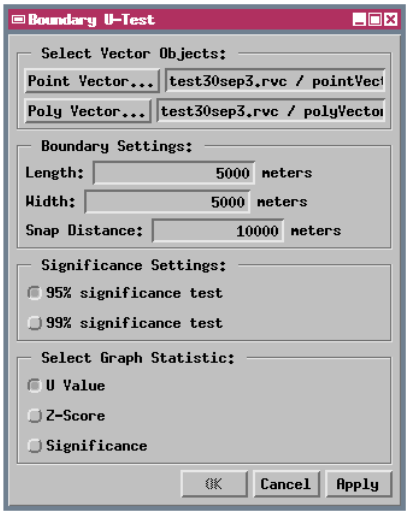
Sample SML Tool Script

Mosquito Habitat Statistics (U-Test)

MicroImages has collaborated with Dr. (Col.) Tony Sweeney of the Institute for the Biotechnology of Infectious Diseases at the University of Technology, Sydney, Australia, to develop an SML tool script to help in determining the environmental variables that control the distribution of different malaria-bearing mosquito species. In northern Australia, a particular species of malaria-bearing mosquito is limited in range to a coastal strip no more than 5 kilometers wide. The purpose of the tool script is to perform an analysis of test areas straddling the inland boundary of this strip to determine statistically which environmental parameters define the actual range of this species.

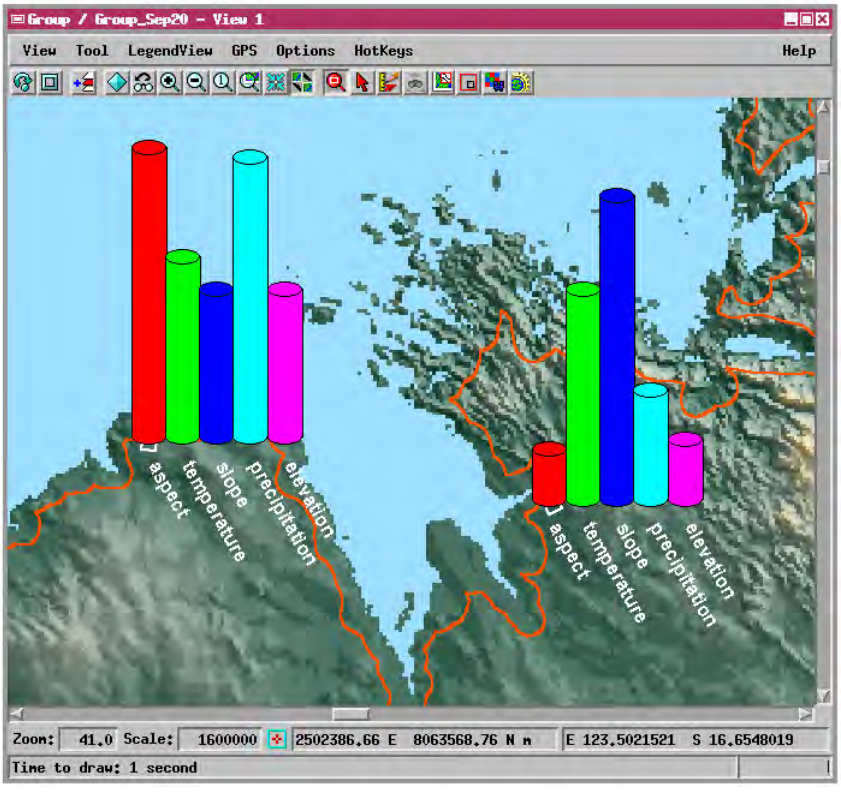
The uTest script (excerpts of which are shown on the reverse side of this color plate) is designed to work with a display group containing raster layers depicting the spatial variation of various environmental variables (elevation, slope, rainfall, and so on) and a vector layer representing the inland boundary of this malarial mosquito habitat. The script was created as a tool script so that it can provide interactive selection of test areas in the View, perform the statistical analysis, and present the results graphically in the View (using a vector object styled with a CartoScript generated automatically by the tool script). Run statistics for each test area are also saved in a record in a database table created by the script.

To create rectangular test areas across the habitat boundary, the user simply left-clicks in the View near each desired boundary location. For each environmental variable raster, the script tabulates the raster cell values on either side of the boundary within each test area. The script tests for significant differences across the boundary by applying the Mann-Whitney U-Test to the paired sets of values. A dialog window created and opened by the script provides controls to set the dimensions of the test area and the level of statistical significance to be used in the analysis, and to select the statistical measure to be graphed in the View.



Control window created by the tool script. The user can set the dimensions of each test area, the level of statistical significance to be used in the analysis, and which statistic is graphed in the View.

Two test sites (white rectangles) straddling the inland limit of the malaria-bearing mosquito species along the coast of northern Australia. Sites were located by left-clicking with the mouse near the orange boundary line in the View. For each mouse click the tool script finds the closest point on the boundary line and generates a test rectangle across the boundary with dimensions specified by the user in the Boundary U-Test window.



Graphical result of the U-Test for the two test sites shown at left. The bar graphs show relative U values for the five parameter rasters in the group. The graphs of the run are created by a CartoScript that is automatically generated by the tool script using the statistical results.

Many sample scripts have been prepared to illustrate how you might use the features of the TNT products' scripting language for scripts and queries. These scripts can be downloaded from www.microimages.com/freestuf/scripts.htm.

Script Excerpt for Malaria Statistics Test (uTest.sml)

Calculate Z based on the U value. For large samples the normal approximation *z = (U - mU)/oU* can be used where *mU* and *oU* are the mean and standard deviation of U as given by:
mU = n1n2/2 and *oU* = sqrt(n1n2(n1+n2+1)/12).

```
func calculateZScore(numeric U, numeric n1, numeric n2) {
   local numeric mU = n1*n2/2;
   local numeric oU = sqrt(n1*n2*(n1+n2+1)/12);

   return abs((U - mU)/oU);
}
```

Calculate U value for designated raster (rast). The two sample sets are designated based on a cell's inclusion in a region.

U = n1*n2 + n1(n1+1)/2 - R1
Where n1 and n2 are the two sample sizes, and R1 is the sum of the ranks in sample 1. Sample 1 is taken to be the smaller of the two groups.

```
func calculateUValue() {
   n1=0; n2=0;
   local numeric R1=0, R2=0;
   local numeric U=0;

   foreach rast in regionIn  {
      if(!IsNull(rast)) n1++;
   }
   foreach rast in regionOut {
      if(!IsNull(rast)) n2++;
   }
```

get sample sizes for array declarations

```
   local numeric size = n1 + n2;
   local array numeric mergedArray[size];
   local array numeric rank[size];
   local array numeric bitset[size];
   local numeric inVal = 1, outVal = 0;
```

declare arrays for samples

```
   local class REGION2D myRegIn, myRegOut;
   if (n1 <= n2) {
      myRegIn = CopyRegion(regionIn);
      myRegOut = CopyRegion(regionOut);
   }
   else {
      myRegIn = CopyRegion(regionOut);
      myRegOut = CopyRegion(regionIn);
      local numeric tmpN = n2;
      n2 = n1;
      n1 = tmpN;
   }
```

copy regions to temp regions and flip if necessary

```
   local numeric i=1;
   foreach rast in myRegIn {
      if(!IsNull(rast)) {
         mergedArray[i] = rast;
         rank[i] = i;
         bitset[i] = inVal;
         i++;
      }
   }
   i=1;
   foreach rast in myRegOut {
      if(!IsNull(rast)) {
         mergedArray[i+n1] = rast;
         rank[i+n1] = i+n1;
         bitset[i+n1] = outVal;
         i++;
      }
   }
```

copy samples to arrays

```
   local numeric h=1, first=1, last=n1+n2;

   while ((h * 3 + 1) < last-1) {
      h = 3 * h + 1;
   }

   while (h > 0) {
      for i = h-1 to last {
         local numeric key = mergedArray[i];
         local numeric bkey = bitset[i];
         local numeric j = i;

         while (j>=h && mergedArray[j-h] > key) {
            mergedArray[j] = mergedArray[j-h];
            bitset[j] = bitset[j-h];
            j = j - h;
         }
         mergedArray[j] = key;
         bitset[j] = bkey;
      }
      h = floor(h/3);
   }
```

rank values in mergedArray - do a shellsort

do the sort

for each of the h sets of elements

```
   local numeric next;
   for i=1 to last-1 {
      next = i;
      local numeric sum=rank[next], count=1;
      while(next<last && mergedArray[next] == mergedArray[next+1]) {
         sum = sum + rank[next+1];
         count++;
         next++;
      }
      local numeric j;
      for j=i to next {
         rank[j] = sum / count;
      }
      i = next;
   }
   for i=1 to last
   {
      if (bitset[i]==inVal) R1 = R1 + rank[i];
      if (bitset[i]==outVal) R2 = R2 + rank[i];
   }

   local numeric U1=0, U2=0;
   U1 = n1*n2 + n1*(n1+1)/2 - R1;
   U2 = n1*n2 + n2*(n2+1)/2 - R2;
   U = min(U1,U2);

   return U;
}
```

generate ranks

look for ties in set and resolve all at once

use the average rank in case of tie

calculate the sum of ranks for the smaller sample

calculate the u value

Perform Mann-Whitney U-test on all rasters in the active group.

```
func doMannWhitneyUTest(numeric pointNum, numeric latestRecord) {
   local class GRE_LAYER currentRaster = activegroup.FirstLayer;
   local numeric uValue = 0;

   while (currentRaster != 0) {
      if (isRasterLayer(currentRaster)) {
         DispGetRasterFromLayer(rast, currentRaster);

         uValue = calculateUValue();
```

loop over each raster and perform calculations

calculate the z-score

```
         local numeric zScore = calculateZScore(uValue, n1, n2);
```

compute whether test result is statistically significant

```
         local numeric signif = computeSignificance(zScore, uValue);
```

write records to the database

```
         writeStatisticsToTable(latestRecord, rast.$INFO.Name, uValue, zScore,
            signif);
         PopupMessage("wrote record for " + rast.$INFO.Name);
      }
      currentRaster = currentRaster.NextLayer;
   }
}
```