ButterFlight

Maya Plug-in Design Document

By: Cecilia Chen and Yiding Tian

Based on:

A Practical Model for Realistic Butterfly Flight Simulation.

Chen, Q., Lu, T., Tong, Y., Luo, G., Jin, X., and Deng, Z., 2022.
ACM Transactions on Graphics (TOG).

CIS 6600 — Advanced Topics in Computer Graphics and Animation

Instructor: Dr. Stephen Lane

Project Summary

Realistic butterfly flight animation is a recurring need in film, games, and virtual environments, yet no dedicated authoring tool exists for it. Hand-keying the erratic, noisy trajectories characteristic of real butterfly flight is prohibitively tedious, and CFD-based simulation is far too slow for production use—leaving artists with crude approximations that break the illusion of life.

ButterFlight addresses this gap by providing a Maya plug-in that generates physically plausible butterfly flight animations through a force-based simulation model. The design goals are to faithfully reproduce the characteristic features of real butterfly flight—including wing-abdomen coupling, aerodynamic lift and drag, and inherent trajectory noise—while keeping the interface simple enough for a generalist animator to use without any background in aerodynamics or simulation.

The primary target audience is VFX artists and technical directors working in film, games, or real-time virtual environments who need to populate a scene with one or more butterflies without spending hours on hand animation. Users are expected to have basic Maya modeling and rigging knowledge but require no knowledge of the underlying physics.

ButterFlight exposes a MEL-scripted UI panel from which the user can load a rigged butterfly mesh, set simulation parameters (flapping frequency, amplitude, wind direction and strength, swarm size), optionally attach the butterfly to a path curve, and bake the resulting motion as keyframed skeletal animation. Supported output modes include single-butterfly free flight, path following, wind interaction, and swarm aggregation. The final animation can be exported via Alembic or FBX for use in any downstream pipeline.

The core simulation is implemented as a Maya C++ plug-in (`.mll`) built against the Maya 2026 API. It runs the three-module algorithm from Chen et al. 2022: parametric maneuvering functions with wing-abdomen interaction, a force model combining quasi-steady aerodynamic forces with a curl-noise vortex force, and a maneuvering controller that decouples body translation from posture adjustment. The UI and scene setup utilities are written in MEL/Python.

The development schedule targets an alpha version (single-butterfly simulation with basic aerodynamic forces and MEL UI) by March 25th, a beta version (full force model, path following, and swarm support) by April 15th, and a final polished version with tutorials and demo scenes due May 11th.

1. Authoring Tool Design

1.1 Significance of Problem or Production/Development Need

Butterflies appear in a wide range of production contexts—background ambience in nature documentaries, hero insect shots in animated features, environmental atmosphere in video games and virtual reality, and educational simulations. Despite this demand, no dedicated authoring tool for butterfly flight animation exists in any major DCC package. Artists currently resort to one of three unsatisfactory options: (1) hand-keying every frame of wing and body motion, which is extremely time-consuming given the high flapping frequency ($\approx$9–11 Hz) and the erratic, noisy trajectories that characterize real butterfly flight; (2) looping a pre-made cycle animation, which produces robotic, repetitive motion with no dynamic response to the environment; or (3) outsourcing to a CFD-based simulation, which requires specialist knowledge, days of computation time, and produces results that are difficult to art-direct.

The fundamental challenge is that butterfly flight is governed by closely coupled wing-body interaction: the abdomen oscillates in counterphase to the wings, the thorax pitches with each stroke, and an inherent vortex-driven noise keeps the trajectory unpredictable. Without capturing all three effects simultaneously it is impossible to produce convincing butterfly animation in any setting, whether for a single hero butterfly or a swarm of hundreds. A tool that automates this physics while remaining controllable and fast enough for interactive use in Maya would directly address this production gap.

1.2 Technology

ButterFlight is based on the 2022 ACM SIGGRAPH paper *"A Practical Model for Realistic Butterfly Flight Simulation"* by Qiang Chen, Tingsong Lu, Yang Tong, Guoliang Luo, Xiaogang Jin, and Zhigang Deng. The paper proposes the first force-based model specifically designed for real-time butterfly flight simulation in graphics and animation applications. The approach is organized into three inter-connected modules, as illustrated in Figure 1.

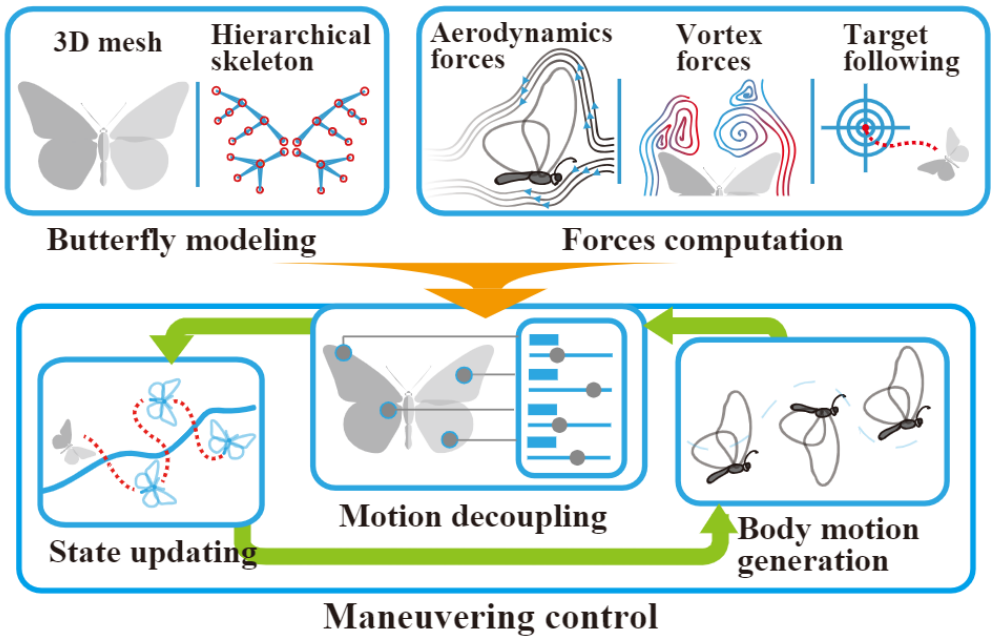


Figure 1: Pipeline overview of the Chen et al. 2022 approach. Starting from a rigged butterfly mesh, the system computes aerodynamic and vortex forces, then applies a maneuvering controller to generate body motion and trajectories (reproduced from Chen et al. 2022, Fig. 1).

The first module, **butterfly modeling**, represents the butterfly as a hierarchical skeleton (thorax as root, four wings, abdomen) driven by parametric maneuvering functions. Five joint angles—thorax pitch θ_β, fore-wing flap θ_γ, fore-wing feather θ_ζ, fore-wing sweep θ_ψ, and abdomen rotation θ_ϕ—are computed each frame via a cosine-based periodic function (Equation 1 of the paper) whose amplitude and frequency scale with the butterfly's current velocity. The abdomen is assigned a phase offset of $-180°$ relative to the wings to reproduce the counterphase oscillation observed in real Monarch butterflies [12]. Figure 2 illustrates the five joint angles and their geometric definitions.

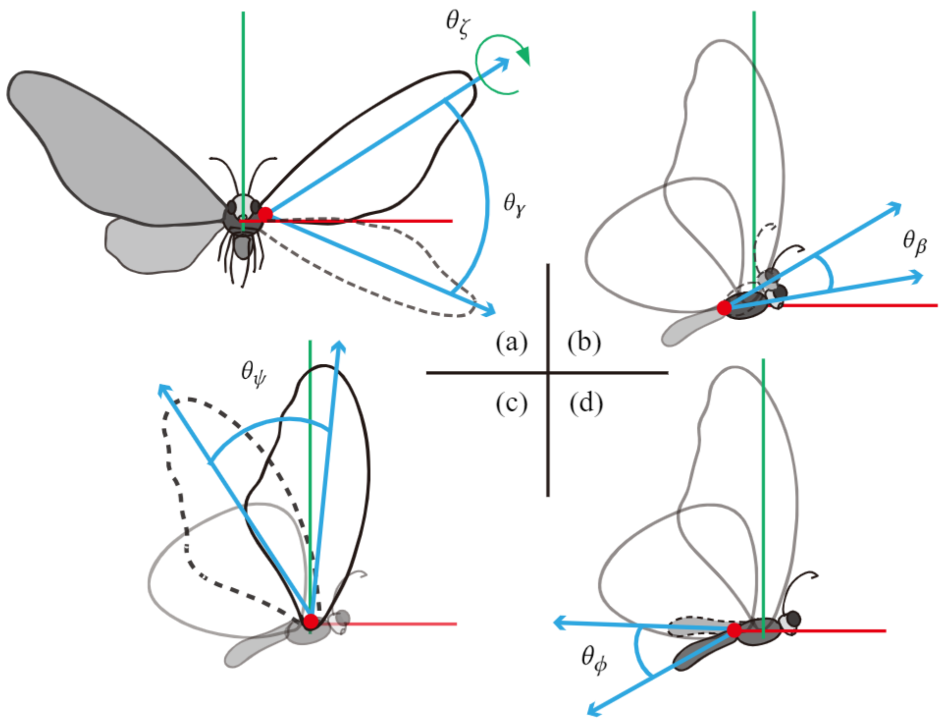


Figure 2: Joint angle definitions of the butterfly skeleton model. The thorax pitch angle θ_β and abdomen rotation θ_ϕ are shown in (d) and (b); the fore-wing flap angle θ_γ and feather angle θ_ζ are shown in (a); the sweep angle θ_ψ is shown in (c) (reproduced from Chen et al. 2022, Fig. 4).

The second module, **forces computation**, computes two forces acting on the butterfly at each time step. A simplified aerodynamic force (lift and drag per wing polygon, Equations 4–6) is derived from the quasi-steady theory of Ellington (1984), using empirical lift and drag coefficient polynomials as a function of the local angle of attack. A vortex force (Equation 7) is computed from a curl-noise field [8] seeded with Perlin noise [6] and applied to the thorax centre of mass, simulating the wake of the flapping wings and producing the inherent chaotic trajectory noise characteristic of real butterfly flight.

The third module, **maneuvering control**, decouples body translation from body posture. At each time step the composite force (aerodynamic + vortex + gravity + optional attraction toward a target or path keypoint) is integrated via Newton's second law to obtain velocity. Wing frequency and amplitude are then updated using a sliding-window smoother (Equation 12) to avoid abrupt changes across flapping cycles. Figure 3 shows the phase relationships of the five joint angles across one full flapping cycle.

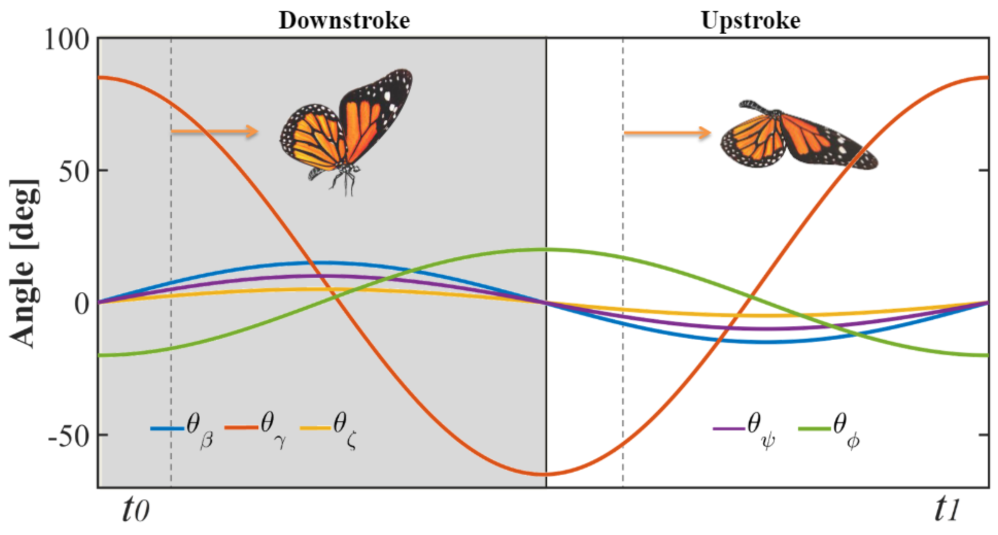


Figure 3: Phase shifts of all five maneuvering angles during one wing-flapping cycle (downstroke t_0 to upstroke t_1). The abdomen angle θ_ϕ is driven at $-180°$ phase offset relative to the flap angle θ_γ, reproducing counterphase oscillation (reproduced from Chen et al. 2022, Fig. 5).

We chose this paper because it is the only published method that addresses all three distinguishing features of butterfly flight (wing-abdomen coupling, aerodynamic force, and inherent trajectory noise) in a single, real-time capable framework. The algorithm is self-contained, mathematically explicit, and maps cleanly onto Maya's joint/keyframe animation system, making it an ideal candidate for implementation as an interactive authoring tool. The paper also includes quantitative parameter tables (Tables 2 and 3) for two butterfly species, providing concrete values to initialize our simulation.

1.3 Design Goals

ButterFlight addresses the production need by providing a Maya plug-in that automates the entire butterfly flight simulation pipeline—from joint-angle computation through trajectory integration to keyframe baking—behind a single, artist-friendly UI panel. The animator supplies a rigged butterfly mesh and a handful of high-level parameters; the plug-in handles all physics. The resulting output is standard Maya keyframe animation that can be refined by hand, cached to Alembic, or exported to any downstream pipeline without any dependency on the plug-in at render time.

1.3.1 Target Audience

The primary users of ButterFlight are **VFX artists and technical directors** in film, television, game development, and real-time virtual production who need to include one or more butterflies in a scene. This includes generalist animators who want a quick, convincing butterfly without manual keyframing, and TDs who need to populate a background environment with a swarm of butterflies in an automated, art-directable way. Secondary users are students and researchers who wish to explore physically-based insect animation.

Users are expected to have basic Maya literacy—they should know how to import meshes, work with joints and skinning, and use basic curve tools for path creation. No background in aerodynamics, physics simulation, or programming is required to operate the tool.

1.3.2 User Goals and Objectives

With ButterFlight, a user should be able to accomplish the following:

- **Single butterfly, free flight.** Simulate one butterfly flying with inherently noisy, physically plausible motion without specifying any path, suitable for ambient background use.
- **Single butterfly, path following.** Attach the simulation to a user-drawn NURBS curve so the butterfly broadly follows an art-directed trajectory while still exhibiting natural erratic deviations and wing-body dynamics.
- **Wind interaction.** Apply a constant or time-varying wind vector to the simulation and observe the butterfly attempt to recover stable flight, producing realistic spiraling responses.
- **Swarm simulation.** Instantiate multiple butterflies (up to $\sim$100 at interactive rates) that fly with collision-free, divergence-free curl-noise trajectories and individually-computed wing-body motion.
- **Parameter tuning.** Adjust species-specific physical parameters (wing area, body mass, flapping frequency range, vortex force gain) to produce different butterfly species or stylized variants.
- **Keyframe bake and export.** Bake the simulation result to Maya keyframes on the input rig, ready for render or export via Alembic or FBX.

The user should spend minimal time on setup—loading a rig, pressing *Simulate*, and baking should take no more than a few minutes for a single butterfly.

1.3.3 Tool Features and Functionality

ButterFlight exposes the following features:

- **Rig assignment.** The user selects a pre-rigged butterfly mesh from the Maya scene. The plug-in expects the rig to follow a standard joint-naming convention (`BF_thorax`, `BF_forewing_L/R`, `BF_hindwing_L/R`, `BF_abdomen`) so it can bind the correct simulation channels to the correct joints automatically.
- **Simulation mode selector.** Choose between *Free Flight*, *Path Following*, and *Swarm* modes from a drop-down menu.
- **Path curve input.** In Path Following mode, the user selects an existing NURBS curve from the scene as the attraction path.
- **Wind force control.** A direction vector and magnitude slider for a constant wind, plus a checkbox to enable time-varying (sinusoidal) wind.

- **Simulation parameter controls.** Numeric fields and sliders for: vortex force gain (η, $gain_{x,y,z}$), simulation duration (frames), playback frame rate, and sliding-window size k.
- **Swarm controls.** Number of agents, spatial spread at initialization, and inter-agent repulsion radius.
- **Simulate button.** Runs the C++ simulation for the specified duration and previews the result in the Maya viewport via a lightweight draw override.
- **Bake to Keyframes button.** Writes the simulation output as `setKeyframe` calls on all rig joints and the root transform, producing standard Maya animation curves.
- **Export.** After baking, the user can export to Alembic (`.abc`) or FBX through the standard Maya export dialogs; no special plug-in support is needed at this stage.

1.3.4 Tool Input and Output

Input:

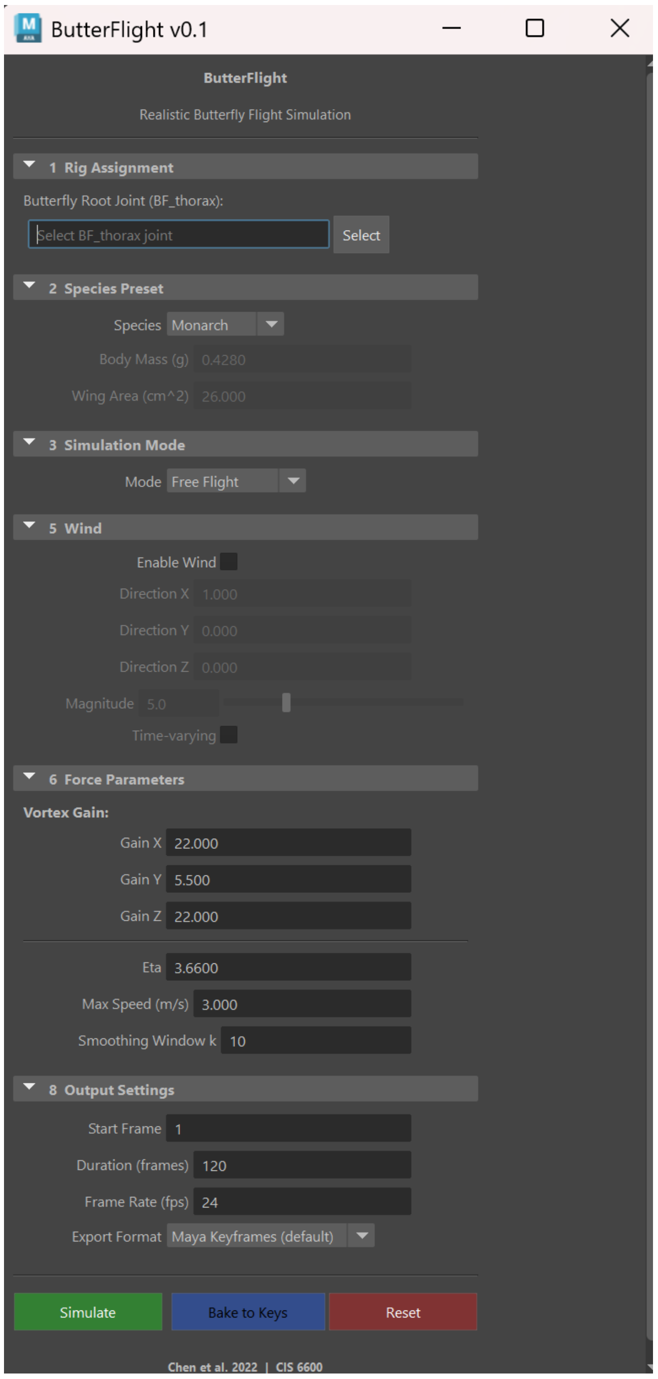
- A Maya scene containing one or more rigged butterfly meshes, with joints named according to the ButterFlight convention.
- (Optional) A NURBS curve in the scene to serve as the flight path.
- (Optional) A wind vector specified through the UI.
- Simulation parameters set through the UI panel (simulation mode, duration, vortex force parameters, swarm count).

Output:

- Maya keyframe animation curves on all rig joints (rotation) and the root transform (translation and rotation), covering the simulated frame range.
- The output is entirely standard Maya animation data—no custom nodes are left in the scene after baking, and the animated rig can be rendered, referenced, exported, or hand-edited like any other Maya animation.

1.4 User Interface

ButterFlight's entire user-facing interface is a single floating panel window inside Maya, opened by sourcing the MEL script `butterFlight_ui.mel` and calling `butterFlightUI`. The panel is implemented as a scrollable `columnLayout` containing eight collapsible `frameLayout` sections—one per logical parameter group—followed by three color-coded action buttons at the bottom. No new menus are added to the main Maya menu bar; the tool is fully self-contained in the floating window. Figure 4 shows the panel as loaded in Maya 2026.



ButterFlight v0.1

ButterFlight

Realistic Butterfly Flight Simulation

1 Rig Assignment

Butterfly Root Joint (BF_thorax):

Select BF_thorax joint | Select

2 Species Preset

Species	Monarch
Body Mass (g)	0.4280
Wing Area (cm^2)	26.000

3 Simulation Mode

Mode	Free Flight

5 Wind

Enable Wind	☐
Direction X	1.000
Direction Y	0.000
Direction Z	0.000
Magnitude	5.0
Time-varying	☐

6 Force Parameters

Vortex Gain:

Gain X	22.000
Gain Y	5.500
Gain Z	22.000
Eta	3.6600
Max Speed (m/s)	3.000
Smoothing Window k	10

8 Output Settings

Start Frame	1
Duration (frames)	120
Frame Rate (fps)	24
Export Format	Maya Keyframes (default)

Simulate | Bake to Keys | Reset

Chen et al. 2022 | CIS 6600

Figure 4: ButterFlight v0.1 UI panel running inside Maya 2026. Sections 4 (Path Settings) and 7 (Swarm Settings) are hidden by default and become visible when the corresponding simulation mode is selected.

1.4.1 GUI Components and Layout

The panel is divided into eight numbered, collapsible sections plus an action-button row. Table 1 summarises each section; a detailed description follows.

#	Section name	Key controls
1	Rig Assignment	Text field + *Select* button for `BF_thorax`
2	Species Preset	Drop-down (Monarch / Swallowtail / Custom); mass, wing area
3	Simulation Mode	Drop-down (Free Flight / Path Following / Swarm)
4	Path Settings	Curve picker + follow-strength slider (conditional)
5	Wind	Enable check; direction XYZ; magnitude slider; time-varying
6	Force Parameters	Vortex gain X/Y/Z; η; max speed; smoothing window k
7	Swarm Settings	Agent count; spawn spread; repulsion radius (conditional)
8	Output Settings	Start frame; duration; fps; export format
	Simulate Bake to Keys Reset	

Table 1: ButterFlight GUI sections and their primary controls.

Section 1 — Rig Assignment. A `textFieldButtonGrp` displays the name of the currently assigned root joint. The *Select* button reads the active Maya selection, validates that the chosen node is a joint and that its name begins with the `BF_` prefix, and writes it into the field. A warning dialog is shown if the prefix check fails, allowing the user to override or cancel.

Section 2 — Species Preset. An `optionMenuGrp` offers three entries: *Monarch*, *Swallowtail*, and *Custom*. Selecting a named species auto-populates the body mass and wing-area fields with the values from Tables 2 and 3 of Chen et al. [1] (Monarch: 0.428 g, 26 cm^2; Swallowtail: 0.34 g, 28 cm^2) and disables those fields to prevent accidental edits. Selecting *Custom* re-enables both fields for free entry.

Section 3 — Simulation Mode. An `optionMenuGrp` controls the overall simulation behavior. Switching modes triggers the `bfUpdateMode` callback, which shows or hides Sections 4 and 7: Section 4 (Path Settings) is visible only in *Path Following* mode; Section 7 (Swarm Settings) is visible only in *Swarm* mode.

Section 4 — Path Settings (conditional). A second `textFieldButtonGrp` with a *Select* button lets the user pick an existing NURBS curve from the Maya scene to serve as the flight path. A `floatFieldGrp` sets the path-following strength (0–1).

Section 5 — Wind. A `checkBoxGrp` enables the wind subsystem. When checked, three `floatFieldGrp` controls for the wind direction vector (X, Y, Z) and a `floatSliderGrp` for magnitude become active. An additional checkbox enables time-varying (sinusoidal) wind. All five controls are disabled when wind is off, preventing irrelevant input.

Section 6 — Force Parameters. Six numeric controls expose the aerodynamic and vortex parameters that are otherwise fixed inside the C++ solver. Vortex gain in each axis ($gain_x = 22.0$, $gain_y = 5.5$, $gain_z = 22.0$) and the curl-noise turbulence scale $\eta = 3.66$ are pre-set to the Monarch defaults from the paper and are editable for fine-tuning. Maximum flight speed (m/s) and sliding-window size k (default 10 frames) complete this section.

Section 7 — Swarm Settings (conditional). Three `intFieldGrp` / `floatFieldGrp` controls set the number of simulated agents, their spatial spread at initialization (in metres), and the inter-agent repulsion radius used to keep butterflies from overlapping.

Section 8 — Output Settings. Integer fields for start frame and simulation duration (in frames), an integer field for frame rate, and an `optionMenuGrp` for export format (*Maya Keyframes*, *Alembic (.abc)*, *FBX (.fbx)*) define how the baked animation will be saved.

Action buttons. Three full-width buttons occupy the bottom row:

- **Simulate** (green): Collects all parameters, validates the rig field, and issues the C++ plug-in command to run the simulation for the specified duration.
- **Bake to Keys** (blue): Calls `bakeResults` on the entire rig hierarchy across the simulated frame range, writing standard Maya keyframe curves. If an Alembic or FBX export format is selected, the appropriate export command is also triggered.
- **Reset** (red): Restores all controls to their default values, clears the rig and path fields, and re-applies the Monarch species preset.

1.4.2 User Tasks

The following operations are available through the ButterFlight panel. Each corresponds to one or more GUI controls described in Section 1.4.1.

- **Assign Rig.** The user selects the `BF_thorax` root joint in the Maya viewport and clicks the *Select* button in Section 1. The plug-in validates the naming prefix and records the joint handle. This operation must be performed before any simulation can run.
- **Choose Species Preset.** The user picks *Monarch*, *Swallowtail*, or *Custom* from the Section 2 drop-down. Named species auto-fill body mass and wing area with paper-derived values and lock those fields; *Custom* leaves both fields editable.
- **Select Simulation Mode.** The Section 3 drop-down switches between *Free Flight*, *Path Following*, and *Swarm* modes. The selection controls which additional sections (4 or 7) are shown and which simulation code path is activated on *Simulate*.
- **Set Path Curve (Path Following mode only).** The user selects an existing NURBS curve in the viewport and clicks *Select* in Section 4 to designate it as the flight path. A follow-strength slider controls how tightly the butterfly tracks the curve.
- **Enable and Configure Wind.** Checking the *Enable Wind* box in Section 5 activates direction and magnitude controls. The user sets a world-space direction vector and a magnitude (m/s), and can optionally enable time-varying sinusoidal wind to simulate gusts.
- **Tune Force Parameters.** Section 6 exposes the six low-level numerical parameters of the simulation. Expert users or TDs may adjust vortex gains ($gain_{x,y,z}$), the curl-noise scale η, maximum flight speed, and the sliding-window size k to achieve different flight characters.
- **Configure Swarm (Swarm mode only).** Section 7 lets the user specify the number of agents, their spatial spread radius at spawn, and the inter-agent repulsion radius to prevent overlap.
- **Set Output Settings.** Section 8 defines the simulation time range (start frame, duration in frames, and frame rate) and the desired export format (Maya keyframes, Alembic, or FBX).

- **Simulate.** Clicking the green *Simulate* button collects all parameters, validates the rig assignment, and calls the C++ plug-in command to run the physics simulation. The resulting joint trajectories are previewed in the Maya viewport but not yet committed as keyframes.
- **Bake to Keyframes.** Clicking *Bake to Keys* converts the live simulation into standard Maya `animCurve` nodes on every rig joint across the specified frame range. If Alembic or FBX is selected, the appropriate export command is triggered immediately after baking.
- **Reset.** The red *Reset* button clears all input fields and restores every control to its factory default, including re-applying the Monarch species preset and hiding the conditional sections.

Prior knowledge required. Users must be comfortable with standard Maya operations: importing and referencing scene assets, selecting and naming joints, creating NURBS curves with the *EP Curve* or *CV Curve* tools, and exporting scenes via Alembic or FBX. No knowledge of aerodynamics, numerical integration, or MEL/Python programming is required to operate the tool; the force parameters in Section 6 carry sensible species-tuned defaults and need not be touched for typical use.

1.4.3 Work Flow

Figure 5 shows the complete ButterFlight workflow as a flowchart. The user sets up the scene and parameters, runs the simulation, then bakes and optionally exports the final animation. The steps below describe a typical single-butterfly session; the Swarm workflow differs only in Section 7 being visible and the bake step writing curves for all agents.

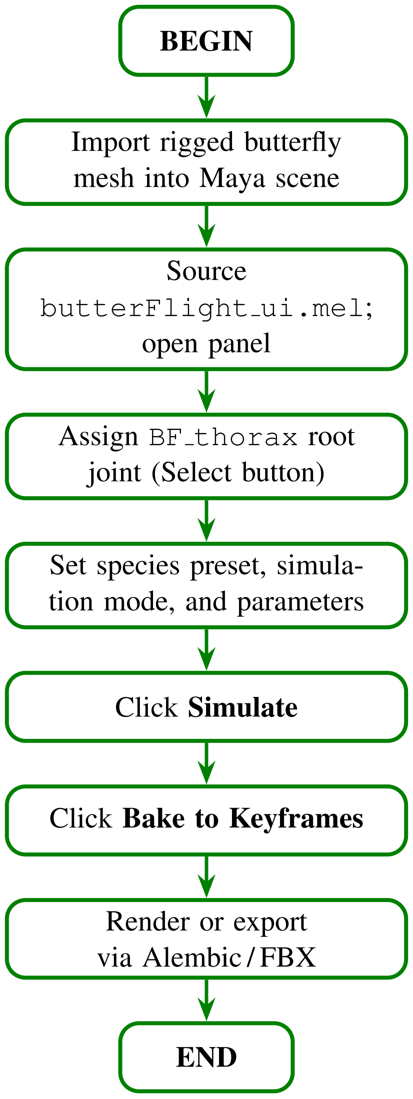

BEGIN
Import rigged butterfly mesh into Maya scene
Source butterFlight_ui.mel; open panel
Assign BF_thorax root joint (Select button)
Set species preset, simulation mode, and parameters
Click Simulate
Click Bake to Keyframes
Render or export via Alembic / FBX
END

Figure 5: ButterFlight workflow. After opening the panel and assigning a rig, the user configures species, mode, and simulation parameters, runs the simulation, then bakes the result to standard Maya keyframes and optionally triggers an Alembic or FBX export.

2. Authoring Tool Development

2.1 Technical Approach

2.1.1 Algorithm Details

The simulation is organized into three inter-connected modules that mirror the structure of Chen et al. 2022.

Module 1: Butterfly Modeling (paper Section 4).

The butterfly body is represented by a hierarchical skeleton of nine joints. The **thorax** is the root and has one degree of freedom (DOF): pitch angle θ_β. Each **fore-wing** (left and right) has three DOFs: flap angle θ_γ, feather angle θ_ζ, and sweep angle θ_ψ. Each **hind-wing** has one DOF: flap angle θ_γ only, because its aerodynamic contribution is secondary and its motion closely tracks the fore-wings. The **abdomen** has one DOF: rotation θ_ϕ along the body longitudinal axis. In total the model has five independently parameterized angles, collected as $\chi = \{\theta_\beta, \theta_\gamma, \theta_\zeta, \theta_\psi, \theta_\phi\}$.

All five maneuvering angles are computed via the **maneuvering function** (Equation 1 of the paper):

$$\theta^*\big(\varphi_a^*(\mathbf{u}),\, f^*(\mathbf{u}),\, \varphi_p^*,\, \varphi_m^*,\, t\big) \;=\; \varphi_a^*(\mathbf{u})\cos\!\big(2\pi f^*(\mathbf{u})\, t + \varphi_p^*\big) + \varphi_m^*, \quad * \in \{\beta, \gamma, \zeta, \psi, \phi\}$$

where $\mathbf{u}$ is the current butterfly velocity, φ_a^* is the amplitude, f^* is the frequency, φ_p^* is the phase offset, and φ_m^* is the mean angle. Both f^* and φ_a^* are sigmoid functions of speed (Equations 2–3):

$$f^*\big(\mathbf{u}(t_0)\big) = \frac{R_f^*}{1 + e^{-16\,(\,|\mathbf{u}(t_0)|/|\mathbf{u}_{\max}| - 0.5\,)}}, \qquad \varphi_a^*\big(\mathbf{u}(t_0)\big) = \frac{R_{\theta_a}^*}{1 + e^{-16\,(\,|\mathbf{u}(t_0)|/|\mathbf{u}_{\max}| - 0.5\,)}}$$

so that both frequency and amplitude increase smoothly with speed. The phase offsets φ_p^* are 0° for the fore-wing and hind-wing flap angles, -180° for the abdomen (to reproduce the counter-phase oscillation observed in real Monarch butterflies [12]), and -90° for the thorax. Frequency and amplitude are held constant within one flapping cycle ($t_0 \to t_1$) and updated at each cycle boundary using a sliding-window smoother (Module 3 below). The parameter ranges used in our implementation are taken directly from Table 3 of the paper and are listed in Table 2.

Angle	R_f^* (Hz)	$R_{\theta_a}^*$ (°)	φ_p^* (°)	φ_m^* (°)
θ_β	0–3	0–30	−90	0
θ_γ	0–11	0–150	0	10
θ_ζ	0–11	0–10	−90	0
θ_ψ	0–11	0–20	−90	0
θ_ϕ	0–11	0–35	−180	−10

Table 2: Maneuvering angle parameter ranges from Chen et al. 2022, Table 3.

Module 2: Forces Computation (paper Section 5).

Two forces act on the butterfly at each time step.

(a) Simplified Aerodynamic Force (Section 5.1). Following the quasi-steady theory of Ellington (1984a) [2], the lift and drag forces on the i-th polygon of wing j are:

$$\mathbf{F}_{i,\text{lift}} = \tfrac{1}{2}\rho A_i |\mathbf{V}|^2 C_l(\alpha), \qquad \mathbf{F}_{i,\text{drag}} = \tfrac{1}{2}\rho A_i |\mathbf{V}|^2 C_d(\alpha),$$

where ρ is air density, A_i is the polygon area, and $\mathbf{V}$ is the air velocity over the wing surface. The local angle of attack is $\alpha = \arctan(|\mathbf{V}_n|/|\mathbf{V}_t|)$, with $\mathbf{V}_n$ the normal component and $\mathbf{V}_t$ the tangential (base-to-tip) component. The empirical lift and drag coefficient polynomials proposed by the paper are:

$$C_l(\alpha) = -0.0095953\alpha^2 + 0.090635\alpha - 0.34182,$$

$$C_d(\alpha) = -0.0000079518\alpha^3 + 0.0011527\alpha^2 + 0.0063148\alpha + 0.51127.$$

The total aerodynamic force on wing j is the polygon sum $\mathbf{F}_j = \sum_i \mathbf{F}_{i,j}$, and all four wings contribute to the resultant $\sum_{j=1}^{4} \mathbf{F}_j$.

(b) Curl-Noise Vortex Force (Section 5.2). To reproduce the inherently chaotic, vortex-driven trajectory noise of real butterfly flight, a vortex force is computed from a curl-noise field (Bridson et al. 2007 [8]) seeded with Perlin noise (Perlin 2002 [6]):

$$\mathbf{F}_{\text{vor}} = \nabla \times \left(\tfrac{s_1(\mathbf{p})}{gain_x}, \tfrac{s_2(\mathbf{p})}{gain_y}, \tfrac{s_3(\mathbf{p})}{gain_z} \right) \cdot \eta,$$

where $\mathbf{p}$ is the center of mass of the thorax, s_1, s_2, s_3 are Perlin noise values at $\mathbf{p}$ with different seeds, and η scales the force magnitude. The paper sets $gain_x = gain_z = 22.0$, $gain_y = 5.5$, and $\eta = 3.66$ empirically. Smaller gain values produce tighter vortices; larger values produce broader, gentler noise. Crucially, $\mathbf{F}_{\text{vor}}$ is applied *only* to the thorax center of mass, not to individual wing polygons, to avoid excessive uncontrolled wing twisting.

Module 3: Maneuvering Control (paper Section 6).

Body translation and posture are decoupled. Velocity is computed from the composite of all forces acting on the butterfly, while posture angles are updated separately via the maneuvering functions above.

(a) Velocity Computation (Section 6.1). If the butterfly is targeting a point $\mathbf{q}_i$ (path keypoint or attraction target), a preferred acceleration is computed from a vision-based ramp function:

$$\mathbf{a}_{\text{pre}} = \frac{R(d)}{m} \cdot \frac{\mathbf{p} - \mathbf{q}_i}{|\mathbf{p} - \mathbf{q}_i|}, \qquad d = \min\!\left(1, \frac{|\mathbf{p} - \mathbf{q}_i|}{L}\right),$$

where m is the butterfly mass and $L = 4.5$ is the field-of-view sensing length. The local acceleration from the force model is:

$$\mathbf{a}_{\text{loc}} = \left(\sum_{j=1}^{4} \mathbf{F}_j + \mathbf{F}_{\text{vor}} + \mathbf{g}\right)/m,$$

and the velocity at time step t is updated as:

$$\mathbf{u}_t = \mathbf{u}_{t-1} + (\mathbf{a}_{\text{loc}} + \mathbf{a}_{\text{pre}})\,\Delta t.$$

(b) Maneuvering Update (Section 6.2). Between flapping cycles, both frequency f^* and amplitude φ_a^* are smoothed by a sliding-window average to prevent abrupt transitions:

$$c_{n+1}^* = 0.5 \sum_{i=\max(n-k,1)}^{n-1} w_i\, c_i^* \;+\; 0.5\, c_n^*, \quad * \in \{f, \varphi_a\}, \quad \sum w_i = 1,$$

where c_n^* is the parameter value at flapping cycle n (computed from Equations 2–3), and $k = 10$ is the sliding window size.

Assumptions and Simplifications.

- **Synchronous bilateral wings.** Left and right wings (fore and hind) are assumed to flap with the same frequency and amplitude at all times. This is consistent with biological observations for normal forward flight but precludes banked turning simulation.
- **Fixed parameters within one flapping cycle.** Both f^* and φ_a^* are held constant from t_0 to t_1 and only updated at cycle boundaries. This simplifies the integration and avoids intra-cycle discontinuities, at the cost of slightly reduced responsiveness to instantaneous velocity changes.
- **Quasi-steady aerodynamics.** We use the quasi-steady thin-wing approximation rather than a full Navier-Stokes/CFD solver. This is orders of magnitude faster but ignores leading-edge vortex dynamics and unsteady wake interactions between wing strokes.
- **Empirical polynomial coefficients.** The lift and drag coefficient curves $C_l(\alpha)$ and $C_d(\alpha)$ are fitted from experimental data and applied uniformly across both species. Species-specific coefficients are not available in the paper and are not modeled.
- **Vortex force at thorax CoM only.** Applying $\mathbf{F}_{\text{vor}}$ to the thorax rather than individual wing polygons avoids uncontrolled twisting. The vortex force therefore influences body trajectory and through-velocity also affects wing frequency and amplitude, but does not directly drive wing deformation.
- **Maya joint system instead of Unity Dynamic Bone.** The paper's original implementation uses Unity's Dynamic Bone component for skeleton spring deformation. In ButterFlight we replicate the kinematic joint angles via direct `setAttr` and `setKeyframe` calls on Maya joints; no spring physics are applied to the skeleton itself.
- **No takeoff or landing.** The simulation assumes the butterfly is always in steady forward or hovering flight. Takeoff momentum from leg jumping and landing coordination are outside the scope of this implementation.

2.1.2 Maya Interface and Integration

ButterFlight is split across two implementation layers: a **MEL scripting layer** responsible for all user-facing interaction, and a **C++ plug-in layer** (`butterFlight.mll`) that performs the computationally intensive simulation and keyframe writing. The two layers communicate through a single Maya command registered by the plug-in.

MEL / Python layer (`butterFlight_ui.mel`). All GUI elements are created with standard MEL UI commands (`window`, `frameLayout`, `columnLayout`, `optionMenuGrp`, `floatFieldGrp`, `button`, etc.). The MEL layer has no simulation logic; its only responsibilities are:

- Constructing and displaying the panel window.
- Validating user inputs (joint prefix check, empty-field guards).
- Collecting parameter values from all controls and forwarding them as flags to the C++ command `butterFlightSimulate`.
- Calling `bakeResults` on the rig hierarchy after simulation, and optionally invoking `AbcExport` or the FBX plug-in for export.
- Providing the *Reset* function that restores all control defaults.

Scene-setup utilities (e.g., checking that the required `BF_` joints exist, querying NURBS curve CVs for path keypoints) may be implemented in Python using the `maya.cmds` API for easier string handling.

C++ plug-in layer (`butterFlight.mll`). The plug-in is built as a Maya API plug-in using the Maya 2026 C++ SDK and registered via the standard `MFnPlugin` mechanism. It exposes one custom `MPxCommand` subclass:

- **`BFSimulateCmd` (subclass of `MPxCommand`)**. Parses all simulation flags (rig root, mode, species parameters, force parameters, frame range), runs the per-frame simulation loop, and writes joint rotations and root translations directly to the scene using `MFnAnimCurve` and `MAnimControl`. The command is undoable: it caches the pre-existing animation curves and restores them in its `undoIt()` method.

The key Maya API classes used by the plug-in are summarised below:

Maya API class	**Role in ButterFlight**
`MPxCommand`	Base class for `BFSimulateCmd`; provides undo support
`MFnDagNode`	Traverses the joint hierarchy from `BF_thorax`
`MFnIkJoint`	Reads and sets joint rotation attributes (θ_β, θ_γ, etc.)
`MFnTransform`	Sets world-space root translation at each simulated frame
`MFnAnimCurve`	Creates and writes `animCurveTA/TL` nodes for baked output
`MFnNurbsCurve`	Samples CVs of the user-specified path curve for Path Following mode
`MAnimControl`	Queries current frame rate and time unit for Δt computation
`MVector / MPoint`	Stores velocity $\mathbf{u}$, position $\mathbf{p}$, and force vectors
`MArgDatabase`	Parses command-line flags passed from the MEL layer

Table 3: Maya API classes used by `butterFlight.mll`.

The simulation loop inside `BFSimulateCmd::doIt()` iterates over every frame in the requested range. At each frame it: (1) evaluates the five maneuvering angles using Module 1 equations; (2) computes per-polygon aerodynamic forces by querying wing mesh normals via `MFnMesh`; (3) evaluates the curl-noise vortex force at the thorax CoM; (4) integrates velocity via

Newton's second law; (5) updates joint rotations and root position; and (6) writes one keyframe per joint via `MFnAnimCurve::addKey`.

2.1.3 Software Design and Development

ButterFlight follows a three-tier architecture that separates user-facing controls, simulation logic, and Maya scene manipulation into distinct layers (Figure 6).

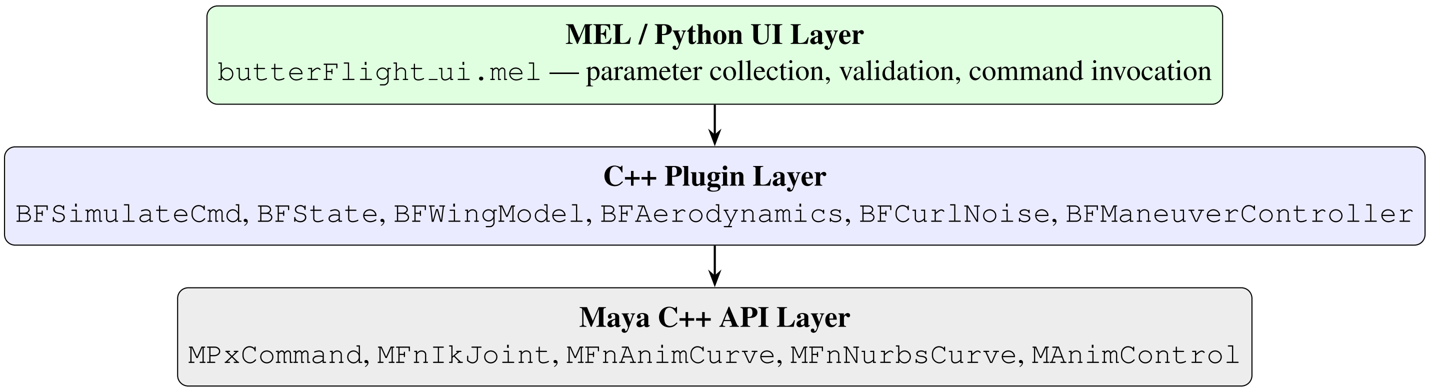

MEL / Python UI Layer
butterFlight_ui.mel — parameter collection, validation, command invocation
C++ Plugin Layer
BFSimulateCmd, BFState, BFWingModel, BFAerodynamics, BFCurlNoise, BFManeuverController
Maya C++ API Layer
MPxCommand, MFnIkJoint, MFnAnimCurve, MFnNurbsCurve, MAnimControl

Figure 6: ButterFlight three-tier software architecture.

Class hierarchy. The C++ plugin layer is organized around six classes:

- **BFSimulateCmd** (`MPxCommand` subclass) — Entry point registered with Maya as the `bfSimulate` command. Parses `MArgDatabase` flags, orchestrates the per-frame simulation loop, and implements `undoIt`/`redoIt` by caching original keyframe data before any curves are written.
- **BFState** (plain struct) — Per-butterfly value aggregate: world-space position (`MPoint`), velocity (`MVector`), the nine joint-angle values θ^*, phase accumulator t, and a circular buffer of the last $k = 10$ control-point positions used by the sliding-window smoother.
- **BFWingModel** — Evaluates Equations 1–3 each frame to compute target joint angles given the current maneuvering inputs. Stores species-specific parameters: mass, wing area, nominal flapping frequency, amplitude, and phase offsets for each joint.
- **BFAerodynamics** — Computes quasi-steady lift and drag forces (Equations 4–6) for each of the four wings using the angle of attack between the wing-panel normal and the local velocity vector relative to the surrounding air.
- **BFCurlNoise** — Generates the curl of a 3-D Perlin noise field (Equation 7) at a query position using a finite-difference curl approximation. Applies the species-independent gain and η constants to produce the vortex force applied to the thorax centre of mass.
- **BFManeuverController** — Implements the preferred-acceleration and local-acceleration decoupling (Equations 8–11) and the sliding-window smoother (Equation 12) that drives the body trajectory along a path or swarm target.

Key data flow. `BFSimulateCmd::doIt` initialises a `BFState` at the start frame, then iterates frame by frame. Each iteration calls `BFWingModel::update` → `BFAerodynamics::computeForc`
→ `BFCurlNoise::eval` → `BFManeuverController::step`, accumulates the resulting net acceleration into `BFState::velocity` and `position`, and finally writes joint rotations and root position to Maya animation curves via `MFnAnimCurve::addKey`.

Third-party dependencies.

- **Maya 2026 C++ devkit** (Autodesk, included with Maya 2026) — provides all `MFn*` and `MPx*` headers and import libraries.
- **No additional runtime libraries** — all vector and matrix arithmetic is performed with Maya's built-in `MVector`, `MPoint`, and `MMatrix` types; no external linear-algebra library (e.g., Eigen) is required.
- **CMake 3.25+** (build system only) — used to locate the Maya devkit and generate the Visual Studio solution; not required at runtime.

2.2 Target Platforms

2.2.1 Hardware

Table 4 lists the minimum and recommended hardware configurations for running ButterFlight inside Autodesk Maya 2026 on Windows.

Table 4: Hardware requirements.

Component	**Minimum**	**Recommended**
Processor	Intel Core i5 (8th gen) or AMD Ryzen 5 3600, 2.5 GHz+, 4 cores	Intel Core i7/i9 or AMD Ryzen 7/9, 3.0 GHz+, 8+ cores
RAM	8 GB	16 GB or more
GPU	Any GPU supported by Maya 2026 Viewport 2.0	NVIDIA RTX or AMD RDNA GPU with 4 GB+ VRAM
Storage	10 GB free disk space	SSD with 20 GB free
Display	1920×1080	2560×1440 or higher

The ButterFlight simulation itself is a CPU-bound, single-threaded computation; no GPU is used by the plug-in directly. However, Maya's Viewport 2.0 renderer leverages the GPU for scene display, so a capable GPU meaningfully reduces total iteration time when previewing animations.

2.2.2 Software

Table 5 lists all required and optional software for building and running ButterFlight.

Table 5: Software requirements.

Software	Required Version	Notes
Operating System	Windows 10 64-bit (v1903+) or Windows 11	No macOS or Linux support at this time
Autodesk Maya	Maya 2026	Plug-in compiled against the Maya 2026 devkit
C++ Compiler	Visual Studio 2022 (v17.x), MSVC 14.3+	Desktop Development with C++ workload required
Windows SDK	10.0.19041.0 or later	Included in the VS 2022 installer
Maya devkit	Maya 2026 Developer Kit	Required to build the `.mll` plug-in
Build system	CMake 3.25 or later	Generates the Visual Studio solution; build-time only
VC++ Redistributable	Microsoft VC++ 2022 Redistributable (x64)	Bundled with Maya 2026; not needed separately
FBX plug-in	Bundled with Maya 2026	For keyframe animation export
Alembic I/O plug-in	Bundled with Maya 2026	Optional; for point-cache export

ButterFlight is compiled as a Maya plug-in (`.mll`) targeting Maya 2026 on Windows. The MEL script layer (`butterFlight_ui.mel`) runs inside Maya's embedded MEL interpreter and requires no separate Python or scripting-runtime installation. All build instructions assume Visual Studio 2022 with the Desktop Development with C++ workload; CMake locates the Maya devkit headers and generates the Visual Studio project automatically.

2.3 Software Versions

2.3.1 Alpha Version Features (first prototype)

The alpha release establishes the core single-butterfly simulation pipeline with path-following locomotion. It targets the milestone date of **March 25**.

Included features.

- **Single butterfly, path-follow mode** — the user selects a NURBS curve as the flight path; the butterfly's thorax centre of mass tracks the curve while maneuvering control (Module 3) handles orientation and speed.
- **Default species model (Monarch preset)** — fixed mass $0.428\,\text{g}$, wing area $26\,\text{cm}^2$, and nominal flapping parameters; no custom-species input in this version.

- **Full wing kinematics (Module 1)** — all nine joints animated per frame using the maneuvering function (Eq. 1) with sigmoid frequency and amplitude (Eqs. 2–3).
- **Quasi-steady aerodynamics (Module 2)** — lift and drag computed each frame from wing-panel normals and local velocity; curl-noise vortex force applied to the thorax to produce natural trajectory noise.
- **Bake to Keyframes** — simulation output written to Maya animation curves on the rig joints, playable without the plug-in loaded.
- **Basic MEL UI panel** — Rig Assignment, Path Settings, Force Parameters, and Output Settings sections; Simulate and Bake to Keys buttons.

Alpha demo scene. The alpha demo consists of a rigged Monarch butterfly mesh and a gently curving NURBS path placed above a flat ground plane. Running *Simulate* produces roughly 200 frames of path-following flight with visible wing-abdomen coupling and aerodynamic trajectory noise. The scene ships as `scenes/alpha_demo.mb` and requires only the `.mll` plug-in and `butterFlight_ui.mel` to reproduce.

2.3.2 Beta Version Features

The beta release extends the alpha with wind simulation, swarm locomotion, and custom species support. It targets the milestone date of **April 15** and delivers a feature-complete tool ahead of the final demo.

New features added upon alpha.

- **Wind (curl-noise vortex)** — a Wind section in the UI exposes per-axis gain and the η scale factor; enabling wind applies `BFCurlNoise` forces each frame, producing visibly erratic, turbulent trajectories that vary between simulation runs.
- **Swarm mode** — the user specifies a butterfly count (2–20) and an aggregation centre point; each butterfly receives its own `BFState` and an individual preferred-acceleration target derived from the swarm centroid, producing loosely coordinated flock behaviour.
- **Alembic / FBX export integration** — the Bake to Keys step is complemented by an *Export Cache* button that fires Maya's built-in Alembic or FBX export after baking, writing a self-contained cache file.
- **Full MEL UI panel** — the Species Preset, Wind, and Swarm Settings sections (hidden in alpha) become active; mode-switching between Path and Swarm dynamically shows or hides the relevant control groups.

Beta demo scene. The beta demo provides two scenes:

1. `scenes/beta_wind_demo.mb` — a single Monarch on a curved path with wind enabled; the trajectory deviates noticeably from the path at moderate gain values, illustrating the vortex-force contribution.

2. `scenes/beta_swarm_demo.mb` — five Monarchs initialised at random offsets around a shared aggregation point, simulated for 300 frames to show loose cohesion and individual wing-motion variation.

2.3.3 Description of Demos and Tutorials

The final release ships with three ready-to-open Maya scene files and an in-panel *Quick-Start* guide, all located in the `scenes/` and `doc/` directories of the repository.

Demo scenes.

- **Alpha path-follow demo** (`scenes/alpha_demo.mb`) — a Monarch butterfly following a curved NURBS path; demonstrates baseline wing kinematics, abdomen coupling, and quasi-steady aerodynamic noise with no wind. Intended as the simplest entry point for first-time users.
- **Wind turbulence demo** (`scenes/beta_wind_demo.mb`) — the same path scene with wind enabled at moderate gain values. Highlights how curl-noise vortex forces perturb the trajectory while the maneuvering controller continues to track the path. Useful for comparing wind-on versus wind-off output.
- **Swarm demo** (`scenes/beta_swarm_demo.mb`) — five Monarchs aggregating around a shared centroid point over 300 frames. Demonstrates per-agent `BFState` independence, loose cohesion, and natural variation in individual flapping phase.

Quick-Start tutorial. A *Quick-Start* PDF (`doc/QuickStart.pdf`) walks a new user through the complete workflow in ten annotated steps, mirroring the Work Flow described in Section 1.4.3.

3. Work Plan

3.1 Tasks and Subtasks

Task 1 — Environment Setup and Maya Plugin Scaffolding
Duration: 1 week **Assigned:** Cecilia Chen, Yiding Tian

This task establishes the build environment and the minimal plugin skeleton required by all subsequent tasks. It covers configuring CMake to locate the Maya 2026 devkit, producing a loadable `.mll` file, and verifying the MEL-to-C++ command round-trip. Completing this task unblocks parallel development of the simulation modules.

- **Subtask 1.1 — CMake Build Configuration**: Configure `CMakeLists.txt` to locate the Maya 2026 devkit headers and import libraries, generate a Visual Studio 2022 solution, and produce a signed `butterFlight.mll` that loads cleanly in Maya's Plug-in Manager.
- **Subtask 1.2 — BFSimulateCmd Skeleton**: Implement and register a minimal `BFSimulateCmd` (MPxCommand subclass) that accepts a `-startFrame`/`-endFrame` flag pair and prints a confirmation string to the Script Editor without performing any simulation.
- **Subtask 1.3 — MEL Round-Trip Verification**: Write the opening `butterFlight_ui.mel` stub (window with a single Simulate button) and confirm that clicking it successfully invokes `bfSimulate` and reads the Script Editor output, proving the full MEL-to-C++ pipeline is functional.

Task 2 — Butterfly Skeleton Rig and Maneuvering Functions
Duration: 2 weeks **Assigned:** Yiding Tian

This task implements `BFWingModel` and the `BFState` data structure, which together form the kinematic backbone of the simulation. The maneuvering function (Eq. 1) and its sigmoid modifiers (Eqs. 2–3) must produce visually plausible wing motion for the Monarch preset before aerodynamic forces are added. Correct joint-rotation application via `MFnIkJoint` and `MFnTransform` is verified by scrubbing the timeline manually.

- **Subtask 2.1 — BFState Struct and Joint Enum**: Define the `BFState` plain-old-data struct (position, velocity, nine joint angles, phase accumulator, sliding-window circular buffer) and a `BFJoint` enum mapping the nine skeleton joints to array indices.
- **Subtask 2.2 — Maneuvering Function (Eqs. 1–3)**: Implement `BFWingModel::update`, which computes target joint angles θ^* each frame using the cosine maneuvering function and the sigmoid frequency and amplitude modifiers, seeded with the species-specific nominal values from Table 2.
- **Subtask 2.3 — Joint-Rotation Applicator**: Write the helper that retrieves each named joint (`BF_thorax`, `BF_forewing_L`, etc.) via `MFnDagNode` and sets its local rotation via `MFnTransform`, enforcing the correct rotation order (XYZ) to match the paper's DOF conventions.
- **Subtask 2.4 — Visual Verification**: Manually scrub a 60-frame preview with the Monarch

preset and confirm that forewing flap, feather, and sweep joints animate correctly, the hindwings follow in phase, and the abdomen rotates counter-phase to the thorax pitch.

Task 3 — Aerodynamic Force Model
Duration: 1.5 weeks **Assigned:** Cecilia Chen

This task implements `BFAerodynamics`, which computes per-wing quasi-steady lift and drag forces (Eqs. 4–6) each frame using the empirical polynomial coefficients from the paper. Wing-panel normals are derived from the joint world transforms rather than mesh face normals, avoiding a costly mesh query. The resulting forces are accumulated into `BFState::velocity` via Newton's second law and the per-frame timestep.

- **Subtask 3.1 — Wing-Panel Normal Computation**: Approximate each wing panel's outward normal from the cross product of the forewing joint's local X and Y axes in world space, providing the geometric input needed for angle-of-attack computation without querying `MFnMesh`.
- **Subtask 3.2 — Lift/Drag Polynomial Evaluation (Eqs. 4–6)**: Implement the $C_l(\alpha)$ and $C_d(\alpha)$ empirical polynomial evaluators and the lift/drag force equations using air density $\rho = 1.225\,\text{kg/m}^3$ and per-wing panel area from `BFWingModel`.
- **Subtask 3.3 — Newton Integration**: Accumulate net aerodynamic force across all four wings, divide by butterfly mass to obtain acceleration, and add it to `BFState::velocity` scaled by the per-frame timestep $\Delta t = 1/\text{fps}$.

Task 4 — Curl-Noise Vortex Force
Duration: 1 week **Assigned:** Cecilia Chen

This task implements `BFCurlNoise`, which generates turbulent vortex forces (Eq. 7) at the thorax centre of mass by taking the curl of a three-dimensional Perlin gradient noise field. The finite-difference curl approximation avoids the closed-form derivative, keeping the implementation straightforward. Gain constants (*gain_x* = *gain_z* = 22.0, *gain_y* = 5.5, $\eta = 3.66$) are hardcoded as defaults and exposed through the MEL UI Wind section.

- **Subtask 4.1 — 3-D Gradient Perlin Noise**: Implement a gradient-based Perlin noise function suitable for three-dimensional queries, using a permutation table and quintic interpolation to match the smoothness properties required by the curl approximation.
- **Subtask 4.2 — Finite-Difference Curl (Eq. 7)**: Compute $\nabla \times (s_1/g_x,\, s_2/g_y,\, s_3/g_z)$ via forward finite differences with step size $\epsilon = 0.01$, then scale by η to produce the vortex force vector applied to the thorax.
- **Subtask 4.3 — UI Integration**: Wire the Wind section of the MEL panel (*Enable Wind* checkbox and gain sliders) to flags on `bfSimulate`, and verify that toggling wind on and off produces noticeably different trajectory noise in a 100-frame test run.

Task 5 — Maneuvering Control and Body Motion Decoupling
Duration: 1.5 weeks **Assigned:** Yiding Tian

This task implements `BFManeuverController`, which decouples body locomotion from wing kinematics by computing a preferred acceleration toward the path or swarm target and a local aerodynamic correction (Eqs. 8–12). The sliding-window smoother ($k = 10$) is stored in the circular buffer inside `BFState` and is updated each frame before joint angles are applied. Correct path-following behaviour at this stage is the primary acceptance criterion for the alpha milestone.

- **Subtask 5.1 — Preferred Acceleration (Eq. 8)**: Implement the ramp function that computes preferred acceleration toward the next path CV or swarm centroid, clamped by the species maximum speed, and blended with a look-ahead distance to prevent overshooting sharp curves.
- **Subtask 5.2 — Local Acceleration Decoupling (Eqs. 10–11)**: Compute local aerodynamic acceleration from the net wing forces and subtract the component already accounted for by preferred acceleration, then integrate both into `BFState::velocity` and `position` per Eq. 11.
- **Subtask 5.3 — Sliding-Window Smoother (Eq. 12)**: Implement the $k = 10$ weighted circular buffer in `BFState` and apply the smoother to the body control-point position each frame, verifying that the resulting trajectory is smooth relative to a raw (unsmoothened) reference run.

Task 6 — MEL/Python UI Panel
Duration: 1 week **Assigned:** Cecilia Chen

This task completes `butterFlight_ui.mel` with all eight collapsible frameLayout sections, species-preset auto-fill, mode-dependent section visibility, and the three action buttons. The panel must correctly collect every parameter and forward it as a flag to the `bfSimulate` command, and the Bake to Keys button must invoke Maya's `bakeResults` on the correct joint set. This task also includes usability testing to ensure the panel behaves correctly on first load.

- **Subtask 6.1 — Full Control Layout**: Implement all eight frameLayout sections (Rig Assignment, Species Preset, Simulation Mode, Path Settings, Wind, Force Parameters, Swarm Settings, Output Settings) with correct control types, default values, and collapsible state.
- **Subtask 6.2 — Callbacks**: Implement `bfUpdateMode` (show/hide Path and Swarm sections), `bfUpdateSpecies` (auto-fill and lock fields for named presets, unlock for Custom), and `bfWindToggle` (enable/disable gain sliders).
- **Subtask 6.3 — Command Invocation and Bake**: Implement `bfSimulate` (collect all control values and call the C++ command with assembled flags) and `bfBakeKeyframes` (query the rig root, enumerate descendant joints, and call `bakeResults` over the simulation frame range).

Task 7 — Single Butterfly Path-Following Demo (Alpha)
Duration: 0.5 weeks **Assigned:** Cecilia Chen, Yiding Tian

This integration task assembles Tasks 1–6 into a working alpha build and creates the `alpha_demo.mb` scene. Any joint-rotation-order bugs, world-space transform errors, or flag-passing mismatches between MEL and C++ that appear during end-to-end testing are resolved here. Completion of this task marks the **Alpha milestone (March 25)**.

- **Subtask 7.1 — End-to-End Integration Test**: Load the plugin and MEL panel, assign a rigged Monarch mesh to a NURBS path, run Simulate for 200 frames, and confirm that joint animation curves are written correctly to the rig without Maya errors or crashes.
- **Subtask 7.2 — Integration Bug Fixes**: Resolve any discrepancies between expected and observed joint motion (rotation-order mismatches, incorrect world-to-local transforms, frame-offset errors) identified during Subtask 7.1.
- **Subtask 7.3 — Alpha Scene Packaging**: Save the verified scene as `scenes/alpha_demo.mb`, document the required plugin and MEL file in the README, and tag the Git repository at the alpha commit.

Task 8 — Swarm Simulation and Aggregation
Duration: 1.5 weeks **Assigned:** Yiding Tian

This task extends `BFSimulateCmd` to manage multiple independent `BFState` instances and computes per-agent preferred acceleration offsets toward the evolving swarm centroid. Each butterfly in the swarm runs the full Module 1–3 pipeline independently, so wing-motion variation arises naturally from different initial phase accumulators. The swarm count (2–20) and the aggregation target transform are exposed through the Swarm Settings section of the MEL UI.

- **Subtask 8.1 — Multi-Agent State Management**: Extend `BFSimulateCmd::doIt` to accept a `-count` flag, allocate a `BFState` vector of length N, and initialise each agent at a random position within a user-specified radius of the aggregation target.
- **Subtask 8.2 — Centroid-Based Preferred Acceleration**: Each frame, compute the current swarm centroid from all agent positions and inject it as the preferred-acceleration target for every agent via `BFManeuverController::step`, producing loose cohesion without explicit collision avoidance.
- **Subtask 8.3 — Swarm UI and Demo Scene**: Wire the butterfly count spinner and aggregation-target picker in the Swarm Settings section, add a `-swarmTarget` flag to `bfSimulate`, and create `scenes/beta_swarm_demo.mb` with five Monarchs for the beta milestone.

Task 9 — Export, Wind Polish, and Beta Integration
Duration: 0.5 weeks **Assigned:** Cecilia Chen

This task completes the beta feature set by adding the Alembic/FBX Export Cache button, finalising wind parameter round-tripping, and creating the `beta_wind_demo.mb` scene. It also includes a final UI polish pass (tooltip text, error messages for missing rig, and graceful handling of an empty path selection). Completion of this task marks the **Beta milestone (April 15)**.

- **Subtask 9.1 — Export Cache Button**: Add an *Export Cache* button to the Output Settings section that first calls `bfBakeKeyframes` and then invokes Maya's `AbcExport` or `FBXExport` command on the rig hierarchy, writing a timestamped cache file to the project's `cache/` directory.

- **Subtask 9.2 — Wind Parameter Round-Trip**: Verify that all Wind gain and η values entered in the MEL UI are correctly forwarded as `bfSimulate` flags, stored in `BFCurlNoise`, and produce reproducible output when the same seed is used across runs.
- **Subtask 9.3 — Beta Scene Packaging and Tag**: Save `scenes/beta_wind_demo.mb` demonstrating visible trajectory deviation under moderate wind, verify both beta demo scenes play back without the plugin loaded (baked curves only), and tag the Git repository at the beta commit.

Task 10 — Testing, Debugging, and Final Demo Scenes
Duration: 1 week **Assigned:** Cecilia Chen, Yiding Tian

The final task validates all three demo scenes against expected behaviour, profiles swarm performance for $N = 20$ butterflies, and produces the Quick-Start PDF and any remaining documentation. Any bugs surfaced by full-scene regression testing are triaged and fixed here. Completion of this task satisfies the **Final Demo (May 5)** and **Code Submission (May 11)** milestones.

- **Subtask 10.1 — Regression Testing**: Run all three demo scenes from a clean Maya session (plugin loaded fresh, no cached state) and verify that wing motion, trajectory noise, and baked curves match the expected output documented in Section 2.3.
- **Subtask 10.2 — Swarm Performance Profiling**: Simulate a $N = 20$ swarm for 300 frames and record per-frame compute time; if any frame exceeds 2 seconds on the recommended hardware configuration, profile the inner loop and apply targeted optimisations (e.g., precomputed normals, SIMD noise evaluation).
- **Subtask 10.3 — Quick-Start PDF and Code Submission**: Write the ten-step Quick-Start tutorial with annotated screenshots, export it as `doc/QuickStart.pdf`, verify the repository README includes build and load instructions, and submit the final codebase by May 11.

3.2 Milestones

3.2.1 Alpha Version

Target date: March 25. The alpha milestone delivers a single-butterfly, path-following simulation with full wing kinematics and aerodynamic forces. The following tasks and subtasks must be complete:

- **Task 1 (all subtasks)** — Build system configured, `.mll` loads in Maya's Plug-in Manager, and MEL-to-C++ round-trip verified.
- **Task 2 (all subtasks)** — `BFState` struct defined; `BFWingModel::update` animates all nine joints correctly for the Monarch preset; visual verification passed.
- **Task 3 (all subtasks)** — `BFAerodynamics` computes per-wing lift and drag each frame; Newton integration accumulates forces into velocity.

- **Task 4, Subtasks 4.1–4.2** — `BFCurlNoise` generates vortex force at the thorax CoM with correct gain and η constants. (UI exposure, Subtask 4.3, may be deferred to beta if time is tight.)
- **Task 5 (all subtasks)** — `BFManeuverController` implements preferred and local acceleration decoupling; sliding-window smoother ($k = 10$) produces a smooth path-following trajectory.
- **Task 6 (all subtasks)** — MEL panel with Rig Assignment, Path Settings, Force Parameters, and Output Settings sections; Simulate, Bake to Keys, and Reset buttons all functional.
- **Task 7 (all subtasks)** — End-to-end integration test passed; `scenes/alpha_demo.mb` created and Git repository tagged at the alpha commit.

Acceptance criteria. An evaluator opening `scenes/alpha_demo.mb`, loading the plugin and MEL panel, clicking *Simulate*, and then *Bake to Keyframes* should see: (1) a Monarch butterfly tracking a curved NURBS path for 200 frames; (2) all nine joints animated with visible wing-abdomen counter-phase coupling; (3) mild aerodynamic trajectory noise even in the absence of wind; and (4) fully baked animation curves on the rig joints that play back without the plugin loaded.

3.2.2 Beta Version

Target date: April 15. The beta milestone extends the alpha with wind simulation, swarm locomotion, custom species support, and export. All alpha tasks remain complete; the following additional tasks must also be finished:

- **Task 4, Subtask 4.3** — Wind section in the MEL panel wired to `BFCurlNoise`; enable/disable toggle and per-axis gain sliders functional.
- **Task 6 (beta UI additions)** — Swarm Settings section visible when Swarm mode is selected; Wind section active.
- **Task 8 (all subtasks)** — Multi-agent `BFState` management in `BFSimulateCmd`; centroid-based preferred acceleration for swarm cohesion; Swarm Settings UI wired; `scenes/beta_swarm_demo.mb` created.
- **Task 9 (all subtasks)** — Export Cache button invokes Alembic or FBX export after baking; wind parameter round-trip verified; `scenes/beta_wind_demo.mb` created; Git repository tagged at the beta commit.

Acceptance criteria. In addition to all alpha criteria, an evaluator should be able to: (1) enable Wind and observe visibly erratic trajectory deviation from the NURBS path in `scenes/beta_wind_demo.mb`; (2) open `scenes/beta_swarm_demo.mb`, run Simulate with five butterflies, and see loosely cohesive swarm behaviour with per-agent wing-motion variation; (3) click *Export Cache* and receive a valid `.abc` or `.fbx` file; and (4) select the Swallowtail preset and observe different mass and wing-area values auto-filled in the panel.

3.3 Schedule

4. Related Research

References

[1] CHEN Q., LU T., TONG Y., LUO G., JIN X., DENG Z.: A Practical Model for Realistic Butterfly Flight Simulation. *ACM Transactions on Graphics (TOG)* 1, 1 (2022), 12 pages.

[2] ELLINGTON C. P.: The aerodynamics of hovering insect flight. I. The quasi-steady analysis. *Philosophical Transactions of the Royal Society of London. B, Biological Sciences* 305, 1122 (1984), 1–15.

[3] ELLINGTON C. P.: The aerodynamics of hovering insect flight. V. A vortex theory. *Philosophical Transactions of the Royal Society of London. B, Biological Sciences* 305, 1122 (1984), 115–144.

[4] REYNOLDS C. W.: Flocks, herds and schools: A distributed behavioral model. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques* (1987), pp. 25–34.

[5] BETTS C. R., WOOTTON R. J.: Wing shape and flight behaviour in butterflies (Lepidoptera: Papilionoidea and Hesperioidea): a preliminary analysis. *Journal of Experimental Biology* 138, 1 (1988), 271–288.

[6] PERLIN K.: Improving noise. In *ACM Transactions on Graphics (TOG)*, Vol. 21. ACM, 2002, pp. 681–682.

[7] WU J., POPOVIĆ Z.: Realistic modeling of bird flight animations. *ACM Transactions on Graphics (TOG)* 22, 3 (2003), 888–895.

[8] BRIDSON R., HOURIHAM J., NORDENSTAM M.: Curl-noise for procedural fluid flow. *ACM Transactions on Graphics (TOG)* 26, 3 (2007), 46–es.

[9] WILSON T., ALBERTANI R.: Wing-flapping and abdomen actuation optimization for hovering in the butterfly *Idea leuconoe*. In *52nd Aerospace Sciences Meeting* (2014), 0009.

[10] WANG X., JIN X., DENG Z., ZHOU L.: Inherent noise-aware insect swarm simulation. In *Computer Graphics Forum*, Vol. 33. Wiley Online Library, 2014, pp. 51–62.

[11] WANG X., REN J., JIN X., MANOCHA D.: BSwarm: biologically-plausible dynamics model of insect swarms. *Proceedings of the 14th ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2015), 111–118.

[12] SRIDHAR M., KANG C.-K., LANDRUM D. B.: Instantaneous lift and motion characteristics of butterflies in free flight. In *46th AIAA Fluid Dynamics Conference* (2016), 3252.

[13] CHEN Q., LUO G., TONG Y., JIN X., DENG Z.: Shape-constrained flying insects animation. *Computer Animation and Virtual Worlds* 30, 3–4 (2019), e1902.

[14] SRIDHAR M., KANG C.-K., LEE T.: Geometric formulation for the dynamics of monarch butterfly with the effects of abdomen undulation. In *AIAA Scitech 2020 Forum* (2020), 1962.

[15] WILL H.: Dynamic Bone. Unity Asset Store, 2020. `https://assetstore.unity.com/packages/tools/animation/dynamic-bone-16743`.