A Framework for Integrated Flight Simulation and Design

Jeff Parrish
University of Georgia

parrishj@uga.edu

Micah Cooper
University of Georgia

mrcooper@uga.edu

ABSTRACT
In this paper we present a framework for rapid prototyping of airplane designs as an example of the benefits of integrating simulation, gaming, and design interfaces into a cohesive package, which we have labeled the Integrated Flight Simulation and Design framework. We discuss the design of this framework and detail the progress of a simple prototype implementation. We discuss the advantages close integration of the simulation and design components of aerospace vehicle design has for required time and cost of the prototyping process.

Categories and Subject Descriptors
I.6.5 [**Simulation and Modeling**]: Model Development – *modeling methodologies.*

General Terms
Design, Economics, Reliability, Experimentation, Human Factors, Standardization, Verification.

Keywords
Newt, IFSID, flight simulation, aerospace design, aerospace editor, simulation integration, design framework, simulation framework, multiparadigm modeling

1. INTRODUCTION
The integration of aerospace vehicle design and simulation is an important step in the streamlining of the overall vehicle development process. Until recently, design tools and design models were largely separate from simulation tools and models, requiring a lengthy and expensive translation process to perform simulation-based tests. In addition, after the simulation results have been analyzed, they must undergo another translation process to determine the appropriate design changes. Integration between editor and simulator has benefits in addition to the minimization of these translation costs. The frameworks developed for this integration are excellent examples of applications of multiparadigm modeling methodologies. Several other researchers have developed integration frameworks for these two components, in varying degrees of detail, scope, and goals. Our framework differs from these in that it is not intended for use as an engineering-quality package, but rather to serve as an illustration of the benefits of even moderate integration between editor and simulator, and to provide a backdrop for education about flight simulation, design, and physics.

We have developed an integration framework which allows for modular representation of a plane both in the design editor and the simulator. The framework uses this plane model to communicate between the editor's model representation and the simulator's physics calculations. It also provides for utility functionality including simulation analysis, design optimization, and alternate information presentations. The framework is designed to be modular, extensible, and scalable. The primary design goal of the framework is to allow rapid transitions between the design editor and the simulator. Secondary goals included an acceptable level of accuracy in the simulation physics model, interactivity in the simulator to motivate learning, and an intuitive, powerful, and flexible editor. Our primary motivation was to experiment with simulation and aerospace vehicles as a learning method.

The framework for editor and simulator integration draws heavily from elements of multiparadigm modeling. A key challenge in designing such a framework is combining multiple modeling methodologies and their interactions in a uniform way. Integration between design editor and simulator has been attempted in other research. [Liu and Berndt 2006] apply this methodology to develop their own integration framework, which they use to examine a flight control system for a short-takeoff-vertical-landing (STOVL) plane. They designed their framework to emphasize modularity and computational distribution, as well as minimizing or removing the design model to simulator model translation. Similarly, the RIPTIDE system is part of a design pipeline intended to streamline the editor-simulator relationship [Mansur et al. 2000].

This paper will present the design for this framework as intended to aid rapid prototyping of aerospace vehicle designs. This framework illustrates the benefits of close integration between simulation, gaming, design, and optimization. Section 2 discusses the background and motivation for this framework. The design of the framework is detailed in Section 3. The current implementation is described in Section 4. Finally, Section 5 summarizes our statements.

2. BACKGROUND
In 2006 the aerospace industry averaged over $52.5 billion, as show in [AIA 2006], and the industry continues to grow. The vast bulk of this money is spent on operating costs. Decreasing these costs, in both time and money, is a primary pursuit of these companies. A cheaper, more rapid development of aerospace vehicle prototypes is precisely what is desired.

Modern aerospace design is an iterative process. Potential designs are subjected to exhaustive testing and analysis, the results of which feed back into the design process. Aerospace vehicles are also unified mechanisms - every aspect of their design has potentially critical relationships with other design aspects. Even a minor change to the design may require costly and time-consuming re-evaluation and re-testing of the entire system. When dealing with physical prototypes, this translates into enormous costs, which is one of the motivations for the rigorous nature of aerospace design methodologies. The advent of computer-based simulation allows for designs to be tested at a fraction of the cost of traditional testing with physical models. There are a variety of tools designed specifically to simulate various aspects of a vehicle design. These range from component-level testing, such as simulation to determine airfoil characteristics, to full-scale performance testing, such as pilot-in-the-loop (PIL) flight simulators, used to test control systems or train pilots. Interactive flight simulators are an invaluable training tool for pilots, and their history begins almost as early as the invention of powered flight. Commercial single-computer flight simulators are certified to replace a significant portion of actual flight hours required for obtaining a pilot's license. Mechanical flight simulators were used to train pilots in World War I and II, as well as astronauts during the Gemini and Apollo programs. The military relies enormously on flight simulation in training its pilots. All of these examples illustrate the beneficial nature of PIL simulation on the pilot's understanding of the vehicle he or she is to operate. As in [Liu and Berndt 2006], pilots are also involved in the analysis of new control systems through simulation, providing valuable feedback to the design.

From the design point of view, simulation provides a plethora of analytical data that is difficult, expensive, and/or impossible to collect from physics testing. Temperature, stress and response profiles are only a few examples of the data simulation can provide. The SpaceShipOne suborbital vehicle was designed almost entirely by computational fluid dynamics (CFD) simulation of each successive design iteration. The design process itself has also seen a variety of tools emerge, such as the multiple computer-aided design (CAD) programs. Some of these are packaged with associated simulation capabilities, such as Matlab and Simulink, but typically a full-scale flight simulation requires additional time and effort to create a simulation-ready flight model. Integration between these design packages and flight simulation, especially PIL simulation, has a number of advantages. First, the time, effort, and money spent on translation between design and simulation models, as well as simulation results and and design changes, can be minimized with a close and efficient integration scheme. Second, PIL simulation provides the designer with the same valuable knowledge of the vehicle as a pilot receives, giving them extra insight into the characteristics of the design. In addition, there is nothing to prevent testing with actual pilots. Third, the close integration and low translation costs between design editor and simulator allow changes and their effects to be quickly and cost-efficiently analyzed, greatly streamlining the iterative design process.

Our approach to this integration is two-fold. First, we closely integrate the simulation and design portions of vehicle design using a common plane model. The naturally cyclic relationship between the two portions is best expressed when the logical distance between them is smallest. This has the benefit of both decreasing the time required to design and simulate a prototype, as well as the cost of transferring a design to a simulation format. The plane model we present allows an algorithmic transformation of the plane model to the appropriate simulation model, making designs modular. The close integration means that these simulations may be performed and evaluated, changes made to the design, and the same simulations continued or preformed again without any intermediate process. The quick turnaround time also promotes affordable exploration of multiple design directions.

Second, we combine the simulation with user input to generate a PIL gaming environment. As shown in [Jain and McLean 2005], this provides the user with a clear mental model of the relationships between their design choices and the specific effects each design choice has on the overall design. The relationship between learning and gaming has been the subject of several studies, and it has generally been found that targeted application of gaming aids overall learning [Kincaid et al. 2003, Randel et al. 1992, Gopher et al. 1994]. The large quantity of qualitative information that is obtained from interacting with the prototype design helps the user make better design choices regarding the prototype. Simulation and gaming naturally have a large measure of similarity, but the integration of the two provides both qualitative design guidance and quantitative analytical results. The quantitative results have a corrective influence on the qualitative model the user has of the prototype. This allows insight and understanding of specific design choices that may not be immediately clear when the simulation portion of the process is non-interactive.

3. DESIGN AND ARCHITECTURE

The ultimate goal of the Integrated Flight Simulation and Design (IFSID) system is to provide a framework for rapid design of aircraft by integrating simulation of the craft closely with the design process. A simulation of a possible design should reflect changes in the design instantaneously, so that the designer may evaluate the effects of each change made and experiment with alternative development paths quickly, easily, and cheaply. To achieve this goal, IFSID is divided into three main subsystems. The simulation subsystem evaluates the current model in a given environment, producing performance data and giving the designer an accurate portrayal of the operation of that design. The editor subsystem provides an intuitive but powerful interface that allows modification of the design on multiple levels of detail. A utilities subsystem provides a variety of functionality to aid in the design process by providing miscellaneous aids such as data analysis and representation, physics model examinations, and similar utilities. All of these operate on an object oriented representation of the plane model.

3.1 Plane Model

The plane model must be able to represent a variety of plane configurations and designs. As such, the model must be flexible in the configuration of the distinct plane components, and allow a range of specification of their attributes. The model must also capture the behavior of each component of the system, as well as the emergent behavior of their interactions. This means that the model must have a uniform way of representing how each component should be simulated within the simulator subsystem. This representation should not consist of actual physics logic,

however, since the physics system should be independent of the plane model representation.

The plane is considered a single object which is composed of multiple linked components, as shown in Figure 1. The plane may have some global attributes or functionality, such as altitude, orientation, or implementation specific parameters, but the main function of this general object is to serve as a composition of the components and their interrelations. Each component represents a part or subpart of the plane, and its relation to one or more other parts. For example, the left horizontal stabilizer of a small civil plane would be represented by a single component. A large jetliner, however, may have a separate component for each portion of a multi-segmented wing. These wing components would have specified relationships with each other, indicating their spatial relation, physics joint strength, position, and so on. These are typically parent-child relationships, although a tree structure is by no means required - for example, a biplane model may have a circular relationship between the fuselage, a wing, a wing spar, the other wing, and the fuselage. The only stipulation is that each component must have a relation to one or more of the previously existing components. One special component is designated the root component, such as a fuselage, which has no parent relationship.

Each component contains its own attributes, including a mesh for its geometry, material information, mass, center of mass, mass distribution, and so on. Depending on the type of component, some of the attributes may be more important than others for the physics simulation of the component. The type of component determines how it is treated in the physics model. For example, an airfoil component may be simulated using a vortex panel method to determine coefficients of lift, moment, and induced drag, while a fuselage may be simulated primarily based on body drag equations. The components are unaware of how their simulation types are handed by the simulation subsystem, only that they are of a particular type of simulation object. These component types are the interface between component behavior representation and the simulation subsystem.

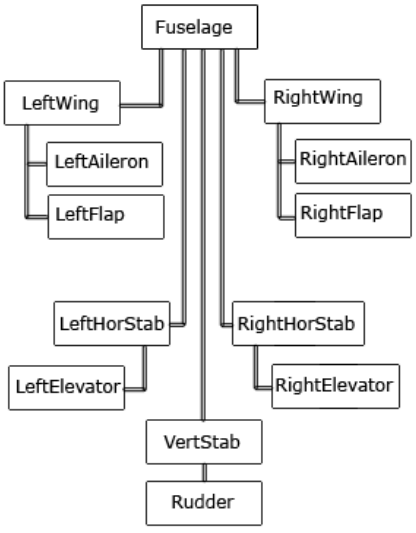

Figure 1: An example of component relationships in a plane model.

3.2 Simulator Subsystem

The simulator subsystem contains the physics model, the physics engine, the environment, the visualization system, the control interface, and data generation. The physics model is responsible for executing the appropriate physics logic for each component based on its component type. This logic may be as complicated or simplistic as the engine allows. The result of the model logic is a series of physical effects on the components, such as forces, torques, heating, deformation, et cetera. These are applied on a per component basis. The physics engine then takes the disparate effects and integrates them based on the component relationships in order to determine the net effect of the results on the plane. These effects are influenced by, and may influence, the environment in which the plane is operating. Altitude of the plane will greatly affect the atmospheric density the plane encounters, and thus the lift it can generate through differential pressure. Similarly, a plane crossing the supersonic threshold may generate shockwaves in the environment which affect control surfaces downstream. The visualization subsystem takes the geometry and any special effects and renders them for the user to interpret. The user may then modify control surfaces or environmental settings through the control interface. Finally, the data generated by the physics model and engine can be recorded to be analyzed by the utilities package or to verify performance characteristics.

The most important of these functions is the physics model. The detail and accuracy of the physics model determines in large part the reliability of the results of the simulation of the plane design. The interface of component types allows a plane model to be specified independently of a physics model implementation, and it also allows each component type to be simulated by a variably detailed model. This modularity allows the user to specify multiple levels of detail for varying components, and also allows for simple comparison of alternative methods. It also provides for easy extensibility of the system as more accurate or efficient methods are implemented. For example, an airfoil may be simulated in a variety of ways. It may be considered to have a static performance table which can be

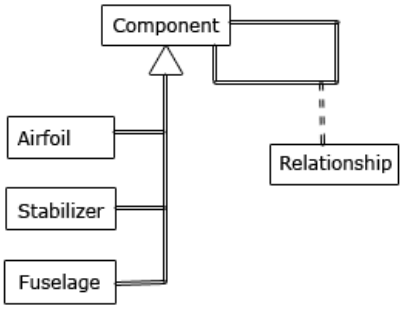

Figure 2: An example of component types.

referenced quickly, it may have performance determined dynamically by applying a panel method, or it may be subjected to an in-depth computational fluid dynamics simulation in order to determine precise characteristics. All of these data may be recorded separately, before the physics engine combines the results of the system update to determine the net effect.

The second most important feature is that of user involvement. The simulation allows the user to actively manipulate attributes and control surfaces of the plane while

receiving realistic feedback through an instrumentation heads-up display, giving a much greater feel for the performance of a design than numerical analysis alone can provide. This qualitative assessment can be an enormous asset when paired with the quantitative analysis the data generated provides. The simulator should also support real-time viewing of the forces, torques, velocities, and other physics effects occurring on each component. The simulator's modular structure allows the plane model to be altered dynamically during the simulation, allowing the user to immediately effect changes without having to restart the scenario. This allows rapid exploration of modifications of the design.

3.3 Editor Subsystem

The advantage of this framework lies in its close integration between simulation of a plane model and editing that model. This means that the editor subsystem should provide a simple, intuitive user interface for modification of important plane components. The editor must also be able to provide the user with the ability to alter detailed information of each component. This necessitates multiple levels of detail in the editor. Finally, the editor should be able to generate as wide a variety of model designs as possible. The editor should only be restricted by the limits of the general plane model.

To create an intuitive user interface while still containing a sufficient amount of information required to be useful, the editor makes use of multiple levels of detail. In general, the editor makes use of two of these levels. The first is an overall perspective on the plane model. It deals primarily with the configuration of the plane components. The editor provides a variety of basic component objects, which the user may add to the plane model at will. One component, such as the fuselage, is specified as the root component, and all other components are attached to previously added components, as discussed in section 3.1. The user is able to determine the spatial and physics relations between separate components, and to alter general characteristics of each component such as mass, basis dimensions, material, or texture. An example realization might include a selection of basic components, a drag-and-drop interface for adding components, and a similar interface for manipulating their positions, dimensions, or other general attributes. The user should have the option of mirroring the model across a given plane, in order to ensure symmetry. To simplify the editing process, the model may be presented in the standard top, front, and side orthogonal views, separately or simultaneously, to give the user a simple interface while maintaining maximum flexibility.

More detailed editing can then be performed in a component-specific interface. This would provide access to the attributes of the particular component type, in a specialized component view. For example, editing an airfoil component type would yield an airfoil cross-section view allowing the user to edit the geometry, thickness at root and tip, and other attributes. A planform[1] view may also be presented to provide the user with the ability to taper or skew the airfoil. This provides the user with the ability to alter the actual geometry of the plane model without having to deal with the complexity of a full-scale three-dimensional editor.

[1] A planform view is an overhead view of an object, such as would be shown in a blueprint.

After the user has created a model, he or she has essentially created a list of basic component forms and attributes for each. These are the inputs for a construction algorithm, which performs the translation from the editor's model representation to the general plane model. This includes the creation of each component object, including its attributes and specified mesh geometry, as well as defining the exact spatial and physics relationships existing between components. This translation allows the editor to maintain its flexibility and ease of use while enabling the simulation to operate on an exactly specified design.

If the user wishes, the modular structure of the plane model makes it possible to design plane geometry in a three-dimensional editor and specify the attributes of each component manually. While this approach is much more difficult and time consuming than use of the editor, it should allow for previously built models to be imported into this framework.

3.4 Utilities Subsystem

The simulation subsystem generates a plethora of data that must be analyzed to quantitatively evaluate the design. These data must also be presented in multiple ways to aid understanding of what the data represent. There is also the need for a variety of miscellaneous functionality that greatly enhances the ability of the user to evaluate potential designs. Finally, the logical continuation of integration of the simulation and design portions of a design evaluation is that of computational optimization of the design based on simulation results.

The utility simulation should provide functionality to analyze and summarize important data from the simulation. This includes statistical analysis of the results, computation of performance metrics, et cetera. These data must then be represented to the user, either in graphical or textural form. The simulation should also provide the functionality to compare results across multiple simulation runs and display the data and results to the user. The utilities subsystem should contain miscellaneous functionality such as a model viewer, which allows the user to examine the current state of the simulation model and the forces on it in more detail. For example, the user may be able to scale the model to a larger dimension in order to examine small details.

The benefit of integrating the analysis and evaluation results of simulation with the rapid and powerful design modifications of the editing could be greatly extended with computational optimization. Application of genetic algorithms or other techniques have been detailed in a variety of other papers [Cantù-Pax 1998, Padula 1994, Padula and Gillian 2006]. The conversion of the plane model attributes to a suitable format for these techniques, or conversely their application to the plane model data set would both be trivial operations.

4. CURRENT IMPLEMENTATION

We have partially implemented a simple prototype of this framework. The implementation was creating using Java, the jMonkeyEngine (jME) game library, and the jMEPhysics 2 physics engine, which is developed with ODEJava, itself a Java wrapper for the Open Dynamics Engine (ODE) written in C. jME makes use of the Lightweight Java Gaming Library (LWJGL), which provides Java access to OpenGL. Our prototype is called the Newtonian Flight Simulator (Newt).

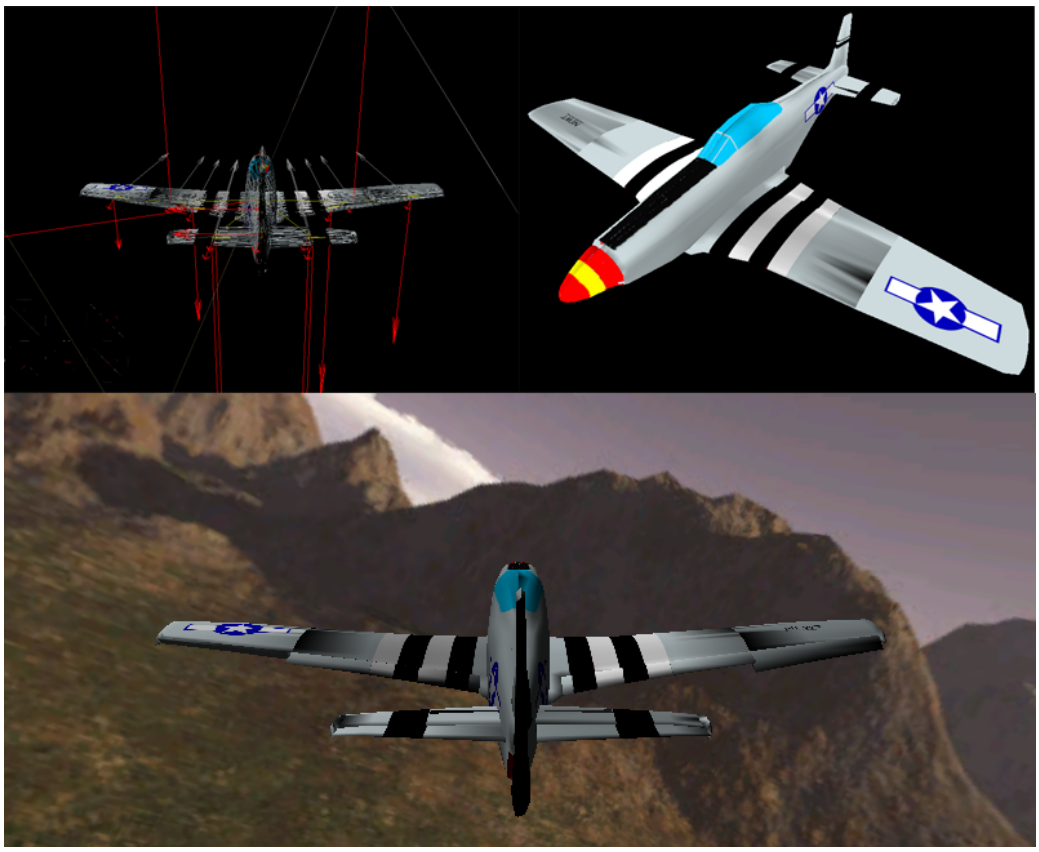

Figure 3: A collection of screenshots from the early prototype implementation. From top to bottom, left to right: a physics debug view illustrating the forces(red), joints(yellow), and velocities(white) of each component; a view of the model in the model viewer utility; a view of the model during flight (instrumentation not shown).

Our implementation at the time of writing consists of the majority of the simulation subsystem and a few minor features of the utilities subsystem. As the editor has not yet been implemented, we have created a default plane model using the Blender three-dimensional modeling system and assigned the model attributes manually. The model is a to scale approximation of a North American Aviation P-51D Mustang created using freely available blueprints and specifications, including airfoil specification. The model is composed of a fuselage root component, with separate components for each wing, stabilizer, and control surface, as shown above in Figure 1.

The physics model is minimal, but can easily be extended in a modular fashion. The components are considered to be of three types: a fuselage component, an airfoil component, and a stabilizer component. The fuselage model calculates thrust $(\vec{T})$ and parasitic drag $(\vec{D}_p)$ forces.

$$\vec{T} = C_t \cdot T_{max} \cdot \vec{f}_{local}, 0 \le C_t \le 1$$

where C_t is the coefficient of thrust, a control setting, T_{max} is the maximum output of the engine, and $\vec{f}_{local}$ is the local normalized forward vector.

$$\vec{D}_p = C_{d,p} \cdot A \cdot \left(\frac{1}{2} \cdot \rho \cdot |\vec{v}_{eff}|^2\right) \cdot (-1) \cdot \frac{\vec{v}_{eff}}{|\vec{v}_{eff}|}$$

where $C_{d,p}$ is the coefficient of parasitic drag, A was the area of the component orthogonal to effective velocity, ρ was the local air density, and $\vec{v}_{eff}$ the effective local velocity.

The wing models are considered airfoils, calculating their coefficients of lift $(\vec{L})$ and induced drag $(\vec{D}_i)$ for use when determining their respective forces. The respective equations were:

$$\vec{L} = C_l \cdot A_l \cdot \left(\frac{1}{2} \cdot \rho \cdot |\vec{v}_{eff}{}^2|\right)$$

$$\vec{D}_i = C_{d,i} \cdot A_l \cdot \left(\frac{1}{2} \cdot \rho \cdot |\vec{v}_{eff}{}^2|\right)$$

where C_l and $C_{d,i}$ are the coefficients of lift and induced drag, respectively, as determined based on angle of attack (AoA or α), and A_l is the lifting surface area of the airfoil.

Most of the control surfaces and the stabilizers are considered symmetric airfoils that generate no lift at zero degrees angle of attack, and therefore they are dominated primarily by parasitic drag depending on the angle of attack. Their drag force was directly proportional to the local α, and given by:

$$\vec{D}_p = \sin(\alpha) \cdot C_{d,p} \cdot A \cdot \left(\frac{1}{2} \cdot \rho \cdot |\vec{v}_{eff}|^2\right) \cdot (-1) \cdot \frac{\vec{v}_{eff}}{|\vec{v}_{eff}|}$$

In the case of the horizontal stabilizers, positive α followed the aerospace engineering standard of positive values indicating stabilizer toward the top and back of the stabilizer. In the case of the vertical stabilizer, positive α_{VS} was considered to be airflow towards the back and right of the stabilizer. The components are connected using joints from jMEPhysics. Each component is simulated to calculate the forces acting on it at a given timestep, and these forces are then input into the physics engine where they are composited based on the physics joints.

The simulator supports full control of the control surfaces, including ailerons, flaps, elevators, and rudder, as well as control over the level of thrust. Changes to the model made by the user only alter the angle of the specified control surfaces, ie ailerons; the physical model of each component then determines the realistic behavior of the plane. An instrumentation head-up display (Figure 4) give feedback to the user on current altitude, velocity, orientation, et cetera. The simulator also integrates a physics debug view from jMEPhysics, which allows the user to examine the forces acting on each component, their geometric and mass centers, their velocities, and other attributes in real time.

The user may return to the menu at any time during the simulation to access other functionality. The simulation may be reset, or the user may continue where they left. A viewer utility is available to the user at any time from the menu, which displays the current status of the plane model in the simulator, with free camera and scaling movement. This allows the user to examine any of the aspects of the forces, components, or joints in more detail at any time during the simulation.

Figure 4: An example of the instrumentation (artificial horizon is not shown).

We plan to complete the editor and provide basic statistical analysis functionality. We also plan to extend the complexity of the physics model to incorporate more realistic simulation techniques and environments, such as adding weather. The utilities subsystem will be developed incrementally as the other subsystems mature. At the time of writing the implementation is still in progress; for more up to date information, please visit http://cs.uga.edu/~parrish/newt.html.

5. CONCLUSIONS

We have presented an example of a framework for close integration between simulation and design for aerospace applications, and the amplification of their mutually beneficial relationship due to that integration. Such a framework is modular and flexible, allowing for a wide range of applications tailored to the specific needs of the user. Such an integration supports the application of computational optimization of designs. Finally, the close integration of the three of these system - simulation, design, and optimization - have the potential to greatly reduce prototype design time and cost.

6. ACKNOWLEDGMENTS

The authors would like to thank Dr. John A. Miller of the University of Georgia for sponsoring this project and providing directed guidance throughout its realization. They would also like to thank Hsin Hsiao Ma at the Georgia Institute of Technology for providing aerospace engineering and physics expertise.

7. REFERENCES

[1] Aerospace Industries Association. 2006. Annual Income Report for 2006. Aerospace Industries Association. http://www.aia-aerospace.org/stats/aero_stats/stat0806.pdf.

[2] Cantù-Pax, E. 1998. A survey of parallel genetic algorithms. *Calculateurs Paralleles, Reseaux et Systems Repartis, 10*, 2, 141-171.

[3] Gopher, D. et al. 1994. Transfer of skill from a computer game trainer to flight. *Human Factors*, *36*, 387-405.

[4] Jain, S., and McLean, C. R. Integrated simulation and gaming architecture for incident management training. In *Proceedings of the 2005 Winter Simulation Conference*, pages 904-913. Winter Simulation Conference, December 2005.

[5] Kincaid, J. P., et al. 2003. Chapter 19: Simulation in Education and Training. In *Applied System Simulation: Methodologies and Applications*, Obiadat, M. S. and Papadimitriou, G. I., Eds. Kluwer Academic Publishers, Norwell, MA, 437-456.

[6] H. H. Liu and H. Berndt. Interactive design and simulation platform for flight vehicle systems development. In *AIAA Modeling and Simulation Technologies Conference and Exhibit*, AIAA-2006-6812, 16 pages. AIAA Modeling and Simulation Technologies Conference and Exhibit, Keystone, CO, August 2006.

[7] Mansur et al. Rapid prototyping and evaluation of control systems designs for manned and unmanned applications. In *Proceedings of the American Helicopter Society 56th Annual Forum*. American Helicopter Society 56th Annual Forum, Virginia Beach, VA, May 2000.

[8] Padula, S. L. 1994. *Progress in multidisciplinary design optimization at NASA Langley*, *NASA Langley Research Center, TR No. NAS 1.15:107754, NASA-TM-107754*. NASA Langley Research Center, Hampton, Virginia.

[9] Padula, S. L. and Gillian, R. E. 2006. Multidisciplinary environments: a history of engineering framework development. *11th AIAA ISSMO Multidisciplinary Analysis and Optimization Conference*, Portsmouth, VA, September, 2006.

[10] Randel, J. M., et al. 1992. The effectiveness of gaming for educational purposes: a review of recent research. *Simulation & Gaming*, *23*, 261-276.