Torospherical Head Visualization Approach

A torospherical head is a **surface of revolution** — cross-section + rotation is the correct and standard technique. The 4-step breakdown is logical. One refinement worth making upfront:

The cross-section should be a closed 2D profile (not just the inner surface line), enclosing the material cross-section. This makes the 3D solid watertight and visualizable as a solid, not just a surface shell.

Cross-Section Components (preview for Step 1)

Working from the centreline apex downward, the closed profile has these geometric segments:

#	Segment	Geometry
1	**Apex flat**	Horizontal line of thickness `t` at the top (connects outer and inner crown at apex)
2	**Outer crown arc**	Spherical arc, radius = `R_crown + t`, same centre as inner
3	**Outer knuckle arc**	Toroidal arc, radius = `r_knuckle + t`, same torus-ring centre as inner
4	**Outer straight flange**	Vertical line at `r = D/2 + t`, height = `h_flange`
5	**Bottom rim**	Horizontal line connecting outer to inner at base
6	**Inner straight flange**	Vertical line at `r = D/2`, height = `h_flange`
7	**Inner knuckle arc**	Toroidal arc, radius = `r_knuckle`
8	**Inner crown arc**	Spherical arc, radius = `R_crown`

The key geometric constraint that ties everything together is the **tangency condition** between the crown sphere and knuckle torus. The crown sphere centre z_c is derived analytically from the inputs:

$$z_c = h + \sqrt{(R_{crown} - r_{knuckle})^2 - (D/2 - r_{knuckle})^2}$$

This also gives us the **tangency point** between segments 2↔3 and 7↔8.

Suggestions

1. **Add input validation** before any geometry — check constraints like $r_k < D/2$, $R_c \geq D/2 - r_k$, $r_k > t$ — and raise clear errors, since invalid combinations will produce degenerate or imaginary geometry.
2. **Parameterise mesh resolution** — add `n_meridional` (points along the profile) and `n_azimuthal` (angular steps around the axis) as optional parameters with sensible defaults.

3. **Separate geometry from visualisation** — keep the cross-section builder and the 3D mesh rotation as pure data functions, so the visualisation code (Step 4) only deals with plotting. This will also make it easy to export the geometry later.

4. **Interactive sliders** (Step 4 enhancement) — `matplotlib.widgets.Slider` can give real-time parameter adjustment, which is very useful for this kind of tool.

5. **Coordinate convention** — agree now: $z = 0$ at the bottom of the straight flange, z increasing toward the apex. $r = 0$ on the centreline axis.
