Real-time Piloted Simulation using Rotorcraft Comprehensive Analysis with a Virtual Reality Interface

Ananth Sridharan *
Assistant Research Scientist
Alfred Gessow Rotorcraft Center
University of Maryland, College Park, MD

David Moy & Greg Rubenstein
Undergraduate Research Assistants
Alfred Gessow Rotorcraft Center
University of Maryland, College Park, MD

Michael Avera
Aerospace Engineer
U.S. Army Research Laboratory
Aberdeen Proving Ground, MD

ABSTRACT

The physics engines for pilot-in-the-loop helicopter simulators have traditionally been realized through successive stages of model simplification to achieve real-time execution speeds. In this paper, another approach is presented, where modern parallel computing paradigms are used to accelerate a high-fidelity rotorcraft comprehensive analysis using `OpenMP` and `CUDA-Fortran` to real-time speeds, without loss of accuracy. The elastic flap-lag-torsion rotor dynamics, flight mechanics, and the free wake model have been preserved in their original form without any simplifying assumptions. Such simulations are particularly relevant for slowed-rotor configurations that operate at high advance ratios, where legacy simulation techniques may not yield representative flight characteristics. The coupled blade dynamics and flight mechanics can be executed at speeds $90\times$ faster than the baseline implementation through a combination of algorithmic acceleration techniques and parallel computing. Graphics Processing Unit (GPU) computing is also leveraged to achieve $31\times$ faster simulations, crossing the threshold to qualify as real-time. For visualization, a virtual reality (VR) compatible pilot interface is integrated into an open-source framework that allows for modularity in choice of flight dynamics model and user interface.

NOMENCLATURE

Ω	Rotor rotational speed, rad/s
f	Governing rotor-body coupled equations
$\mathbf{J}$	Dual time stepping Jacobian
$\mathbf{u}$	Vector of pilot control inputs
$\mathbf{y}$	Vector of system states
$\dot{\mathbf{y}}$	Time derivative of state vector
α	Time stepping scheme parameter
Δt	Time step, seconds
ε	Perturbation value for state vector
θ_0	Main rotor collective pitch input, deg
θ_{1c}, θ_{1s}	Main rotor cyclic pitch inputs, deg
θ_{TR}	Tail rotor collective pitch input, deg
θ_1	Euler rotation angle about deformed x-axis
w', v'	Blade flap and lag bending slope

INTRODUCTION

A flight simulator is a vital tool that helps rotorcraft manufacturers and operators significantly reduce development and operating costs through virtual testing and pilot training/feedback. Previous work (Ref. 1) has identified the need for high-fidelity flight dynamic models of a helicopter, for accurate prediction of vehicle Handling Qualities ratings in a simulator. This requirement is of particular importance for new hybrid VTOL and high speed/slowed rotor configurations, where rotor-body couplings and reverse flow aerodynamics play a significant role in the vehicle response to pilot inputs.

Presently, piloted helicopter simulations are limited by a critical trade-off between accuracy and execution speed, i.e., to obtain rapid execution, the model fidelity must be reduced to lower the computing workload. Trade-offs between computational workload and model fidelity are assessed to determine whether a simulation is sufficiently representative of helicopter flight. This evaluation is particularly important (and potentially cumbersome) for new rotorcraft configurations such as coaxial helicopters with elastic blades and significant inter-rotor interference., The threshold of "minimum fidelity" must be repeatedly reassessed prior to creation of reduced-order models. This repeated cost (of selectively reducing model fidelity) may outweigh the long-term benefits of the approach for obtaining real-time simulation in a timely and efficient manner.

* ananth@umd.edu
Presented at the 43^{rd} European Rotorcraft Forum, Milan, Italy, September 12–15, 2017.

In this paper, an alternate approach is presented: the goal is to preserve the accuracy of the comprehensive analysis with elastic flap-lag-torsion blade dynamics, and systematically accelerate the simulation using modern parallel computing paradigms until real-time execution is achieved. Such a generalized approach takes into account all aspects of the elastic blade dynamics, flight dynamics and rotor-body couplings, and so accuracy is not compromised. Simulations developed in this manner are applicable to a greater class of rotary-wing platforms, because of fewer ad-hoc simplifications made in the construction of the mathematical model.

Previous Work

The requirements for a next-generation comprehensive rotorcraft analysis tool are laid out by Johnson (Ref. 2), along with a detailed background on the history of rotorcraft comprehensive analysis capabilities. Johnson noted, at the time, that real-time capability at the same fidelity level as the aeromechanics may not be achievable without reducing resolution of either the rotor structural dynamics or the aerodynamics. For next-generation high advance ratio VTOL platforms, a higher-fidelity physics model is necessary to accurately simulate vehicle response to pilot inputs. The modeling fidelity required to capture these physical effects requires computational power that exceeds traditionally available levels.

Even with the steady growth of computational power since 1990, the use of reduced-order models have been necessary (Refs. 3 – 6) to achieve real-time piloted simulations for rotorcraft. One such approach (Ref. 7) avoids executing the original physics-based model in real-time. Instead, frequency-domain models (linearized about various trim conditions) are "stitched" to obtain a representative flight dynamic model over a pre-determined envelope. The restrictions on this technique are that the flight envelope must be known apriori, and nonlinearities must be small. The assumptions used to construct these reduced-order model are valid for conventional VTOL that operate at low advance ratios, where linearized models approximate the aerodynamic loads with sufficient accuracy. A similar approach, but in the time domain, is given by Ref. 8, wherein linearized models of the rotor dynamics are constructed at various flight conditions. Hub load sensitivities to control inputs are also pre-computed and used to predict the vehicle response in free-flight.

The accurate prediction of helicopter wake interaction with a ship are also of great interest for both civilian and military applications. Though the present work is not explicitly geared towards ship airwake interactions, it may be applied in this domain. Recent strides by the U.S. government, academia as well as the industry have elevated the fidelity level of the aerodynamic interaction models. Notable examples are the works in Ref. 10 and Ref. 11, which used rigid blade representations of the rotor dynamics.

The multi-disciplinary nature of rotorcraft analysis requires consistent coupling of various aerodynamic and structural/flight dynamic models. Efficient implementation of each of these components requires a varied combination of programming languages and computing hardware to optimize execution speeds. For example, GPUs are attractive platforms to simulate N-body problems like free wake, while a modern multi-core CPU is capable of performing the calculations needed for flight and structural dynamics using beam representations of the rotor blade. The bottleneck for real-time high-fidelity simulation then lies in the communication overhead between hardware components, and translation across programming languages.

Previous work by the authors (Ref. 12) analyzed the breakdown of computational cost for real-time simulation with comprehensive analysis. Serial bottleneck alleviation strategies together with parallel computing were outlined for vehicle trim and time marching analysis. The present work aims to refine the flight simulator from Ref. 12 by improving the underlying physics model further, and coupling it to a more immersive pilot interface. The specific objectives are

Objectives

1. Assess requirements to achieve real-time simulation with original blade flap-lag-torsion dynamics
2. Obtaining a real-time free wake model with heterogeneous CPU-GPU computing
3. Coupling a VR pilot display to the real-time simulation framework

These objectives will be achieved without resorting to ad-hoc modifications or model simplifications to reduce computational workload, by leveraging modern parallel computing paradigms.

METHODOLOGY

Comprehensive Analysis

A rotorcraft comprehensive analysis was developed in-house at the University of Maryland's Alfred Gessow Rotorcraft Center (Refs. 12 and 13). This analysis is used as a basis for the present work to model the vehicle flight dynamics and rotor aeromechanics. The airframe, horizontal and vertical stabilizers are modeled as rigid bodies with table look-up aerodynamics. Rotor-body couplings are modeled using multibody-type rotations with exact kinematics. Rotor blades are modeled as geometrically exact Euler-Bernoulli beams with flap, lag and torsion dynamics. Tail rotor loads are computing using a disc model with uniform dynamic inflow. The Maryland Free-vortex Wake model (Ref 14) is coupled to the aero/flight mechanics solver and may be used to compute the rotor inflow. A modified implementation of the Peters-He dynamic inflow model (Ref 15) is integrated into the analysis, and is available for use in place of the free wake model. This simulation has been validated for single and multi-rotor configurations, and was recently used for CFD-CSD analysis of coaxial rotorcraft (Ref 16). For time marching, an implicit dual-step

integrator is implemented based on the work in Ref 17. Several acceleration strategies were combined with initial parallelization strategies to achieve real-time simulation in Ref. 12 with 6 rotor blade modes for a 4-bladed medium lift utility helicopter.

Solver organization

The equations of motion governing the system dynamics (except for the free-vortex wake model) are formulated in state-space form as a system of first-order nonlinear coupled Ordinary Differential Equations of the form

$$f(\mathbf{y}\,,\dot{\mathbf{y}}\,,\mathbf{u}\,,t) \quad = \quad \mathbf{0} \tag{1}$$

$\mathbf{y}$ is a vector of system states, $\mathbf{u}$ is a vector of control inputs and t is the current time in seconds. Numerical solutions of these equations provide representative solutions for vehicle maneuver response to pilot stick inputs. The state vector consists of the following components

$$\mathbf{y} \quad = \quad \left\{\mathbf{y}_{\mathrm{F}}^{\mathrm{T}} \quad \mathbf{y}_{\lambda}^{\mathrm{T}} \quad \mathbf{y}_{\mathrm{rotor}}^{\mathrm{T}}\right\}^{\mathrm{T}} \tag{2}$$

where $\mathbf{y}_{\mathrm{F}}$ is the vector of the 12 airframe rigid-body states, $\mathbf{y}_{\lambda}$ are the induced inflow coefficients for all rotors present in the system, and $\mathbf{y}_{\mathrm{rotor}}$ is the vector of rotor generalized deflection coordinates for all blades.

The vector of control inputs is

$$\mathbf{u} \quad = \quad \{\delta_0 \qquad \delta_{\mathrm{lat}} \qquad \delta_{\mathrm{lon}} \qquad \delta_{\mathrm{ped}}\}^{\mathrm{T}} \tag{3}$$

Joystick inputs are provided at every time step, re-scaled and used as the pilot inputs to the simulation. The four controls are manipulable by the helicopter pilot and represent, in order, the positions of the collective lever, lateral and longitudinal cyclic stick and the foot pedal.

A pilot exerts indirect control over the attitudes of the rotorcraft by flying the vehicle into the desired pitch and roll attitudes through manipulation of four *direct* control inputs. However, each rotor in the rotorcraft may have three types of pitch control, namely the lateral cyclic, longitudinal cyclic and collective inputs ($\theta_{1\mathrm{c}}$, $\theta_{1\mathrm{s}}$, θ_0). Internally, pilot inputs are converted to main and tail rotor swashplate pitch inputs using an offset (or bias) $\mathbf{b}$ and control mixing matrix $\mathbf{M}$ as follows

$$\{\theta_0 \qquad \theta_{1\mathrm{c}} \qquad \theta_{1\mathrm{s}} \qquad \theta_{\mathrm{TR}}\}^{\mathrm{T}} \quad = \quad \mathbf{b} + \mathbf{M}\mathbf{u} \tag{4}$$

The rotor control inputs are used, together with the current vehicle states, to evaluate the blade loads and residuals of the governing equations (Eqn. 1) for the coupled body dynamics and rotor aeromechanics. A dual step time integration scheme is used to advance the solution forward in time based on the pilot inputs, and is described in the following section.

Time marching

For time marching, an implicit dual-step integrator is implemented based on existing work in the literature (Ref. 17). Assuming that the controls are known during each time step (constant or linear extrapolation), the only unknowns are the values of the system states at the next time step. The coupled rotor-body dynamics cannot be easily recast into an explicit notation, i.e.,

$$[\mathbf{M}]\,\ddot{\mathbf{y}} + [\mathbf{C}]\,\dot{\mathbf{y}} + [\mathbf{K}]\,\mathbf{y} = \mathbf{F} \tag{5}$$

Such an explicit notation is cumbersome to obtain from semi-analytical expressions. Further, it may imposes restrictions on the type of problems that can be analyzed (e.g., small angles for both rotor blades and body attitudes). To preserve the generality of the approach, the system is locally linearized at each time step and advanced forward in time using dual time stepping as follows:

1. The initial condition is denoted by $\mathbf{y}_n$ and time step is Δt. Given a control input $\mathbf{u}$, the goal is to find the solution $\mathbf{y}_{n+1}$ after a time Δt has elapsed.

2. Generate a guess value of $\mathbf{y}_{n+1}$. Let this guess be $\mathbf{y}_{n+1}^{m}$.

3. Update $\mathbf{y}_{n+1}^{m}$ to $\mathbf{y}_{n+1}^{m+1}$ using sub-iterations. If $\mathbf{y}^{m+1}$ does not satisfy the governing equations, then repeat sub-iterations until the solution converges, i.e.

$$f(\mathbf{y}_{n+1}^{m+1}, \dot{\mathbf{y}}_{n+1}^{m+1}, \mathbf{u}, t) < 10^{-6} \tag{6}$$

The choice of time marching update scheme is detailed below. Rewriting the governing equations at time steps n and $n+1$, and including the sub-iteration index m, we obtain

$$f(\mathbf{y}_n, \quad \dot{\mathbf{y}}_n, \quad \mathbf{u}_{n-1}, \quad t) \;=\; 0 \tag{7}$$

$$f(\mathbf{y}_{n+1}^{m+1}, \quad \dot{\mathbf{y}}_{n+1}^{m+1}, \quad \mathbf{u}_n, \quad t) \;=\; 0 \tag{8}$$

The time derivative $\dot{\mathbf{y}}_{n+1}^{m+1}$ is approximated using an Euler implicit scheme as

$$\dot{\mathbf{y}}_{n+1}^{m+1} \quad \approx \quad \frac{\mathbf{y}_{n+1}^{m+1} - \mathbf{y}_n}{\Delta t} \tag{9}$$

From this point forward, the subscript $n+1$ is dropped when using the superscripts m, $m+1$, since sub-iterations are performed only for the solution at t_{n+1}. Expanding the residual function f in a Taylor series, we obtain

$$f^{m+1} \quad = \quad f^m + \frac{\partial f}{\partial \mathbf{y}}(\mathbf{y}^{m+1} - \mathbf{y}^m) + \frac{\partial f}{\partial \dot{\mathbf{y}}}(\dot{\mathbf{y}}^{m+1} - \dot{\mathbf{y}}^m) + \cdots \tag{10}$$

Substituting for the time derivative $\dot{\mathbf{y}}$ from Eqn. 9, it follows that

$$\dot{\mathbf{y}}^{m+1} \;-\; \dot{\mathbf{y}}^m \quad \approx \quad \frac{\mathbf{y}^{m+1} \;-\; \mathbf{y}^m}{\Delta t} \tag{11}$$

Therefore, the Taylor series approximation with linear terms is

$$f^{m+1} \approx f^m + \frac{\partial f}{\partial \mathbf{y}}(\mathbf{y}^{m+1} - \mathbf{y}^m) + \frac{\partial f}{\partial \dot{\mathbf{y}}}\frac{1}{\Delta t}(\mathbf{y}^{m+1} - \mathbf{y}^m) \quad (12)$$

$$= f^m + \left[\frac{\partial f}{\partial y} + \frac{1}{\Delta t}\frac{\partial f}{\partial \dot{\mathbf{y}}}\right](y^{m+1} - \mathbf{y}^m) \quad (13)$$

$$= f^m + \mathbf{J}(\mathbf{y}^{m+1} - \mathbf{y}^m) \quad (14)$$

If the solution at the next sub-iteration satisfies the governing equation, then $\mathrm{f}^{m+1} = \mathbf{0}$. Using this condition in Eqn. 14, we obtain

$$\mathbf{0} = f^m + \mathbf{J}(\mathbf{y}^{m+1} - \mathbf{y}^m) \quad (15)$$

The update scheme is then obtained as

$$\mathbf{y}^{m+1} = \mathbf{y}^m - \mathbf{J}^{-1} f^m \quad (16)$$

The sub-iteration update may also be interpreted as

$$\mathbf{y}^{m+1} = \mathbf{y}^m - \Delta\mathbf{y} \quad (17)$$

Where

$$\Delta\mathbf{y} = \mathbf{J}^{-1} f$$

Instead of inverting the matrix $\mathbf{J}$, a system of linear equations $\mathbf{J}\Delta\mathbf{y} = f$ is solved at every time step to determine the sub-iteration update $\Delta\mathbf{y}$. The concept can be extended to use more accurate multi-point backwards difference formulae for the time derivative $\dot{\mathbf{y}}^{m+1}$. The relevant derivative approximations are

$$\dot{\mathbf{y}}^{m+1} \approx \frac{\mathbf{y}_{n+1}^{m+1} - \mathbf{y}_n}{\Delta t} \quad (1^{st}\text{order}) \quad (18)$$

$$\approx \frac{3\mathbf{y}_{n+1}^{m+1} - 4\mathbf{y}_n + \mathbf{y}_{n-1}}{2\Delta t} \quad (2^{nd}\text{order}) \quad (19)$$

$$\approx \frac{11\mathbf{y}_{n+1}^{m+1} - 18\mathbf{y}_n + 9\mathbf{y}_{n-1} - 2\mathbf{y}_{n-2}}{6\Delta t} \quad (3^{rd}\text{order}) \quad (20)$$

The corresponding update Jacobians are given by

$$\mathbf{J} = \frac{\partial f}{\partial \mathbf{y}} + \frac{\alpha}{\Delta t}\frac{\partial f}{\partial \dot{\mathbf{y}}} \quad (21)$$

Where α assumes various values depending on the update scheme, given by

$$\alpha = 1.0 \quad \text{(first order)} \quad (22)$$

$$1.5 \quad \text{(second order)} \quad (23)$$

$$11/6 \quad \text{(third order)} \quad (24)$$

$\mathbf{J}$ represents the update Jacobian for sub-iterations. Usually, $\mathbf{J}$ is computed using the two-point forward difference formula using the initial condition $\mathbf{y}_n$ as a reference i.e.

$$\frac{\partial f}{\partial \mathbf{y}} \approx \frac{f(\mathbf{y}_n + \Delta\mathbf{y}, \dot{\mathbf{y}}_n, \mathbf{u}, t) - f(\mathbf{y}_n, \dot{\mathbf{y}}_n, \mathbf{u}, t)}{\Delta\mathbf{y}} \quad (25)$$

$$\frac{\partial f}{\partial \dot{\mathbf{y}}} \approx \frac{f(\mathbf{y}_n, \dot{\mathbf{y}}_n + \Delta\dot{\mathbf{y}}, \mathbf{u}, t) - f(\mathbf{y}_n, \dot{\mathbf{y}}_n, \mathbf{u}, t)}{\Delta\dot{\mathbf{y}}} \quad (26)$$

Accelerating Jacobian Calculations

The time marching Jacobian $\mathbf{J}$ is effectively a linearized representation of the rotor/blade dynamics, which consumes 90% of the total time during time integration. If the generation of the Jacobian is accelerated, then the run-times decrease proportionally.

Using Eqns. 25 and 26, the Jacobian may be computed by individually calculating the sensitivities of the residuals to the states and their first time derivatives, and then adding the two columns with appropriate scaling factors. A more efficient approach is adopted in Ref. 17, and that approach is used in the present work. Let $\mathbf{y}_o$ and $\dot{\mathbf{y}}_o$ represent baseline values of the system states and their corresponding time derivatives. Recognize that

$$f(\mathbf{y}_o + \varepsilon, \dot{\mathbf{y}}_o + \frac{\alpha}{\Delta t}\varepsilon, \mathbf{u}, t) = f(\mathbf{y}_o, \dot{\mathbf{y}}_o, \mathbf{u}, t) + \varepsilon\frac{\alpha}{\Delta t}\frac{\partial f}{\partial \dot{\mathbf{y}}} + \varepsilon\frac{\partial f}{\partial \mathbf{y}} + \cdots \quad (27)$$

Ignoring the higher-order terms, we obtain an expression for the update Jacobian $\mathbf{J}$ as

$$\mathbf{J} = \frac{1}{\varepsilon}\left[f(\mathbf{y}_o + \varepsilon, \dot{\mathbf{y}}_o + \frac{\alpha}{\Delta t}\varepsilon, \mathbf{u}, t) - f(\mathbf{y}_o, \dot{\mathbf{y}}_o, \mathbf{u}, t)\right] \quad (28)$$

From an implementation standpoint, each column of the Jacobian $\mathbf{J}$ is obtained as follows:

1. Perturb entry in row i of the state vector $\mathbf{y}$(i) by ε
2. Perturb the corresponding entry in the time derivate of the state vector $\dot{\mathbf{y}}(i)$ by $\alpha\varepsilon/\Delta$ t
3. Obtain the residuals of the governing equations
$$f(\mathbf{y}_o + \varepsilon, \dot{\mathbf{y}}_o + \frac{\alpha}{\Delta t}\varepsilon, \mathbf{u}, t)$$
4. Subtract the residuals of the baseline vector $f(\mathbf{y}_o, \dot{\mathbf{y}}_o, \mathbf{u}, t)$ and divide the resultant by ε
5. The resulting vector is the i^{th} column of the Jacobian matrix. Repeat for all i till the entire matrix $\mathbf{J}$ is populated

The generation of columns of the Jacobian in one step (instead of two steps, using Eqns. 25 and 26) reduces computational workload by 50%.

The strategies developed in previous work by the authors (Ref. 12) are adopted here to speed up the calculation of the Jacobian $\mathbf{J}$, and are summarized below.

To compute the columns of the Jacobian, it is advantageous to know a priori the nature of the system degrees of freedom. Based on the number of blades for which loads need to be computed for each of the perturbations, the states are classified into 3 types: Type 0 (no blade loads need to be computed), Type 1 (only loads from one blade are needed) and Type 2 (loads from all blades are affected). For example, vehicle position in space does not affect any of the blade loads, and so these three states are of Type 0. Any of the rotor generalized coordinates and their corresponding time derivatives

are of Type 1, because the loads from the other blades are unaffected directly by motion of any one blade. Finally, the rigid-body fuselage degrees of freedom and rotor inflow coefficients are of Type 2, because they affect the loads from all rotor blades. Based on this knowledge, the following measures are used

1. **Selective calculation of rotor blade loads:** The rotor blade loads calculations constitute the largest fraction of computation time (89% of total calculations). For the system of interest (4-bladed single main rotor helicopter), the effective number of rotor blade loads calculations is reduced by 60%, through selective update of blade loads depending on the Type of state (0, 1 or 2).

2. **Parallelized generation of Jacobian:** the columns of the Jacobian matrix **J** are generated in parallel using the OpenMP `PARALLEL DO` directive. For a 10-core processor, 20 parallel threads are concurrently used, resulting in a 2.9 $\times$ speed-up over serial execution.

3. **Load balancing:** The columns corresponding to each of the states are identified based on the Type of the state (0, 1 or 2). All columns of the Jacobian corresponding to Type 0 states are generated together, followed by all columns of **J** corresponding to Type 1 states, and then those dependent on Type 2 states. Thus, three sets of parallel sets are launched, one set after another. By changing the order in which columns of the Jacobian are generated, load balancing is achieved, because the same number of operations are executed by one parallel thread relative to another within a set. Load balancing results in a speed-up of 2.7 times over and above simple parallelization.

4. **Parallel calculation of blade loads**: During the sub-iteration update in Eqn. 16, the loads for all the rotor blades are required to compute the hub loads, beam equation residuals and rigid-body force and moment equilibrium residuals. Each time the sub-iterations are updated, the blade loads are computed in parallel using OpenMP `PARALLEL DO` directives. For a 4-bladed rotor, 3.5$\times$ acceleration is obtained.

   Through parallel execution of the blade dynamics and accelerated calculation of the time marching Jacobian **J**, 12$\times$ acceleration is achieved using 18 threads on a 10-core processor with hyper-threading.

5. **Small elastic deflections:** The comprehensive analysis is formulated using a geometrically exact beam model with trigonometric expressions to describe the orientations of the blade cross-sections due to simultaneous flap, lag and torsion. However, for most platforms (especially high-speed configurations with variable RPM) the blades are designed so that elastic deflections never exceed the "small-angle limit" (0.2 radians or 12 degrees). Thus, sines and cosines of the flap bending slope and lag bending slope (w', v') and their corresponding time derivatives may be optionally approximated using quadratic polynomials. In a practical situation, this simplification is activated by using pre-processing directives `#ifdef`. The advantage of this step is a significant reduction in clock cycle count, because trigonometric terms are relatively expensive to compute compared to polynomials. Carrying out this simplification results in less than 1% error in the sectional inertial and aerodynamic loads, but reduces computation time by 25% on each thread.

   There is a subtle but important distinction here: small elastic deflections do not imply the assumption of small angles throughout the formulation. For example, regardless of whether the small bending angle assumptions are invoked or not, inflow angles are computed exactly, and the reverse flow model is unaffected. Similarly, the geometric twist and elastic twist are accounted for exactly. The only potential source of error is when blade bending angles are large.

   The rotation matrix from the undeformed preconed rotating blade coordinate system to the deformed blade cross-section is approximated as

$$\mathbf{T}_{\text{DU}} = \begin{bmatrix} T_{11} & T_{12} & T_{13} \\ T_{21} & T_{22} & T_{23} \\ T_{31} & T_{32} & T_{33} \end{bmatrix} \tag{29}$$

   Where

$$\begin{aligned} T_{11} &= 1 - \frac{w'^2 + v'^2}{2} \\ T_{12} &= v' \\ T_{13} &= w' \\ T_{21} &= -v' \cos\theta_1 - w' \sin\theta_1 \\ T_{22} &= (1 - \frac{v'^2}{2}) \cos\theta_1 - v'w' \sin\theta_1 \\ T_{33} &= (1 - \frac{w'^2}{2}) \sin\theta_1 \\ T_{31} &= v' \sin\theta_1 - w' \cos\theta_1 \\ T_{32} &= -(1 - \frac{v'^2}{2}) \sin\theta_1 - v'w' \cos\theta_1 \\ T_{33} &= (1 - \frac{w'^2}{2}) \cos\theta_1 \end{aligned}$$

   θ_1 is the total rotation angle about the beam deformed x-axis, and is equal to the sum of control inputs, geometric twist, elastic twist and the kinematic integral twist. More details of the formulation may be found in Ref. 13.

Free wake model

In this study, an in-house free wake model (Ref. 14) was used as an initial baseline, and acceleration strategies to achieve real-time execution are detailed below. The aerodynamically significant elements of the free wake model are shown in Figs. 1 and 2: (a) the blade bound vortex (that moves with the blade structure), along the quarter-chord line (b) the trailed

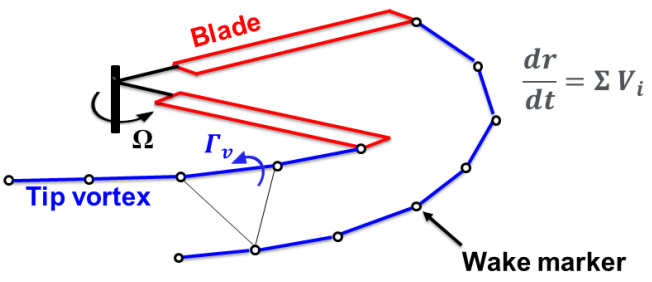

Fig. 1. Free-Vortex Wake Model

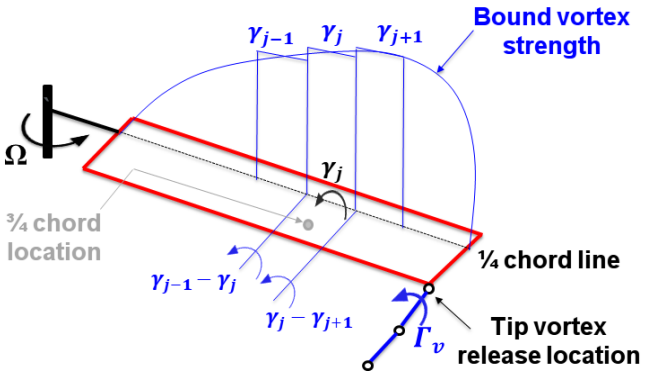

Fig. 2. Blade Near Wake

wake sheet that extends 30 degrees behind the blade (referred to as the "near-wake") and (c) the rolled up tip vortex (trailed), referred to as the "far-wake". The curved tip vortex is approximated as a series of straight-line segments connecting a set of wake markers.

The bound vortex line is discretized into numerous segments from the root cut-out to the blade tip. The near-wake sheet is assumed to be rigid, and its effect on the blade is modeled using an influence coefficient matrix. The far wake is free to move in space (the "free" component of the wake) according to the convection equation

$$\frac{d\mathbf{r}}{dt} = \mathbf{V}_i \tag{30}$$

In Equation 30, $\mathbf{r}$ refers to the location of a collocation point on a wake trailer and $\mathbf{V}_i$ is the total induced velocity on the collocation point, due to the blade bound vortex, the near-wake and all vortex line segments in the far wake. The bound vortex strength is computed by enforcing flow tangency at the three-quarter chord locations along the rotor blade, and total rotor inflow velocities at these locations are used to compute the effective angles of attack at the blade sections. The peak of blade bound circulation (computed from the sectional lift distribution using the Kutta-Joukowski theorem) is used to set the tip vortex strength. The velocity induced by a line vortex at a point is computed using the Biot-Savart law, assuming a viscous vortex core. The initial core radius is set to 5% of the rotor blade chord, and a core growth model is used to model dissipation of the vortex over time.

The free wake algorithm proceeds according to the following steps:

1. Initialize tip vortex geometry from a Landgrebe prescribed wake model
2. Compute tip vortex strength from blade lift distribution
3. Set bound vortex and near wake locations in space using blade motion information
4. Find vortex-induced velocities at the rotor blade and update bound vortex strengths using the flow tangency criterion
5. Find induced velocities on the tip vortices at time step "p-1"
6. Advance one time step using the Euler explicit scheme (predictor)
7. Find induced velocities on the tip vortices at time step "p"
8. Advance solution from time step "p-1" to time step "p" using 4-point velocity average
9. Compute inflow at the rotor blades
10. Repeat steps 2 – 9 for additional time steps

In the current implementation, a new wake marker is generated on the tip of each blade at every time step as shown in Fig. 2, and the last wake marker (at the end of the vortex trailer) is discarded, assuming that it is no longer aerodynamically active. Therefore, for the free wake model, the wake resolution and time step are linked together. Though the distinction between wake age and azimuth step can be easily separated, these quantities are treated as one entity in the present work.

When implemented on a Intel Xeon 2687w-v3 10-core CPU, the time required for a serial implementation is 25.1 seconds for 6 rotor revolutions. With `OPENMP` parallelization and 10 threads, the wall clock time for computations reduces to 3.35 seconds. For a UH-60 rotor, the rotation speed is 27 rad/s, so the rotor actually completes 6 revolutions in 1.396 seconds. Therefore, the parallelized CPU implementation needs to be accelerated by 2.4× to achieve real-time. It may be possible to use additional processors or more recent hardware to achieve this speed-up. However, the CPU is already assigned for simulating the structural dynamics/flight mechanics, and the target for the present work is to achieve a real-time simulation using a single desktop computer. Therefore, using CPUs to perform both the rotor aeromechanics and the aerodynamics in real-time is not currently feasible for the hardware considered.

The computationally expensive part of the simulation is the calculation of the induced velocities on the tip vortex markers and the three-quarter chord locations along the rotor blades,

i.e. the total vortex-induced velocities from all line vortices on the various collocation points in the flowfield. To obtain a grid-converged solution for a 4-bladed rotor, 6 wake turns are required with a ten degree azimuthal discretization (868 wake markers). The number of bound vortex segments required for each blade is 40. In the free wake model, the number of vortex-vortex interactions is approximately 1 million at a time instant, all of which are independent of each other. With a predictor-corrector type implementation, the number of vortex-induced velocity computations per time step is in excess of 2 million. Each of these vortex-induced velocity computations can be performed in parallel, and GPU computing presents a viable option to accelerate these computations.

GPU Implementation

For this paper, the free wake model was implemented in `CUDA Fortran` to take advantage of the massive parallelism offered by Graphics Processing Units. The parallelization algorithms applied to each of the steps 2 – 9 in the time marching process are described in this section.

1. **Step 2: The tip vortex strength** is provided by comprehensive analysis using airfoil tables, and is not recalculated by the free wake model. Therefore, there is no scope for parallelization at this stage.

2. **Step 3: The blade bound vortex locations** in space must be computed at $n_s + 1$ locations, where n_s is the number of spanwise segments used to discretize the blade. Using $n_b \times n_s$ simultaneous parallel threads (where n_b is the number of blades), i.e. one thread for each end of every bound vortex segment for every blade, these vortex locations are updated simultaneously. For 40 segments and 4 blades, 160 coordinates can be updated simultaneously. With the current hardware (NVidia GTX 1080), 640 simultaneous threads may be executed simultaneously, so there is a lot of scope for introducing more rotors or blades.

3. **Step 4a: The calculation of induced velocities at rotor blades** involves the calculation of $n_b^2 \times n_z \times n_s$ independent Biot-Savart computations, where n_z is the number of line segments used to discretize a blade tip vortex. Each of these calculations are performed in parallel and stored in a temporary array. The total induced velocity at the $n_b \times n_s$ blade control points are accumulated using a binary summation, which is also parallelized across forcing sources as well as target blades and control points.

4. **Step 4b: Calculation of blade bound vortex strengths** requires the component of induced velocity from the "far-wake" perpendicular to the local blade segment at the three-quarter chord location. Using the total induced velocities accumulated at the end of Step 4a, a cosine projection operation is carried out to find the normal velocity component $\mathbf{V}_i \cdot \hat{n}$ at each of the $n_s \times n_b$ blade control points. This operation is also parallelized over each rotor blade and each control point, i.e. using $n_s \times n_b$ parallel threads.

5. **Step 4c: Computation of bound vortex strengths** involves solving the equation

$$\mathbf{A}\Gamma \quad = \quad \mathbf{V}_i \cdot \hat{n} \tag{31}$$

Where $\mathbf{A}$ is the influence coefficient matrix (representing the proportional induced velocity effect of near-wake and bound vortices on each of the control points). If the normal velocity data is transferred to the CPU and the solution for the bound vortex strengths Γ transferred back, the time overheads are excessive. Instead, the linear equations are themselves solved using standard Gaussian Elimination. No pivoting is required for this case, because the diagonal entries in $\mathbf{A}$ are much larger than the off-diagonal components. Further, direct Gaussian Elimination is preferred to L-U decomposition for this problem, because the matrix $\mathbf{A}$ is sensitive to compressibility effects and changes at each time step; preserving the influence coefficient matrix presents no computational advantage.

The Gaussian Elimination operations are also performed on the GPU in parallel, using the following algorithm:

   (a) Loop over rows: k = 1 to n_s

   (b) Loop over the operated row i from 1 to n_s

   (c) if the operated row $i \neq k$, then $R_i \rightarrow R_i - \frac{A_{ik}}{A_{kk}} R_k$ and $b_i \rightarrow b_i - \frac{A_{ik}}{A_{kk}} b_k$. Each of the row operations during one elimination step (i.e. for each k) are independent of each other, and are parallelized in CUDA-Fortran.

   (d) For each elimination, only operate on columns $j \leq k$; each of these operations is parallelized over one thread dimension in CUDA-Fortran

   (e) Store the diagonal entry A_{kk} after each elimination

   (f) When $k = n_s$, then the solution is $\Gamma_i = \frac{b_i}{A_{ii}}$

The matrix A and right hand side vector $b = \mathbf{V}_i \cdot \hat{n}$ are loaded into shared memory, with all data for one blade placed on one CUDA block. All threads within the block operate on the influence coefficient matrix for a particular blade, and so the loop over k is executed without exiting the kernel. Using shared memory speeds up memory updates by a factor of 2 compared to standard GPU code. For a 4-bladed rotor with 40 bound vortex segments, $4 \times n_s \times n_s$ = 640 parallel threads are generated and executed simultaneously, the exact limit of the hardware used.

6. **Step 5: The induced velocities on the free wake markers** are the result of $n_b^2 n_z^2$ Biot-Savart computations, all of which are performed in parallel. The velocity on each marker is accumulated using binary trees parallelized over each target point on a wake marker as well as over each of the blades. The accumulation algorithm used in

this step is identical to that presented in **4a**. For a 4-bladed rotor with 6 wake turns and 10° discretization, the number of simultaneous calculations that can be performed is approximately 0.75 Million. These calculations are executed in parallel in batches of 640 threads by the hardware, and this step contributes the largest computation bottleneck ($\geq 50\%$) even after parallelization.

7. **Step 6: Advancing the wake geometry** using the Euler explicit scheme involves convecting the tip vortex collocation points based on the free-stream velocity and total vortex-induced velocity, and the rotor rotational speed. The update of wake marker positions is parallelized over rotor blades as well as collocation points along each tip vortex segments, i.e. the number of simultaneous calculations is $n_b \times n_z$. For the sample problem described in Step 5, the number of simultaneous updates is 864.

8. **Step 7** is parallelized in the manner described in Step 5, while **Step 8** is parallelized as described in **Step 6**. **Step 9** is parallelized as described in Step **4a**.

PYTHON INTEGRATION

Comprehensive Analysis Interface

The methods and variables in the comprehensive analysis are implemented and compiled in Fortran 95. The procedures for which Python interfaces are required are compiled separately through f90wrap (Ref 18) to generate, in sequence, intermediate layer C code, a shared object and a Python module. Importing the Python module grants access to the compiled routines and their dependencies. Another Python module interfaces with the pilot input device using native Linux drivers (Ref. 19) and communicates pilot controls to the flight dynamics engine. The total overhead per time step for communicating system states through the Python wrapper is 0.1 ms.

Free Wake Interface

The free wake analysis is integrated with the comprehensive analysis, but is compiled separately using a specific Fortran compiler that supports GPU-Fortran (Ref. 20). This separation between free wake and comprehensive analysis is necessary, because in the present work, the flight dynamics and rotor dynamics are handled by the CPU, wheras the free wake is simulated using the GPU. For the present applications, different Fortran compilers are required to optimize execution for the different hardware. For example, the Intel Fortran compiler (Ref. 21) produces heavily optimized code for the CPU, and so it is advantageous to use it when available. For GPU free wake, the PGI compiler is presently the only Fortran compiler with official support for CUDA routines.

Using CMake-generated scripts to create Python interfaces, the top-level routines that handle time marching in free wake are wrapped separately into another module. The CUDA runtime library and CUDA-Fortran module are also linked when creating the API, thus allowing Python access to this custom library of routines that handle GPU computing.

Data exchange between comprehensive analysis and free wake is handled through a single data structure (called derived type in Fortran) with other nested derived types. Each of the derived types at every level uses fixed-sized arrays that are known at compile time. The same memory, in the same data structure is passed between code compiled with different Fortran compilers - one for the CPU, and one for the GPU. On either end, each of the data structures that handle information transfer is inter-operable with C, and so they are inter-operable with each other. Therefore, no additional routines for memory handling are necessary, i.e. information conversion overhead is negligible across compilers.

Visualization Engine: Unity

The visualization engine used for this paper is Unity, a game visualization engine with a capability to execute C# scripts. Unity is initialized when the simulation starts, and launches a Python wrapper for the comprehensive analysis. Inter-process communication with shared memory is used to extract vehicle positions and attitudes from the Python-wrapped flight dynamics model. Unity's built-in Application Programming Interfaces (APIs) are subsequently used to update the position and orientation of the helicopter in space. The blades are visualized as rigid rotating blurred shapes consistent with the original CAD model. For a representative view from inside the cockpit of the helicopter, this reduction in realism (animating blade motions) did not affect the quality of the simulation, but eliminated a significant part of the information transfer from the physics engine to the screen. The visualization shell is created from CAD drawings and imported directly into Unity. A representative optionally manned quad-rotor (Ref. 22) is shown in Fig. 3.

The high workload placed on the GPU by Unity may interfere with the performance requirements needed from the same device to simulate the free wake. Therefore, the Unity visualization package was ported to another computer running Windows 7 with a separate GPU, and the flight dynamics was isolated on a server tower with a 10-core Xeon 2687Wv3 processor with an Nvidia GTX 1080 GPU. The User Datagram Protocol was used to communicate between the two computers using a LAN cable, with a few microseconds of delay (mean value). The scatter in these measurements was of the order of the mean time lag, and so this overhead is negligible.

Virtual Reality (VR) Interface

Unity (on Windows) also supports Virtual Reality headset plug-ins, and allows users to choose a perspective tied to a particular object (in this case, the VTOL platform). Figure 4 shows the overall Python framework, and depicts an operator using the HTC Vive VR goggles. Optionally, a chase camera (which does not tilt with the rotorcraft, but translates with it) may also be shown on screen to aid observers. Flight simulation trials with the HTC Vive (Ref. 23) indicated that a display

Fig. 3. JTARV in Unity Environment

update rate of at least 90 Hz was required to avoid operator disorientation.

Real-time Simulation: Criterion

The following time ratios are useful in determining whether a simulation of the helicopter executes in real-time or not:

1. The ratio of of time required for advancing the coupled rotor aeromechanics/flight dynamics by one time step, to the wall clock time corresponding to that time step is t_{fm}

$$t_{\text{fm}} = \frac{T_{\text{flight}}}{\Delta t} \tag{32}$$

   If the dynamic inflow model is being used to simulate the rotor aerodynamics, then real-time simulation is achieved when t_{fm} is less than unity, i.e. the simulation completes faster than the real-time clock. For most helicopter simulators, only the dynamic inflow model is included in a real-time framework. However, with hardware advancements, the execution speed of the free wake model can be dramatically improved. Therefore, the results section also illustrates the fidelity of the free wake model that can be executed in real-time.

2. The ratio of wall clock time needed to advance the free wake model forwards by one time step, to the actual duration of the time step is t_{fw}

$$t_{\text{fw}} = \frac{T_{\text{wake}}}{\Delta t} \tag{33}$$

If the free wake time ratio t_{fw} is greater than 1, then the simulation is slower than real-time. Additionally, the flight mechanics and free wake models must run sequentially at least once, so the criterion for the coupled free wake model and flight dynamics model executing in real-time is

$$t_{\text{fw}} + t_{\text{fm}} \leq 1 \tag{34}$$

Practical Considerations

The Virtual Reality apparatus requires an update rate of 90 Hz to obtain reasonable operator comfort levels. Thus, the total allowed computation time per time step is 11.1 ms, including all communication overhead.

For the actual rotor/body coupled dynamics, a time step corresponding to 5 degrees of rotor azimuth is adequate (Ref. 12) to enable accurate simulation of the high-frequency rotor modes. For a UH-60 type rotor, the corresponding clock time is 3.23 milliseconds, yielding an update frequency of 309 Hz. Comparing the minimum required update frequencies for the display (90 Hz) and the flight dynamics (309 Hz), it is apparent that for every 3 time steps (each of 5 deg rotor azimuth, or 3.23 ms), the display can be updated once. Thus, the physics engine must execute at 309 Hz, whereas the display can be updated at one-third of that frequency, i.e. 103 Hz.

The communication overhead is approximately 0.5 ms for the combination of pilot input processing, a simple state feedback control system, data handling/interpreting for the Python wrapper, and the delay between issuing output visualization commands and return of control to the Python framework. Assuming a worst-case scenario in which every time step incurs the entire 0.5 ms overhead, the available time for computation is 2.7 ms for a single main rotor system.

In case the free wake model cannot execute with a time step corresponding to a 5 degree azimuth change, the flight dynamics and rotor dynamics can be advanced for two time steps (each of 5 degrees) and the free wake model can be run for one time step of 10 degrees. Alternately, three time steps can be taken in the structural dynamics (each of 5 degrees) and the free wake model can be advanced by a time step of 15 degrees. However, using vortex segments of constant vortex strength, significant error accrues if the azimuth step (and wake age) are increased beyond 10 degrees.

Timings: Comprehensive Analysis

The comprehensive analysis was executed for 1000 time steps, with each time step corresponding to 3.23 ms, i.e. 5 deg of rotor azimuth at 256 RPM. The rotor speed is assumed constant for these simulations. The number of blade modes (rotor generalized degrees of freedom) and blade finite elements were varied. Twelve parallel threads were used for these timing runs to generate the time marching Jacobian, and the calculation of ODE residuals for the sub-iteration update was performed with 4 parallel threads (one for calculating loads from each blade). With the 0.5 ms overhead and no free wake, the comprehensive analysis executes in real-time with most combinations of simulation parameters.

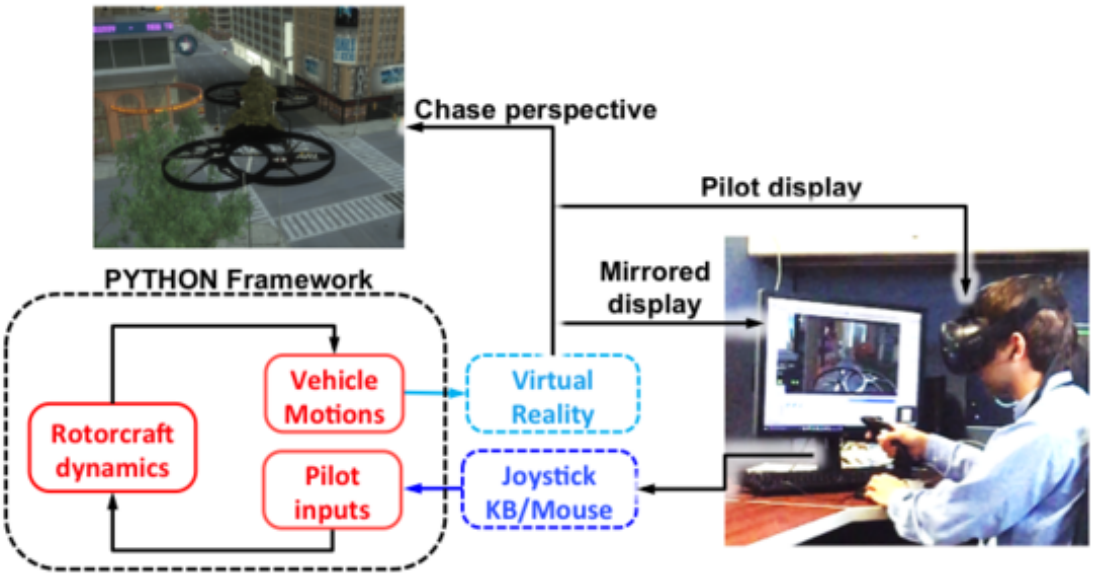

Fig. 4. Python Framework

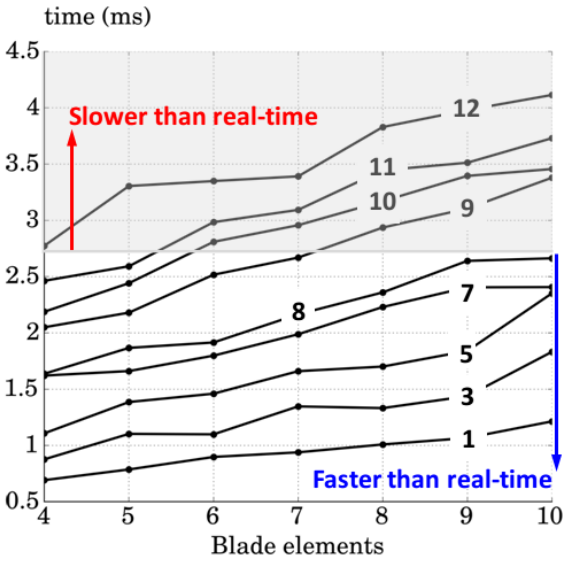

Fig. 5. Wall clock time required to execute one time step (Δt = 3.23 ms) of comprehensive analysis. Each line represents a different number of blade modes.

Figure 5 shows the execution time per time step (in ms) for various blade finite elements. Eight quadrature points along each of the finite elements are used to compute the sectional inertial, aerodynamic and structural loads, which are subsequently used to compute the blade loads, hub loads and system ODEs. With 5 finite elements (40 points along the blade, with an additional element for the swept tip), up to 11 blade modes can be simulated in real-time, by applying the small-angles formulation. The acceleration techniques outlined in the methodology enable real-time execution with the elastic blade modes without decreasing the fidelity of the simulation.

Integration of accelerated Jacobian into DASSL

DASSL (Ref. 17) is a generalized time marching solver for Differential Algebraic Equations (DAEs). It features an automatically calibrated variable-order and adaptive time step adjustment scheme, which guarantees that both relative and absolute error at each time step are reduced below a user-specified threshold. For research-fidelity analysis, this time marching scheme is preferred to a fixed time step and fixed order method, where errors may creep in due to linearization or choice of order.

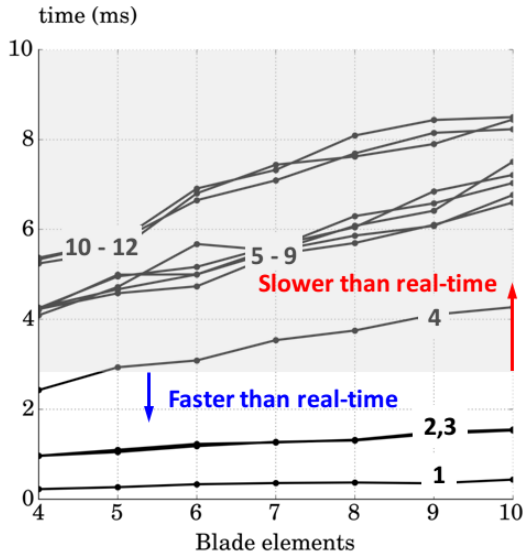

Fig. 6. Wall clock time required to execute one time step of the coupled rotor-body dynamics with DASSL. Each line corresponds to a different number of blade modes

DASSL provides a function handle for a user-specified routine to compute the time marching Jacobian **J**. Such a cus-

tom routine is created using the procedures and steps outlined in the previous subsections and provided to DASSL. The finite difference method in Eqn 28 was modified to choose the perturbation ε for each column in a manner identical to DASSL's internal logic. To preserve the same numerical values of the perturbations, additional arguments had to be passed to the user-specified Jacobian calculation routine, necessitating minor functional changes to the baseline logic within. After implementing the load-balanced parallelization for calculating the Jacobian within DASSL, a speed-up of 10× was achieved. Using DASSL to perform time marching of the coupled blade-body-inflow dynamics, the time required to perform time marching for 20 rotor revolutions was measured for various blade finite elements and various rotating modes, and plotted in Fig. 6.

After implementing the load-balanced parallelization for calculating the Jacobian within DASSL, a speed-up of 10× was achieved. When DASSL is used to perform time marching of the coupled blade-body-inflow dynamics, the model executes at 50% of real-time speeds with 18 parallel threads on a 10-core processor for a 4-bladed rotor with 10 modes per blade.

For 5 blade elements (40 spanwise quadrature points) up to 4 blade modes can be simulated in real-time (including the 0.5 ms overhead) with DASSL using the variable-step and variable order time marching scheme. With a fixed-time step scheme, additional modes can be simulated compared to using a variable time step and variable order method. Nevertheless, the timing improvements show that algorithmic acceleration techniques developed for real-time analysis can be incorporated into the research-fidelity rotor analysis to accelerate the computations.

Results: Free Wake Timings

The GPU fre wake code was executed on the NVidia GTX 1080, and the wall clock time for advancing the UH-60 rotor dynamics was measured for various wake age discretizations, wake turns and blade segments. The free wake execution timings were assessed for real-time execution independent of the comprehensive analysis. Only the free wake model timings are considered for identifying potential parameter combinations that can be combined with comprehensive analysis for real-time execution.

The 0.5ms overhead is included in the analysis, and subtracted from the allocated computation time to determine the real-time threshold. Figure 7 shows the variation of computation time per time step (10 deg azimuth, or 6.46 ms) for various blade bound vortex segments and wake turns. Each line represents a different number of wake turns. For all combinations of parameters chosen, the isolated free wake simulation runs in real-time with a 10-degree time step.

Analysis of the results showed that a grid-converged solution is obtained with 40 bound vortex segments along the blade. For low speed ($\mu \leq 0.1$) 6 wake turns are required to obtain accurate rotor power and hub loads. At intermediate

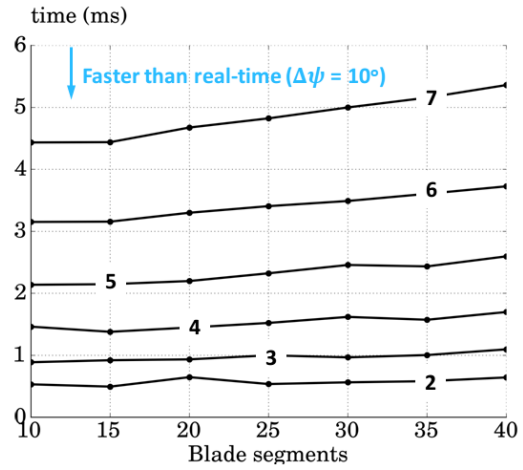

Fig. 7. Wall clock time required to execute one time step (Δt = 6.46 ms) of the free wake for various segments. Each line represents a different number of wake turns.

speeds ($0.1 \leq \mu \leq 0.25$) 4 wake turns are required for a grid-converged solution. At high speeds ($\mu \geq 0.25$) 2 wake turns are sufficient for accurate inflow predictions. To obtain accurate inflow predictions at both low speed and high speed flight conditions, 6 wake turns must be preserved in the simulation at all times.

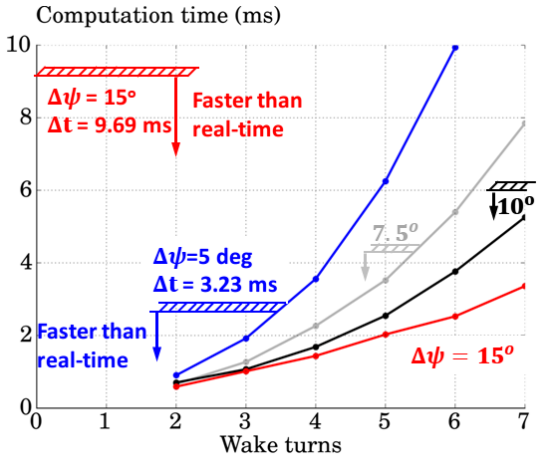

Fig. 8. Wall clock time required to execute one time step of the free-vortex wake, for varying azimuthal discretization and number of wake turns

Figure 8 shows the variation of computation time per time step for different wake age discretizations (time step sizes) for various wake turns, with 40 bound vortex segments per blade. Each of the lines correspond to a different time step and a different number of wake markers in the simulation. For example, with a 15 degree discretization and 4 wake turns, the number of wake markers per blade tip vortex trailer is 97. If the wake discretization is 5 degrees, then the number of particles in the simulation increases by a factor of 3, and the computational cost increases to 9 times that of the first case considered. With a wake discretization of 15 degrees (red line), the corresponding time step is Δt = 9.69 ms. According to the results,

even 7 wake turns can execute in real-time. By decreasing the wake age discretization to 10 degrees, 7.5 degrees and 5 degrees, the corresponding time steps are 6.46 ms, 4.85 ms and 3.23 ms. At 10 degrees discretization, 7 turns can still execute in real-time. At 7.5 deg (and 5 deg) discretization, only 5 turns (and 3 turns) can be simulated in real-time.

At high speeds, the number of wake turns can be decreased as the tip vortices are swept away by the free-stream flow. However, the discretization must be reduced to capture the change in vortex strength around the azimuth. It may be advantageous to use an adaptive scheme that changes both the wake discretization and number of wake trailers so that the total number of particles in the simulation is preserved, but preserves the dominant features of the flowfield.

Analysis: Wake and CSD Timings

To combine the comprehensive analysis model with the free wake, it is necessary to pick appropriate combinations of blade elements, blade modes, free wake time step and wake turns. One possible combination is to use a 10 degree time step for the free wake with 6 wake turns, which requires 3.8 ms. To perform time marching in real-time for 10 degrees, the execution must be completed in approximately 6 ms. The time remaining for the comprehensive analysis is (6 - 3.7) = 2.3 ms for two time steps of 5 degrees each. The current execution speed of the simulation suggests that only one blade mode can be used to obtain real-time execution for the coupled simulation.

If the free wake resolution is reduced to 15 degrees, the execution time for the free wake is 2.3 ms, with a wall clock time of (9.69 - 2.3) = 7.46 ms left over for executing 3 time steps with the comprehensive analysis. The allowed clock time for executing the comprehensive analysis is 2.5 ms per time step. Examining the timing results in Fig. 5, it is apparent that up to 10 modes with 5 blade finite elements may accommodate real-time execution. Further studies into sub-iteration convergence between free wake and comprehensive analysis are necessary to determine the resolution of the models (both free wake and rotor dynamics) that can run in real-time.

SUMMARY AND CONCLUSIONS

A fully immersive piloted simulation was achieved and demonstrated by integrating a rotorcraft comprehensive analysis (with an elastic blade model) into a Python framework and coupling it to a Virtual-Reality compatible visualization package (Unity). A breakdown of the computation time for various parts of the real-time physics model was used to identify and alleviate computation bottlenecks. The primary advantage of this approach is the elimination of ad-hoc tuning parameters and simplifying assumptions that are usually invoked in reducing a comprehensive physics-based model to one that can execute in real-time. Further, the original rotor-body coupled dynamics are time marched forwards in their undiluted forms, without any simplifying assumptions. The work in this paper has also presented a method to adapt a helicopter vibration, loads, performance and flight dynamics analysis tool for real-time pilot-in-the-loop simulation with minimal duplication of inputs.

1. For the implicit dual-time stepping algorithm investigated (a variant of DASSL), the most significant bottleneck is the calculation of the time marching update Jacobian $\mathbf{J}$. 90% of the computation time is spent in calculation of $\mathbf{J}$, while 10% of the time is spent in the sub-iteration update.

2. Parallel computing using a shared-memory standard (`OpenMP`), when applied simultaneously with load balancing, to provide 11.8$\times$ speed-up over serial execution when executed on 10 processing cores with 18 threads and hyper-threading.

3. The parallelization techniques developed for real-time simulation was also used to accelerate the open-source implicit time marching solver DASSL, by accelerating the computation of the time marching Jacobian $\mathbf{J}$. The effective speed-up for DASSL using parallelized generation of $\mathbf{J}$ is 10$\times$.

4. The free wake model was recast to execute on GPUs using CUDA-Fortran. By transferring the computationally intensive part of the N-body simulation to the GPU, the execution was accelerated by a factor of 31$\times$ over CPU serial (4$\times$ over CPU parallel), including all information transfer delays. With this speed-up, the stand-alone free wake model can execute in real-time with 40 bound vortex segments, 6 wake turns and a 10 degree azimuthal discretization.

5. Though each of the models individually can achieve real-time execution, the trade-off between accuracy and speed in sub-iteration convergence must be quantified before calling the entire coupled simulation as achieving "real-time". These areas, together with the exploration of additional acceleration strategies for the free wake model, are subjects of future work.

ACKNOWLEDGMENTS

This work was supported by the Cooperative Research Agreement between the University of Maryland (UMD) and the U.S. Army Research Laboratory at Aberdeen, MD. The authors would like to thank Rajneesh Singh and Hao Kang from ARL for their insights, suggestions and continued support.

REFERENCES

[1]Padfield, G.D., Pavel, M., Casoralo, D., Roth, G., Hamers, M. and Taghizad, A., "Fidelity of Helicopter Real-time Simulation Models," 61th Annual Forum of the American Helicopter Society International, Grapevine, TX, June, 2005.

[2]Datta, A. and Johnson, W., "Requirements for Next Generation Comprehensive Analysis of Rotorcraft", American Helicopter Society Specialists Conference, San Francisco, CA, Jan 23-25, 2008.

[3]Duhon, J.M., Harvey, K.W. and Blankenship, B.L., "Computer Flight Testing of Rotorcraft," *Journal of the American Helicopter Society*, Vol. 10, No.4, 1965, pp 36–48.

[4]Dreir, M., "Development of a Real-time Blade Element Aeroelastic Rotor", AIAA Flight Simulation Technologies Conference, Monterey, CA, 1987.

[5]Corrigan, J.J., Meyer, A., Bothwell, M. and Brown, H., "Computer Flight Testing of Rotorcraft," American Helicopter Society Vertical Lift Aircraft Design Conference, San Francisco, CA, January 2006.

[6]Horn, J.F., Bridges, D. O., Wachspress, D.A. and Rani, S.L., "Implementation of a Free-Vortex Wake Model in Real-Time Simulation of Rotorcraft", *Journal of Aerospace Computing, Information, and Communication*, Vol. 3, No. 3 (2006), pp. 93-107.

[7]Zivan, L. and Tischler, M. B.,"Development of a Full Flight Envelope Helicopter Simulation using System Identification", *Journal of the American Helicopter Society*, Vol. 55, No. 2 (2010): 22003-22003.

[8]Gori, R., Pausilli, F., Pavel, M.D. and Gennaretti, M., "State-Space Rotor Aeroelastic Modeling for Real-Time Helicopter Flight Simulation," *Advanced Material Research*, Vol. 1016, pp. 451-459, 2014. (DOI: 10.4028/www.scientific.net/AMR.1016.451)

[9]Keller, J., Wachspress, D. and Hoffler, J., "Real Time Free Wake and Ship Airwake Model for Rotorcraft Flight Training Application," 71^{st} Annual Forum of the American Helicopter Society International, Virginia Beach, VA, May 5–7, 2015.

[10]Keller, J., Wachspress, D. and Hoffler, J., "Real Time Free Wake and Ship Airwake Model for Rotorcraft Flight Training Application," 71^{st} Annual Forum of the American Helicopter Society International, Virginia Beach, VA, May 5 – 7, 2015.

[11]Oruc, I., Shenoy, R., Shipman, J. and Horn, J.F., "Towards Real-Time Fully Coupled Flight Dynamics and CFD Simulations of the Helicopter/Ship Dynamic Interface," 72^{nd} Annual Forum of the American Helicopter Society International, West Palm Beach, FL, May 17–19, 2016.

[12]Sridharan, A., Rubenstein, G., Moy, D.M. and Chopra, I.,"A Python-based Framework for Real-time Simulation using Comprehensive Analysis", 72^{nd} Annual Forum of the American Helicopter Society International, West Palm Beach, FL, May 17–19, 2016.

[13]Sridharan, A., "Simulation Modeling of Flight Dynamics, Control and Trajectory Optimization of Rotorcraft Towing Submerged Loads," Ph.D. Dissertation, Department of Aerospace Engineering, University of Maryland at College Park, 2014.

[14]Govindarajan, B., and Leishman, J. G., "Predictions of Rotor and Rotor/Airframe Configurational Effects on Brownout Dust Clouds," 70^{th} Annual Forum of the American Helicopter Society International, Montreal, Quebec, May 20–22, 2014.

[15]Peters,D.A. and He, C.J., "Comparison of Measured Induced Velocities with Results from a Closed-form Finite State Wake Model in Forward Flight", American Helicopter Society 45th Annual National Forum, May 1989.

[16]Passe, B., Sridharan, A. and Baeder, J.D., "Computational Investigation of Coaxial Rotor Interactional Aerodynamics in Steady Forward Flight", 33^{rd} AIAA Applied Aerodynamics Conference, Aviation 2015, Dallas, TX, June 22–26 2015.

[17]Brenan, K. E., Campbell, S. L. and Petzold, L.R., " The Numerical Solution of Iniitial Value Problems in Differential-Algebraic Equations" Elsevier Science Publishing Co., 1989.

[18]f90wrap: Python wrapping for fortran's derived types, modules and subroutines. Open-source project at `https://github.com/jameskermode/f90wrap`

[19]Python-based interpreter for Joystick Inputs, adapted from `https://gist.github.com/rdb/8864666`

[20]Fortran compiler that supports GPU/CUDA syntax, partners with NVidia. More details at `http://www.pgroup.com/resources/technologies.htm`

[21]Intel Fortran compiler. More details at `https://software.intel.com/en-us/fortran-compilers`

[22]Avera, M., Kang, H. and Singh, R., "Performance and Controlability Assessment of an Overlapping Quad-Rotor Concept", 72^{nd} Annual Forum of the American Helicopter Society International, West Palm Beach, FL, May 17– 19, 2016.

[23]High Tech Computer Corporation's Virtual Reality Goggles `https://www.vive.com/us/product/`