Process Analysis Tradeoff Tool (PATT). Typical inputs to PATT models include industry-average values of product size (expressed as number of lines of code), productivity (number of lines of code per hour), and number of defects per source line of code. The user provides the number of resources, the overall percent of effort that should be allocated to each process step, and the number of desired staff members for each step. The output of PATT includes the size of the product, a measure of effort, a measure of rework effort, the duration of the entire process, and the numbers of injected, detected, and corrected defects as well as a number of other interesting features.

In the development of the present model, steps were added to the IEEE 12207 waterfall process, and this model and its implementing software were made to run repeatedly through the sequence of steps, each repetition representing an iteration in a spiral process. Because the IEEE 12207 model is founded on a waterfall paradigm, it enables direct comparison of spiral and waterfall processes. The model can be used throughout a software-development project to analyze the project as more information becomes available. For instance, data from early iterations can be used as inputs to the model, and the model can be used to estimate the time and cost of carrying the project to completion.

This work was done by Carolyn Mizell of Kennedy Space Center, Charles Curley of ASRC Aerospace Corp., and Umanath Nayak of Portland State University. Further information is contained in a TSP (see page 1). KSC-13094

Algorithm That Synthesizes Other Algorithms for Hashing

A synthesized algorithm is guaranteed to be executable in constant time.

NASA's Jet Propulsion Laboratory, Pasadena, California

An algorithm that includes a collection of several subalgorithms has been devised as a means of synthesizing still other algorithms (which could include computer code) that utilize hashing to determine whether an element (typically, a number or other datum) is a member of a set (typically, a list of numbers). Each subalgorithm synthesizes an algorithm (e.g., a block of code) that maps a static set of key hashes to a somewhat linear monotonically increasing sequence of integers. The goal in formulating this mapping is to cause the length of the sequence thus generated to be as close as practicable to the original length of the set and thus to minimize gaps between the elements.

The advantage of the approach embodied in this algorithm is that it completely avoids the traditional approach of hash-key look-ups that involve either secondary hash generation and look-up or further searching of a hash table for a desired key in the event of collisions.

This algorithm guarantees that it will never be necessary to perform a search or to generate a secondary key in order to determine whether an element is a member of a set. This algorithm further guarantees that any algorithm that it synthesizes can be executed in constant time. To enforce these guarantees, the subalgorithms are formulated to employ a set of techniques, each of which works very effectively covering a certain class of hash-key values. These subalgorithms are of two types, summarized as follows:

- Given a list of numbers, try to find one or more solutions in which, if each number is shifted to the right by a constant number of bits and then masked with a rotating mask that isolates a set of bits, a unique number is thereby generated. In a variant of the foregoing procedure, omit the masking. Try various combinations of shifting, masking, and/or offsets until the solutions are found. From the set of solutions, select the one that provides the greatest compression for the representation and is executable in the minimum amount of time.
- Given a list of numbers, try to find one or more solutions in which, if each number is compressed by use of the modulo function by some value, then a unique value is generated.

This work was done by Mark James for Caltech for NASA's Jet Propulsion Laboratory. Further information is contained in a TSP (see page 1). NPO-45175

Algorithms for High-Speed Noninvasive Eye-Tracking System

One of the algorithms enables tracking at a frame rate of several kilohertz.

NASA's Jet Propulsion Laboratory, Pasadena, California

Two image-data-processing algorithms are essential to the successful operation of a system of electronic hardware and software that noninvasively tracks the direction of a person's gaze in real time. The system was described in "High-Speed Noninvasive Eye-Tracking System" (NPO-30700) *NASA Tech Briefs*, Vol. 31, No. 8 (August 2007), page 51.

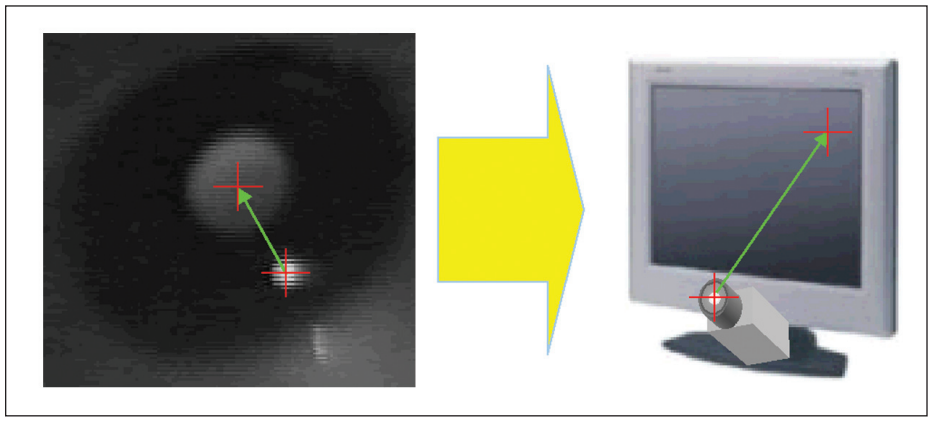
To recapitulate from the cited article: Like prior commercial noninvasive eye-tracking systems, this system is based on (1) illumination of an eye by a low-power infrared light-emitting diode (LED); (2) acquisition of video images of the pupil, iris, and cornea in the reflected infrared light; (3) digitization of the images; and (4) processing the digital image data to determine the direction of gaze from the centroids of the pupil and cornea in the images. Most of the prior commercial noninvasive eye-tracking systems rely on standard video cameras, which operate at frame rates of about 30 Hz. Such systems are limited to slow, full-frame operation.

The video camera in the present system includes a charge-coupled-device (CCD) image detector plus electronic circuitry capable of implementing an advanced control scheme that effects readout from a small region of interest (ROI), or subwindow, of the full image. Inas-

much as the image features of interest (the cornea and pupil) typically occupy a small part of the camera frame, this ROI capability can be exploited to determine the direction of gaze at a high frame rate by reading out from the ROI that contains the cornea and pupil (but not from the rest of the image) repeatedly.

One of the present algorithms exploits the ROI capability. The algorithm takes horizontal row slices and takes advantage of the symmetry of the pupil and cornea circles and of the gray-scale contrasts of the pupil and cornea with respect to other parts of the eye. The algorithm determines which horizontal image slices contain the pupil and cornea, and, on each valid slice, the end coordinates of the pupil and cornea. Information from multiple slices is then combined to robustly locate the centroids of the pupil and cornea images.

The other of the two present algorithms is a modified version of an older algorithm for estimating the direction of gaze from the centroids of the pupil and cornea. The modification lies in the use of the coordinates of the centroids, rather than differences between the coordinates of the centroids, in a gaze-mapping equation. The equation locates a gaze point, defined as the intersection of the gaze axis with a surface of interest, which is typically a computer display screen (see figure). The expected advantage of the modification is to make the gaze computation less dependent on some simplifying assumptions that are sometimes not accurate.



The **Vector Between the Centroids** of pupil and corneal reflections is computed and then used to compute the direction of gaze and the gaze point.

This work was done by Ashit Talukder, John-Michael Morookian, and James Lambert of Caltech for NASA's Jet Propulsion Laboratory. Further information is contained in a TSP (see page 1).

In accordance with Public Law 96-517, the contractor has elected to retain title to this invention. Inquiries concerning rights for its commercial use should be addressed to:

Innovative Technology Assets Management
JPL
Mail Stop 202-233
4800 Oak Grove Drive
Pasadena, CA 91109-8099
E-mail: iaoffice@jpl.nasa.gov

Refer to NPO-30699, volume and number of this NASA Tech Briefs *issue, and the page number.*

Adapting ASPEN for Orbital Express

Declarative modeling brings efficiency to encoded procedures and allows for guarantees on resource usage and time usage.

NASA's Jet Propulsion Laboratory, Pasadena, California

By studying the Orbital Express mission, modeling the spacecraft and scenarios, and testing the system, a technique has been developed that uses recursive decomposition to represent procedural actions declaratively, schema-level uncertainty reasoning to make uncertainty reasoning tractable, and lightweight, natural language processing to automatically parse procedures to produce declarative models.

Schema-level uncertainty reasoning has, at its core, the basic assumption that certain variables are uncertain, but not independent. Once any are known, then the others become known. This is important where a variable is uncertain for an action and many actions of the same type exist in the plan. For example, if the number of retries to purge pump lines was unknown (but bounded), and each attempt required a sub-plan, then, once the correct number of attempts required for a purge was known, it would likely be the same for all subsequent purges. This greatly reduces the space of plans that needs to be searched to ensure that all executions are feasible.

To accommodate changing scenario procedures, each is ingested into a tabular format in temporal order, and a simple natural-language parser is used to read each step and to derive the impact of that step on memory, power, and communications. Then an ASPEN (Activity Scheduling and Planning Environment) model is produced based on this analysis. The model is tested and further changed by hand, if necessary, to reflect the actual procedure. This results in a great savings of time used for modeling procedures.

Many processes that need to be modeled in ASPEN (a declarative system) are, in fact, procedural. ASPEN includes the ability to model activities in a hierarchical fashion, but this representation breaks down if there is a practically unbounded number of sub-activities and decomposition topologies. However, if recursive decomposition is allowed, HTN-like encodings are enabled to represent most procedural phenomena.

For example, if a switch requires a variable (but known at the time of the attempt) number of attempts to switch on, one can recurse on the number of remaining switch attempts and decompose into either the same switching activity with one less required attempt, or not decompose at all (or decompose into a dummy task), resulting in the end of the decomposition. In fact, any bounded procedural behavior can be modeled using recursive decompositions assuming that the variables impinging the dis-