PhD proposal

Software System Development for Real-Time Simulations Coupled to Virtual Reality for Air-Traffic Control Applications

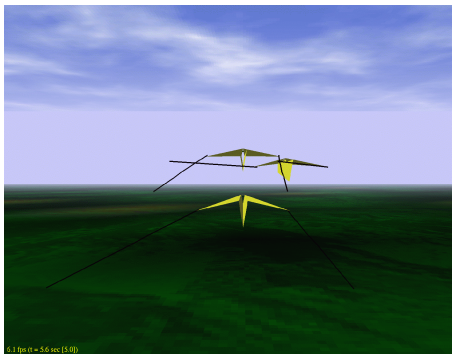
Anirudh Modi

4/20/2001

Advised by:

Prof. Lyle N. Long

Prof. Paul E. Plassmann



OUTLINE

- Motivation
- Introduction/Applications
- Integrated Virtual Environments
- Simulation Approaches
- Preliminary Results
- Timeline
- Questions/Suggestions?

Motivation

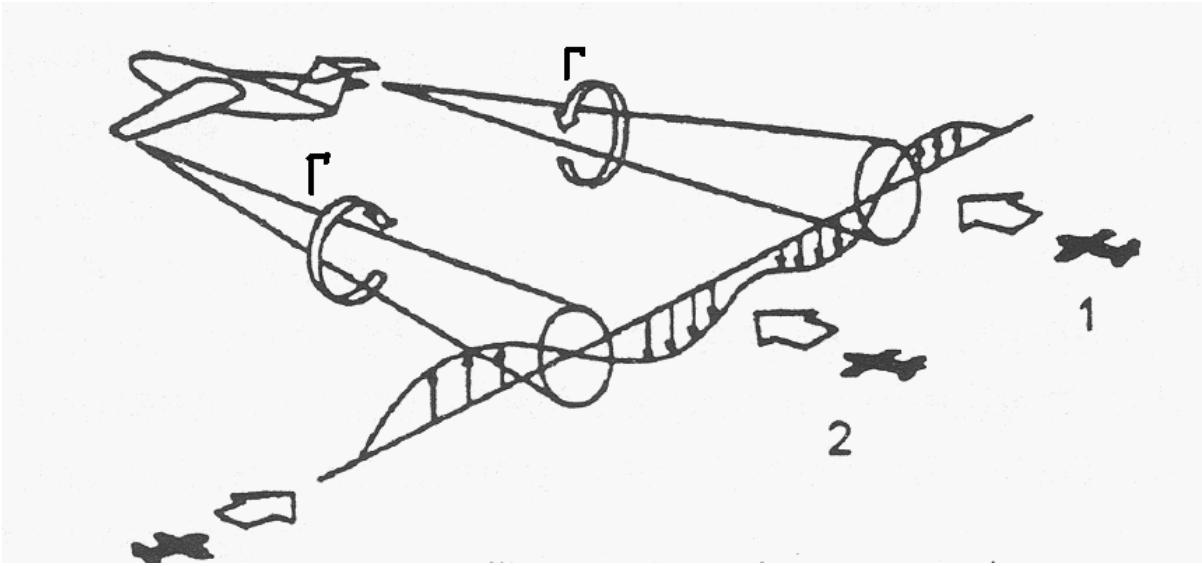
- Tremendous growth in speed and memory of parallel computers making complex simulations possible. Beowulf clusters are getting cheaper and more powerful by the day.
- Advances in hardware rendering systems and displays have made realistic Virtual Environments (VE) feasible.
- Combining VEs and complex simulations in real-time gives us a powerful tool to tackle a new world of problems [not much work has been done in this area].

Motivation

- NASA predicts that air-traffic will triple by 2022. If Air Traffic Control (ATC) systems do not improve significantly, there might be a major accident every week!
- VEs are becoming a very viable choice for future ATC systems.
- Wake-vortex hazard problem is major bottleneck for airport capacity, and a challenge for ATC.
- Wake-vortex prediction for an entire fleet of aircraft taking-off and landing at a busy airport is a computationally intensive problem.

Wake-Vortex Hazards

- Moving aircraft generate wakes in the form of two counter-rotating swirling rolls of air, termed *wake-vortex pair*.
- These wake-vortex pairs stretch for several miles behind the aircraft and last for several minutes.



Wake-Vortex Hazards

Wake-vortex generated by a Boeing 727 (Courtesy NASA)

Wake-Vortex Hazards

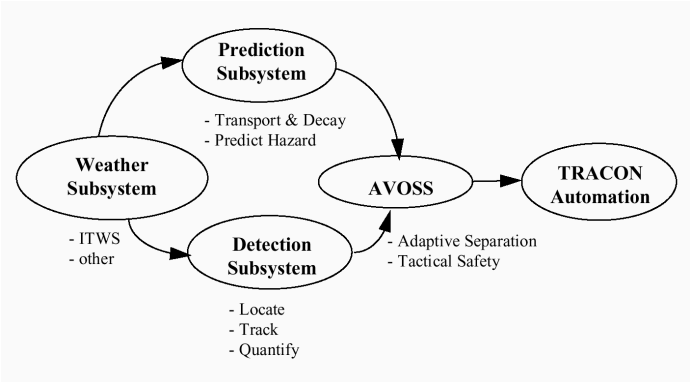
- The rapid swirling of air in a vortex can have a potentially fatal effect on the stability of a following aircraft.
- These vortices are usually invisible making the problem worse.
- Even a commercial aircraft like a Boeing 737 can be thrown out of control it is follows too close behind a large aircraft such as a Boeing 747 (or even a smaller 727).
- The infamous US Air Flight 427 (Boeing 737) disaster on September 8, 1997 is attributed to this phenomenon (following a Boeing 727).

Wake-Vortex Hazards

- To avoid such accidents, the International Civil Aviation Organization (ICAO) lays down strict rules for aircraft spacing based on aircraft size.
- e.g., a small aircraft should follow atleast 6 nautical miles (6.9 miles) behind a heavy jet such as a Boeing 747.
- These spacings are very conservative to compensate for the lack of understanding of the strengths and positions of the vortices (based on "worst-case" scenario).

Current proposed system: AVOSS

- NASA researchers have designed a system to predict wake-vortices: Aircraft VOrtex Spacing System (AVOSS).
- AVOSS determines how winds and other atmospheric conditions affect the wake-vortex patterns of different types of aircraft.
- It integrates the output from a number of subsystems: weather, wake prediction, wake sensors.



Current proposed system: AVOSS

- AVOSS is being extensively tested at the Dallas/Fort Worth (DFW) airport since 1997.
- Although AVOSS carries out a rigorous simulation of wake-vortices, it does not implement any system for their visualization.
- Hence, it is unable to provide information like alternate trajectories for the take-off and landing of aircraft, etc.

Other applications: Aircraft Noise

- Aircraft noise has been a quality of life issue near airports for many years and is another good example of the need to link real-time simulations and visualization.
- Airports in the U.S. are facing strong resistance to adding new runways because of the increase in sound levels.
- FAA projections show a 3-5% annual increase in air-travel, hence noise control is an important issue.
- Future airport planning and runway, landing/take-off schedule should include the effect due to aircraft noise to control the noise levels.

Other Applications

- Proposed software system is very general in nature.
- Can be used for almost any application that requires a virtual reality based interface for real-time parallel simulations.
- e.g., satellite image processing and visualization, weather forecast and visualization, molecular dynamics simulations, real-time flow simulations over simple aircraft and ship geometries, real-time vehicle simulations, vehicle traffic management, vision-based object-tracking and simulation, etc.

Integrated Virtual Environments

- Virtual Reality (VR) represents a Human-Computer Interface (HCI) technology designed to leverage our natural human capabilities.
- VR removes the tight, unnatural, two-dimensional constraints and allows the user to interact with real-time 3D graphics in a more intuitive and natural manner.
- A VR system that interactively simulates a particular environment or scenario in real-time is termed an “Integrated Virtual Environment” (IVE).
- An IVE lets the user experience the data directly by providing an interactive simulation via a 3D output device.

Integrated Virtual Environments

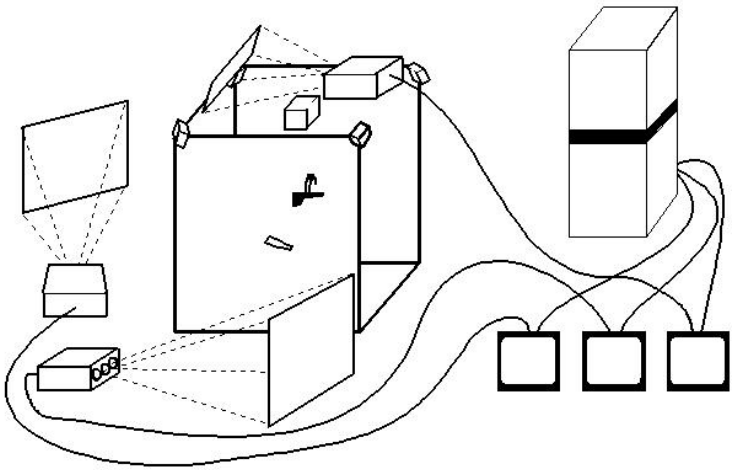
- Wickens and Baker have defined VR as incorporating the following five features:
  1. Three-dimensional viewing.
  2. Dynamic display (moving images).
  3. Closed loop (interactive).
  4. Inside-out (user-centered) frame of reference.
  5. Multimodal interaction.

Integrated Virtual Environments

- VR is broadly classified as:
  - Desktop VR or Window on World (WoW) systems
    - Monitor
  - Video mapping
    - Variation of WoW; user's silhouette merged with 2D/3D graphics; e.g., Doom, Quake
  - Immersive Systems
    - Most popular; e.g., BOOM, HMD, CAVE, RAVE
  - Telepresence
    - Robots, remote space exploration, surgery
  - Augmented/Mixed Reality (AR)
    - Mix of Real and Virtual worlds; e.g., highway in the sky, surgery
  - Fish Tank Virtual Reality
    - stereo LCD shutter glasses with head tracker to look at a monitor.

VR Hardware

- **CAVE**: Cave Automatic Virtual Environment
  - Most popular as well as sophisticated VR installation; developed in 1992 by EVL, UIC.
  - Projection-based VR system that surrounds the viewer with 4 or more screens.



CAVE diagram

VR Hardware

Screenshot of a CAVE application (Courtesy: SARA, Netherlands)

VR Hardware

- **RAVE** (Reconfigurable Automatic Virtual Environment) (1-4 screens) and **Immersadesk** (1 screen) are smaller alternatives to the CAVE.
- Input devices
  - Lots of them available; e.g., motion trackers, gloves, wands, 6D spaceballs, force/haptic feedback device, 3D digitizers, etc.
  - Most can be added to existing VR installation like a CAVE or RAVE very easily.

VR Software

- VR hardware devices are very complex.
- Specialized Application Programming Interfaces (APIs) are essential to make the development of VR applications easier.
- CAVELib is an API that provides general support for building virtual environments for various types of immersive displays.
- It was originally developed by Dave Pape of EVL, UIC, and is now marketed commercially by VRCO.

CAVELib

- CAVELib configures the display device, synchronizes processes, draws stereoscopic views, creates a viewer-centered perspective and provides basic networking between remote VEs.
- Provides standardization: Allows single program to be available on a wide variety of display devices.
- It uses threads to obtain simple parallelization in the calculation process by using by using one process for projection on each wall.
- VRJuggler is another C++ API similar to CAVELib (C). Unlike CAVELib, VRJuggler is public domain software, developed and maintained by Iowa State University.

OpenGL

- CAVELib by itself does not incorporate functions for the actual graphics programming (display). OpenGL API is used for that.
- OpenGL is an industry standard for graphics programming and is thus portable across platforms
- With the combination of OpenGL and CAVELib, powerful VR applications can be created.
- OpenGL is enhanced by the use of additional utility libraries like GLUT and GLX.

Other High-level Software

- While CAVELib and OpenGL provide basic functionality for development of VR applications, they are usually not sufficient and are often cumbersome.
- They lack important software components like one for *hierarchical scenegraph rendering*, which is necessary for efficient rendering of complex scenes common in most VR applications.
- SGI's IRIS Performer provides this, but works only under SGI and Linux. Most existing CAVE applications are written in CAVELib+OpenGL+Performer.

Other High-level Software

- Several high-level software development systems are available for building high-performance, real-time, integrated 3D applications. e.g., WorldToolKit, MR Toolkit, dVISE, DIVE.
- WorldToolKit (WTK) by Sense8 Corp is the most popular. It is an object-oriented toolkit that greatly simplifies programming of VR applications.
- It has 1100+ high-level functions.
- It integrates high-performance graphics, intuitive interface components, 3D or spatialized 3D sound, support for multiple database and media formats, and a comprehensive list of I/O peripherals.

Other High-level Software

- CAVERNSoft is another toolkit for creating high-performance tele-immersive applications (developed by EVL).
- It provides networking and database functionality needed in a VR system.
- CAVEStudy is another toolkit built on top of CAVERNSoft. It allows interactive and immersive analysis of a simulation running on a remote computer.
- Disadvantage: Both CAVERNSoft and CAVEStudy depend on Performer for rendering, and are hence limited to SGI IRIX and Linux platforms.

Air-Traffic Control (ATC)

- ATC related delays cost the airline industry an estimated US $5.5 billion annually!

Year	Passengers (Millions)	Passenger Miles (Billions)	Jet Aircraft in Fleet	Domestic Departures (Millions)
2000	639.8	661.8	5610	7.4
2005	767.4	830.3	6903	8.6
2010	931.1	1041.6	8360	9.9
2015	1139.7	1306.2	10034	11.5

- According to FAA, if ATC systems do not improve significantly, there could be a major accident every 7-10 days by 2015.
- FAA has been trying hard to replace and modernize ATC systems for the past two decades.

Air-Traffic Control (ATC)

- U.S. ATC is organized around three types of facilities:
  1. Airport towers: monitor aircraft on ground and give take-off and landing clearances.
  2. Terminal Radar Approach CONtrol (TRACON): handle aircraft ascend and descend to and from airport.
  3. Enroute centers: handle aircraft flying between airports at higher altitudes.
- ATC is a highly demanding human-machine system.
- Intelligent systems combining satellite navigation and automated speech synthesis/recognition are expected to replace some human functions of ATC in the near future.

Air-Traffic Control (ATC)

- Next logical step is for ATC to transform into an IVE.
- FAA has declared that ATC system will eventually transition into a new system known as **“Free Flight”.**
- Free Flight will reduce centralized control by allowing pilots greater freedom in choosing alternate routes.
- Pilots will be allowed to change their routes in real-time, with controllers intervening only when necessary to ensure adequate separation.
- This is expected to reduce costs and increase airport capacity.

Air-Traffic Control (ATC)

- Azuma et al at Hughes Research Lab (HRL) have worked extensively on advanced HCI and visualization tools to assist in the Free Flight air-traffic management.
- They have created a VR based testbed using C and WTK 7 for visualization of conflicts that may occur in a “Free Flight” scenario.
- However, they do not deal with any vortex-wake problem, and limit their work to conflict resolution algorithms.

Air-Traffic Control (ATC)

- Among other improvements to air-traffic systems, NASA has been working on a "Highway In The Sky" (HITS) system, which will use Augmented Reality to guide the pilots on a preprogrammed destination on a "virtual highway". This should ease the pressure on the ATC.
- NASA has also been working on FutureFlight Central, a national ATC test facility dedicated to solving the present and emerging capacity problems of airports.
- It offers a 360-degree full-scale real-time simulation of an airport, where controllers, pilots and airport personnel interact to optimize operating procedures and to test new technologies.

Air-Traffic Control (ATC)

NASA FutureFlight Central's tower cab

Simulation Approaches

- Parallel Computing
  - Beowulf Clusters
  - MPI
- Real-time Monitoring and Steering
  - Previous Work
  - Proposed approach
- Vortex-Wake Hazards
  - Problem Size and Other Complexities

Parallel Computing

- Strategy for performing large, complex tasks faster by using multiple computers/processors which work together on a common task which is decomposed into smaller tasks.
- Motivation: single processor computers have limited amount of memory and speed.
- Parallel computing not only allows us to solve problems that do not fit on a single processor machine, but also does it faster.
- It opens new horizons: we can solve complex computational problems in reasonable amount of time, and some in real-time too.

Parallel Computing

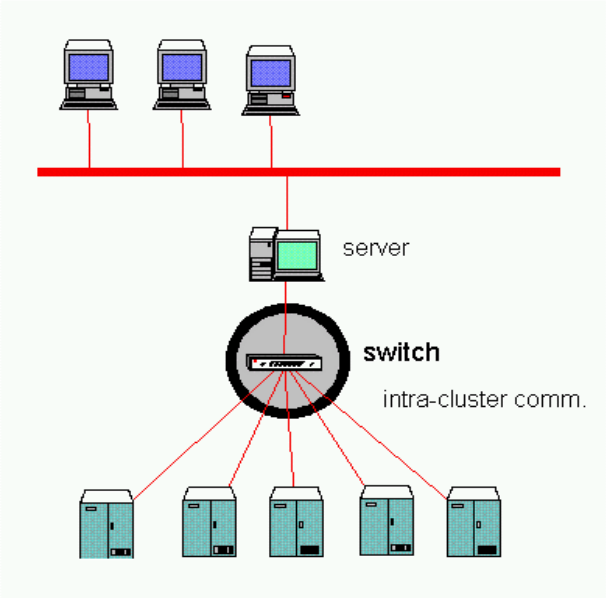
- Two broad classes: Functional Decomposition (break the tasks) and Domain Decomposition (break the data).
- A common way is to use the Single Program Multiple Data (SPMD) model.
  - The same code is replicated to each process but works on different data.
  - One process acts as the master and distributes and collects work from the other processes.
  - Each processor works on its section of the problem and is allowed to exchange information with other processors.

Beowulf Clusters

- Multi-computer architecture which can be used for parallel computations.
- Uses commodity personal computers, standard network adaptors and switches.
- Does not contain any custom hardware components and is trivially reproducible.
- Using commodity (and usually public domain) software like the Linux OS, MPI, and other widely available open-source software.
- First Beowulf cluster was built by NASA in 1994 (consisted of 16 486DX4-100 MHz machines).

Beowulf Clusters

- Price-performance (or $/Mflops) for these clusters is far better (often 10 times or more) than the traditional supercomputers from SGI, IBM or Hitachi.
- Ideal for small organizations and research groups.

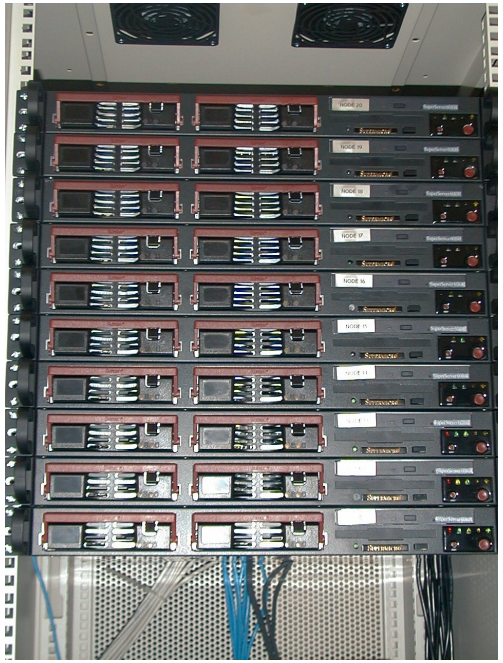


Schematic of a typical Beowulf Cluster

Beowulf Clusters

- IHPCA has its own clusters: COCOA (COst effective COmputing Array) and COCOA-2.
- COCOA: 50-proc PII-400 cluster with 12.5 GB RAM, fast ethernet
- COCOA-2: 40-proc PIII-800 cluster with 20 GB RAM, dual fast ethernet

COCOA



COCOA-2

Message Passing Interface (MPI)

- MPI is a specification for message passing libraries. It is designed to be a standard for message passing in parallel computing, especially for distributed memory machines.
- It is highly portable and easy to use.
- It has over 155 constructs using which a programmer can parallelize his/her program.
- Written in C and have C++/FORTRAN/FORTRAN90 bindings.
- Several open-source MPI libraries are available for most operating systems and architectures (MPICH, LAM-MPI).
- Several other parallel programming paradigms available: PVM, OpenMP, etc. but not very popular.

Real-Time Monitoring and Steering

- Running a complex program on a high-performance computing (HPC) system poses major difficulties in observing output.
- Usually, simulation severely limits interaction with the program during the execution.
- Makes visualization and monitoring very slow and cumbersome (if at all possible).
- It the program is run remotely, additional difficulties arise.
- How do we monitor (observe the output) and steer (change the input) for such a program in real-time?

Real-Time Monitoring and Steering

- Software tools that support these activities are called computational steering environments.
- They operate in three phases:
  1. Instrumentation: Application code is modified to add monitoring functionality.
  2. Monitoring: Program is run with some initial input data, the output of which is observed by retrieving important data about the program's state change.
  3. Steering: Program's behavior is modified (by modifying the input) based on the knowledge gained during the previous phase by applying steering commands, which are injected on-line.

Real-Time Monitoring and Steering

- Several well-known computational steering systems exist: Falcon from Georgia Tech, VASE (Visualization and Application Steering) from UIC, SCIRun from Univ of Utah, CUMULVS from Oak Ridge National Lab, CSE (Computational Steering Environment) from Center of Mathematics and Computer Science in Amsterdam, Virtue from UIUC.

	Scope			User Interface
	model	algorithm	performance	
Falcon			x	2-D visualization; steering user interface
VASE	x	x		visualization through existing packages; steering through textual input
SCIRun	x	x		visualization modules; steering user interface
CUMULVS	x		x	visualization extern through AVS; steering through textual input
CSE	x			graphical editor for multi-dimensional data sets; steering GUI
Virtue			x	interactive VR visualization and steering

Summary of characteristics of existing steering systems (Courtesy Reitinger)

Computational Steering and VR

- Computational steering has also been attempted by coupling a 2D solver with the CAVE by Carolina Cruz-Neira et al. They worked on simulations of simple Rayleigh-Taylor instability problem.
  - Provided a menu to change input parameters.
  - Simulation would restart when a parameter was changed.
- Mulder et al. have also worked on computational steering I the CAVE environment and have designed an intuitive user-interface to facilitate this.
- Juler et al. have implemented a software architecture called Dragon for real-time battlefield visualization.
- Taylor et al. have done some interesting work which explores the lag time involved in interactive VR applications.

Proposed Approach

- For computational steering, an approach similar to Falcon will be used.
- Falcon is an real-time monitoring and steering toolkit for large parallel programs.
- Consists of steering server and steering client.
- Steering server is a separate execution thread of the application to which local monitors forward the relevant monitoring events.
- Steering client receives the application run-time information from the application, displays it, accepts steering commands, and enacts changes that affect the application's execution.
- Communication between application and steering client, and between steering client and server, is handled by the transmission tool, *Data Exchange*. Data Exchange uses UNIX sockets for the actual communications.

Wake-Vortex Hazard Simulation

- For steering, the real-time computational steering method described previously will be used.
- Parallel numerical wake-vortex prediction code is written in C++ and MPI.
- It will be running on a remote Beowulf cluster and will be communicating to the computational steering server using network sockets.
- RAVE will be used as the display device, which will be driven by an HP Visualize J-class workstation running HP-UX.

Wake-Vortex Hazard Simulation

- A typical metropolitan airport in US is very busy.
- Dallas/Fort Worth (DFW) airport (3rd busiest): 7 runways, handle nearly 2,300 take-offs and landings everyday!
- For the wake-vortex code to track the vortices shed by an aircraft for 5 miles after take-off, assuming that a vortex code is tracked every 5 meters, 5x1600/5x2=3200 vortex filaments have to be tracked.
- For 2,300 take-offs and landings: 3200x2300/24=306,667 vortex filaments have to be tracked every hour.
- An average vortex does not decay before 15 minutes, therefore vortices due to typically half the take-offs and landings every hour need to be tracked at any given time. This is roughly 153,333 vortex filaments.
- However, the wake-vortex simulation is an $O(N^2)$ problem. Even if the induced velocity effect due to vortices from other aircraft are ignored: 3200^2=10.24 M computations/airplane/timestep.
- For 2300 planes/day => 10.24x2300/24/2 = 490.7 M calc/timestep!!

Wake-Vortex Hazard Simulation

- Code is parallelized to track vortex elements from each plane on a different processor, so that we get an almost real-time solution with tolerable lag (thanks to the "dimensional reduction" in communication).
- Aircraft keep entering and leaving the airport => data-structure in the wake-vortex program should be able to handle this. Hence, doubly linked list is used for the aircraft data-structure, which adds an aircraft in constant time, and deletes in linear time.
- Special data-structure has to be maintained for wake-vortices, as they may remain even after the aircraft has left the domain of interest.

Wake-Vortex Hazard Simulation

- The trajectory for each aircraft needs to be continuously supplied using network sockets. In practice, this will come from the GPS on the aircraft, but for our case, there will be a remote subroutine generating a random (but practical) trajectory.
- The VR application will be written in C++ using OpenGL and GLUT (openGL Utility Library) on top of CAVELib.
- A noise prediction code will also be run to generate the noise levels around the plane due to its engines (as another example of the approach).

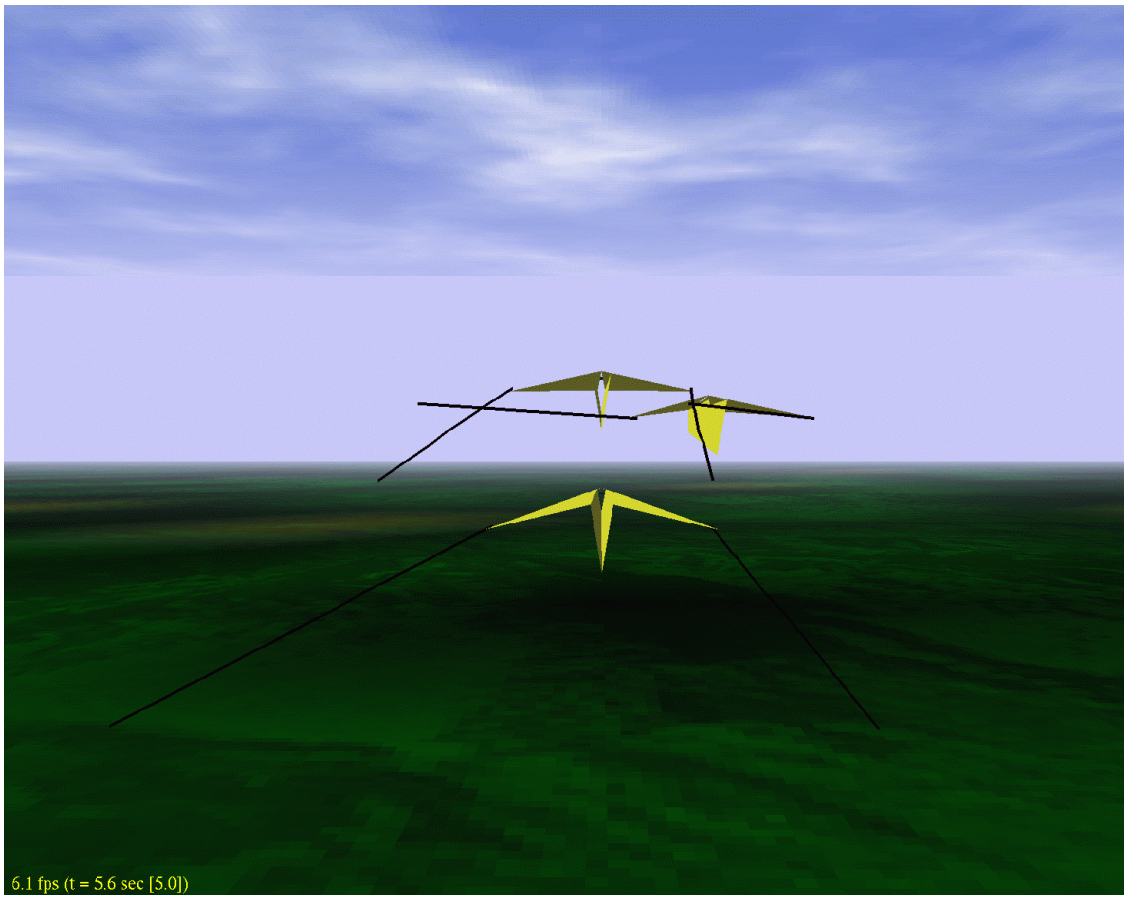
Preliminary Results

- A wake-vortex prediction code has been written in C++ using the Standard Template Library (STL). STL provides optimized and highly efficient implementation of basic data-structures employed. Due to the proposed real-time nature of the program, this is very important.
- While the wake-vortex prediction model is very basic, it performs adequately for the purpose of testing the entire system as a whole.
- Wake-vortex prediction code is written as a separate library, so as not to interfere with the internals of the VR code.
- Eventually, instrumentation will be added to this code, so that monitoring and steering of the application can be carried on smoothly.

Preliminary Results

- Currently, the aircraft trajectory is provided through a separate input file for each aircraft. Eventually, this will be provided over the network in real-time from a remote location.
- A basic OpenGL and GLUT based visualization program to display the output from the wake-vortex code has been written and successfully integrated with the wake-vortex library.
- The current implementation works well on the RAVE system.
- CAVELib support will be soon added to achieve the stereo effect and to use the tracking equipment.

Preliminary Results



A Screenshot of the OpenGL program

Timeline

- Aug ‘97-May ‘99: MS in Aerospace Engg under Prof. Lyle N. Long. Worked under a thesis involving Parallel Computing and CFD. Built and managed COCOA during that period.
- Aug ’99-May ‘00: Enrolled in the CSE PhD program. Completed 6 courses and passed the PhD candidacy exam, English competency exam, presentation skills test and oral competency test.
- May ’00-Aug ’00: Did basic research on VR. Attended the CAC seminar on CAVE programming and attended a 3 day CAVE workshop at University of Michigan.
- Aug ’00-Dec ’00: Started research on computational steering, visualization and VR. Took the VE course under Dr. Sharma.
- Jan ’01-Apr ’01: Final topic for PhD decided. Did literature survey on the wake-vortex hazard problem. Wrote intital code in C++ and OpenGL. Prepared the PhD proposal.

Timeline

- Jun ’01-Aug’01: Will add CAVELib support. Enhance the graphics and make the airport look more realistic. Also work on parallelizing the wake-vortex code and instrumenting it for steering. Will be presenting a paper on “Beowulf Clusters” in the Linux Revolution Conference at UIUC on June 26.
- Sep ’01-Oct ‘01: Complete the computation steering work and ensure that it works well on the RAVE. Deal with all the complexities that arise in the parallelization and steering of the code. Also start working on the aircraft noise simulation problem.
- Nov ’01-Dec ‘01: Get the aircraft noise simulation work completed.
- Jan ’02-Feb ‘02: Try to get realistic input for the weather conditions, etc and simulate a busy airport. Do some analysis of the system and look for possible improvements.
- Mar ’02-Jun ’02: Write the final thesis dissertation, do final analysis and wrap up all work.
- Jul ’02: *Final PhD defense*.