ChatGPT

Horn Geometry for the Everything Chalice

Overview

The “Everything Chalice” features a continuous, elegant horn structure spiraling vertically from top to bottom. Four horn geometry types can be considered for this design: **helical**, **logarithmic**, **tractrix**, and **exponential**. Each horn type is defined by a parametric centerline curve (the path the horn follows) and a varying radius profile (the *flare* of the horn). Unlike a simple surface of revolution, these horns are generated by sweeping a cross-section (e.g. a circle) along a 3D curve, allowing for more dynamic shapes. Below we define and compare these horn types – focusing on helical and logarithmic horns as the primary design basis – and provide their parametric equations. We also discuss parameter ranges that ensure smooth, continuous geometry, and how to implement the horn in a Three.js graphics pipeline. Finally, we include notes on constructing the mesh via `THREE.BufferGeometry` and exporting the result to **OBJ** or **GLTF** format.

Centerline Parametrics

Each horn’s centerline is described by a parametric equation in 3D space. This curve defines the path along which the horn will be swept. For the Everything Chalice, the horn’s centerline runs vertically down the chalice. We detail the centerline for each horn type:

- **Helical Centerline:** A helical horn’s path is a helix (corkscrew) winding around a central axis. A typical parametric form for a vertical helix is:

$$x(t) = R\cos t, \quad y(t) = R\sin t, \quad z(t) = \frac{P}{2\pi}\,t,$$

where R is the helix radius (distance from the central axis) and P is the vertical *pitch* (rise per 2π radians of angle) [1]. Here t (in radians) runs from 0 up to $N\cdot 2\pi$ for N turns of the helix. For example, $0 \le t \le 4\pi$ would produce two full turns. The pitch P can be tuned so that the helix spans the full vertical distance of the chalice. Ensuring N is an integer yields a smooth continuous spiral with no abrupt end orientation. A moderate number of turns (e.g. 2–4) often works well visually – enough to spiral elegantly, but not so many that the coils overlap too tightly.

- **Logarithmic Spiral Centerline:** A logarithmic horn follows a planar spiral that expands outward as it descends. In polar coordinates (r,φ), a logarithmic spiral is defined by $r(\varphi) = a\,e^{k\varphi}$ [2], where a controls the initial radius and k the growth rate (angle in radians). In Cartesian form, the 2D spiral can be parameterized as:

$$x(\varphi) = a\,e^{k\varphi}\cos\varphi, \qquad y(\varphi) = a\,e^{k\varphi}\sin\varphi,$$

with φ as the parameter [3]. To use this in a vertical 3D horn, we introduce a z-component that drops the spiral downward: for instance $z(\varphi) = -\,C\,\varphi$, which gives a gentle downward slope (here C relates vertical length to the number of turns). As φ increases, the curve spirals outwards (increasing r) while moving down. The parameter φ can run from 0 to a few full rotations (e.g. $\varphi_{\text{max}} = 4\pi$ for two turns). A small growth rate k (e.g. 0.1–0.5) yields a wide, smooth spiral without an excessively large radius at the bottom [4] [2]. By adjusting k and the vertical factor C, one can ensure the bottom end of the spiral reaches the desired radius and vertical position smoothly. This 3D logarithmic spiral provides a graceful, self-similar curve often seen in nature (e.g. nautilus shells and ram's horns) and ensures geometric continuity along the horn.

- **Tractrix Centerline:** A tractrix horn (from acoustical horn theory) is traditionally an axisymmetric shape, so its centerline is typically a straight line along the axis. For integration into the chalice (vertical orientation), we can take the centerline as the vertical z-axis itself. In parametric form: $x(t)=0,\;y(t)=0,\;z(t)=t$, where t runs from 0 at the horn mouth (top of chalice) to L at the throat (bottom) – or vice versa depending on orientation. Essentially, the horn does not spiral; it flares outward around this central axis. This straight centerline ensures the tractrix horn's symmetry is preserved. (In theory, one could impart a slight curve or tilt for artistic effect, but a straight line is the standard for tractrix and exponential horns.) The length L is typically chosen such that the horn flare is complete by the bottom (more on this under flare profile). Using the full vertical span of the chalice for L guarantees a continuous connection from top to bottom.

- **Exponential Centerline:** Like the tractrix, an exponential horn's centerline is usually straight along the axis of revolution. We parametrize it as $x(t)=0,\;y(t)=0,\;z(t)=t$ for $0\le t \le L$, with L the horn length. The horn will flare outward symmetrically around this axis. By aligning this line with the chalice's vertical axis, the horn extends vertically. The parameter L (end of the centerline) is set to the vertical height available for the horn. This yields a simple, continuous centerline for the exponential horn – essentially a vertical line segment – ensuring no bends or kinks in the path.

Parameter Range and Continuity: For each centerline, we choose parameter ranges that produce a smooth curve matching the chalice's dimensions. The helical and logarithmic spirals require specifying the number of turns or angular range to span the full height gracefully. Their parameters (angle t or φ) should start and end where the horn needs to attach to other geometry (e.g. starting at the top with a small radius and ending at the bottom with a large radius), avoiding any gaps. The tractrix and exponential centerlines use a linear parameter from 0 to L (the horn's length); L can be tuned so that the horn's mouth and throat meet the chalice's top and bottom exactly. All curves described are C^1 continuous (continuous in position and tangent), ensuring the horn's geometry will not have sharp corners. This smoothness is crucial when sweeping a mesh along the curve.

Flare Profile Equations

The flare profile specifies how the horn's radius (cross-sectional radius) changes along the centerline. It determines the horn's overall shape (narrow at one end, wide at the other). Below are the radius profiles for each horn type, given as functions of the path parameter. We also note suitable parameter ranges or constraints for each to maintain a realistic shape without discontinuities.

- **Helical Horn Flare:** Since a helical horn's centerline winds in 3D, we define its cross-section radius r as a function of the helix parameter t. One simple choice is a *linear flare*: for example,

$$r(t) = r_{\text{throat}} + (r_{\text{mouth}} - r_{\text{throat}})\,\frac{t}{T}\,,$$

where \$r_{\text{throat}}\$ is the radius at the top (throat, small) and \$r_{\text{mouth}}\$ at the bottom (mouth, large), and \$T\$ is the total parameter value at the bottom end. As \$t\$ runs from \$0\$ to \$T\$, \$r(t)\$ increases smoothly from the throat radius to the mouth radius. For a smoother growth, an **exponential flare** can be used:

$$r(t) = r_{\text{throat}} \cdot \exp\Bigl(\alpha\, t\Bigr)\,,$$

where \$\alpha\$ is chosen such that \$r(T)=r_{\text{mouth}}\$. Solving \$\alpha = \frac{1}{T} \ln(r_{\text{mouth}}/r_{\text{throat}})\$ ensures the radius reaches the desired mouth value at \$t=T\$. Exponential flares produce a gentle, continuously accelerating expansion. Both linear and exponential profiles produce a monotonic increase in radius, avoiding any contractions or bulges. In practice, the range of \$t\$ (e.g. \$0\$ to \$4\pi\$ for two helix turns) and the corresponding radii should be set so that the horn's start and end radii blend into the adjoining geometry of the chalice. To maintain geometric continuity, \$r(t)\$ should be differentiable; the above functions are smooth for all \$t\$. The helix's flare is not derived from classical acoustic formulas, so its profile can be adjusted artistically as needed, provided it increases steadily from top to bottom.

- **Logarithmic Horn Flare:** In a logarithmic spiral horn, the centerline already spirals outward, but we still define a radius profile for the horn's tube cross-section. Often, designers use a flare that complements the spiral's growth. For instance, one can tie the cross-section radius to the spiral's radial distance from the center. A convenient choice is a **proportional flare**: let

$$r(\varphi) = w \cdot r_{\text{centerline}}(\varphi)\,,$$

where \$r_{\text{centerline}}(\varphi)=a\,e^{k\varphi}\$ is the distance of the centerline from the axis (from the spiral equation) and \$w\$ is a fixed proportion (controlling thickness). In other words, as the spiral's distance from the center grows geometrically, the tube radius grows in proportion, maintaining a roughly constant aspect ratio of the horn's coil. Another approach is an exponential or power-law flare in terms of the spiral parameter \$\varphi\$ or arc length \$s\$. For example, we could use

$$r(\varphi) = r_{\text{throat}}\left(\frac{\varphi}{\varphi_{\max}}\right)^{\beta},$$

with \$\beta>0\$ (if \$\beta=1\$, linear increase; if \$\beta>1\$, accelerating growth). Setting \$\varphi_{\max}\$ as the total angle span ensures \$r(\varphi_{\max})=r_{\text{mouth}}\$. The key is to start with a small radius at \$\varphi=0\$ (top) and increase smoothly to a large radius at \$\varphi_{\max}\$ (bottom). Logarithmic horns often emulate forms found in nature (e.g. kudu horns or seashells), which tend to follow exponential or logarithmic growth patterns for both the centerline and flare. Ensuring the flare function is continuous (and ideally differentiable) in \$\varphi\$ yields a smooth horn surface without abrupt changes. For instance, a truly logarithmic flare would mean the **cross-sectional area** grows exponentially with length – similar to an exponential horn – but wrapped into a spiral. In summary, choose \$r(\varphi)\$ to monotonically increase;

typical parameter ranges (like φ from 0 to 4π) combined with a mild growth exponent will keep the geometry continuous and visually pleasing.

- **Tractrix Horn Flare:** The tractrix horn's flare is defined by a specific mathematical curve – the tractrix – which gives a particular profile optimal for certain acoustics. The cross-sectional radius r as a function of axial distance x from the horn mouth (on a straight centerline) is given implicitly by the tractrix equation:

$$x \;=\; a \ln\frac{a + \sqrt{a^2 - r^2}}{r} \;-\; \sqrt{a^2 - r^2}\,,$$

where a is the horn's mouth radius (at $x=0$) and r is the radius at a distance x from the mouth [5]. At $x=0$, this yields $r=a$ (mouth); as x increases, r decreases according to the tractrix curve, reaching the throat radius at the horn's full length. Notably, the tractrix profile approaches zero radius only as $x \to \infty$, but in practice we truncate the horn at a finite $x=L$ when r equals the desired throat radius. The result is a horn of finite length. For example, if we choose a throat radius r_{throat}, we solve the above equation for $x=L$ such that $r(L)=r_{\text{throat}}$. (Alternatively, one can parameterize the tractrix by an angle or hyperbolic function: a convenient parametrization is $x(t)=a\,(t - \tanh t)$ and $r(t)=a\,\sech t$ [6]. By letting t run from 0 at the mouth to some $t_{\max}$ at the throat, one can generate the same profile. Here $\sech t = 1/\cosh t$, and when $r(t_{\max})$ equals the throat radius, that defines $t_{\max}$.) The tractrix flare starts with a gentle slope at the mouth and steepens toward the throat, yielding a characteristic "curve" that ensures sound waves expand with minimal internal reflection [7]. Geometrically, this means the horn's sides curve such that they meet the axis at a right angle at the throat. In terms of continuity, the tractrix function is smooth for $0 < r \le a$; one must be cautious numerically near the throat since r approaches zero if extended too far. By cutting off at a reasonable throat size, we avoid any abrupt geometry. In summary, a tractrix horn embedded in the chalice would start at radius a at the top and follow the above equation to narrow to the bottom. This contour is well-defined and continuous in x, making it suitable for generating via a sweep along a straight path.

- **Exponential Horn Flare:** The exponential horn is defined by an exponentially increasing cross-sectional area along its length. For a circular horn, the radius r as a function of axial distance z can be derived from the area condition $S(z) = S_{\text{throat}}\,e^{\varepsilon z}$ [8]. Since $S = \pi r^2$, the radius follows:

$$r(z) = r_{\text{throat}}\, e^{(\varepsilon/2)\, z}\,.$$

If the horn extends from $z=0$ at the throat to $z=L$ at the mouth, we can determine ε by the desired mouth radius r_{mouth}: $\varepsilon = \frac{2}{L}\ln!\frac{r_{\text{mouth}}}{r_{\text{throat}}}$. Plugging this in yields

$$r(z) = r_{\text{throat}} \exp\left[\ln\frac{r_{\text{mouth}}}{r_{\text{throat}}}\,\frac{z}{L}\right],$$

which simplifies to $r(z) = r_{\text{throat}}\Big(\frac{r_{\text{mouth}}}{r_{\text{throat}}}\Big)^{z/L}$. At $z=L$, indeed $r(L)=r_{\text{mouth}}$. This formula produces an ever-increasing flare that is smooth and convex. In practice, exponential horns are often characterized by a *cut-off* frequency in acoustics, but here we focus on the geometry: an exponential profile gives a very smooth, gradual expansion at first that becomes quite wide near the mouth. One thing to note is that an ideal exponential horn is infinitely long (as $z \to \infty$, $r \to \infty$ in theory), but by selecting a finite L (where r reaches the target mouth size), we truncate it. The transition at the mouth to open air (or in our case, to the chalice's interior space) is typically handled by ensuring the mouth is large enough. Geometrically, $r(z)$ is differentiable for all z, so the surface will have continuous slope. For integration into the chalice, we might set L equal to the chalice's interior height and r_{mouth} to the chalice's inner radius at the bottom. The exponential law guarantees no sudden radius jumps – it's a smooth exponential easing from throat to mouth. This type of flare is *steeper* than the tractrix near the mouth (the radius grows faster in the last portions), which might influence the visual appearance (more flared at the bottom). However, it is straightforward to implement and adjust by tweaking r_{mouth}, r_{throat}, and L.

In summary, **helical and logarithmic horns** offer free-form control over flare shape (which can be tailored with linear or exponential functions for smooth visuals), while **tractrix and exponential horns** come with established mathematical profiles (tractrix giving a finite-length curve defined by the above implicit equation, and exponential giving a simple exponential growth). All four, when parameterized as above, ensure geometric continuity – the radius changes gradually along the centerline without breaks. Table 1 (conceptual) could summarize these profiles: Helical (custom flare, e.g. linear/exponential in t), Logarithmic (e.g. proportional or power-law in φ), Tractrix (derived from tractrix curve, finite L), Exponential (radius grows $\propto e^{z}$). Each can be used to generate the horn geometry by sweeping a circle of radius $r(t)$ along the centerline curve.

Mesh Construction (Three.js)

To implement these horn geometries in Three.js, we will construct a 3D mesh by sweeping a small cross-sectional shape (a circle or polygon approximating a circle) along the parametric centerline, scaling it according to the flare profile. There are a couple of approaches to build this **swept surface** using `THREE.BufferGeometry`:

1. **Using TubeGeometry with a Custom Path:** Three.js provides `THREE.TubeGeometry(path, tubularSegments, radius, radialSegments, closed)` which extrudes a tube along a given 3D curve [9] [10]. We can define our parametric centerline as a `THREE.Curve` subclass. For example, we can implement a custom curve class for the helix or spiral by overriding its `getPoint(t)` to return $(x(t),y(t),z(t))$. Once the path is defined, we pass it to `TubeGeometry`. However, the default TubeGeometry takes a fixed `radius` – it produces a tube of constant radius. To account for a varying radius (flare), we have a few options:
2. **Segmented Tube:** Build the horn in sections. For instance, one could generate multiple TubeGeometries for different segments of the path, each with a slightly different radius, and merge them. This is a crude approximation and can lead to noticeable seams if not finely subdivided.
3. **Modify TubeGeometry (Custom Taper):** A more elegant solution is to modify or extend TubeGeometry to accept a radius function or array. A known method is to sample the intended radius at each tubular segment and scale the vertices accordingly [11] [12]. Essentially, for each point along the path, one computes a local cross-section circle of the desired radius (instead of a fixed radius). For example, on Stack Overflow a user demonstrated subclassing TubeGeometry to take an array of radii corresponding to each path point [11]. In the TubeGeometry generation loop, one can lookup the radius for the current segment (using the segment fraction `u` from 0 to 1) and use that `posRadius` when computing the tube's vertices [12]. This effectively "tapers" the tube geometry.

Another approach is to incorporate a **taper function** that maps the normalized length (0 at throat to 1 at mouth) to a scale factor [13] [14]. While Three.js's core TubeGeometry does not directly support a taper function in the constructor, the concept is to modify the geometry generation such that the circle's radius is multiplied by taper(u) at each step. By customizing TubeGeometry or writing a new function that constructs the geometry from scratch, we can achieve a smoothly varying cross-section.

4. **Custom Sweep (BufferGeometry):** For full control, we can manually construct the mesh via a *sweep algorithm*. This involves:

    1. Sampling the centerline curve at many points (the more segments, the smoother the result).
    2. At each sample point, computing a local orthonormal frame (tangent, normal, binormal vectors) using the Frenet-Serret formulas or Three.js's `Curve.computeFrenetFrames`. This gives the orientation of the horn's cross-section plane.
    3. Creating a circular ring of vertices around the centerline point in the plane perpendicular to the tangent, with radius equal to the flare function at that point. For example, create `radialSegments` points around the circumference of a circle of radius $r(t)$, and transform those points to the world coordinates centered at the sample point.
    4. Adding faces (triangles) or indices between consecutive rings to form the tube surface. Each vertex on ring i connects to the corresponding vertex on ring $i+1$ (and possibly the next vertex, forming quads that we split into triangles). Using this method, we can exactly apply our parametric equations: for each parameter value t_i, compute centerline (x_i,y_i,z_i) and radius r_i. The result is a `THREE.BufferGeometry` containing all the vertices and faces of the horn. This approach requires careful bookkeeping of indices, but yields a highly customizable mesh.

5. **Alternative: ExtrudeGeometry (for planar profiles):** If a horn were simply a 2D profile revolved, one might use `THREE.LatheGeometry` or `THREE.ExtrudeGeometry`. However, our case involves a 3D curve (for helical/logarithmic horns), so those are less straightforward. `ExtrudeGeometry` could sweep a 2D shape along a 3D path, but supporting a varying scale would still require custom modifications (similar to the taper approach above).

For our needs, the **TubeGeometry with a custom radius** is a convenient starting point. For example, one can implement a function `taper(u)` that returns the relative radius at fraction u of the path. If we want a linear increase from 0 to full size: `taper(u) = u` [13]. For an exponential flare, `taper(u) = \exp(\alpha u)` minus some normalization. The stackoverflow solution essentially did this by providing an array of radii and interpolating per segment [12]. By integrating such a solution, we leverage TubeGeometry's built-in ability to handle orientation (normals) while introducing our flare profile. We must ensure the number of samples (tubularSegments) is high enough to capture curvature and radius changes – e.g., 100+ segments for a smooth large horn.

Three.js will generate vertex normals for the tube automatically, which is useful for realistic lighting. We should also consider capping the ends of the horn (throat and mouth) if needed – e.g., adding a disk or connecting to other geometry – but if the horn seamlessly connects to existing chalice parts, caps might not be necessary.

Memory- and performance-wise, a procedurally generated horn with a few hundred segments and, say, 16–32 radial segments (to approximate the circle) is quite manageable in Three.js. The geometry can be created once and then reused or cloned if needed. If the horn's shape will be static, we can even merge it into the chalice's geometry to reduce draw calls.

Integration Notes

Integrating the horn into the Everything Chalice scene involves positioning, orienting, and exporting the generated mesh. Key considerations include:

- **Orientation & Positioning:** By construction, we oriented the parametric equations so that the horn is vertical (along the z-axis) with the desired endpoints. For instance, the helical and logarithmic centerlines were defined to spiral downward. We may need to translate the horn so that its top (throat) begins at the chalice's top interior. If $z=0$ corresponds to the top, then initially our horn goes from $z=0$ (top) to $z=L$ (bottom). In Three.js coordinates, one might set the horn's position such that $z=0$ aligns with the chalice rim and $z=L$ reaches near the base. If the chalice's coordinate system is different (e.g., y-up), we might rotate accordingly. A helical horn might also need to be rotated around the vertical axis to align aesthetically with any existing features (though since it's symmetric around its own axis, rotation about z doesn't change its form, just its starting angle alignment).

- **Scaling:** Ensure the units of the parametric equations match the chalice's scale. For example, if the chalice is modeled in meters and has an interior height of 1 m, then the horn length L should be 1 to span it. If not, we can uniformly scale the mesh after creation to fit. Non-uniform scaling should be avoided if we want to preserve the intended flare proportions.

- **Mesh Combination:** If the horn is to be a non-standalone part of the scene (attached to the chalice geometry), we have a couple of options. We can either keep it as a separate `THREE.Mesh` object and position it appropriately (the simplest approach), or merge it with the chalice's geometry. Merging (via `BufferGeometry.mergeVertices()` / `BufferGeometry.toNonIndexed()` and manual vertex merging) is possible since it's all geometry, but it might be easier to treat it as its own mesh node parented to the chalice object. This way, we maintain flexibility in adjusting the horn or replacing it with a different type if needed, without reprocessing the entire chalice mesh.

- **Exporting to OBJ/GLTF:** Once the horn's mesh is finalized and placed, we may want to export it for use in other tools or for saving the scene. Three.js offers **exporters** as addons. For OBJ format (geometry + simple material), we can use `THREE.OBJExporter`. This exporter will output the mesh (or an entire scene/object) as a text in Wavefront `.obj` format. For example:

```
import { OBJExporter } from 'three/addons/exporters/OBJExporter.js';
const exporter = new OBJExporter();
const objText = exporter.parse(scene);
console.log(objText);
```

Using `exporter.parse(scene)` will serialize all children of the scene into one OBJ string [15]. We could also pass the specific horn mesh to `parse` if we only want that object. Note that OBJExporter does not export materials or textures to an MTL file, it only exports geometry [16]. In our case, the horn is likely just a single material (maybe a procedural or basic material), so this is fine. After obtaining the OBJ text, we can initiate a download (in a browser context) or send it to a server as needed.

For GLTF/GLB (the modern, portable format), Three.js provides `THREE.GLTFExporter`. GLTF can include geometry, hierarchy, materials, and more. Using it looks like:

```
import { GLTFExporter } from 'three/addons/exporters/GLTFExporter.js';
const exporter = new GLTFExporter();
exporter.parse( scene, function(result) {
    // result is a JSON object for .gltf or an ArrayBuffer for .glb
    saveArrayBufferOrJson(result);
},
options );
```

There is also a promise-based `parseAsync` that returns the data directly [17]. The exporter has options – for example, `{ binary: true }` to produce a `.glb` binary. In a browser, one can trigger a download of the resulting blob. If exporting just the horn mesh, ensure you pass the mesh (or an array containing it) instead of the whole scene. The GLTF exporter will preserve the horn's shape and any material (if it's a MeshStandardMaterial or similar that is compatible). One thing to note: if the horn uses a custom shader or non-standard material, the exporter might not fully capture that; but geometry and basic materials are handled.

- **Verification:** After exporting, it's wise to import the OBJ/GLTF into a modeling tool or viewer to ensure the horn appears as expected – continuous spiral, correct orientation, etc. Because our horn is generated procedurally, it should be *watertight* (no holes) unless intentionally left open at one end. OBJ doesn't encode the scene scale or units explicitly, whereas GLTF does store units (generally meters by default). As long as we were consistent in Three.js, both exporters will produce a correctly scaled model.

- **Material and Appearance:** While not the focus of geometry, note that a complex horn surface can benefit from smoothing. Three.js BufferGeometry by default will average normals across shared vertices (if the geometry is not indexed, or if we ensure consistent indices). TubeGeometry and our custom sweep will produce a smooth shade across the horn. If a sharp crease is needed (likely not here), we'd duplicate vertices along that crease. But since this horn is continuous, smooth shading is appropriate. We may apply a material similar to the chalice's interior, or something contrasting to highlight the horn's form. These choices carry over on export (GLTF will include material; OBJ requires an MTL or manual material assignment after import).

- **Embedding in Scene:** The horn being "embedded" in the chalice means it should aesthetically and spatially integrate. Likely, the top of the horn (small radius) might originate at the center of the chalice's bowl, and the bottom might terminate near the base. Because our horn parametrics yield a single continuous mesh, it inherently connects those two points. We just need to ensure alignment: e.g., the bottom end of the horn might need to coincide with a hole or receptacle in the chalice base.

Minor adjustments like translating the horn or extending its length slightly can ensure it fits perfectly. Using the parametric definitions, small tweaks (like increasing T or L by a few percent) will lengthen the horn without altering its fundamental character, allowing a snug fit.

In conclusion, constructing the horn mesh with Three.js involves generating the path and radius profile, creating the geometry (via TubeGeometry or a custom routine), and then positioning it into the chalice model. By following the parametric equations and methods above, we ensure the horn is smooth and accurate. The final mesh can be exported to standard formats for further use or archived as part of the Everything Chalice design.

References

1. Math.net – *3D Spiral (Helix) Definition:* Parametric equation of a helix (spiral in 3D) given as $x=r\cos t,\;y=r\sin t,\;z=at$ [1] .

2. Wikipedia – *Logarithmic Spiral:* Definition in polar form $r = a e^{k\varphi}$ and Cartesian parametric form $x = a e^{k\varphi}\cos\varphi,\;y = a e^{k\varphi}\sin\varphi$ [2] [3] .

3. Wikipedia – *Tractrix Curve:* Parametrization of the tractrix (equitangential curve) as $x=t-\tanh t,\;y=\sech t$ which describes the 2D tractrix profile [6] . Also describes the tractrix horn as the surface of revolution of this curve and its acoustic advantages [7] .

4. AudioHeritage Forum – *Tractrix Horn Equation:* Implicit formula $x = a \ln\frac{a+\sqrt{a^2 - r^2}}{r} - \sqrt{a^2 - r^2}$ relating axial distance x, mouth radius a, and local radius r for a tractrix horn [5] .

5. M. Zollner, *Horn-Loudspeakers (Gitec Forum Translation)* – Notes that for an exponential horn, cross-sectional area grows as $S(z) = S_{\text{throat}}\exp(\varepsilon z)$ [8] , implying radius $r(z)$ grows exponentially with z.

6. Three.js Documentation – *TubeGeometry:* Usage example of `TubeGeometry(curve, segments, radius, radialSegments, closed)` and explanation of parameters [9] [10] . Indicates TubeGeometry extrudes along a 3D path with a fixed radius by default.

7. Stack Overflow – *Variable Radius TubeGeometry:* Discussion and solution for modifying TubeGeometry to accept a radius array. Code snippet shows interpolating a `posRadius` for each segment and computing tube vertices with that radius [11] [12] .

8. Reddit (r/threejs) – *Tube Taper Function:* Suggestion to provide a taper function to TubeGeometry to vary radius along the tube (e.g., `taper(u)=u` for linear scaling) [13] .

9. Three.js Docs – *OBJExporter:* Example showing how to create an OBJExporter and parse a scene to get OBJ data [15] . Confirms it exports geometry (and that materials must be handled separately if needed).

10. Three.js Docs – *GLTFExporter:* Example usage of GLTFExporter's `parseAsync` to export a scene to glTF (JSON or binary) [17]. Describes glTF as an efficient format for 3D scenes which we use to export the horn and chalice.

---

[1] Spiral
https://www.math.net/spiral

[2] [3] [4] Logarithmic spiral - Wikipedia
https://en.wikipedia.org/wiki/Logarithmic_spiral

[5] Bruce Edgar Midrange Horn for 2445 - LANSING HERITAGE
https://www.audioheritage.org/vbulletin/printthread.php?t=36373&pp=100

[6] [7] Tractrix - Wikipedia
https://en.wikipedia.org/wiki/Tractrix

[8] PotEG_Ch11TMT
https://www.gitec-forum-eng.de/wp-content/uploads/2019/03/poteg-11-10-horns.pdf

[9] [10] TubeGeometry - Three.js Docs
https://threejs.org/docs/pages/TubeGeometry.html

[11] [12] three.js tube with variable radius - Stack Overflow
https://stackoverflow.com/questions/20955852/three-js-tube-with-variable-radius

[13] [14] Draw a custom tube geometry : r/threejs
https://www.reddit.com/r/threejs/comments/3se0xy/draw_a_custom_tube_geometry/

[15] [16] OBJExporter - Three.js Docs
https://threejs.org/docs/pages/OBJExporter.html

[17] GLTFExporter - Three.js Docs
https://threejs.org/docs/pages/GLTFExporter.html