MatterLab

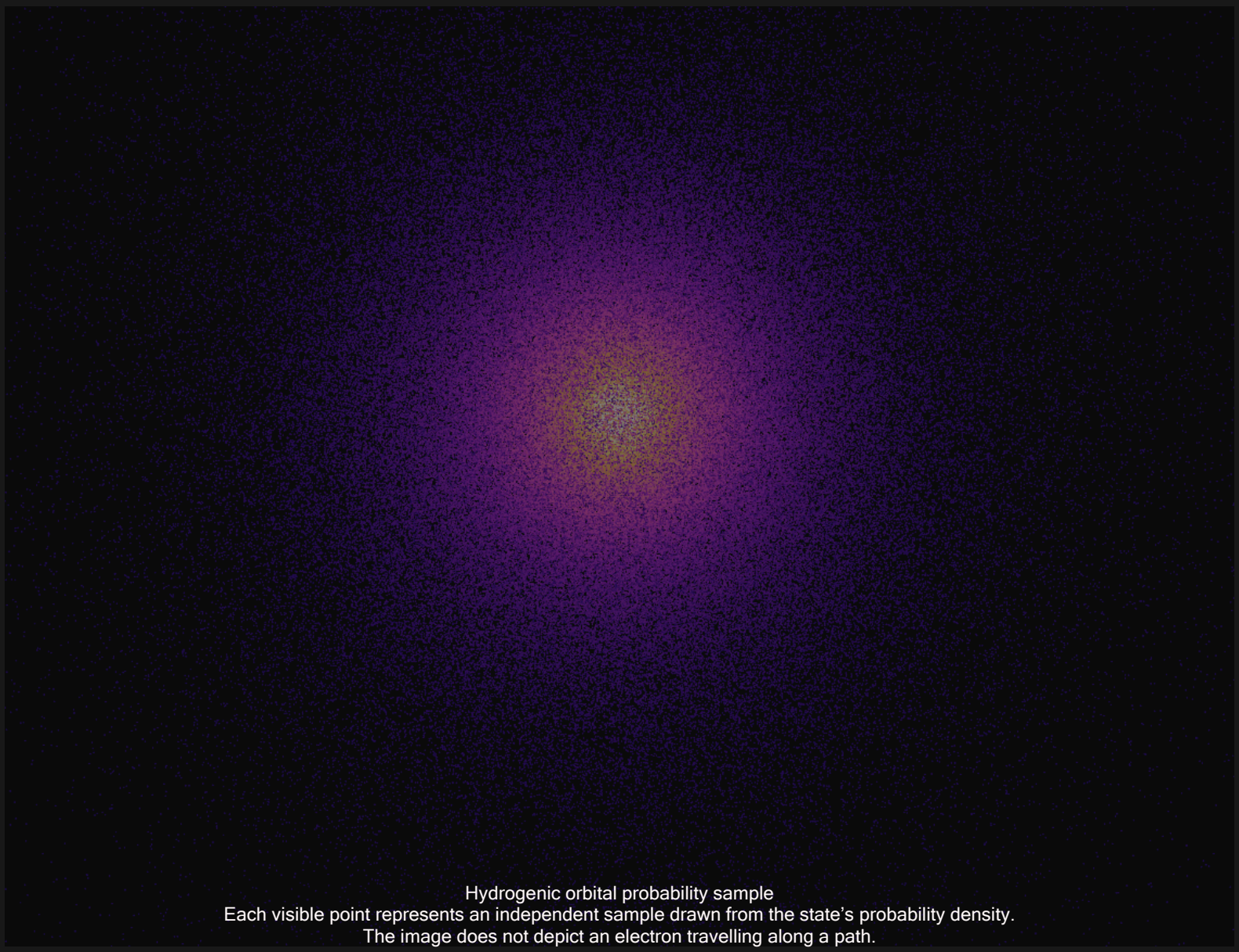
Quantum Orbital Laboratory

A PhantomLabs Technology
research project

Defining the Problem

How do you make a quantum orbital visible without turning it into a misleading picture? Atomic orbitals are not miniature planetary paths, solid clouds, or objects that can be photographed directly. They are mathematical quantum states whose wavefunctions assign a complex probability amplitude to every point in space.

MatterLab began with a visualization problem: convert that abstract mathematical description into an explorable three-dimensional form while preserving the distinction between the underlying science and the artistic decisions required to display it.



Hydrogenic orbital probability sample
Each visible point represents an independent sample drawn from the state's probability density.
The image does not depict an electron travelling along a path.

POSITION
Sampled from $|\psi|^2$

CONCENTRATION
Represents relative probability density

FORM
Determined by n, l, m and Z

The translation problem

From wavefunction to probability

MatterLab starts with the stationary hydrogenic wavefunction, separated into a radial function and an angular function. The wavefunction itself is generally complex-valued and cannot be displayed directly as a conventional physical object.

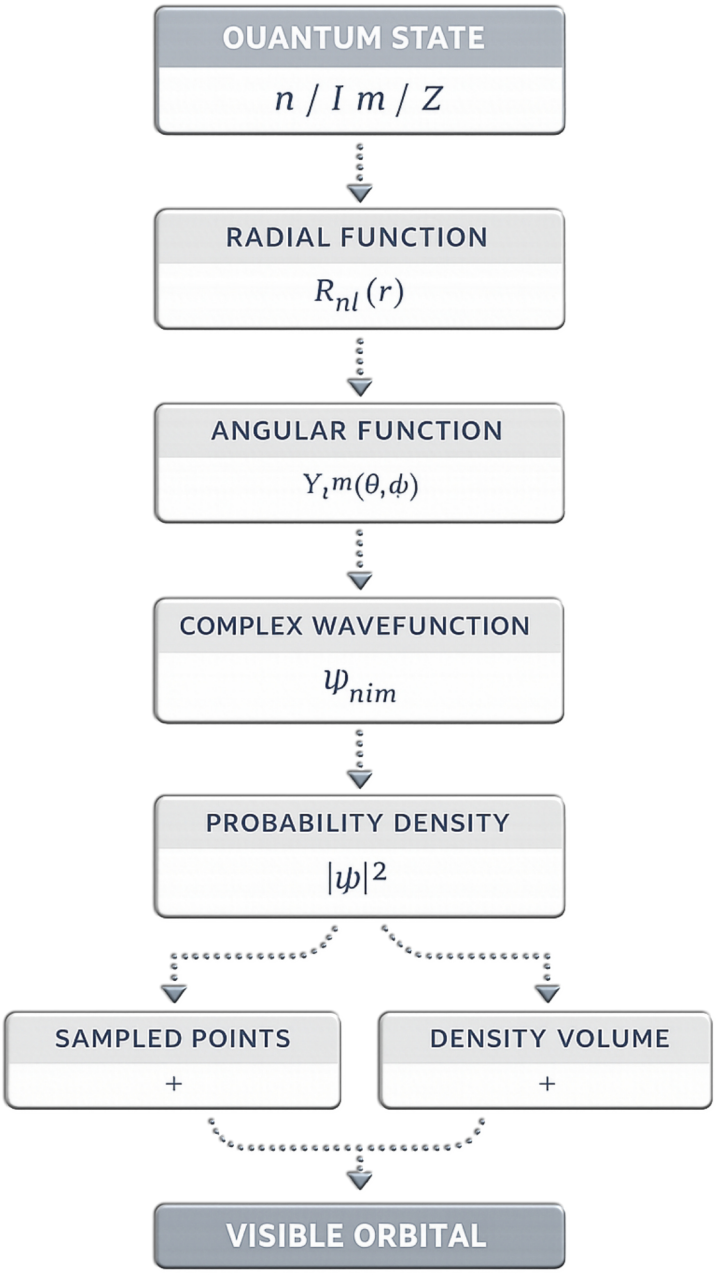
The quantity used to construct the visualization is its squared magnitude, $|\psi|^2$

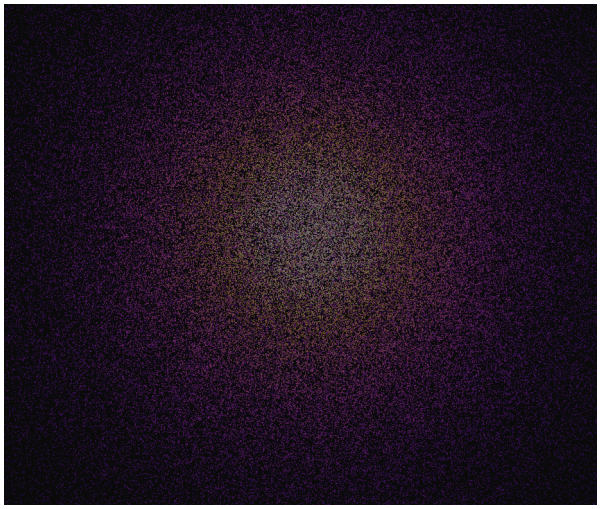
This produces a real, non-negative probability density describing where a position measurement is more or less likely to find the electron.

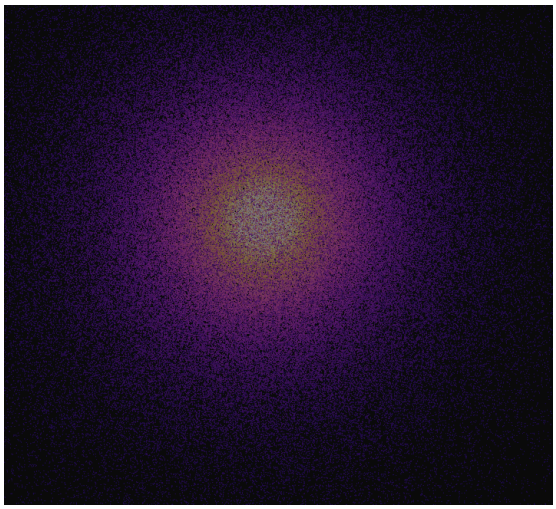
IMPLEMENTED IN
HydrogenWavefunction.cpp
ProbabilityDensity.cpp

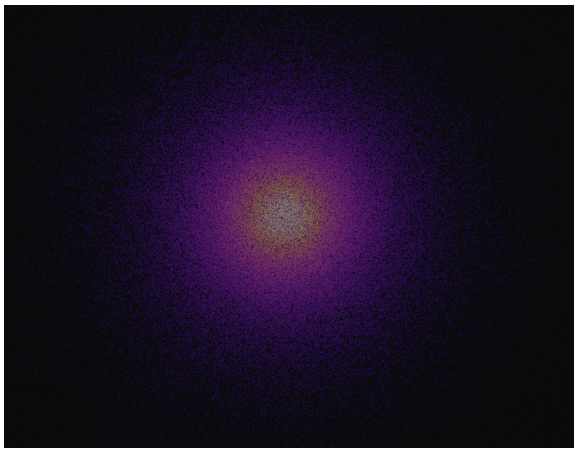
(C++)

$$\psi_{nlm}(r,\theta,\phi)=R_{nl}(r)Y_{lm}(\theta,\phi)$$
$$\rho(r,\theta,\phi)=|\psi_{nlm}(r,\theta,\phi)|^2$$


OUANTUM STATE
$n\ /\ I\ m\ /\ Z$
RADIAL FUNCTION
$R_{nl}(r)$
ANGULAR FUNCTION
$Y_{\iota}{}^{m}(\theta,\phi)$
COMPLEX WAVEFUNCTION
ψ_{nim}
PROBABILITY DENSITY
$|\psi|^2$
SAMPLED POINTS
+
DENSITY VOLUME
+
VISIBLE ORBITAL





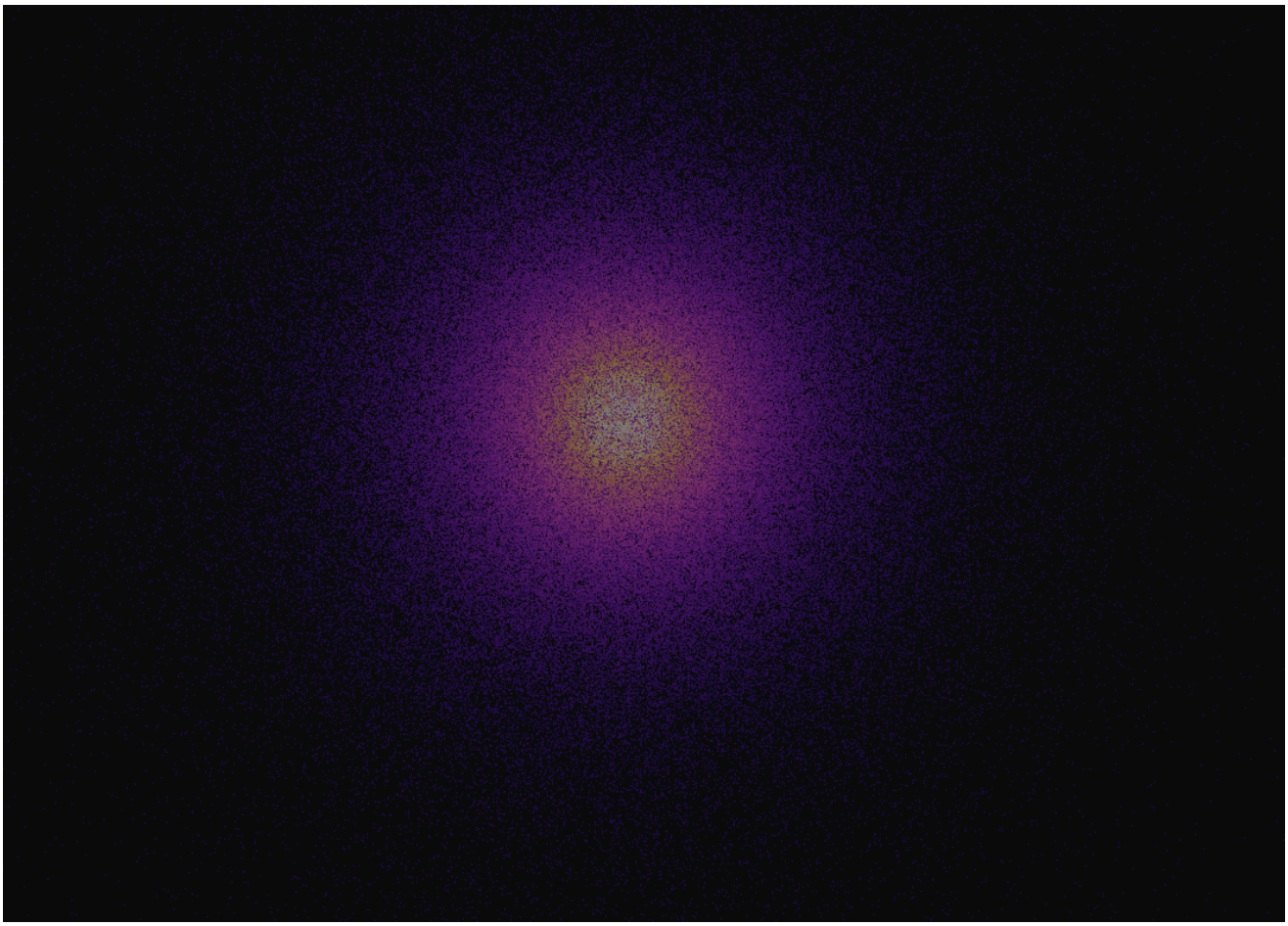


Scientific Boundary

MatterLab is intentionally limited to stationary hydrogenic systems: atoms or ions containing a single electron moving in the Coulomb field of a nucleus. This includes hydrogen and hydrogen-like ions such as He^{+} and Li^{2+}. The restriction is important because the hydrogenic Schrödinger equation has analytical solutions that can be evaluated directly, normalized, sampled, and compared against known results.

The present model is nonrelativistic and uses a point nucleus. It does not include electron spin, many-electron interactions, molecular bonding, fine structure, hyperfine structure, finite nuclear size, or quantum-electrodynamic corrections. The interface permits nuclear charge values up to $Z = 118$ for exploration, but the software warns that omitted relativistic and nuclear effects become increasingly important as Z rises.

This boundary allowed MatterLab to prioritize mathematical traceability. Every visible orbital begins with an implemented analytical state rather than a handcrafted mesh or an illustrative image.



Design Requirements

Five requirements shaped the project.

First, the visual result had to originate from the scientific model. Orbital geometry could not be drawn manually and then labeled as physical output.

Second, the software had to preserve the probabilistic meaning of the wavefunction. A rendered point represents a sampled position, not an electron travelling along a path.

Third, changes to the quantum state had to generate a new result rather than switch between prerecorded images.

Fourth, the calculation and sampling pipeline had to remain responsive while continuously producing large numbers of points.

Fifth, the final image had to remain legible. Nodes, lobes, density differences, and orientation had to be visible without presenting color, opacity, point size, or threshold values as fundamental laws of nature.

These requirements separated the project into three layers: the scientific model, the numerical method used to evaluate and sample it, and the visual treatment used to display the result.

Part II – Building the Scientific Engine

Atomic Units

MatterLab performs its scientific calculations in atomic units. In this system, the electron mass, elementary charge, reduced Planck constant, and Coulomb constant are normalized to one. Distances are expressed in Bohr radii, energies in Hartrees, and many equations become simpler because repeated physical constants disappear from the working form.

Atomic units are useful computationally, but they can obscure physical scale if they are not identified clearly. MatterLab therefore retains conversion constants for presenting quantities in more familiar units, including electronvolts for energy. One Hartree is approximately 27.211386 electronvolts, while one Bohr radius is approximately 5.29177×10^{-11} meters.

Using a single internal unit system also reduces dimensional inconsistency. Radial boundaries, expected radii, probability distributions, sample positions, camera framing, and plotted values are all derived from the same scale before any visual transformation is applied.

Primary implementation: PhysicalConstants.hpp

Quantum State: n, l, m, and Z

A MatterLab state is defined by four values: the principal quantum number n, orbital angular-momentum quantum number l, magnetic quantum number m, and nuclear charge Z.

The principal quantum number controls the energy level and general radial scale. The orbital quantum number controls angular structure and the number of angular nodes. The magnetic quantum number selects an orientation-dependent component of that angular structure. Nuclear charge changes both the binding energy and the spatial scale of the state.

Valid states satisfy:

$$n \geq 1$$

$$0 \leq l < n$$

$$-l \leq m \leq l$$

$$Z > 0$$

The application interface limits n to values from 1 through 8 and Z to values from 1 through 118. These are product limits rather than changes to the underlying mathematical definitions. State validation occurs before sampling so that impossible combinations cannot enter the scientific pipeline.

Energy, Nodes, and Expected Radius

Within the nonrelativistic hydrogenic model, the energy depends only on n and Z:

$$E_n = -Z^2 / (2n^2) \text{ Hartree}$$

Increasing n moves the state closer to zero energy, while increasing Z strengthens the Coulomb attraction and produces more negative energies. MatterLab converts Hartrees to electronvolts for display while preserving Hartrees internally.

The state also predicts its node structure. The number of radial nodes is n − l − 1. The number of angular nodes is l, and the total number of nodes is n − 1. Nodes are regions where the wavefunction becomes zero and therefore where the probability density vanishes.

The analytical expected radius is:

$$\langle r \rangle = [3n^2 - l(l + 1)] / (2Z)\, a_0$$

This value provides a useful validation target. A sufficiently large set of correctly sampled points should produce an average radius that approaches the analytical expectation.

Primary implementation: QuantumState.cpp, QuantumStateController.cpp, RadialDistribution.*

Dimensionless Radius and Radial Structure

The radial solution is expressed using the dimensionless coordinate:

$$\rho = 2Zr / n$$

Here, r is measured in Bohr radii. The transformation absorbs the state's principal scale and nuclear charge into a dimensionless variable. This makes the analytical radial function suitable for a wide range of states without changing its fundamental form.

The normalized hydrogenic radial wavefunction is:

$$R_{nl}(r) = N_{nl} \exp(-\rho/2) \rho^{l} L^{(2l+1)}_{n-l-1}(\rho)$$

The exponential term controls long-range decay. The factor ρ^{l} controls behavior near the nucleus, and the generalized associated Laguerre polynomial produces the state's radial nodes and oscillatory structure. The normalization constant N_{nl} is evaluated in logarithmic form with the gamma function to reduce overflow risk from large factorial expressions.

The radial function is an amplitude, not a probability by itself. The physically relevant radial distribution also includes the spherical volume factor r^2.

Primary implementation: HydrogenWavefunction.*

Associated Laguerre Polynomials

Generalized associated Laguerre polynomials provide the polynomial component of the hydrogenic radial function. MatterLab evaluates them with a three-term recurrence rather than storing a separate hard-coded expression for every orbital.

The first two terms are:

$$L^{(\alpha)}_0(x) = 1$$

$$L^{(\alpha)}_1(x) = 1 + \alpha - x$$

Higher orders are generated by:

$$kL^{(\alpha)}_k(x) = (2k - 1 + \alpha - x)L^{(\alpha)}_{k-1}(x) - (k - 1 + \alpha)L^{(\alpha)}_{k-2}(x)$$

For a hydrogenic state, the polynomial order is $n - l - 1$ and the parameter α is $2l + 1$. The order therefore matches the number of radial nodes. This connection is visible in the final orbital: higher-order radial states form separated regions of probability divided by spherical nodal surfaces.

Primary implementation: AssociatedLaguerre.*

Associated Legendre Functions

The angular solution begins with associated Legendre functions. These functions determine how amplitude changes with the polar angle θ. MatterLab includes the Condon–Shortley phase and evaluates the functions through recurrence relations.

The diagonal term is:

$$P^m_m(x) = (-1)^m(2m - 1)!!(1 - x^2)^{m/2}$$

The next term is:

$$P^m_{m+1}(x) = x(2m + 1)P^m_m(x)$$

Higher values of l are then generated recursively. In the orbital calculation, x is cos θ.

These functions are responsible for much of the lobe and node structure associated with p, d, and f orbitals. Their zero crossings become angular nodes after the functions are combined with azimuthal dependence in the spherical harmonics.

Primary implementation: AssociatedLegendre.*

Spherical Harmonics and Orbital Bases

The normalized complex spherical harmonics are:

$$Y^l_m(\theta, \varphi) = N_{lm}\, P^l_m(\cos\theta) e^{im\varphi}$$

They combine the polar behavior of the associated Legendre function with an azimuthal complex phase. The quantum number l controls total angular structure, while m controls the azimuthal component and orientation behavior.

MatterLab supports the complex eigenstate basis and two real tesseral visualization bases. Real orbitals are formed from scaled real or imaginary combinations of $Y^l_{|m|}$. These combinations produce familiar oriented forms such as p_x and p_y while preserving the same underlying angular subspace.

The choice of basis changes the visible orientation and sign structure, but it does not invent a new energy level. Complex and real combinations can represent different bases within the same degenerate hydrogenic state. MatterLab therefore treats basis selection as part of state representation rather than as a change to the Coulomb Hamiltonian.

Primary implementation: SphericalHarmonics.*, QuantumStateController.*

Constructing the Full Wavefunction

The stationary hydrogenic wavefunction separates into radial and angular components:

$$\psi_{nlm}(r, \theta, \varphi) = R_{nl}(r)Y^{l}_{m}(\theta, \varphi)$$

MatterLab evaluates this expression in spherical coordinates and also provides a Cartesian entry point. Cartesian positions are converted using:

$$r = \sqrt{(x^2 + y^2 + z^2)}$$

$$\theta = \arccos(z / r)$$

$$\varphi = \operatorname{atan2}(y, x)$$

The separated form is central to the project's architecture. Radial and angular distributions can be precomputed independently for sampling, while the authoritative full wavefunction can still be evaluated at every generated Cartesian point.

This second evaluation is important because the sampler determines where candidate points are placed, while the wavefunction evaluation determines the exact amplitude, probability density, phase, and sign data associated with each accepted point.

Primary implementation: HydrogenWavefunction.*

Probability Density, Phase, and Sign

The wavefunction is generally complex and is not itself a directly observable spatial density. MatterLab converts it into probability density using:

$|\psi|^2 = \psi^*\psi$

This quantity is real and non-negative. Regions with larger $|\psi|^2$ are more likely to produce sampled positions, while nodal regions remain empty because the density becomes zero.

MatterLab also calculates the complex phase:

$\text{phase} = \text{atan2}(\text{Im}\ \psi, \text{Re}\ \psi)$

For real visualization bases, it records the sign of the real amplitude. Density controls where points appear. Phase and sign contain additional information about the state, particularly for interference and basis interpretation.

The current point pipeline stores density, phase, and sign for every generated point. However, the active fragment shader primarily maps density to color. Phase and sign color modes exist as foundations but are not fully connected to the visible interface. The paper should therefore distinguish between calculated data and currently displayed data.

Primary implementation: ProbabilityDensity.*, QuantumPoint, point shaders

Part III – Turning Equations into Data

Radial Probability Distribution

The probability of finding the electron within a thin spherical shell depends on both the radial amplitude and the volume of that shell. MatterLab therefore uses the radial distribution:

$$p_r(r) = r^2|R_{nl}(r)|^2$$

The factor r^2 is essential. A state may have a large amplitude near the origin while the corresponding shell contains very little volume. The radial distribution combines these effects and provides the correct one-dimensional probability density for sampling r.

MatterLab uses this function to build radial plots, identify peaks, locate radial nodes, estimate a finite sampling boundary, and construct the cumulative distribution used by the sampler. The function also provides a direct way to compare numerical output with analytical expectations.

A radial node appears where $R_{nl}(r)$ crosses zero. A radial peak appears where the shell probability reaches a local maximum. These features explain the layered shells visible in states such as 2s and 3s.

Primary implementation: RadialDistribution.*

Numerical Integration and Normalization

Analytical wavefunctions are normalized over an infinite domain, but a computer must evaluate them over a finite interval. MatterLab uses recursive adaptive Simpson integration to estimate radial integrals while concentrating evaluations where the function changes most.

For an interval from a to b, Simpson's estimate is:

$$S(a, b) = (b - a)[f(a) + 4f((a + b)/2) + f(b)] / 6$$

The algorithm subdivides the interval, compares the combined smaller estimates with the larger estimate, and continues recursively until the difference falls within tolerance. It applies a $\Delta/15$ Richardson correction to improve the result.

MatterLab uses numerical integration to check radial normalization and to support state-dependent radial boundaries. The integration result structure contains a field for estimated error, although that field is not currently populated. The method is implemented, but its diagnostic reporting remains incomplete.

Primary implementation: NumericalIntegration.*, RadialDistribution.*

Finite Radial Boundaries, Nodes, and Peaks

Although the mathematical state extends to infinity, the probability density eventually becomes negligible. MatterLab estimates a finite radial boundary large enough to contain the useful distribution for the selected n, l, and Z. This boundary is used for plots, numerical integration, cumulative distributions, sampling, and camera framing.

The boundary cannot be fixed globally. Higher principal quantum numbers spread farther from the nucleus, while larger Z values contract the state. MatterLab therefore derives the working range from the active state and expands it as needed until the tail contribution becomes sufficiently small.

Within that range, the software searches for sign changes and local extrema to identify radial nodes and peaks. These numerical features are compared against the expected node count n − l − 1. This provides both a scientific diagnostic and an explanation for the visible shell structure in the point cloud.

Primary implementation: RadialDistribution.*

Designing the Probability Sampler

A three-dimensional orbital cannot be represented by testing every possible position. MatterLab instead generates independent positions whose statistical distribution follows $|\psi|^2$. Over time, the accumulation of points reveals the spatial probability structure.

The sampler separates the problem into radial and angular components. It builds one cumulative distribution for $p_r(r)$ and another for the polar-angle distribution derived from $|Y^l_m|^2$. Random values uniformly distributed between zero and one are then mapped through the inverse cumulative distributions to obtain r and θ.

The azimuthal angle φ is uniform for complex eigenstates because $|e^{im\varphi}|^2 = 1$. Real tesseral bases have explicit azimuthal structure, so MatterLab uses rejection sampling to preserve the correct cosine- or sine-dependent probability.

The final spherical coordinates are converted to Cartesian coordinates and evaluated with the full wavefunction before becoming a QuantumPoint.

Primary implementation: OrbitalSampler.*

Inverse-CDF Sampling

A cumulative distribution function records the probability accumulated up to a value. For the radial coordinate, MatterLab numerically constructs a table representing:

$$F_r(r) = \int_0^r p_r(s)\, ds$$

After normalization, F_r increases from zero to one. A uniform random value u is generated, and the sampler searches the table for the radius where $F_r(r) \approx u$. Interpolation between neighboring entries produces a continuous result.

The same strategy is used for the polar angle. Precomputing these tables moves much of the repeated work outside the inner sampling loop. Once a state is prepared, large point batches can be generated by repeated table lookup, interpolation, azimuth selection, and wavefunction evaluation.

The quality of inverse-CDF sampling depends on table resolution, numerical normalization, interpolation, and the chosen radial boundary. MatterLab exposes table sizes and normalization values through its diagnostic structures so these assumptions can be inspected.

Primary implementation: OrbitalSampler.*, SamplingDiagnostics.*

Rejection Sampling for Real Orbitals

Real tesseral orbitals introduce azimuthal probability patterns that are not uniform in φ. Depending on the chosen basis, the angular amplitude contains cosine or sine terms. MatterLab samples these distributions with rejection sampling.

A candidate φ is drawn uniformly. The code evaluates the relative azimuthal probability at that angle and compares it with another random value. The candidate is accepted only when it falls beneath the target probability. Repeating this process produces the correct distribution without requiring a separate inverse-CDF table for every real-basis azimuthal function.

Rejection sampling is simple and flexible, but its efficiency depends on the acceptance ratio. MatterLab includes diagnostic fields for acceptance statistics. Some throughput-related diagnostic fields are declared but are not currently populated, so performance reporting should not be treated as complete.

This stage is numerical rather than analytical: the target distribution comes from the exact basis function, while the accepted sample set approximates that distribution statistically.

Independent Hydrogen 1s Reference Sampler

MatterLab includes a second sampling method for the hydrogen 1s state. Instead of relying on the general radial CDF pipeline, it uses the known analytical radial distribution for 1s.

The radius is sampled with:

$$r = -(1/2)\ln(U_1U_2U_3)$$

where U_1, U_2, and U_3 are independent uniform random values between zero and one. This expression samples the corresponding gamma distribution for the hydrogen 1s radius. Direction is then selected isotropically.

The independent sampler is valuable because it does not share the same radial table construction as the general method. Agreement between the two approaches provides stronger evidence than checking a method against itself. Sampled mean radius, radial histograms, and spatial symmetry can be compared with the analytical 1s expectations.

This reference does not validate every orbital, but it provides a controlled baseline for the most fundamental state.

Primary implementation: OrbitalSampler.*

Continuous Generation and Reproducibility

MatterLab is designed as a live system rather than a one-time offline calculation. An orbital stream holds the active state and produces batches of samples continuously. The stream can be active, paused, or frozen, and each generation can be associated with identifiers and random seeds.

Deterministic seeds make a sampling sequence reproducible. The same state, basis, table configuration, and seed should reproduce the same pseudo-random sequence. Reproducibility is useful when comparing rendering changes, validating statistical behavior, or documenting a specific result.

The current application clears pending generated batches when the state changes, but points already stored in the renderer are not immediately cleared. Old samples can therefore remain until the GPU ring buffer overwrites them. This can briefly mix two states and should be corrected before state-transition behavior is presented as fully isolated.

Primary implementation: LiveOrbitalStream.*, AsyncOrbitalGenerator.*, Application.*

Part IV – Building the Visualization System

Native Application Architecture

MatterLab is implemented as a native C++20 desktop application. CMake defines the scientific library, core systems, executable, and test targets. GLFW creates the window and input context, GLAD loads OpenGL functions, GLM provides graphics mathematics, Dear ImGui builds the interface, and ImPlot renders scientific graphs.

The main application initializes OpenGL, shaders, framebuffers, renderers, camera controls, UI systems, the quantum-state controller, and the asynchronous generator. During each frame, it processes input, receives completed sample batches, uploads point data, renders the selected visualization modes, applies postprocessing, and draws the interface.

This architecture keeps the analytical quantum code separate from the presentation layer. The scientific engine can evaluate states without depending on OpenGL, while the rendering layer consumes prepared numerical data. That separation is important for testing, future export, and possible reuse in other interfaces.

Primary implementation: CMakeLists.txt, main.cpp, Application.*

The QuantumPoint Data Record

Every accepted sample is converted into a scientific point record. A QuantumPoint contains its Cartesian position and the evaluated properties associated with that position, including density, phase, sign, and generation-related metadata.

The scientific record uses double-precision values. Before rendering, PointCloudRenderer converts it into a smaller float-based GPU vertex containing the attributes needed by the shader. The active GPU representation is approximately 28 bytes per point, while the full CPU-side structure is substantially larger.

This distinction matters for memory planning. The current MemoryEstimator assumes 48 bytes for each scientific point, but the declared structure is ordinarily closer to 72 bytes on the target layout before container and queue overhead. The estimator therefore understates some CPU memory costs.

The point record forms the boundary between the scientific and rendering systems. It allows the renderer to remain independent of how the point was sampled.

Asynchronous Sample Generation

Continuous orbital sampling can be computationally expensive, especially at high point counts or when a state requires more rejection attempts. MatterLab moves sample generation to a worker thread using std::jthread.

The worker repeatedly asks the orbital stream for a batch, places completed batches into a bounded queue, and waits when the queue grows too large. The main thread periodically transfers available batches into the point renderer. This prevents scientific computation from blocking window input, interface updates, or GPU presentation.

Only the rendering thread interacts with the OpenGL buffer. The worker produces ordinary CPU data, which reduces graphics-context complexity.

The present design has a potential synchronization risk: the main thread can change the active quantum state while the worker is sampling from the same stream or sampler, and the reviewed code does not show a clear mutex around every scientific state transition. A future revision should stage immutable sampler configurations and swap them between generations safely.

Primary implementation: AsyncOrbitalGenerator.*, LiveOrbitalStream.*

GPU Ring-Buffer Point Cloud

The main visualization path is a continuously updated point cloud. PointCloudRenderer owns an OpenGL vertex buffer large enough for the selected quality policy. New point batches are written into the buffer in ring-buffer order. When the write position reaches the end, it wraps and begins replacing the oldest samples.

This produces a bounded-memory live visualization. The orbital can continue accumulating apparent detail without allowing CPU or GPU memory use to grow indefinitely. The quality setting controls the maximum population and therefore affects visual density, upload work, and memory use.

Each vertex is transformed by the camera and rendered as a point sprite. The fragment shader discards pixels outside a circular footprint and softens the remaining edge, turning each square raster point into a smooth particle.

The point cloud is not a physical collection of simultaneous electron positions. It is a statistical visualization formed from many independent samples of the same probability distribution.

Primary implementation: PointCloudRenderer.*, point_cloud.vert, point_cloud.frag

Density Color and Transparency

The active point fragment shader maps probability density into a violet-to-white palette. Lower-density points appear darker or more saturated, while higher-density points move toward white. Soft alpha falloff around each point helps overlapping samples blend into a cloud-like result.

Color, point radius, alpha, and brightness are visual encodings. They do not change the sample positions or the quantum probability law. A different palette could display the same scientific dataset without changing the orbital state.

MatterLab also stores phase, sign, age, and additional density information, and a CPU-side color-map helper defines several intended display modes. The current active shader does not fully expose those alternatives. Weighted order-independent transparency buffers and a composite shader are also present as foundations, but the point shader is not yet writing the complete accumulation and revealage outputs required by that method.

The publication should therefore label density coloring as implemented and phase/sign coloring and weighted transparency as partially integrated foundations.

Volumetric Density Grid

MatterLab includes a second representation based on a three-dimensional density grid. VolumeGrid evaluates the wavefunction on a 128 × 128 × 128 Cartesian lattice, stores the resulting probability density, and uploads it to an OpenGL R32F three-dimensional texture.

Unlike the point cloud, the volume stores values at fixed grid locations. It can therefore be sampled continuously by the graphics pipeline, but its accuracy is limited by grid resolution, spatial bounds, interpolation, and transfer-function settings. Increasing resolution improves spatial detail but raises CPU evaluation time, texture memory, and upload cost.

In the reviewed application, the grid is generated at startup for the initial state using the complex basis. It is not rebuilt when n, l, m, Z, or the selected basis changes. The volume renderer is therefore a partially integrated system and may not correspond to the state shown by the live point cloud after user interaction.

Primary implementation: VolumeGrid.*

Ray Marching the Probability Field

The volume renderer draws a bounding cube around the density texture. For each visible fragment, the shader computes the viewing ray, intersects it with the cube, and advances through the texture in small steps. At each step, it samples density, converts the value into color and opacity, and composites the contribution from front to back.

The sample opacity uses a transfer function of the form:

$$\alpha = 1 - \exp(-\text{density} \times \text{scale} \times \text{step size} \times \text{opacity} \times 100)$$

This equation controls visual accumulation. It is not a quantum-mechanical law. The result depends on user-defined or hard-coded display parameters, including density scaling, cutoff, step size, and opacity.

The current shader uses fixed assumptions, including a density cutoff and mapping range initially suited to the hydrogen 1s state. A more complete implementation would derive transfer parameters from the active state or from density percentiles and rebuild the volume whenever the quantum configuration changes.

Primary implementation: VolumeRenderer.*, volume.vert, volume.frag

Camera and Spatial Interaction

MatterLab provides orbit, pan, zoom, framing, standard views, and perspective or orthographic projection. These controls do not alter the orbital calculation. They alter only the transformation through which the existing point or volume data is viewed.

Perspective projection provides depth cues but causes distant regions to appear smaller. Orthographic projection preserves apparent scale and can make node structure easier to compare. Standard orientations provide repeatable front, side, and top views for documentation.

Camera framing must respond to the scale of the active state. High-n states extend farther from the nucleus, while high-Z states contract. A fixed camera distance would make some orbitals appear clipped and others appear too small. MatterLab uses state-derived spatial information to support framing, although transition behavior can be improved as the rest of the state update pipeline is refined.

Primary implementation: Camera.*, Application.*

HDR and Postprocessing Foundations

The application renders through an HDR framebuffer using a floating-point RGBA16F color attachment. The postprocessing shader applies exposure and Reinhard tone mapping before presenting the image to the screen. A simple bright-neighbor bloom foundation is also present.

HDR rendering allows bright overlapping points to accumulate without immediately clipping to standard display range. Tone mapping then compresses that range into a visible image. Exposure and bloom are visual controls, not scientific quantities. They can improve readability, but excessive values can hide nodes or exaggerate density differences.

Weighted-transparency attachments and an OIT composite shader are included in the framebuffer system, but the full technique is disabled and incomplete. Bloom is also disabled by default and is not exposed through the current interface.

These systems demonstrate a planned high-quality rendering pipeline while remaining separate from the validated scientific model.

Primary implementation: Framebuffer.*, hdr_postprocess.*, oit_composite.*

Part V – Productizing MatterLab

Scientific Controls and Presets

The interface exposes the principal scientific state directly. Users can change n, l, m, Z, and the orbital basis while the application validates dependent ranges. Changing l constrains m, and invalid combinations are prevented before they reach the sampler.

Quick-select buttons provide hydrogen, He^{+}, and Li^{2+} configurations. Presets cover representative 1s, 2s, 2p, 3s, 3p, 3d, and 4f states. A preset is not a stored image. It is a defined quantum-state configuration that is passed through the same mathematical and sampling pipeline as a manually selected state.

The controller also calculates derived values such as spectroscopic notation, energy, node counts, and expected radius. High-Z warnings communicate the boundary of the nonrelativistic point-nucleus model.

The interface therefore functions as a scientific state editor rather than a gallery selector.

Primary implementation: UIManager.*, QuantumStateController.*, OrbitalPreset.*

Live Plots and Diagnostics

MatterLab uses ImPlot to display scientific information alongside the three-dimensional view. The radial distribution plot shows $p_r(r)$, allowing peaks and nodes to be interpreted independently of perspective, color, and point overlap. The energy-level plot shows the Z^2/n^2 structure of the hydrogenic spectrum.

Diagnostic structures track radial and angular CDF sizes, normalization values, working radius, density range, acceptance ratio, and other sampling information. Hardware information and quality controls help relate scientific and rendering choices to system capacity.

Not every displayed diagnostic is complete. Fields for sampling duration and samples per second are declared but are not populated by the current generator, so throughput values may remain zero. Invalid-sample counts and numerical integration error reporting also require further wiring.

The diagnostic layer is best described as a strong foundation with several implemented scientific checks and several unfinished performance measurements.

Primary implementation: UIManager.*, SamplingDiagnostics.*

Quality Policies and Memory Planning

MatterLab uses quality policies to control maximum point capacity and related rendering demands. Increasing the point population improves the apparent continuity of the orbital, but it also increases GPU memory use, upload bandwidth, fill rate, and the time required for the distribution to converge visually.

The point renderer’s bounded ring buffer guarantees that the live view cannot grow without limit. The asynchronous queue is also bounded to prevent the worker from generating an uncontrolled backlog.

A memory estimator attempts to predict resource use, but its current CPU-point assumption is too low and omits some container and queue overhead. The full scientific QuantumPoint structure ordinarily occupies more memory than the estimate uses. GPU estimates are closer because the renderer explicitly packs a smaller float vertex.

Future quality planning should report CPU scientific data, queued batches, GPU point data, volume textures, framebuffer attachments, and overhead separately.

Validation Strategy

Validation must occur at several levels. State validation checks that n, l, m, and Z satisfy their mathematical constraints. Analytical checks compare energy, node counts, and expected radius against closed-form expressions. Numerical checks confirm that radial distributions integrate to approximately one over the selected finite boundary.

Sampling checks compare generated histograms with the target radial and angular distributions. The independent hydrogen 1s sampler provides a second method for the ground state. Statistical mean radius should approach the analytical value as sample count increases.

Visual checks verify symmetry and node structure, but visual agreement alone is not sufficient because color, opacity, camera angle, and point count can create convincing but incorrect images.

CMake references seven Catch2 test files, and compiled test artifacts were included in the uploaded project. However, the test source files and populated test-results log were absent from the review archive. The present case study can document the intended validation strategy, but it should not claim that the complete test suite was independently audited or certified.

The Finished Workspace

The completed MatterLab workspace brings the scientific state, numerical diagnostics, plots, and live visualization into one native application. A user can select a hydrogenic state, inspect its derived properties, watch the point cloud accumulate, navigate the orbital in three dimensions, and compare the visual result with radial and energy plots.

The strongest completed path is the continuously generated probability point cloud. It begins with validated quantum numbers, evaluates analytical hydrogenic functions, constructs probability distributions, samples positions asynchronously, transfers batches to the GPU, and renders the accumulated result interactively.

The volume renderer, alternative color modes, weighted transparency, bloom, export, and transition staging demonstrate expansion paths but remain incomplete or partially connected. MatterLab should therefore be presented as a functioning orbital laboratory with a mature point-cloud core and several advanced visualization foundations, not as a finished general-purpose quantum simulator.

Part VI – Interpretation and Reflection

Scientific Meaning Versus Visual Treatment

MatterLab separates scientific output from display decisions.

The scientific model determines valid states, energies, node counts, radial scale, wavefunction amplitude, probability density, phase, sign, and the statistical distribution of sampled positions.

The numerical method determines finite boundaries, integration tolerances, table resolution, interpolation, random sampling, batch size, grid resolution, and statistical convergence.

The visual treatment determines point size, palette, brightness, opacity, exposure, bloom, camera projection, density cutoff, and the transfer function used by the volume renderer.

This distinction is essential. A bright region is not automatically a new physical structure; it may be the result of the selected tone map. A rendered surface is not a literal orbital boundary; it is a threshold or opacity accumulation. The orbital is the quantum state and its probability structure. The rendered image is one controlled interpretation of that structure.

Known Limitations

MatterLab currently models only stationary, nonrelativistic, one-electron Coulomb states. It does not model interacting many-electron atoms, molecules, spin, time evolution, relativistic effects, finite nuclear size, or QED corrections.

The point cloud updates when the state changes, but existing GPU points are not immediately cleared, allowing temporary mixing between generations. State changes may also interact with the sampling worker without a fully isolated immutable configuration.

The volume grid is generated only for the initial state and is not rebuilt with later control changes. Phase and sign are calculated but are not fully exposed as active color modes. Weighted order-independent transparency, bloom, and image export are foundations rather than completed features.

Several diagnostics are incomplete, the memory estimator understates CPU point size, and the included archive did not contain auditable unit-test source or test results. These limitations define the next engineering pass and prevent the case study from overstating the current product.

Next Development Steps

The first priority is correctness during state transitions. Changing n, l, m, Z, or basis should create an immutable sampler configuration, stop or retire the previous generation safely, clear or deliberately crossfade the renderer, and rebuild every dependent representation, including the volume grid.

The second priority is scientific validation. Unit-test source should be restored, reference values documented, radial and angular distributions compared against analytical or independently generated data, and error estimates added to numerical integration reports.

The third priority is completing the rendering foundations: active phase and sign color modes, state-adaptive density scaling, completed weighted transparency, controlled bloom, reproducible export, and synchronized point and volume views.

Longer-term expansion could include superpositions, time-dependent phase evolution, molecular orbitals, many-electron approximations, and broader material systems. Each addition would require a new scientific model rather than merely a new visual preset.

Conclusion

MatterLab began with a simple question: how can a mathematical quantum state become an interactive visual object without losing the distinction between physics and presentation?

The resulting system answers that question through a traceable pipeline. Valid quantum numbers define a hydrogenic state. Analytical radial functions and spherical harmonics construct the wavefunction. Probability density becomes a numerical distribution. Independent samples become scientific point records. An asynchronous native pipeline moves those records into a bounded GPU renderer, where they accumulate into a navigable orbital visualization.

The project’s value is not only the final image. It is the connection between each image and the equations, numerical methods, code paths, assumptions, and limitations that produced it. MatterLab demonstrates how scientific software can remain visually compelling while openly separating validated output from approximation and artistic treatment.

Technical Appendix — Core Formula Index

State validity:

$n \geq 1,\ 0 \leq l < n,\ -l \leq m \leq l,\ Z > 0$

Energy:

$$E_n = -Z^2 / (2n^2)$$

Dimensionless radius:

$$\rho = 2Zr / n$$

Radial wavefunction:

$$R_{nl}(r) = N_{nl} \exp(-\rho/2)\rho^l L^{(2l+1)}_{n-l-1}(\rho)$$

Spherical harmonic:

$$Y^l_m(\theta, \varphi) = N_{lm} P^l_m(\cos\theta) e^{im\varphi}$$

Full wavefunction:

$$\psi_{nlm} = R_{nl} Y^l_m$$

Probability density:

$$|\psi|^2 = \psi^* \psi$$

Radial probability distribution:

$$p_r(r) = r^2 |R_{nl}(r)|^2$$

Expected radius:

$$\langle r \rangle = [3n^2 - l(l+1)] / (2Z)\ a_0$$

Cartesian conversion:

$$x = r \sin\theta \cos\varphi$$

$$y = r \sin\theta \sin\varphi$$

$$z = r \cos\theta$$

Hydrogen 1s reference radius:

$$r = -(1/2)\ln(U_1 U_2 U_3)$$

Technical Appendix — Primary Code Map

PhysicalConstants.hpp — atomic units and conversion constants

QuantumState.* — state validation, energy, notation, and node counts

QuantumStateController.* — UI-safe state management and derived values

AssociatedLaguerre.* — generalized Laguerre recurrence

AssociatedLegendre.* — associated Legendre recurrence

SphericalHarmonics.* — complex and real angular bases

HydrogenWavefunction.* — radial and complete wavefunction evaluation

ProbabilityDensity.* — density, phase, and sign

NumericalIntegration.* — adaptive Simpson quadrature

RadialDistribution.* — normalization, boundaries, nodes, and peaks

OrbitalSampler.* — CDF construction and orbital sampling

AsyncOrbitalGenerator.* — background batch generation

PointCloudRenderer.* — GPU ring buffer and point rendering

VolumeGrid.* — CPU density lattice and 3D texture upload

VolumeRenderer.* — volume cube and ray-march controls

UIManager.* — controls, plots, and diagnostics

Application.* — system initialization and main loop