Immersive Systems III — Deep Dive in HMD Simulator

Developing Immersive Applications

Created by: Chek Tien TAN

Learning Objectives:

- explain how HMD parameters (IPD, FOV, eye relief, distortion) affect stereo rendering and user comfort
- describe the key components of the HMD simulator codebase and their roles in the rendering pipeline
- troubleshoot common HMD rendering issues using the HMD simulator
- relate optical parameters to rendering artifacts and comfort trade-offs

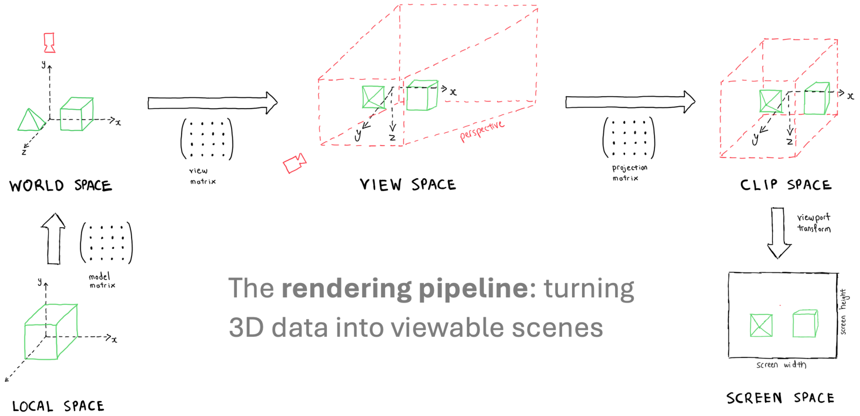
Recap: HMD Optics from Week 06

Key concepts we covered:

- Thin lens equation and image formation
- Eye relief, focal length, and their relationship to FOV
- View frustum parameters (symmetric vs asymmetric)
- Projection and view matrices for stereo rendering

Today we go hands-on with these concepts using the HMD simulator.

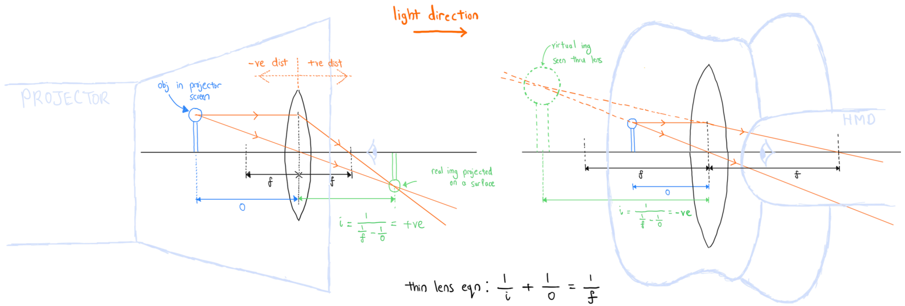
Recap: Core Graphics Concepts



The **rendering pipeline**: turning 3D data into viewable scenes

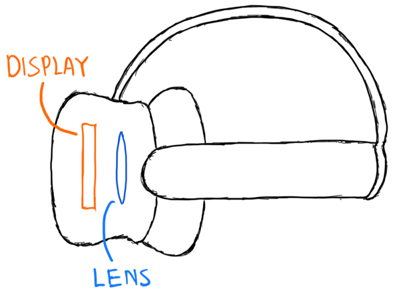
VR requires rendering this pipeline **twice per frame** at high FPS (e.g., 90+).

Recap: Core Physics Concepts

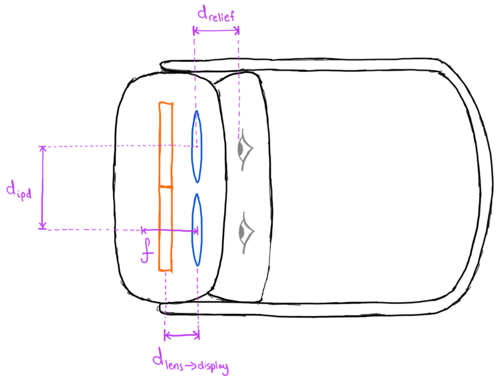

light direction
PROJECTOR
-ve dist
+ve dist
obj in projector screen
f
f
o
$i = \frac{1}{\frac{1}{f} - \frac{1}{o}} = +ve$
real img projected on a surface
virtual img seen thru lens
HMD
f
f
o
$i = \frac{1}{\frac{1}{f} - \frac{1}{o}} = -ve$

thin lens eqn: $\frac{1}{i} + \frac{1}{o} = \frac{1}{f}$

Recap: Key HMD Hardware Components

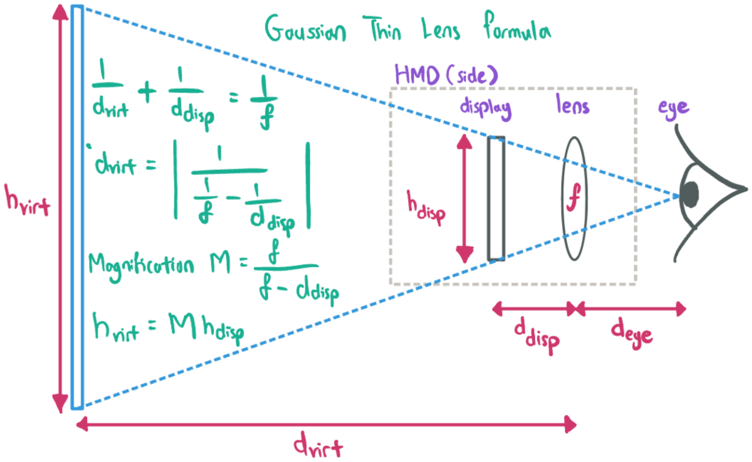


SIDE VIEW



TOP VIEW

Recap: Virtual Image Dist & Height



$$\frac{1}{d_{virt}} + \frac{1}{d_{disp}} = \frac{1}{f}$$

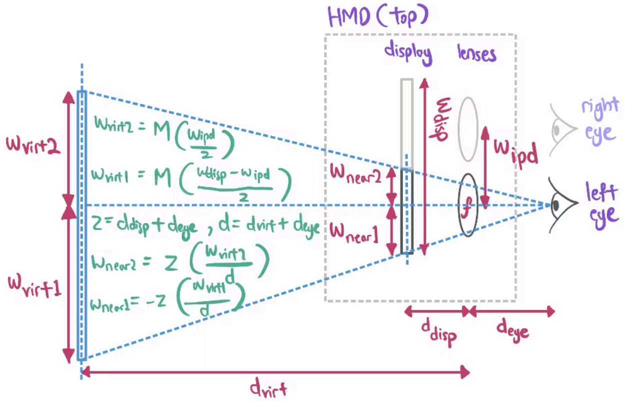
$$d_{virt} = \left| \frac{1}{\frac{1}{f} - \frac{1}{d_{disp}}} \right|$$

$$\text{Magnification } M = \frac{f}{f - d_{disp}}$$

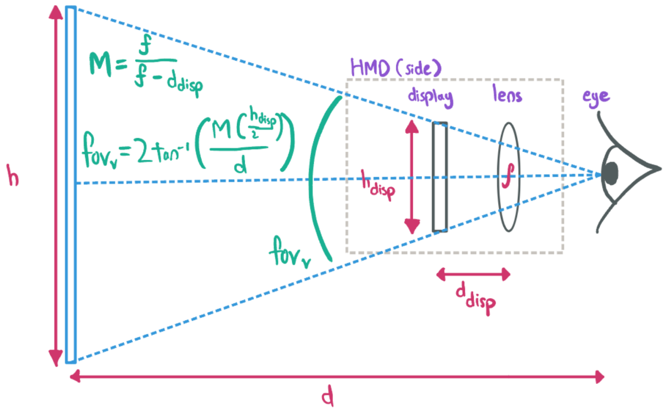
$$h_{virt} = M h_{disp}$$

Video segment: `https://youtu.be/OKD4jrnn4WE?t=406`

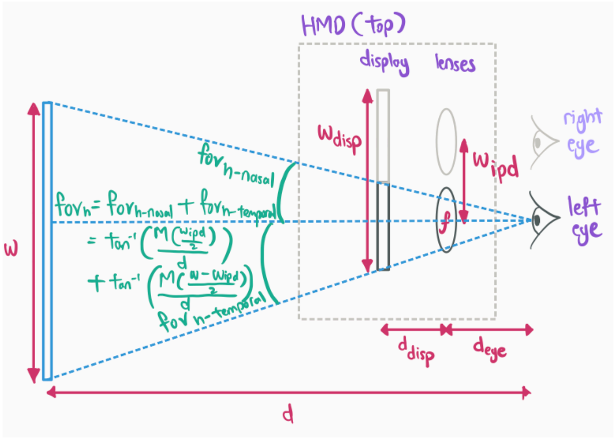
Computation: Virtual Image Width


HMD (top)
display
lenses
right eye
left eye
W_{disp}
W_{ipd}
f
W_{virt2}
W_{virt1}
W_{near2}
W_{near1}
$W_{virt2} = M\left(\frac{W_{ipd}}{2}\right)$
$W_{virt1} = M\left(\frac{W_{disp} - W_{ipd}}{2}\right)$
$z = d_{disp} + d_{eye}$, $d = d_{virt} + d_{eye}$
$W_{near2} = z\left(\frac{W_{virt2}}{d}\right)$
$W_{near1} = -z\left(\frac{W_{virt1}}{d}\right)$
d_{disp}
d_{eye}
d_{virt}

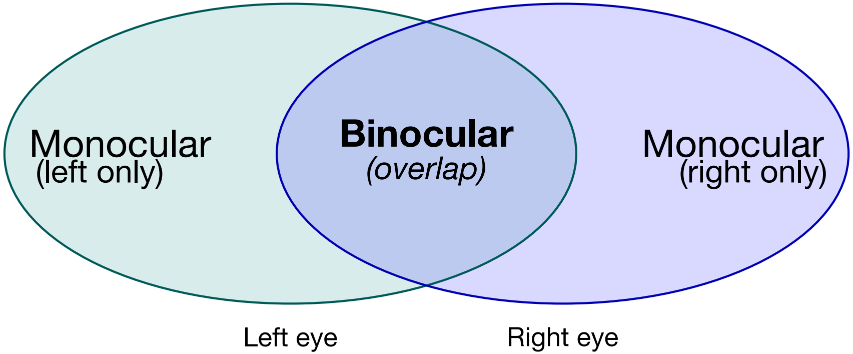
Computation: Vertical FOV


$M = \frac{f}{f - d_{disp}}$
$fov_v = 2\tan^{-1}\left(\frac{M\left(\frac{h_{disp}}{2}\right)}{d}\right)$
HMD (side)
display
lens
eye
h
h_{disp}
f
fov_v
d_{disp}
d

Computation: Horizontal FOV


HMD (top)
display
lenses
right eye
left eye
w_{disp}
w_{ipd}
f
w
$fov_{h\text{-}nasal}$
$fov_h = fov_{h\text{-}nasal} + fov_{h\text{-}temporal}$
$= \tan^{-1}\left(\frac{M\left(\frac{w_{ipd}}{2}\right)}{d}\right)$
$+ \tan^{-1}\left(\frac{M\left(\frac{w - w_{ipd}}{2}\right)}{d}\right)$
$fov_{h\text{-}temporal}$
d_{disp}
d_{eye}
d

Binocular and Monocular Regions



HMDs render two views offset by IPD; overlap provides retinal disparity for stereoscopic depth.

- **Binocular region:** overlap seen by both eyes; provides depth cues
- **Monocular region:** peripheral areas visible to one eye only

HMD Simulator Architecture

The HMD simulator (`https://github.com/singaporetech/hmd`) consists of these modules:

- `main.ts` — Entry point: WebGPU/WebGL engine init, render loop
- `app.ts` — Scene management: 5 cameras, environment loading, PIP viewports, display mode toggling
- `hmd.ts` — Core HMD class: optical parameters, off-axis projection matrix, barrel distortion shader, render targets
- `ui.ts` — Babylon.js GUI: parameter sliders, stats panel, toggle buttons
- `constants.ts` — Layer masks, Cardboard V2 presets, DisplayMode enum
- `frustumVisualizer.ts` — Frustum edge rendering as tubes, per-frame updates

Code: Magnification and Virtual Image

From `hmd.ts` — `calcProjectionMatrix()`

```
// magnification (f must be > distLens2Display)
this.magnification = this.f / (this.f - this.distLens2Display);
// virtual image height
this.imgHeight = this.displayHeight * this.magnification;
// aspect ratio
this.aspectRatio = this.displayWidth / this.displayHeight;
// thin lens: 1/f = 1/do + 1/di  =>  di
this.distLens2Img = Math.abs(1/(1/this.f - 1/this.distLens2Display));
// distance from eye to virtual image
this.distEye2Img = this.distLens2Img + this.eyeRelief;
this.near = this.distEye2Display;
// vertical FOV from virtual image height
this.fovVertical = 2 * Math.atan(this.imgHeight / 2 / this.distEye2Img);
```

Source: `hmd.ts` — `calcProjectionMatrix()`. Based on thin lens equations (see Week 06 recap).

Code: Off-Axis Projection Matrix

From `hmd.ts` — asymmetric frustum for Babylon.js (LHS)

```
// Off-axis projection (left eye shown)
let x = (2*this.near) / (this.rightForLeftEye - this.leftForLeftEye);
const y = (2*this.near) / (this.top-this.bottom);
let a = (this.rightForLeftEye+this.leftForLeftEye) /
        (this.rightForLeftEye-this.leftForLeftEye);
const b = (this.top+this.bottom)/(this.top-this.bottom);
const c = (this.far+this.near)/(this.far-this.near);
const d = (-2*this.far*this.near)/(this.far-this.near);
this.projMatL = Matrix.FromValues( x, 0, 0, 0,
                                   0, y, 0, 0,
                                   a, b, c, 1,
                                   0, 0, d, 0); // LHS col-major
```

Left/right bounds differ per eye (offset by IPD/2), creating an **asymmetric** horizontal frustum.

Refs: three.js Matrix4.makePerspective; Scratchapixel: Projection Matrix

Code: Barrel Distortion Shader (GLSL)

From `hmd.ts` — post-process fragment shader

```
const float k1 = 0.34; // Cardboard V2
const float k2 = 0.55;
const vec2 center = vec2(0.5, 0.5);
void main() {
    vec2 uv = vUV;
    vec2 delta = uv - center;
    float r2 = dot(delta, delta);
    vec2 distorted = delta*(1.0+k1*r2+k2*r2*r2);
    vec2 corrected = center + distorted;
    vec4 texColor = texture2D(textureSampler, corrected);
    float inBounds = step(0.0, corrected.x) * step(corrected.x, 1.0) * step(0.0, corrected.y
     )
    * step(corrected.y, 1.0);
    gl_FragColor = mix(vec4(0.55,0.38,0.4,1.0),texColor,inBounds);
}
```

Barrel distortion: $r' = r(1 + k_1 r^2 + k_2 r^4)$. Google Cardboard V2.

IPD and Depth Perception

Inter-Pupillary Distance (IPD)

- Software IPD controls the separation between virtual cameras
- Mismatch with physical IPD causes depth distortion
- Too large: world appears miniaturized (hyperstereopsis)
- Too small: world appears flat, reduced depth cues

In the simulator, the IPD slider adjusts this value and the stats panel shows the resulting nasal/temporal FOV split.

Lens Distortion Correction

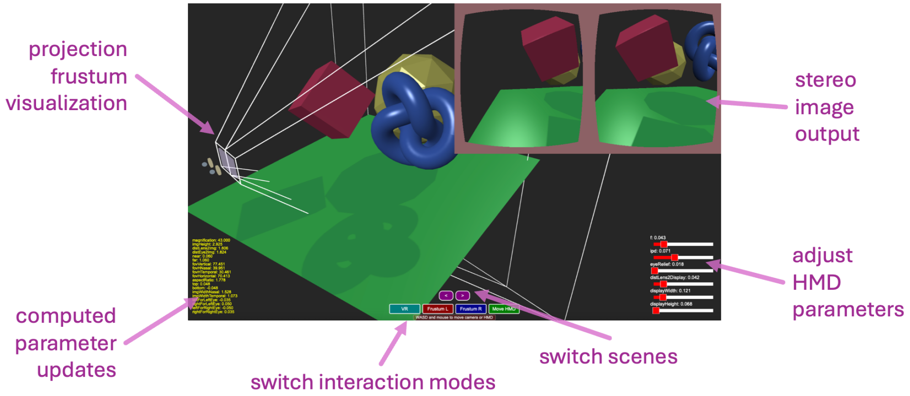
HMD lenses introduce **pincushion distortion** — straight lines bow inward.

- The simulator applies **barrel distortion** (inverse) as a GLSL post-process shader on each eye’s render target
- This pre-compensates so the image looks correct through the lenses
- Distortion coefficients k1=0.34, k2=0.55 are hardcoded from Google Cardboard V2 specifications
- The distortion is always active — there is no toggle to disable it

The greyish-pink border visible in VR mode is the out-of-bounds fallback region from the shader.

The HMD Simulator

Live: `https://hmd.diaversity.org` **Source:** `https://github.com/singaporetech/hmd`



```
magnification: 43.000
imgHeight: 2.925
distLens2Img: 1.806
distEye2Img: 1.824
near: 0.060
far: 1.060
fovVertical: 77.451
fovHNasal: 39.951
fovHTemporal: 30.461
fovHorizontal: 70.413
aspectRatio: 1.778
top: 0.048
bottom: -0.048
imgWidthNasal: 1.528
imgWidthTemporal: 1.073
leftForLeftEye: -0.035
rightForLeftEye: 0.050
leftForRightEye: -0.050
rightForRightEye: 0.035
```

```
f: 0.043
ipd: 0.071
eyeRelief: 0.018
distLens2Display: 0.042
displayWidth: 0.121
displayHeight: 0.068
```

Activity 1: Exploring the Simulator

Get oriented:

1. Open `https://hmd.diaversity.org` and use **left/right arrow keys** to switch scenes
2. Click **Move HMD** to enable it (also enables mouse rotation)
3. Use **WASD** to translate and **mouse** to rotate the HMD
4. Watch the frustum tubes and PIP viewports update in real-time

Observe:

- The frustum visualization follows the HMD position
- PIP viewport content changes as the HMD “looks” at different scene areas
- Move the HMD close to objects — notice how nearby objects **clip** (disappear) when they cross the near plane
- Toggle Move HMD off to return to orbiting the main camera

Activity 2: IPD and Frustum Overlap

Try this in the simulator:

1. Toggle **Frustum L** and **Frustum R** on
2. Move the **IPD** slider; watch frustum separation and the PIP viewports
3. Orbit the main camera to a **top-down** view; compare each frustum's shape
4. Push IPD to an extreme and observe the asymmetry

Observe:

- How does the binocular overlap region change with IPD?
- Each frustum is wider on the temporal side (off-axis projection)
- Check `fovHNasal` and `fovHTemporal` in the stats panel

Activity 3: Eye Relief and FOV

Try this in the simulator:

1. Keep frustum visualization on
2. Move the `eyeRelief` slider from low to high
3. Set eye relief to a high value and note FOV, then reduce to minimum
4. Check `fovVertical` and `fovHorizontal` in the stats panel

Observe:

- Reducing eye relief **widens** the FOV — why?
- Real HMDs need minimum eye relief for comfort and glasses clearance
- This is a key design trade-off: wider FOV vs physical comfort
- Wider FOV increases immersion but may increase distortion at edges

Activity 4: Focal Length

Focal length — try this in the simulator:

1. Move the **f** (focal length) slider slowly
2. Watch `distEye2Img` and `magnification` in the stats panel
3. Try setting f very close to `distLens2Display` — what happens?

Observe:

- When f approaches `distLens2Display`, magnification goes toward infinity
- If f < `distLens2Display`, the image inverts (real image, not virtual)
- In a real HMD, this inversion is invisible: the real image forms too close to the eye
- `distEye2Img` determines where the eye must **accommodate** (focus)
- Mismatch between accommodation and vergence causes discomfort (VAC)

Optics Explorer: `https://optics.diaversity.org` — visualize ray changes with focal length

Activity 5: Lens-to-Display Distance

Building on Activity 4:

Now see how lens-to-display distance interacts with focal length.

Try this:

1. Move the `distLens2Display` slider
2. Watch `magnification` and `near` in the stats panel
3. Bring it close to the focal length value

Observe:

- As `distLens2Display` approaches f, magnification grows rapidly
- Near plane (= eye-to-display distance) shifts accordingly
- If `distLens2Display` > f, image inverts (projector mode)

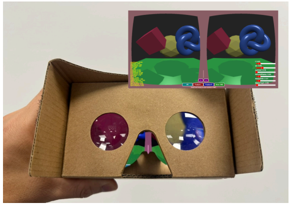
Activity 6: VR Mode & Cardboard

Building on Activities 4 and 5:
Now let's see the full stereo pipeline in action.
Try this:

1. Click the **VR** toggle to see side-by-side stereo output
2. Note how barrel distortion appear on each eye
3. Open the simulator on your **phone** at `https://hmd.diaversity.org`
4. Toggle **VR mode** and place the phone in a Google Cardboard viewer
5. Adjust IPD and eye relief; observe **stereopsis** — the perception of depth from binocular disparity

Observe:

- Barrel distortion should be mitigated through the lenses
- Compare depth perception (stereopsis) with different IPD settings



Activity 7: Impossible Rendering

Try creative or "impossible" configurations:

1. Set `f` **below** `distLens2Display` — observe the inverted (real) image
2. Push IPD to an extreme value (very large or negative if possible)
3. Set `distLens2Display` very close to `f` — watch magnification spike
4. Try extreme display dimensions (very wide, very tall) and observe the FOV

Advanced (code modification):

- Clone the HMD repo and add a **skew** term to the projection matrix in `calcProjectionMatrix()` to create an Escher-inspired impossible viewpoint
- What happens when left and right eyes see geometrically contradictory perspectives?

Slido: Describe what impossible or creative configuration you tried and what effect you observed.

Summary

Today we covered:

- HMD simulator codebase: 6 modules (`main.ts`, `app.ts`, `hmd.ts`, `ui.ts`, `constants.ts`, `frustumVisualizer.ts`)
- Key code: magnification/virtual image, off-axis projection matrix, barrel distortion shader
- How IPD, eye relief, focal length, display dimensions, and `distLens2Display` affect the visual output
- Troubleshooting rendering issues by adjusting simulator parameters

Next: Week 09 — Interaction theory and implementation