ChatGPT

Real-Time Star Simulator Rendering Techniques

Photosphere Rendering

- **Procedural Granulation:** Simulate the Sun's grainy photosphere texture using multi-octave noise. For example, 3D/4D noise (like Perlin or Simplex noise) can approximate convective granules [1]. Using ~4 octaves strikes a balance between detail and performance [2] (higher octaves add finer detail at a steep performance cost [3]). Animate the noise over time by including time as a 4th input to the noise function, creating evolving granulation patterns [4]. This mimics the boiling motion of convective cells on the photosphere.
- **Limb Darkening Shader:** Implement limb darkening so the star's edge appears dimmer than the center. Use a polynomial intensity falloff based on the view angle ($\mu = \cos\theta$). For example, a second-order model for the Sun at 550 nm is $I(\mu)/I(1) \approx 0.3 + 0.93\,\mu + (-0.23)\,\mu^2$ [5], meaning only ~30% intensity at the limb compared to center [6]. In practice, you can use a dot(normal, view) term (or Fresnel term) in the fragment shader to modulate brightness across the disk [7]. This ensures a bright center and progressively darker edges, reproducing the observed limb darkening effect.
- **Sunspot Visualization:** Incorporate dark sunspots with realistic umbra–penumbra structure. One approach is using a low-frequency noise mask to carve out spots in the granulation texture [8]. For example, take the base noise and subtract an "umbra" value derived from a secondary noise function of larger scale [9]. Clamp this secondary noise to positive values (using max(...,0)) so that it produces discrete dark spots instead of just dimming everywhere [10]. The result darkens certain areas (umbrae) and yields a spotty appearance on the surface. To mimic penumbrae, consider surrounding the dark umbra with a slightly higher noise threshold ring or radial gradient. Additionally, implement the **Wilson effect**: sunspots are slightly depressed below the photosphere. You can simulate this by adjusting the surface normal or depth within spot regions, causing foreshortening at the limb so spots appear as shallow "dishes" [11]. This geometry tweak, combined with limb darkening, will make limb-hugging spots look inset.
- **Faculae Near Limb:** Add bright faculae, especially near the solar limb. Faculae are small magnetic bright spots that are usually invisible at disk center but become obvious toward the edge where the background is dimmer [12]. In the shader, you can boost intensity in tiny regions (e.g. along granule boundaries) that are at high viewing angles. One way is to use a facula noise mask tied to magnetic field regions (if available) or simply procedural bright speckles, and amplify them by a factor that increases as the view angle to normal increases. Limb darkening naturally enhances their contrast [13], so faculae will "pop" near the edge [14]. Ensure these faculae are tightly localized and perhaps slightly larger than granules, and tie them to areas around sunspots or random patches, since faculae often cluster around active regions.

Corona & Outer Atmosphere

- **Volumetric Corona Raymarching:** Render the corona as an emitting, optically-thin volumetric layer around the star. Cast rays through a spherical volume and integrate the plasma emissivity along each ray [15]. Because the corona is optically thin, assume no significant absorption – simply sum up emissivity contributions at each step (line-of-sight integration of *j(ds)*). Use a **raymarching shader** that starts at the inner corona (just above the photosphere) and samples outward. The basic radiance addition per step: `I += emissivity(position) * step_length`. A physically-realistic emissivity can be tied to electron density and temperature (e.g. via an n^2 dependency for collisional processes), but an artistically tuned function may suffice for real-time. To optimize, implement **early termination** (if the density becomes negligible) and **depth skipping**. Leverage the corona's smooth density profile with adaptive stepping: far from the Sun where density changes slowly, take larger steps; near the solar limb or in dense features, use smaller steps for accuracy [16]. If using a Signed Distance Field for the star (see below), you can start the ray at the corona's inner boundary to skip the empty photosphere.

- **Density Falloff Models:** Provide both a physically-based and an adjustable density falloff for the coronal plasma. For high realism, use a hydrostatic equilibrium or empirical model. For example, an exponential drop: $n(r) = n_0 \exp[-(r-R_\odot)/H]$, where H is the scale height (~50–100 Mm for million-degree plasma), or a piecewise power-law (the classic Baumbach-Allen model uses a sum of power laws to fit observed corona density [17] [18]). This can be encoded in the shader to compute density vs. radius r. Also consider a temperature profile (e.g. ~1–2 MK base corona with slight increase further out) if computing emissivity with temperature dependence. **Quality toggle:** On high settings, use the full formula (or even 2-3 component power laws for inner/mid/outer corona [17]); on low settings, simplify to a single exponential or even a constant falloff (which is less accurate but faster). Expose tweakable parameters (scale height, base density) so the look can be artist-directed when needed.
- **HDR Bloom & Extreme Brightness:** The corona is ~10^6 times fainter than the photosphere [19], yet visible in eclipse due to the photosphere being occulted. In a simulator, use high dynamic range rendering to handle this contrast. Render the photosphere at an intense brightness and the corona at a much lower base intensity. Then apply an **HDR bloom** post-process to make the bright Sun disk "bleed" light into surrounding pixels, imitating glare. This will create a glowing halo effect for the corona. For example, in John Whigham's star rendering, he added an extra halo sprite to "max out" bright areas and create a lens flare and glow [20]. In our context, a bloom filter with a large radius can make the faint corona visible as a hazy glow around the saturated stellar disc. This approach lets the photosphere appear blindingly bright (washed out white with bloom), while still allowing coronal structures to be seen in relative terms. Tune the bloom intensity so that inner corona features get a slight bloom and the overall scene mimics the visual dynamic range of real observations [21] [22].
- **Magnetic Loop Streamlines:** To convey coronal structure, render representative magnetic field lines as glowing loops or streamlines threading through the corona. Field lines in active regions guide the bright coronal loops seen in EUV/X-ray images. Implement a **streamline tracer** that follows the magnetic field vector field (see *Magnetic Field Visualization* below) and outputs a polyline or tube. Render these lines with additive blending, perhaps using a soft glow or tapered opacity toward the ends. Assign them a brightness or color tied to local plasma density or temperature (e.g. make closed active-region loops brighter, open field lines faint). This technique was used in NASA's ENLIL CME visualization: field lines were drawn emanating from the Sun to show structure, with the Sun drawn as an emissive sphere at the center [23]. In real-time, you can seed a dozen or more such lines from random points on the photosphere (biased to stronger field regions) and update them each frame or intermittently. **Optimization:** at lower quality settings, reduce the number of field lines, or draw only short segments (e.g. only the most visible portions) to save performance. These streamlines do not affect physics but greatly enhance the visual impression of a structured, magnetic corona.

Multi-Wavelength Visualization Modes

- **SDO/AIA EUV Channels & False Color:** Implement preset modes to visualize the star in specific extreme ultraviolet wavelengths, following SDO/AIA's channels. Each EUV channel highlights plasma at different temperatures and ions [24]. Key wavelengths and their typical color mappings: **171 Å** (Fe IX, ~0.6 MK) often colored gold/blue; **193 Å** (Fe XII & Fe XXIV, ~1.5 MK and 20 MK for flares) colored green; **211 Å** (Fe XIV, ~2.0 MK) colored purple; **304 Å** (He II, ~0.05 MK, chromosphere/ transition region) colored red. You can provide a composite "SDO multi-color" view where 304, 193, 171 Å are mapped to R, G, B channels respectively [25] [26] – this yields a vivid false-color image often used in public outreach. Alternatively, allow switching to each wavelength in grayscale or its characteristic color. Use the physical data: e.g., for 171 Å mode, only show structures at ~0.6 MK (the quiet corona and upper transition region) – in practice, this could mean sampling the volumetric corona with a filter for temperature around 10^6 K and displaying those voxels brightly, while cooler or hotter plasma is deemphasized. Similarly, 335 Å (Fe XVI, ~2.8 MK) will highlight active region cores at ~2–3 MK [27] [28]. Providing these modes requires simulating or storing multi-temperature plasma data. For simplification, you can derive each channel's brightness from the base simulation by weighting by an approximate response function (e.g., use a gaussian or window around the characteristic temperature for that channel).
- **Physical Interpretation of Wavelengths:** In the prompt, include guidance on what each wavelength reveals so Claude can implement the proper mapping. For example: 171 Å shows the **upper transition region/quiet corona** (Fe IX emission) – visualize fine coronal loops and large fan loops [24]. 193 Å shows the **active corona** (Fe XII) and also **flare hot plasma** (Fe XXIV) – so in 193 Å mode, active region loops and any flare core should be bright green [29]. 304 Å shows the **chromosphere & prominences** (He II) – depict the chromospheric network and filament structures in red. 211 Å (Fe XIV) highlights slightly hotter corona (~2 MK) – emphasize active region loops (color them orange or purple as per AIA). 335 Å (Fe XVI, ~2.5 MK) also shows hot active regions (perhaps color azure) [27] [30]. **94 Å** (Fe XVIII, ~7 MK) and **131 Å** (Fe VIII/Fe XXI, dual peaks at ~0.4 MK and ~10 MK) are special: they are designed for flares. In 94 Å or 131 Å mode, the core of solar flares and very hot post-flare loops should light up intensely (these channels often appear teal-blue in NASA imagery). Document the primary ions and temperatures for each channel so the code can use or approximate them [31]. This way, Claude can incorporate these specifics (perhaps as lookup tables or conditionals) to produce visually distinct maps for each wavelength.
- **Soft X-Ray Telescope (XRT) View:** Include a mode to simulate the Sun's appearance in soft X-rays (similar to Hinode XRT or Yohkoh images). X-rays reveal the hottest plasma (several million K) in the corona, typically during flares or in active regions. The visual is usually monochromatic or false-colored (often blue-white). To emulate this, take only the hottest portion of the corona volume (e.g. >5 MK) and render it as a high-intensity structure. Flare loops should become dazzling bright, and quiet Sun regions should vanish (since XRT is less sensitive to <1 MK plasma). Because X-ray images are usually high contrast, consider applying a different tone mapping – maybe a simple grayscale or inverted grayscale (bright features on dark background) or a blue tint. For reference, Hinode XRT images show bright active region loops and void-like coronal holes in a single band [32]. We can instruct Claude to use a similar approach: compute an "X-ray brightness" as an integral of emission measure above a certain temperature, and display that. The result should be a stark, high-contrast view emphasizing flare geometry and hottest loop tops.
- **H-α Chromosphere Mode:** Provide a mode for the chromosphere using an H-alpha filter view (656.3 nm, deep red). This mode should display **filaments** and **prominences**, **plage**, and low-altitude flares. The Hα appearance is very different: filaments appear as long, dark ribbons on the disk (they

are cool plasma above the surface absorbing background light) [33], while the same structures at the limb appear as bright prominences against space [34]. To simulate this, the code can use the magnetogram or field data to identify filament channels (they typically form above polarity inversion lines in magnetograms). Claude could draw filament structures as dark lines or elongated blobs on the bright disk. Surround active region sunspots with **plages** – bright chromospheric patches (which correspond to faculae in photosphere, but in Hα they appear as bright areas around spots) [35]. These can be implemented by brightening areas of strong magnetic field on the disk in Hα mode. Also, implement **flare ribbons**: in Hα, flares are seen as sudden bright ribbons or kernels near sunspots [36]. When a flare event is triggered (perhaps by a reconnection event in the simulation), draw two bright streaks on either side of the magnetic neutral line in the active region, and have them expand apart over time (simulating the classic two-ribbon flare; we can use the field line topology to determine the orientation of ribbons). The Hα view should be colored in a reddish palette (as Hα is red light). Optionally, you can overlay a darkening of the overall disk (since the Hα continuum is fainter than visible light) to mimic the narrower bandpass. By instructing Claude in these terms – dark filaments on disk, bright prominences at limb (perhaps using the 3D field lines with cool dense plasma attached to them), plage around spot regions, and dynamic flare ribbons – we ensure a realistic chromospheric visualization [33] [35].

- **Magnetogram Mode:** Allow viewing of the star's surface magnetic field (line-of-sight component) in a simplified black/white or bicolor map. This is akin to SDO/HMI magnetograms. In magnetogram mode, map positive polarity (field lines emerging from the surface) to one color and negative polarity (entering the surface) to another. Commonly, one uses white for positive and black for negative in greyscale [37], or colorized versions like yellow vs. green [38]. We should instruct to use a **perceptually uniform** colormap or simple high-contrast colors for this. For instance, **white** for $B>0$, **black** for $B<0$ (classic magnetogram), or a blue-red scheme. The user's clarification suggests using a colored version, e.g. yellow for positive, green for negative [38]. The magnetogram is essentially a 2D texture on the photosphere where intensity = B (clamped to some max range for visibility). Include guidance on scaling the values (solar field strengths can be thousands of Gauss in spots – likely map the maximum to full white/black to avoid out-of-range). This mode doesn't require 3D rendering complexities, but it's important for debugging and for scientific context, so Claude should implement it as a separate render pass or as an optional surface shader.

Magnetic Field Visualization

- **Field Line Integration (RK4 vs. Euler):** To depict magnetic field topology, seed field lines and integrate through the 3D magnetic field. Use a robust integration method like 4th-order Runge-Kutta for accuracy [39]. A simple Euler integrator will be faster but accumulates error and can lead to visibly jagged or incorrect lines, especially over long distances [40]. We want smooth, correct field lines (particularly if they need to connect back to the surface or extend radially). So instruct Claude to implement an RK4 integrator that, at each step, samples the magnetic field **B(x)** (either from an analytical model or a grid) and advances the line in small increments. The step size might be adaptive: smaller steps in strong-field regions or when curvature is high, larger steps in uniform regions. **Quality scaling:** For lower tiers, it's acceptable to use fewer integration steps or even a second-order RK2 (midpoint) to save computation, but emphasize that high/ultra quality should use RK4 with a fixed small step for precision. The integration should proceed in both directions from the seed point (to trace the full line through positive and negative directions). The resulting set of points forms a polyline representing a field line. These can be rendered as thin glow lines as mentioned above. Optionally, implement **adaptive termination**: stop integrating when a line reaches a certain height (e.g. beyond the “source surface” if it’s open) or when it re-enters the photosphere.

- **PFSS Extrapolation Algorithm:** For generating a global coronal field, suggest using the Potential Field Source Surface model. Claude should understand the basics: assume the magnetogram (surface radial field) as the boundary condition at $r=R_\odot$, and that the field becomes purely radial at some outer radius (the source surface, typically $2.5R_\odot$). The code can use a spherical harmonic expansion to solve Laplace’s equation $\nabla\cdot B=0$ in the volume [41]. Specifically, mention that algorithms like the one by Altschuler & Newkirk (1969) or Schatten et al. (1969) are classical approaches [42] – these expand the surface field into spherical harmonics and compute the 3D field as the gradient of a scalar potential Φ. (We can include references or even pseudo-code for how coefficients are computed if needed, though Claude can probably fetch details from those names.) Alternatively, note that **pfsspy** (a Python library) exists and implements this; while Claude can’t use it directly in Rust/WGSL, he could take inspiration from it. For our prompt, outline the steps: (1) sample or define a surface field (e.g. a simple dipole plus some random active region contributions), (2) expand it in spherical harmonics up to some degree L_max for desired detail, (3) apply radial boundary condition at source surface ($B\theta=0$, $B\varphi=0$ at r=R_SS), (4) compute coefficients and reconstruct B_r, B_θ, B_φ in space. This is heavy for real-time if done fully; thus propose using a **precomputed field** on a spherical grid or simplifying to a superposition of analytical fields (dipoles, etc.) for real-time. For example, for a procedural approach: use a dipole term for global field and add a few localized bipolar regions (which can be approximated by submerged dipoles or multipoles). The prompt can instruct Claude on simpler formulas (e.g. a dipole: **B** = *(μ·r) r*/r^5 - μ/r^3 for a dipole moment μ). But also mention that at high quality an actual PFSS solution yields the iconic large-scale coronal loops and open field lines. This gives Claude both an easy route and a difficult route. Emphasize that the code should allow toggling between a **procedural field** (for performance or different star types) and an optional **real data import** (e.g. from a solar magnetogram FITS file) followed by PFSS – but the latter can be a stretch goal. By including terms like “spherical harmonic PFSS” and references [42], Claude can look up specifics if needed.
- **Volume Rendering of |B|:** Another way to visualize the magnetic field is by rendering the magnitude of the field (|B|) in the corona or volume. This could be useful for showing where the field is strongest (e.g. close to active region loops, or in flux ropes) even when not drawing individual lines. Instruct implementing a 3D texture or sampling function for |B| throughout space, and using a color map to render it. For instance, one could raymarch a semi-transparent volumetric representation of |B|, with a transfer function that highlights intermediate field strengths. However, to keep things simpler, perhaps use this technique for a static image or diagnostic mode rather than default. If implemented, stress **perceptually uniform colormaps** for mapping field strength to color, so as not to mislead the eye [43]. Avoid the classic “rainbow” (jet) map; instead use something like **viridis** or **plasma**, which are designed to vary smoothly in perceived brightness [43]. For example, viridis goes from dark purple (low) to bright yellow (high) in a uniform way. This ensures that equal changes in field strength look like equal changes in brightness, preventing bias. The prompt can note: “use a viridis colormap for field strength visualization for clarity”. Practically, the shader would take |B| at each sample point and convert to an RGBA color, accumulating with alpha blending. **LOD consideration:** volume rendering |B| will be expensive; perhaps only enable this on high settings or for static screenshots. On lower settings, rely on field lines and surface maps instead.
- **Magnetic Reconnection Visualization:** Include methods to show magnetic reconnection events, which are critical for flares and CMEs. In physics, reconnection happens in current sheets where field lines break and rejoin, converting magnetic energy to kinetic and thermal energy. Graphically, we can represent this by highlighting regions of strong electrical current or where field line connectivity

changes. For instance, in a simulation if two field lines from opposite polarities are forced together (e.g. in a low corona current sheet), we could momentarily draw a bright **flash** or glow at that location to indicate energy release [44]. Additionally, after reconnection, field lines reconfigure – closed loops might open or vice versa. To portray this, we can animate field lines: maybe have some field lines disappear and new ones appear connecting different footpoints. For a flare ribbon manifestation (as described in Hα mode), the footpoints of reconnected loops light up (that's what causes the optical ribbons). So Claude's code could, upon a triggered reconnection, do a few things: (a) spawn a short-lived bright sheet or blob in the corona (to mimic the bright current sheet or hot plasma blob, like in EUV images a cusp-shaped flare loop top glows), (b) brighten the lower atmosphere at the footpoints (flare ribbons), and (c) generate **particle jets** or ejecta – for example, a burst of high-speed particles moving along newly open field lines (this ties into the particle system for solar wind/CME). We might reference a conceptual description: NASA visualizations show that in turbulent magnetic fields, reconnection sites appear as bursts of light and eject bi-directional jets of plasma [44]. In implementation, one could emit a set of particles from the reconnection point that shoot out along the local field direction (both directions). Another visual trick: field lines that undergo reconnection could be drawn with a different color or as dashed lines that then snap to a new configuration. In summary, instruct Claude to **visualize reconnection** by a combination of transient glow (flash of light), dynamic field line changes, and particle ejecta, to really capture the dramatic nature of flares.

Particle & Dynamic Effects

- **Solar Wind Particle System (Parker Spiral):** The continuous solar wind outflow can be depicted with a GPU particle system. Emit a steady stream of particles from the star's surface (or just above it), with initial velocities radially outward. To capture the **Parker spiral** effect (caused by the Sun's rotation twisting the outflowing plasma), give the particles a slight transverse velocity component equal to the local corotation speed. In practice, at the solar surface the plasma is tied to ~2 km/s rotational speed at the equator (the Sun rotates ~25 days). By the time the wind reaches 1 AU, this imparts a ~45° spiral angle for a ~400 km/s wind [45] [46]. We can simplify: in the particle update shader, besides radial outward velocity, add an azimuthal velocity = $\omega \times r$ (where ω is Sun's angular speed, and r is radial distance). As r increases, this yields a spiraling trajectory. This effect will be subtle near the Sun but more apparent further out (if the view zooms out to heliospheric scales). To visualize the wind, each particle could be a tiny glowing point or a streak (stretch its trail in direction of motion for a "speed line" look). Use additive blending so that dense areas (near the Sun or in stream interaction regions) appear brighter. Also consider coloring the particles by their source latitude or associated magnetic polarity (e.g. fast wind from coronal holes could be colored differently than slow wind from streamer belt). **Performance:** The user targets mid-range GPUs, so a moderate number of particles (a few thousands) updated with a compute shader should be fine at 60fps. For lower quality settings, you could lower the emission rate or particle count. You could also freeze the particle system on low-end to a simpler representation (like just a static textured flow).
- **Coronal Mass Ejection (CME) Animation:** CMEs are large expulsions of plasma and magnetic field. Implement a mechanism to trigger CMEs that propagate outward. Graphically, one approach is to create an expanding translucent shell or cloud emanating from a region on the star. For example, instantiate a spherical frustum or "bubble" volume at the low corona, give it an initial radius of a few percent of $R_\odot$, and expand it over time (scaling up). The interior of the CME can be a slightly higher-density plasma than background corona. You can raymarch or sprite this cloud with an outward-moving velocity. Increase its opacity or brightness at the shock front (leading edge). In real CMEs, the

front is bright (compressed plasma) [47] , so possibly render a bright ridge on the outer boundary of the volume (this could be achieved by having the volumetric shader highlight the outermost isosurface). Also, leave behind a **dimming** in the corona where the plasma evacuated – i.e., after CME launch, reduce the base coronal density in that region to mimic coronal dimming. If including magnetic field lines, you might show the eruption of a flux rope: perhaps a bundle of field lines bulging out and lifting off with the CME. For simplicity, you could predefine a torus or loop of field lines and animate it moving outward, then fade it out as it leaves the domain. Provide Claude with details like typical CME speeds (could be 100–1000 km/s) and the idea of a shock if speed > ambient solar wind speed [48] so that he can incorporate a fast expansion and maybe even a secondary shock visual (maybe a faint outer shell ahead of the main ejecta). Overall, the CME should be an eye-catching event: quickly expanding volume, leading edge bright, possibly trailing structure. **Quality scaling:** on low settings, maybe skip volumetric raymarch and just enlarge a transparent sphere; on high, do full volumetric with internal texture (Perlin noise to give structure) and interactive lighting (bloom on the shock, etc.).

- **Solar Flare Brightening:** Simulate solar flares as sudden intense brightenings in various layers. In EUV/X-ray views, a flare manifests as a rapid increase in brightness of some active region loops and the appearance of **flare loops** (post-flare arcades) that glow in X-ray/EUV. We instruct Claude to implement flares by temporarily boosting the emission of plasma in the region of a reconnection event. For example, if a reconnection is detected (or triggered manually), take a set of low-lying field lines in that active region and ramp their temperature and density up (e.g. set them to 10 MK, which makes them shine in 94 Å and 131 Å channels). Also create a small volumetric glow at the loop tops (flaring loops often have bright cusps). In optical (Hα) view, as mentioned, two bright **ribbons** should form on the surface on either side of the neutral line [36] . These ribbons can be animated to move apart over a timescale of minutes (in simulation time: perhaps a few seconds of real-time if time-accelerated). Essentially, start them near the initial flare site (sunspot region) and gradually increase their separation while fading them out. The code might accomplish this by having pre-defined ribbon geometry (like textured quads or a parametric curve) that is made visible during flares. Additionally, flares produce **particle beams** (which cause aurora on Earth etc., but for the Sun itself you might not show that). However, you can incorporate some energetic particle effect – maybe a burst of fast particles along newly reconnected open field lines (representing SEP – solar energetic particles). This overlaps with the reconnection jets idea above. Also, integrate the **time-domain** aspect: flares have classes (C, M, X by increasing strength) and durations. Provide some parameters like rise time and decay time. For simplicity, use a short rise (a second or two) to peak brightness in the simulation, then let the flare decay over, say, 5–10 seconds, during which the bright loops remain visible. This will allow Claude to code a timing function or state machine for flares. Each flare event could log its energy and be classified (which could tie into the UI or selection of “flare class” if desired – e.g., the user could trigger an X-class flare, resulting in a very bright effect).
- **Prominence Dynamics:** Simulate prominences – these are cool, dense plasma suspended in magnetic loops, which can slowly flow and sometimes erupt. In the visualization, prominences could be shown either in a dedicated H-alpha mode (dark filaments on disk, which we covered) or even in the EUV view as bright structures in 304 Å. To animate prominence **flows**, one strategy is to use a particle system or animated texture that follows along field lines. For example, for each major field line in an active region, spawn a set of particles that have a heavy mass (to move slowly) and make them drift from one footpoint to the other following the field line shape (this could be done by parameterizing the field line curve and moving particles along it). Give them a reddish or bright 304 Å appearance. This would simulate the motion of plasma within a prominence (often you see blobs of material streaming along the prominence spine). If using a more continuous approach, perhaps map a flowing noise texture onto the field line (treat the field line like a tube and slide a noise

pattern along it). That can create an illusion of material sliding. Additionally, allow prominences to **erupt**: if a magnetic configuration breaks, the prominence can lift off as part of a CME. You might tie this to the CME system: if a CME is triggered from an active region, find if a prominence exists there and then animate it lifting off (the particles speeding up and moving outward), eventually disappearing or dissipating. Graphically, during eruption, you can transition the prominence from cool/dark (if in Hα mode) to bright (if it gets heated in EUV). This is complex, but even a simple upward motion plus fade-out of the filament particles would convey that the prominence has been expelled. Ensure that on normal conditions, prominences are primarily around the limb (for visual effect in 3D, since on disk they'd be invisible except Hα). Perhaps spawn them along strong field lines that arch high off the surface (like helmet streamer cores or near sunspots). Provide Claude guidelines on typical speeds (prominence flows are tens of km/s slow, eruptions can accelerate to >100 km/s). The combination of these details will let him implement a compelling prominence system.

Shader Architecture & Performance

- **Adaptive Raymarching and SDFs:** The simulator will employ raymarching for volumes (atmosphere, corona) and possibly distance fields. Emphasize techniques to optimize this. For instance, use **sphere tracing** with Signed Distance Fields for known geometric bounds [49]. Define an SDF for the star's surface (a sphere of radius R☉). This allows the raymarcher to skip immediately to the corona's start by jumping to the sphere intersection point instead of tiny steps. Similarly, if we impose an outer boundary for the corona (say, fading out by 2–3 R☉), that could be another sphere SDF to know when to terminate. The Wikipedia entry on ray marching notes that SDFs let you take as large a step as possible without missing surfaces [50] – Claude can leverage that idea to accelerate traversing empty space. Also mention **empty space skipping**: maintain a low-res 3D grid that marks regions of non-negligible density, and when raymarching, skip ahead when in truly empty cells (this is more relevant for irregular volumes, maybe not needed if corona is smoothly varying). Additionally, use **early ray termination**: for example, if rendering in HDR and after adding the photospheric intensity the value is already above a bloom threshold, further corona contribution might not visibly matter – you could cut it off. But that's optional, since we do want to see corona.
- **Compute Shaders for Physics:** The project can utilize compute shaders for heavy computations that need to run per-frame. For example, magnetic field integration (finding field lines) can be done in a compute shader dispatch where each thread traces one line, storing the vertices in a SSBO. This offloads the CPU. Similarly, procedural noise generation for the granulation texture could use a compute shader to generate a dynamic texture each frame (especially if doing real-time granule evolution beyond simple scrolling). Particle systems (solar wind, CMEs) definitely benefit from compute: update particle positions in parallel, perhaps with a thread per particle. Compute shaders can also handle volume preprocessing: e.g., if you want a 3D texture of coronal density or temperature, a compute shader could fill that based on a formula so the raymarch shader only does lookups (though computing on the fly in the raymarch might be fine too). We want Claude to structure the rendering pipeline such that as much math as possible is done on GPU in parallel, avoiding CPU bottlenecks. If the final language is Rust with wgpu (WGSL shaders), the prompt should specify that **WGSL is similar to GLSL**, with notable differences (e.g. WGSL uses `let` and requires explicit types, and texture sampling is done through bindings rather than GLSL's global functions). We can provide pseudocode in GLSL, and note any needed translation tips to WGSL/HLSL (for instance, no implicit `gl_FragColor`, one must output to a `var<location>` in WGSL, etc.). By giving Claude a clear mapping, he can translate the shader code accordingly. Also, consider **parallel**

reduction tasks: if needed (maybe computing global brightness for exposure adaptation), compute shaders can sum values efficiently. It's worth mentioning if there's a need for dynamic exposure ("eye adaptation"), though not explicitly asked, but perhaps as a stretch feature.

- **Multi-Pass Rendering Pipeline:** Clarify the render pipeline structure in the prompt so Claude knows how to layer the effects. A likely approach: **Pass 1** – render the star's photosphere (the opaque sphere) with its shader (granulation, limb darkening, sunspots) into the color and depth buffer. **Pass 2** – volumetric corona raymarch. This can be done either in a fullscreen quad shader that samples along rays, or by raymarching in the fragment shader of a drawn geometry (like drawing a big sphere that encompasses the corona and raymarching between entry/exit). The corona blending should be additive or alpha-blended. Since the photosphere was written to depth, use depth testing so that the raymarch skips pixels covered by the opaque disc (or in shader, start the ray at the photosphere's depth). **Pass 3** – magnetic field lines. These can be rendered as line primitives or thin triangle strips, with an additive glow material. Use depth test so they get occluded by the star (or deliberately render them on top if you want x-ray vision, but probably occlusion is more realistic). **Pass 4** – particles (solar wind, CMEs). These are likely point sprites or billboards, also additive blending (since they represent glowing plasma or scattered light). They should be depth-tested and depth-written if they are opaque; for small glowy particles, depth writing can be off. **Pass 5** – screen-space postprocessing: apply HDR bloom and tonemapping. The bloom will make bright points (flares, sun disk) bleed. Possibly also implement motion blur on fast particles if needed. Also consider an **exposure adjustment** if toggling between viewing modes (for example, photosphere vs corona might require different exposure; could auto-expose to the brightest feature each frame or allow manual exposure settings). All these passes need to run efficiently, so mention using **framebuffer re-use** where possible, and ensuring not too many passes for low-end. The engine-agnostic request suggests we outline this generally, not in terms of a specific API. So describe it conceptually as above. Also, mention using **double buffering or triple buffering** for things like the dynamic procedural textures (so that one can update one while using another).
- **Level-of-Detail & Scalability:** The prompt should explicitly instruct on quality tiers to target performance vs fidelity. Summarize some strategies:
- *Granulation*: On "Low", perhaps use a static precomputed texture or 1-2 octaves of noise only. On "Medium", use 4 octaves of noise but update it infrequently or at lower resolution. On "High", 4+ octaves animated in real-time (maybe even a compute shader simulating a cellular automata for granules if ambitious).
- *Field Lines*: On Low, draw only a handful of field lines or none at all (just rely on volume rendering of |B| for example). Or use simpler integration (Euler) with big steps (which might make lines kinked but it's low quality anyway). On Medium, more lines with RK4 but maybe truncated length. On High/Ultra, many field lines covering active regions and polar areas, with full RK4 and smooth curves. Also, high quality can increase line rendering quality (tessellated tubes instead of thin lines, for example, or more sample points per line for smoother curvature).
- *Corona Raymarch*: On Low, skip volumetric marching and instead render a billboarding sprite for the corona (like a simple textured halo as was done in some three.js demos [51]). On Medium, do raymarch with a coarser step size (or lower resolution render target upscaled). On High, fine step raymarch with adaptive stepping and maybe even secondary effects like anisotropic scattering (if we ever simulate polarization or so). Also possibly increase the maximum radius of the simulated corona at high quality (so you see farther streamers).
- *Particles*: On Low, fewer particles and no collisions. On Medium, moderate particles. On High, high particle counts with maybe compute-based collision avoidance or more sub-stepping for physics. Perhaps even GPU compute fluid effects on Ultra for things like a more continuous solar wind flow.

- *Post-processing*: On Low, minimal bloom (or none) to save on the extra blur passes. On High, full HDR bloom with many samples, maybe depth-of-field if camera is close to surface, etc.

Also include a note about memory: target GPU ~8 GB VRAM – keep volume textures reasonable (e.g. a 256^3 or 512^3 grid for corona if precomputing values). Ensure any 3D textures (noise or density) are clamped in size. Use half precision floats (fp16) in buffers if possible to save memory and bandwidth on medium settings. We might also mention using **LOD for 3D noise**: e.g., John Whigham used a 64^3 noise texture for star detail [52] which was enough and presumably smaller than generating per-pixel noise in shader. Claude could similarly use a small tileable 3D noise texture for corona density modulations or for surface detail, rather than heavy math in shader.

By providing these detailed techniques and references to algorithms, the Claude Code prompt will enable a comprehensive, physically-inspired real-time star renderer. Each item above gives Claude explicit guidance (equations, algorithm names, data values) that it can use to search for further documentation or implement directly, rather than vague instructions like "make it look cool." The combination of physically-based models and approximate tricks ensures the simulator can be both realistic and performant across different hardware tiers.

Sources: Procedural star graphics techniques [53] [9]; limb darkening model [5]; sunspot noise algorithm [10]; faculae contrast near limb [12]; corona volumetric rendering [15]; PFSS and field visualization [42] [39]; multi-wavelength details [24]; H-alpha feature descriptions [33] [35]; reconnection and CME descriptions [44] [47].

---

[1] [4] [8] [53] Procedural Star Rendering news - Seed of Andromeda - IndieDB
https://www.indiedb.com/games/seed-of-andromeda/news/procedural-star-rendering

[2] [3] [9] [10] [51] Procedural star rendering with three.js and WebGL shaders – Ben Podgursky
https://bpodgursky.com/2017/02/01/procedural-star-rendering-with-three-js-and-webgl-shaders/

[5] [6] Limb darkening - Wikipedia
https://en.wikipedia.org/wiki/Limb_darkening

[7] [21] [22] astronomy people, what can i do to make this render more realistic? : r/Astronomy
https://www.reddit.com/r/Astronomy/comments/1mkb6mu/astronomy_people_what_can_i_do_to_make_this/

[11] A Solar Observing Refresher Course - Sky & Telescope
https://skyandtelescope.org/observing/celestial-objects-to-watch/a-solar-observing-refresher-course/

[12] [13] [14] Solar faculae | Research Starters | EBSCO Research
https://www.ebsco.com/research-starters/astronomy-and-astrophysics/solar-faculae

[15] [49] [50] Ray marching - Wikipedia
https://en.wikipedia.org/wiki/Ray_marching

[16] Sky Renderer by Peter Kutz: Adaptive Step Size
http://skyrenderer.blogspot.com/2012/11/adaptive-step-size.html

[17] [18] Chapter 5
https://science.gsfc.nasa.gov/671/staff/bios/cs/Nelson_Reginald/Chapter5.pdf

19 23 47 48 NASA SVS | Carrington Class Coronal Mass Ejection - ENLIL Simulation of A Series of CMEs
https://svs.gsfc.nasa.gov/5375/

20 52 John Whigham's Blog: Solar Power
http://johnwhigham.blogspot.com/2011/09/solar-power.html

24 31 AIA - Atmospheric Imaging Assembly "First Light" images and movies
https://aia.lmsal.com/public/firstlight.html

25 SDO First Light - NASA SVS | Search
https://svs.gsfc.nasa.gov/search/?series=SDO%20First%20Light

26 Recent Views of the Sun
https://smithsonian-eclipse-app.simulationcurriculum.com/recent-views-of-the-sun.html

27 28 30 37 38 Wavelengths of Light - The Sun Today with Dr. C. Alex Young
https://www.thesuntoday.org/sun/wavelengths/

29 Atmospheric Imaging Assembly - AIA - lmsal
https://aia.lmsal.com/public/CSR.htm

32 XRT Picture of the Week (XPOW) - Hinode/XRT
https://xrt.cfa.harvard.edu/xpow/20250701.html

33 34 35 36 4. What are hydrogen alpha and calcium filters? - Solar Section
https://www.popastro.com/solar/solar-observing-guide/4-what-are-hydrogen-alpha-and-calcium-filters/

39 [PDF] 2D Vector Field Visualization
https://www.sci.utah.edu/~cscheid/scivis_fall07/2d_vector_vis.pdf

40 [PDF] Vector Field Visualization: Introduction
https://www2.cs.uh.edu/~chengu/Teaching/Fall2012/Lecs/Lec9.pdf

41 Comparison of Synoptic Maps and PFSS Solutions for The Declining ...
https://agupubs.onlinelibrary.wiley.com/doi/full/10.1029/2020JA028870

42 technical.dvi
https://vcg.iwr.uni-heidelberg.de/static/publications/machado_solarVis_technical_addendum.pdf

43 Introduction to the viridis color maps - CRAN
https://cran.r-project.org/web/packages/viridis/vignettes/intro-to-viridis.html

44 NASA SVS | Magnetic Reconnection 2
https://svs.gsfc.nasa.gov/20101/

45 On the variations of the solar wind magnetic field about the Parker ...
https://agupubs.onlinelibrary.wiley.com/doi/full/10.1029/2009JA015040

46 On the Determination of the Speed of a Fast Solar Wind Stream ...
https://www.mathematicsgroup.com/articles/AMP-8-244.php