Generating Solid Models From Topographical Data

Topographical data are converted into forms useable by rapid-prototyping machines.

Goddard Space Flight Center, Greenbelt, Maryland

A method of generating solid models of terrain involves the conversion of topographical data into a form useable by a rapid-prototyping (RP) machine. The method was developed to enable the use of the RP machine to make solid models of Martian terrain from Mars Orbiter laser-altimeter topographical data. The method is equally applicable to the generation of models of the terrains of other astronomical bodies, including other planets, asteroids, and Earth.

Topographical data describe a terrain in terms of a set of three-dimensional coordinates [e.g., Cartesian (x,y,z) or polar (latitude, longitude, radius) coordinates] of points or nodes on the terrain surface. The input data for the RP machines are required to provide a three-dimensional description, not of a single surface, but of a volume — in this case, a ground volume that underlies the terrain surface. The description is required to be in the form of triangular elements that connect the nodes of all the surfaces and that completely bound the volume, with no open areas, no overlap of triangles, and no extraneous geometric elements.

The software used in the present model-generation method was written in IDL — an advanced programming language that affords a number of tools, including subroutines that triangularize surfaces. The software creates a volume from the topographical surface data by adding sides to the edges of the terrain surface and joining the sides with a bottom surface. Each of the sides is triangularized by use of IDL subroutines, and then the software searches for extraneous elements and removes them.

Topographical data are usually presented in a grid corresponding to polar coordinates, so that a model generated from such data is equivalent to a topographical map in Mercator projection. However an RP machine is fully capable of including the curvature of a planetary body in a model that it makes. Therefore, the software also offers a capability to transform the topographical data to a projection onto a surface having a curvature corresponding to that of the surface of the modeled planet.

This work was done by John W. Keller of **Goddard Space Flight Center**. *Further information is contained in a TSP (see page 1).*
GSC-14897-1

Computationally Lightweight Air-Traffic-Control Simulation

This algorithm simulates ATC functions for a busy airport.

NASA's Jet Propulsion Laboratory, Pasadena, California

An algorithm for computationally lightweight simulation of automated air-traffic control (ATC) at a busy airport has been derived. The algorithm is expected to serve as the basis for development of software that would be incorporated into flight-simulator software, the ATC component of which is not yet capable of handling realistic airport loads. Software based on this algorithm could also be incorporated into other computer programs that simulate a variety of scenarios for purposes of training or amusement.

The ATC simulation problem that the algorithm is meant to solve can be summarized as follows: ATC is responsible for all aircraft that enter an arbitrarily specified hemisphere, denoted the flight-simulator bubble, that is centered at the airport. ATC must guide all the aircraft to safe landings in a sequence that is as fair as possible under the circumstances. Information about the airport that is taken into account includes the lengths, directions, and locations of end points of runways. Information about each aircraft that is taken into account includes the current three-dimensional position and velocity, maximum and minimum speeds, and mathematical relationships among turning times and the starting and ending points of turns between specified headings. The solution generated by the algorithm must be a set of instructions to the aircraft that enable all aircraft to land without violating any constraints.

The algorithm consists of four components denoted the controller, the plan, the vector generator, and the constraint verifier. The controller is event-driven and relatively simple: It responds to any of three events, as follows:

1. An aircraft enters the bubble. The controller tries all vector options in a shortest-path-first order, checking each by use of the constraint verifier. When a solution is found, the controller issues instructions to the appropriate pilots.
2. An aircraft leaves the bubble. The controller removes the aircraft from the plan.
3. An aircraft changes location as prescribed by the plan. The controller causes the plan to be updated with the new location and time of arrival of the aircraft at the location. If necessary, it gives instructions to the appropriate pilot.

Thus, the controller is the input/output interface for the ATC.

The plan is a data structure that is used to verify current and hypothetical routing for each aircraft. The plan consists of a simple temporal network (STN) augmented by labeling of time points with identities of aircraft and of other time points with which overlaps must be prevented. The approach followed by an aircraft is represented as a directed path in the STN. Inasmuch as each aircraft has its own unique path,

the overlap labels are used to enforce the fundamental constraint that no two aircraft may be in the same place at the same time. If two time points overlap in space, then it is necessary to ensure that they do not overlap in time. This is done by introducing temporal constraints. Operations on the plan include insertion of an aircraft, deletion of an aircraft, and updating of the position and time of an aircraft. An aircraft is inserted into the plan by inserting its approach.

The approach generator generates a series of time points and temporal constraints that represent a hypothetical approach for an aircraft. It also generates the information for the overlaps for each time point of the plan. The approach generator includes a vector generator that iterates over options from most preferred (a full-procedure approach) to least preferred (one or more turns in a holding pattern, ending in a direct approach). Once a vector is generated, it must be verified as described in the next paragraph. If verified, it is incorporated into the plan.

The constraint verifier checks a plan and introduces temporal constraints where necessary to maintain veracity. The constraint verifier identifies time points that lack temporal ordering between themselves and members of their overlap set. It then incrementally inserts ordering constraints and verifies that each constraint is possible using temporal propagation. It can preferentially schedule new points either before or after pre-existing points, according to the preferences of the designers of the ATC system. Once a verified plan is found, the verifier returns "true." If a verified plan cannot be found, the verifier returns "false" and removes any extraneous temporal constraints it inserted.

This work was done by Russell Knight of Caltech for **NASA's Jet Propulsion Laboratory**. *Further information is contained in a TSP (see page 1).*

The software used in this innovation is available for commercial licensing. Please contact Karina Edmonds of the California Institute of Technology at (818) 393-2827. Refer to NPO-40445.