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

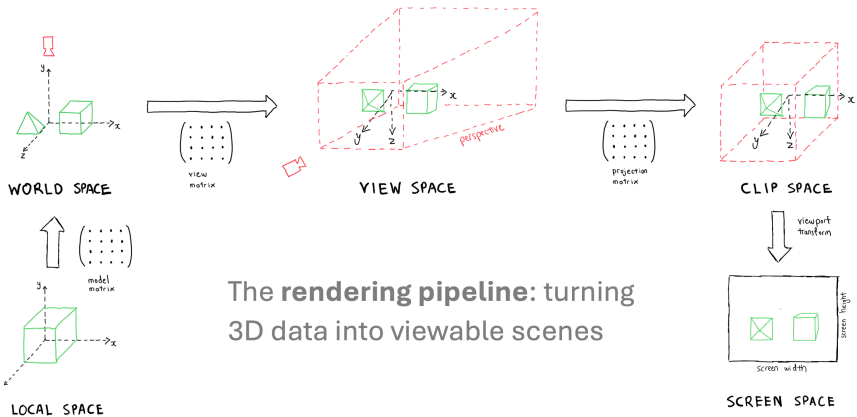
Recap: HMD Optics from Week 06

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

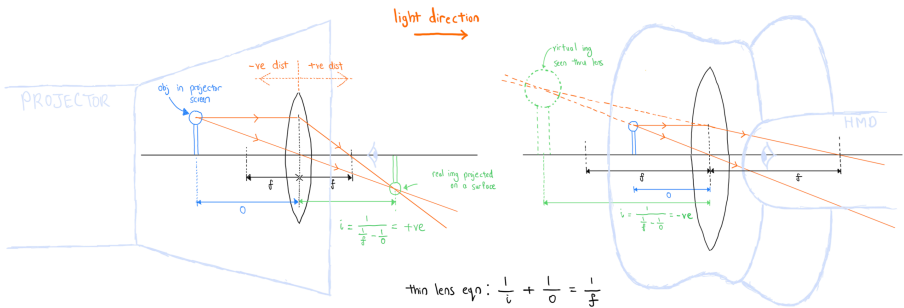
Today we go hands-on with these concepts using the HMD simulator.

Recap: Core Graphics Concepts

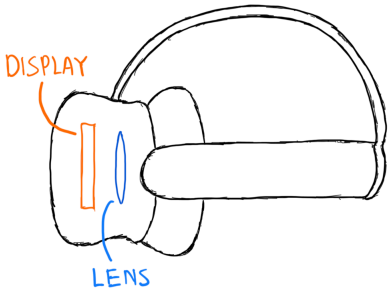


VR requires rendering this pipeline **twice per frame** at high FPS (e.g., 90+).

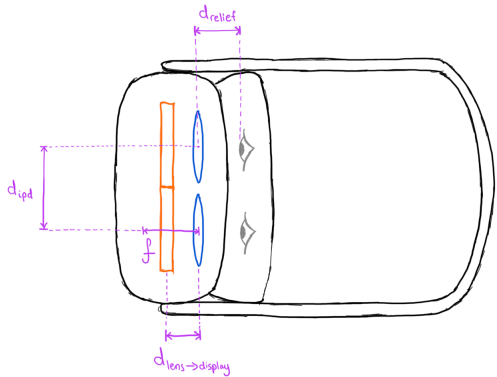
Recap: Core Physics Concepts



Recap: Key HMD Hardware Components

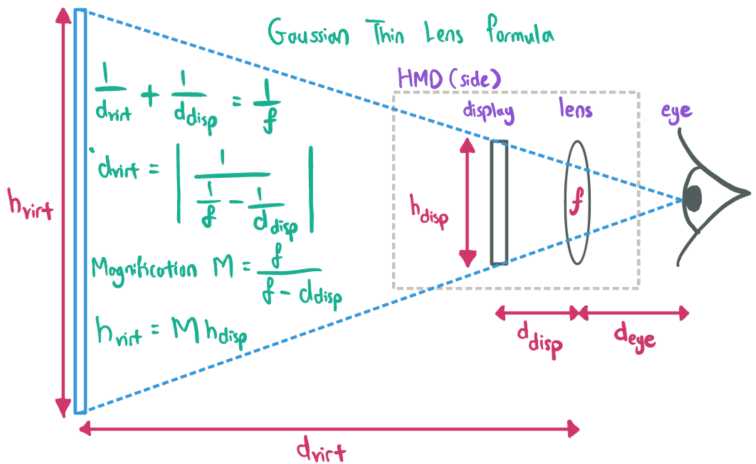


SIDE VIEW



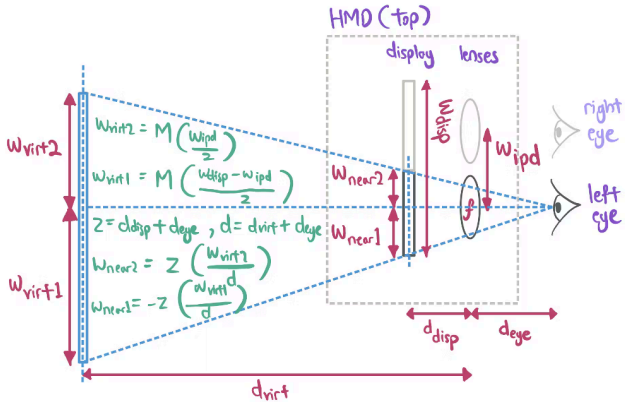
TOP VIEW

Recap: Virtual Image Dist & Height

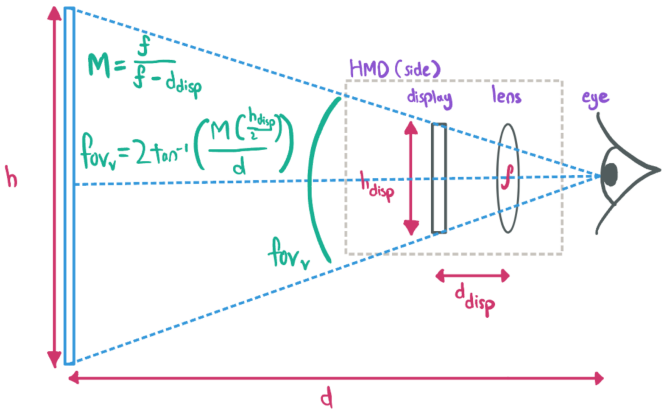


Video segment: https://youtu.be/OKD4jrnn4WE?t=406

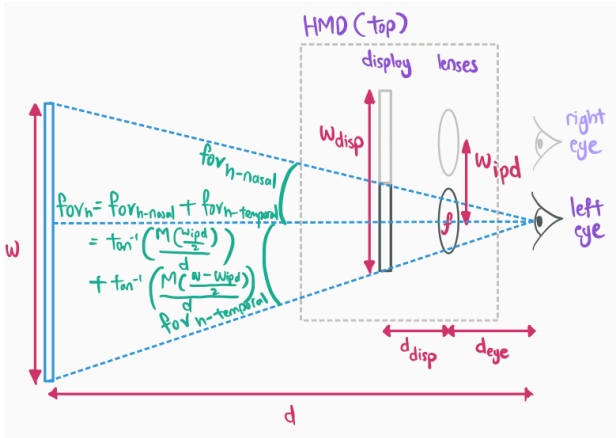
Computation: Virtual Image Width



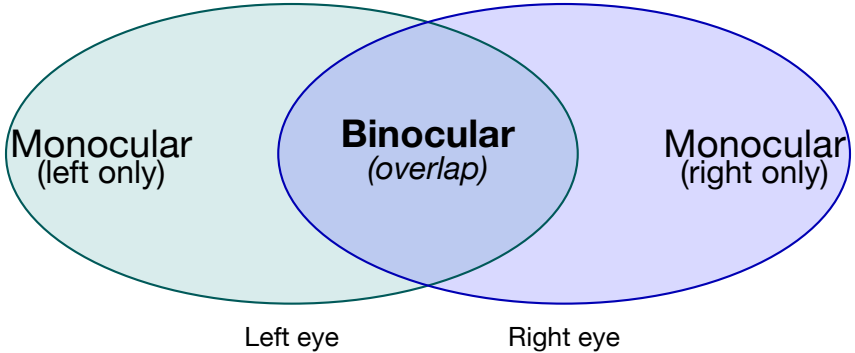
Computation: Vertical FOV



Computation: Horizontal FOV



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 stereoscopic depth cues
- **Monocular region:** peripheral areas visible to one eye only

HMD Simulator Architecture

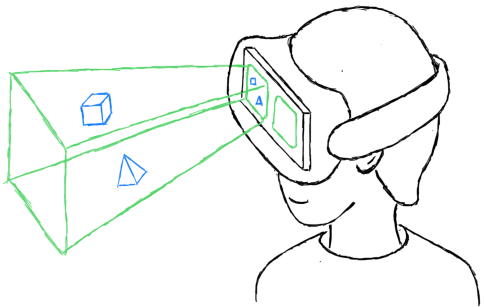
The HMD simulator (`https://github.com/singaporetech/hmd`) consists of these modules:

- `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

Stereo Rendering Pipeline

For each frame, the HMD simulator:

1. Read HMD parameters (IPD, eye relief, focal length, display dims)
2. Compute magnification and virtual image distance (thin lens)
3. Calculate left/right camera positions (offset by IPD/2)
4. Generate per-eye **off-axis asymmetric** projection matrices
5. Render scene twice (once per eye) to separate render targets
6. Apply barrel distortion as a post-process shader
7. Composite into side-by-side PIP viewports



Each step maps to code in `hmd.ts` and `app.ts`.

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; Song Ho Ahn

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);
    // radial distortion: r' = r(1+k1*r^2+k2*r^4)
    vec2 distorted = delta*(1.0+k1*r2+k2*r2*r2);
    vec2 corrected = center + distorted;
    vec4 texColor = texture2D(textureSampler, corrected);
    // bounds check via step() for WebGPU
    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: $r' = r(1 + k_1 r^2 + k_2 r^4)$. Coefficients from 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 (exposing k1/k2 as UI sliders is a planned feature)

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

Hands-on: Exploring HMD Rendering

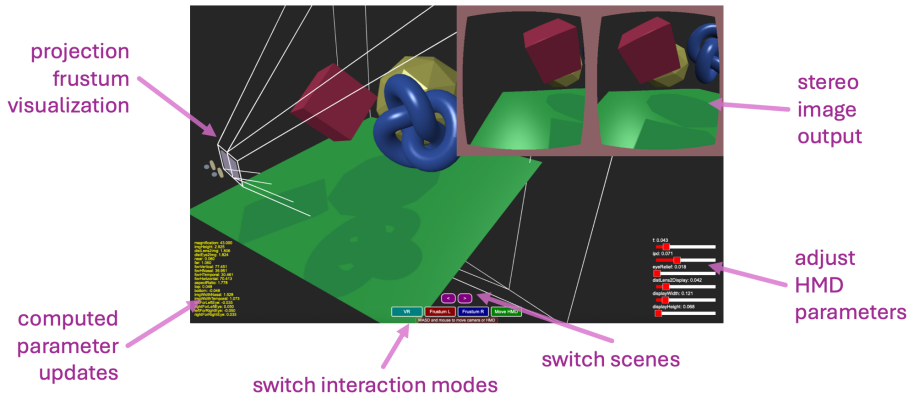
Now let's use the HMD simulator to explore how optical parameters affect rendering.

For each activity, you will adjust specific parameters, observe the visual and numerical effects, and answer a Slido question.

Open the simulator at `https://hmd.diaversity.org` and follow along.

The HMD Simulator

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



Hands-on: HMD Simulator Controls

Available controls:

- **6 sliders:** focal length (`f`), IPD, eye relief, lens-to-display distance, display width, display height
- **Toggle buttons:** VR mode, Frustum L, Frustum R, Move HMD
- **Environment switcher:** < > buttons to cycle scenes
- **Camera:** WASD + mouse to orbit the main camera
- **Stats panel:** shows calculated values (`magnification`, FOV, `distEye2Img`, etc.)

Each activity below will ask you to try something specific, followed by a Slido question.

Activity 1: IPD and Frustum Overlap

Try this in the simulator:

1. Toggle **Frustum L** and **Frustum R** on
2. Move the **IPD** slider from minimum to maximum; 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 2: Adjust Eye Relief

Try this in the simulator:

1. Keep frustum visualization on
2. Move the `eyeRelief` slider from low to high
3. Observe the frustum shape and the PIP viewports

Observe:

- As eye relief increases, what happens to the FOV?
- Check `fovVertical` and `fovHorizontal` in the stats panel
- Think about why glasses wearers need more eye relief

Activity 3: Change Focal Length

Try this in the simulator:

1. Move the **f** (focal length) slider slowly
2. Watch the stats panel values, especially `distEye2Img` and `magnification`
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)
- The virtual image distance (`distEye2Img`) determines where your eyes focus

Activity 4: Toggle VR Mode

Try this in the simulator:

1. Click the **VR** toggle button
2. Observe how the viewport layout changes
3. Toggle it off and compare

Observe:

- In Simulation mode: small PIP viewports show left/right eye views alongside the 3D scene
- In VR mode: left and right eye views fill the entire screen side-by-side
- The barrel distortion is visible in both modes on the PIP/eye views
- The greyish-pink border is the shader's out-of-bounds fallback

Activity 5: Move HMD with WASD

Try this in the simulator:

1. Click the **Move HMD** toggle to enable it
2. Use **WASD** keys to translate the HMD in the scene
3. 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 parts of the scene
- Toggle Move HMD off to return to orbiting the main camera

Activity 6: Display Settings and Pixel Density

Try this in the simulator:

1. Adjust `displayWidth` and `displayHeight` sliders; watch the frustum shape and stats panel
2. Set `displayWidth` to minimum, note PIP sharpness, then increase to maximum
3. Compare the angular pixel density at each extreme

Observe:

- Wider/taller display increases horizontal/vertical FOV respectively
- Wider display spreads pixels over a larger FOV, reducing **angular pixel density**
- Real HMDs use higher-resolution panels or foveated rendering to compensate
- Check `aspectRatio` and `fovHorizontal` in the stats panel

Activity 7: Adjust Lens-to-Display Distance

Try this in the simulator:

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

Observe:

- As `distLens2Display` approaches f, magnification grows rapidly
- The near plane (= eye-to-display distance) shifts accordingly
- If `distLens2Display` > f, you get a real inverted image (projector mode)
- Thin lens equation: $M = \frac{f}{f - d_o}$

Activity 8: Observe the Stats Panel

Try this in the simulator:

1. Adjust **f** (focal length) slowly upward
2. Watch the `distEye2Img` value in the stats panel
3. Note how it corresponds to where your eyes would focus

Observe:

- As focal length increases, the virtual image moves further away
- `distEye2Img` represents the vergence distance (where eyes converge)
- In real HMDs, this determines the accommodation demand on the eye
- Mismatch between vergence and accommodation causes discomfort (VAC)

Activity 9: Eye Relief and FOV Trade-off

Try this in the simulator:

1. Set eye relief to a high value and note the FOV values
2. Reduce eye relief to minimum and compare
3. Think about why this matters for glasses wearers

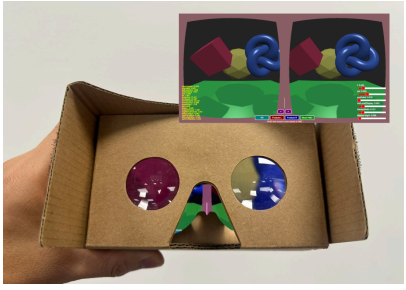
Observe:

- Reducing eye relief **widens** the FOV
- But real HMDs need minimum eye relief for comfort and glasses clearance
- This is a key design trade-off in HMD engineering

Activity 10: View Through Google Cardboard

Try this:

1. Open the simulator on your **phone** at `https://hmd.diaversity.org`
2. Toggle **VR mode** on for side-by-side stereo
3. Place phone in a Google Cardboard viewer and look through the lenses
4. Adjust IPD and eye relief to match your comfort



Observe:

- Barrel distortion should look correct through the lenses
- Greyish-pink borders disappear through lens optics
- Compare depth perception with different IPD settings

Activity 11: 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