Rendering

Tue, July 21 (Week 5)

Rendering: Mesh to Pixels

1. Meshes + Textures + Matrices + Shaders gets prepared in the CPU.

2. They get sent to GPU that creates pixels using them.

3. The pixels from the GPU gets sent to the monitor.

Rendering: Mesh to Pixels

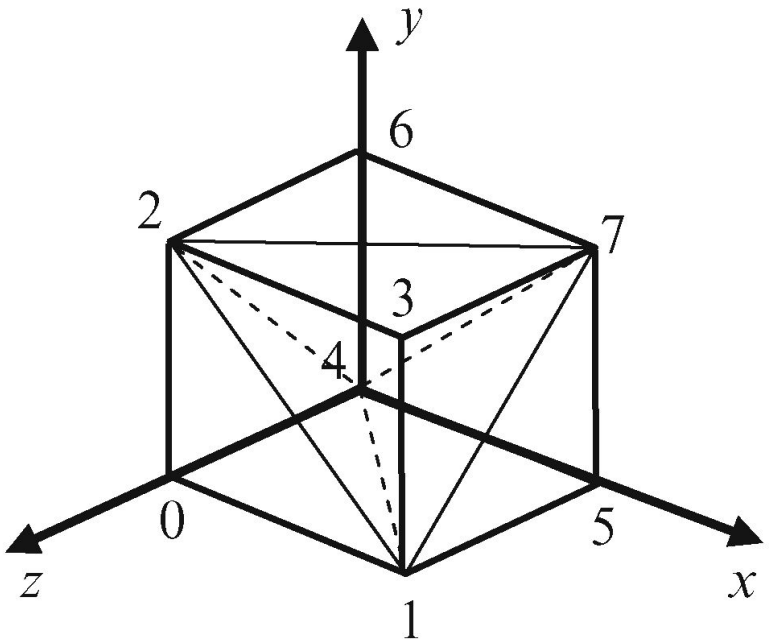
1. Meshes + Textures + Matrices + Shaders gets prepared in the CPU.

2. They get sent to GPU that creates pixels using them.

3. The pixels from the GPU gets sent to the monitor.

Haven't talked about this part that much yet, actually.

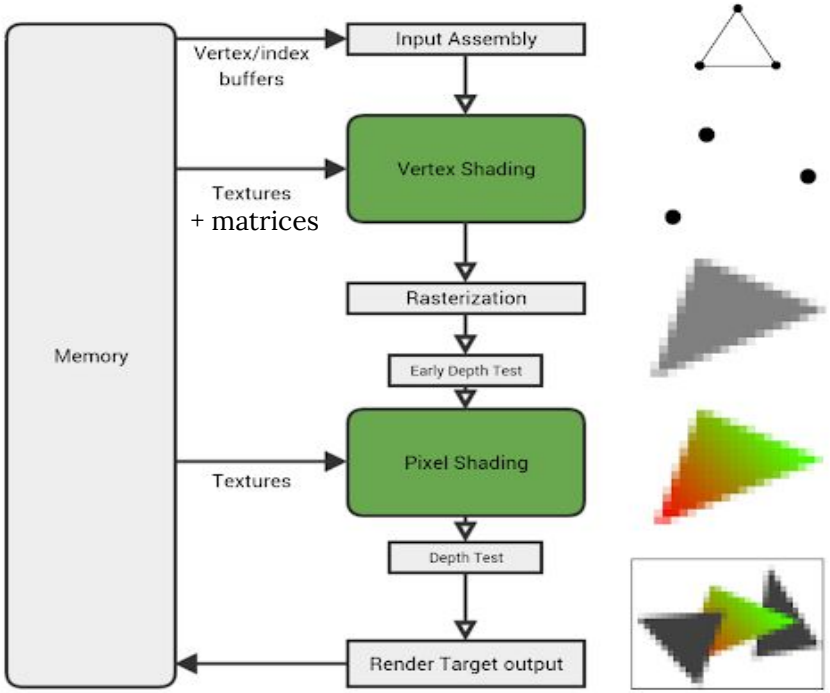
Mesh



Vertex List		
x	y	z
0.0	0.0	1.0
1.0	0.0	1.0
0.0	1.0	1.0
1.0	1.0	1.0
0.0	0.0	0.0
1.0	0.0	0.0
0.0	1.0	0.0
1.0	1.0	0.0

Triangle List		
i	j	k
0	1	2
1	3	2
2	3	7
2	7	6
1	7	3
1	5	7
6	7	4
7	5	4
0	4	1
1	4	5
2	6	4
0	2	4

Rendering Pipeline



Vertex Shading

Projects vertices in a mesh on the screen using *matrices*.

View matrices includes

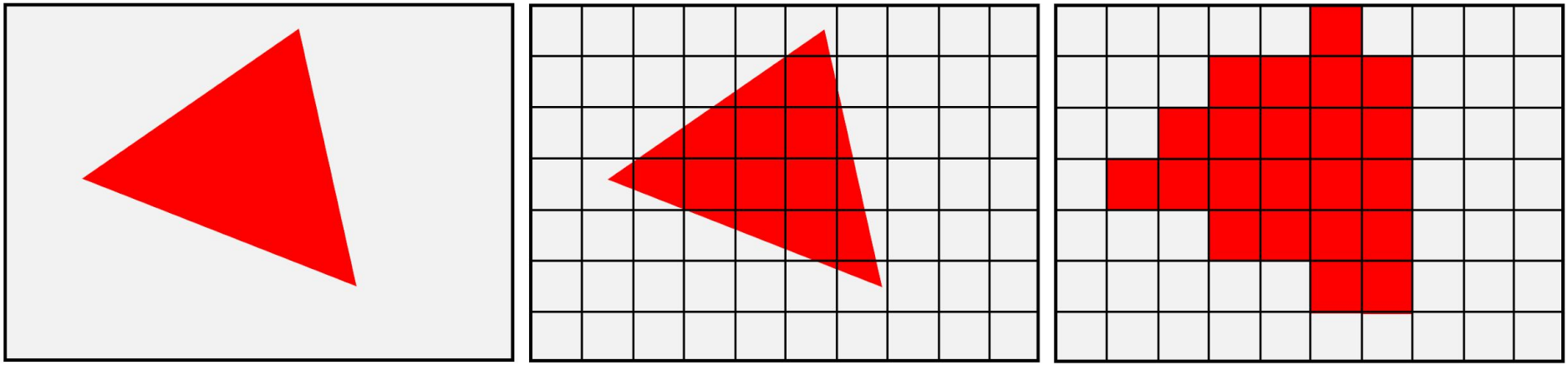
1. Tracking data of person is in the real world
2. IPD of the person

Also, for AR/VR, there are two view matrices for each of the eyes!

Rasterization

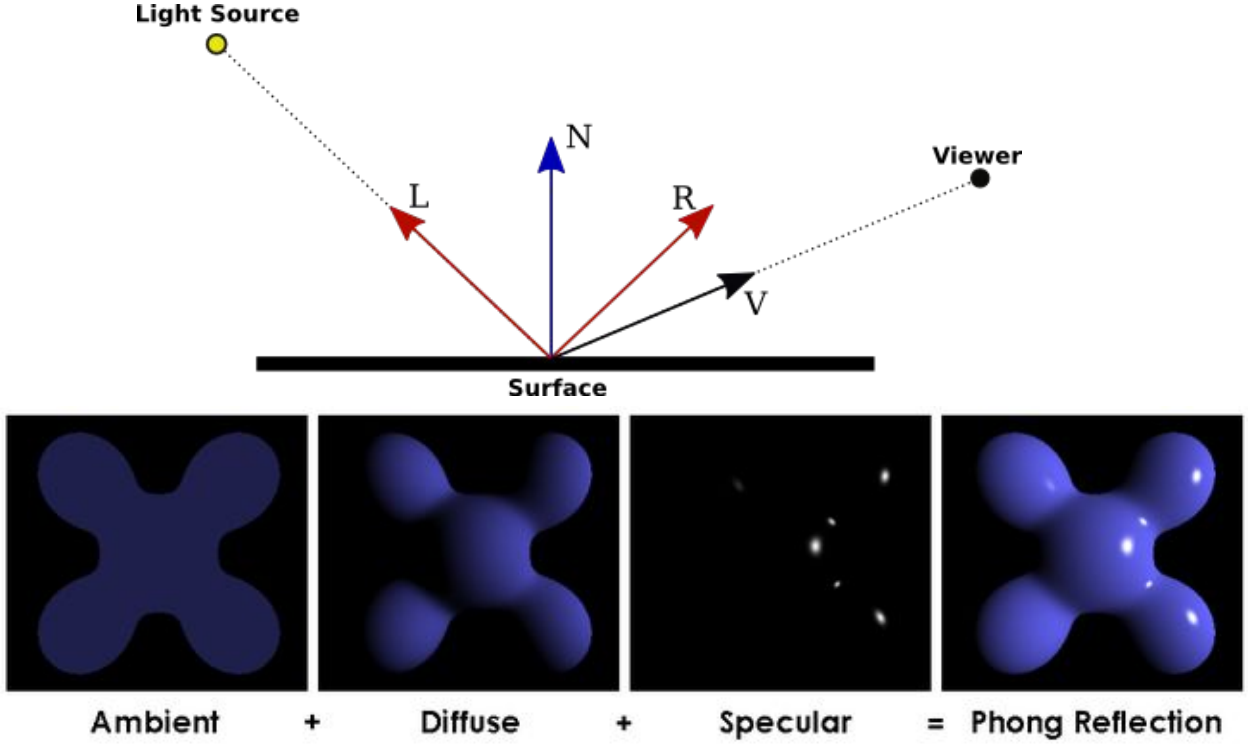
Find the pixels on the screen that are inside the triangles of projected vertices.

The triangle list of the mesh is used here.



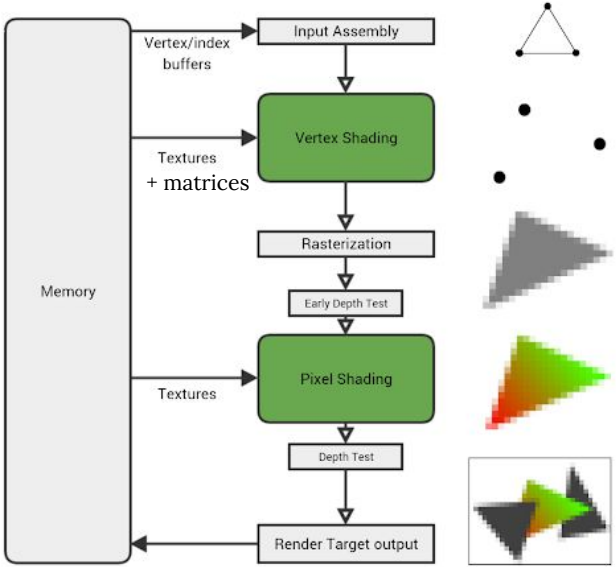
Pixel Shading

Find a color for each pixel in the projected triangles.

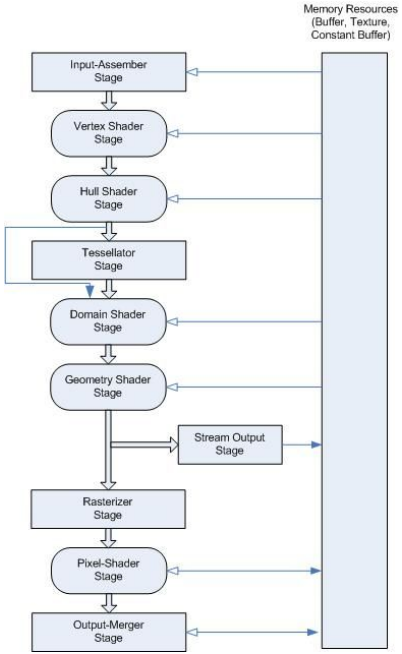


The Sophistication of the Pipeline

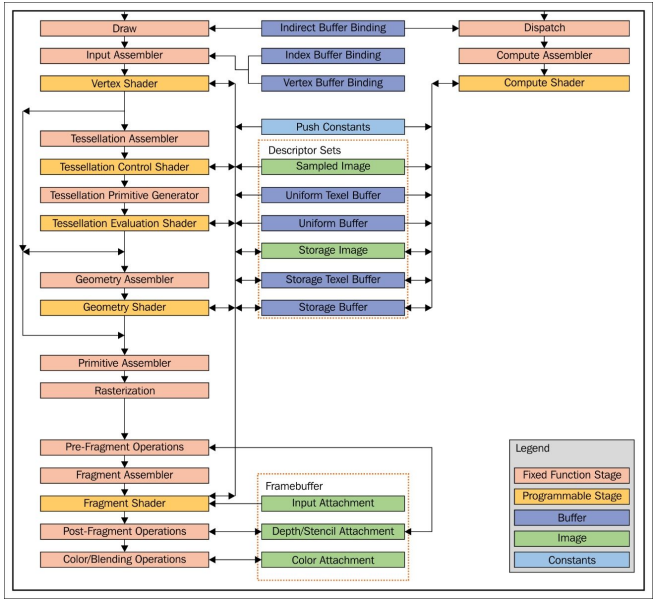
Year 2000



Year 2010



Year 2020

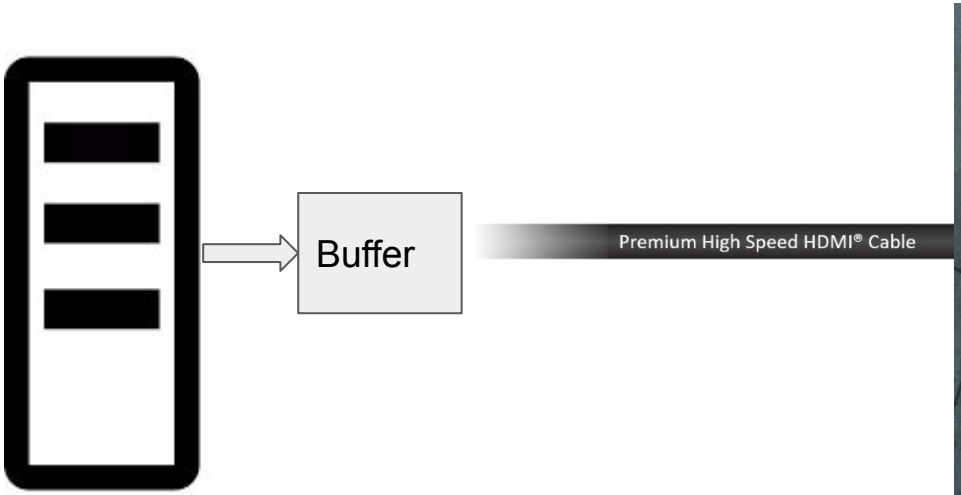


Single Buffering

Simply send the rendered pixels to the monitor.

Single Buffering

Tearing!

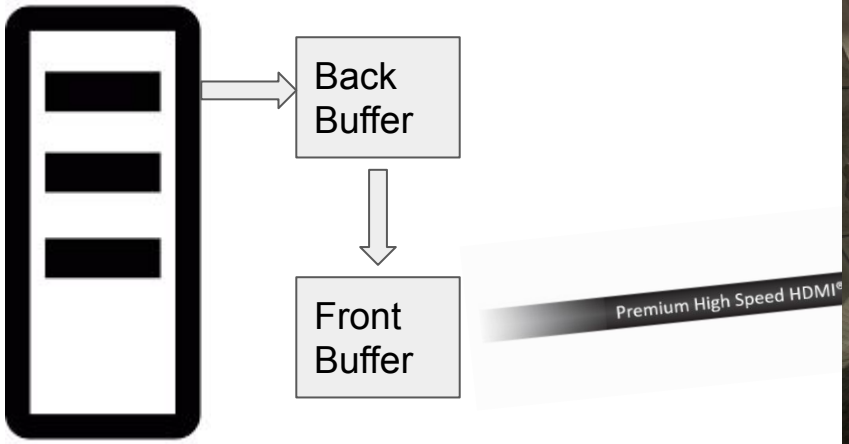


Double Buffering

Update the back buffer and then copy it to the front buffer when the back buffer is ready.

Double Buffering

No Tearing!



Double Buffering with AR/VR

While double buffering removes tearing, it adds an additional step.

Additional step means adding latency (e.g., 1/120 sec in average for 60 Hz setting), which is terrible for AR/VR.

(1/120 sec is the average of 0~1/60 sec.)

Example of AR/VR being fundamentally different from PC:

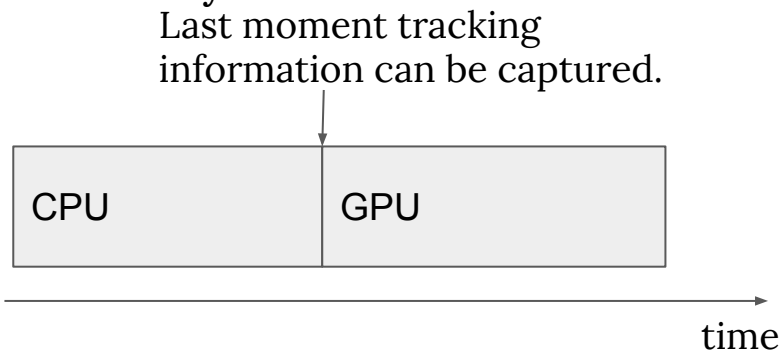
Some AR/VR devices skip double buffering since latency is worse than tearing.

Asynchronous Reprojection

Another approach for AR/VR, but not PC.

Preparing larger field of view and then picking where to actually show.

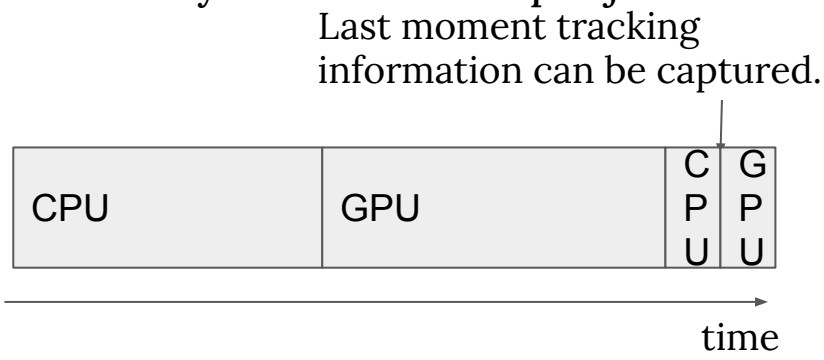
Normally...



Asynchronous Reprojection

Another approach for AR/VR, but not PC.

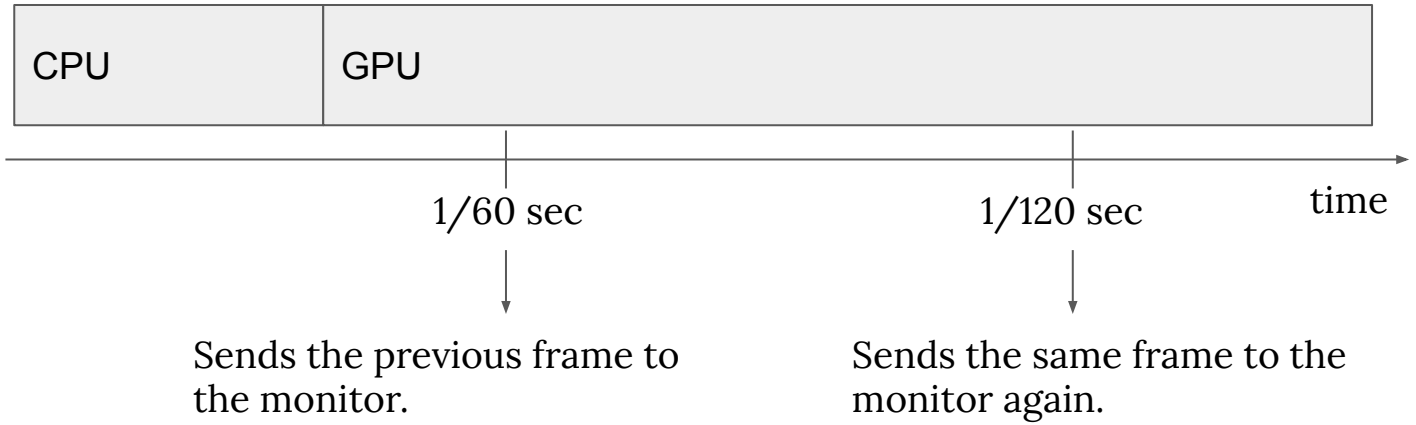
With Asynchronous Reprojection



Asynchronous Reprojection

Another reason: applying tracking information even when the GPU fails to update in time.

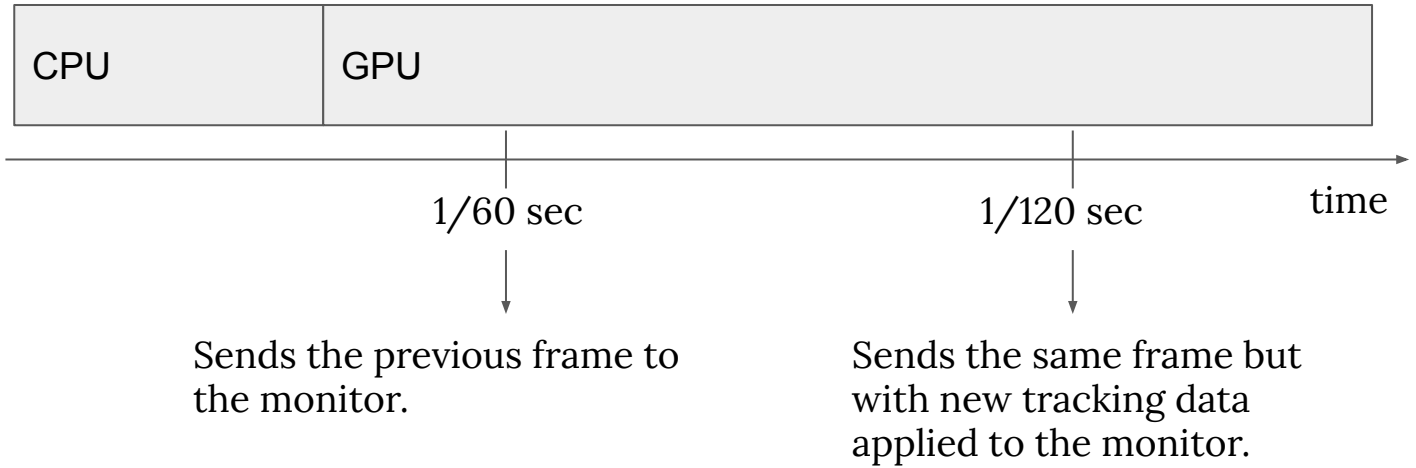
Normally,



Asynchronous Reprojection

Another reason: applying tracking information even when the GPU fails to update in time.

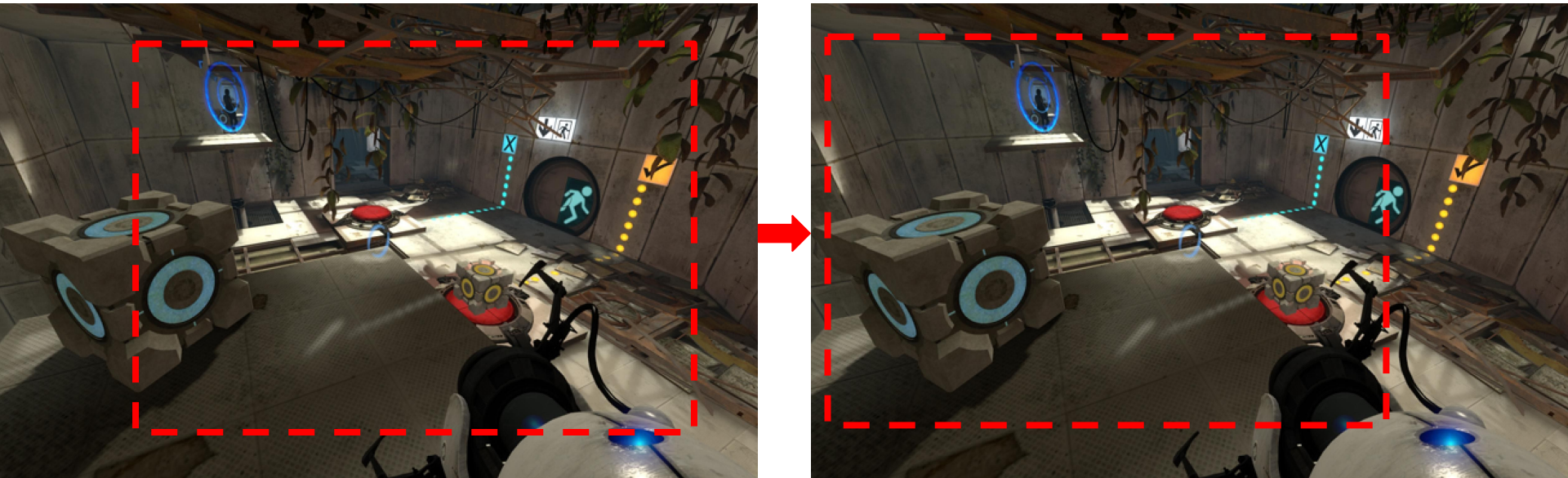
With Asynchronous Reprojection,



Asynchronous Reprojection

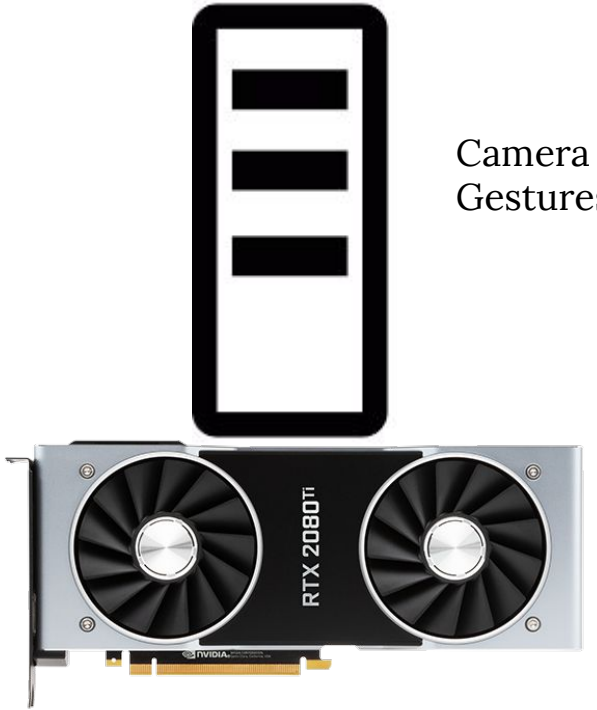
Another reason: applying tracking information even when the GPU fails to update in time.

With Asynchronous Reprojection,

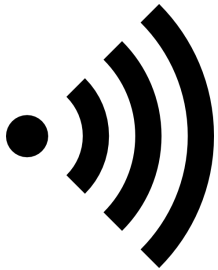


Remote Rendering

Borrowing the computational power from larger machines.



Camera Input,
Gestures



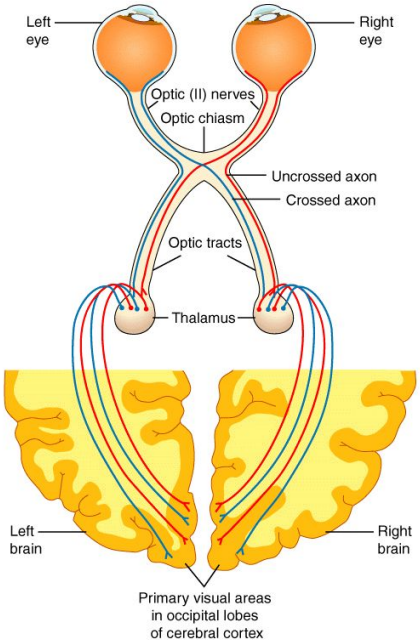
Pixels to Render

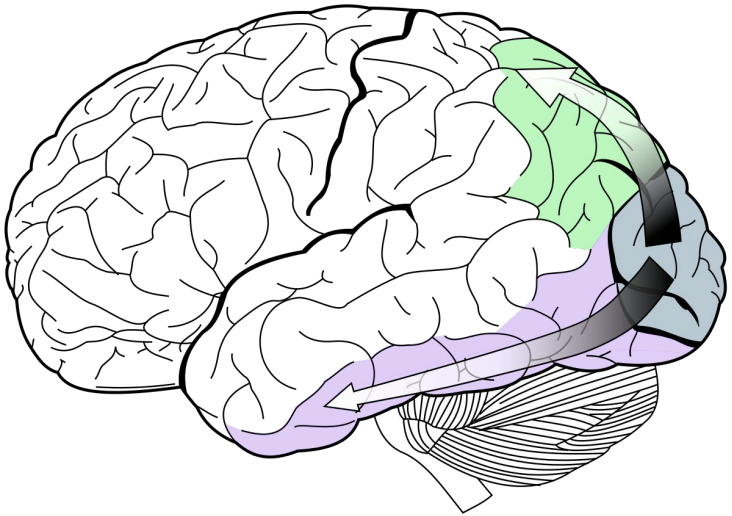
Remote Rendering

Similar to below, but through wireless communication.

Selective Attention

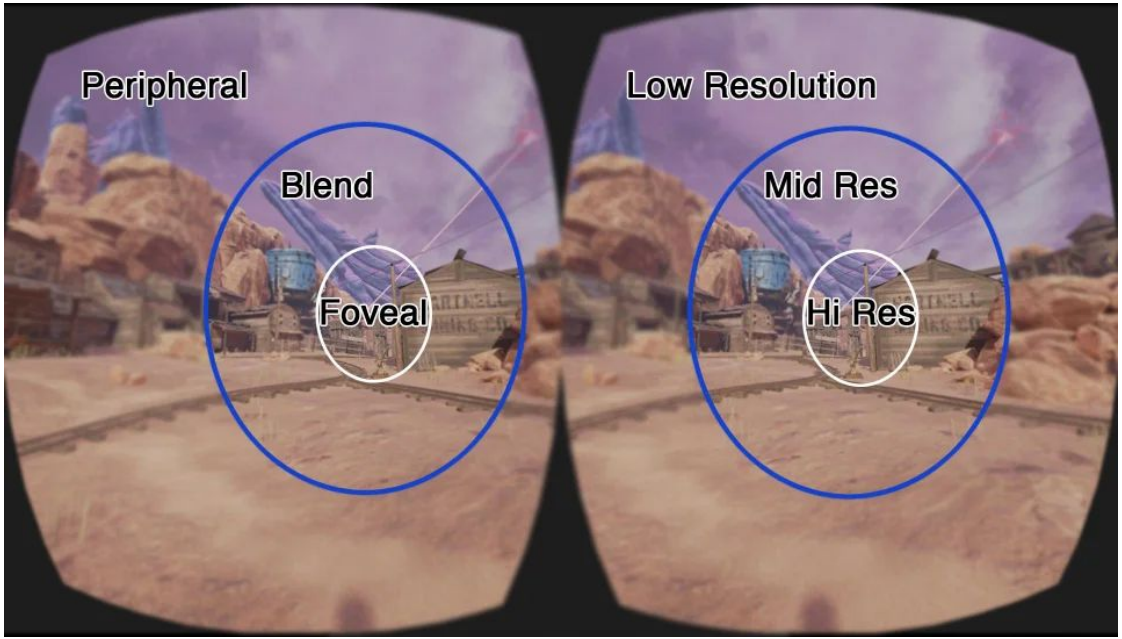
It is very inefficient and also impossible for humans to look at and understand the whole scene.





Foveated Rendering

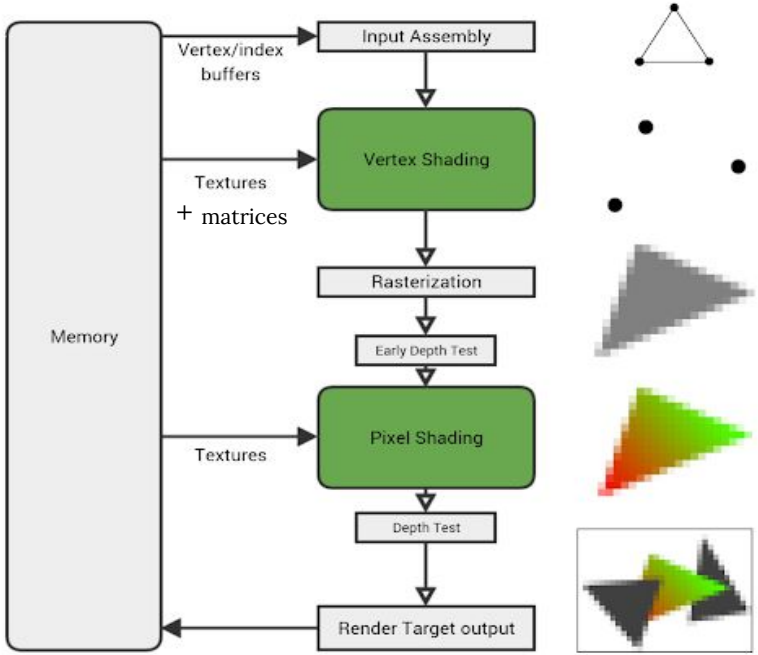
Saving computational power by considering where the user is looking at.



Double Pass Rendering

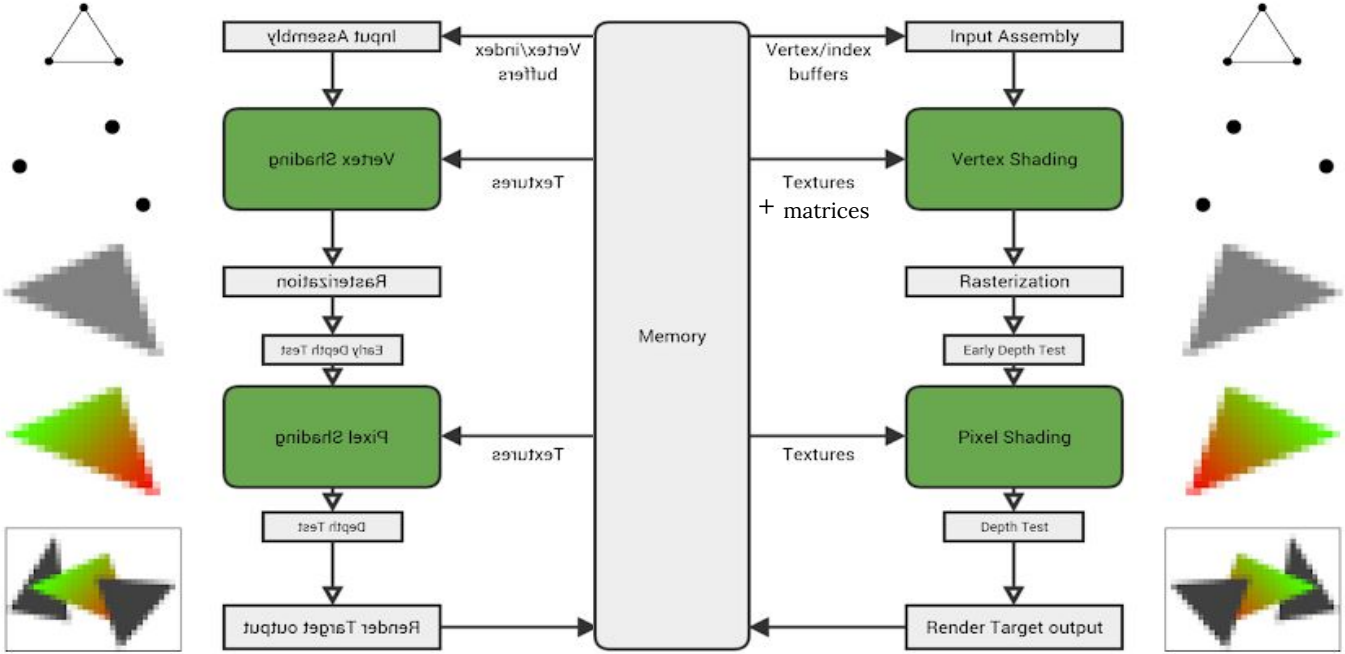
The most straightforward method for AR rendering.

Copy mesh and textures 2 times from CPU to GPU.



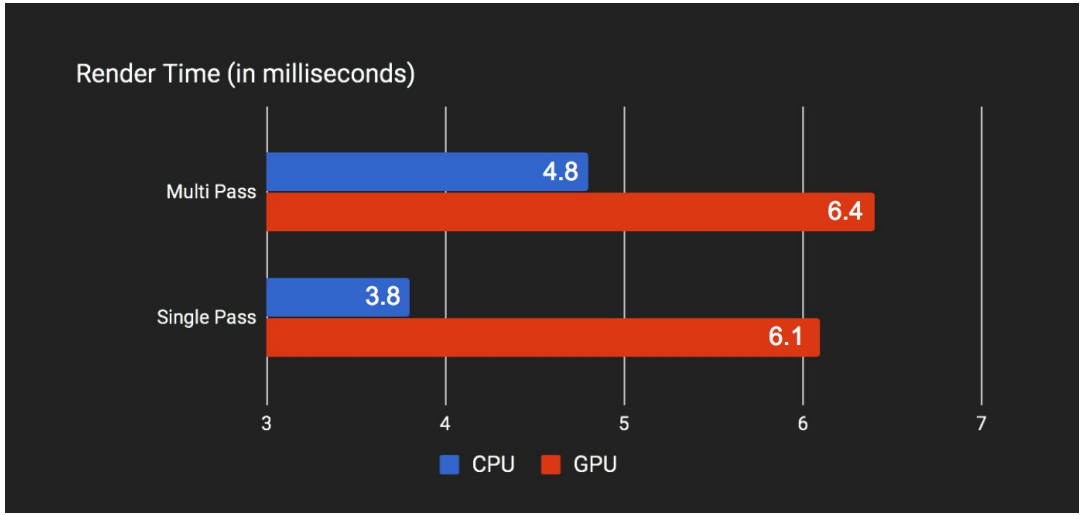
Single Pass Rendering

A better method with only one set of copies from the CPU to the GPU!



Single Pass Rendering

Skipping a copy saves time!



Techniques for Lower Latency

Single Buffering: tearing is bad, but for AR/VR, latency is evil.

Asynchronous Reprojection: at least approximately apply tracking data.

Remote Rendering: borrow a neighbor PC’s power if that helps.

Foveated Rendering: skip details of the pixels that people would care less.

Single Pass Rendering: leverage that scenes for left and right eyes are similar.