ChatGPT

Building a Star Simulator: Data Sources, Presets, and Code References

Real Solar Data Sources

NASA Solar Dynamics Observatory (SDO) data – SDO provides high-resolution solar images and magnetograms in FITS format. For example, the HMI (Helioseismic and Magnetic Imager) produces full-disk magnetograms at 4096×4096 resolution (0.5″/pixel) every 45 seconds [1] [2]. AIA (Atmospheric Imaging Assembly) captures the Sun in **7 EUV channels** plus UV/visible filters at 4096×4096 resolution (0.6″/pixel) with **12-second cadence** [2] [3]. These datasets can be accessed via the **JSOC** (Joint Science Operations Center) or through APIs. **Helioviewer API** offers a convenient REST interface to retrieve images (as JPEG2000 or PNG) by specifying observatory/instrument/wavelength and time [4] [5]. For instance, one can request an AIA 171Å image or an HMI magnetogram closest to a given timestamp via Helioviewer (sourceId 10 for AIA 171, 19 for HMI magnetogram) [4] [5]. These APIs return compressed images or even video streams, making it feasible to import real solar frames into the simulator.

- **SDO/HMI Magnetograms:** Line-of-sight photospheric magnetic field maps in FITS; cadence 45s for LOS field (and 720s for vector field) [6]. Each 4096×4096 FITS contains a snapshot of the radial field (white = north polarity, black = south) [7]. The field strengths are derived from Fe I 6173Å line splitting [7]. *Data access:* HMI data can be downloaded via JSOC (e.g. series *hmi.M_45s* for 45s magnetograms). SunPy's Fido API can search the VSO/JSOC for HMI data and download FITS files in Python [8]. Helioviewer can also provide magnetogram images (sourceId=19) as JPEG2000 streams [5] [9].

- **SDO/AIA EUV Images:** Multi-wavelength EUV images at 10 wavelengths (e.g. 171Å, 193Å, 304Å, etc.) covering chromosphere and corona [3] [10]. Each channel highlights plasma at characteristic temperatures (e.g. AIA 171Å shows ~0.6 MK corona, 304Å shows ~50,000 K chromosphere) [11] [10]. *Data access:* AIA Level-1 FITS are available via JSOC/VSO and can be fetched by SunPy. The Helioviewer API provides quick JPEG2000 images by specifying instrument "AIA" and wavelength (e.g. 304) [4]. The AIA images are 4096^2 at 12 s cadence [2], but Helioviewer can down-sample or serve frame sets for movies. Use SunPy's `Fido` to query by time and wavelength and retrieve FITS files [8].

- **SOHO/LASCO Coronagraphs:** The SOHO spacecraft's LASCO C2 and C3 provide white-light corona images (covering ~2–30 solar radii). LASCO C2 images are ~1024×1024 with ~23″/pixel resolution [12] (C3 has ~112″/pixel) [13]. These show the solar corona and CMEs as seen in scattered light. *Data access:* LASCO data (FITS or JPEG) can be fetched via NASA archives or Helioviewer (sourceId=4 for LASCO C2, 5 for C3) [14]. Helioviewer can return recent LASCO frames in convenient image format.

- **Sunspot Number and Cycle Data:** The **SILSO** database provides daily, monthly, and yearly sunspot numbers (International Sunspot Number). These are available as text or CSV files (e.g. daily values from 1818-present) on the SILSO website [15] [16]. *Data usage:* The sunspot number time series (e.g.

monthly mean sunspot number) can be used to set the simulator's "cycle phase" or activity level. These data are open (CC BY-NC license for SILSO) [17]. For integration, one might ingest a CSV of monthly sunspot numbers and interpolate the solar activity level (e.g. more sunspots ⇒ stronger magnetic field visuals, more flares in simulation).

- **FITS Format and Parsing:** Most solar mission data comes in FITS files (Flexible Image Transport System). FITS files contain metadata headers and multi-dimensional image or spectral data. Programmatically, FITS can be parsed in Python via Astropy or SunPy (`sunpy.map.Map` reads a FITS into a solar metadata-aware object). In Rust, one can use libraries like `fitrs` or `fitsio` (crates for FITS I/O) to read the binary data [18]. The simulator should be able to load a FITS magnetogram or EUV image, which entails reading the 2D array of intensities. **Astropy**'s documentation (Python) is a good reference for the FITS format structure, and the FITS standard is public [19] [20]. Key point: ensure endianness and data types are handled (e.g. 16-bit or 32-bit images, FITS header gives `BITPIX` and array dimensions [19]). If real data import is optional, providing a utility to load FITS into a texture will enable users to overlay real magnetogram or emission images onto the simulated star.

Stellar Preset Data

We compile **preset parameter sets** for a variety of star types. Each preset includes fundamental parameters (mass, radius, effective temperature, luminosity, etc.) as well as surface features like rotation, magnetism, limb darkening, and granulation scale. Below are six star examples with their representative properties:

- **Sun (G2V)** – The Sun serves as the baseline. Mass ≈ 1.0 $M_\odot$ and radius ≈ 1.0 $R_\odot$ by definition. Effective temperature ~5778 K and bolometric luminosity ~1 $L_\odot$ [21]. It rotates differentially: about **25.7 days at the equator** and ~33 days near poles [22] [23]. The Sun's surface magnetic field averages ~**1 Gauss** in quiet regions [24], with active sunspots up to 3000 G (not continuous, but localized). Activity level follows an 11-year cycle (sunspot cycle) – from quiet minima to active maxima with numerous spots and flares. **Limb darkening:** The Sun shows strong limb darkening in visible light; a typical linear limb-darkening coefficient is ~0.65 in the optical band [25]. For more accuracy, two-parameter limb darkening laws (quadratic, etc.) from literature can be used – e.g. solar quadratic coefficients (~a=0.47, b=0.28 in some filters). Granulation on the Sun is characterized by convection cells ~1,000 km in size on the photosphere. Supergranulation (~30,000 km cells) and other scales exist, but for visualization one might depict a mottled texture of ~1 Mm cells to represent granules.

- **Proxima Centauri (M5.5V)** – A **red dwarf** star, much smaller and cooler than the Sun, and very magnetically active. Mass ≈ 0.122 $M_\odot$, radius ≈ 0.154 $R_\odot$ [26]. Teff ~3000 K (spectral type M5.5 V) [27] [28], giving it a dim red output (L ~0.0016 $L_\odot$ bolometric) [29]. Luminosity is extremely low (~0.2% of the Sun's in total, and only 0.00005 $L_\odot$ in visible light) [29]. **Rotation:** Proxima rotates slowly with a period ~83 days [27] (observations range ~83–89 days, indicating a slow spin for such a small star). **Magnetic field:** Despite slow rotation, Proxima's fully convective interior sustains a strong magnetic dynamo. Zeeman measurements indicate a mean surface field on the order of **600 Gauss** (measured magnetic flux 0.2–0.3 kG) [30]. As a result, Proxima is a flare star – its **activity level is high**, with frequent flares and a possible 7-year cycle in long-term brightness [31]. **Limb darkening:** Not well measured, but as a cool dwarf, models suggest pronounced limb darkening in infrared bands and less in optical. The simulator can use M-dwarf limb darkening coefficients (e.g. from Claret tables) for a ~3000 K, log g ~5.2 star. **Granulation:** Being fully convective, Proxima's "granulation" might be exotic – convection could encompass the whole star. The surface might display smaller, short-lived convective cell patterns or starspots. We could assume granulation cells perhaps ~100 km (very small) or unresolved, but huge starspots are observed on some M dwarfs. In visualization, one might simply show an overly active, flaring surface with random bright flares and spots rather than resolved granules.

- **Alpha Centauri A (G2V)** – A solar twin for comparison. Mass ≈ 1.08–1.10 M⊙ [32], radius ≈ 1.22 R⊙ [33]. Teff ~5804 K, very close to Sun [21]. Luminosity ~**1.5 L⊙** (about 50% more luminous than Sun) [21]. **Rotation and activity:** Alpha Cen A's rotation period is similar to the Sun's – roughly **~22–25 days** (estimated to be about the same as solar) [34]. It has a magnetic activity cycle: observations showed Sun-like coronal X-ray variability. Notably, it entered a low-activity phase after 2005, possibly analogous to a Maunder Minimum [35]. Overall, its magnetic field and cycle are comparable to the Sun's, though currently subdued [35]. **Magnetic field:** likely a few Gauss average; not directly measured, but no strong fields like an Ap star. **Limb darkening:** virtually the same as Sun (G2V star) – use solar limb darkening coefficients. **Granulation:** very similar to Sun's granulation (~1,000 km cells). As a star only slightly larger and more massive, convection cell size and pattern should be Sun-like. This preset lets the simulator show a Sun-like star that can be contrasted in activity (e.g. if Sun is active, Alpha Cen A might be shown in a quieter state to emulate its Maunder minimum-like epoch [35]).

- **Betelgeuse (M1–M2 Ia-ab)** – A **red supergiant**, representing the extreme end of cool, huge stars. We adopt radius ~**700 R⊙** (on the order of 5–6 AU; Betelgeuse's radius estimates range 640–764 R⊙ [36]). Mass ~**15 M⊙** (initial mass ~20 M⊙, current mass may be 11–19 M⊙ after mass loss) [36] [37]. Effective temperature ~**3600 K** [38], giving it a red-orange hue. Luminosity on the order of **10^5 L⊙** (Betelgeuse is about 65,000–100,000 times the Sun's luminosity) [39] [40]. **Rotation:** Very slow – Betelgeuse's observed rotation period is ~**36 years**(!) [41]. Its v*sin(i) ~5.5 km/s* [42]*, meaning an extremely leisurely spin due to its enormous size (it likely rotated faster as a young star and has slowed after expanding). Magnetic field: Betelgeuse has a weak, patchy magnetic field detected on its surface (on the order of <1 G, a "weak dynamo")* [43]*. As a cool supergiant, it doesn't have a strong global field like the Sun; instead, giant convective cells may locally amplify magnetism. Activity: Not a solar-type cycle with flares, but Betelgeuse shows semi-regular pulsations and the famous occasional dimming events (e.g. the Great Dimming of 2019–2020 was due to a dust eruption). It's classified as a semi-variable star (irregular pulsations with a ~420-day period and a longer ~5–6 year secondary variation)* [41]*. Limb darkening: Very pronounced and complex – Betelgeuse has no sharp "edge." A spherical limb darkening model is needed; the optical limb is diffuse due to its extended atmosphere. For simulation, one might render a fuzzy limb with brightness dropping gradually (and perhaps asymmetric bright spots). Granulation: Betelgeuse's convective cells are colossal – interferometry suggests only a few huge cells cover the surface, each cell a significant fraction (50–60%) of the star's radius* [44] [45]*. In other words, granules the size of entire stars**! The preset should reflect this by showing a very coarse convective pattern (e.g. 2–3 giant bright regions on the star). Betelgeuse is a prime example for demonstrating large-scale surface dynamics and heavy mass loss (you might include a semi-transparent halo or outflow).

- **Rigel (B8 Ia)** – A **blue supergiant** star, hot and massive. Mass ≈ **21 M⊙** (estimated 18–24 M⊙ range) [46]. Radius ~**70–78 R⊙** [47] [48] – despite being much hotter, it's still dozens of times the Sun's size. Effective temperature ~**12,000 K** (B8 spectral class, ~12,100 K noted) [49]. Bolometric luminosity ~**120,000 L⊙** [48] (Rigel is extremely luminous; in visible light it's ~47,000 L⊙ [50], but including UV output it reaches ~1.2×10^5 L⊙). **Rotation:** Blue supergiants can have moderate rotations, but after expansion from the main sequence they tend to slow. Rigel's projected rotation is not very high; if we assume v ~ a few tens of km/s, the rotation period might be on the order of **weeks**. (No specific period is easily observed; as an *α Cygni* variable, it has small pulsations instead of obvious rotational modulations.) The simulator can leave Rigel's rotation at a generic slow value (e.g. 1–2 months) unless better data is found. **Magnetic field:** Most B-type supergiants do *not* have strong magnetic fields unless they are of the rare magnetic subclass. Rigel is not known to have a significant field; any surface field would be weak (< a few Gauss) and "fossil" in nature. We assume essentially **non-magnetic** for visual purposes (no starspots or magnetically confined plasma loops). **Activity:** Rigel exhibits slight **non-radial pulsations** (Alpha Cygni variability) causing light to vary by a few hundredths of a magnitude [51]. It also drives a steady **stellar wind** (mass loss ~10^-7 M⊙/yr) [52] [53] – in visualization, a faint blue glow or haze flowing outward could represent the wind. No flares or corona (hot stars have weak coronae, but strong radiation-driven winds). **Limb darkening:** Hot stars have less dramatic limb darkening in the optical, but still present. Use a limb darkening law for Teff ~12k K, log g ~2 (supergiant gravity ~10^2–10^3 cm/s^2). Typically, limb darkening coefficient might be ~0.3–0.4 in visible for a hot supergiant (the simulator can use a model atmosphere table). **Granulation:** Hot blue supergiants have predominantly radiative envelopes (no deep convective surface zones), so classical granulation is absent. Instead, any surface pattern might come from **subsurface convection** (due to iron opacity bump) which can create small bright spots or micro-turbulence. For practical purposes, Rigel's surface can be shown as mostly uniform or with very subtle large-scale variation (plus maybe slight pulsation-based breathing in brightness). The key visual will be its color (blue-white) and maybe an animated shimmering to indicate instability.

- **Sirius A (A1V)** – A **bright A-type main sequence** star. Mass ≈ **2.06 M⊙**, Radius ≈ **1.71 R⊙** [54], Teff ≈ **9900 K** (around 10,000 K) [54], Luminosity ≈ **25 L⊙** (visibly bright; 24.7 L⊙ reported) [54]. Sirius A is the standard for a hot main-sequence star with no strong convective envelope. **Rotation:** It spins fairly rapidly but not extreme: projected v ≈ 16 km/s [55]. Given its inclination, the true rotation period is on the order of **5–6 days** (Sirius's equatorial rotation likely a few days). This is **much faster than the Sun**, but typical for an A-type star. **Magnetic field:** A-type stars generally have very weak magnetic fields unless they are Ap chemically peculiar stars (which Sirius is not). In fact, a *weak field of ~0.2 G* has been detected on Sirius [56] – essentially negligible for dynamics. So **no significant magnetic activity** (no starspots or flares; A-stars don't have strong coronae because they lack a convective dynamo). **Activity level:** Very low – Sirius has a stable photosphere, with no flare events. It does not have an 11-year cycle or similar; any variability is minimal (except for the binary companion's effects). **Limb darkening:** As a 10,000 K star with log g ~4.3, limb darkening is present but moderate. Use coefficients appropriate for Teff=10000 K, log g ~4 – roughly linear limb-darkening u ≈ 0.4–0.5 in visible. **Granulation:** There is a very shallow convective layer for A1V (near the surface for hydrogen/helium ionization), but effectively, *granulation is insignificant visually*. The star's surface would appear featureless to the eye. The simulator can render Sirius A as a clean blue-white sphere with limb darkening and perhaps gentle rotation. (One might optionally include a very subtle granulation texture or very small amplitude flickering to hint at subsurface convection or pulsation, but it's not prominent like on Sun.) If desired, add a faint glow to indicate it's a bright star, and remember Sirius is part of a binary – though the companion (Sirius B) is a white dwarf not directly visible in a normal rendering except as a point source nearby.

Each preset above can be encoded in a JSON format. For example, a preset could be a JSON object like:

```
{
  "name": "Sun",
  "spectralType": "G2V",
  "mass": 1.0,
  "radius": 1.0,
  "temperature": 5778,
  "luminosity": 1.0,
  "rotationPeriodDays": 25.7,
  "magFieldAverage": 1.0,
  "activityIndex": 1.0,
  "limbDarkening": {"coeffType": "quadratic", "u1": 0.47, "u2": 0.28},
  "granuleSizeKm": 1000
}
```

And similarly for others (with appropriate values and notes). The **activityIndex** could be a relative measure (Sun = 1 at solar maximum, Proxima might be 5 for very active, Sirius ~0 for no activity). Magnetic field could be a representative surface field strength in Gauss (or a dipole moment if a model is used). These JSON presets will allow Claude (and the code) to easily switch between star configurations.

Existing Open-Source Code Resources

Building a star simulator will benefit from studying existing **codebases and libraries** in solar physics, stellar visualization, and general graphics. Below is a curated list of open-source projects and specific components to guide Claude in implementation. *We emphasize algorithmic understanding over direct code use* (especially for GPL-licensed sources) – Claude should extract the methods and reimplement in Rust, ensuring compatibility with MIT/Apache licensing.

- **SunPy (Python, BSD-2 license)** – A rich library for solar data analysis [8]. *What to use:* SunPy's data retrieval and map objects. For example, `sunpy.map.Map` handles solar images with world-coordinate information (mapping pixel coordinates to heliographic latitude/longitude) – this can guide how to project spherical surface data [8]. The `sunpy.visualization.colormaps` module includes official SDO/AIA colormaps for different wavelengths (these can be ported so the simulator colors EUV images in the familiar way). SunPy's Fido client shows how to search and download data from VSO/JSOC [8]. Also, SunPy's sample code for plotting a solar map with limb and grid can inform how to overlay coordinate grids on a sphere [57]. While SunPy is Python, its well-documented functions (e.g. solar coordinate transformations, image rescaling) can be translated to Rust logic.

- **JHelioviewer (Java, GPL)** – A desktop application for browsing solar images. It leverages JPEG2000 data from Helioviewer. *Useful aspects:* Its **streaming image architecture** and timeline interface for layering multiple data sources. The code (in Java) is GPL, so we won't copy it, but we can replicate its ideas: e.g. the way it requests image tiles from a server and composites multi-wavelength overlays. JHelioviewer also implements fast zoom/pan on 4k solar images – likely using hardware acceleration or tile pyramids. Claude can inspect JHelioviewer's documentation or developer notes to see how they manage performance with large images and implement coordinate mappings. (Another clue:

Helioviewer's **data source IDs** as given in the API docs [4] [5] could be integrated so the simulator can fetch real data frames by ID and time.)

- **SolarSoft (IDL, various licenses)** – NASA's solar software library in IDL, not strictly open-source, but widely used in the community. It contains many routines for image calibration (AIA prep), field line tracing (PFSS – potential field source surface models), etc. We cannot use IDL code directly, but documentation from SolarSoft can help. For example, SolarSoft's PFSS package describes how to trace magnetic field lines from a photospheric magnetogram into the corona (potential field extrapolation) – this could be useful if the simulator includes magnetic fieldline visualization. The algorithms (e.g. source surface radius, spherical harmonic decomposition) are described in scientific literature, which Claude can rederive. In summary, SolarSoft is a reference for *physics algorithms* (not to implement verbatim, but to ensure our simulator's math is correct). We note that IDL code is not easily accessible to Claude, so focus on concepts gleaned from SolarSoft user guides.

- **yt Project (Python, BSD)** – yt is a general volumetric data analysis and visualization library (often used for astrophysical simulations) that supports interactive slicing, volumetric rendering, and streamlines. *Applicability:* If our star simulator will incorporate 3D data (say a 3D MHD simulation cube of a convection zone or corona), yt's approach to reading data and computing derived quantities is instructive. For instance, yt can load a 3D grid of vector fields and generate **streamlines (field lines)** through them. The algorithms for streamline integration (Runge-Kutta integration through a vector field) are available in yt (Python) and can guide how to implement similar in Rust. Additionally, if we create a 3D grid for a star (for a magnetic field or density), we might use yt's data structures as inspiration for how to index and query the grid in code. yt is well-documented and Python code is permissively licensed, so Claude can safely draw from it.

- **ParaView and VTK (C++, BSD)** – The Visualization Toolkit (VTK) underlies ParaView, a popular scientific visualization tool. *Relevant features:* VTK includes algorithms for **streamlines, isosurfaces, and volume rendering**. ParaView has been used for solar data (e.g. visualizing coronal mass ejection simulations). We can look at how VTK implements streamline integration (e.g. the class `vtkStreamTracer`) – it uses integrators like Runge-Kutta 4 for vector fields. Understanding the integration step size control and termination conditions (like when field line leaves domain) is key [58]. VTK is large, but we might specifically check its **sun-specific modules** if any (sometimes, NASA and community contributed code for reading solar data in VTK). Also, VTK's **volume rendering** of scalars with GPU shaders could inform how to render a glowing corona or star interior. For example, a procedural volumetric shader could be used to render the hot corona around the Sun – VTK/ParaView code or shaders might illustrate how to do emission and absorption in a volume. Since VTK is permissive (BSD), Claude can directly read and adapt portions of it, as long as it's not too heavyweight.

- **Unity/Unreal Engine Star Rendering** – While game engine assets are not all open-source, there are community tutorials and code snippets for rendering stars. For example, tutorials on procedural suns in Unity (C# + HLSL shaders) show techniques for an animated surface and **post-processing like lens flares** [59] [60]. One common approach is using **noise textures** scrolling to simulate boiling motion and using a fresnel effect for the limb glow. Claude can glean ideas from Unity shader graphs: e.g. using Perlin noise to modulate brightness and color, or using a **blackbody color lookup** to make the star color temperature-accurate (some Unity shaders map temperature to RGB). There's also the concept of **bump mapping** to fake granulation or starspots on the surface. Unreal Engine's materials for stars (often seen in space games) sometimes layer multiple noise octaves for a turbulent look. While not one specific repository, searching GitHub for terms like “procedural star shader” or reading blog posts (such as “Reaching for the Stars | Unity URP Shader Graph” by Jannik Boysen) can provide concrete shader code that can be ported to Rust’s shader (WGSL or GLSL). Shaders are typically small and often posted publicly (and can be considered factual/algorithmic descriptions), so Claude can safely incorporate similar noise functions or blending techniques.

- **GPU-accelerated Field Line Tracing:** To visualize magnetic loops or stellar magnetospheres, we may need to trace field lines efficiently. An example of open-source code in this domain is Marek Fiser’s CUDA-based streamline integrator [58] [61]. The project demonstrates RK4 integration on the GPU to draw streamlines and streamtubes in real-time, using a 3D vector field texture for lookups [58]. The source is on GitHub [62]. Studying this code will show how to structure the computation: e.g. launching many threads, each integrating a different seed point’s field line, and storing the vertices. We can mimic this in Rust using compute shaders (via wgpu or OpenGL) – effectively implementing RK4 in a shader over a 3D texture representing the field. *Important details:* step-size selection (fixed vs adaptive), interpolation of field (likely trilinear from grid points), and termination criteria (stop after X length or when field magnitude drops). By examining Fiser’s implementation, Claude can translate it into Rust GPU code (WGSL) within the wgpu framework. This will drastically speed up field line rendering for possibly thousands of lines (important for an interactive simulator).

- **Rust Graphics Libraries:** Since the simulator will be written in Rust, any existing Rust crates for rendering can jump-start development. Notably, **Bevy** (Rust game engine) or **wgpu** examples. For instance, `bevy_sprite` or `bevy_shader_utils` might have relevant examples of using shaders for particle effects (which we can repurpose for, say, flares). The **wgpu-rs repository** and its examples include how to feed uniform buffers (which can be our star parameters) and textures (for data like magnetograms or noise maps) to shaders. While there’s no solar-specific Rust code published, general graphics techniques (billboards, particle systems) in Rust can be pulled from existing code. Also, **rust-sdl2** or **glium** could be options for 2D overlays if needed (though likely wgpu covers all we need). Keep in mind to use Rust-native libraries where possible (for windowing, input, etc., consider egui or winit for UI overlays like time sliders).

- **Shader libraries (GLSL/HLSL)** – Many open repositories of shaders exist. For star rendering, two aspects are particularly useful: **noise functions** and **color mapping**. We can consult open noise implementations (Perlin, Simplex noise – there are public domain versions of these algorithms) to use in fragment shaders for procedural textures. Also, for mapping temperature to color, we might use an existing formula (there are known approximations for blackbody RGB values; some shader code implements this via piecewise approximation or via a LUT texture). Claude can embed a small precomputed color table (for 2000K to 20000K, for example) or use the known analytic fits [63] [47]. Since these are essentially data or formulas, it’s safe to use.

When incorporating code from these sources, remember the **licensing**: permissive-licensed code (MIT, BSD, Apache) can be adapted freely, but GPL code should **only inform** your approach (clean-room reimplementation). For example, reading SunPy’s documentation or VTK’s algorithm description is fine, but do not copy any GPL-licensed source code line-by-line. Instead, extract the logic (e.g. “SunPy does X to convert heliographic to image coordinate – okay, implement that math fresh in Rust”). We have identified above which sources are permissive. For GPL ones (JHelioviewer, some Unity references if any), Claude should only use the conceptual knowledge.

Reference Visualizations and Techniques

To guide the visual style and ensure scientific accuracy, consider some reference visualizations:

- **NASA Scientific Visualization Studio (SVS) Solar Videos:** The SVS has produced numerous solar animations using SDO data. For example, “The Active Sun from SDO” series (ID 3989, 3990, etc.) shows the Sun in different wavelengths side-by-side [64] [65]. The HMI magnetogram visualization (SVS #3989) demonstrates how to colorize magnetic field images (white/black polarity) and sync them with EUV images [7]. Another SVS product shows a **slow rotation of the Sun in 4K detail** [66], illustrating the level of detail visible in 4096^2 images. By watching these, you can emulate camera motions (slow pans and rotations), and how transitions between wavelengths are presented (often as fades or wipes). SVS also has **corona field line visualizations** for the Sun – e.g. depictions of model magnetic loops during solar storms. These usually trace field lines from a model and animate along them. Observing those can inspire how to animate field lines in our simulator (perhaps drawing them gradually or highlighting ones that reconnect during flares). Techniques often used: *mixing real data with models* (e.g. overlaying a PFSS magnetic field extrapolation on an AIA image). Our simulator could allow a mode where a simple dipole field is drawn over the Sun image, etc.

- **SDO Data Processing Pipelines:** Understanding how raw data becomes the beautiful images is useful. For instance, AIA images are typically shown with an exposure normalization and a color table. The simulator should apply AIA’s color tables (available via SunPy or SDO website) so that, say, 171 Å is gold, 304 Å is orange, etc., matching real-life visuals [3]. The pipeline also involves gamma or log scaling of intensities to bring out features (the corona has high dynamic range). We may implement a similar **logarithmic brightness scaling** for EUV images to visualize both bright flares and faint corona simultaneously. The HMI magnetogram data may require smoothing or downsampling for visual clarity (4096^2 is very high resolution; SVS often down-samples to 1080p frames [65]). Knowing this, our simulator can offer detail vs performance settings (e.g. use full 4k texture or a half-size). The **Helioviewer** project itself is a reference for pipeline: they convert FITS to JPEG2000 with some preprocessing (flat-field, etc.). While we probably won’t implement full calibration, using high-level processed data (Level 1.5 AIA, HMI) ensures the images are ready to use (no need to reinvent calibration).

- **Academic Papers (Visualization Methodology):** There are scientific publications on visualizing complex stellar data. For example, papers on stellar surface convection simulations often include rendered images of giant-cell convection on supergiants. They describe techniques to represent intensity from simulation output, sometimes adding limb darkening or radiative transfer effects. Another example: papers on **magnetic field topology** might visualize field lines and use color/width to denote field strength or twist. We should search for terms like “solar visualization methodology” or “3D MHD visualization” in journals. These can provide cues like: using semi-transparent volume rendering for coronal density (to show a glowing corona), or using particle tracers to illustrate solar wind outflow. If the simulator aims to have a 3D mode for the corona or wind, these techniques are valuable. One specific reference: the **Predictive Science Inc.** team (PSI) that visualizes solar eclipse predictions often uses volumetric rendering of corona brightness; their imagery (often public on NASA’s pages) could inform how to render a realistic corona given a density model.

- **Visual Effects**: Not strictly scientific, but worth noting – to make the simulator visually compelling, we can implement a few effects inspired by reference visuals. For instance, **bloom/glow shaders** can mimic the way bright stars bloom on camera. NASA's eclipse corona visualizations have a glow around the Sun to show the corona extent. Lens flares or diffraction spikes (as seen in telescope images of bright stars) could be optional effects for realism. Many game engines implement these, and Claude can adapt from there.

In summary, the references above should guide both the scientific accuracy (ensuring values and scales are right) and the visual presentation (ensuring the simulator is engaging and reminiscent of actual observations). Combining real data capabilities with presets will allow users to compare, say, the Sun's actual magnetogram to the simulator's procedural active regions on a different star – a powerful feature for education and analysis.

Data and Licensing Considerations

Finally, a note on **data licensing and code usage**:

- **Solar data licensing:** Most solar data from NASA (SDO, SOHO, etc.) and ESA is public domain or publicly available. For example, SDO data is free for use; SILSO sunspot data is provided under CC BY-NC (allowing free use with credit for non-commercial purposes) [16]. Since our project will be open-source (MIT/Apache), including such data or allowing users to download it is fine. Just ensure we attribute sources where required (e.g. give credit to SILSO for sunspot numbers [16], and NASA/SDO for imagery).

- **Code licensing:** We prefer **permissive licenses** for any included code. That means leveraging MIT/BSD/Apache licensed libraries and references. If we reference GPL code (like JHelioviewer or some SunPy subcomponents), we must not incorporate their code directly – only the ideas. Claude will perform a *clean-room reimplementation* of any algorithms learned from GPL sources, writing all code from scratch in Rust. This avoids any contamination. All original code Claude writes will be under MIT/Apache (double-licensing possible), which is compatible with including parts of MIT or BSD licensed code from references. We will **avoid any non-commercial or proprietary licensed code** entirely. For instance, if some great visualization code was only in MATLAB or under a restrictive license, we won't use it; we'll find an alternative in literature.

- **Use of Python/C++ resources in a Rust project:** This is encouraged for algorithmic insight. The solar community's knowledge is largely in Python (SunPy, Astropy) and older languages. We will take advantage of that by reading those implementations and then replicating the functionality in Rust. For example, Python's Astropy has a routine to calculate limb darkening coefficients from model atmospheres – we could call Astropy at runtime via FFI, but better to translate the formula into Rust so our simulator is standalone. Another example: if no Rust FITS reader was available, we might have needed to bind to CFITSIO (C library), but since Rust has `fitsio` crate, we can stick to pure Rust.

- **Data integration formats:** We should define how external data feeds into the simulator. Possibly use **JSON or TOML config** for presets, and maybe support reading certain data file formats (FITS for images, CSV for time series). All these are open formats. We might include small sample data (like a sample magnetogram FITS of the Sun) in the repo for testing – those should be chosen from public datasets. The code should clearly note the data source and any credit (maybe in a README or an "About" dialog in the app).

In conclusion, Claude Code will be directed to use the above sources to construct the star simulator. We've gathered the necessary parameters for star presets, indicated how to fetch and parse real observations, identified relevant algorithms (coordinate mapping, field line tracing, procedural texturing), and pointed to reference implementations. With this information, Claude should be able to formulate a comprehensive prompt and eventually produce a robust, accurate star simulation tool.

Sources:

- Real solar data specifics from NASA/SDO and Helioviewer [2] [4]
- Sun and Alpha Centauri A parameters [21] [33]
- Proxima Centauri parameters and activity [29] [27]
- Betelgeuse parameters (mass, radius, convective cells) [36] [44]
- Rigel parameters (mass, radius, luminosity) [48] [47]
- Sirius A parameters (mass, radius, lum, rotation, magnetic field) [54] [56]
- Sun's magnetic field and rotation [24] [22]
- Limb darkening data for Sun (by analogy for others) [25]
- Helioviewer API documentation [4] [5]
- SunPy usage for data retrieval [8]
- NASA SVS description of HMI and multi-wavelength visualization [7] [65]
- CUDA field line tracer project by Fiser [58] [61]

---

[1] The SoHO/MDI, and SDO/HMI Line-of-sight Magnetogram Dataset
https://colab.research.google.com/github/spaceml-org/helionb-mag/blob/main/notebooks/01_los_mag_dataset_2019/mdi-hmi_los_mag_dataset_colab.ipynb

[2] [3] [10] [11] Solar Dynamics Observatory - Wikipedia
https://en.wikipedia.org/wiki/Solar_Dynamics_Observatory

[4] [5] [9] [14] [19] [20] Helioviewer API Documentation
https://api.helioviewer.org/docs/v1/

[6] Magnetic Fields Measured by HMI / SDO
http://hmi.stanford.edu/magnetic/

[7] [64] [65] NASA SVS | The Active Sun from SDO: HMI Magnetogram
https://svs.gsfc.nasa.gov/3989/

[8] [57] Acquiring Data — sunpy 8.0.dev35 documentation
https://docs.sunpy.org/en/latest/tutorial/acquiring_data/index.html

[12] [13] SOHO LASCO C2 Level 1.0 FITS Data - Dataset
https://data.nasa.gov/dataset/soho-lasco-c2-level-1-0-fits-data

[15] [16] [17] Sunspot Number | SIDC
https://www.sidc.be/SILSO/datafiles

[18] fitrs - Rust - Docs.rs
https://docs.rs/fitrs

[21] [32] [33] [35] Alpha Centauri - Wikipedia
https://en.wikipedia.org/wiki/Alpha_Centauri

[22] Solar rotation - Wikipedia
https://en.wikipedia.org/wiki/Solar_rotation

[23] The Solar Cycle | Astronomy - Lumen Learning
https://courses.lumenlearning.com/suny-astronomy/chapter/the-solar-cycle/

[24] How strong is Sun magnetic field on Moon surface? And on Mars?
https://space.stackexchange.com/questions/41246/how-strong-is-sun-magnetic-field-on-moon-surface-and-on-mars

[25] Limb-darkening coefficients of late-type stars - NASA ADS
https://adsabs.harvard.edu/full/1990A%26A...230..412C

[26] [27] [28] [29] [31] Proxima Centauri - Wikipedia
https://en.wikipedia.org/wiki/Proxima_Centauri

[30] The moderate magnetic field on the flare star Proxima Centauri
https://inspirehep.net/literature/793596

[34] Rigil Kentaurus - JIM KALER
http://stars.astro.illinois.edu/sow/rigil-kent.html

[36] [37] [38] [39] [40] [41] [42] [43] [44] [45] Betelgeuse - Wikipedia
https://en.wikipedia.org/wiki/Betelgeuse

[46] [48] [49] [51] [52] [53] Rigel - Wikipedia
https://en.wikipedia.org/wiki/Rigel

47 63 Rigel | Orion's Brightest Star
https://astrobackyard.com/rigel-star/

50 Rigel | Blue Supergiant, Orion Constellation & Binary System | Britannica
https://www.britannica.com/place/Rigel

54 55 Sirius - Wikipedia
https://en.wikipedia.org/wiki/Sirius

56 kitp.ucsb.edu
https://www.kitp.ucsb.edu/sites/default/files/preprints/2013/13-183.pdf

58 61 62 Realtime visualization of 3D vector fields using CUDA | Marek's blog
https://marekfiser.com/projects/realtime-visualization-of-3D-vector-fields-using-CUDA/

59 [Unity 3D Tutorial] Procedural Sun and Stars - Part 2 of 2 - YouTube
https://www.youtube.com/watch?v=eW1Twkztr0w

60 Procedural Star Rendering : r/gamedev - Reddit
https://www.reddit.com/r/gamedev/comments/33xuyj/procedural_star_rendering/

66 SDO 4k Slow-rotation Sun Resource Page - NASA SVS
https://svs.gsfc.nasa.gov/12613/