ChatGPT

Deformable Balloon Envelope Simulation Design

The balloon envelope is modeled as a closed **triangular mesh** (e.g. an icosphere) whose vertices are connected by elastic springs. An icosphere (a subdivided icosahedron) is a good choice because it yields a roughly uniform triangular tessellation of the sphere. Other options (UV-sphere, subdivided octahedron, etc.) can work, but icospheres avoid pole clustering. At initialization, generate the mesh geometry (vertices and triangle indices) and record the **rest lengths** of each edge. Store the original (rest) mesh so that strains can be measured relative to the undeformed state. The mesh should be watertight and oriented consistently so that face normals point outward. For example, Wang et al. simulate balloon inflation using a volume-preserving thin-shell model on a triangular mesh [1]. In practice, this mesh is created in Bevy as a `Mesh` resource, and each vertex has an associated mass. We also build a connectivity structure (list of edges and adjacent faces) to compute forces.

- *Mesh resolution:* Higher subdivisions yield smoother shapes but more springs to compute. Choose resolution to balance accuracy and performance (e.g. 256 triangles vs 1024). We may also consider dynamic tessellation or level-of-detail if needed.
- *Mesh alternatives:* An icosphere gives even vertex spacing. A subdivided cube or octahedron can also work but produce irregular triangles. Simpler low-poly balloons (e.g. dodecahedron) may reduce cost but distort shape. Ultimately, an icosphere is recommended for plausibility and even stretch distribution.
- *Thin-shell assumption:* We treat the envelope as a thin elastic shell. Bending/shear stiffness is minimal (balloon rubber bends easily), so we focus on in-plane stretching only [1]. This greatly simplifies simulation: the main deformation force is edge stretching, with negligible bending torque.

Volume and Surface-Area Calculation

At each step we compute the **enclosed volume** of the deformed mesh, which drives buoyancy and pressure. We use the standard tetrahedralization method: for every triangular face with vertices v_1, v_2, v_3, form a tetrahedron with the origin (or any fixed reference point) and compute its signed volume. The formula for one triangle is $V_{\text{tetra}} = \frac{1}{6}(v_1 \times v_2) \cdot v_3$ (taking vertex coordinates relative to the reference). Summing these signed volumes over all faces yields the total volume of the closed mesh [2]. We must ensure the mesh is closed and consistently oriented so that inward and outward faces cancel correctly. Similarly, we compute surface area by summing the area of each triangle: $A_{\text{tri}} = \frac{1}{2}\|(v_2 - v_1) \times (v_3 - v_1)\|$, accumulating over all faces.

[2] To calculate enclosed volume, sum the signed volumes of tetrahedra formed by each triangle and a reference point. For a triangle with vertices v_1, v_2, v_3, one uses the well-known formula $V = \frac{1}{6}(v_1 \times v_2) \cdot v_3$; positive and negative contributions from face orientations cancel to give the interior volume. The mesh surface area is computed by summing triangle areas $\frac{1}{2}\|(v_2 - v_1) \times (v_3 - v_1)\|$ over all faces. (In practice one can use available geometry libraries or code this directly.)

Strain and Elasticity Model

Each edge in the mesh acts like a spring. We let each edge e have a rest length L_0 (from the original mesh) and current length L . The *strain* in that edge is $\epsilon = \frac{L-L_0}{L_0}$. By Hooke's law, an edge applies a force $F = -k(L - L_0)$ directed along the edge (plus damping), where k is the elasticity constant set to approximate rubber stiffness. Because balloons stretch a lot, we choose moderate k so the envelope can expand substantially without numerical instability. (Wang et al. note that balloon deformation is dominated by stretching with negligible bending/shear [1], so this spring network is appropriate.) We distribute each edge's spring force equally to its two endpoint vertices.

Strain limiting: We track the maximum absolute stretch of each edge. If L/L_0 exceeds a critical threshold (design-dependent, e.g. 2× or 3×), we consider the envelope torn/popped. When popping occurs, the balloon entity can be removed or switched to a "popped" state. In practice, we check after each update: if any edge's $\frac{L-L_0}{L_0}$ exceeds the material limit, trigger the pop event.

- *Rationale:* Latex balloons can stretch ~100–200%. We set a threshold (e.g. 2× length) as the failure point. (In contrast, fabrics allow very little strain [3], but rubber is more forgiving.)
- *Implementation:* Maintain an array of rest lengths at initialization. Each frame compute current lengths, apply spring force, and check `if L/L0 > strain_limit { popped = true; }`.

Internal Pressure and Gas Model

We assume the balloon is filled with a fixed amount of ideal gas. By the ideal gas law $pV = nRT$, as the envelope expands its internal pressure p drops inversely with volume. For simplicity, one can assume isothermal conditions and use

$$p = p_0 \frac{V_0}{V},$$

where p_0, V_0 are the initial pressure and volume [4]. (This follows from n, R, T constant in a closed balloon.) In each update, after computing the new volume V we set the internal pressure accordingly.

We then apply **pressure forces** to the mesh: each triangular face of area A_e exerts a force $F_e = p\,A_e$ in the outward normal direction n_e . We distribute this force to the triangle's vertices equally. Concretely, for each face e with normal n_e , we add $\frac{1}{3} p A_e n_e$ to each of its three vertices' net force [5]. This approximates a continuous internal pressure load on the shell. Note that Wang et al. derive the same discrete nodal pressure force formula for a triangular mesh [5] .

[4] [5] We model gas pressure via $pV = \text{const}$, so $p = p_0 V_0 / V$ and pressure inversely scales with volume [4]. Each triangle face with area A_e applies a force $p\,A_e$ along its outward normal. This force is split evenly to the face's vertices, i.e. each vertex in face e receives a nodal force $\frac{1}{3} p A_e n_e$ [5]. Summing all such face contributions gives the total internal pressure force on each vertex.

Time Integration and Vertex Update

Every simulation tick we accumulate forces on each vertex: elastic spring forces from edges, pressure forces from faces, (optionally gravity or collision responses), and damping forces (to stabilize vibrations). We then update each vertex's position and velocity using a time integrator (explicit Euler or Verlet), using Bevy's fixed timestep (through Avian's physics schedule). For example, with mass m , acceleration $a = F/m$, a simple explicit Euler step is $v \leftarrow v + a\Delta t$, $x \leftarrow x + v\Delta t$. We also apply a damping term (e.g. $F_{\text{damp}} = -c\,v$) to dissipate energy and prevent explosion of oscillations. In practice a small damping (e.g. 1–5% of elastic force) suffices.

This loop (compute volume, update pressure, compute forces, integrate positions) runs each frame. Since balloons deform slowly, explicit integration is usually stable if timesteps are small (Avian's fixed 60 Hz or higher). For robustness one could use semi-implicit integration or sub-stepping. One key check is to re-compute volume and pressure after each position update to reflect the change in shape.

Buoyancy and Drag Estimation

Using the current volume, we compute buoyant force by Archimedes' principle:

$$F_{\text{buoy}} = \rho_{\text{fluid}}\, g\, V,$$

where ρ_{fluid} is the surrounding fluid density and V is the enclosed volume [6] . We apply this upward force at the balloon's centroid (or uniformly distribute to vertices). This ensures the balloon rises or sinks according to displaced fluid weight.

For drag, we need an effective cross-sectional area $A_\perp$. A simple approximation is to treat the balloon as a sphere of equal volume: let radius $r = (3V/4\pi)^{1/3}$ and area $A_\perp = \pi r^2$. This gives an order-of-magnitude drag. For more accuracy, one can compute the projected (silhouette) area in the wind direction. In aerodynamic analysis, the *projected area* is the outline (silhouette) area of the model from a given direction [7] . Practically, we project each triangular face onto a plane perpendicular to the velocity and sum their area contributions (or sum $A_e \cos\theta_e$ for faces facing the flow). In either case, this yields $A_\perp$ which enters $F_{\text{drag}} = \frac{1}{2} C_d \rho v^2 A_\perp$. At first, a constant sphere-based $A_\perp$ is simplest; later one can refine using the mesh normals and faces (or even an enclosing bounding box) to approximate the true silhouette.

[6] [7] We apply buoyancy $F_b = \rho g V$ according to Archimedes' principle [6] . Drag can be approximated via a sphere of equal volume (area $= \pi r^2$) or by computing the mesh's projected area (silhouette) along the velocity direction [7] . The projected-area method yields a more accurate $A_\perp$ by summing face areas weighted by the normal's alignment with the flow.

Strain Limit and Popping

We continuously monitor the strain of each edge: $\epsilon = (L - L_0)/L_0$. If $|\epsilon|$ exceeds a predefined **strain threshold** (based on material strength), we trigger a pop. For example, if an edge stretches beyond 150% (or any chosen limit), we mark the balloon as burst. This is a Boolean check each frame. Upon popping, the balloon entity can be removed or flagged to disable forces (and perhaps spawn a debris effect). Physically, latex fails at finite stretch, so this threshold models rupture. (In many cloth systems, similar strain limits or breakable springs are used to simulate tearing.) No complex solver is needed here—just a max-extension check.

- *Threshold selection:* Based on real balloon properties, ~1.5×–2× rest length is reasonable. This is a tunable parameter.
- *Implementation:* After updating vertex positions, loop over edges: if `current_length > pop_factor * rest_length`, set `popped = true`.

Trade-offs and Alternatives

- **Mesh type:** An icosphere is preferred for uniformity. UV-spheres (latitude-longitude) have pinched poles that can over-stress. Low-poly shapes (octahedron, cube) are lighter but distort heavily. We could also use a tetrahedral solid mesh for full 3D FEM, but that is overkill. We focus on surface meshes (thin-shell).
- **Simulation method:** We use a **mass-spring** model for simplicity. More advanced methods exist (Position-Based Dynamics, Finite Elements), but springs are easy to implement and efficient for cloth-like behavior. PBD could enforce exact volume preservation, but is more complex and not necessary here. (Wang et al. use FEM+volume constraint [1], but that's complex to code in Bevy.) Our approach allows large strains and naturally couples pressure and elasticity.
- **Time integration:** Explicit integration is straightforward but requires small timesteps. An implicit or semi-implicit solver would allow larger timesteps, but is harder to implement from scratch. Given moderate performance needs (only one balloon), explicit Euler or Verlet usually suffices with $\Delta t \sim 1/60s$.
- **Collision handling:** If the balloon hits obstacles, one could wrap the deformable mesh in a collider (Avian supports mesh colliders) or do custom collision. This is advanced; initially we might ignore collisions or assume free float.
- **Simplification:** If performance is a concern, use a coarse mesh (fewer vertices) or reduce spring count by fixing some vertices (e.g. tether points). One could also fuse edges or use a lower-dimensional model (spherical harmonic shape), but at loss of fidelity.
- **Bevy/Avian integration:** We run this custom balloon simulation within Bevy's ECS. Avian3D handles other physics (rigid bodies, buoyancy plugin, etc.), but since Avian has no built-in soft bodies, we manually update vertex transforms each frame. We can still register a dynamic **RigidBody** for the balloon as a whole (for example to apply buoyant force), but we actually override its shape each frame according to our mesh's deformation. Alternatively, skip Avian bodies entirely and integrate motion "by hand", using Avian only for constants like gravity. Either way, we output the updated mesh as a Bevy `Mesh` component so it is rendered.

In summary, the balloon is a pressure-driven elastic sphere simulated as a mass-spring shell. Its volume is computed exactly from the triangle mesh [2], driving both buoyancy [6] and pressure. Edges resist stretch via Hooke's law, and we pop the balloon if any edge's strain is too high. Drag is estimated from cross-sectional area (sphere or mesh silhouette) [7]. This design balances realism (volume preservation, plausible shapes) with implementation simplicity (mass-spring on an icosphere) in Bevy + Avian.

Sources: Standard triangle-mesh volume formulas [2]; aerodynamic projected-area concept [7]; Archimedes' buoyancy law [6]; and balloon elasticity literature [1] [4] inform the above design.

---

[1] [4] [5] eprints.bournemouth.ac.uk
https://eprints.bournemouth.ac.uk/21970/1/Balloon-conference.pdf

[2] Calculating the volume of a mesh – Nervous System blog
https://n-e-r-v-o-u-s.com/blog/?p=4415

[3] cs.columbia.edu
http://www.cs.columbia.edu/cg/pdfs/131-ESIC.pdf

[6] Archimedes' principle - Wikipedia
https://en.wikipedia.org/wiki/Archimedes%27_principle

[7] Projected Area – OpenVSP Ground School
https://vspu.larc.nasa.gov/training-content/chapter-3-model-analysis-in-openvsp/projected-area/