JJM
SYSTEMS INC

NASA-CR-192897

C931021-U-2R07

AN EXPERT SYSTEM SHELL FOR INFERRING VEGETATION CHARACTERISTICS - INTERFACE FOR THE ADDITION OF TECHNIQUES (TASK H)

22 April 1993

Prepared for:

National Aeronautics and Space Administration
Goddard Space Flight Center
Greenbelt, MD 20771

Prepared by:

JJM Systems, Inc.
One Ivybrook Boulevard, Suite 190
Ivyland, PA 18974

JJM SYSTEMS INC

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

KEE	Knowledge Engineering Environment
VEG	VEGetation Workbench

SECTION 1.0

INTRODUCTION

All the NASA VEGetation Workbench (VEG) goals except the Learning System provide the scientist with several different techniques. When VEG is run, rules assist the scientist in selecting the best of the available techniques to apply to the sample of cover type data being studied. The techniques are stored in the VEG knowledge base. The design and implementation of an interface that allows the scientist to add new techniques to VEG without assistance from the developer have been completed.

In the previous version of VEG, the addition of a new technique was a complex process. For each new technique, extra units were added manually to the VEG knowledge base and additional Common Lisp code was added to the methods file. Changes were also made manually to the interface that allow the scientist to select which techniques to use.

A new interface that enables the scientist to add techniques to VEG without assistance from the developer has been designed and implemented. This interface does not require the scientist to have a thorough knowledge of Knowledge Engineering Environment (KEE) by Intellicorp or a detailed knowledge of the structure of VEG. The interface prompts the scientist to enter the required information about the new technique. It prompts the scientist to enter the required Common Lisp functions for executing the technique and the left hand side of the rule that causes the technique to be selected. A template for each function and rule and detailed instructions about the arguments of the functions, the values they should return, and the format of the rule are displayed. Checks are made to ensure that the required data have been entered, the functions compiled correctly and the rule parsed correctly before the new technique is stored. The additional techniques are stored separately from the VEG knowledge base.

When the VEG knowledge base is loaded, the additional techniques are not normally loaded. The interface allows the scientist the option of adding all the previously defined new techniques before running VEG. When the techniques are added, the required units to store the additional techniques are created automatically in the correct places in the VEG knowledge base. The methods file containing the functions required by the additional techniques is loaded. New rule units are created to store the new rules. The interface that allow the scientist to select which techniques to use is updated automatically to include the new techniques.

Task H has been completed. The interface that allows the scientist to add techniques to VEG has been implemented and comprehensively tested. The Common Lisp code for the Add Techniques system is listed in Appendix A. A Sun cartridge tape containing KEE and Common Lisp code for the new version of VEG, including the new interface, has been delivered to the NASA GSFC technical representative.

SECTION 2.0

THE ADD TECHNIQUES INTERFACE

When the ADD.TECHNIQUES option is selected from the VEG Administration screen, the Add Techniques interface, shown in Figure 2-1, is opened. The option DEFINE.NEW. TECHNIQUE allows the user to define a new technique and store it ready for subsequent loading into VEG. Selecting the menu option ADD. PREVIOUSLY.DEFINED.TECHNIQUES causes the data, functions and rules for previously defined new techniques that have been defined using DEFINE.NEW.TECHNIQUE to be read from files and added to VEG. The option PURGE.PREVIOUSLY.DEFINED.TECHNIQUES is used to delete all the techniques defined using DEFINE.NEW.TECHNIQUE from the files so they are no longer available to VEG. All three options are described in detail in this section.

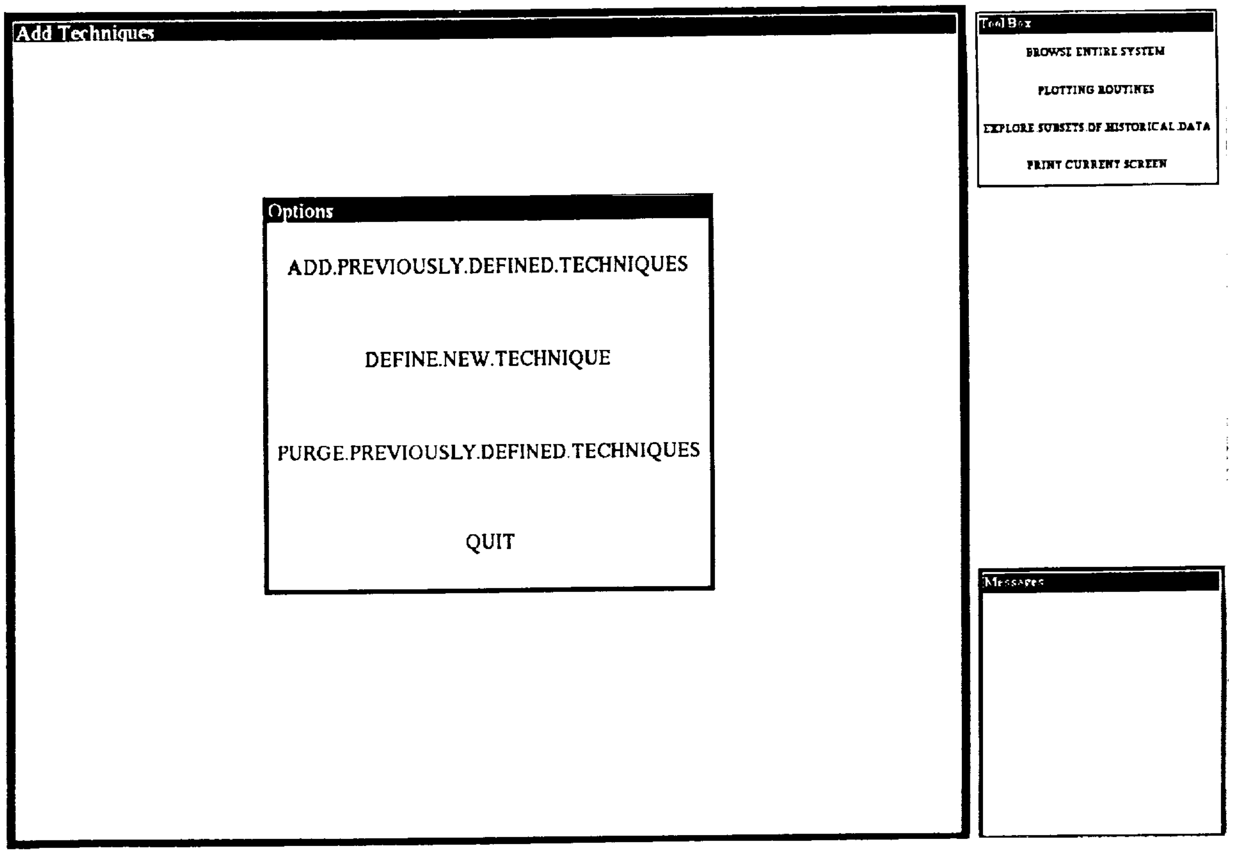
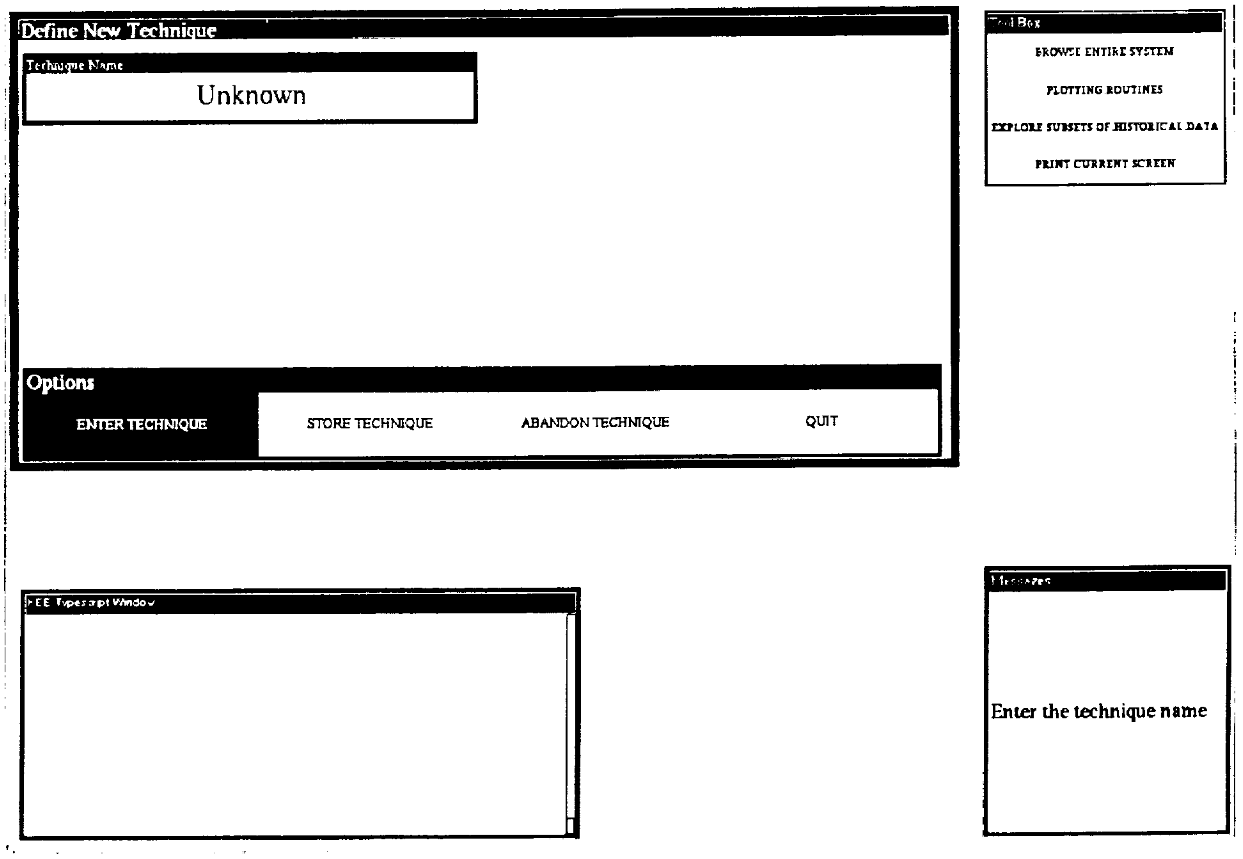


Figure 2-1
The Add Techniques Interface

SYSTEMS INC

2.1 DEFINING A NEW TECHNIQUE

When the user selects the option DEFINE.NEW.TECHNIQUE from the Add Techniques Interface (Figure 2-1), the Define New Technique Screen is opened. This screen allows the user to enter the data, functions and rule for the new technique, store the new technique, abandon the new technique or quit the screen. When the screen is first opened, only the "Technique Name" and "Options" subwindows are opened. The user is prompted to enter the name of the new technique. Figure 2-2 shows this. When the Define New Technique Screen is open, the KEE Typescript Window is visible. This allows the user to see any error messages that are displayed in the Typescript window when the functions for the new techniques are compiled or the rules are parsed.



Enter the technique name

Figure 2-2
The Define New Technique Screen When it is First Opened

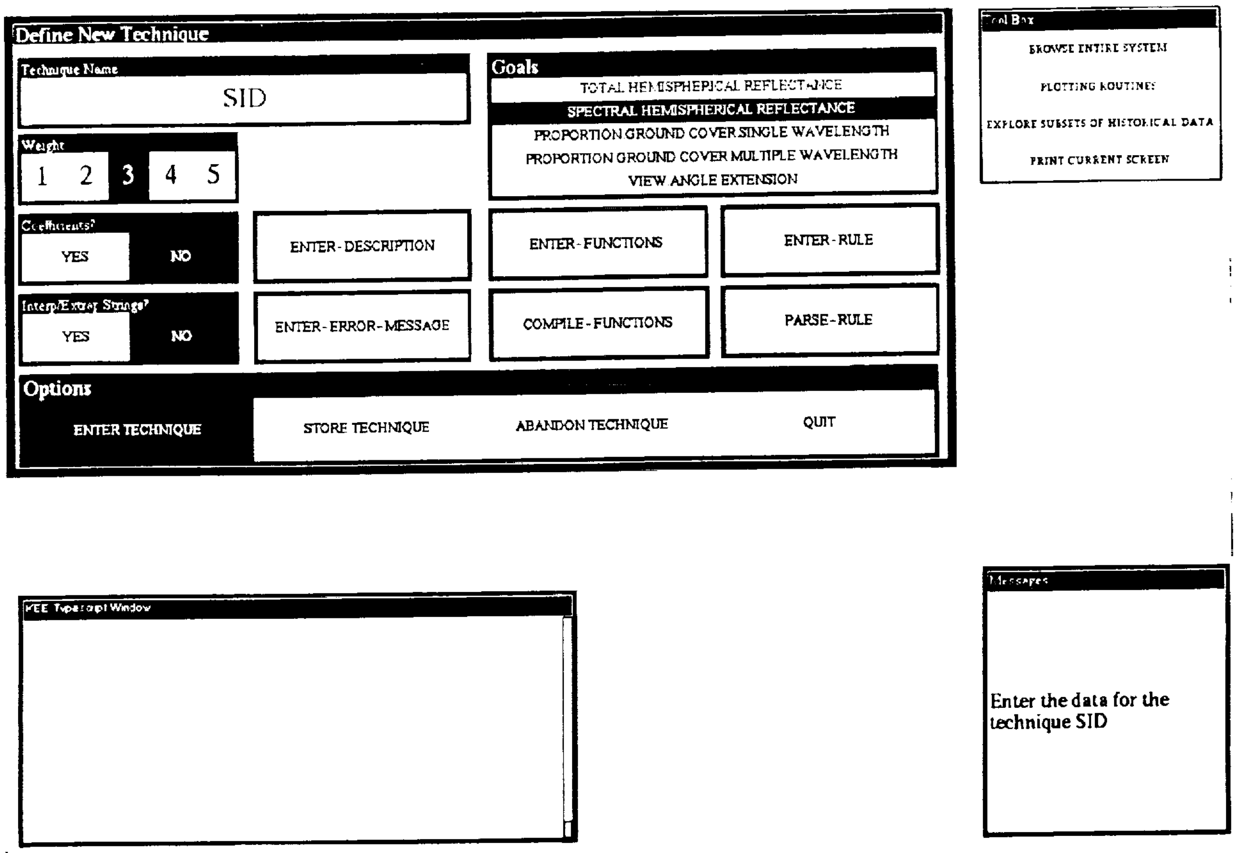
A new unit called ADD.TECHNIQUES has been created in the VEG knowledge base. Figure 2-3 shows the slots in this unit. Each subwindow in the Define New Techniques Screen is a KEE ActiveImage connected to a slot in the unit ADD.TECHNIQUES. Data for the new technique are entered via the interface and stored in slots such as DESCRIPTION and GOALS of the ADD.TECHNIQUES unit. The slots ENTER.DESCRIPTION, ENTER.ERROR.MESSAGE, ENTER.FUNCTIONS, COMPILE.FUNCTIONS, ENTER.RULE and COMPILE.RULE are methods slots. They contain methods which are executed when the user left-clicks on the method-actuator ActiveImage attached to the slot.

```
COEFFS.METHOD
COEFFS.P
COMPILE.FUNCTIONS
DESCRIPTION
ENTER.DESCRIPTION
ENTER.ERROR.MESSAGE
ENTER.FUNCTIONS
ENTER.RULE
ERROR.MESSAGE
GOALS
INITIALIZED.FUNCTION
INITIALIZED.RULE
INTERPOLATE.EXTRAPOLATE?
MESSAGE
NEW.TECH.OPTIONS
OK.TO.USE
OPTIONS
PARSE.RULE
PREVIOUS.TECHS
RULE.PARSED
TECH.NAME
TECHNIQUE.METHOD
WEIGHT
YES.NO
```

Figure 2-3
Slots in the Unit ADD.TECHNIQUES

The first step in defining a new technique is to enter the name of the new technique into the subwindow labelled "Technique Name" (Figure 2-1). When the Define New Technique screen is opened, the names of any previously defined new techniques are read from the file and stored in the slot PREVIOUS.TECHS of the unit ADD.TECHNIQUES. If the name of the new technique matches a value in the PREVIOUS.TECHS slot or an existing VEG unit, a message is displayed. This message indicates that the technique has already been defined. In this case, the technique name is not stored. Otherwise, the technique name is stored in the TECH.NAME slot of the

ADD.TECHNIQUES unit. For a function named "SID," for example, the function names "tech-SID," "coeffs-SID" and "SID.ok" are then constructed and stored in the slots TECHNIQUE.METHOD, COEFFS.METHOD and OK.TO.USE, respectively. After the function name has been stored, the rest of the subwindows of the Define New Technique interface are automatically opened. The user must enter or select data or activate methods in all the subwindows of this interface before the technique can be saved. Figure 2-4 shows the Define New Technique screen after all the data for a new technique has been entered. If the user enters a technique name and then subsequently enters another technique name before storing the previously named technique, the interface is re-initialized and any data, functions, or rule entered for the previously named technique are lost.


Define New Technique

Technique Name: SID

Weight: 1 2 [✓ 3] 4 5

Goals:
TOTAL HEMISPHERICAL REFLECTANCE
[✓ SPECTRAL HEMISPHERICAL REFLECTANCE]
PROPORTION GROUND COVER SINGLE WAVELENGTH
PROPORTION GROUND COVER MULTIPLE WAVELENGTH
VIEW ANGLE EXTENSION

Coefficients?: YES [✓ NO]

Interp/Extrap Strings?: YES [✓ NO]

Options: [✓ ENTER TECHNIQUE] STORE TECHNIQUE ABANDON TECHNIQUE QUIT

Messages: Enter the data for the technique SID

Figure 2-4
The Define New Technique Screen After All the Data for a New Technique have been Entered

The "Weight" subwindow (Figure 2-4) holds a number between 1 and 5 which indicates the priority to be given to the technique when the techniques are ranked. The highest priority is 5. Selecting "YES" in the subwindow labelled "Coefficients" indicates that the function used for executing the technique requires coefficients. The default selection for the "Interp/Extrap Strings?" subwindow is "NO." If the technique requires the strings in the reflectance data to be interpolated and extrapolated before the technique is applied, "YES" must be selected in this subwindow. The string techniques for the goal SPECTRAL.HEMISPHERICAL.REFLECTANCE require this extrapolation.

The subwindow labelled "Goals" holds the VEG goal to which the technique applies. New techniques can be added for the VEG goals SPECTRAL.HEMISPHERICAL.REFLECTANCE, PORTION.GROUND.COVER (either single or multiple wavelength) and VIEW.ANGLE. EXTENSION. Only one total hemispherical reflectance technique is currently available in VEG. The interface to select or rank total hemispherical reflectance techniques has not yet been implemented. If the user selects the goal TOTAL.HEMISPHERICAL.REFLECTANCE, a message is displayed. The message indicates that the techniques for this goal have not been implemented and the new technique will not be stored.

When the user is running VEG and chooses to select the techniques manually, the User Pick Techniques screen is opened. Each time a technique is selected, a description of the technique is displayed on the screen. If the selected technique is suitable for the sample being studied, the message "This technique is suitable for this sample" is displayed. Otherwise an error message is displayed. Left-clicking on ENTER-DESCRIPTION and ENTER-ERROR-MESSAGE in the Define New Technique screen enables the user to enter the description and error message that will be displayed in the User Pick Techniques screen when the technique is added to VEG .

When the user left-clicks on ENTER-FUNCTIONS, the temporary file "temp.lisp" is opened. Templates for the functions required by the new technique are written to this file. The names for the technique functions that were constructed when the technique name was entered, are automatically incorporated in the templates. Then the editor is opened and the user is prompted to enter the new functions. Functions are required to execute the technique, calculate the coefficients (if any) required by the technique, and to determine whether the technique is suitable for a particular sample. A prompt in the "Message" window tells the user how to save the temporary file and exit the editor. The method actuator COMPILE-FUNCTIONS is used to compile the functions for the new technique. The method first checks that functions have been entered. If the file "temp.lisp" is not found, an error message is displayed and no attempt at compilation is made. Otherwise, the function file is compiled and any compilation errors or warnings are displayed in the KEE Typescript window. If the compilation is successful, the compiled functions are stored in the binary file "temp.sbin." If any errors occurred during the compilation, this file remains empty. Note that the functions are compiled both for efficiency and also to create additional error checking. If the user reselects ENTER-FUNCTIONS while still entering data for the same technique, the previously edited temporary file is opened once again. The changes made in the previous edit session are not lost. This allows the user to edit a file repeatedly until the functions are correct and the file compiles successfully.

When the user selects ENTER-RULE, another temporary file is opened and a template for the rule is written to the file. The name of the new technique is incorporated in the template. The editor is then opened, and the user is prompted to modify the template to create the rule required for the new technique. If the user left-clicks on PARSE-RULE, checks are made to confirm that a rule has been entered, and that the rule contains the same number of left and right parentheses. The failure of either of these checks causes an error message to be displayed. Otherwise, an attempt is made to parse the rule using a user-defined function named TEST-RULE-PARSES. This function sets up the structure so that the KEE parser can parse the new rule. The function creates a temporary rule unit as an instance of the KEE unit VEG.RULES. The newly entered rule is stored in the EXTERNAL.FORM slot of the temporary rule unit. The KEE PARSE function is then applied to the rule unit. If no parse errors occur, the function returns T. Otherwise the function returns NIL. Before the value is returned, the temporary rule unit is deleted from VEG. If the rule did not parse correctly, an error message is displayed in the "Message" box. The slot RULE.PARSED in the ADD.TECHNIQUES unit is used as a flag to indicate whether or not a correctly parsed rule has been entered. Note that even though a rule contains the same number of left and right parentheses and parses correctly, it might still be incorrect.

When the user selects the option STORE-TECHNIQUES, checks are made to make sure that all the required data has been entered, the functions have been defined and compiled successfully, and a correctly parsed rule has been entered. If the checks are unsuccessful, nothing is stored and the user is prompted to complete entry of the required data. Otherwise, the data and rule are appended to the file "new-tech-data," and the functions are appended to the file "new-tech.lisp" which is immediately compiled. Table 2-1 shows the format in which the data taken from the ActiveImages shown in Figure 2-4 would be stored in the file "new-tech-data." Next, the new technique name is added to the PREVIOUS.TECHS slot of the unit ADD.TECHNIQUES. The interface is then re-initialized, and the subwindows, except the "Technique Name" and "Options" subwindows, are closed, as in Figure 2-2.

Table 2-1
The Format in which the Data taken from the ActiveImages in Figure 2-4 would be Stored in the File "new-tech-data"

DESCRIPTION	VALUE
Goal	SPECTRAL.HEMISPHERICAL.REFLECTANCE
Technique Name	SID
Description	"TECHNIQUE SID - A NEW TECHNIQUE FOR CALCULATING THE SPECTRAL HEMISPHERICAL REFLECTANCE OF A SAMPLE THAT HAS DATA AT 4 VIEW ANGLES"
Error Message	"TECHNIQUE SID IS UNSUITABLE FOR THIS SAMPLE BECAUSE IT DOES NOT HAVE DATA AT 4 VIEW ANGLES"
Technique function	tech-SID
Interp/extrap Strings?	NO
Function uses Coefficients?	YES
Coefficients function	coeffs-SID
Suitability Function	SID.ok
Weight	3
Rule	(IF (THE CURRENT.SAMPLE.WAVELENGTHS OF ESTIMATE.HEMISPHERICAL.REFLECTANCE IS ?X) (THE NUMBER.VIEW.ANGLES OF ?X IS 4) THEN (LISP (ADD.VALUE ?X (QUOTE TECHNIQUES) (QUOTE SID))))

Selecting the option ABANDON-TECHNIQUE causes the deletion of any data, functions, and rules that have been entered but not stored. The interface is then re-initialized. This is, in effect, a panic button that the user can activate to stop the process at any point.

The Define New Technique interface can be exited by selecting the QUIT option. Any partially entered technique is deleted when this option is selected. It is important to note that at this stage any newly defined and stored techniques have been saved in files, but they have not yet been added to VEG.

2.2 ADDING PREVIOUSLY DEFINED NEW TECHNIQUES

Adding a new technique to VEG involves several steps. A new unit must be created in the VEG knowledge base to hold the data required by the technique. Another unit is required to hold the rule that will enable the technique to be selected when it is appropriate for the cover type sample being studied. The functions required by the technique must be loaded. The interface must be updated so that the displays that list the available techniques for a VEG goal include the additional techniques. It is important to note that this system automatically places newly created units in their proper location in the system.

When the user selects the option ADD.PREVIOUSLY.DEFINED.TECHNIQUES from the Add Techniques screen (Figure 2-1), VEG first checks that the files "new-tech-data" and "new-tech.lisp" are present. These files hold the data, functions and rules for the new techniques. If either of these files is missing, the message "No techniques available" is displayed in the "Messages" box and processing stops. If the required files are present, the file "new-tech.sbin" is loaded. This file is the compiled version of the file "new-tech.sbin" that contains the functions required by the new techniques. Processing of the data then begins. The data are read from the file "new-tech-data." The format of this file was shown in Table 2-1.

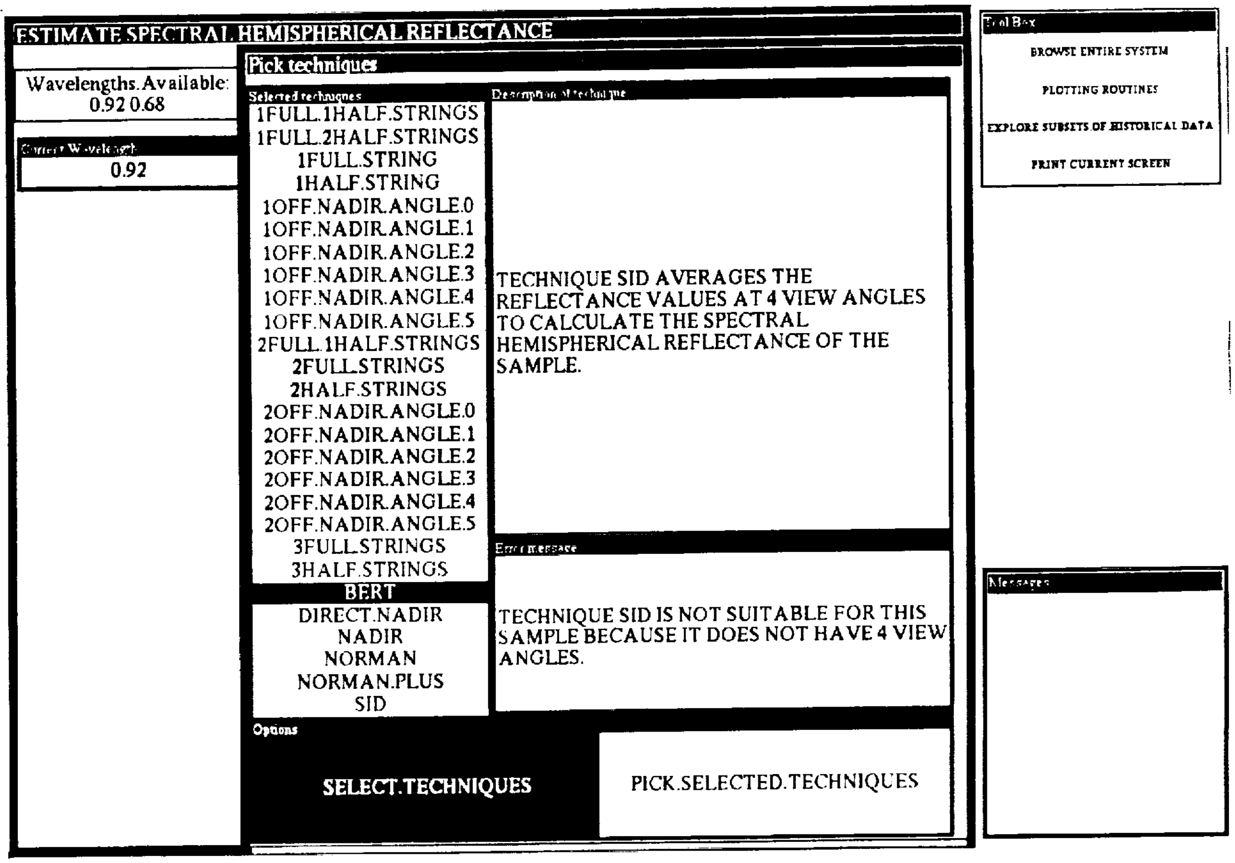
The VEG goal to which the new technique applies is read first. The techniques for each VEG goal are stored in instances of different subclasses of the unit TECHNIQUES. For example, techniques for the goal SPECTRAL.HEMISPHERICAL.REFLECTANCE are stored in instances of the subclass unit SPECTRAL.HEMISPHERICAL.REFLECTANCE.TECHNIQUES. The rules for each goal have names that reflect the goal to which they apply, and they are stored in ruleclasses that are subclasses of the unit VEG.RULES. For example, rules for the goal SPECTRAL.HEMISPHERICAL. REFLECTANCE have names that are prefixed by "HRTR," and they are stored in instances of the unit HEMISPHERICAL.REFLECTANCE.TECHNIQUES. The techniques for different goals are displayed in different windows in the interface. After the goal has been read from the file, the names of the technique subclass, rule prefix, ruleclass, and interface window that apply to the goal are identified.

The technique name is next read from the file. The system will not allow the same technique to be added more than once to VEG. If the technique has already been added to VEG, the remainder of the data for this technique is skipped in the file. Otherwise, a new unit is created as an instance of the correct VEG subclass to which the technique applies. Information about the technique, such as its description, the name of the technique function, and whether the technique function requires coefficients, are read from the file and stored in this unit. A rule unit is then created in the correct ruleclass. The name of the rule unit is constructed using the appropriate prefix. The rule is read from the file and stored in the "External Form" slot of the rule unit.

When VEG is running and a cover type sample is being processed, the user can select the techniques to apply to the sample using the Pick Techniques Screen. The names of all the available VEG techniques for the appropriate goal are displayed on this screen. The final step in adding a new technique to VEG is to update this screen to include the new technique. Figure 2-5 shows the

Pick Techniques Screen for the goal SPECTRAL.HEMISPHERICAL.REFLECTANCE after two new techniques called SID and BERT have been added. In the example in the figure, BERT is in dark because it has already been selected. The user has attempted to select SID. However, an error message is being displayed because SID is not suitable for the sample being studied.

Adding new techniques continues until the end of the file "new-tech-data" is reached. The message "Loading ...." is then removed from the screen.



1FULL.1HALF.STRINGS
1FULL.2HALF.STRINGS
1FULL.STRING
1HALF.STRING
1OFF.NADIR.ANGLE.0
1OFF.NADIR.ANGLE.1
1OFF.NADIR.ANGLE.2
1OFF.NADIR.ANGLE.3
1OFF.NADIR.ANGLE.4
1OFF.NADIR.ANGLE.5
2FULL.1HALF.STRINGS
2FULL.STRINGS
2HALF.STRINGS
2OFF.NADIR.ANGLE.0
2OFF.NADIR.ANGLE.1
2OFF.NADIR.ANGLE.2
2OFF.NADIR.ANGLE.3
2OFF.NADIR.ANGLE.4
2OFF.NADIR.ANGLE.5
3FULL.STRINGS
3HALF.STRINGS
BERT
DIRECT.NADIR
NADIR
NORMAN
NORMAN.PLUS
SID

TECHNIQUE SID AVERAGES THE REFLECTANCE VALUES AT 4 VIEW ANGLES TO CALCULATE THE SPECTRAL HEMISPHERICAL REFLECTANCE OF THE SAMPLE.

TECHNIQUE SID IS NOT SUITABLE FOR THIS SAMPLE BECAUSE IT DOES NOT HAVE 4 VIEW ANGLES.

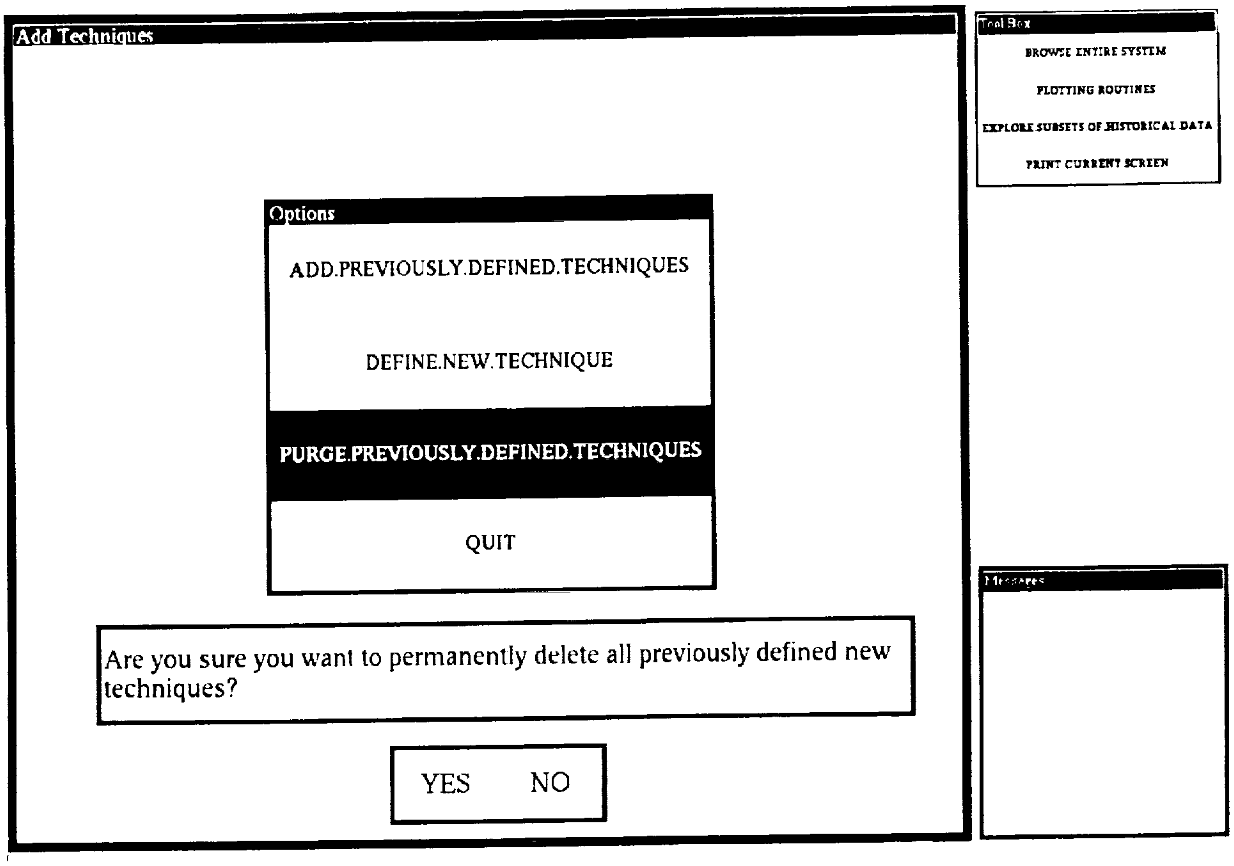
Figure 2-5
The Pick Techniques Screen for the Goal SPECTRAL.HEMISPHERICAL.REFLECTANCE

2.3 PURGING PREVIOUSLY DEFINED NEW TECHNIQUES

This option allows the user to delete from the files any techniques defined using the DEFINE.NEW.TECHNIQUE option. When this option is activated the techniques are permanently deleted from the files so they are no longer available to VEG.

When the user selects the option PURGE.PREVIOUSLY.DEFINED.TECHNIQUES from the Add Techniques Interface (Figure 2-1), additional subwindows are opened, as shown in Figure 2-6. The user is asked to confirm that the techniques should be deleted. If the user left-clicks on "YES," the file "new-tech-data," that contained the data and rules for the new techniques, is deleted. The technique functions are then removed form the file "new-tech.lisp." If the user selects "NO," the techniques are not deleted. Finally, the additional subwindows are removed from the screen.

Selecting QUIT from the Add Techniques screen (Figure 2-1) returns the user to the Administration screen.



Are you sure you want to permanently delete all previously defined new techniques?

Figure 2-6
The Add Techniques Interface with the Option PURGE.PREVIOUSLY.DEFINED.TECHNIQUES Selected

SECTION 3.0

TESTING AND RESULTS

The Add Techniques options were tested using both valid and invalid data. When errors were detected, they were corrected and the test runs were repeated to ensure that the corrections were successful. The tests were designed to test the typical range of user behavior. The test runs and results are described in this section.

3.1 TEST 1

The purpose of Test 1 was to test the navigation back and forth through the various menu levels from the VEG top level to the Add Techniques Screen. The user left-clicked on RUN.VEG, ADMINISTRATION and ADD.TECHNIQUES on successive screens. As expected, the Add Techniques Screen was opened. The user then selected QUIT in each successive menu to navigate back to the top level of VEG. This test showed that the screens between the VEG top level and the Add Techniques Screen were opened and closed in the correct sequence.

3.2 TEST 2

This test was designed to test the operation of the Add Previously Defined New Techniques option before any new techniques had been defined. At this time, no additional techniques had been defined so the file "new-tech-data" had not been created and the file "new-tech.lisp" contained only comments. The option ADD.PREVIOUSLY. DEFINED.NEW.TECHNIQUES was selected from the Add Techniques menu. The message "No techniques available" was displayed in the "Messages" window. This test showed that the system could deal correctly with an attempt to add previously defined new techniques before any new techniques had been defined.

3.3 TEST 3

Test 3 was intended to test the entry of valid data to define a new technique. The DEFINE.NEW.TECHNIQUE option was selected from the Add Techniques menu. The Define New Technique Screen was opened and the user was prompted to enter the name of the new technique. At this time, most of the subwindows in the Define New Technique screen were closed. The user entered the name "SID" for the new technique. Then the rest of the subwindows were automatically opened and the user was prompted to enter the rest of the data for the new technique. Note that the technique SID was invented by the developer for testing the system. It is not a real technique. The user selected "NO" in both the "Coefficients" and the "Interp/Extrap?" boxes. A weight of three was specified. The goal SPECTRAL.HEMISPHERICAL. REFLECTANCE was selected. A description and an error message were entered. The user left clicked on ENTER-FUNCTIONS. The function SID.ok was edited so that it would return T if the sample had four view angles and nil otherwise. The function tech-SID was edited to return the average reflectance value of the four view angles. Because the function to execute the technique SID does not require coefficients, the function coeffs-SID was deleted. The file was then saved and compiled. Compilation was successful. The user selected ENTER-RULE and edited the template so that the rule would fire if the cover type sample had four view angles. The rule parsed correctly. Finally, the STORE.TECHNIQUE option was selected from the menu at the bottom of the screen. No error messages were displayed. Inspection of the files "new-tech-data" and "new-tech.lisp" showed that the new technique had been successfully saved. The user was then prompted to enter the name of the next new technique.

This test showed that the Define New Technique interface was working correctly when valid data were entered. The data, functions, and rule for the new technique were entered via the interface and successfully stored in the appropriate files. Inspection of the created files confirmed this.

3.4 TEST 4

The Define New Technique Interface should not allow the same technique name to be used more than once. In Test 4, attempts were made to use the same technique name twice.

In the first part of Test 4, the technique name "SID" was entered again. The message "Technique SID has already been defined" was displayed in the messages box.

In the second part of Test 4, the technique name "NORMAN" was entered. The message "Technique NORMAN has already been defined" was displayed in the messages box. Technique NORMAN is a technique for estimating spectral hemispherical reflectance that is part of the VEG knowledge base.

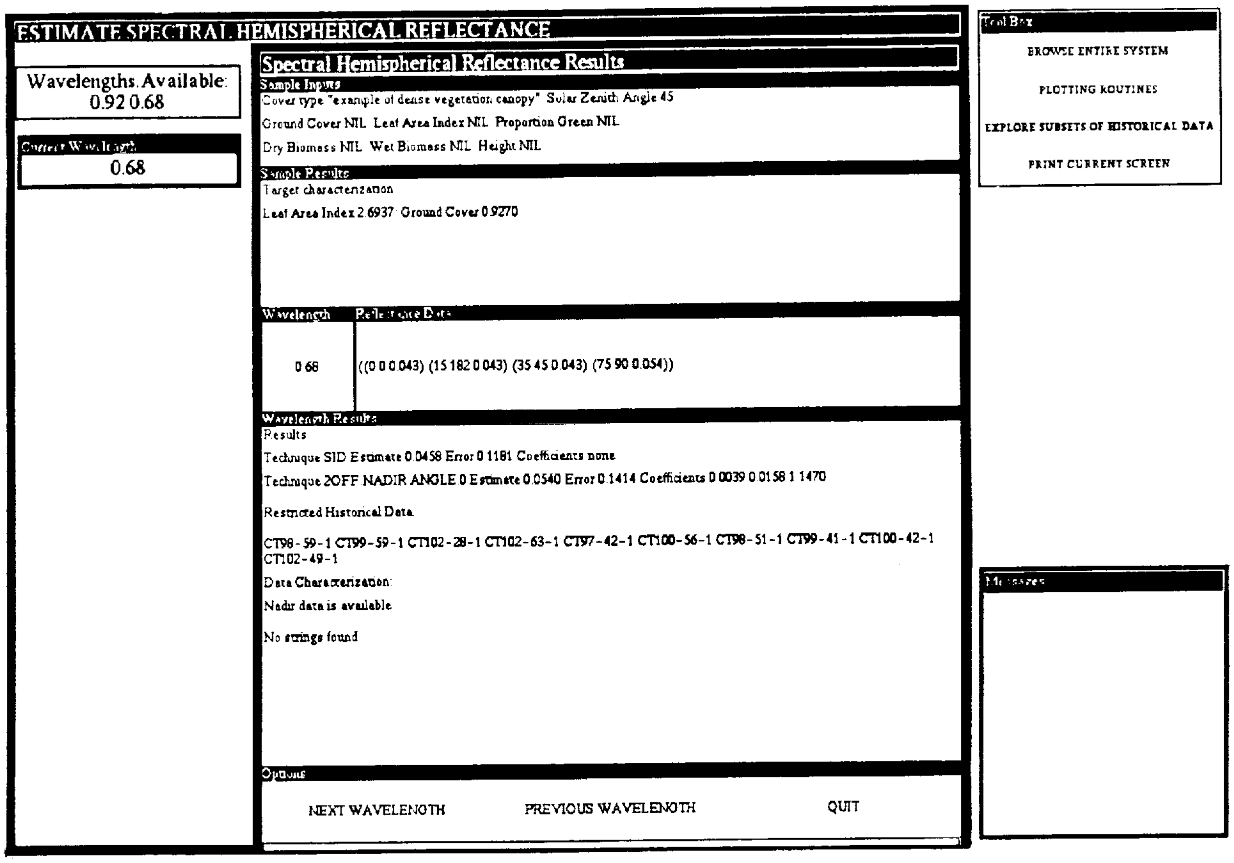
Test 4 showed that the Add Techniques interface will not allow the same technique name to be used twice. It can detect attempts to re-use a technique name either already stored in VEG, or saved in the file of previously defined new techniques.

3.5 TEST 5

This test was designed to test the addition of multiple, previously defined new techniques to VEG from files and the operation of VEG using the newly added techniques. At the beginning of this test, another new technique called BERT was defined for the goal SPECTRAL. HEMISPHERICAL.REFLECTANCE. This technique was also invented by the developer for testing purposes. The technique was effectively the technique DIRECT.NADIR, applied only to samples with one view angle. Technique BERT was saved in the files "new-tech-data" and "new-tech.lisp." Then the ADD. PREVIOUSLY.DEFINED.NEW.TECHNIQUES option was selected for the Add Techniques interface. The message "Loading ...." was displayed. After loading had been completed, this message was removed. Inspection of the VEG knowledge base showed that new units had been created in the correct places in the VEG knowledge base. These units held the data and the rules for the new techniques.

The user ran VEG in Research Mode using the goal SPECTRAL.HEMISPHERICAL. REFLECTANCE. Techniques SID and BERT were designed to operate on samples with data at four and one view angles, respectively. Sample 4 was selected as the sample to be studied so that both new techniques could be tested. This sample has four view angles at wavelength 0.68 μm and one view angle at wavelength 0.92 μm.

Sample 4 was processed. The manual method of selecting techniques was chosen for the data at both wavelengths. As shown in Figure 2-5, both new techniques were displayed on the User Pick Techniques screen. The user attempted to select both new techniques for the data at both wavelengths. The functions that check the suitability of each technique for each sample worked correctly. The user was prevented from selecting technique SID for the wavelength 0.92 μm since data at only one view angle were available at this wavelength. Similarly, the user was prevented from selecting technique BERT for the wavelength 0.68 μm. After selecting the techniques manually, the user also chose to have VEG choose the techniques automatically. The purpose of this was to test the rules for selecting the techniques. Technique SID was among the techniques chosen for the wavelength 0.68 μm and technique BERT was among the techniques chosen for the wavelength 0.92 µm. This proved that the new rules were operating correctly. Then the techniques were ranked and the user chose to use the best two techniques for each wavelength so that both new techniques were used. The techniques were then executed and the results were displayed. Figure 3-1 shows some of the results obtained. The results were correct. Test 5 showed that newly defined techniques could be added to VEG and used correctly within VEG.



```
ESTIMATE SPECTRAL HEMISPHERICAL REFLECTANCE

Wavelengths Available:
0.92 0.68

Current Wavelength
0.68

Spectral Hemispherical Reflectance Results
Sample Inputs
Cover type "example of dense vegetation canopy"  Solar Zenith Angle 45
Ground Cover NIL  Leaf Area Index NIL  Proportion Green NIL
Dry Biomass NIL  Wet Biomass NIL  Height NIL

Sample Results
Target characterization
Leaf Area Index 2.6937  Ground Cover 0.9270

Wavelength   Reflectance Data
0.68         ((0 0 0.043) (15 182 0.043) (35 45 0.043) (75 90 0.054))

Wavelength Results
Results
Technique SID Estimate 0.0458 Error 0.1181 Coefficients none
Technique 2OFF NADIR ANGLE 0 Estimate 0.0540 Error 0.1414 Coefficients 0.0039 0.0158 1.1470

Restricted Historical Data

CT98-59-1 CT99-59-1 CT102-28-1 CT102-63-1 CT97-42-1 CT100-56-1 CT98-51-1 CT99-41-1 CT100-42-1
CT102-49-1
Data Characterization:
Nadir data is available

No strings found
```

Figure 3-1
The Output Screen at the End of Test 5

3.6 TEST 6

In Test 6, new techniques were defined and added to VEG for the goals VIEW.ANGLE. EXTENSION and PORTION.GROUND.COVER. Then VEG, along with the new techniques, was run. The test showed that the addition of new techniques for these goals was successful.

3.7 TEST 7

In Test 7, a new technique was defined for the VEG goal TOTAL.HEMISPHERICAL. REFLECTANCE. This option is not yet available. When the user selected this goal from the Define New Technique Screen, an error message was displayed. The user ignored this error message and continued to enter data for the new technique. When the user attempted to store the data for the new technique, another error message was displayed and the data were not stored. This test showed that the system was correctly blocking attempts to store new techniques for the goal TOTAL.HEMISPHERICAL.REFLECTANCE.

3.8 TEST 8

Test 8 was designed to ensure that an incomplete set of data and functions for a new technique would not be stored. It was also designed to ensure that attempting to compile functions before they were defined, or to parse a rule before it was entered, would produce appropriate error messages.

The user opened the Define New Technique screen and immediately attempted to store a new technique, even though none had yet been defined. The message "Technique name not found - data not stored" was displayed and nothing was stored. After entering a technique name, the user again attempted to store the new technique. This time the user was prompted to select the goal. The user continued to enter the data items, one at a time, in response to the prompts, each time attempting to store the new technique. As expected, every attempt to save the incomplete technique data was unsuccessful.

After all the data items had been entered, the message "Functions not found - data not stored" was displayed when the user attempted to store the technique. The user then attempted to compile the functions before entering them. This time the error message "Functions not found - enter them before compiling" was displayed. The user entered the required functions and then once again attempted to store the technique. This time the error message informed the user that the functions must be compiled. The functions were then successfully compiled. The next attempt to store the technique produced the error message "Rule not found - data not stored." The user attempted to parse the rule before entering it. Again an error message was displayed. After entering a rule, the user tried again to store the data. This time the user was prompted to parse the rule. After the rule had been successfully parsed, an attempt to store the new technique succeeded.

This test showed that incomplete data for a technique could not be stored. It also showed that an appropriate error message is displayed if the user attempts to compile functions before defining them or to parse a rule before entering it.

3.9 TEST 9

The user may enter invalid functions for a new technique. The Add Techniques interface tests whether new functions will compile and it does not store a new technique unless the new functions compile correctly. Test 9 was designed to test the behavior of the system with invalid technique functions.

Various errors such as unmatched right parentheses, undefined functions, incorrect arguments to functions, and missing arguments to functions were introduced into the function file. These produced warnings which were reported in the KEE Typescript window when the function file was compiled. However, a compiled function file was created in each of these cases and attempts to store the function were successful.

An unmatched left parenthesis error was also introduced into the function file. When this file was compiled, the "End of file reading in a list" error was signaled and a list of debugging action options was shown in the KEE Typescript Window. The user entered the debugging action number 1 to kill the process in this case. No compiled file was created and the user was prevented from storing the new technique.

This test showed that many errors in the technique functions produce warnings rather than error messages. Although the interface prevents the user from storing a technique function that produces a compiler error, it does not prevent the user from storing a function that produces a compiler warning. The user should correct warnings before storing a technique, even though the interface does not insist on this. Test 9 showed that there are limitations on the detection of invalid functions by the Define New Function system.

3.10 TEST 10

This test was designed to test the parsing of a new rule and to determine the limitations of the system in preventing invalid rules from being stored. Various errors were introduced into a rule to determine how the system would respond. For example, an extra term was added to a rule clause. It was interpreted by the rule compiler as a literal so the rule parsed successfully even though it was incorrect. In this case, the invalid rule was stored.

In separate tests, extra left and right parentheses were added to the rule. These were detected before the rule was parsed, and in each case, the technique with the invalid rule was not stored.

In separate tests, the IF and THEN clauses of the rule were omitted. Despite these omissions, the rules parsed successfully.

Test 10 showed that the use of the KEE rule parser to detect errors in a rule is limited to some syntactic errors. Note that adding new rules is the most difficult part of the Add Techniques system to control. It is quite possible to add rules that are nonsense. The user is cautioned, therefore, to be careful when adding rules.

3.11 TEST 11

Test 11 was designed to test the ABANDON.TECHNIQUE option from the Define New Technique screen. Several new techniques were entered. Each time, the entry of the new technique was abandoned at a different point. In every case, the interface was initialized correctly and all the data, functions and rule for the abandoned technique were correctly deleted. This test showed that the ABANDON.TECHNIQUE option was operating correctly.

3.12 TEST 12

This test was designed to test the operation of the PURGE.PREVIOUSLY.DEFINED. TECHNIQUES option from the Add Techniques interface (Figure 2-1). When this option was selected, additional subwindows were opened. The user was prompted to confirm that the techniques should be deleted. The user left-clicked on "NO." The message "Techniques not deleted" was displayed in the "Messages" box and the subwindows were then closed. Inspection of the files "new-tech-data" and "new-tech.lisp" confirmed that the techniques had not been deleted. The user then selected the PURGE.PREVIOUSLY.DEFINED.TECHNIQUES option again. This time the user left-clicked on "YES" to confirm that the techniques should be deleted. The message "Techniques deleted" was displayed in the "Messages" box and the subwindows were closed once again. Inspection of the files confirmed that the file "new-tech-data" had been deleted and the file "new-tech.lisp" contained only headings. Test 12 showed that the PURGE.PREVIOUSLY. DEFINED.TECHNIQUES option was operating correctly.

SECTION 4.0

CONCLUSIONS

The Add Techniques system implements a software component for defining additional analysis techniques that are used to evaluate samples of cover type data. The system provides a detailed, window driven, user interface which organizes the entry of the technique definitions. Dynamic error checking, file management, object creation, and definition management facilities are provided.

The technique definition has multiple components that include description, error message, function body, rule for determining when the technique can be used, and technique priority. The user follows instructions on various windows to input technique elements. Error checking is done interactively by the system. The function component of the definition is compiled for efficiency.

The new definition is managed so that it is logically isolated from the basic VEG system. In a separate step, the new technique may be loaded for use.

Testing of the Add Techniques system focused on the expected range of typical user behavior. It proved to be reasonably robust and user-friendly.

JJM SYSTEMS INC

APPENDIX A

LISTING OF METHODS FILES FOR THE ADD TECHNIQUES SYSTEM

```
;;; veg-methods5.lisp
;;;
;;; Code to allow the user to add techniques to VEG
;;;
;;; Written by Ann Harrison
;;; Created April 1, 1993
;;; Last modified April 20, 1993

(in-package 'kee)

(defun open-add-techniques-menu ()
"Open the screen for adding techniques."
  (unitmsg 'viewport-add.techniques.1 'open-panel!)
  (remove.all.values 'add.techniques 'options))

;;;------------------------------------------------------------------------
;;; Functions required to add previously defined techniques from files to VEG
;;;------------------------------------------------------------------------

(defun add-previously-defined-techniques ()
"Loads previously defined additional techniques from a file."
  (cond ((and (probe-file "new-tech.sbin")
              (probe-file "new-tech-data"))
          (my-documentation-print "Loading ........")
          (load "new-tech")      ; Load the file containing the functions
                                 ; required by the techniques
          (with-open-file (str "new-tech-data" :direction :input)
            (load-tech-data-from-file str))
          (clear-prompt))
         (t (my-documentation-print "No techniques available"))))

;;; Note that the function> read-file is in the methods file veg-methods1.lisp

(defun load-tech-data-from-file (str)
"Sets up the appropriate arguments and calls the function to create the units
to store the data for the technique and rule units in VEG."
  (do ((goal (read-file str)(read-file str)))
      ((null goal) nil) ;End of file
    (case goal
      (total.hemispherical.reflectance
            (my-documentation-print "This option is not yet implemented"))
      (spectral.hemispherical.reflectance
                (load-tech str
                        'spectral.hemispherical.reflectance.techniques
                        'hemispherical.reflectance.technique.rules
                         "HRTR."
                        (unit
             'windowpane-selected.techniques-of-6.generate.techniques.3)))
    (proportion.ground.cover.single.wavelength
              (load-tech str
                'proportion.ground.cover.single.wavelength.techniques
                'proportion.ground.cover.single.wavelength.rules
                "PGCSWR."
                (unit
                'windowpane-selected.techniques-of-portion.ground.cover.5)))
    (proportion.ground.cover.multiple.wavelength
              (load-tech str
                'proportion.ground.cover.multiple.wavelength.techniques
                'proportion.ground.cover.multiple.wavelength.rules
                "PGCMWR."
                (unit
          'windowpane-selected.mw.techniques-of-portion.ground.cover.5)))
    (view.angle.extension
              (load-tech str
                        'view.angle.extension.techniques
                        'view.angle.extension.rules
                        "VAER."
                        (unit
            'windowpane-selected.techniques-of-view.angle.extension.6))))))

(defun load-tech (str tech-class rule-class prefix window)
"Creates the units to store the data for the technique and rule in VEG. Loads
the data from the file."
    (let ((new-tech (read-file str)))
        (if (unit.exists.p new-tech)       ; Technique already read in
           (dotimes (n 9) (read-file str)) ; Read past this technique
           (let ((new-tech-unit          ; Read in technique
                  (create.unit new-tech 'veg nil tech-class))
                 (new-rule-unit
                  (create.unit (gentemp prefix) 'veg nil rule-class)))
             (put.value new-tech-unit 'description (read-file str))
             (put.value new-tech-unit 'error.message (read-file str))
             (put.value new-tech-unit 'technique.method (read-file str))
             (put.value new-tech-unit 'interpolate.extrapolate?
                  (if (eq (read-file str) 'YES)
                      t
                      nil))
             (cond ((eq (read-file str) 'YES)
                    (put.value new-tech-unit 'coeffs.p t)
                    (put.value new-tech-unit 'coeff.method (read-file str)))
                   (t (put.value new-tech-unit 'coeffs.p nil)
                      (read-file str))) ;Ignore coeffs method from file
             (put.value new-tech-unit 'ok.to.use (read-file str))
             (put.value new-tech-unit 'weight (read-file str))
             (put.value new-rule-unit 'external.form (read-file str))
             (slot-image-toggle-enable window) ; Update the user pick
                                      ; technique interface
             (slot-image-toggle-enable window)))))
```

```
;;;-----------------------------------------------------------------------
;;; Functions required to define a new technique
;;;-----------------------------------------------------------------------

(defun define-new-techniques ()
"Opens and initializes the interface to guide the user through entering the
required data for a new technique."
  (unitmsg 'viewport-add.techniques.1 'close-panel!)
  (unitmsg 'viewport-add.techniques.2 'open-panel!)
  (remove.all.values 'add.techniques 'tech.name)
  (initialize-add-techniques)
  (put.value 'add.techniques 'new.tech.options 'enter.technique)
  (store-previously-defined-tech-names)
  (my-documentation-print "Enter the technique name"))

(defun close-new-tech-windows ()
"Close the subwindows of the define new technique interface."
  (unitmsg 'windowpane-coeffs.p-of-add.techniques.4 'close!)
  (unitmsg 'windowpane-interpolate.extrapolate?-of-add.techniques.10 'close!)
  (unitmsg 'windowpane-weight-of-add.techniques.4 'close!)
  (unitmsg 'windowpane-goals-of-add.techniques.2 'close!)
  (unitmsg 'windowpane-enter.description-of-add.techniques.4 'close!)
  (unitmsg 'windowpane-enter.error.message-of-add.techniques.3 'close!)
  (unitmsg 'windowpane-enter.functions-of-add.techniques.6 'close!)
  (unitmsg 'windowpane-compile.functions-of-add.techniques.1 'close!)
  (unitmsg 'windowpane-enter.rule-of-add.techniques.7 'close!)
  (unitmsg 'windowpane-parse.rule-of-add.techniques.2 'close!))

(defun store-previously-defined-tech-names ()
"If any techniques have been defined, calls the function to collect the
technique names."
   (when (probe-file "new-tech-data")
     (with-open-file (str "new-tech-data" :direction :input)
         (remove.all.values 'add.techniques 'previous.techs)
         (read-tech-names-from-file str))))

(defun read-tech-names-from-file (str)
"Saves the names of the techniques that have already been defined in the slot
PREVIOUS.TECHS of the unit ADD.TECHNIQUES."
  (do ((data (read-file str)(read-file str)))
      ((null data) nil)                    ; End of file
    (add.value 'add.techniques 'previous.techs (read-file str))
    (dotimes (n 9)(read-file str))))

(defun already-defined (tech-name)
"Returns t if a technique of the same name has already been defined and nil
otherwise."
  (or (unit.exists.p tech-name)
     (member tech-name (get.values 'add.techniques 'previous.techs)
              :test #'equal)))
```

```
(defun open-new-tech-windows ()
"Open the subwindows of the define new technique interace. This function is
called after the new technique has been named."
  (unitmsg 'windowpane-coeffs.p-of-add.techniques.4 'open!)
  (unitmsg 'windowpane-interpolate.extrapolate?-of-add.techniques.10 'open!)
  (unitmsg 'windowpane-weight-of-add.techniques.4 'open!)
  (unitmsg 'windowpane-goals-of-add.techniques.2 'open!)
  (unitmsg 'windowpane-enter.description-of-add.techniques.4 'open!)
  (unitmsg 'windowpane-enter.error.message-of-add.techniques.3 'open!)
  (unitmsg 'windowpane-enter.functions-of-add.techniques.6 'open!)
  (unitmsg 'windowpane-compile.functions-of-add.techniques.1 'open!)
  (unitmsg 'windowpane-enter.rule-of-add.techniques.7 'open!)
  (unitmsg 'windowpane-parse.rule-of-add.techniques.2 'open!))

(defun initialize-add-techniques()
"Initializes the slots in the unit ADD.TECHNIQUES, ready for entering the new
technique. If they exist, deletes the files that have temporarily held the
functions and the selection rule for a previously entered new technique."
  (remove.all.values 'add.techniques 'technique.method)
  (remove.all.values 'add.techniques 'coeffs.method)
  (remove.all.values 'add.techniques 'ok.to.use)
  (remove.all.values 'add.techniques 'goals)
  (put.value 'add.techniques 'description "")
  (put.value 'add.techniques 'error.message "")
  (put.value 'add.techniques 'interpolate.extrapolate? 'no)
  (put.value 'add.techniques 'coeffs.p 'yes)
  (put.value 'add.techniques 'weight 1)
  (put.value 'add.techniques 'initialized.function t)
  (put.value 'add.techniques 'initialized.rule t)
  (put.value 'add.techniques 'rule.parsed nil)
  (when (probe-file "temp.lisp")
   (lcl::shell "rm temp.lisp")) ; Remove temporary function file
  (when (probe-file "temp.lsbin")
   (lcl::shell "rm temp.sbin")) ; Remove temporary compiled function file
  (when (probe-file "temp-rule")
   (lcl::shell "rm temp-rule"))) ; Remove temporary rule file

(defun enter-description (self)
"Prompts the user to enter the description of a new technique into a file.
Then reads it from the file into the description slot of ADD.TECHNIQUES."
 (declare (ignore self))
 (my-documentation-print "Complete the description of the technique.
Save the file and exit the editor to save the description.")
 (sleep 1)
 (when (equal (get.value 'add.techniques 'description) "")
  (with-open-file (str "temp-desc" :direction :output :if-exists :supersede)
   (princ (format () "Technique ~A " (get.value 'add.techniques 'tech.name))
          str)))
 (lcl::shell "textedit temp-desc")
 (put.value 'add.techniques 'description
   (with-open-file (str "temp-desc" :direction :input)
       (let ((desc ""))
         (do ((dat (read-file str)(read-file str)))
            ((null dat) desc)
           (setf desc (format () "~A ~A" desc dat))))))
 (clear-prompt))

(defun enter-error-message (self)
"Prompts the user to enter the error message of a new technique into a file.
Then reads it from the file into the error.merssage slot of ADD.TECHNIQUES."
 (declare (ignore self))
 (my-documentation-print "Complete the description of the error message.
Save the file and exit the editor to save the error message.")
 (sleep 1)
 (when (equal (get.value 'add.techniques 'error.message) "")
  (with-open-file (str "temp-error" :direction :output :if-exists :supersede)
   (princ (format () "Technique ~A " (get.value 'add.techniques 'tech.name))
           str)))
 (lcl::shell "textedit temp-error")
 (put.value 'add.techniques 'error.message
   (with-open-file (str "temp-error" :direction :input)
       (let ((desc ""))
         (do ((dat (read-file str)(read-file str)))
            ((null dat) desc)
           (setf desc (format () "~A ~A" desc dat))))))
 (clear-prompt))

(defun enter-functions(self)
"Enter the functions required by the new technique."
 (declare (ignore self))
 (let ((tech-name (get.value 'add.techniques 'tech.name)))
  (cond (tech-name
          (when (get.value 'add.techniques 'initialized.function)
            (with-open-file (str "temp.lisp" :direction :output
                                 :if-exists :supersede)
             (princ (format ()
";;; Templates for adding technique ~A

(in-package 'kee)

;;; Replace the body of this example function with the correct function for
;;; the new technique. The function checks whether the technique is suitable
;;; for the sample. Here the sample is the wavelength level unit. The
;;; function should return t if the technique is suitable for the sample and
;;; nil otherwise.
(defun ~A (sample)
\"Checks the suitability of the function ~A for the sample.\"
 (= (get.value sample 'number.view.angles) 1))
;;; Replace the body of this example function with the technique function for
;;; the new technique.  In this function the arguments are the sample unit at
;;; the wavelength level and the vector of coefficients, if any.  The function
;;; should return a number which is the result of applying the technique to the
;;; sample.
(defun ~A (thisunit coeffs)
\"Applies the function ~A to the sample.\"
  (declare (ignore coeffs))  ;Remove this line if technique uses coefficients
  (third (first (get.value thisunit 'reflectance.data))))

;;; If the technique uses coefficients, replace the body of this example
;;; function with the coefficient function.  Otherwise delete the template.
;;; In this function the argument is the list of restricted historical data
;;; units to be used for calculating the coefficients.  The function should
;;; return the vector of coefficients of the correct length for the technique.
(defun ~A (data)
\"Calculates the coefficients for the technique ~A.\"
  (declare (ignore data))    ;Replace these lines with the new function body
  nil)~%"

                    tech-name
                     (get.value 'add.techniques 'ok.to.use) tech-name
                    (get.value 'add.techniques 'technique.method) tech-name
                    (get.value 'add.techniques 'coeffs.method) tech-name)
                    str))
                (put.value 'add.techniques 'initialized.function nil))
          (my-documentation-print
"Edit the file.  Then save the file and exit the editor.")
          (sleep 1)
          (lcl::shell "textedit temp.lisp")
          (clear-prompt))
        (t (my-documentation-print
                "Enter technique name before entering the functions")))))

(defun compile-functions (self)
"Compiles the functions for the new technique."
  (declare (ignore self))
  (when (not (probe-file "temp.lisp"))
    (my-documentation-print
     "Functions not found - enter them before compiling")
    (return-from compile-functions nil))
  (my-documentation-print "Compiling the new functions")
  (when (probe-file "temp.sbin")
    (lcl::shell "rm temp.sbin"))
  (compile-file "temp.lisp" :messages nil :file-messages nil)
  (my-documentation-print "Finished compilation"))

(defun compiled-ok ()
"Returns t if the function complied correctly and nil otherwise."
   (when (probe-file "temp.sbin")
      (with-open-file (str "temp.sbin" :direction :input)
           (let ((len (file-length str)))
             (and (numberp len)
                   (> len 0))))))
```

```
(defun enter-rule (self)
"Sets up a file to temporarily store the new rule.  Prompts the user to enter
the rule.  Attempts to parse it.  If parsing fails, prompts the user to correct
the rule until it parses correctly."
 (declare (ignore self))
 (put.value 'add.techniques 'rule.parsed nil)
 (let ((tech-name (get.value 'add.techniques 'tech.name)))
  (cond (tech-name
         (when (get.value 'add.techniques 'initialized.rule)
          (with-open-file (str1 "temp-rule" :direction :output
                                 :if-exists :supersede)
           (princ (format ()
";;; Template for rule for selecting technique ~A

;;; Edit the lefthand side of this example rule to create the required rule
(IF (THE CURRENT.SAMPLE.WAVELENGTHS OF
                        ESTIMATE.HEMISPHERICAL.REFLECTANCE IS ?X)
      (THE NUMBER.VIEW.ANGLES OF ?X IS 1)
     THEN (LISP (ADD.VALUE ?X (QUOTE TECHNIQUES)
                        (QUOTE ~A))))"
          tech-name tech-name)
         str1))
        (my-documentation-print
"Edit the file.  Then save the file and exit the editor.")
        (put.value 'add.techniques 'initialized.rule nil))
      (sleep 1)
      (lcl::shell "textedit temp-rule"))
     (t (my-documentation-print
         "Enter technique name before entering the rule")))))

(defun parse-rule (self)
"Returns t if the rule parses correctly and nil otherwise.  Note that parsing
is not a complete test of correctness for a rule."
 (declare (ignore self))
 (my-documentation-print "Parsing rule")
 (when (not (probe-file "temp-rule"))
  (my-documentation-print
   "Rule not found - enter it before parsing")
  (put.value 'add.techniques 'rule.parsed nil)
  (return-from parse-rule nil))
 (with-open-file (str1 "temp-rule" :direction :input)
  (cond  ((not (parens-ok str1))
          (my-documentation-print
           "Rule has unequal number of left and right parens - edit again")
          (put.value 'add.techniques 'rule.parsed nil))
         ((test-rule-parses str1)
          (my-documentation-print "Rule parsed OK")
          (put.value 'add.techniques 'rule.parsed t))
         (t
          (my-documentation-print
           "Rule does not parse correctly - edit again.")
          (put.value 'add.techniques 'rule.parsed nil)))))
```

```
(defun test-rule-parses (str1)
"Sets up a temporary unit to hold the new rule.  Attempts to parse it.  Deletes
the temporary rule unit.  Returns t if the rule parsed OK and nil otherwise."
  (let ((new-rule-unit
          (create.unit 'TEMP 'veg nil 'vegrules)))
    (put.value new-rule-unit 'external.form (read-file str1))
    (prog2 (unitmsg new-rule-unit 'parse)
         (not (get.value new-rule-unit 'parse.errors))
     (delete.unit new-rule-unit))))

(defun store-data ()
"Stores the data about the new technique in the file."
  (let ((goal (get.value 'add.techniques 'goals))
         (tech-name (get.value 'add.techniques 'tech.name))
         (description (get.value 'add.techniques 'description))
         (error-message (get.value 'add.techniques 'error.message)))
    (cond ((not tech-name)
             (my-documentation-print
              "Technique name not found - data not stored"))
            ((not goal)
             (my-documentation-print
              "Goal not found - data not stored"))
            ((eq goal 'total.hemispherical.reflectance)
             (my-documentation-print
              "Techniques for this goal are not yet implemented - not stored"))
            ((equal description "")
             (my-documentation-print
              "Description not found - data not stored"))
            ((equal error-message "")
             (my-documentation-print
              "Error message not found - data not stored"))
            ((not (compiled-ok))
             (my-documentation-print
              "Functions not correctly compiled - data not stored"))
            ((not (probe-file "temp-rule"))
             (my-documentation-print "Rule not found - data not stored"))
            ((not (get.value 'add.techniques 'rule.parsed))
             (my-documentation-print
              "Rule not successfully parsed - data not stored"))
            (t (store-data-on-file goal tech-name description error-message)))))

(defun store-data-on-file (goal tech-name description error-message)
"Stores the technique data in the file new-tech-data.  Calls the function to
store the technique functions."
  (my-documentation-print "Saving the new technique")
  (with-open-file (str "new-tech-data" :direction :output :if-exists :append
                        :if-does-not-exist :create)
    (princ (format ()
                  "~A~%~A~%\"~A\"~%\"~A\"~%~A~%~A~%~A~%~A~%~A~%~A~%"
                  goal
                  tech-name
                  description
                  error-message
                  (get.value 'add.techniques 'technique.method)
```

```
                    (get.value 'add.techniques 'interpolate.extrapolate?)
                    (get.value 'add.techniques 'coeffs.p)
                    (get.value 'add.techniques 'coeffs.method)
                    (get.value 'add.techniques 'ok.to.use)
                    (get.value 'add.techniques 'weight))
           str)
   (add.value 'add.techniques 'previous.techs tech-name)
   (store-functions str)))

(defun store-functions (str)
"Adds the function for the new technique to the file new-tech.lisp.  Compiles
the file.  Adds the new rule to the file new-tech-data."
 (lcl::shell "cat new-tech.lisp temp.lisp > temp1")
 (lcl::shell "mv temp1 new-tech.lisp")
 (compile-file "new-tech.lisp" :messages nil :file-messages nil :warnings nil)
 (with-open-file (str1 "temp-rule" :direction :input)
   (princ (read-file str1) str) ; Read the rule from the temporary file and
   (terpri str))              ; store it in the file new-tech-data
 (clear-prompt)
 (remove.all.values 'add.techniques 'tech.name)
 (initialize-add-techniques)
 (close-new-tech-windows)
 (my-documentation-print "Enter the name of the new technique")
 (put.value 'add.techniques 'new.tech.options 'enter.technique))

(defun abandon-data ()
"Initializes the values in the add.techniques unit.  Deletes any recently
entered but not yet stored functions or rules.""
stored."
 (remove.all.values 'add.techniques 'tech.name)
 (initialize-add-techniques)
 (close-new-tech-windows)
 (my-documentation-print "Enter the name of the new technique")
 (put.value 'add.techniques 'new.tech.options 'enter.technique))

(defun read-char-file (str)
"Reads a charcter from a file.  Returns the character read, or nil if the end
of the file has been reached."
  (flet ((eof-p (obj)
            (eq obj '*eof*)))
    (let ((obj (read-char str () '*eof* ())))
      (if (eof-p obj)
          nil
          obj))))

(defun parens-ok (str)
"Returns t if the file contains the same number of left and right parens and
nil otherwise."
 (let ((left 0)
       (right 0))
   (do ((char (read-char-file str)(read-char-file str)))
       ((null char) (if (zerop (- left right))
                        t
                        nil))
    (case char
        (#\) (incf right))
        (#\( (incf left))))))

(defun purge-previously-defined-techniques ()
"Opens the required subwindows ready to remove all previously defined
new techniques from the files."
  (remove.all.values 'add.techniques 'yes.no)
  (put.value 'add.techniques 'message
"Are you sure you want to permanently delete all previously defined new techniques?")
  (unitmsg 'windowpane-message-of-add.techniques.2 'open-panel!)
  (unitmsg 'windowpane-yes.no-of-add.techniques.3 'open-panel!))

(defun purge-techniques()
"Removes all previously defined new techniques from the files so they are no
longer available to be added to VEG."
  (when (probe-file "new-tech-data")
   (lcl::shell "rm new-tech-data"))
  (with-open-file (str "new-tech.lisp" :direction :output
                               :if-exists :supersede)
   (princ (format ()
";;; new-tech.lisp
;;;
;;; Holds Functions Required by Newly Defined Techniques
;;; Functions are entered through the Define New Technique Interface

") str)))
```

NASA
National Aeronautics and Space Administration

Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle An Expert System Shell for Inferring Vegetation Characteristics - Interface for the Addition of Techniques (Task H)		5. Report Date April 1993
		6. Performing Organization Code
7. Author(s) P. Ann Harrison		8. Performing Organization Report No. C931021-U-2R07
		10. Work Unit No. 462-61-14
9. Performing Organization Name and Address JJM Systems, Inc. One Ivybrook Blvd., Suite 190 Ivyland, PA 18974		11. Contract or Grant No. NAS5-30127
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, DC 20546-0001 NASA/Goddard Space Flight Center Greenbelt, MD 20771		13. Type of Report and Period Covered Task Report for Task H March - April 1993
		14. Sponsoring Agency Code

15. Supplementary Notes

The Lisp and KEE code for this work is available on a Sun Cartridge Tape.

16. Abstract

VEG is an expert system that infers vegetation characteristics from reflectance data. VEG provides the scientist with several different analysis techniques which are stored in the knowledge base. When VEG is run, rules assist the scientist in selecting the best of the available techniques to apply to the sample of cover type data being studied. In the previous version of VEG, the addition of a new technique was a complex process. A new interface that enables the scientist to add techniques to VEG without assistance from the developer has been designed and implemented. It guides the scientist through entering the data, Common Lisp functions and the rule required by the new technique. Once the technique has been defined, adding it to VEG requires only the selection of the appropriate menu option. The Add Techniques System was tested using both valid and invalid data. The tests were designed to test the typical range of user behavior. They confirmed that the interface was operating correctly.

17. Key Words (Suggested by Author(s)) EXPERT SYSTEM, ARTIFICIAL INTELLIGENCE, REMOTE SENSING	18. Distribution Statement UNCLASSIFIED - UNLIMITED

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of pages	22. Price
UNCLASSIFIED	UNCLASSIFIED	33	

NASA FORM 1626 OCT 86

For sale by the National Technical Information Service, Springfield, VA 22161-2171