(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
15 August 2002 (15.08.2002)

PCT

(10) International Publication Number
WO 02/062436 A2

(51) International Patent Classification[7]: A63F

(21) International Application Number: PCT/US02/03881

(22) International Filing Date: 6 February 2002 (06.02.2002)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/778,185 6 February 2001 (06.02.2001) US

(71) Applicant: **ELECTRONIC ARTS INC.** [US/US]; 209 Redwood Shores Parkway, Redwood City, CA 94065 (US).

(72) Inventor: **PISANICH, Gregory, M.**; Morgan Hill, CA (US).

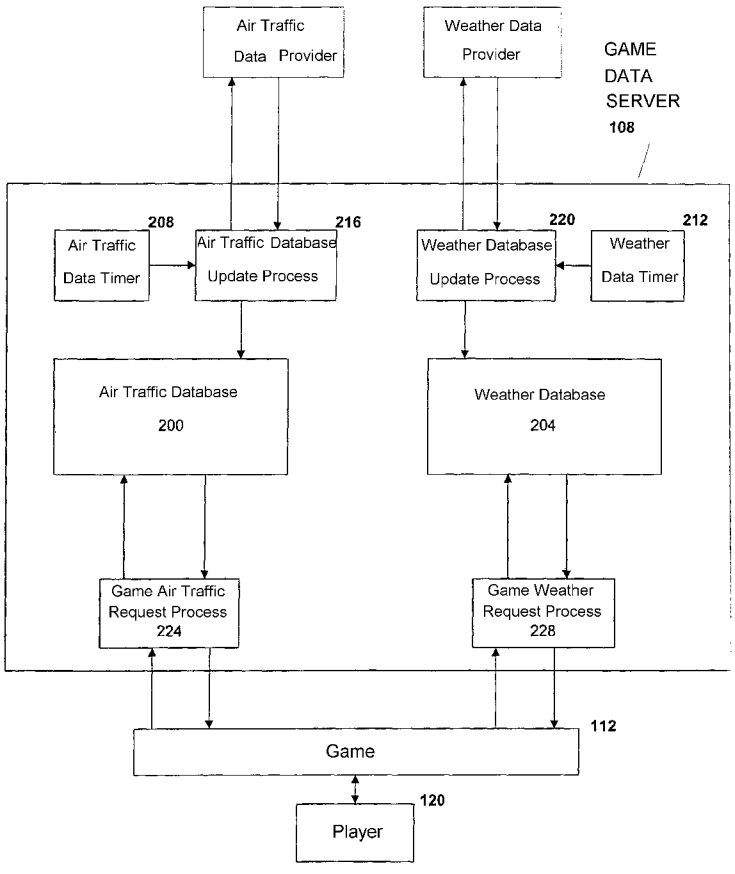
(74) Agents: **GRAY, Gerald, T.** et al.; TOWNSEND AND TOWNSEND AND CREW LLP, Two Embarcadero Center, Eighth Floor, San Francisco, CA 94111 (US).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, OM, PH, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TN, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZM, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

[Continued on next page]

(54) Title: INTEGRATION OF REAL-TIME DATA INTO A GAMING APPLICATION



(57) Abstract: Real world data is integrated into a gaming experience to influence the game and actions of a player. A game module creates game elements from the real world data associated with the location. Weather conditions are simulated based on weather information for different observation locations in the world, and traffic conditions are simulated as well. In a flight simulator game, the game integrates real aircrafts and real weather conditions as the player's aircraft navigates in the world.

WO 02/062436 A2

Published:
— *without international search report and to be republished upon receipt of that report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

INTEGRATION OF REAL-TIME DATA INTO A GAMING APPLICATION

Field of the Invention

The present application relates to electronic games, and more specifically to games using real world data to simulate game play.

Background of the Invention

In conventional computer gaming experiences, the player is immersed in a closed environment driven by the application developer. Even in games simulating real world locations, the environmental conditions, actors, and objects are predefined and only vary according to predetermined parameters or algorithms, or, at best, incorporate pseudo-random changes. The player's experience is confined to the boundaries of the application developer's imagination and the player's own selections. Conventional gaming architectures are only designed to take advantage of the known conditions within which the game will operate, and are not designed to account for new or randomly changing elements of the real world.

Conventional games deprive the player of the excitement and variability that real world conditions provide. For example, in a flight simulator game, the player typically chooses different weather conditions, traffic conditions, and other features of the gaming environment. The application generates a simulated environment which then conforms to those parameters. As such, conventional flight simulators may "simulate" external data but this data does not reflect actual current weather or traffic conditions. Conventional flight simulator add-ons may also be executed prior to initiating the game to update cloud data files for the flight simulator to provide cloud coverage reflective of real world conditions at the time of the update. However, the

environment within the game is updated only this one, pre-execution, time, and therefore does not change during the game and has no capacity to enable the real time data to affect game play. In contrast, in the real world, the weather can change suddenly and unexpectedly, air traffic can become unexpectedly heavy, mechanical failures can occur, airports can close, planes crash occasionally, and actual pilots must adapt to these conditions. However, none of these events are modeled in a typical game, or, if they are modeled, they generated in some non-random pattern or occur upon selection by a user. Even if the conditions occur in a psuedo random pattern, the player quickly learns the range of possible conditions and is still deprived of the excitement of knowing that conditions he or she are flying in are "real" which can provide a significant difference in enjoyment level for players.

Accordingly, a new gaming architecture is needed in which real world events, environments, actors, and objects can be incorporated during a gaming experience, providing a player with the true simulation of a real world experience, and also the added excitement of taking part in actual real world events as they occur.

Summary of Invention

In accordance with the present invention, real world information is incorporated into a virtual environment provided for a game. In a software product embodiment, an application module receives a location selection from a player. The location selection selects a geographic location within which the game is to take place, for example, San Francisco. The application module then retrieves real world information. The real world information may differ depending on the nature of the game. For a flight simulator game, the real world information includes weather information such as cloud cover, precipitation, wind, fog, as well as air traffic data,

including the location of other planes near the player, altered routes of other planes, and airport conditions, for the selected location, at a time period at or near to the time the information is requested. After receiving the real world information the application module generates simulated real world constructs responsive to the real world information. For example, in a flight simulator game, constructs may include clouds, wind, rain, and air traffic. Clouds, for example, are generated in the virtual environment responsive to the location and nature of the clouds as identified in the real world database. Air traffic, i.e., other airplanes flying near the player's plane, are generated responsive to actual locations of real world airplanes. In a car racing embodiment, car traffic on the simulated streets is generated responsive to real world car traffic on the corresponding real world streets. Thus, in accordance with the present invention the real world is simulated for the player to provide the player with the excitement of the inherent unpredictability of the real world.

In one embodiment, the application module determines whether the information for the location has changed since it was last retrieved or if the player has moved to a new location. If either has changed, the simulated constructs are regenerated responsive to the information, or the simulated constructs are generated responsive to an interpolation procedure. This allows real world information to be used in gaming environments when the information is not available continuously. For example, currently, air traffic information is only available every two to three minutes, and therefore the traffic displayed to the player is interpolated until the new information is received into the real world information database based on prior heading and flight plan information. Weather data is not available at every location, and therefore weather data is simulated for those locations with which real world data is not associated. Additionally, all weather elements that can be depicted are not represented in weather database, and therefore the

existence of these weather elements must be inferred from the available data. Thus, the present invention allows the real world data to interact with the player. Thus, if traffic is heavy on a particular day, the player must fly through heavy traffic, if the weather is bad, the player must use his skill to combat the adverse weather conditions, all in response to the actual weather or traffic currently occurring in the world at a selected location. Thus, a system, method, and application are provided for integrating real world data into a database to effect the ongoing gaming experience of a player.

Brief Description of the Drawings

Figure 1 is a block diagram of a system for incorporating real-time data in accordance with the present invention.

Figure 2 is a more detailed block diagram of a game data server in accordance with the present invention.

Figure 3 is a block diagram of a game system in accordance with the present invention.

Figure 4 is a block diagram of a weather manager in accordance with the present invention.

Figure 5 is a block diagram of weather element dependency in accordance with the present invention.

Figure 6 is a diagram of a camera pass in accordance with the present invention.

Figure 7 is a diagram illustrating interpolation by location in accordance with the present invention.

Figure 8 is a diagram of a cloud object grid in accordance with the present invention.

Figure 9 is a block diagram of the traffic manager system in accordance with the present invention.

Detailed Description of the Preferred Embodiments

Although the description herein uses a flight simulation game as an exemplary embodiment for the teachings of the present invention, the present invention is not limited herein and can easily be extended to other games such as car racing, ship racing, bike racing, skiing, or any other type of competitive outdoor gaming activity. In a flight simulator embodiment, two real world conditions that are useful to be integrated into a game are the weather and the air traffic. Weather and air traffic data are both gathered by third parties and are made available in publicly accessible databases. The databases store information regarding certain weather conditions for certain locations throughout the world (called "observation locations") and store traffic information including the location, position, heading, and type of aircrafts. The present invention uses this information to create an exciting, realistic, and unpredictable gaming experience for a player of a game created in accordance with the present invention.

Referring to Figure 1, a game data server 108 continually gathers "real time" data from traffic 100 and a weather 104 data provider systems. On request, the game data server 108 permits data available to a game 112, 116 and players 120, 124. The data providers 100, 104 are located remotely and are typically accessed via the internet or through some other communications means (such as a satellite, or a leased line). The data providers 100, 104 are conventional and gather and encode observations about the world (weather, airplane traffic, etc.) and make it available in real-time (as quickly as it is processed and available) in a known format and structure. The game data server 108 of the present invention accesses the data and transfers the data to a database storage area. Transferring a "snapshot" of the data from the providers 100, 104 to the game data server allows the main load of data access from the games 112, 116 to be

handled between the games and the game data server 108 and not between the games and the providers.

An example of the weather data provided includes the temperature, wind direction and velocity, cloud cover and type at several altitudes, and type of precipitation. An example of air traffic data would be the position (latitude and longitude), speed, heading, identification, and flight plan for each aircraft within a geographic area (for instance, over the United States). The real time nature of the data is related to the rate of change of the process monitored and the frequency is which it is available for update. The weather data is normally updated by physical (by a person) or electronic means, for example, in the METAR database, and is made available once an hour, which is sufficient to capture the normal speed of change of weather conditions. In special cases (in the case of hurricane force winds, or extremely bad weather), the weather data may be updated more often. For this discussion, real-time is considered once an hour, or, as often as necessary to reflect the perceptible rate of change in the real world. The national air traffic control system updates traffic data once every three minutes typically to capture the positions, headings and speed changes of the aircraft. This is sufficiently fast enough to allow the positions of aircrafts to be interpolated in accordance with the present invention to provide a realistic simulation of the flight path of the aircrafts.

As a player 120 controls a gaming unit within a game 112, for example, an aircraft, a car, or the like, the game 112 requests data from the game data server 108 as required to provide a realistic environment for the game. The data requested by the game 112 includes data such as weather reports or air traffic in the direction that the player 120 is guiding the vehicle in the game 112. Thus, the game 112 requests data based on a current location and time of a player gaming unit. The time information may be explicitly or implicitly provided. The frequency of

the game's data requests preferably conforms to the frequency of updates of the data in the databases maintained by the providers 100, 104. This allows the game 112 to provide a real time representation of the world and the vehicles in the world to the player at the fastest rate possible. Alternatively, the requests may be asynchronous, and request a large amount of data to be used for longer periods of time. The game data server 108 processes the data requests of the game 112 and provides data requested to the game 112 for rendering and simulation within the game 112. Multiple players 124 can play different copies of the game 116 and can access the game data server 108 for data responsive to some other area of the world or the same area of the world as the player 120. Moreover, a second player 124 may play an entirely different game 116 (for instance, a car racing game, or a football game) than the game 112 being played by the first player 120. However, the different game 116 may request the same type of data (weather, for example) as the first game 112. The architecture of the present invention can be extended to meet the data requirements of other games. For example, in addition to weather reports, a car road racing game may generate and use a database that stores real time data about automobile traffic. A maritime game may collect and store real time data about sea conditions that could be used in the game and so forth.

Referring to Figure 2, the game data server 108 maintains its own air traffic and weather databases 200, 204 using data requested from the external air traffic 100 and weather 104 data providers as described above. To control the type and frequency of requests for data made of the external data systems 100, 104, the game server 108 uses timer processor 208, 212 to initiate the data requests. At start up, or any full reset, the timer processes 108, 212 instruct data requestors 216, 220 to request all current data available about weather and traffic. This snapshot of data is used by the data update processes 216, 220 to initially populate the databases 200, 204. After the

databases 200, 204 are fully populated, the timers 208, 212 then instruct the data request processes 216, 220 to request data and update the respective databases 200, 204 at the best update frequency and time for the individual data. As discussed above, in the example of weather data, this would be once an hour at approximately 10 minutes past the hour. In the case of air traffic data, this would be approximately every three minutes. The update times and frequency used by the timers 208, 212 are chosen and optimized to match the update times and frequency of the data providers 100, 104.

As a player 120 plays the computer game 112, the game will make requests for air traffic and weather data corresponding to the change in location and in time of the player's gaming unit. The requests for data are preferably made through separate air traffic 224 and weather request 228 processes. The game 112 requests data in a form that specifies the data needed. For example, the game 112 may specify the current position (latitude, longitude, and altitude), state heading, and speed of the game aircraft in flight simulation game. The requests for data by the game 112 are created in response to choices and control inputs by the player 120, e.g., where and how the player 120 is flying a plane in a flight simulation application. The game data requests are used by the air traffic request process 224 and weather request process 228 to generate data queries for the respective traffic and weather databases 200, 204.

For example, in one embodiment, to determine the most appropriate data, the request processes 224, 228 consider the aircraft performance (747 vs. Piper cub), state (cruise, landing, takeoff, taxiing), current aircraft data (position, heading, speed and direction) and the previous request (position, heading, speed, and direction). Aircraft state is implied by the position of the aircraft controls (for example, a landing state could be implied for a 747 style aircraft as: speed

below 120kts, gear down, flaps > 15) and position of the aircraft (below 4000 feet and within 10 miles of an airport).

The request processes 224, 228 gather the data at a frequency and distance around and in the direction of movement of that aircraft appropriate for that aircraft and state. In the previous example of the landing 747, the request processes 224,228 request data within a circle 30 miles around the aircraft at a slow (once every 10 minute) rate, because the plane's location changes less frequently than when it is in full flight. While in a cruise state, a 747 typically flies at over 400 miles an hour at 35,000 feet. In this case, the request processes 224, 228 request data every 5 minutes within an ellipse that was 200 miles wide and 400 miles long, with the vast majority of length pointed toward the current heading of that aircraft. Of course, other parameters for the data request may be used in accordance with the present invention. Thus, the game 112 processes subsets of the larger databases 200, 204 that describe the data (weather or aircraft) that the game aircraft may encounter but filtered to relate the aircraft's position or route. One advantage of the process described above is that the data that required to be passed back to the game is minimized, thus saving time and bandwidth.

In the case of the aircraft data, a subset may consist of a list of aircraft data (for example, identification, flight path, heading, altitude, and speed) that were determined to be within the region that the request process determined. In the case of weather, the data returned would consist of weather reports (temperature, wind direction, dew point (freezing level), precipitation, etc.) within the region that the weather request process algorithms determined. The traffic and weather request processes 224, 228 may differ due to their specific needs (for example, traffic might only be needed within 50 miles for the traffic process 224 to work well, whereas the weather process 228 may need data within 100 miles to work well.)

Referring to Figure 3, a game system 107 is shown implemented using an object oriented design. The specialized objects preferably store their own data, request data from other objects when needed, and provide data to other objects when requested. Figure 3 illustrates a connection between the traffic manager 300, weather manager 304, aircraft manager 308, world manager 312, rendering engine 316, and player control 328. Since all of these processes request and process requests for data from each other, they can be visualized as being on a bus architecture. The other game processes, the audio engine 320, and force feedback engine 324, are less closely coupled, taking input from the aircraft manager 308 and traffic manager 300 only.

The traffic manager 300 maintains an internal database of the real-time traffic in the world that the user experiences. It also simulates the movement of the traffic and its control by air traffic controllers. Based on player aircraft type, state and position received from the player control/game state process 328, the traffic manager 300 requests traffic data from the game data server 108 that will be within a predefined geographic area around the player's aircraft. The data is maintained in an internal database located in the player's computer that is continuously updated by these requests and by the simulation of the traffic by the traffic manager's internal processes.

The traffic manager 300 processes requests for air traffic data position (location, heading, altitude, speed, and identification) from the render engine 316 on a continual basis. The render engine 316 may be any conventional rendering process. This data is used to display the aircraft within the world on the game display. The traffic manager 300 also simulates the communications of air traffic within the game world, based on real time data. The external realization of this is through communications commands that are sent to the audio engine 336. The audio engine 336 uses these commands to output realistic audio communications messages

to the user (for example, voice communications from air traffic control (ATC) and aircraft) based on the state of the player and other aircraft in the game. These messages include messages between ATC and the other aircraft and between ATC and the player 120. The voices are simulated using conventional technology as is known in the art.

The weather manager 304 maintains an internal database of the real-time weather in the world that the user experiences. Based on player aircraft state data (location, heading, altitude, speed, and identification) received from the player control/game state process 228, the weather manager 304 requests weather data that will be near the player's aircraft from the game data server 108. That data is maintained in an internal database that is continuously updated by these requests (based on the rate of change of the weather data and aircraft movement) and by the simulation of the weather by the weather manager rule-based processes. The weather manager 304 processes requests for weather state (temperature winds, precipitation, visibility) over a general area from the render engine 316 on a continual basis. This data is used to display real time changing weather within the world on the game display.

The weather manager 304 also simulates realistic, changing weather based on the real time weather feed and rule-based processes within weather manager 304 itself to provide weather effects not provided for from available real world data, for example, wind gusts, thunderstorms, cloud drift rate, the intensity of any precipitation, and the like. Rule based processes that react to weather changes and trends close in distance and time, such as wind gusts, are referred to as micro level processes. Macro processes describe trends that may occur due to daily or calendar trends. Examples of macro processes would include those triggered by time of day (thunder storms tend to occur during the late afternoon) or time of the year (heavy stratus and not thunderstorm clouds tend to develop in the United States during the winter). The

changing weather manifests itself to the user through the rendering engine 316. Thus, the use of micro and macro processes extends the real world data information to create new game conditions that are random, because their existence depends on weather in the real world. In addition, these processes create weather effects that will interact with the player's gameplay, as a thunderstorm or wind gust can severely affect the pilot's ability to fly a plane, in a flight simulation game, or drive a car, in a car racing game.

Micro Processes

As the weather data is updated only once per hour, one embodiment of the present invention uses micro process rules to project movement and changes to world data between these updates once an hour. These rulesets look at how the data typically changes over time and use simple rules developed to project where the real world data obtained might change over the following hour, prior to receiving the next update.

Wind micro process

Wind is reported and represented as a direction, speed and gusts (as in wind from 360, at 10 kts, with gusts to 20kts. This data is typically updated every hour by each reporting station. The weather manager 304 maintains weather observation data of each reported location for the current and previous hour. The wind micro processes compare the wind direction, speed, and gust values of these observations and actively changes the wind conditions that the player experiences over the next hour.

Transitioning game weather to the current weather observation

When the new weather observation comes in, the weather manager 304 slowly changes the wind that the player would see at that location over the next 10 minutes. Although winds can change quickly, changing the values over a reasonably short period makes the environment more

believable. In one embodiment, the game 116 changes the values of direction, speed, and gusts by about 10 percent each minute. For example, assume the current game weather for a location is: direction: 180, speed 5, and gusts to 10; and the new observation for that same location at time t is direction 150, speed, 10 and gusts to 15. In one embodiment, the wind micro processes calculates the differences between the current and new observation as

direction: -30 (150 – 180), speed +5 (10 –5) and gusts +5 (15 – 10).

The wind micro processes then projects over the next ten minutes that the winds at this location would change by this amount to match the values observed. The weather manager 304 implements 1/10 of this projected change (preferably rounded to the nearest integer) to each value each minute over the next 10 minutes. The values the player would see would change approximately as follows:

:00 Direction 180, Speed 5, Gust 10
:01: Direction 177, Speed 5, Gusts 10
:02 Direction 174, Speed 6, Gusts 11
:03 Direction 171, Speed 6, Gusts 11
:04 Direction 168, Speed 7, Gusts 12
.....
:10 Direction 150, Speed 10, Gusts 15

Projecting the weather trend

After the 10 minute period where the game weather gradually changes to match that of the actual weather, the game micro processes then work to vary the weather during the rest of the hour in the direction of the perceived trend. The process compares to the current weather observation, the previously reported observation, calculates a random variation within that range of change, and then varies the weather over the balance of the hour to match that change.

For example, the reported observation for a location at time t-1 may be: direction: 200, speed 5, and gusts to 10; and the reported observation for that same location at time t may be

direction 150, speed, 10 and gusts to 15. Then, the wind micro processes calculates the differences between the two observations as direction: -50 (150 – 200), speed +5 (10 –5) and gusts +5 (15 – 10). The wind micro processes then selects a random value within that range of change (for example, -20, 3, 2) and adds this value to the current weather values resulting in the following projected weather: direction: 120, speed 13, gusts 17. The weather manager 300 preferably implements 20% of this projected change (rounded to the nearest integer) each 10 minutes until the next hourly update. The changes might look as follows:

:10 Direction 150, Speed 10, Gusts 15
:20: Direction 146, Speed 10, Gusts 15
:30 Direction 142, Speed 11, Gusts 16
:40 Direction 138, Speed 11, Gusts 16
:50 Direction 134, Speed 12, Gusts 17
:60 Direction 130, Speed 13, Gusts 17

Thus, the present invention uses the wind micro process to actively create weather effects based on real world conditions to control the gaming experience.

Macro Processes

As described earlier, micro processes work to fill in or project data within time when there is no data. They work within the time between the data updates. Macro processes on the other hand, work over a larger scale to add information to the weather data where there is information, but less detail than might be desirable for the simulation.

Frequency of Thunderstorms Macroprocess

In one embodiment, a macro process is used to estimate the thunderstorm frequency within an area. Weather observations for a particular area only list that thunderstorms may exist and their relative severity (reported as severe or not). In this embodiment, the present invention augments this data by using a simple rule that there are typically more thunderstorm clouds or

cells within a give area that has been projected to have thunderstorms in the summer months, than in the fall or winter. For example, given that when generating the cloud formations for areas that have been reported to have thunderstorms, the program averages a base level of 3 thunderstorm clouds per generated sector during most of the year. During the summer months (between June and August) in the northern hemisphere, additional warming can cause more frequent and additional thunderstorms.

In assigning thunderstorms to the sector, the macro process determines the month that the player is currently playing the game 112 within. This information is easily made available from any number of sources, including the player's computer or from an external database. Between the months of June and August, the process adds to that base average value a random number that could result in as many as twice as many thunderstorms per given area. In this way, the game 112 causes the weather change in response to the current weather data and also to the time of year, providing the player a differing experience that is based on realistic processes.

Referring again to Figure 3, the aircraft manager 308 processes and maintains information about the player's aircraft and internal systems. The aircraft manager 308 takes control input from the player control process 328 (movement of the aircraft, and control of systems). The aircraft manager 308 then interfaces with the weather manager 304 for real time weather information that may affect the operation of the aircraft (for example, precipitation, wind, temperature), traffic manager 300 to provide display of real time traffic within the cockpit displays, and the world manager 312 for navigational and physical information about the world. The aircraft manager 308 provides aircraft specific information about the aircraft to the rendering engine 316. The aircraft manager 308 also outputs directives to the audio engine 320 and force

feedback engine 324 as needed to simulate the sound and real time feel of an aircraft as is known in the art.

The world manager 312 maintains information about the physical state of the world as is known in the art. This includes information about the terrain, ground cover, hydrology, buildings, roads, and navigation features. The world manager 312 takes real-time input from the weather manager 304 that might affect the look of terrain (temperature, precipitation). The world manager 312 provides navigation and terrain information to the traffic manager 300 and aircraft manager 304. The world manager 312 also provides information to the rendering engine 316.

The rendering engine 316 displays to the user the internal (cockpit) and external (terrain, weather, other aircraft) state of the game via the display 332 using conventional technology. The audio engine 320 is a conventional engine that generates realistic aircraft, communications, and world sounds to the user. Based on input from the aircraft manager 308 and traffic manager 300, it outputs event based sounds and spoken communications to the user via the audio system 336. Event based sounds (such as engine noise, cockpit switch movement, or aircraft crashes) are generated by the audio engine 320 in reaction to events sent to it by the aircraft manager 308. These sounds are either output once in the case of a switch movement, or continuously until changed (in the case of an engine drone).

The audio system also produces spoken audio messages generated by the traffic manager 300 from the air traffic controller (ATC) and other aircraft that can be heard by the user. The allowable communication phrases in the game (for example: "Turn left to heading 350", or "Roger, left to 350") are developed in advance for all positions (ATC, pseudo aircraft, and player). These are placed in a database that can be looked up by command. Voice actors may be

used to record all the words and numbers that would be contained in all the phrases done by that position (for example, the word: turn, left, right, to, from, heading, 0, 1,2,3,4,5, etc. would be recorded by a person representing an ATC controller.). These are placed in a database that can be looked up by position and word.

When a communication command is received by the audio engine 320, it contains all the information needed to specify and fill in the phase: the position (ATC, player, pseudo aircraft), the command (for example, TURN_TO_HEADING), and any specific data needed to fill out that phrase (Direction: left, Heading: 350). The audio engine 320 then looks up the phrase for that position, looks up the voice data for the word data recorded for that position, and then outputs that voice data out to the audio speaker using a conventional process called "stitching" in the order indicated by the phrase.

The force feedback engine 324 provides force feedback information to the user via haptic hardware contained in the joystick, as is known in the art. The feedback information is generated in response to input from the aircraft manager 308. Examples of this feedback include control pressures, wheel spin vibrations, touchdown bumps, or vibrations due to aircraft engine problems.

The player control/game state process 328 controls the startup and execution of the game through selection and control information from the user and provides the other engines the current information regarding the gaming unit controlled by the player. It also maintains data about the state of the game and the world.

The joystick 340 allows the user to also provide input to the game via the player control/game state process 328. These inputs can include control information (yaw, pitch, and roll), throttle position, and function selection (for example, lowering gear or flap control). The

user also receives appropriate force feedback information from the game via the force feedback engine 324. The keyboard 344 is a standard computer keyboard. The player uses coded keypresses and macros to control the configuration of the game and the aircraft.

Thus, the present invention provides a system for integrating real world data into a game to influence the player's experience of a game in a realistic and random (to the extent the real world is random) manner, that allows the real world weather and traffic data to affect and influence the outcome of the game.

The following description describes two important elements in a flight simulator embodiment of the present invention: The Weather Engine and the Traffic Manager.

I. The Weather Engine

Figure 4 represents the relationship between a weather engine 400 and the weather manager 304. The weather engine 400 receives weather input data (observations) from the weather manager 304 and performs current weather state calculations on the weather manager 304 requests for data. As discussed above, the weather manager 304 maintains weather data that are used to describe the weather around the aircraft or any other viewing position needed in the game. In a preferred embodiment, the weather engine 400 is an independent module that performs interpolation for weather simulation. As actual real world data is not available for every location and time a player may be playing the game 112, the game 112 interpolates data based on real world data to control the game play.

The input for the weather engine 400 is preferably a set of weather observations with specified location and time values. The output of the weather engine 400 is an interpolated weather status for any specified location and time value. The weather engine 400 then changes the weather rendered in a game smoothly as the player's aircraft moves from one location to

another, or while the player's aircraft moves in time, to provide a realistic real time weather experience for the player 120. Additionally, the weather engine 400 produces both "point" weather status for most weather elements and "environmental" status for some clouds types. The weather engine 400 is coupled to a weather element database 404 that stores attribute information regarding various weather elements to be used for generating interpolated weather.

Weather Elements

The following is an exemplary list of weather elements maintained in the weather element database 404:

- **Clouds.** Clouds are separated by clouds types and clouds layers. Clouds are specified with following attributes: clouds type, layer base altitude, layer top altitude, density (coverage).
- **Winds.** Winds are separated by layers, also winds change between layers linearly. Winds are specified with following attributes: height, direction and speed. Also wind gusts and squalls may be specified.
- **Visibility.** Visibility is specified with the number of miles that can be seen horizontally. Limited visibility is implemented by displaying haze and fog.
- **Temperature.** Temperature is specified in degrees Celsius.
- **Pressure.** Pressure specified in millibars.
- **Precipitation.** Precipitation supported is rain, snow and hail, which may coexist. A precipitation is specified with density.
- **Thunderstorm.** Thunderstorm is specified with power.
- **Icing.** The possible existence of icing is predicted according to other weather elements and is discussed below.

- **Turbulence.** The possible existence of turbulence is predicted according to other weather elements.

Weather Elements Dependencies

An observation point is a set of weather data that was collected and reported in the weather data base prepared by the weather data provider 104. Some of the weather elements are defined for an observation point while other elements are synthesized according to other weather elements. For example, "Icing" and "Turbulence" are not specified for an observation point, and therefore are calculated according to the presence and degree of other weather elements. Figure 5 is a block diagram illustrating different elements, their outputs, and the other elements (if any) upon which they depend. Different weather elements require different outputs. For example, precipitation will require rendering and audio and is synthesized based on cloud and temperature information. Icing is synthesized based on temperature and visibility elements. However, clouds themselves are rendered, and are based on actual data for a given observation point, or are interpolated when the current location of a gaming unit is in between observation locations.

A. Weather Element Interpolation

The main goal of the weather interpolation is to change weather smoothly while moving from one location to another or moving in time based on an interpolation of weather elements for those observation locations near in time and location to the current location and time of the player's gaming unit. The following algorithm is one embodiment of implementing weather interpolation both by time and location using observation points that have actual weather data.

Step 1 -- Filter Weather Observations by Location

The first step for the weather interpolation is to choose relevant observation locations to be used for interpolation. Figure 6 illustrates the process of recurrently selecting blocks to select

observation locations. All the observations are stored as blocks, where a block is specified by latitude and longitude degree value. The task is to choose observation locations that are close to the current location of the player's aircraft in space and in time. First, the weather engine 400 processes the block encompassing the aircraft's location to determine if the block contains an observation location. Then, if the block does not contain an observation point, the weather engine 400 processes the blocks around the block containing the aircraft's location, as shown in Figure 6. However, if there are no observation points within the aircraft's location the radius value of the search is incremented to include the closest block to the current location. The weather engine 400 then processes the next closest blocks in the next pass as shown in Figure 6. The weather engine 400 stops on any pass when any observation points are found in the pass. These observation points are used for interpolation.

Step 2. Interpolate by Time

The second step for the weather interpolation is to choose relevant observations according to current time value. For each of the observation locations picked in the first step, the observation location that is closest to the current time value (closest minimal and maximal time values) is chosen.

The following cases may occur:

- All of the observation locations found in Step 1 are located in the past. In this situation, the weather engine 400 will select the observation location latest in time for the interpolation.
- All of the observations are in the future. In this example, the oldest observation is selected.
- Both observations in the future and in the past exist. In this case, the weather engine 400 selects the latest observation in the past and the closest observation in the future, and an interpolated observation based on the weather elements for each observation location is

produced. In one embodiment, the interpolated observation location is selected linearly according to the current time value and selected observations time values.

Step 3 -- Interpolate by Location

Next, an interpolated observation for the camera location is produced according to all of the observation locations selected and interpolated in the previous steps. Figure 7 illustrates one embodiment of this interpolation mechanism where *Di* is square of the distance between camera location and observation *#i* location; the unnormalized weight for an observation *#i* is calculated as *1/Di*; the sum weight for normalization is calculated as sum of not normalized weights: {*S* = *1/D1 + 1/D2 + 1/D3 ...*}, and the normalized weight for an observation #i is calculated as not normalized weight divided by sum weight: *1/(Di*S)*.

A special case exists when one or more very close (*Di* is about *0*) observations exist. In this case, the closest observation by time is selected as the observation point for the camera location.

Step 4 -- Calculate Undefined Elements

Some weather elements may not be present for the selected observation locations. In this case, the values have to be calculated after interpolation of present weather elements according to specific weather element rules. For example, pressure is calculated via a standard pressure lapse rate table. Icing is calculated accounting for consideration, precipitation, cloud type, and temperature at that location.

Step 5 -- Interpolate by Height

Some weather elements are defined as layers (for example, winds) but are calculated for a specific camera height location (wind, then temperature, icing and turbulence). For these elements, the weather element value has to be interpolated between closest layers values linearly.

Additionally, wind calculation at specific height preferably accounts for gusts that may vary wind speed and direction randomly.

B. Cloud Interpolation

Cloud coverage, important in a flight simulation game, are also preferably interpolated in accordance with the present invention. In one embodiment, cloud objects are implemented with a grid that moves when the player's aircraft moves. Cloud parameters are calculated according to real world observation data that is interpolated for each vertex of the grid. Figure 8 illustrates cloud objects grid for about 2/8 cloud coverage. Each cell in the grid contains one cloud object. The location of a cloud object is random for each cell, however, bigger clouds (greater cloud coverage) will have less location "freedom." Thus, for full coverage, each cloud object is placed exactly in a grid vertex. The size of a cloud object is calculated according to the current cloud coverage value as obtained from observation data. In one embodiment, a random value is added (both positive and negative sizes) to provide more realistic simulation. Each cloud object has corresponding seed value to generate the form of the cloud. The seed value dictates the amount a cloud will billow up per cycle. The seed value is preferably changed smoothly with time to emulate a realistic change of a cloud form. Additionally, each cloud object has a yaw random value that is not changed with time. The yaw value specifies the angle from vertical that the cloud will billow toward.

Cloud Grid Generation

A cloud grid is generated for each new aircraft and for each cloud layer. For each grid vertex (a cloud object), the following values are generated at grid creation:

- A Size Variation coefficient, *SizeVariation* in range from a minimum cloud size variation to a maximum size variation

- Offset variation coefficient (*OffsetVariation*), range from 0 to 1.
- Initial seed value (*SeedValue*) , range from 0.0 to 255.0

Cloud Grid Handling For Moving Aircraft

The cloud grid is always aligned to a Cloud Grid Step value *(cloud grid step)*. Therefore, when an aircraft moves to a new location, it is possible, (when the movement is large enough), that some grid rows and/or columns are deleted and new grid rows and/or columns are added. In this case memory blocks (for still existing but moved vertexes) are moved and new vertex (cloud objects) values (as defined in previous section) are generated.

Vertex Values Calculations

The values for each vertex are also recalculated for each weather cycle in one embodiment. In this embodiment, the following values are recalculated:

Size

The cloud size is calculated according to cloud coverage (*Cloud Density*, a value in range from 0 to 1) at the location as *CloudDensity * cloud grid step * SizeVariation.* The *cloud size* calculated should be within the range from *0* to *cloud grid step*. This value should be adjusted in overflow cases.

Cloud Height

Cloud height is the current height of the cloud layer at the location.

Offset

The *cloud offset* from the grid vertex is calculated as *(cloud grid step - CloudSize) * OffsetVariation.*

Seed Value

The seed value is calculated according to the elapsed time value (*time_t*, in seconds) as *CurrentSeed* + *TimeElapsed* * *CLOUD_SEED_CHANGE*. (*CLOUD_SEED_CHANGE* is a subject for fine-tuning, *0.2* value may be assumed for the first time).

C. The Icing Module

In a preferred embodiment, the present invention uses real-time data to simulate icing conditions on the aircraft. The icing conditions are weather effects that are not provided by real time data but are generated and caused by real time data. The icing conditions is also a weather effect that has a direct effect on the player's gaming experience, as severe icy conditions can cause a plane to malfunction and even crash. In one embodiment, the icing module is a module contained within the weather engine 400. The icing module takes as input information about the weather that determines the existence and severity of icing. The icing module also accepts configuration information about the aircraft, including the status of de-icing systems on the aircraft. The icing module calculates and maintains the existence and level of icing on the aircraft based on the weather, aircraft configuration, position of aircraft and actions of the pilot. It can be made to reflect differences in the type and severity of icing, and the affects on differing aircraft configurations. The output of the icing module is a percent of maximum (0.0 to 1.0) level of icing for several critical sections of the aircraft (windshield, side windows, wings, fuselage, and engine intakes) that would be adversely affected by icing. This data is used by other modules within the aircraft system model (dynamics, engine) to effect realistic changes in the performance and handling of the aircraft. This data is also used by the renderer to display to the user realistic effects of icing (icing on airframe, reduced visibility of iced-over windows).

1. Icing High Level Processes

In one embodiment, at the highest level, the icing module is represented by a loop consisting of:

1. Input current state of world and aircraft
2. Calculate Icing Accumulation
3. Calculate Icing Deaccumulation (melting)
4. Output updated state of icing of aircraft.
5. Repeat

Because this model takes as input some variables that are affected by the user, the module is preferably scheduled to run at a rate that would show changes in icing to the user. However, as ice does not freeze or melt at any fast rate, executing the module as infrequently as once every 5 seconds is typically sufficient.

2. Icing General Variables

From game state control 328:

- Aircraft Position (latitude and longitude)
- Aircraft Altitude (in feet)
- Windshield_deice_on (Boolean that indicates whether the windshield deicing system is currently operating.)
- Wing_deice_on. (Boolean that indicates whether the wing deicing system is currently operating.
- These variables will FALSE in aircraft that do not contain these systems. In one embodiment, the system can be made to fail so that the system could be off (FALSE) even though the user has moved the control to on (TRUE).

From weather manager 304:

- Ground_temperature at current position
- OAT (outside air temperature) This is calculated using a standard lapse rate calculation of 3 degrees per 1000 feet from the ground_temperature (given a 70 degree F ground_temperature, this would result in an OAT estimate of 40 degrees F at 10,000 feet.
- Cloud_coverage (Cloud coverage at that position and altitude). This would be a number between 0 (no clouds) and 1 (full coverage) at that altitude within the weather rectangle that contains that position.

3. Icing Specific Variables:

The following variables are maintained by the icing module and are made available to other models. The variables can have a value between 0.0 (no ice) and 1.0 (fully iced).

Windshield_icing (level of icing of the visible windshield on the front of the aircraft)

- Side_windows_icing (level of icing on the side windows)
- Wing_icing (level of icing on the wings)
- Fuselage_icing (level of icing on the fuselage)
- Engine_intake_icing (level of icing of the engine intakes)

Each of these variables could be made to vary in their level to the others based on the platform and configuration of the aircraft.

4. Icing Processing:

An approximation of the effects of icing on aircraft in the real world is used by the present invention to calculate icing effects.

i) Input State Data

The game state and weather data accessed and read into the icing module

ii) Calculate Icing Accumulation

Icing on the aircraft is accumulated on the various surface variables when the aircraft is within visible moisture (clouds) and the outside air temperature (OAT) is within a temperature where the moisture will freeze on contact (between 10 and 30 degrees Fahrenheit). Warmer temperatures will melt the ice. With colder temperatures, the moisture will not melt on contact.

For example, a higher rate of accumulation may be assigned to cloud coverage (clouds that would be flown through) that is higher than seventy percent coverage. This rate could also be shown to differ due to the aircraft configuration and ice type or altitude.

In one embodiment, the following algorithm is used to calculate icing accumulation.

If ((cloud_coverage >= .1) and (cloud_coverage < .7)) and
((OAT >= 10) and (OAT =< 30)) then

- Windshield_icing = Windshield_icing + .01
- Side_windows_icing = Side_windows_icing + .005
- Wing_icing = Wing_icing + .01
- Fuselage_icing = Fuselage_icing + .01
- Engine_intake_icing = Engine_intake_icing + .01

If (cloud_coverage >= .7) and
((OAT >= 10) and (OAT =< 30)) then

- Windshield_icing = Windshield_icing + .02
- Side_windows_icing = Side_windows_icing + .01
- Wing_icing = Wing_icing + .02
- Fuselage_icing = Fuselage_icing + .02

Engine_intake_icing = Engine_intake_icing + .02

iii) Calculate Icing Deaccumulation (Melting due to atmosphere or deice equipment)

Melting due to atmospheric effects

There are two ways to combat icing in the real world. One is flying to an area or altitude where the temperature is warm enough to melt the ice. This would be at an OAT greater than 32

degrees F. However, although a pilot can fly out of icing conditions such that the ice will no longer accumulate, it is possible that the OAT is not sufficiently warm to melt the ice.

One algorithm to calculate deicing based on flying to a warmer area is:

If (cloud_coverage < .1) and (OAT >= 32) then

- Windshield_icing = Windshield_icing - .01
- Side_windows_icing = Side_windows_icing - .005
- Wing_icing = Wing_icing - .01
- Fuselage_icing = Fuselage_icing - .01
- Engine_intake_icing = Engine_intake_icing - .01

iv) Deicing due to the use of aircraft equipment

The other way to combat icing is to use deicing equipment. Icing equipment is often unreliable and difficult to apply consistently and effectively. If a pilot does not apply the equipment frequently and regularly, the ice can accumulate such that the icing equipment becomes incapable of melting the ice. For example, the deicing value may be set at one half of the full amount if the equipment is not used with sufficient deicing equipment.

Although deicing equipment may operate effectively, deicing equipment does not cover all of the aircraft. Finally, it is also possible with higher levels of accumulation frequency during icy conditions, the deicing equipment may not be capable of maintaining a de-iced condition.

One algorithm to calculate deicing using deicing equipment is:

```
If Windshield_deice_on then
        Windshield_icing = Windshield_icing - .01
If Wing_deice_on then
        Wing_icing = Wing_icing - .01
```

The first algorithm emulates ice melting (deicing) due to temperature being warm enough and the aircraft not being in clouds. The second algorithm emulates removing ice due to having either the windshield deicing on or the wing deicing on.

Thus, for each loop (cycle in the game), the deicing equipment will attempt to remove ice from the wings and windshield. However, as discussed above, the de-icing condition of the present invention realistically provides for the possibility that the icing may overpower the equipment and/or aircraft if the pilot does not fly away from the icy conditions.

v) Output state of icing variables

The input state of the icing variables is modified by the accumulation and deaccumulation models. The resultant values are made available to other modules and external modules. Some examples of how these variables are used are illustrated below.

a) Aircraft System Model

The aircraft system model is a conventional model that simulates the flight of an aircraft.

1. Calculation of lift

Ice accumulations on the wing reduce the lift produced by the wing by up to one half. Thus, the icing model would adjust the aircraft system model as follows: Lift = Lift * (Wing_icing * .5).

2. Calculation of Drag

Ice accumulation on the wing and fuselage add up to 25 percent more drag on the performance of the aircraft. Thus, in this example, the aircraft system model is adjusted as: Drag = Drag * (1 + (.125 * (wing_icing + fuselage_icing)))

3. Calculation of Weight

Ice accumulation on the wings and fuselage can add significant weight to the aircraft (for this illustration, up to 25% more). The adjustment may be:

Aircraft_weight = Aircraft_weight * (1 + (.125 *(wing_icing + fuselage_icing)))

4. Calculation of Engine performance

Engine intake icing can affect engine performance by restricting the amount of air that is available. The adjustment may be:

Thrust = Thrust – (Thrust * Engine_intake_icing)

5. Calculation of effects on aircraft equipment due to fuselage icing.

Icing can also affect the operation of aircraft gear, flaps, and instruments (icing over):

If (fuselage_icing > .6) then flaps_operational = FALSE);

If (fuselage_icing > .7) then gear_operational = FALSE);

If (fuselage_icing > .8) then static_instruments_operational = FALSE);

b) Icing Render Model

These are examples of render decisions that would be effected by the results of the icing model. The renderer displays the view of the outside world and the aircraft to the user. This can include the windshield, side windows and the aircraft in general.

1. View out of windshield

The view out of the windshield is generally clear. In icing conditions, the amount of visibility is reduced by the amount of windshield_icing. Here, an array of iced windshield displays would be pre-rendered and displayed based on the windshield_icing value. For example:

Windshield_panel = windshield_icing_panels [windshield_icing]

2. View out of side windows

The view out of the side windows would be obscured by the percent of window_icing indicated by that variable. Here, an array of iced window displays would be pre-rendered and displayed based on the window_icing value.

Window_panel = window_icing_panels [window_icing]

c) Aircraft Rendering

The aircraft itself as shown by the user would be modified depending on the values associated with the windshield_icing, window_icing, wing_icing, and fuselage_icing. Pre-rendered representations of icing effects at various realistic accumulation levels would be added to the aircraft model depending on the values of those variables. The user would see this accumulation when looking out the windows, as well as when an external view was chosen. This would involve a number of arrays and calculations as shown above.

Thus, the icing module uses real world data as a basis for generating icing effects that interact with the control of a player's aircraft and the rendering of the aircraft in accordance with the present invention to provide a realistic and exciting gaming experience that is based upon the current and real weather conditions.

II. The Traffic Manager

Referring to Figure 9, the traffic manager 300 is composed of several independent processes that implement its functionality. A traffic engine 900 creates and manages and updates all of the aircrafts in the player's game 112. The traffic engine 900 first requests player aircraft data (position, altitude, heading, speed, and configuration) from the player control/game state engine 328. It uses this data to formulate requests to the game data server 108 for aircraft traffic around the player aircraft as discussed above. The traffic engine 900 uses the data returned to create the aircraft in the game 112. The traffic engine 900 manages this data through the use of a traffic database 904, which contains a list of all the active aircraft near the player aircraft, including the player aircraft. The information stored for each database entry typically includes the aircraft identification number (for example, TWA714) and a unique pointer to the pseudo and

player aircraft objects created by the traffic engine 900 later in the process. The traffic database 904 is also used by other processes in the game 112 to identify and access the traffic objects.

In addition to adding aircraft entries to the traffic database 904, the traffic engine 900 also manages the database 904 by removing entries from the database 904 that are now out of the player's area. In one embodiment, the traffic engine 900 removes entries from the traffic database 904 for any aircraft for which it has not received a report for 3 update cycles. As discussed above, updates to the database 904 are received for aircrafts within a certain geographic area of the player's aircraft. Thus, a lack of reports indicates the aircraft that has landed or its position is now far enough away from the player's aircraft that the aircraft is no longer relevant to the player's game 116.

The traffic engine 900 first generates a single ATC process 908. The ATC process 908 controls and communicates with pseudo aircraft processes 916 and the player aircraft process 932 such that the ATC process 908 appears to the player as a real ATC controller. The ATC process 908 also includes a link to the traffic database 904 so that it has access to the data (for example, current speed, heading, and altitude) of all of the relevant aircraft.

The traffic engine 900 then creates a unique pseudo aircraft process 916 for each non-player aircraft as information regarding a new non-player aircraft is entered into the traffic database 904. The traffic engine 900 also subsequently sends data report updates on the real aircraft to the pseudo aircraft process 916 as they become available. This data is used by the pseudo aircraft process 916 to match its position and configuration to that of the real aircraft. Each pseudo aircraft process 916 appears and behaves to the player 120 as real aircraft in the world. The pseudo aircraft data includes data such as position in the world (x, y position, heading, speed and altitude), aircraft identification, and state (gear position, flap position, etc).

This data is maintained internally and can also be issued in response to data requests from other game processes such as the ATC process 908 and the rendering engine 316. A communication/control link with ATC process 908 is also established during creation. This link allows the pseudo aircraft process 916 to make and respond to communications commands with the ATC process 908. Audio communications generated by the pseudo aircraft process 916 are accomplished by sending communications commands to the audio engine 320.

The traffic engine 900 also creates a player aircraft process 912 which is linked externally to the player control/game state engine 328 and internally to the ATC process 908. This allows the ATC process 908 to monitor and interact with the player. Information about the player aircraft and intentions are accessible to the ATC process 908 through a process of mirroring data from the player control/game state engine 328 into the player aircraft process 912. The ATC process 908 also preferably interacts with the player through spoken communications commands sent to the audio engine 320 (that the player ultimately hears). The player responds and interacts with the controller through interfaces (control, menu and keyboard commands) provided by the player control/game state engine 328.

A. Pseudo Aircraft Process Algorithms

Each pseudo aircraft process 916 preferably comprises a simulation executive, a common database, and individual modules that handle communications, navigation and control, and rule-based decisionmaking. The simulation executive schedules the order of execution of the individual modules. In one embodiment, after initialization, the order of process execution is:

1. Communication module (input new data and commands)
2. Rule Based Logic Module (react to the updated data)
3. Navigation and Control Module (move aircraft to follow current aircraft goals)

4. Communications Module (output data and commands)

5. Repeat.

1. Common Database

The common database is used to store the reports received from the traffic engine 900, the current state of the aircraft as controlled by the logic and navigation modules, and data about the internal goals and state of the pseudo aircraft itself (for example, the current goal of the navigation and control system is). The common database is also used to provide common communication architecture for the process modules. Changes to the database are input and acted on by the modules.

2. Communication Module

The communication module manages all communication between the pseudo aircaft process 916 and other external processes. It uses the common database as an area to store new data and messages, or send data to the external processes. Data input by the communications module include update reports from the traffic engine 900, commands from the ATC process 908, and data requests by other processes. The communications module outputs data to external processes based on messages directed by the other modules. Data output by the communications module include messages to the ATC process 908 and audio engine 328, data requests to other modules, and position updates to the rendering engine 316 and other processes.

3. Rule-Based Logic Module

The rule-based logic module controls the actions of the psuedo aircraft in reaction to the changing state of the pseudo aircraft and the changing world around it. As the real time data received by the pseudo aircraft varies due to the day, position, and weather, it is the interaction with these rules that provides the influence of real time data on the player's experience within the

game. Examples of these preprogrammed and flexible action rules include those that react to the changing position of the aircraft, ATC communications and control, and the position of the player aircraft (as controlled by the player). For example, one rule reacting to the position of the aircraft would include the need to announce to the air traffic controller process 908 that the pseudo aircraft is within 50 miles of the airport and ready to land. This would also result in an audio message that would be heard by the player.

An example of the rule that describes this would be:

If (x,y) position is less than 30 miles from the destination airport, then
send (announce_near_airport, aircraft_id, position, altitude) to atc_controller and
send (announce_near_airport, aircraft_id, position, altitude) to audio engine.

The pseudo aircraft also reacts to commands by the air traffic controller. When the aircraft gets close enough to the airport, the ATC controller tells the (pseudo) aircraft to stop following the path of the real aircraft and follows its instruction to land. The pseudo aircraft reacts by changing its navigation module to follow ATC commands, follows that command, acknowledges to the ATC process 908 that it received the command, and announced via audio that same command.

One algorithm to achieve this behavior:

If under_atc_control = FALSE and
received (turn_left_heading, 350) from atc_controller
then
Set under_atc_control to TRUE,
Stop_normal_navigation,
Send (turn_left_heading, 350) to navigation_and_control_process,
Send (acknowledge_turn, left,heading) to atc_controller,
Send (acknowledge_turn, left,heading) to audio_engine.

The pseudo aircraft 916 can also react to the movements of the player aircraft. The pseudo aircraft 916 could include rule-based logic to monitor and try to move away from any

aircraft that gets to close to it (for example, if the player aircraft tried to bump into it). If this occurred, the pseudo aircraft rule-based logic could instruct the aircraft to break off normal navigation, fly away from the position of the player aircraft, and send a complaining message to the ATC. One example of an algorithm to achieve this behavior is:

```
If player_aircraft (position) < 1 mile then
        stop_normal_navigation,
        move_away_from (player_aircraft(position)),
                Send (complain_aircraft_near) to atc_controller,
                Send (complain_aircraft_near) to audio_engine.
```

Once the player aircraft moved away from the pseudo aircraft the following rule would allow the pseudo aircraft to go back to operating normally:

```
        If player_aircraft (position) >= 1 mile then
        resume_normal_navigation,
                Send (aircraft_has_left) to atc_controller,
                Send (aircraft_has_left) to audio_engine.
```

B. Navigation and Control

The navigation and control module contains the functionality of a conventional autopilot. It is capable of navigating and controlling the pseudo aircraft by following report updates stored in the common database using conventional technology. However, in accordance with the present invention, the navigation and control module is also capable of following navigation commands issued to it by either its internal rule-based logic module or the external ATC process 908.

The navigation and control module uses the report updates to match the aircraft position altitude, speed, and heading of the pseudo aircraft to that of the real aircraft. As the report data (position, heading speed and altitude) is updated by the traffic engine 900 approximately every 3 minutes (as discussed above), in one embodiment, the navigation and control module interpolates

and generates its own data that approximates the movement of the aircraft between these updates and is discussed in more detail below. These interpolations of the present position, heading, altitude, and speed of the pseudo aircraft are sent to the rendering engine and made available to other processes within the game.

In addition to matching the movements of the real aircraft, the navigation and control module is also capable of responding to specific navigation commands. These commands can be issued to it by the internal rule based logic module or by the ATC process 908. Examples of these commands include:

- Turn (Left or Right) to a specified heading.
- (Descend Climb, or Maintain) an altitude.
- (Increase or Decrease) speed to some value.
- Fly directly to a point (x, y position and altitude).
- Hold at a point (circle over a specific x, y position and altitude).
- Perform an approach and land at an airport.
- Move away from (some x, y) position.
- (stop, resume) normal navigation (following the report updates).
- Cleared for approach
- Cleared to land

These commands are in a simple format that are easily transferred between processes (for example, the command to "Turn left to heading 350" could be represented as (turn_left_to_heading, 350).

C. Navigating and averaging between aircraft observations.

In order for the pseudo aircraft to operate in a believeable fashion, it must smoothly transition between position reports. These reports include a reported heading, speed, altitude and X, Y (Latitude, Longitude) position. The pseudo aircraft is initially created at the position (X, Y Latitude and Longitude) of the most recent report for the real aircraft it represents. It is also created at the altitude of the real aircraft, and with the heading and airspeed of that aircraft. The pseudo aircraft is then "driven" forward from that position using that heading, speed and altitude. The changes in position of the pseudo aircraft are calculated internally by the aircraft agent and output to the rendering engine 328. This continues until the next report (latitude, longitude, altitude, heading and speed) from the real aircraft is received. Then, the pseudo aircraft process 916 compares the difference between the position of the pseudo aircraft in the game 116 and that of the real aircraft. The process 916 first calculates a vector including changes in heading, speed, and altitude that would move it more directly toward that position (so that the aircraft can "catch up" to the actual position of the aircraft). It then adds that vector (using vector addition) to the heading, altitude, and speed vector of the report from the real aircraft (The vector of the real aircraft reflects where the aircraft is intending to go). The pseudo aircraft process 916 then commands the pseudo aircraft to alter course, heading and speed to match that vector. The resultant vector therefore reflects both the changes needed to better reflect the real aircraft position as well as moving along the actual course.

Finally, in accordance with the present invention, the autopilot is programmed to make smooth turns and transitions in heading and altitude. Rather than commanding a quick movement to the change in vector, the autopilot manages the change over approximately 30

seconds. After the pseudo aircraft stabilizes at that vector, it continues to move along it until the next update where the process is repeated.

D. Avoiding the Player aircraft

Another issue in integrating real-time data with a game is controlling the interaction between player aircraft and pseudo aircraft. In one embodiment, rules are used to cause the pseudo aircraft to ground a player-controlled circuit. First, the pseudo aircraft flies away if the player aircraft is within range of the pseudo aircraft and moving toward it quickly. Second, the pseudo aircraft stops moving away from the player aircraft if the player approaches at a slower rate. Third, the pseudo aircraft allows the player aircraft to fly close to it (within a maximum range) as long as the player first approaches at a sufficient slow speed. Fourth, the pseudo circuit attempts to fly away from the player aircraft if the player aircraft moves with the maximum sole range. These rules allow the player aircraft to interact with the real time pseudo traffic without causing collisions. The player can then develop the skill of flying in formation by moving slowly close to a pseudo aircraft. The player can also try to intentionally fly into the pseudo aircraft or force the pseudo aircraft to fly off its course or into a ground object or mountain. In one embodiment, if the pseudo aircraft or the pseudo and player aircraft collides with another object, then the object colliding explodes and the game session ends.

The following is one embodiment of algorithms to achieve the above rules:

Calculating Distance $_{ps\ to\ p}$, Rate of Closure $_{ps\ to\ p}$, and Avoidance Vector $_{ps\ to\ p}$

The distance from the pseudo aircraft to the player aircraft is calculated as the cube root of the sum of squares of the X, Y, and Altitude differences. For each, X, and Y are converted from latitude, longitude to feet from the 0 latitude and 0 longitude point. Altitude (A) is in feet.

$Distance_{ps\ to\ p} = ((X_{ps} - X_p)^2 + (Y_{ps} - Y_p)^2 + (Z_{ps} - Z_p)^2)^{1/3}$, where P represents values from the Player aircraft and PS represents the pseudo aircraft.

The Rate of Closure $_{ps\ to\ p}$ is calculated by comparing the values of $Distance_{ps\ to\ p}$ from one calculation cycle to another. If the $Distance_{ps\ to\ p}$ is calculated each second, the Rate of Closure per second for that time period would be the previous value of $Distance_{ps\ to\ p\ (t-1)}$ - the current value of $Distance_{ps\ to\ p\ (t)}$. For example, f the previous value was 1400feet and the current was 1200 feet, the rate of closure would be 200ft/sec.

The Avoidance Vector

The pseudo aircraft monitors the $Distance_{ps\ to\ p}$ and Rate of Closure $_{ps\ to\ p}$ of the player aircraft to its centerpoint each game cycle (at approximately once per second)

If $Distance_{ps\ to\ p} < 1500$ ft, AND the Rate of Closure $_{ps\ to\ p} >= 25$ ft/minute, then the pseudo aircraft will:

a. Turn off the autopilot (e.g. stop following position reports)
b. Send out a randomized audio message each minute to the effect that "There is an aircraft that is too close to me".
c. Move away from the player aircraft in the direction of the Avoidance Vector $_{ps\ to\ p}$ at a speed that is twice that of the rate of closure, up to 1500 ft/minute or the maximum that the aircraft performance will allow.

This algorithm addresses the situation where a player begins to move the aircraft toward the pseudo aircraft at a high rate of speed. If the Rate of Closure $_{ps\ to\ p} < 25$ ft/minute AND the Distance $_{ps\ to\ p} > 50$ Ft. AND the pseudo aircraft autopilot is off, then the pseudo aircraft process 916 will:

a. Turn on the autopilot.
b. Resume normal navigation.
c. Move the pseudo aircraft back to its original course.
d. In one embodiment, output an audio message to the effect of "Someone is flying in formation with us."

This algorithm addresses situations when the player has moved toward the aircraft quickly, but has slowed down to a reasonable closure rate.

If the Distance $_{ps\ to\ p}$ is < 50 ft then the pseudo aircraft process 916 will:

a. Turn off the autopilot (stop following the position reports)
b. Send out a randomized audio message each 15 seconds to the effect that "You are getting too close".
c. Move away from the player aircraft in the direction of the Avoidance Vector $_{ps\ to\ p}$ at a speed that is twice that of the rate of closure, up to 1500 ft/minute or the maximum that the aircraft performance will allow.

This algorithm addresses the situation where player has been flying close to the pseudo aircraft (in formation) and then begins to move toward the pseudo aircraft intentionally or by mistake.

If the pseudo aircraft collides with the ground or any object other than the player aircraft then

a. The pseudo aircraft outputs a random audio message.
b. The pseudo aircraft explodes.
c. The game session ends.

If the Distance $_{ps\ to\ p}$ <10 feet (indicating that the aircraft have probably collided) then:

a. The pseudo aircraft outputs a random audio message to the effect that "that was dumb".
b. The pseudo aircraft explodes.
c. The player aircraft also explodes.
d. The game session ends.

D. ATC Process Algorithms

The ATC Process 908 has an architecture that is similar to the pseudo aircraft process 916. The ATC Process includes a simulation executive, a common database, and individual modules that handle communications, and rule-based decision making.

The simulation executive schedules the order of execution of the individual modules. After initialization, the order of process execution is:

1. Communication module (input new data and commands)
2. Rule Based Logic Module (react to the updated internal and external data)
3. Communications Module (output data and commands)
4. Repeat.

1. ATC Common Database

The ATC common database is used to store data about the aircraft under control and data about the internal goals and state of the ATC process 908 itself (for example, the plans for how aircraft are to be controlled). The common database is also used to provide common communication architecture for the other process modules. Changes to the database are input and acted on by the modules.

2. ATC communication module

The basic operation of the communications module is similar to that used in the pseudo aircraft module. The ATC communication module manages all communication between the ATC process 908 and other external processes. It uses the common database as an area to store new data and messages, or send data to the external processes. Data input by the ATC communications module include position reports from the pseudo aircraft and player aircraft processes 912, 916, requests and responses from the psuedo aircraft and player aircaft processes 912, 916. The ATC communications module outputs data to external processes based on messages directed by the other modules. Data output by the ATC communications module include messages to the pseudo aircraft process 916, player aircraft process 112, and audio engine 328 and data requests to player and pseudo aircraft processes 912, 916.

3. ATC Rule-Based Logic Module

The ATC rule-based logic module controls the actions of the ATC process 908 in reaction to the changing state of the aircraft in the world. The ATC process 908 reacts to the change in aircraft flow due to the day, position, and weather in which the game is played, and thus provides a realistic real-time response to real time traffic and weather conditions. The rulesets within this module work to simulate the active control of major airports that within the airspace areas that the player may select to fly in. The ATC process 908 interacts with aircraft that are either flying into or taking off from that airport.

E. Inbound aircraft control.

Aircraft destined to land at that airport (psuedo and player) must contact the ATC process 908 and state this intention within 50 miles of that airport. The pseudo aircraft do this automatically, however, the player must manually make this communication. When aircraft make this communication, they are added into a landing queue. The ATC process 908 uses this queue to manage communications with the aircraft and also to adjust the real time traffic as needed to sequence the player aircraft in for landing. The landing queue consists of a FIFO (first in, first out) queue of the aircraft that have contacted the ATC process for landing. Data included in this queue for each aircraft include the sequence order, an identifying ID for that aircraft, and the altitude it is initially assigned to when it is brought under control and sent to an initial approach point, and the point it is currently cleared to.

This queue is used by the rule-based logic to simulate the control of aircraft approaching and landing at the airport. This includes passing through several points in the approach where control and audio commands are issued to the aircraft. The points, their location from the airport runway, and the commands issued are as follows:

Point	Location	Command Issued
Aircraft first contacts ATC	> 50 miles from airport	Proceed to Initial Approach Point (IAP)
Aircraft reaches IAP	20 miles from airport	Cleared to Final Approach Fix (FAF)
Aircraft reaches FAF	10 miles from airport	Cleared for approach
Aircraft reaches approach point	2 miles from airport	Cleared to land

As the aircraft come under control, they follow the commands of the ATC rules that sense where they are and forward them on to the next point. An example of a rule that would use the data in the queue would look as follows:

```
If (position_from_airport AC1) <= position (cleared_to_point) then
    Send_message (cleared_to_FAF) to AC1,
    Set cleared_to_point to FAF.
```

When the aircraft lands, the aircraft is removed from the in bound aircraft queue. The aircraft would be removed by the traffic engine 900 as soon as the data from it disappeared.

1. Holding aircraft at the Initial Approach Point (IAP) point.

When traffic is busy bound for the airport, there may be times when there are more aircraft wanting to approach and land than can be accommodated. (Only one aircraft is cleared to the Final Approach Fix (FAF) at a time, this sequences the aircraft smoothly in). This can also occur when the player aircraft chooses to approach the airport. The ATC process 908 accommodates this by requiring the aircraft to hold at the IAP before being cleared the Final Approach Fix.

When the aircraft is initially cleared to the IAP, it is given an altitude to maintain. This altitude is generated using the sequence order in the queue to determine an offset from a base altitude. The altitude separation prevents the aircrafts from colliding. For example, if an aircraft

approaching is the 4th aircraft in the queue and the base altitude of the IAP is 10,000 feet, that aircraft would be cleared to 10000 + (4 * 1000) = 14,000 feet to the IAP. When the sequence number exceeds 10, and the total number of aircraft in the queue is less than ten (indicating that some aircraft have been cleared) then the aircraft sequence number is reset to 1. This would result in the next aircraft to be cleared to 11,000 ft. (10,000 + (1* 1000ft). Aircraft exit holding as the FAF becomes clear and the next aircraft holding at the IAP is cleared to the FAF.

2. Outbound aircraft.

Aircraft departing the airport (taking off) are added to a takeoff queue when they send a communication message to the ATC process 908 indicating that they are ready to takeoff. This message is automatically issued by the pseudo aircraft when they are generated by the traffic engine 900 to be on that airport and their flight plan indicates that they have not yet taken off. This message can also be generated by the player aircraft when they are on the airport and want to take off. The takeoff queue maintains as data a sequence number, an ID number for the aircraft, and the point that they are cleared to takeoff to. There are 3 points that the takeoff logic is concerned with.

Point	**Location**	**Command Issued**
Aircraft first contacts ATC	On runway	Cleared to takeoff
Runway Sequence point	1 miles from airport	Resume navigation
Clear of airport	50 miles from airport	Send message "Have a good day"

When an aircraft requests to takeoff, the aircraft is added to the takeoff queue. If it is safe for the ATC process 908 to allow it to takeoff (any previous aircraft have reached or moved beyond the runway sequence point), the aircraft is cleared to takeoff. Otherwise, the aircraft is asked to hold, and is allowed to takeoff when that point becomes clear. Once it reaches the

sequence point, the ATC process 908 issues a command to the aircraft to resume normal navigation but does not remove the aircraft from the queue. The aircraft can then start to follow its report data and match the real aircraft route. When the aircraft moves beyond the 50 mile point, it is removed from the queue and is sent and auditory sendoff message.

3. Holding Aircraft on the ground.

As in the landing queue, there can be situations where there are more aircraft that want to takeoff than can be accommodated. This can often occur when the player aircraft asks to takeoff when the airport is busy. In this case, when the aircraft sends a ready to takeoff message, the aircraft is directed to hold. As aircraft ahead in the queue depart, that aircraft is eventually cleared to takeoff. There is no limit to the number of aircraft that can be holding the takeoff queue, which unfortunately mirrors real life.

4. Dealing with Rogue Aircraft.

The landing and takeoff queue also serve a secondary purpose of allowing the ATC process 908 to monitor its airspace for aircraft that are not under its control (particularly the player aircraft). If the ATC process 908 that the player aircraft is within 50 miles of the airport and has not contacted the ATC process 908 and has been placed on the landing queue determines then the ATC process 908 will react by sending audio messages to that unknown aircraft asking it to identify itself and asking what its intentions or similar messages up until the aircraft has landed. The same process is used for the player aircraft if they proceed onto the runway and takeoff without asking for clearance. The ATC process 908 senses that it is on the runway without requesting clearance to takeoff (is not on the takeoff queue) and will send an appropriate string of audio messages until the aircraft has gone beyond 50 miles.

5. Use of real time weather data.

The ATC process 908 also can use real-time weather data as discussed above for the airport that it is controlling to respond to pilot requests for weather data (for example, what is the wind direction). This data could also be used to allow the ATC process 908 to choose the most appropriate runway on which to command the aircraft to land, and generate realistic rendering of the CN part itself based on real-time conditions.

Thus, a system, method, and apparatus are described for enabling the use of real-time data to influence and drive a gaming experience. Although a preferred embodiment has been described in some detail, a flight simulation application, the teachings of the present invention can be extended to other games as well, such as car or ship racing games, or other sporting events. Numerous additional modifications may be made to the apparatus described above without departing from the true spirit of the invention. Moreover, although specific functionality has been ascribed to different steps of the above described method and modules of the above described circuitry, these functionalities can be performed in different orders and by different modules as would be known to one of ordinary skill in the art.

CLAIMS

1. A method for simulating a real world environment in a virtual world represented in a game, comprising the steps of:

receiving a location selection from a player at an identifiable time;

interpolating simulated environmental conditions in the virtual world for the selected location responsive to real world information associated with the selected location from a time other than the identified time; and

generating simulated environmental conditions in the game responsive to the interpolated simulated environmental conditions.

2. A method for simulating a real world environment in a virtual world represented in a game, comprising:

receiving a location selection from a player;

interpolating simulated environmental conditions in the virtual world for the selected location responsive to real world information associated with locations other than selected location; and

generating simulated environmental conditions in the game responsive to the interpolated simulated environmental conditions.

3. A method for simulating a real world environment in a virtual world represented in a game, comprising:

receiving a location selection from a player at an identifiable time;

interpolating simulated environmental conditions in the virtual world for the selected location responsive to real world information associated with locations other than the selected location and the identified time; and

generating simulated environmental conditions in the game responsive to the interpolated simulated environmental conditions.

4. A method for simulating a real world environment in a virtual world represented in a game, comprising:

receiving a location selection from a player;

determining whether real world environmental condition information is associated with the selected location; and

responsive to determining that real world environmental condition information is associated with the selected location, applying the real world environmental condition information to generate simulated environmental conditions in the virtual world for the selected location.

5. The method of claim 4 further comprising:

responsive to determining that real world environmental condition information does not exist for the selected location, analyzing locations surrounding the selected location to determine whether the surrounding locations are associated with real world environmental condition information.

6. A method for integrating real world data into a gaming experience comprising:

determining a geographic location of a gaming unit controlled by a player;

responsive to current that real world data being available for the location of the gaming unit, applying the current real world data to simulate a portion of the gaming experience; and

responsive to current real world data not being available for the location of the gaming unit, interpolating real world data based upon current real world data associated with locations near the location of the gaming unit to simulate a portion of the gaming experience.

7. The method of claim 6 wherein the gaming experience is a flight simulation game, the gaming unit is an aircraft simulation, and the real world data comprises weather data or traffic data.

8. The method of claim 6 wherein the gaming experience is a flight simulation game, the gaming unit is an aircraft simulation, and the real world data comprises weather data, the method further comprising:

generating simulated weather conditions for a location based on real world data regarding other weather conditions for the location.

9. The method of claim 8 wherein a simulated weather condition is icing, and the generation of icing conditions is determined based on real world data regarding temperature and precipitation.

10. The method of claim 6 wherein the gaming experience is a flight simulation game, the gaming unit is an aircraft simulation, and the real world data comprises surrounding aircraft data, the method further comprising:

generating representations of real world aircrafts to display as part of the gaming experience based on real world data identifying real world aircrafts.

11. The method of claim 10 further comprising:

selecting real world aircrafts to display that are located within a predetermined radius of the location of the aircraft simulation.

12. A method for simulating a real world environment in a virtual world represented in a game, comprising:

receiving a location selection from a player for a gaming unit controlled by the player at an identifiable time; and

generating representations of gaming units to display in the game responsive to real world units being associated with locations near the selected location of the player controlled gaming unit at the identified time.

13. The method of claim 12 wherein the game is a flight simulation game, the gaming unit is an aircraft, and generating representations comprises generating representations of aircrafts to display in the game responsive to real aircrafts being associated with locations near the selected location of the player controlled aircraft at the identified time.

14. The method of claim 13 wherein the representations of aircrafts corresponding to real aircrafts are displayed in locations corresponding to where the real aircrafts are located in the real world.

15. The method of claim 14 wherein the locations of the representations are updated to account for changing positions of the real aircraft.

16. The method of claim 14 further comprising:

determining whether there is new data regarding the location of a real aircraft for which a representation is being displayed in the game; and

responsive to determining that there is new data regarding the location of the aircraft, updating the location of the representation responsive to the new data.

17. The method of claim 16 further comprising:

responsive to determining that there is not new data regarding the location of the aircraft, interpolating a new location for the representation of the aircraft based on the latest data regarding the aircraft.

18. The method of claim 18 wherein the latest data includes speed and heading information of the real aircraft.

19. The method of claim 14 wherein a gaming unit corresponding to a real world unit moves away from the gaming unit controlled by the player responsive to the player's gaming unit moving within a predetermined radius of the gaming unit corresponding to a real world unit.

20. The method of claim 13 wherein an air traffic controller process maintains information regarding locations of player aircraft and the real world aircrafts, and alerts the player regarding movements of the real world aircrafts.

21. The method of claim 20 wherein the air traffic controller transmits audio messages to the player regarding the movement of real world aircraft to emulate radio traffic a real world pilot would experience.

22. A system for integrating real world data into a game, comprising:

a real world data manager for requesting a selected grouping of real world data from a comprehensive real world database to use in the game;

a real world data database, coupled to the real world data manager, for storing the selected grouping of data relating to real world events; and

a real world data interpolation engine, coupled to the real world data manager and the real world data database, for receiving a request for interpolated data from the real world data manager, retrieving selected data from the real world data database, and generating interpolated game data to be used in the execution of the game.

23. The apparatus of claim 22, wherein the game is a flight simulation game, and the real world data is weather data, and the interpolation engine is a weather engine for interpolating weather data to be displayed during execution of the game.

24. The apparatus of claim 22 wherein the weather engine simulates weather conditions based on an analysis of other real world weather conditions.

25. The apparatus of claim 24 wherein the weather engine simulates icing conditions based on an analysis of temperature and precipitation data.

26. The apparatus of claim 25 wherein the weather engine will alter icing conditions responsive to a player using a deicing application in the game.

27. The apparatus of claim 22, wherein the game is a flight simulation game, and the real world data is traffic data, and the interpolation engine interpolates traffic data to display other aircraft during execution of the game.

28. A computer readable medium for integrating real world data into a game, the computer readable medium storing instructions to cause a processor to:

display a gaming unit corresponding to player input at an identifiable time;

retrieve real world data regarding locations of real world units at a time near in time to the identified time the player input is received; and

display representations of real world units responsive to the retrieved data in the game.

29. The computer readable medium of claim 28 further comprising instructions to cause the processor to:

determine whether real world data exists for a real world unit that is currently being displayed for a selected moment in time;

responsive to determining that real world data does not exist for the selected moment in time, interpolate a location for the real world unit; and

display the real world unit at the interpolated location.

30. The computer readable medium of claim 29 wherein the game is a flight simulation game, the gaming units are aircrafts, and the instructions cause the processor to interpolate the position of a real world aircraft based on heading and speed information.

31. A computer readable medium for integrating real world data into a game, the computer readable medium storing instructions to cause a processor to:

display a gaming unit corresponding to player input at an identifiable time;

retrieve real world data regarding environmental conditions at a time near in time to the identified time the player input is received; and

display representations of environmental conditions in the game responsive to the retrieved data in the game.

32. The computer readable medium of claim 31 wherein the instruction causing the processor to retrieve real world data regarding environmental conditions further causes the processor to:

retrieve weather data for a location corresponding to a location of the gaming unit; and

display representations of environmental conditions based on the retrieved data.

33. The computer readable medium of claim 32 wherein the instructions cause the processor to:

determine whether weather data exists for the location of the gaming unit;

responsive to determining that weather data does not exist, analyze surrounding locations until real world weather data is found; and

interpolate the representations of environmental conditions for the location of the gaming unit in the game from the real world weather data found.

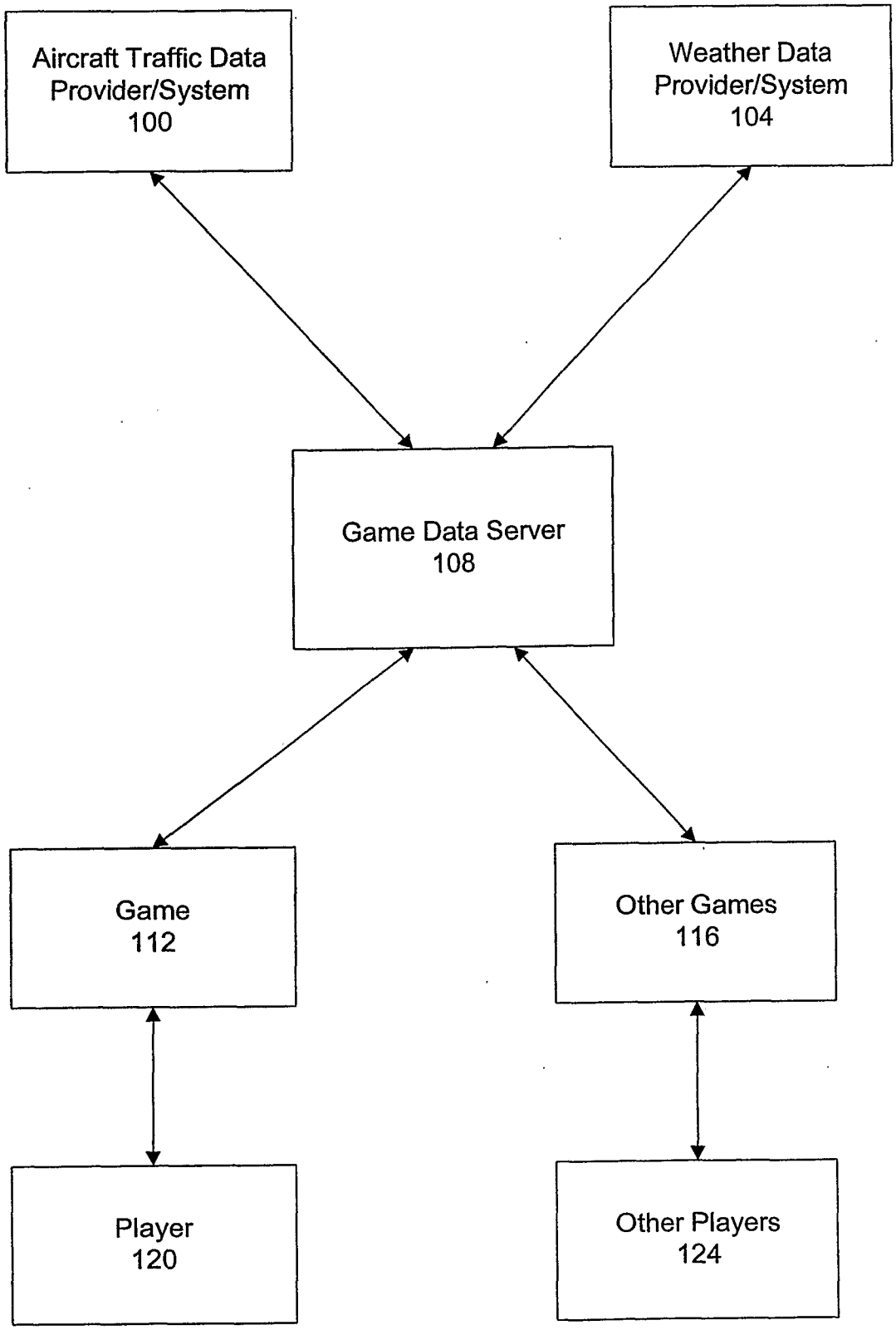

Aircraft Traffic Data Provider/System 100
Weather Data Provider/System 104
Game Data Server 108
Game 112
Other Games 116
Player 120
Other Players 124

Figure 1

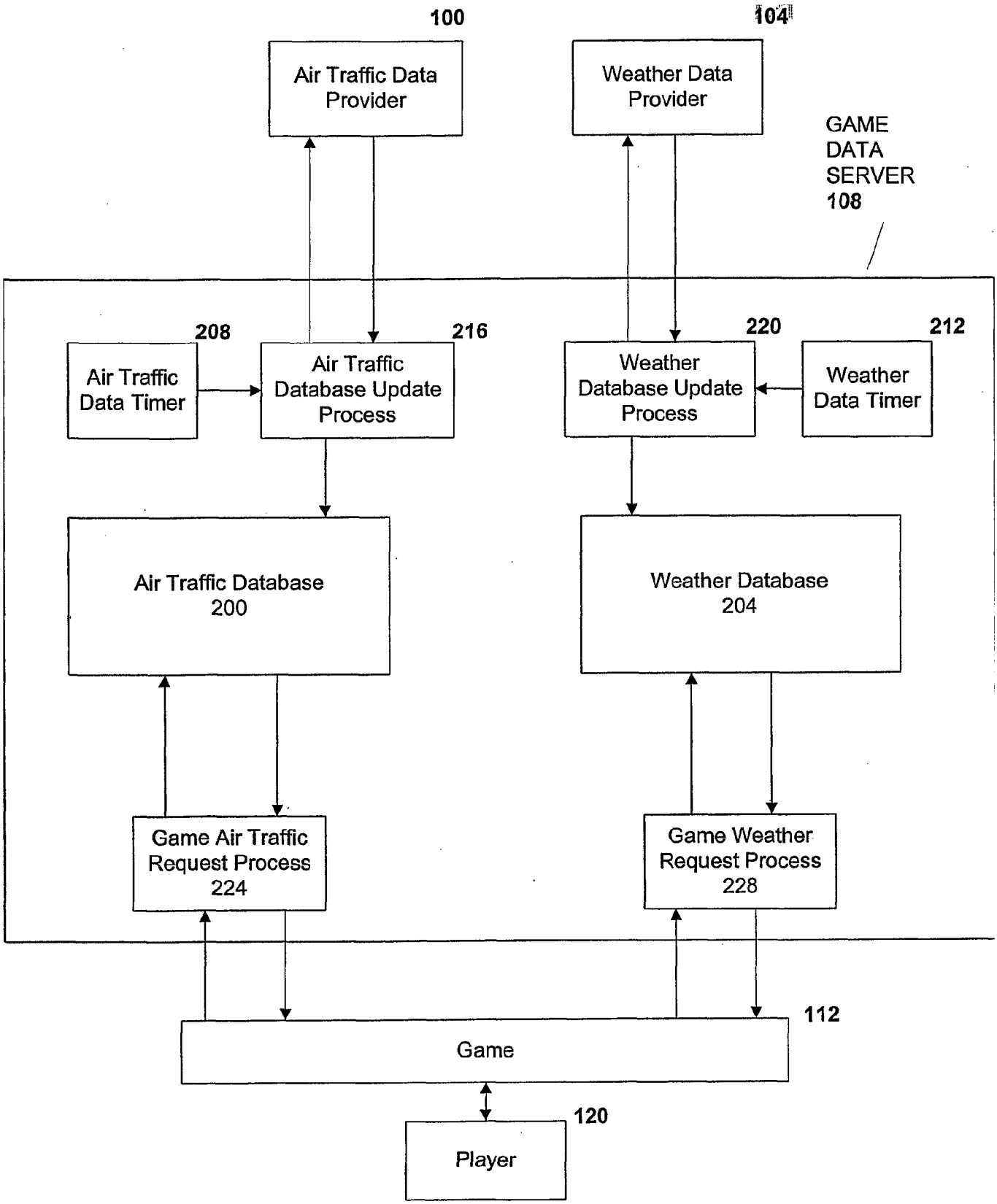

100
104
Air Traffic Data Provider
Weather Data Provider
GAME DATA SERVER 108
208
216
220
212
Air Traffic Data Timer
Air Traffic Database Update Process
Weather Database Update Process
Weather Data Timer
Air Traffic Database 200
Weather Database 204
Game Air Traffic Request Process 224
Game Weather Request Process 228
112
Game
120
Player

Figure 2

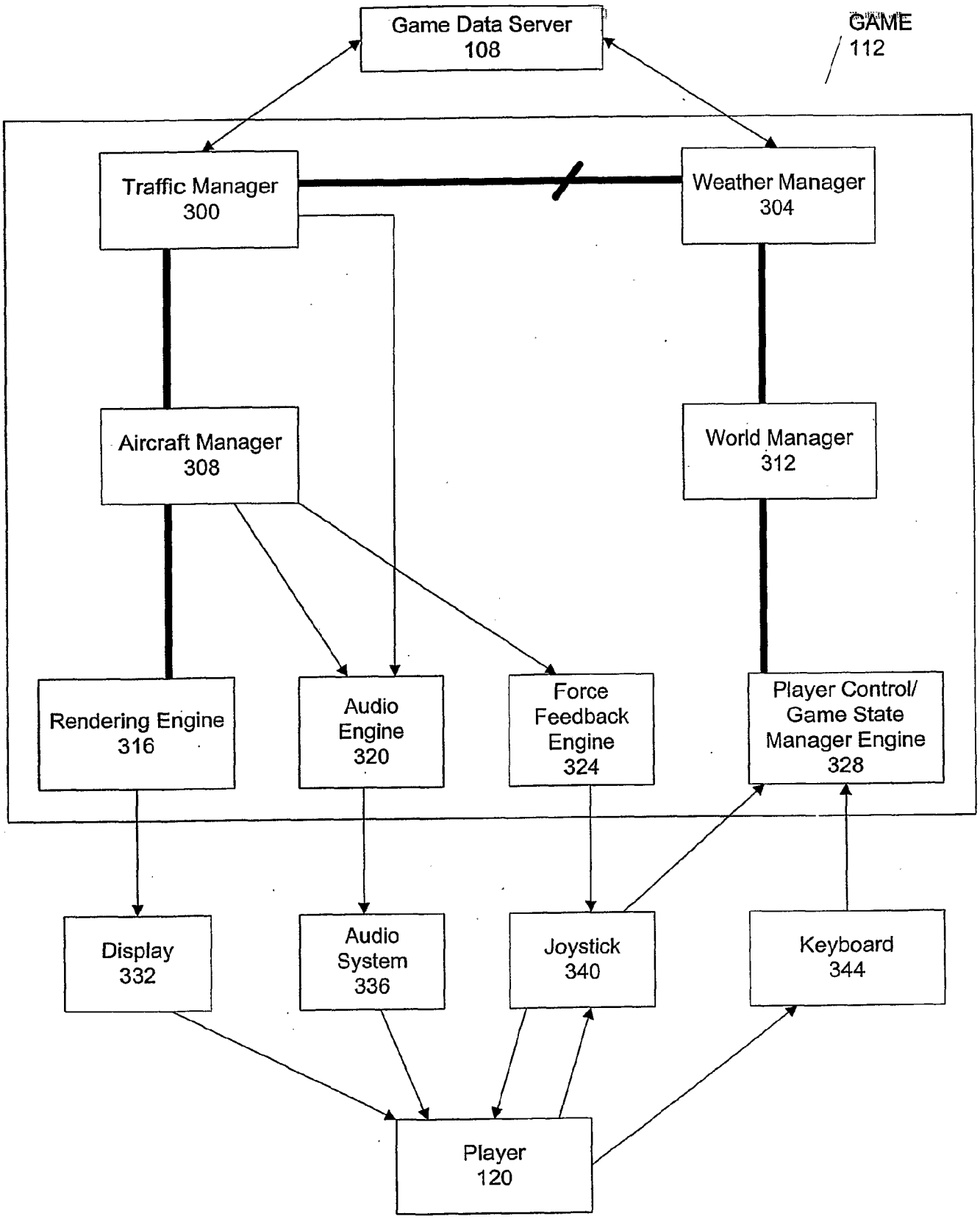

Game Data Server
108
GAME
112
Traffic Manager
300
Weather Manager
304
Aircraft Manager
308
World Manager
312
Rendering Engine
316
Audio
Engine
320
Force
Feedback
Engine
324
Player Control/
Game State
Manager Engine
328
Display
332
Audio
System
336
Joystick
340
Keyboard
344
Player
120

Figure 3

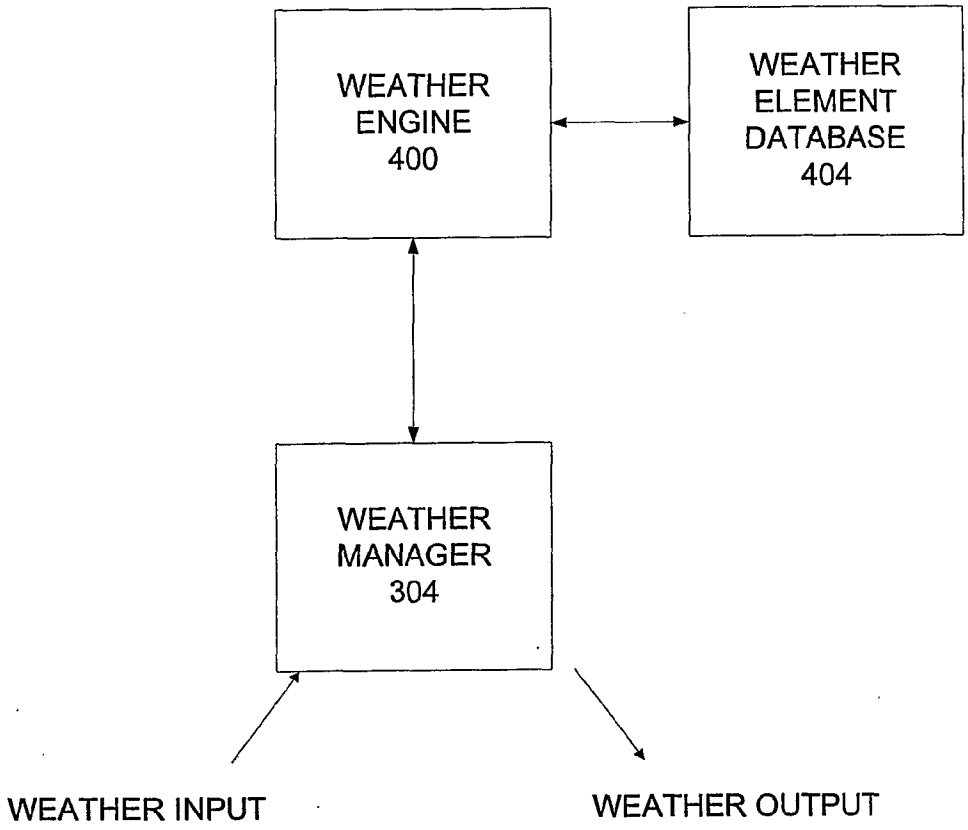
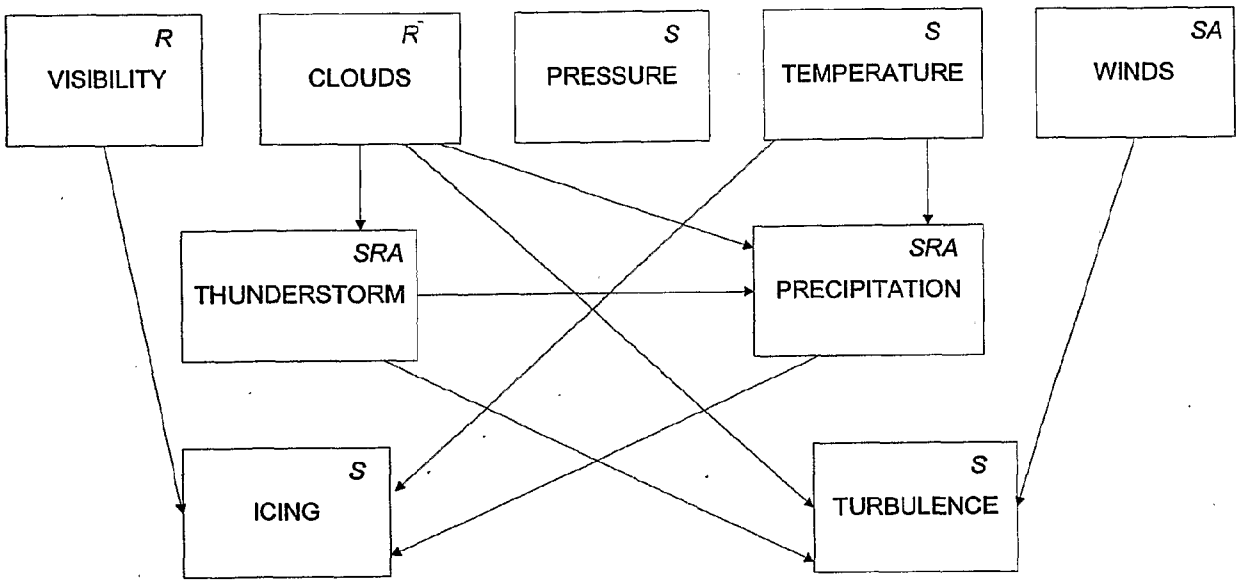

WEATHER ENGINE 400
WEATHER ELEMENT DATABASE 404
WEATHER MANAGER 304
WEATHER INPUT
WEATHER OUTPUT

Figure 4



R: RENDERING
S: SIMULATION
A: AUDIO

Figure 5

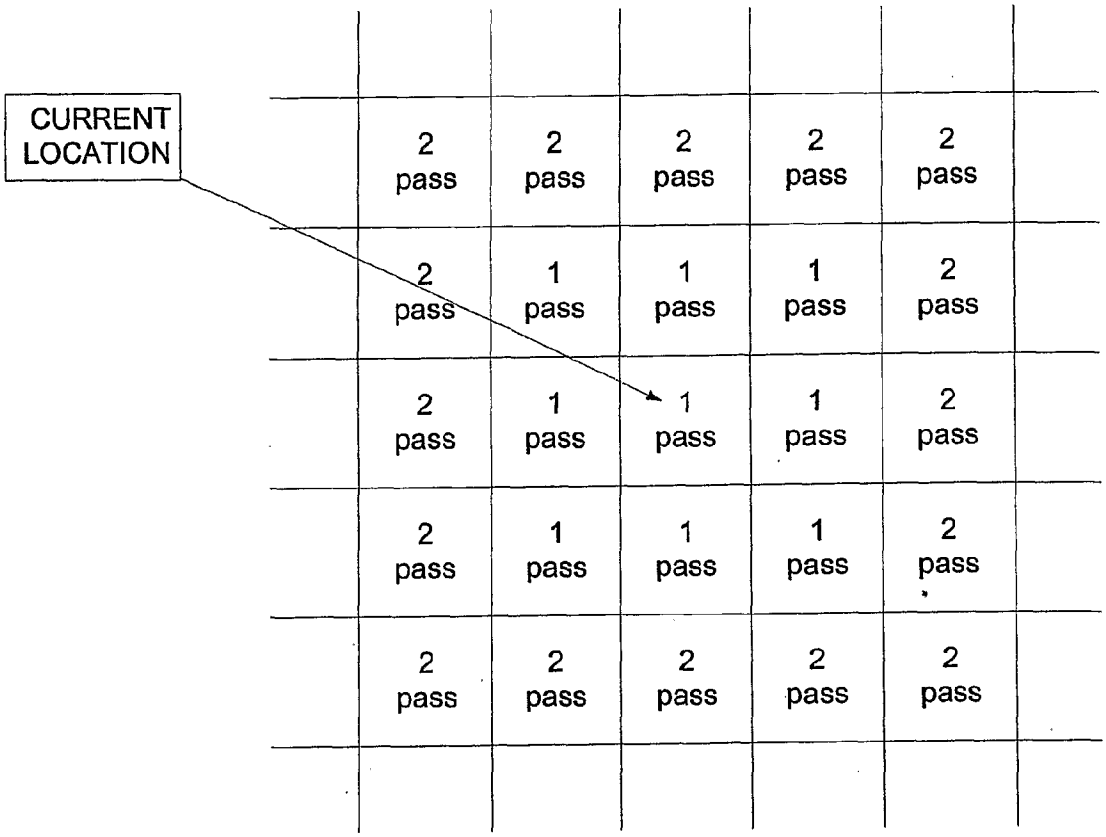

CURRENT LOCATION
2 pass
2 pass
2 pass
2 pass
2 pass
2 pass
1 pass
1 pass
1 pass
2 pass
2 pass
1 pass
1 pass
1 pass
2 pass
2 pass
1 pass
1 pass
1 pass
2 pass
2 pass
2 pass
2 pass
2 pass
2 pass

Figure 6

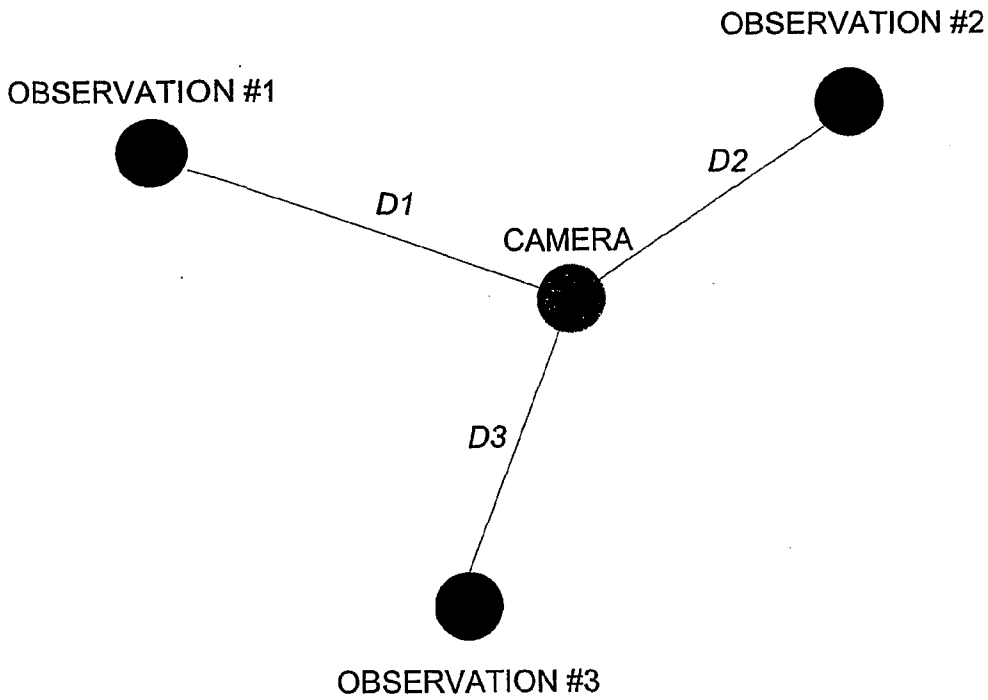


Figure 7

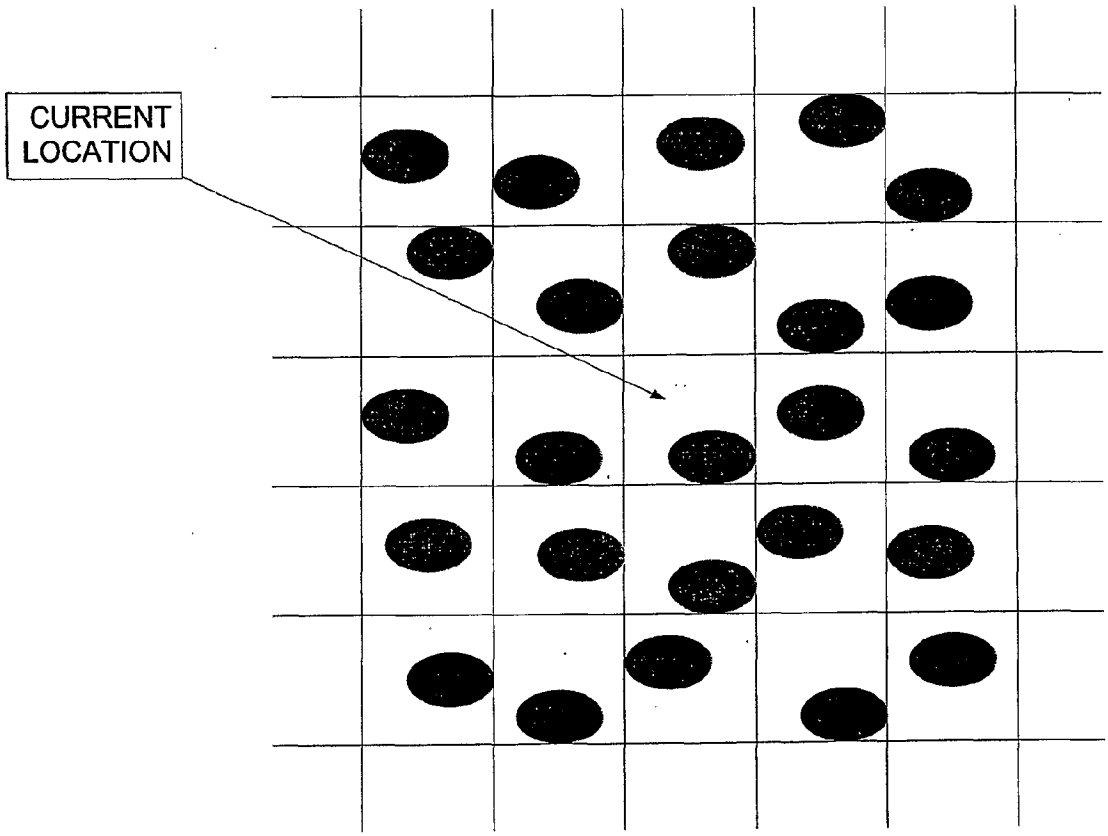


Figure 8

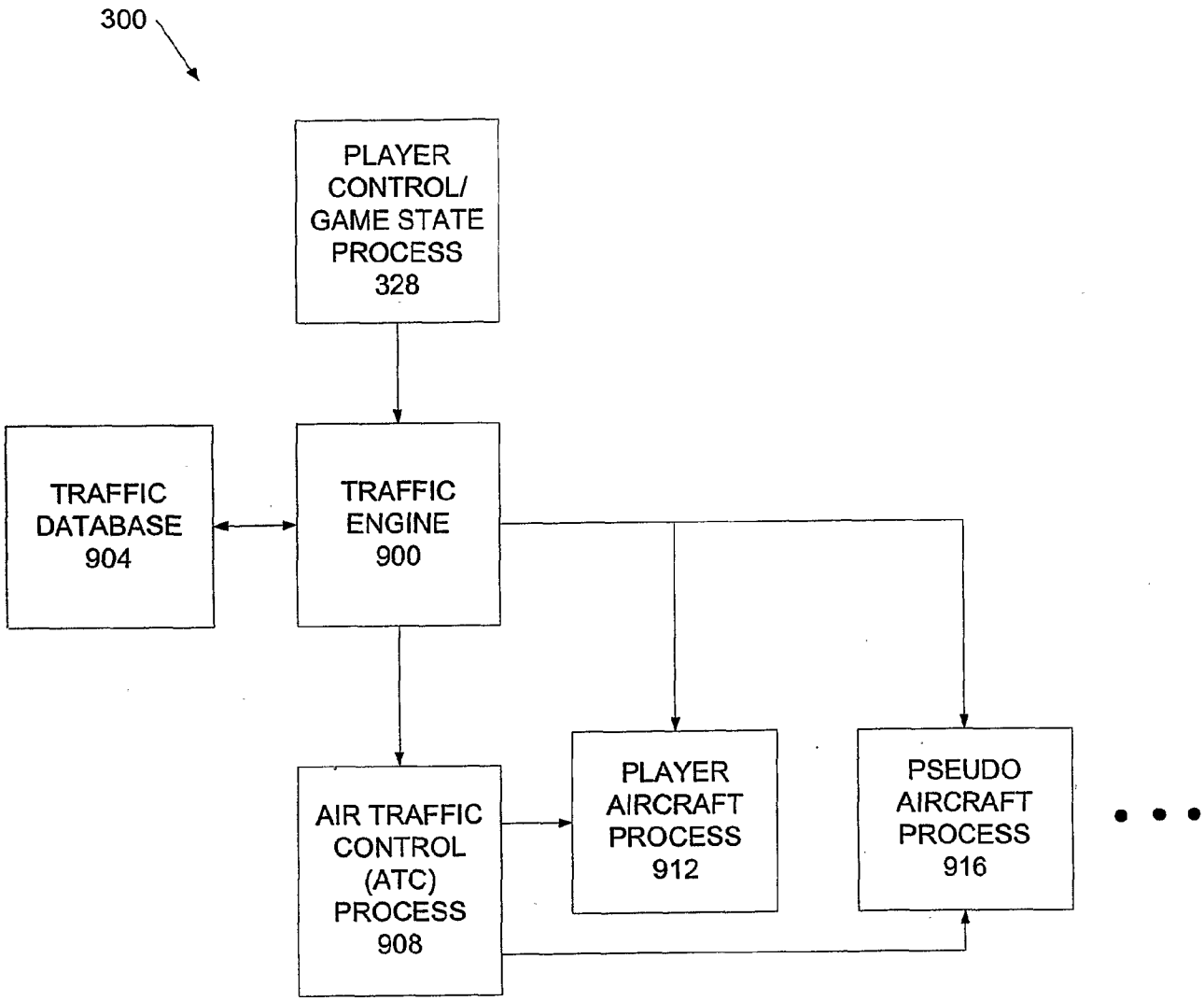

300
PLAYER CONTROL/ GAME STATE PROCESS 328
TRAFFIC DATABASE 904
TRAFFIC ENGINE 900
AIR TRAFFIC CONTROL (ATC) PROCESS 908
PLAYER AIRCRAFT PROCESS 912
PSEUDO AIRCRAFT PROCESS 916

Figure 9