional Tracking - Optical

- for pose tracking, need to track multiple 3D points with known relative coordinates!



Positional Tracking - Optical

- when object is closer, projection is bigger

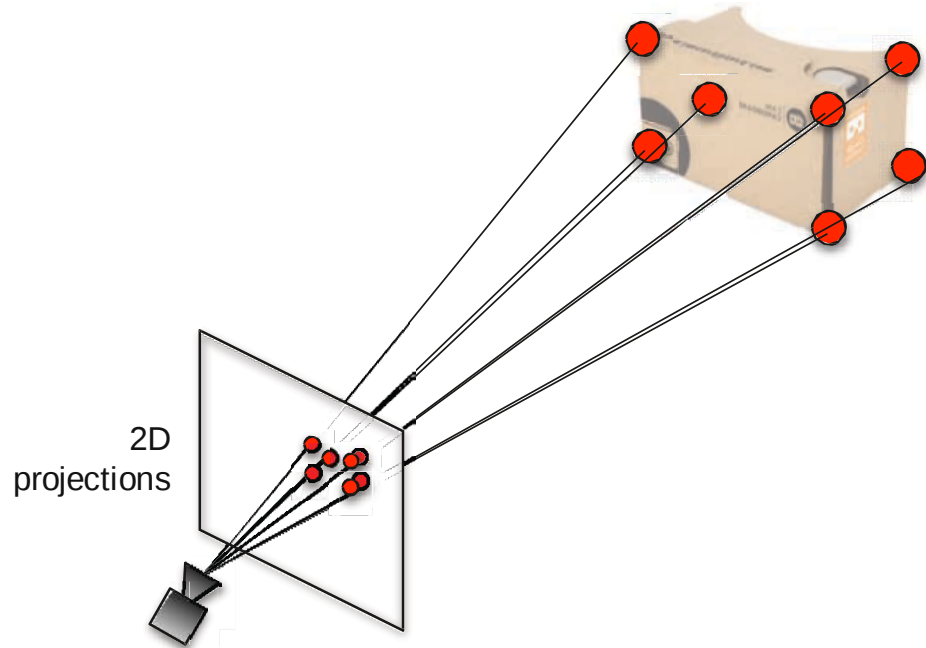


Positional Tracking - Optical

- when object is farther, projection is smaller

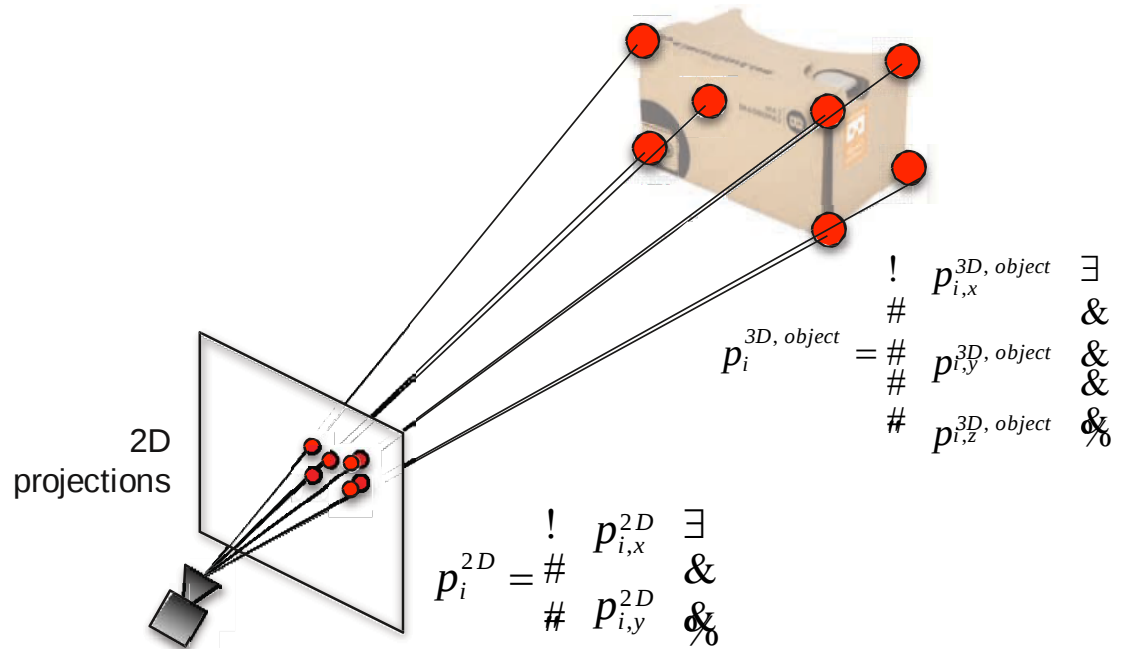
... and so on

...



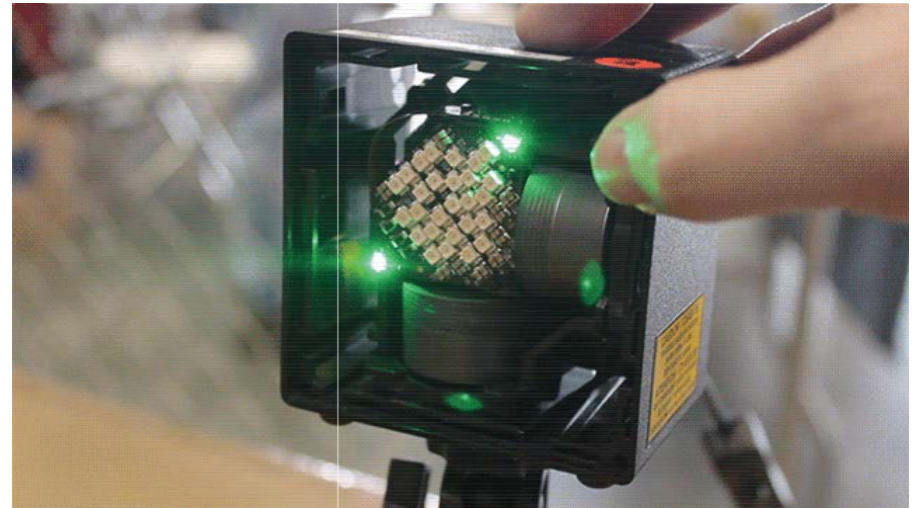
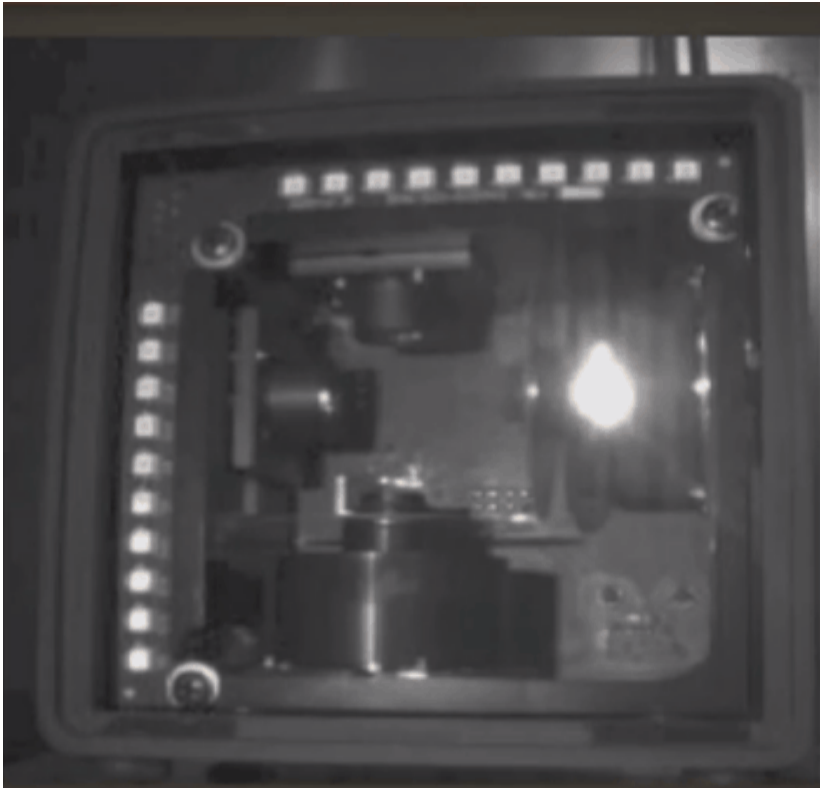
Positional Tracking - Optical

- pose estimation via optimization!
- nonlinear least squares problem



$$\underset{\{R, T\}}{\text{minimize}} \left\| \underbrace{\begin{pmatrix} p_1^{2D}, p_2^{2D}, \dots, p_N^{2D} \end{pmatrix}}_{\text{observed 2D points}} \underbrace{f \left(\begin{pmatrix} p_1^{3D, object}, p_2^{3D, object}, \dots, p_N^{3D, object} \end{pmatrix}, \underbrace{R, t}_{\text{unknown pose}} \right)}_{\text{known 3D points}} \right\|_2^2$$

HTC Lighthouse



<http://gizmodo.com/this-is-how-valve-s-amazing-lighthouse-tracking-technol-1705356768>

HTC Lighthouse



<https://www.youtube.com/watch?v=J54dotTt7k0>

HTC Lighthouse

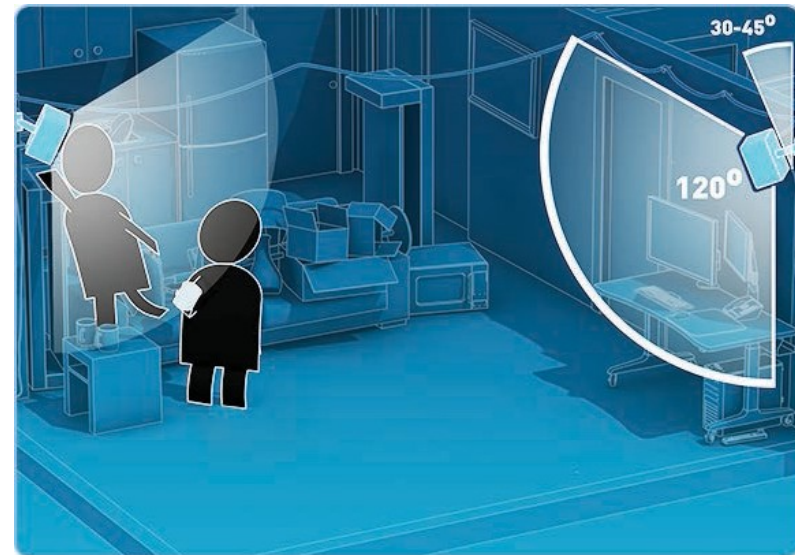
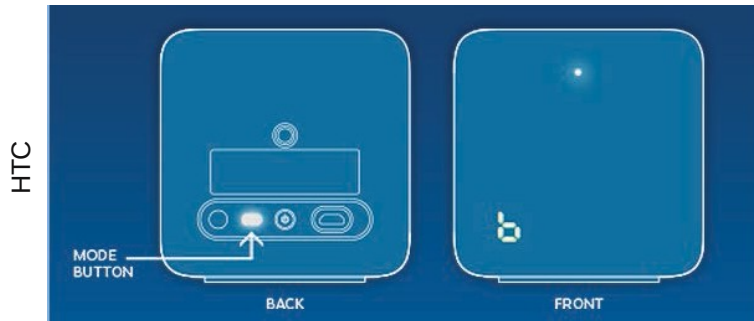
important specs:

- runs at 60 Hz
 - i.e. horizontal & vertical update combined 60 Hz
 - broadband sync pulses in between each laser sweep (i.e. at 120 Hz)
- each laser rotates at 60 Hz, but offset in time
- useable field of view: 120 degrees



HTC Lighthouse – Base Station

- can use up to 2 base stations simultaneously via *time-division multiplexing* (TDM)
- base station modes:
 - A: TDM slave with cable sync
 - B: TDM master
 - C: TDM slave with optical sync



HTC Lighthouse – Base Station

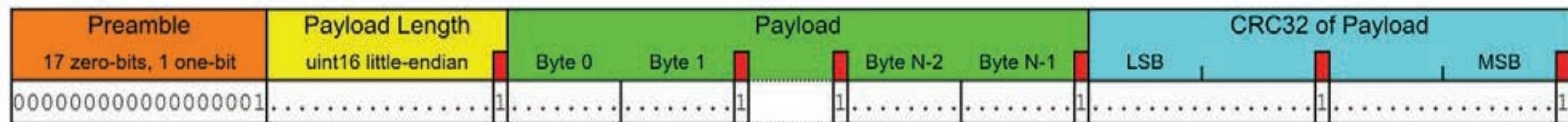
- sync pulse periodically emitted (120 times per second)
- each sync pulse indicates beginning of new sweep
- length of pulse also encodes additional 3 bits of information:

Name	skip	data	axis	length (ticks)	length (μs)
j0	0	0	0	3000	62.5
k0	0	0	1	3500	72.9
j1	0	1	0	4000	83.3
k1	0	1	1	4500	93.8
j2	1	0	0	5000	104
k2	1	0	1	5500	115
j3	1	1	0	6000	125
k3	1	1	1	6500	135

- axis: horizontal or vertical sweep to follow
- skip: if 1, then laser is off for following sweep
- data: data bits of consecutive pulses yield OOTX frame

HTC Lighthouse – Base Station

- OOTX frame used to communicate between base stations or with sensors
- can send calibration data and all kinds of info
- detailed info here: <https://github.com/nairol/LighthouseRedox/blob/master/docs/Light%20Emissions.md#sync-pulse>



- Sync Bit / Stuffed Bit**
Inserted after every 16 bits. Every 17th bit is a Sync Bit. Counting starts after the Preamble sequence.
- Preamble**
Unique bit pattern that marks the beginning of a new frame. It is unique because Sync Bits make blocks of more than 16 zero-bits impossible inside a frame.
- Payload Length**
Number of payload bytes in this frame. Sync Bits are ignored and do not contribute to this number.
- Payload**
Array of data bytes. If the Payload Length field is not a multiple of 2, a padding byte (zero-byte) is appended so that the Payload field can always end with a Sync Bit.
- CRC32 of Payload**
Little-endian representation of the CRC32 of all payload bytes excluding the padding byte (if used) and ignoring all Sync Bits.