Foveated Stereo Rendering in Blender: A Unified Sensor Framework

Luiz Velho
with Claude (Anthropic)
VISGRAF Lab, IMPA

Technical Report · course reading · revised 13 November 2026
Software: https://github.com/visgraf/fov-render

Abstract

A camera samples a scene uniformly; an eye does not. This report describes *fov-render*, a framework that turns Blender's Cycles path tracer into a binocular, gaze-directed observer. It is open source (CC0) at `https://github.com/visgraf/fov-render` [1]: a Python package, `fov_render`, and a command line, `fov-render`. Two eyes on a head that may move between looks can each carry one of four sensors: a log-polar foveated camera, a perspective tangent-plane camera, an equirectangular panorama and a wide reference view. Every look, whatever the sensor, returns the same record: one ray per sample with its direction, radiance, solid-angle footprint and, as oracle, the surface it hit. The framework unifies two research code bases from the VISGRAF laboratory, fov-3d-vision [5] and f3d-vision [9], and was built by migrating their rendering code into one package under checks against the code it replaced.

The reader should come away with five things: why foveated, active sampling is worth simulating (Section 1); how the framework is layered (Section 2); the geometry and sampling mathematics behind each sensor and each measurement (Sections 3–7); how to drive it from Python and from the shell, and how to bring in a scene of one's own (Section 8); and how its correctness is checked (Sections 9–10). Implementation detail is kept at the level of modules and interfaces; the code is the reference for the rest.

1 Motivation

Foveation is first a bandwidth decision and only then an attention mechanism: an observer that samples finely where it looks and coarsely elsewhere sees the same field with about fifty times fewer rays, and must then decide where to look.

1.1 The bandwidth argument

Take a field of 45° around the gaze and a foveal spacing of $s_0 = 0.05°$ per sample. A uniform sampler at that spacing needs about 2.5 million rays. The log-polar sensor of Section 4 covers the same field with 50,269 rays: **50.8× fewer**, as measured in fov-3d-vision's step

A3 [7]. The saving comes from where the samples go: 16% of them land within 5° of the gaze, a cone that holds only 1.3% of the field's solid angle (0.19% of the full sphere).

The price is that most of the scene is seen coarsely at any moment. Fine detail must be collected by moving the gaze, so sensing becomes a sequence of decisions.

1.2 The active loop

Both code bases organise vision as a loop: **fixate** (render a binocular look), **reconstruct** (turn the fovea's matches into metric 3-D points), **update memory** (fuse them into a persistent map and mark the sphere of directions already seen) and **saccade** (choose the next gaze). Two results show what the loop buys:

- In fov-3d-vision, fifty foveated fixations of the Classroom scene measured 92% of a 60° field of regard at a median inverse-depth error of $0.04\,\mathrm{m}^{-1}$, with coverage-first, random and information-seeking policies within 12% of each other [8].
- In f3d-vision, a greedy explorer with perspective foveae saw 100.00% of the sphere around the head and reconstructed depth for 99.26% of it, in 356 fixations [9, 4].

1.3 Why a simulator with an oracle

A rendered scene knows, for every ray, the point it hit and the object that point belongs to. That oracle lets an experiment separate three sources of error that a real camera mixes together: the sensor, the stereo matcher and the gaze policy. A path tracer also produces physically plausible radiance, with noise whose level is set by the number of samples, so sensing cost can be controlled and measured (Section 7).

1.4 Why one framework

The two code bases grew apart. fov-3d-vision built the log-polar foveated camera and a spherical stereo field; f3d-vision built a 12° perspective pair with oracle correspondence, a persistent surfel map and a render server. Questions that cross the line, such as whether the log-polar sensor changes the explorer's coverage at equal rays, need both sensors behind one rig, one render configuration and one output record. That is what this framework provides, for any `.blend` scene.

Three working rules carry over from both projects and shape the design: every number is either measured or labelled an estimate; every component ships a check that can fail; and data the observer may see is kept apart from data only the evaluator may see.

1.5 The software

The framework is the repository `visgraf/fov-render` [1], released under CC0 1.0 like the two code bases it comes from. It needs Blender 5.2.1 LTS (Cycles), and either the Blender binary or the matching `bpy` wheel for Python 3.13; the numpy half runs without Blender. It installs as a package or is used as a git submodule:

```
git clone https://github.com/visgraf/fov-render
cd fov-render
pip install -e ".[bpy,checks]"     # with the bpy wheel and OpenEXR; "." alone: numpy only
fov-render selftest --device CPU   # the package's own checks (Sec. 9)
```

The tag `v0.1.0` marks the end of the migration from the two code bases (Section 10); the command line of Section 8.7 came after it (issue #18), and so did head motion (Section 3.5, issue #20). The repository's `README.md` is the user's guide, `docs/PLAN.md` records the design decisions and `docs/HISTORY.md` the migration. This report describes the design and its reasons; where the two differ in a detail, the repository is right.

2 Architecture

The framework is a stack of layers that every sensor shares, with only the camera swapped per look: a caller sends a request, one handler inside Blender renders it, and one record per eye comes back (Figure 1).

Each layer has one job:

- **Scene.** Opens any `.blend` file once per session and takes the session's head from the scene's `EYE` object. It removes anything in the file that would override the framework's settings (an animated seed, extra render passes), numbers every renderable object so the Object Index pass can name it, and runs the sanity checks of Section 9.
- **RenderConfig.** One frozen set of Cycles settings for the session: samples per pixel, seeds, a box pixel filter, no adaptive sampling or denoising, the GPU back-end. After every render the settings Cycles actually used are read back and stored with the look (Section 7).
- **Rig.** Turns a gaze request and the look's head into a world pose for each eye, with one formula for every sensor (Section 3).
- **Sensors.** Each sensor has two halves: a numpy *model* that maps pixels to rays and gives each pixel's footprint, and a Blender *builder* that creates the camera. Analysis code uses the model without Blender (Sections 4–5).
- **Record.** Converts the rendered passes into a `SampleRecord`, one entry per valid sample, with each field labelled as observation or oracle, behind an access gate (Section 8.3). References are panorama looks rendered at foveal spacing and kept for evaluation.
- **Execution.** A `Session` either runs Blender as a persistent background server fed one JSON request per look, or, inside a Blender script, calls the same handler directly (Section 8.1). The `fov-render` command line sends the same requests through a `Session` (Section 8.7).

Two properties follow from this layout. The record does not depend on the sensor, so downstream code (a matcher, a map, a gaze policy) can switch sensors without change. And the observer's view and the evaluator's truth are separated by construction, not by convention: an observation session cannot read an oracle field.

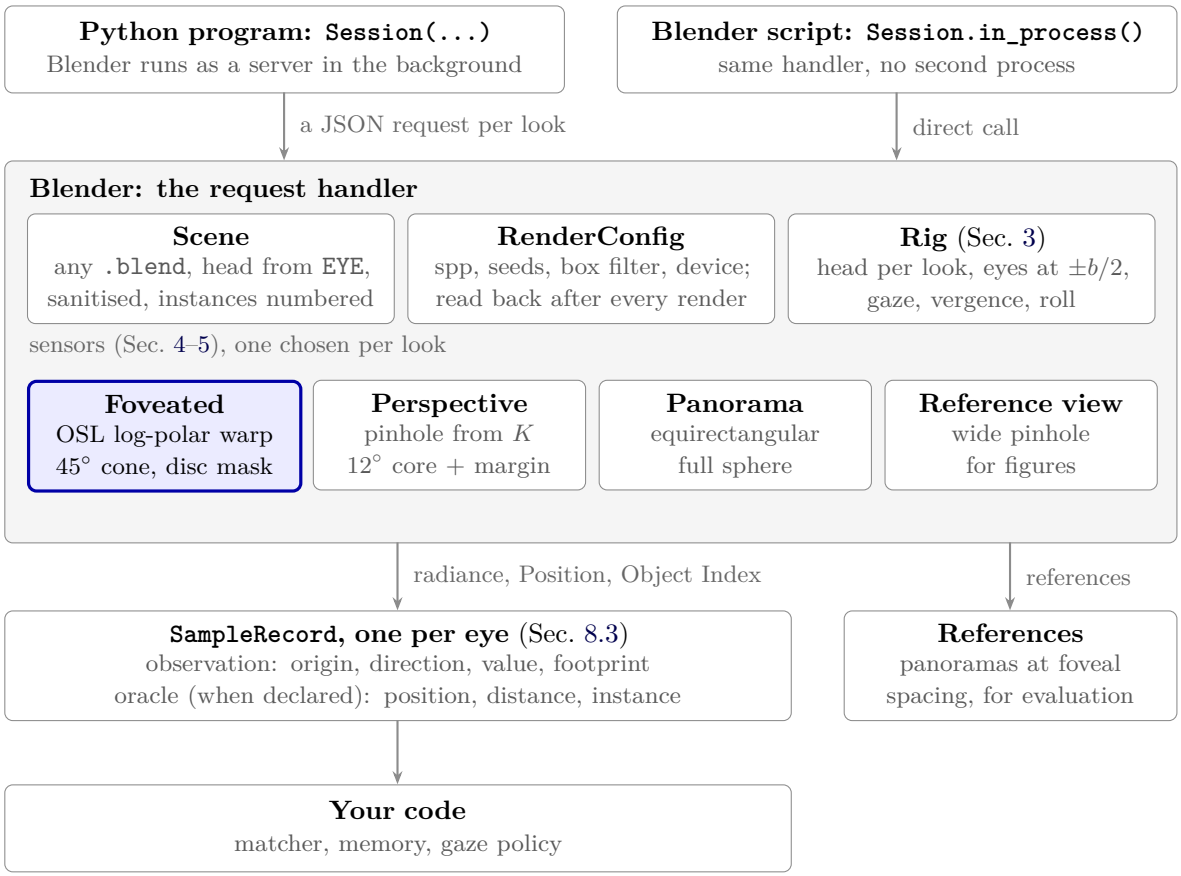


Figure 1: Architecture of fov-render (package `fov_render`): two entry points, one request handler, four sensors and one record per eye; the command line of Section 8.7 is a third caller of `Session`. The foveated sensor is highlighted because it is the one sensor no off-the-shelf camera provides.

In the repository the layers are modules (Table 1). Everything that does not need Blender is plain numpy and runs anywhere; the Blender half lives in `fov_render/bl/` and imports only `bpy`, `mathutils`, numpy and the standard library.

Module	What it holds
`profiles`, `warp`	the `small` and `full` profiles; the log-polar warp of Section 4
`rig`, `geometry`	frames, gaze, vergence, roll (Section 3); epipolar angles and triangulation (Section 6)
`sensors/`	the four sensor specs and their numpy models; the OSL shader of the foveated camera
`record`, `exr_lite`	`SampleRecord` and its access gate; a reader for uncompressed EXR files
`request`, `session`	the JSON schemas and their validation; `Session`, `Look`
`server`, `bl/handler`	the persistent render server and the request handler (Blender side)
`bl/`	`config` (`RenderConfig`, presets), `engine`, `passes`, `cameras`, `scene` (opening, sanitising, numbering, audits, sanity), `eye_scene`
`cli`, `blcli`, `launch`	the `fov-render` command line and how it finds Blender
`checks/`	the gates of Section 9, with the known answers they compare against

Table 1: The package `fov_render`. The Blender-side modules run in either interpreter: the Blender binary (`blender -b -P`) or Python with the `bpy` wheel.

3 Coordinate frames and the binocular rig

All sensors share one rig: a head with two eyes that rotate about their own centres. The head may move between looks but not within one, so every look is fully described by its head pose, which eye, a gaze direction and a roll convention.

3.1 Frames

- **World W.** Blender's frame, in metres, with $+Z$ up.
- **Head H.** A rigid frame with $+X$ to the right (along the baseline), $+Y$ up and $-Z$ forward, given by a rotation R_{WH} and an origin $\mathbf{o}$, the cyclopean point. The session's head is the scene's `EYE` object; a look may carry its own (Section 3.5).
- **Eye camera E.** Blender's camera convention: the camera looks along its local $-Z$ with $+Y$ up and $+X$ right. Computer-vision code uses the OpenCV convention (x right, y down, z forward); the two differ by $C = \mathrm{diag}(1, -1, -1)$.

The eye centres lie on the head's X axis, a baseline b apart ($b = 63\,\mathrm{mm}$ by default):

$$\mathbf{c}_{L,R} = \mathbf{o} \mp \tfrac{b}{2}\, R_{WH}\, \hat{\mathbf{x}}. \tag{1}$$

3.2 Gaze

A gaze is a yaw ψ (positive to the right) and a pitch θ (positive up), measured in the head frame. The gaze direction is

$$\mathbf{d}(\psi, \theta) = (\sin\psi\cos\theta,\ \sin\theta,\ -\cos\psi\cos\theta), \tag{2}$$

and its inverse is $\psi = \mathrm{atan2}(d_x, -d_z)$, $\theta = \mathrm{atan2}(d_y, \sqrt{d_x^2 + d_z^2})$. The same formula, read with ψ as longitude and θ as latitude, is the panorama's pixel-to-direction map (Section 5.2), which is why every sensor agrees on what "straight ahead" means.

The world pose of an eye camera is one composition, used by every sensor:

$$M_{\text{eye}} = T(\mathbf{o})\; R_{WH}\; T\big(\pm\tfrac{b}{2}\hat{\mathbf{x}}\big)\; R_y(-\psi)\; R_x(\theta)\; R_{\text{roll}}. \tag{3}$$

Applying $R_y(-\psi)\,R_x(\theta)$ to the camera's forward axis $(0, 0, -1)$ gives exactly $\mathbf{d}(\psi, \theta)$: yaw is applied after pitch, so the elevation of the gaze equals θ. Within a look the head is fixed and only the eyes rotate; between looks R_{WH} and $\mathbf{o}$ may change (Section 3.5).

3.3 Vergence

A binocular look fixates one point $\mathbf{P}$. Each eye's gaze is the unit vector from its centre to $\mathbf{P}$, expressed in the head frame and converted to (ψ_i, θ_i) by the inverse above:

$$\mathbf{d}_i = R_{WH}^{\top}\,\frac{\mathbf{P} - \mathbf{c}_i}{\|\mathbf{P} - \mathbf{c}_i\|}, \qquad i \in \{L, R\}. \tag{4}$$

The vergence angle is $\gamma = \arccos(\mathbf{d}_L \cdot \mathbf{d}_R)$. For a point straight ahead at distance D it is $\gamma = 2\arctan(b/2D)$; at the 2.10 m used by the perspective experiments that is 1.72°. A look can be requested three ways: by a world point $\mathbf{P}$, by a cyclopean gaze plus a distance D (then $\mathbf{P} = \mathbf{o} + D\,\mathbf{d}$), or by giving each eye its own (ψ, θ).

3.4 Roll

A gaze direction fixes the camera's forward axis but not its rotation about that axis. The framework offers two conventions, and each fails in a different place:

- **Upright** (zero torsion): the image x axis is kept in the head's horizontal plane, $\hat{\mathbf{x}} = \text{normalize}(\hat{\mathbf{z}} \times \hat{\mathbf{y}}_H)$, with $\hat{\mathbf{z}}$ the forward axis in OpenCV convention. It is undefined when looking straight up or down.
- **Baseline-projected**: the head's baseline direction $\mathbf{h} = \hat{\mathbf{x}}_H$ is projected into the image plane, $\hat{\mathbf{x}} = \text{normalize}\big(\mathbf{h} - (\mathbf{h} \cdot \hat{\mathbf{z}})\,\hat{\mathbf{z}}\big)$. It is undefined only when looking along the baseline, where binocular depth is undefined anyway, which makes it the right choice for stereo all around the sphere.

For a verged pair the fixation's epipolar plane contains the baseline and both gaze axes, so baseline-projected roll keeps that plane on the central image row of both eyes. The foveated sensor's warp is circularly symmetric, so roll only rotates its disc and does not change what it measures; the perspective pair's rectification depends on it. Roll is therefore a property of the rig, recorded with every look, not of the sensor.

3.5 Head motion

The head moves between looks, never within one. A look's request may carry its own head pose $(R_{WH}, \mathbf{o})$; without one the look uses the session's head, the scene's `EYE`. Every formula of this section then holds per look, with that look's head, and both eyes of a look share it.

Relative moves. `HeadPose.moved` moves the head in its own frame: the origin goes to the head-frame point $\mathbf{t}$, then the head turns about its new origin,

$$R'_{WH} = R_{WH}\, R_d, \qquad R_d = R_y(-\psi_h)\, R_x(\theta_h)\, R_z(-\phi_h), \qquad \mathbf{o}' = \mathbf{o} + R_{WH}\, \mathbf{t}, \tag{5}$$

with the gaze's signs: $+\psi_h$ turns right, $+\theta_h$ tilts up and $+\phi_h$ lowers the right ear. The turned head's forward axis is $R_{WH}\, \mathbf{d}(\psi_h, \theta_h)$, as for an eye's gaze in Eq. (2). `HeadPose.translated` moves the origin by a world vector and does not turn.

The gaze under head motion. The gaze requests of Section 3.3 are of two kinds. A yaw–pitch gaze, cyclopean or per eye, is measured in the head frame, so it turns with the head, as eyes held still in their sockets do. A verged point is a world point, and Eq. (4) aims both eyes at it from wherever the head now is, so the eyes stay on the target while the head moves, as the vestibulo-ocular reflex keeps them. Each record stores its look's head, and the eye centre of Eq. (1) for that head is the record's origin; the response and `meta.json` carry both the look's head and the session's.

A field of the request. The head is part of each request, not state kept by the server (decision D19 in `docs/PLAN.md`). A look therefore stays self-contained and can be replayed from its request alone; a refused or failed request cannot leave the head somewhere unknown; and a request without a head is byte for byte what it was before head motion existed. A caller that thinks in relative moves uses `Session.move_head`, which keeps a current head on the client side and puts it in every later request (Section 8.6). There is no body frame (D20): a caller with a body or neck model composes it into the head pose of each look.

The cyclopean eye. The rig's third eye, C, sits at the head origin $\mathbf{o}$. A foveated look on C alone is a monocular sensor on the head: with gaze $(0, 0)$ it is fixed to the head, and with a yaw–pitch gaze it is an eye that saccades within it. With upright roll, an eye turned in a still head and the same turn of the head with the eye straight ahead give the same camera pose, to 10^{-16} (Chat sandbox, numpy), so a turn can be split between eye and head freely. Baseline-projected roll does not share this property: the baseline turns with the head but not with an eye, so once yaw and pitch combine the image rolls about the gaze, by 8.3° at $(35°, 12°)$ and 19.5° at $(-40°, 25°)$. Monocular work uses upright roll.

The pivot of a turn. Eq. (5) turns the head about $\mathbf{o}$, which is where C is: a pure head turn does not move C and produces no motion parallax. A real head turns about the neck, some centimetres behind and below the eyes. A turn R_d about a pivot $\mathbf{p}$, given in the head frame, is the move of Eq. (5) with $\mathbf{t} = \mathbf{p} - R_d\, \mathbf{p}$, which leaves the pivot in place:

$$\mathbf{o}' = \mathbf{o} + R_{WH}\, (\mathbf{p} - R_d\, \mathbf{p}), \qquad R'_{WH} = R_{WH}\, R_d. \tag{6}$$

This is the first step of a neck model, and for now it is the caller's.

4 The foveated log-polar warp

The foveated sensor is a Cycles camera, written in OSL, that maps a square raster onto a cone of directions so that the spacing between samples grows linearly with eccentricity. One parameter, E_2, sets the budget; the field's half-angle $e_{\max}$ costs only logarithmically.

4.1 The spacing law

Let e be the angle between a ray and the gaze. The sensor places samples with spacing

$$s(e) = s_0 \left(1 + \frac{e}{E_2}\right), \tag{7}$$

where s_0 is the foveal spacing and E_2 is the eccentricity at which the spacing has doubled. This is the linear form used to describe cortical magnification in primate vision. The framework uses $E_2 = 2°$ and $e_{\max} = 45°$; fov-3d-vision found its disparity objective flat in E_2 (decision D16 [6]), so E_2 is chosen for cost, not accuracy.

4.2 From the spacing law to the raster

Walking outward from the centre, each pixel advances the eccentricity by one spacing, $de/dw = s(e)$. Integrating gives the raster radius, in pixels, at which eccentricity e is reached:

$$w(e) = \int_0^e \frac{de'}{s(e')} = \frac{E_2}{s_0} \ln\!\left(1 + \frac{e}{E_2}\right). \tag{8}$$

The raster must reach $e_{\max}$ at its rim, so it is $N = 2\,w(e_{\max})$ pixels across:

$$N = 2\,\frac{E_2}{s_0} \ln\!\left(1 + \frac{e_{\max}}{E_2}\right). \tag{9}$$

Writing $r = w(e)/w(e_{\max}) \in [0, 1]$ for the normalised radius and inverting gives the mapping the shader evaluates:

$$e(r) = E_2 \left[\left(1 + \frac{e_{\max}}{E_2}\right)^r - 1\right]. \tag{10}$$

s_0 has dropped out: the shader needs only E_2 and $e_{\max}$, and the foveal spacing is set by the resolution chosen, $s_0 = 2E_2 \ln(1 + e_{\max}/E_2)/N$. The two profiles are `small` ($s_0 = 0.1°$, $N = 126$) and `full` ($s_0 = 0.05°$, $N = 253$).

Pixel to ray. For a raster position $(u, v) \in [0, 1]^2$, let $\mathbf{p} = (u - \frac{1}{2}, v - \frac{1}{2})$, $r = 2\|\mathbf{p}\|$ and $\varphi = \operatorname{atan2}(p_y, p_x)$. In the shader's frame ($+Z$ forward) the ray is

$$\mathbf{d} = (\sin e \cos\varphi,\ \sin e \sin\varphi,\ \cos e), \qquad e = e(r). \tag{11}$$

Pixels with $r > 1$ receive zero throughput. They are masked by the analytic test $r \le 1$, never by alpha, which Cycles leaves at 1 even where no radiance arrives.

4.3 Footprint and anisotropy

Let $\rho = rN/2$ be the radius in pixels. Along the radius, consecutive pixels are one spacing $s(e)$ apart by construction. Around the circle they are $\sin(e)/\rho$ apart, which is smaller, so peripheral pixels are elongated radially. The solid angle one pixel covers is the product:

$$\omega(\rho) = \frac{\sin e}{\rho}\,\frac{de}{d\rho}, \qquad \frac{de}{d\rho} = \frac{2}{N}\,\ln\!\left(1 + \frac{e_{\max}}{E_2}\right)(e + E_2). \tag{12}$$

At the centre both factors tend to s_0, so $\omega \to s_0^2$. At the rim of the `full` profile the radial spacing is 1.17° and the tangential spacing 0.32°, an elongation of 3.7, and one pixel covers about $151\,s_0^2$. Because spacing alone does not describe a non-square pixel, the record carries each sample's footprint ω (Section 7.2).

4.4 What the warp saves

The disc holds about $\pi N^2/4$ samples: 50,269 at the `full` profile. A uniform sampler needs about Ω/s_0^2 samples for the cone of solid angle $\Omega = 2\pi(1 - \cos e_{\max}) = 1.84\,\mathrm{sr}$, which estimates 2.42 million; counting the pixels of a uniform raster at s_0, fov-3d-vision measured 2,553,563, hence the 50.8× of Section 1. The ratio does not depend on s_0, since both counts scale as $1/s_0^2$.

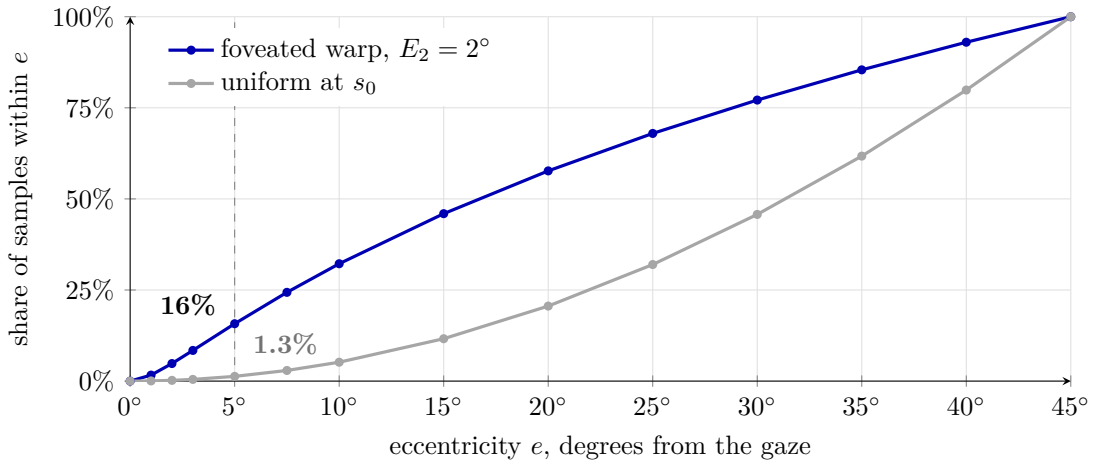


Figure 2: Within 5° of the gaze the foveated warp places 16% of its samples, uniform sampling 1.3%. Cumulative share of samples inside eccentricity e over a 45° field, computed from Eqs. (8)–(10).

The share of samples inside eccentricity e is $r(e)^2$ for the warp and $(1-\cos e)/(1-\cos e_{\max})$ for uniform sampling (Figure 2). The warp puts 16% of its samples within 5° of the gaze, twelve times the uniform share. Because the raster size grows with $\ln(1+e_{\max}/E_2)$, widening the field is cheap: from 20° to 45° costs 1.7× in raster area, and from 45° to 80° a further 1.4×.

5 Perspective and panoramic sensors

The other three sensors are standard Blender cameras described by the same three quantities as the warp: a pixel-to-ray map, a footprint per pixel and a valid region. That shared description is what lets one record hold any of them.

5.1 The perspective tangent pair

Each eye is a pinhole camera on the plane tangent to the sphere of directions at the gaze. Its square raster has a **core** that spans 12° and a **margin** of $\frac{3}{4}$ of the core on every side:

$$n = c + 2 \cdot \tfrac{3}{4}c, \qquad f = \frac{c}{2\tan 6^\circ}, \qquad K = \begin{pmatrix} f & 0 & \frac{n-1}{2} \\ 0 & f & \frac{n-1}{2} \\ 0 & 0 & 1 \end{pmatrix}, \tag{13}$$

with core $c = 128$ pixels (`small`) or 256 (`full`). At `full` that gives $n = 640$, $f = 1217.8$ pixels, a raster about 29.4° across and a spacing of $1/f = 0.047^\circ$ at the centre, close to the warp's $s_0 = 0.05^\circ$. Blender is given the same camera through its lens length, lens $= f \cdot 36\,\text{mm}/n$ on a 36 mm sensor.

In the OpenCV frame the ray through pixel (u, v), counted from the top-left pixel centre, is

$$\mathbf{d} \propto K^{-1}(u, v, 1)^\top = \left(\frac{u - c_x}{f},\ \frac{v - c_y}{f},\ 1 \right), \tag{14}$$

and the matrix C of Section 3.1 converts it to Blender's camera frame. A pixel at offset $(x, y) = (u - c_x, v - c_y)$ from the principal point covers the solid angle

$$\omega(x, y) = \frac{f}{(f^2 + x^2 + y^2)^{3/2}}, \tag{15}$$

which is $1/f^2$ at the centre and only 18% smaller at the corner: unlike the warp, this sensor is nearly uniform over its small field. The margin exists for the stereo matcher, which rectifies the pair and searches for correspondences there; only measurements in the central core of the rectified raster are accepted.

5.2 The equirectangular panorama

The panorama sees the whole sphere from an eye centre. For a $W \times H$ raster with $H = W/2$ and normalised pixel centre (u, v), $v = 0$ at the top row,

$$\lambda = (u - \tfrac{1}{2})\, 2\pi, \quad \beta = (\tfrac{1}{2} - v)\, \pi, \quad \mathbf{d} = (\sin\lambda\cos\beta,\ \sin\beta,\ -\cos\lambda\cos\beta), \tag{16}$$

which is the gaze formula (2) with longitude λ as yaw and latitude β as pitch: the panorama's centre pixel is the head's primary gaze. Each pixel covers

$$\omega(\beta) = \frac{2\pi}{W}\,\frac{\pi}{H}\,\cos\beta, \tag{17}$$

and the footprints sum to 4π sr, a check every panorama passes (Section 9). The panorama has four uses: the uniform reference at foveal spacing ($W = 7200$ gives 0.05° at the equator), the noise-floor measurement, the post-hoc evaluation sphere (720×360, 0.5° cells) and, rendered from each eye centre, the per-eye truth for stereo. An optional second render overrides every material with its back-face flag, so a scene whose set dressing is seen from behind can be rejected.

5.3 The sensors side by side

Sensor	Rays per eye at `full`	Spacing at the gaze	Field	Used for
foveated	50,269	0.05°	45° half-angle cone	fixations in the active loop
perspective	409,600 (65,536 in the core)	0.047°	12° core, 29.4° raster	local stereo with oracle correspondence
panorama	25.9 million (7200×3600)	0.05° at the equator	full sphere	references, noise floor, evaluation
reference view	4.9 million (2400×2048)	0.024°	54° wide	figures

Table 2: The four sensors at the `full` profile.

The reference view is the perspective model of Section 5.1 with a wide field (half-angle 27° horizontally) and no margin; it renders what a reader of a figure expects to see.

6 From pixels to geometry

A binocular look becomes 3-D geometry through two angles per matched ray pair: the epipolar plane they share and the parallax between them. Inverse depth is then, to first order, proportional to the parallax, and that fact sets both the measurement and its uncertainty.

6.1 What a render returns

Each pixel carries three passes: **Combined**, the radiance L the eye sees; **Position**, the world point $\mathbf{X}$ where the pixel's ray first hit a surface; and **Object Index**, the instance k of that surface (0 for a miss). From them the record derives the range and checks the sensor model:

$$r = \|\mathbf{X} - \mathbf{c}\|, \qquad \mathbf{X} \approx \mathbf{c} + r\,\mathbf{d}, \tag{18}$$

where $\mathbf{c}$ is the eye centre and $\mathbf{d}$ the ray the sensor model predicts for that pixel. Range is computed from Position rather than read from Cycles' Depth pass, because Depth is the ray length for panoramic and custom cameras but need not be for a perspective one; Position means the same thing for every sensor.

Position and Object Index are **oracle** data: they describe the first surface hit, which a real sensor cannot observe. An observer that is meant to learn from images reads only

radiance and the sensor model; an experiment that declares oracle correspondence may also read the oracle fields (Section 8.3).

6.2 Oracle correspondence

The oracle answers the hardest question in stereo, which pixels match. For a left pixel with first hit $\mathbf{X}$, the right eye sees $\mathbf{X}$ along $\mathbf{d}_R = (\mathbf{X} - \mathbf{c}_R)/\|\mathbf{X} - \mathbf{c}_R\|$; the sensor model inverts that direction into a right pixel. The match is accepted only if the right eye's own Position at that pixel equals $\mathbf{X}$ within a tolerance. If it does not, $\mathbf{X}$ is hidden from the right eye by something nearer, and the left pixel is half-occluded. This is how f3d-vision's explorer gets "perfect" stereo, and how fov-3d-vision measures the error of a real matcher against truth.

6.3 Epipolar coordinates on the sphere

Within a look both eyes share one head, so every epipolar plane contains the baseline, that head's X axis. A head-frame direction $\mathbf{d}$ is therefore best described by two angles:

$$\vartheta = \arccos(d_x), \qquad \varphi = \operatorname{atan2}(d_y,\, -d_z). \tag{19}$$

ϑ is the angle from the baseline and φ picks the epipolar plane, a rotation about the baseline. Two rays that see the same point share φ, and their parallax is $\delta = \vartheta_R - \vartheta_L > 0$. Unlike image disparity, these coordinates are defined over the whole sphere, so they work for every sensor and every gaze.

6.4 Triangulation and inverse depth

In the triangle formed by the two eye centres and the point, the law of sines gives the distance from the left eye exactly:

$$r_L = b\,\frac{\sin\vartheta_R}{\sin(\vartheta_R - \vartheta_L)}, \qquad \mathbf{X} = \mathbf{c}_L + r_L\,\mathbf{d}_L. \tag{20}$$

For small parallax, $\sin\delta \approx \delta$ and $\vartheta_R \approx \vartheta_L = \vartheta$, so inverse depth is linear in parallax:

$$\rho = \frac{1}{r} \approx \frac{\delta}{b\,\sin\vartheta}, \qquad \sigma_\rho \approx \frac{\sigma_\delta}{b\,\sin\vartheta}, \qquad \sigma_r \approx r^2\,\sigma_\rho. \tag{21}$$

This is why the stereo field measures **inverse depth** with a variance: its error is the parallax error scaled by a constant. A worked example: with $b = 63\,\mathrm{mm}$, a point straight ahead ($\vartheta = 90°$) and a parallax error of one foveal sample, $\sigma_\delta = 0.05°$, the inverse-depth error is $0.014\,\mathrm{m}^{-1}$, which at 2.7 m is $\pm 10\,\mathrm{cm}$. That is the scale fov-3d-vision measured where the fovea had looked (0.015–$0.019\,\mathrm{m}^{-1}$, 10–14 cm at 2.7 m [8]). Depth precision falls with the square of distance, and nothing can be measured along the baseline, where $\sin\vartheta = 0$.

When the two rays come from a real matcher they rarely meet exactly. The framework then takes the midpoint of their closest approach: with $\mathbf{w} = \mathbf{c}_L - \mathbf{c}_R$ and $\kappa = \mathbf{d}_L \cdot \mathbf{d}_R$,

$$s = \frac{\kappa\,(\mathbf{d}_R \cdot \mathbf{w}) - \mathbf{d}_L \cdot \mathbf{w}}{1 - \kappa^2}, \quad t = \frac{\mathbf{d}_R \cdot \mathbf{w} - \kappa\,(\mathbf{d}_L \cdot \mathbf{w})}{1 - \kappa^2}, \quad \mathbf{X} = \tfrac{1}{2}\big[(\mathbf{c}_L + s\,\mathbf{d}_L) + (\mathbf{c}_R + t\,\mathbf{d}_R)\big], \tag{22}$$

and the gap between the two closest points is a useful check on the match. What happens to the points afterwards (fusion into a surfel map at 12 mm, the coverage sphere, the gaze policy) is the business of the code that consumes the record, not of the rendering framework.

6.5 Motion parallax

Two looks of one eye from different head positions form a stereo pair whose baseline is the displacement $\mathbf{t} = \mathbf{c}_2 - \mathbf{c}_1$ between their eye centres. The geometry above carries over with $\mathbf{t}$ in place of $b\,\hat{\mathbf{x}}$: every plane that contains $\mathbf{t}$ is an epipolar plane, angles are measured from the direction of travel $\hat{\mathbf{u}} = \mathbf{t}/\|\mathbf{t}\|$, $\vartheta_k = \arccos(\mathbf{d}_k \cdot \hat{\mathbf{u}})$, and the law of sines gives the distance from the first position,

$$r_1 = \|\mathbf{t}\| \, \frac{\sin \vartheta_2}{\sin(\vartheta_2 - \vartheta_1)}. \tag{23}$$

For an observer moving forward every static point falls back, $\vartheta_2 > \vartheta_1$, which is the parallax. The two directions must be compared in one frame. Records hold world directions, so a turn of the head between the looks is already compensated, and only the translation makes parallax. Eq. (21) holds with $b \sin \vartheta$ replaced by $\|\mathbf{t}\| \sin \vartheta$: depth from motion is undefined towards the direction of travel, the focus of expansion, and weakest near it. An observer that looks where it is going puts its fovea where motion parallax vanishes. Section 8.6 measures this on a rendered pair.

7 Sampling and noise

Every pixel the framework returns is a Monte Carlo estimate of the average radiance over that pixel's footprint. Its noise falls as one over the square root of the samples per pixel, and the settings are chosen so that this statement stays exactly true.

7.1 Monte Carlo error

Cycles estimates a pixel by averaging n independent light paths, $v = \frac{1}{n} \sum_j L_j$. The estimate is unbiased and its standard deviation falls as

$$\sigma_v(n) = \frac{\sigma_1}{\sqrt{n}}, \tag{24}$$

where σ_1 is the spread of a single path, which depends on the scene's lighting and materials. Sensing cost is therefore controlled by one number, the samples per pixel (spp): 64 for a fixation at `small`, 256 at `full`. Two Cycles features would break the $1/\sqrt{n}$ law and are always off for measurement: adaptive sampling, which stops some pixels early, and denoising, which mixes neighbouring pixels.

7.2 The box filter makes a pixel a footprint average

With a box pixel filter of width 1, every path for a pixel starts at a uniformly random point inside that pixel's square on the raster, and contributes only to that pixel. Through the

sensor's pixel-to-ray map the square becomes the pixel's footprint Ω_i on the sphere, so the pixel estimates

$$v_i \approx \frac{1}{\omega_i} \int_{\Omega_i} L(\mathbf{d})\, d\Omega. \tag{25}$$

The approximation is only that sampling is uniform over the raster square rather than over solid angle, which differs little within one pixel. This is why each record entry carries its footprint ω_i: a foveated sample covering $151\, s_0^2$ in the periphery is compared with a reference by averaging the reference over the same patch of sphere. Wider filters, such as Blackman–Harris, spread a path over neighbouring pixels and break this identity; they are used only for previews.

7.3 References must be quieter than what they judge

A reference that is as noisy as the fixation it judges would hide the fixation's error in its own. fov-3d-vision's decision D7 [6] requires the reference's relative noise to be at most a third of the fixation's. With equal per-path spread that means

$$\sqrt{\frac{n_{\text{fix}}}{n_{\text{ref}}}} \le \frac{1}{3} \quad \Longleftrightarrow \quad n_{\text{ref}} \ge 9\, n_{\text{fix}}. \tag{26}$$

The profiles exceed it: 1024 against 64 spp at `small` (16×), 8192 against 256 at `full` (32×). The price is time. A `full` reference panorama took 32–36 minutes on an RTX 4090, against about 270 ms for a foveated pair, so references are rendered once per scene, pinned by checksum and kept.

7.4 Seeds

The random sequence of a render is set by its seed. The two eyes always use different seeds (2111 and 2112 by default) so their noise is independent; identical noise in both eyes would look like a perfect match and bias the stereo. Blender files can animate the seed (the Classroom does), so the framework removes such animation before rendering and checks after each render that the seed it set is the one used.

Two renders of the same look with different seeds measure the noise directly. Their difference has variance $2\sigma^2$, so

$$\hat{\sigma}^2 = \tfrac{1}{2}\, \mathrm{Var}\Big(v^{(s)} - v^{(s')}\Big). \tag{27}$$

This noise floor sets the thresholds of the framework's checks, rather than a fixed constant (decision D10 [6]). One limit applies: on OptiX the same seed does not give bit-identical images, so two conditions that must see identical noise are run on one saved observation rather than two renders.

8 Using the framework

A program opens a `Session` on a scene, chooses a sensor and a gaze, and asks for a `look`; it gets back one `SampleRecord` per eye. Everything else in this section is a variation on those

three calls, from Python (Sections 8.1–8.5) or from the shell (Sections 8.7–8.8). Every listing here runs against the repository.

8.1 A first look

```
from fov_render import Session, RenderConfig, Rig, Gaze, sensors

cfg = RenderConfig(profile="small", device="OPTIX")      # build on small, report on full
rig = Rig(ipd_m=0.063, roll="upright")

with Session("scenes/classroom/classroom_eye.blend", cfg, rig=rig) as s:
    eye = sensors.Foveated(E2_deg=2.0, e_max_deg=45.0)      # s0 comes from the profile
    look = s.look(eye, eyes=("L", "R"), gaze=Gaze.verged_point([0.4, 2.1, 1.1]),
                  outputs=("exr", "records"))
    L, R = look.records
    print(L.n_samples, L.direction.shape, L.footprint_sr.sum())
```

`Session` starts Blender in the background, opens the scene once and keeps it open; each `look` is one request to it, and the context manager stops it. Which Blender: `$FOVRENDER_BLENDER`, else `blender` on the `PATH`, else the current Python if it has the `bpy` wheel (`launcher=` chooses explicitly). Inside a script that already runs in Blender, `Session.in_process(cfg, rig=rig)` gives the same object without a second process. A look writes its EXR files and `meta.json`; the records are written when `outputs` asks for them. With the calibration room on a CPU, the listing prints 12,492 samples per eye at the `small` profile (Chat sandbox, issue #18).

8.2 The main objects

Each sensor's `.model` answers the geometric questions of Sections 4–5 in plain numpy, so analysis code can use it on any machine:

```
m = sensors.Foveated(E2_deg=2.0, e_max_deg=45.0).model(profile="full")
m.shape                          # (253, 253)
m.rays_per_eye                   # 50269
r = m.rays()                     # r.direction (camera frame), r.footprint_sr, r.valid (the disc)
rc, inside = m.pixel_of(r.direction[r.valid])   # inverse map: (row, col) and an inside mask
```

8.3 The record and the truth boundary

A `SampleRecord` holds one entry per valid sample. Its observation fields are `origin` (the eye centre), `direction`, `value` (linear RGB radiance), `footprint_sr`, `raster_index`, and the labels `eye`, `look_id` and `sensor`. Its oracle fields are `position_w`, `distance_m` (infinite on a miss) and `instance_id` (0 on a miss). Helpers convert between views: `rec.as_raster("value")` rebuilds the image, and `rec.direction_head` and `rec.epipolar()` give head-frame directions and the angles (ϑ, φ) of Section 6.3. Every record also carries its look's head, `head_R_wh` and `head_origin_w` (Section 3.5).

Object	Made by	What it holds or does
`RenderConfig`	`RenderConfig(profile, device, device_policy, spp, seed, color, exr_codec, passes)`, or `preset(name, ...)` for a legacy path	the frozen Cycles settings of Section 7; `small` or `full` fixes spp and spacing; `device_policy="strict"` never falls back to the CPU
`Rig`	`Rig(ipd_m, roll)`	baseline and roll convention (Section 3); `roll` is `"upright"` or `"baseline_projected"`
`Gaze`	`Gaze.yaw_pitch(psi, theta, vergence_m=2.1)`, `Gaze.verged_point(P)`, `Gaze.per_eye(L=(psi, theta), R=(psi, theta))`	where the eyes point, in degrees and metres (Sections 3.2–3.3)
sensors	`sensors.Foveated(E2_deg, e_max_deg, s0_deg=None)`, `sensors.Perspective(core_fov_deg=12)`, `sensors.Panorama(preset="preview360")`, `sensors.ReferenceView(width, height, half_fov_x_deg)`	the camera model; `.model` works without Blender
`Session`	`Session(blend, cfg, rig, access="observation", head=None, instance_ids="complete", launcher="auto")`; `move_head(...)`, `reset_head()`	the scene, the render server and the truth boundary; `head=` overrides the session's head, `instance_ids` chooses how objects reached through collection instances are numbered; `move_head` sets the head of later looks, on the client side
`HeadPose`	`session.session_head`, `session.head`; `.moved(translate_h, yaw_deg, pitch_deg, roll_deg)`, `.translated(dt_w)`	a head pose ($R_{WH}, \mathbf{o}$) and the moves of Eq. (5)
`Look`	`session.look(sensor, eyes, gaze, spp=None, seeds=None, out_dir=None, outputs=("exr",), head=None)`	the request, the response, `meta`, `exr` and `.records`; `head=` for this look only
`SampleRecord`	inside a `Look`, or `SampleRecord.load(path)`	one eye's samples (Section 8.3)

Table 3: The objects of the Python API.

The session's `access` decides who may see what. With `access="observation"`, the default, oracle fields are not loaded and reading one raises `AccessError`, so a learning observer cannot cheat by accident. An experiment that declares oracle correspondence or segmentation opens the session with `access="oracle"`, and that choice is written into every look's `meta.json`. References are a third tier: a panorama look of Section 5.2 at foveal spacing, rendered once per scene and kept for evaluation. In the current version keeping them away from the observer is the experiment's job; a store that an observer session refuses to open is planned (Section 11).

8.4 Worked example: inverse depth from oracle stereo

This example implements Sections 6.3–6.4 on one perspective pair: each left sample's oracle point is seen from both eye centres, and the law of sines gives its distance back.

```
import numpy as np
from fov_render import geometry

with Session(blend, RenderConfig(profile="full"), Rig(roll="baseline_projected"),
             access="oracle") as s:
    look = s.look(sensors.Perspective(), eyes=("L", "R"),
                  gaze=Gaze.yaw_pitch(10, -5, vergence_m=2.1),
                  outputs=("exr", "records"))
    L, R = look.records
    hit = np.isfinite(L.distance_m)
    P = L.position_w[hit].astype(float)        # oracle: what each left ray hit
    # epipolar angles in the head frame (Sec. 6.3)
    thL, phL = geometry.epipolar((P - L.origin) @ L.head_R_wh)
    thR, phR = geometry.epipolar((P - R.origin) @ R.head_R_wh)
    assert np.allclose(phL, phR, atol=1e-6)    # one epipolar plane
    # the law of sines (Sec. 6.4)
    rL, rR = geometry.triangulate_sine(thL, thR, s.rig.baseline_m)
    err = rL - L.distance_m[hit]   # zero up to round-off: a check, not a result
```

On the calibration room at the `small` profile the largest `|err|` is 3×10^{-7} m, the float32 precision of the Position pass (Chat sandbox, CPU, issue #18). Visibility in the right eye (Section 6.2) is a further test on `R`: project `P` with the right sensor's `model.pixel_of` and compare `R.distance_m` there. Replacing the oracle point with a real matcher's right-eye ray, and keeping `L.distance_m` only for scoring, turns the same lines into a stereo evaluation.

8.5 Worked example: an active loop

The framework renders; the loop's memory and policy are the student's. A minimal loop:

```
with Session(blend, RenderConfig(profile="small"), Rig(), access="oracle") as s:
    eye = sensors.Foveated(E2_deg=2.0, e_max_deg=45.0)
    gaze, memory = Gaze.yaw_pitch(0, 0, vergence_m=2.0), MyMemory()
    for k in range(50):
        look = s.look(eye, eyes=("L", "R"), gaze=gaze, outputs=("exr", "records"))
        memory.update(measure(look))     # fixate -> reconstruct -> update memory
        gaze = my_policy(memory)         # saccade
        if gaze is None:                 # nothing left worth looking at
            break
```

Because every sensor returns the same record, changing `eye` to `sensors.Perspective()` is the only edit needed to compare the two sensors in the same loop. Compare them at equal cost: `eye.model(profile).rays_per_eye` gives 50,269 rays for the foveated sensor and 409,600 for the perspective one at `full`, so equal rays means about eight foveated looks per perspective look.

8.6 Worked example: depth from motion

This example implements Section 6.5 with the cyclopean foveated sensor of Section 3.5, fixed to the head, which steps 10 cm right and 20 cm forward between two looks.

```
with Session(blend, RenderConfig(profile="small"), Rig(roll="upright"),
             access="oracle") as s:
    eye, recs = sensors.Foveated(E2_deg=2.0, e_max_deg=45.0), []
    for k in range(2):
        look = s.look(eye, eyes=("C",), gaze=Gaze.yaw_pitch(0, 0),   # fixed to the head
                      outputs=("exr", "records"))
        recs.append(look.records[0])
        s.move_head(translate_h=(0.10, 0.0, -0.20))   # 10 cm right, 20 cm forward
A, B = recs
t = B.head_origin_w - A.head_origin_w     # the motion baseline, world
u = t / np.linalg.norm(t)                  # the direction of travel
hit = np.isfinite(A.distance_m)
P = A.position_w[hit].astype(float)        # oracle: what each ray of A hit
unit = lambda v: v / np.linalg.norm(v, axis=1, keepdims=True)
thA = np.arccos(unit(P - A.origin) @ u)    # angles from the direction of travel
thB = np.arccos(unit(P - B.origin) @ u)
assert (thB > thA).all()                   # moving forward, every point falls back
rA, _ = geometry.triangulate_sine(thA, thB, np.linalg.norm(t))
err = rA - A.distance_m[hit]               # zero up to round-off: a check, not a result
```

On the calibration room at the `small` profile all 12,492 samples hit, and the largest `|err|` is 2.4×10^{-11} m (Chat sandbox, CPU). The oracle gives both directions here. Replacing the first one with the record's own `A.direction[hit]`, the ray through the pixel centre while Position comes from a point inside the pixel, turns the check into a measurement of what the sensor's spacing costs. The median error is then 0.6 mm, the 99th percentile 1.0 cm and the largest 4.6 cm. The median relative error is 0.19% within 5° of the direction of travel and 0.01% between 15° and 45° from it, as Eq. (21) with $\|\mathbf{t}\| \sin\vartheta$ predicts. A real matcher's ray in `B`, in place of the oracle's, makes the same lines an evaluation of depth from motion.

8.7 Requests and the command line

Every `look` is serialised as a JSON request (schema `fov-render-request-v1`) holding the sensor and its parameters, the eyes, the gaze, an optional rig override, an optional head, spp, seeds, the output folder and what to write. The session itself is one more request, `fov-render-session-v1`: the scene, the `RenderConfig`, the rig, the access level and the Object Index numbering. Both are validated before anything renders, and a malformed one is refused with every reason at once. Each look's folder holds the request, a `meta.json` with the poses, the settings Cycles actually used and the timings, the raw EXR per eye and, when asked, the records. Between a `Session` and its server the requests travel through a queue folder (`req-NNNN.json` in, `done-NNNN.json` or `fail-NNNN.json` out), the protocol of f3d-vision's render server.

The same requests can be issued from a shell with `fov-render`:

```
fov-render look --blend scene_eye.blend --sensor foveated --eyes L R \
    --point 0.4 2.1 1.1 --profile full --records --out runs/demo/f0000
```

```
fov-render look --blend scene_eye.blend --sensor perspective --gaze 10 -5 --vergence 2.1 \
    --seeds L=2111 R=2112 --out runs/demo/p0000
fov-render look --blend scene_eye.blend --sensor foveated --eyes C --gaze 0 0 \
    --head-move 0 0 -0.5 --head-turn 20 0 0 --out runs/demo/moved
fov-render panorama --blend scene_eye.blend --preset preview360 --eye C \
    --profile small --out previews/ref_C
fov-render requests --blend scene_eye.blend looks.json --out runs/demo
fov-render info --blend scene_eye.blend
```

Sensor fields are passed as `--param KEY=VALUE` (`E2_deg=2.0 e_max_deg=45`); the gaze is one of `--point`, `--gaze YAW PITCH` with an optional `--vergence`, `--per-eye L=YAW,PITCH R=...`, or none, for a fixed look along the head. `--head-move DX DY DZ` (metres) and `--head-turn YAW PITCH ROLL` (degrees) move the session's head for one look, by Eq. (5). A `requests` file is a JSON list of looks, each with the request's fields, an absolute `head` among them; the scene is opened once for all of them. `info` prints the scene report as JSON: the file and its checksum, the head, the view layer that renders, the numbered objects and the material and image audits. A command-line look sends exactly the request `Session.look` sends, and the package's checks verify that the two give bit-identical images and records on the CPU (Section 9). Every folder also gets `fov-render-run.json`: the command, the session, the server's ready answer and every response.

The exit status follows one rule: **0** done; **1** a render, a check or Blender failed; **2** the arguments, a request or a scene specification were refused, every reason printed, and Blender not started. Blender's own `blender -b -P` exits 0 even when its script fails, so every Blender-side entry point of the package exits nonzero itself.

8.8 Bringing in a new scene

Any `.blend` file can be used once it has a head. The framework never edits the original: `fov-render eye` writes a copy with a camera named `EYE` at the head, and a JSON sidecar recording the source's checksum and the pose. The copy is saved next to the original so relative texture paths keep working. Then `info` shows what the framework sees in the file, a first scene gate drafts the scene's specification, and, once the draft has been read, the full gate runs against it:

```
fov-render eye --blend shop.blend --out shop_eye.blend --from-camera --forward-m 1.5
fov-render info --blend shop_eye.blend
fov-render check --blend shop_eye.blend --only S1 S2 --write-spec shop.draft.json
fov-render check --blend shop_eye.blend --spec shop.json --device OPTIX
```

`--from-camera` places the head a given distance along the file's own camera, level, with its yaw: demonstration files are usually framed for one shot camera that stands at the edge of the set. In the Barbershop (Blender Studio, CC-BY) that camera stands 0.53 m behind a wall (issue #4); in the Classroom f3d-vision measured a back-face fraction of 0.136 from the file's camera and 0.0038 from a head inside the room [9].

`fov-render check` runs the *scene gate*, the checks that any file can run (Table 4). What a scene is expected to give is its *specification*, a small JSON file: the hole fraction allowed, the objects allowed to show back-faces, the nearest surface, the images known to be missing, the

gazes to test. `--write-spec` drafts one from what S1 and S2 measure: the missing images as found, the closed meshes that show back-faces as allowed, a hole limit of twice the measured fraction, at least 0.001 and at most the default 0.02. A draft accepts the scene as it is, so it is read before it is kept; what it records is what a later change to the scene must keep. A specification with an unknown or empty expectation is refused. The repository keeps specifications for the calibration room, the Barbershop and the Classroom in `docs/scenes/`.

	What is checked	Control that must fail
S1	the scene opens: the head from `EYE`, the view layer that renders, every renderable object numbered, the missing images as expected	an expectation one image off
S2	a low-resolution panorama from the head: holes, back-faces only on allowed objects (closed meshes count, open ones are reported), the nearest surface	every back-face counted, none allowed (when the scene has any)
S3	fov-3d-vision's `check_foveated` on foveated fixations at the specification's gazes	a shader E_2 of 2.04 against a record that says 2.0
S4	every sensor's record agrees with its Position pass within a pixel	a record built with the other eye's pose
S5	Object Index equals the ray-cast object, through flagged non-opaque objects	an object renumbered after the render
S6	the material and image audits, by solid angle from the head	reported for review, not gated

Table 4: The scene gate, `fov-render check`. On the calibration room it takes 65 s on a CPU, 53 s of which is S4 (Chat sandbox, issue #18).

9 Validation

Every layer ships a check with a known answer and a control: a deliberately broken input that the check must reject, because a check that cannot fail tests nothing. A check whose control passes is reported as failed. The package's checks run with `fov-render selftest` (or `python -m fov_render.checks`); those that need Blender are reported as skipped, never as passed, where Blender is missing. Table 5 lists them by the migration step that introduced them (Section 10).

The comparisons with the code fov-render replaced use *known answers*: what the legacy code gave, frozen in `fov_render/checks/known/` by a tool that re-runs the legacy code. The legacy files themselves are not copied into the repository; they are pinned by repository, commit, path and checksum, and rebuilt from a checkout's history when a check or a re-freeze needs them. A frozen answer comes from one machine, so renders are compared with tolerances and numpy results within 10^{-12}; the tolerances are a pixel or a pose precision, never a constant moved to make a check pass. Each check was also run against mutations of the code (a warp E_2 off by 10^{-9}, a pitch off by 10^{-9} degrees, a filter width of 1.01, the catalogue's order, the head moved 1 mm, the seeds ignored by the command line) and failed on each.

At the time of writing, 35 checks pass in a CPU sandbox with the `bpy` wheel in about a

Step	Known answer	Control that must fail
0 files	the vendored files (shader, EXR reader, test scenes, `check_foveated`) are byte-identical to their sources	a one-byte change
1 foveated	fov-3d-vision's `check_foveated` passes on a fov-render fixation; the fixation equals the frozen legacy fixation within pose precision; the record agrees with Position and Depth	a record built with a mirrored yaw
2 numpy	the warp equals the legacy warp (directions, footprints, mask, inverse); the three historical pose formulas agree; the rig reproduces f3d-vision's calibrations, both rolls; footprints sum to the field's solid angle, $2\pi(1-\cos e_{\max})$ or 4π; `pixel_of` inverts every sensor's rays; triangulation	a yaw sign flip; upright roll at an extreme gaze; the eyes swapped
3 config	the presets reproduce seven legacy render paths (13 comparisons); a change of spp takes effect	`cycles.samples` animated in the file
4 scene	the head from `EYE`; the instance catalogue, both numberings; the calibration room's holes, back-faces, nadir and zenith; a world-only HDRI scene reproduces its source; the material audit; Object Index against a ray cast; the scene-layer fixes found on new scenes (4.7–4.11)	the ceiling removed; an object renumbered after rendering; the old behaviour of each fix
5 requests	requests round-trip through JSON bit for bit; every malformed one is refused with its reason; a refused request leaves the server serving, a failed render stops it with exit 1	a valid request is answered
cli	malformed command lines exit 2 before Blender starts; a command-line look equals `Session.look` bit for bit; an unreadable scene exits 1; a drafted specification passes its scene; `eye` places the head	seeds swapped, the gaze point moved 1 mm, a broken room
head	requests with a head round-trip and malformed heads are refused, while a request without one is byte for byte what it was before; `moved` as defined; `move_head` fills in exactly the requests it should; the session's head given as a look's renders bit for bit as no head (on the CPU); from five head poses every sensor's record sits on the requested head	a yaw sign flip; the head moved 1 mm; a handler that ignores the request's head
Classroom	on the workstation, against f3d-vision's pinned files: `check_foveated`, the AB1a OptiX settings, the head pose, the instance catalogue of breadth-1	shader E_2 2.04, a CPU device, the head moved 1 mm, an object hidden

Table 5: The package's checks and their controls.

minute and a half, twelve of them without Blender (Chat sandbox, issue #20), and the same 35 pass on OptiX in 40 s (RTX 4090 workstation, issue #20); the four Classroom checks need the Classroom files and pass on the workstation (issues #1 and #16).

10 How the framework was built

fov-render was not written from scratch: it is the rendering code of fov-3d-vision and f3d-vision, migrated into one package in seven steps, each closed only when its checks passed

against the code it replaced (`docs/HISTORY.md` [2]).

1. The foveated camera: fov-3d-vision's OSL shader behind the new builder, checked by its own `check_foveated` and against its fixations.
2. The numpy models of the four sensors, the rig and the record, checked against both code bases' formulas.
3. `RenderConfig` and the presets that reproduce each legacy render path.
4. The scene layer: opening any file, the head, the instance catalogue, the audits and the sanity checks.
5. `Session`, the render server and the JSON requests. f3d-vision's greedy explorer ran unchanged on a `Session` and reproduced its frozen Playground-2 run: 356 fixations, 98.82% of the sphere seen, every equivalence check passed (issue #2).
6. fov-3d-vision's foveated active loop on a `Session`: over 50 fixations on the CPU it equalled the legacy loop bit for bit in every file, while a control that moved the IPD by one unit in the last place differed (issue #3).
7. A scene neither code base had used: the Barbershop, through the scene gate. The triage of a second scene, the Barcelona Pavilion, found four bugs in the scene layer, fixed with checks 4.7–4.11 (issue #4).

The tag `v0.1.0` is the last commit that carried the legacy copies and the adapters that ran the two code bases on fov-render; after it, the comparisons became the known answers and pinned files of Section 9 (issue #16). One lesson from step 6 is worth keeping. A statistical check that compared the spread of two runs of the active loop was withdrawn: the loop is chaotic, so even the unchanged legacy loop failed it when its IPD moved by 0.2 µm. Exact equality, where it can be had, is the stronger test; where it cannot, a check must be shown to pass on the legacy code before it judges the new one.

Two capabilities came after the tag, each through one issue with its checks: the command line (issue #18) and head motion (issue #20). Head motion added a second lesson. Its bit-for-bit check failed on the GPU, though not because of the head: two identical looks on OptiX already differ in about 13% of their radiance values, by one or two float32 units in the last place, on the code before the change as well (RTX 4090 workstation). A bit-for-bit check therefore runs on the CPU whatever device the checks are given.

The work was shared between three parties through the repository's GitHub issues: Luiz Velho decided and accepted; Claude in a chat sandbox, with the `bpy` wheel on a CPU, designed, wrote the code and its checks; and Claude Code on an RTX 4090 workstation ran what needed the GPU or the Classroom files and reported. Every number in the issues says where it was measured. The method is the N-Surface method of f3d-vision [9], lightened to one file of rules (`CLAUDE.md`); the issues are the project's record.

11 Limitations and further reading

The framework simulates a controlled observer, not a real one; its limits are the price of an exact oracle.

- **Head motion between looks only.** The head moves from one look to the next, not during a look: no motion blur and no rolling shutter. Head turns pivot on the cyclopean point unless the caller gives a pivot (Eq. (6)), and there is no body or neck model. The scene gate's survey of holes and back-faces (S2) runs from the session's head only, so a moved head can see holes it does not report: in the Classroom, a head moved 0.5 m forward and turned 20° right found no surface for about 3% of its foveated samples, all towards a window (RTX 4090 workstation, issue #20).
- **First-hit truth.** Position and Object Index describe the first surface a ray meets. For glass, cut-out alpha, volumes or emitters this differs from what the pixel shows; such objects are flagged when a scene is opened, and each experiment decides whether to exclude them.
- **Static, synthetic scenes.** No moving objects, no sensor model beyond path-traced radiance at a chosen spp, no real camera.
- **No physiological torsion.** Roll follows one of two engineering conventions (Section 3.4), not the torsion of a real eye.
- **Back-ends and repeatability.** The OSL camera is verified on OptiX and on CPU; other GPU back-ends are untested. OptiX renders are not bit-reproducible: Position and Object Index repeat exactly, radiance does not, so exact pairing of conditions reuses saved observations. CPU renders repeat bit for bit on the calibration room but not on the Classroom (issue #9).
- **Object identity.** Instances of one object share its number in the Object Index pass, and objects made by geometry-nodes instancers are listed but not numbered.
- **Reference cost.** Uniform references at foveal spacing take about half an hour per scene at the `full` profile, so they are rendered once and kept.
- **Not built yet.** Two conveniences of the original design are not in the package: an oracle matcher on a look (pairing the samples both eyes see), and an evaluation-only store for references that an observer session refuses to open. Section 8.3 shows both done by hand.

Further reading. The code, its user's guide and its record are in the repository [1]: `README.md` for use, `docs/PLAN.md` for the decisions taken in building it, `docs/HISTORY.md` for the migration [2], and the GitHub issues for every measurement. The decisions behind the numbers in this report, each with what would overturn it, are in fov-3d-vision's `DECISIONS.md` [6]. The warp's derivation and its first measurement are in step A3 [7], and the active loop's results in the Phase C summary [8]. The perspective explorer's architecture is documented in f3d-vision [10]. The sequence of reports from the laboratory is summarised in TR-09-2026 [3] and TR-11-2026 [4].

References

[1] VISGRAF Lab. *fov-render*: Blender Cycles rendering for gaze-directed binocular observers. https://github.com/visgraf/fov-render

[2] VISGRAF Lab. fov-render, `docs/HISTORY.md`: the migration. https://github.com/visgraf/fov-render/blob/main/docs/HISTORY.md

[3] Luiz Velho and Claude. "Experiments on Active 3D Vision". Technical Report TR-09-2026, VISGRAF Lab – IMPA, 2026.

[4] Luiz Velho and Claude. "Active Foveal 3-D Vision Playground". Technical Report TR-11-2026, VISGRAF Lab – IMPA, 2026.

[5] VISGRAF Lab. *fov-3d-vision*: foveated omnidirectional rendering and 3D reconstruction. https://github.com/visgraf/fov-3d-vision

[6] VISGRAF Lab. fov-3d-vision, `DECISIONS.md`. https://github.com/visgraf/fov-3d-vision/blob/main/DECISIONS.md

[7] VISGRAF Lab. fov-3d-vision, step A3: the foveated camera. https://github.com/visgraf/fov-3d-vision/blob/main/docs/a3-foveated-camera.md

[8] VISGRAF Lab. fov-3d-vision, Phase C summary. https://github.com/visgraf/fov-3d-vision/blob/main/docs/phase-c-summary.md

[9] VISGRAF Lab. *f3d-vision*: the Active Foveal Playground. https://github.com/visgraf/f3d-vision

[10] VISGRAF Lab. f3d-vision, Greedy Foveal Playground architecture. https://github.com/visgraf/f3d-vision/blob/main/docs/architecture/greedy-foveal-playground.md