Environmental Analysis through integration of Geographical Information and Machine Vision systems

by

Paul D. Kelly, M.Eng.

A thesis presented on application for the degree of

DOCTOR OF PHILOSOPHY

Faculty of Engineering

The Queen's University of Belfast

School of Electrical & Electronic Engineering,

May 2004.

Appendix A

Algorithms for Image Pre-processing

A.1 Square Pixel Correction

sqcorrsimple.m

```
function output=sqcorr(a)
%
% function output=sqcorr(a)
% Converts an image from 720x576 rectangular pixels to 768x576
% square pixels using cropping and horizontal interpolation
%
sizeofa=size(a);
size2=size(sizeofa);
if (size2(2) > 2)
   colours=sizeofa(3);
else
   colours = 1;
end


output=zeros(576,768,colours);
for row=1:576
  for column=1:768
     interp = column*54/59 + (504/59);
     if (floor(interp) == interp)
       lowscale = 1;
       highscale = 0;
     elseif (ceil(interp) == interp)
       lowscale = 0;
       highscale = 1;
     else
       lowscale = 1 - (interp - floor(interp));
       highscale = 1 - (ceil(interp) - interp);
     end
     for colour=1:colours
       output(row,column,colour) = (double(a(row,floor(interp),colour)).*lowscale...
           + double(a(row,ceil(interp),colour)).*highscale);
     end
  end
end

output=uint8(round(output));
```

A.2 Distortion Measurement

These MATLAB scripts and functions perform the distortion correction measurements and calculations discussed in Chapter 2. docanonin.m is the top-level script and the others are called from there.

docanonin.m

```
a=imread('canon-mv500-zoomin.bmp');
% Correct to give square pixels
sqcorr=sqcorrsimple(a);
flat=flatten(sqcorr);
% Theshold value will depend on image
circlecentroid=pkfindcentres(flat,0.5);
pkstart;
save canonindistlocs distlocs;
```

flatten.m

```
function [flattened]=flatten(sqcorr)
% function [flattened]=flatten(sqcorr)
% Perform illumination flattening on an RGB image
k=imread('filter.bmp');
k=double(k)./sum(sum(k));
bw=rgb2gray(sqcorr);
smoothed=conv2(bw,k);
o=ones(size(bw));
smoothedones=conv2(o,k);
corrected=smoothed./smoothedones;
illum=corrected(63:(576+62),63:(768+62));
flattened(:,:,1)=double(sqcorr(:,:,1))./illum;
flattened(:,:,2)=double(sqcorr(:,:,2))./illum;
flattened(:,:,3)=double(sqcorr(:,:,3))./illum;
flattened=uint8(round(flattened.*128));
```

pkfindcentres.m

```
function [circlecentroid] = findcentres(sqcorr, threshvalue)
%function [circlecentroid] = findcentres(sqcorr)
%
% findcentres.m
% MATLAB function to find centres of white circles
%

[rows columns colours]=size(sqcorr);
if colours == 1
   grey = sqcorr;
else
   grey = rgb2gray(sqcorr);
end

% Pixels w/ grey level above ?? treated as part of circles
threshold = im2bw(grey, threshvalue);

%%%%%%%%%%%%%%%%%%%%%%%%%
figure,imshow(threshold);
hold on;
%%%%%%%%%%%%%%%%%%%%%%%%%

% bw label searches for connected blobs and gives all the pixels in each
% blob the same number.
labelledcircles = bwlabel(threshold);

% Each circle is numbered consecutively so
numofcircles = max(max(labelledcircles));

circlecent=zeros(numofcircles,3);
for circlenum = 1:numofcircles
   % Get the row and column values of all the pixels in this patch
   clear pixelrows pixelcolumns;
```

```
    [pixelrows, pixelcolumns] = find(labelledcircles == circlenum);
    numofpixels=size(pixelrows);
    % Calculate the centroid of the circle by summing the x and y co-ordinates of all the
    % pixels and dividing by the total number of pixels
    circlecent(circlenum,1) = sum(pixelrows) ./ numofpixels(1);
    circlecent(circlenum,2) = sum(pixelcolumns) ./ numofpixels(1);
    circlecent(circlenum,3) = numofpixels(1);
end
avgpixels = median(circlecent(:,3));
minsize = 0.8 .* avgpixels;
maxsize = 1.2 .* avgpixels;
count = 0;
for circlenum = 1:numofcircles
    if ((circlecent(circlenum,3) > minsize) & (circlecent(circlenum,3) < maxsize))
        count = count + 1;
        circlecentroid(count,:) = circlecent(circlenum,1:2);
    end
end

% Make the original image figure current
%figure(imagefigure)
%figure,imshow(sqcorr)
%hold on
% Plot (x,y) pixel locations with a blue x on top of the image
plot(circlecentroid(:,2), circlecentroid(:,1), 'bx');
```

pkstart.m

```
[rowlines columnlines]=pksortcircles(circlecentroid);
% Approximation from analysing curvature of lines
[distxc distyc]=pkcentreofdist(rowlines, columnlines)
corrxc=distxc; %384.5;
corryc=distyc; %288.5;

% Next make a matrix with all rows and columns the same length
clear maxsamples modrows modcolumns
sizerowlines=size(rowlines);
count = 0;
for sampledrow = 1:sizerowlines(1)
  numsamples=max(find(rowlines(sampledrow,:,1)));
  % Only use fairly complete rows
  if numsamples > (0.9*sizerowlines(2))
     count = count + 1;
     maxsamples(count)=numsamples;
  end
end
absnumsamples=min(maxsamples);
count = 0;
for sampledrow = 1:sizerowlines(1)
  numsamples=max(find(rowlines(sampledrow,:,1)));
  % Only use fairly complete rows
  if numsamples > (0.9*sizerowlines(2))
     count = count + 1;
     modrows(count,:,1)=rowlines(sampledrow,1:absnumsamples,2);
     modrows(count,:,2)=rowlines(sampledrow,1:absnumsamples,1);
  end
end

clear maxsamples
sizecolumnlines=size(columnlines);
count = 0;
for sampledcolumn = 1:sizecolumnlines(2)
  numsamples=max(find(columnlines(:,sampledcolumn,1)));
  % Only use fairly complete rows
  if numsamples > (0.9*sizecolumnlines(1))
     count = count + 1;
     maxsamples(count)=numsamples;
  end
end
absnumsamples=min(maxsamples);
count = 0;
for sampledcolumn = 1:sizecolumnlines(2)
  numsamples=max(find(columnlines(:,sampledcolumn,1)));
  % Only use fairly complete rows
```

```
  if numsamples > (0.9*sizecolumnlines(1))
     count = count + 1;
     modcolumns(count,:,1)=columnlines(1:absnumsamples,sampledcolumn,2);
     modcolumns(count,:,2)=columnlines(1:absnumsamples,sampledcolumn,1);
  end
end



%5th order seems best
res0=[0 0 0 0 1 0];

%options=optimset('GradObj','off', 'Display', 'iter', 'MaxFunEvals', 100000, 'MaxIter', 1000);
options=optimset('Display', 'iter');
% rows
%[rowres rowfval]=fminunc('pkcalcerronly',res0,options,modrows,distxc,...
%distyc,corrxc,corryc)
[rowres rowfval]=fmincon('pkcalcerronly',res0,[],[],[],[],(res0-1),...
(res0+1),[],[],modrows,distxc,distyc,corrxc,corryc)
% columns
%[columnres columnfval]=fminunc('pkcalcerronly',res0,options,modcolumns,...
%distyc,distxc,corryc,corrxc)
[columnres columnfval]=fmincon('pkcalcerronly',res0,[],[],[],[],(res0-1),...
(res0+1),[],[],modcolumns,distyc,distxc,corryc,corrxc)

res=(rowres+columnres)./2

sizemodrows=size(modrows);
distx=modcolumns(:,:,1);
disty=modcolumns(:,:,2);
distrd=sqrt((distx-distxc).^2 + (disty-distyc).^2);
corrx = corrxc + polyval(res,distrd).*((distx-distxc)./distrd);
corry = corryc + polyval(res,distrd).*((disty-distyc)./distrd);
figure, imshow(sqcorr);
hold on;
plot(distxc,distyc,'ro');
plot(corrx,corry,'rx');

sizemodcolumns=size(modcolumns);
distx=modcolumns(:,:,1);
disty=modcolumns(:,:,2);
distrd=sqrt((distx-distxc).^2 + (disty-distyc).^2);
corrx = corrxc + polyval(res,distrd).*((distx-distxc)./distrd);
corry = corryc + polyval(res,distrd).*((disty-distyc)./distrd);
figure, imshow(sqcorr);
hold on;
plot(distxc,distyc,'ro');
plot(corrx,corry,'rx');

rows=576;
columns=768;

sizeres=size(res);
indices=zeros(rows,columns,2);
eqn=zeros(rows,columns,sizeres(2));

for row=1:rows
   for column=1:columns
      indices(row,column,:)=[column row];
      eqn(row,column,:)=res;
   end
end

corrrd=sqrt((indices(:,:,1)-corrxc).^2+(indices(:,:,2)-corryc).^2);
costheta=(indices(:,:,1)-corrxc)./corrrd;
sintheta=(indices(:,:,2)-corryc)./corrrd;
eqn(:,:,sizeres(2))=eqn(:,:,sizeres(2))-corrrd;
eqnanswer=zeros(rows,columns,sizeres(2)-1);
for row=1:rows
   fprintf('row %d\n',row);
   for column=1:column
      eqnanswer(row,column,:)=roots(eqn(row,column,:));
   end
end
distrd=eqnanswer(:,:,sizeres(2)-1);
```

```
distlocs(:,:,1)=distxc + costheta.*distrd;
distlocs(:,:,2)=distyc + sintheta.*distrd;

%distlocsrect(:,:,2) = distlocs(:,:,2);  % y co-ord the same
%distlocsrect(:,:,1) = distlocs(:,:,1).*(54/59) + (504/59); % adjust x to sq pels
distlocsrect=zeros(rows,columns,12);

for x=1:columns
    fprintf('column %d\n',x);
    for y=1:rows
      xd=distlocs(y,x,1).*(54/59) + (504/59);
      yd=distlocs(y,x,2);

      % If point to be re-sampled falls within original image
      if (xd >= 0.5 & xd < 720.5 & yd >= 0.5 & yd <576.5)
        columnvalue = floor(xd);
        rowvalue = floor(yd);
        columndiff = xd - columnvalue;
        rowdiff = yd - rowvalue;

        % If not in outer 0.5 pixel width of image
        if (xd >= 1 & xd < 720 & yd >= 1 & yd < 576)

          % Bi-linear interpolation algorithm as used in GRASS s.sample program
          % See http://grass.itc.it/gdp/sites/s.sample-tutorial.ps.gz
%           p1 = columndiff * Datac(rowvalue, columnvalue + 1, 1:3) + ...
%                (1 - columndiff) * Datac(rowvalue, columnvalue, 1:3);
%           p2 = columndiff * Datac(rowvalue + 1, columnvalue + 1, 1:3) + ...
%                (1 - columndiff) * Datac(rowvalue + 1, columnvalue, 1:3);
%           p = (1 - rowdiff) * p1 + rowdiff * p2;

          distlocsrect(y,x,:)=...
             [rowvalue columnvalue+1 (1-rowdiff).*columndiff ...
                rowvalue columnvalue (1-rowdiff).*(1-columndiff) ...
             rowvalue+1 columnvalue+1 rowdiff.*columndiff ...
                rowvalue+1 columnvalue rowdiff.*(1-columndiff)];

        elseif (xd >= 1 & xd < 768) % not corner pixels

          if (yd >= 0.5 & yd < 1) % top row
            % linear interpolation two pixels below point
%             p2 = columndiff * Datac(rowvalue + 1, columnvalue + 1, 1:3) + ...
%                  (1 - columndiff) * Datac(rowvalue + 1, columnvalue, 1:3);
%             p=p2;
            distlocsrect(y,x,:)=...
               [rowvalue+1 columnvalue+1 columndiff ...
                  rowvalue+1 columnvalue (1-columndiff) ...
                  1 1 0 ...
                  1 1 0];
          elseif (yd >= 576 & yd <576.5) % bottom row
            % linear interpolation two pixels above point
%             p1 = columndiff * Datac(rowvalue, columnvalue + 1, 1:3) + ...
%                  (1 - columndiff) * Datac(rowvalue, columnvalue, 1:3);
%             p=p1;
            distlocsrect(y,x,:)=...
               [rowvalue columnvalue+1 columndiff ...
                  rowvalue columnvalue (1-columndiff) ...
                  1 1 0 ...
                  1 1 0];
          end

        elseif (yd >= 1 & yd < 576) % not corner pixels

          if (xd >= 0.5 & xd < 1) % leftmost column
            % linear interpolation two pixels to right of point
%             p = (1 - rowdiff) * Datac(rowvalue, columnvalue + 1, 1:3) + ...
%                 rowdiff * Datac(rowvalue + 1, columnvalue + 1, 1:3);
            distlocsrect(y,x,:)=...
               [rowvalue columnvalue+1 (1-rowdiff) ...
                  rowvalue+1 columnvalue+1 rowdiff ...
                  1 1 0 ...
                  1 1 0];
          elseif (xd >=768 & xd <768.5) % rightmost column
            % linear interpolation two pixels to left of point
%             p = (1 - rowdiff) * Datac(rowvalue, columnvalue, 1:3) + ...
```

```
%                 rowdiff * Datac(rowvalue + 1, columnvalue, 1:3);
            distlocsrect(y,x,:)=...
                [rowvalue columnvalue (1-rowdiff) ...
                   rowvalue+1 columnvalue rowdiff ...
                   1 1 0 ...
                   1 1 0];
          end

        elseif (xd >= 0 & xd < 721 & yd >=0 & yd < 577) % corner pixels

          % nearest neighbour interpolation
%          p = Datac (round(yd), round(xd), 1:3);
          distlocsrect(y,x,:)=...
               [round(yd) round(xd) 1 ...
                   1 1 0 ...
                   1 1 0 ...
                   1 1 0];

        end
%         newdata(y,x,:)=p;
      else
        % Point to be re-sampled didn't fall within original image so this
        % pixel in the output image doesn't contain valid data
%         validpels(y,x)=0;
         distlocsrect(y,x,:)=...
               [1 1 0 ...
                   1 1 0 ...
                   1 1 0 ...
                   1 1 0];
      end
    end
  end
```

pksortcircles.m

```
function [rowlines, columnlines]=pksortcircles(circlecentroid)
%function [rowlines columnlines]=pksortcircles(circlecentroid)
%
%Sorts locations of distorted circle centres into two matrices
%where they are ordered by ascending rows and columns respectively
%

numcentroids=size(circlecentroid);
count=numcentroids(1);
usedcentroids=zeros(count,1);

% Sort by ascending rows
sorted=sortrows(circlecentroid);

row = 1;
column = 1;
circlenum = 1;
topcircle=circlenum;
direction = 1;
foundnew=1;
%for circlenum = 1:(count-1)
while sum(find(usedcentroids==0)) ~= 0
%   fprintf('circle %d row %d column %d update %d direction %d\n', ...
%circlenum,row,column,foundnew,direction);
   if foundnew == 1
     rowlines(row, column,:) = sorted(circlenum,:);
     usedcentroids(circlenum) = 1;
   end

   if direction == 1
      next=0;
      bestx=[768.5 0];
      while next < count
         next = next + 1;
         if (next ~= circlenum) & (usedcentroids(next) == 0) & ...
         (sorted(next,2) > sorted(circlenum,2)) & (abs(sorted(next,1)-...
         sorted(circlenum,1)) < 4)
            if sorted(next,2) < bestx(1)
               bestx(1)=sorted(next,2);
```

```
                bestx(2)=next;
            end
        end
    end
else
    next=0;
    bestx=[0.5 0];
    while next < count
        next = next + 1;
        if (next ~= circlenum) & (usedcentroids(next) == 0) & (sorted(next,2) < ...
        sorted(circlenum,2)) & (abs(sorted(next,1)-sorted(circlenum,1)) < 4)
            if sorted(next,2) > bestx(1)
                bestx(1)=sorted(next,2);
                bestx(2)=next;
            end
        end
    end
end

% Is the next point in a new row?
if bestx(2) == 0
    if direction == -1
        % Finished on this row so go on to next
        row = row + 1;
        column = 1;
        foundnew = 1;
        next = 1;
        while (next < count) & (usedcentroids(next) == 1)
            next = next + 1;
        end
        topcircle = next;
        circlenum = next;
        direction = 1;
    else
        % Back to top and search in other direction
        foundnew = 0;
        circlenum = topcircle;
        direction = -1;
    end
else
    circlenum = bestx(2);
    column = column + 1;
    foundnew = 1;
end

end
rowlines(row, column, :) = sorted(circlenum,:);


% Reverse order so we can sort by ascending columns
%revcirclecentroid = zeros(size(circlecentroid));
%revcirclecentroid(:,1) = circlecentroid(:, 2);
%revcirclecentroid(:,2) = circlecentroid(:, 1);
sorted=sortrows(circlecentroid,2);

usedcentroids=zeros(count,1);

row = 1;
column = 1;
circlenum = 1;
topcircle=circlenum;
direction = 1;
foundnew=1;
%for circlenum = 1:(count-1)
while sum(find(usedcentroids==0)) ~= 0
%    fprintf('circle %d row %d column %d update %d direction %d\n', ...
%circlenum,row,column,foundnew,direction);
    if foundnew == 1
      columnlines(row, column,:) = sorted(circlenum,:);
      usedcentroids(circlenum) = 1;
    end

    if direction == 1
        next=0;
        bestx=[576.5 0];
```

```
        while next < count
            next = next + 1;
            if (next ~= circlenum) & (usedcentroids(next) == 0) & (sorted(next,1) > ...
            sorted(circlenum,1)) & (abs(sorted(next,2)-sorted(circlenum,2)) < 4)
                if sorted(next,1) < bestx(1)
                    bestx(1)=sorted(next,1);
                    bestx(2)=next;
                end
            end
        end
    else
        next=0;
        bestx=[0.5 0];
        while next < count
            next = next + 1;
            if (next ~= circlenum) & (usedcentroids(next) == 0) & (sorted(next,1) < ...
            sorted(circlenum,1)) & (abs(sorted(next,2)-sorted(circlenum,2)) < 4)
                if sorted(next,1) > bestx(1)
                    bestx(1)=sorted(next,1);
                    bestx(2)=next;
                end
            end
        end
    end

    % Is the next point in a new row?
    if bestx(2) == 0
        if direction == -1
            % Finished on this column so go on to next
            column = column + 1;
            row = 1;
            foundnew = 1;
            next = 1;
            while (next < count) & (usedcentroids(next) == 1)
                next = next + 1;
            end
            topcircle = next;
            circlenum = next;
            direction = 1;
        else
            % Back to top and search in other direction
            foundnew = 0;
            circlenum = topcircle;
            direction = -1;
        end
    else
        circlenum = bestx(2);
        row = row + 1;
        foundnew = 1;
    end

end
columnlines(row, column, :) = sorted(circlenum,:);



%plot(xdata(:), ydata(:), 'bx');
```

pkcentreofdist.m

```
function [xc, yc]=pkcentreofdist(rowlines,columnlines)
% function [xc yc]=pkcentreofdist(rowlines,columnlines)
%
% Calculates centre of distortion by analysing deviation of curves
% passing through each row and column from a straight line connecting
% the first and last circles in each row and column
%
sizerowlines=size(rowlines);
options=optimset('Display', 'off');
count = 0;
for sampledrow = 1:sizerowlines(1)
  numsamples=max(find(rowlines(sampledrow,:,1)));
  % Only use fairly complete rows
  if numsamples > (0.9*sizerowlines(2))
```

```
      count = count + 1;
      y1=rowlines(sampledrow,1:numsamples,1)';
      x1=rowlines(sampledrow,1:numsamples,2)';
      res1=polyfit(x1,y1,2);
      x=1:0.01:768;
      sizex=size(x);
      y=polyval(res1,x);
      %%%%%%%%%%%%%%%%%%%%%%%%%
      plot(x,y,'r');
      %%%%%%%%%%%%%%%%%%%%%%%%%
      num=diff(diff(y));
      den=((1+(diff(y).^2)).^(3/2));
      curv=num(1:(sizex(2)-2))./den(1:(sizex(2)-2));
      abscurv=abs(curv);
      idx=find(abscurv==max(abscurv));
           rowmaxes(count,:)=[x(idx(1)) y(idx(1)) curv(idx(1))];

        if( (count > 1) & (rowmaxes(count-1,3) > 0) & (rowmaxes(count,3) < 0) )
           % Curvature just changed sign
           % interpolate
           yc = rowmaxes(count-1,2)+(abs(rowmaxes(count-1,3))./(abs(rowmaxes(count-1,3))...
              +abs(rowmaxes(count,3)))).*(rowmaxes(count,2)-rowmaxes(count-1,2));
           break;
        end;
          end
end

sizecolumnlines=size(columnlines);
count = 0;
for sampledcolumn = 1:sizecolumnlines(2)
  numsamples=max(find(columnlines(:,sampledcolumn,1)));
  % Only use fairly complete columns
  if numsamples > (0.9*sizecolumnlines(1))
      count = count + 1;
      y1=columnlines(1:numsamples,sampledcolumn,1)';
      x1=columnlines(1:numsamples,sampledcolumn,2)';
      res1=polyfit(y1,x1,2);
      y=1:0.01:576;
      x=polyval(res1,y);
      %%%%%%%%%%%%%%%%%%%%%%%%%
      plot(x,y,'g');
      %%%%%%%%%%%%%%%%%%%%%%%%%
      sizex=size(x);
      num=diff(diff(x));
      den=((1+(diff(x).^2)).^(3/2));
      curv=num(1:(sizex(2)-2))./den(1:(sizex(2)-2));
      abscurv=abs(curv);
      idx=find(abscurv==max(abscurv));
           rowmaxes(count,:)=[x(idx(1)) y(idx(1)) curv(idx(1))];

        if( (count > 1) & (rowmaxes(count-1,3) > 0) & (rowmaxes(count,3) < 0) )
           % Curvature just changed sign
           % interpolate
           xc = rowmaxes(count-1,1)+(abs(rowmaxes(count-1,3))./(abs(rowmaxes(count-1,3))...
              +abs(rowmaxes(count,3)))).*(rowmaxes(count,1)-rowmaxes(count-1,1));
           break;
        end;
         end
end
```

pkcalcerronly.m

```
function [err]=pkcalcerror(res, modrows, distxc, distyc, corrxc, corryc)
%function totalerror=pkcalcerror(res, rowlines)
% Function to use with numerical optimisation
sizemodrows=size(modrows);
sizeres=size(res);
% initial estimate
%res=[0 0 0 0 0 0 1 0];
distx=modrows(:,:,1);
disty=modrows(:,:,2);
distrd=sqrt((distx-distxc).^2 + (disty-distyc).^2);
corrx = corrxc + polyval(res,distrd).*((distx-distxc)./distrd);
corry = corryc + polyval(res,distrd).*((disty-distyc)./distrd);
```

```
% First find b1 (gradient of all lines)
term1=sum(sum(corrx.*corry));
%Transpose twice to get row sum instead of column sum
term2=sum(((sum(corrx')').*(sum(corry')'))./sizemodrows(2));
term3=sum(sum(corrx.^2));
term4=sum(((sum(corrx')').^2)./sizemodrows(2));

b1x=(term1 - term2)./(term3-term4);
b0x=(sum(corry')' - (sum(corrx')').*b1x)./sizemodrows(2);

for count=1:sizemodrows(2)
    b0xfill(:,count)=b0x;
end
err=sum(sum(abs((corry-(corrx.*b1x)-b0xfill)./sqrt(1+b1x.^2)))./...
    sizemodrows(2))./sizemodrows(1);
```

A.3 Distortion Correction

This function performs the distortion correction on an image that has already been corrected to 768×576 square pixel format.

imcorrcanon.m

```
function [A]=imcorrcanon(Datac)

%Datac=imread('c:\phd_Work\Distortion\cameraoldfinal\calibrationimage1.bmp');
rows=576;     % Set these as absolute values because the function is only valid
columns=768; % for an image taken with this camera
colours=3;


newdata=zeros(rows,columns,colours);
%newdata=zeros(rows,columns,colours);

Datac = double(Datac);

load canondistlocs

for x=1:columns
    for y=1:rows
        newdata(y,x,:)=...
        Datac(distlocsrect(y,x,1),distlocsrect(y,x,2),:).*distlocsrect(y,x,3) +...
        Datac(distlocsrect(y,x,4),distlocsrect(y,x,5),:).*distlocsrect(y,x,6) +...
        Datac(distlocsrect(y,x,7),distlocsrect(y,x,8),:).*distlocsrect(y,x,9) +...
        Datac(distlocsrect(y,x,10),distlocsrect(y,x,11),:).*distlocsrect(y,x,12);
    end
end

A=uint8(newdata);
figure;
imshow(A);
```

Appendix B

Geographical Information Processing

B.1 OSNI Vector data import

The following Perl script was written to correct the category and attributes files for data imported into GRASS with the v.in.dxf command. It also extracts the spot height data from OSNI large scale vector files and stores the spot heights in a separate GRASS sites file.

dxffix.pl

```
#!/usr/local/bin/perl
$filename = $ARGV[0]; # Run from dig_att directory; provide filename on cmd line
open(FILE, $filename) || die($filename." not found.\n");
system "cp $filename $filename.orig";  # Save a copy as we're going to overwrite this
($sheetname, $layername) = split(/\./, $filename); # Split filename on .
@atts = <FILE>;  # Read in the whole file into atts array (one line per array element)
close(FILE);
$size=@atts;      # Find the total number of attribute entries (needed for cats file)
open(FILE, ">../dig_cats/".$filename);  # Open a file of same name in cats directory
print(FILE "# ".$size." categories\n\n\n0.00 0.00 0.00 0.00\n"); # Standard header
open(FILE1, ">".$filename); # Ready to overwrite the old attribute file
if ($layername eq "CONTROL_TE")
{                          # If it is a spot heights layer
  open(SITESFILE, ">../site_lists/".$filename);
}
$catno=0;     # Increases by one for every line in att file: unique number
foreach $att(@atts)   # For each line in attribute file
{
   chomp($att);        # Remove new line character
   @data=split(/\s+/, $att, 4);  # Split into A/L, easting, northing, description
   $linetype=$data[0]." ".$data[1]." ".$data[2]; # A/L, east, north stay together
   $category=$data[3];  # Separate out the category description as we're moving it
   if ($layername eq "CONTROL_TE")
   {                          # If it is a spot heights layer
     $category =~ s/BM //;  # remove BM (benchmark) letters so there are only
     $category =~ s/BM//;
     $category =~ s/B M //;
     $category =~ s/B M//;
     $category =~ s/rp//;
   }                          # numbers in the category description
   if ($category)             # Not a null value
   {
     $catno++;
     print(FILE1 $linetype." ".$catno."\n"); # New attributes file contains unique cat no
```

```
      print(FILE $catno.":".$category."\n");  # Cats file contains unique catno + description
      if ($layername eq "CONTROL_TE")
      {                          # If it is a spot heights layer
        print(SITESFILE $data[1]."|".$data[2]."|%".$category."\n");
      }
    }
}
close(FILE);
close(FILE1);
if ($layername eq "CONTROL_TE")
{                          # If it is a spot heights layer
  close(SITESFILE);
}
```

B.2 OSNI Raster DTM Import

OSNI provided no information on the format of the DTM file, nor recommended software to read it. The file format was therefore reverse-engineered and the following Perl script was written to read the data and convert it to a format suitable for importing as a GRASS raster with the r.in.ascii command.

osdtm.pl

```
#!/usr/local/bin/perl
# First 4 command line arguments are the lower left (easting, northing) and
# upper right (easting, northing) Irish Grid co-ordinates of the region we
# want to extract from the file. At present must be multiples of 2000
@requiredrect = @ARGV[0,1,2,3];
# This is the directory into which the converted ASCII raster files will be put.
# It should exist already.
$directoryname = $ARGV[4];
# And this is the file that contains the OSNI DTM data
$filename = "dtm280.txt";
open(FILE, $filename) || die($filename."_not_found.\n");
# Read in the whole file into atts array (one line per array element)
@dtmdata = <FILE>;
close(FILE);

# To start with we are neither in the midst of reading a data header or a data block
$readingheader = 0;
$readingblock = 0;
foreach $dtmline(@dtmdata)
{
   chomp($dtmline);        # Remove new line character
# First two digits in each line tell us what type of line it is
   $startcode = substr $dtmline,0,2;
# First header line starts with 50 and contains lower left easting and northing
   if ($startcode eq "50")
   {
     $readingheader = 1;

     @firstline=split(/\s+/,$dtmline);
     @actualrect[0,1]=@firstline[5,6];
   }
   if ($startcode eq "00")
   {
# Second header line starts with 00 and contains upper right easting and northing
     if ($readingheader == 1)
     {
       @firstline=split(/\s+/,$dtmline);
       @actualrect[2,3]=@firstline[1,2];
     }
# Normal data lines also start with 00
     else
     {
# Need this check in case this block isn't in our required region and we want
# to skip it
       if ($readingblock == 1)
```

```
        {
# The 3rd character on the line is the start of data (ignoring 2-digit start code)
          $lineindex = 2;
          $rasterrow[$rastercolumnindex] = "";
          while ($lineindex < 74)
          {
# All fields are 5 characters wide. Can't separate on white space because there
# sometimes is none (e.g. -1000 null values takes up whole 5 characters)
            $rasterrow[$rastercolumnindex] = substr($dtmline,$lineindex,5);
            $lineindex = $lineindex + 5;
            $rastercolumnindex++;
            if ($rastercolumnindex == 40)
            {
              $raster[$rasterrowindex] = "";
              foreach $rastercolumn(@rasterrow)
              {
# Concatenate the values for a row into one string, separated by spaces
                $raster[$rasterrowindex] = ($raster[$rasterrowindex].$rastercolumn." ");
              }

              $rasterrowindex++;
              $rastercolumnindex = 0;
              if ($rasterrowindex == 40)
              {
# For GRASS r.in.ascii format must have rows ordered from top down
                @revraster = reverse @raster;
                foreach $line(@revraster)
                {
                  print OUTFILE $line;
                  print OUTFILE "\n";
                }
                close(OUTFILE);
                $readingblock = 0;
              }
            }
          }
        }
      }
    }
# Third line of block header contains no useful information but signals that
# we are ready to start reading the block so can write header to output file etc.
    if ($startcode eq "51")
    {
# Requiredrect values should be multiples of 2000 (2km) for this to work properly
      if ( ($actualrect[0] - $requiredrect[0] >= 0) &&
      ($actualrect[1] - $requiredrect[1] >= 0) &&
      ($requiredrect[2] - $actualrect[2] >= 0) &&
      ($requiredrect[3] - $actualrect[3] >= 0) )
      {
        open(OUTFILE, ">".$directoryname."/".substr($actualrect[0],0,3).substr($actualrect[1],0,3));
# Standard header format required by GRASS r.in.ascii command
# GRASS expects outer boundary values of block; OSNI data has lower left
# co-ordinates of the cells at extremities so 50 must be added to these
        print(OUTFILE "north:\t".($actualrect[3]+50)."\n");
        print(OUTFILE "south:\t".$actualrect[1]."\n");
        print(OUTFILE "east:\t".($actualrect[2]+50)."\n");
        print(OUTFILE "west:\t".$actualrect[0]."\n");
        print(OUTFILE "rows:\t40\n");
        print(OUTFILE "cols:\t40\n");
# A null value / gap in the data (e.g. wide river) is signalled by -1000
        print(OUTFILE "null:\t-1000\n");
        print(OUTFILE "type:\tfloat\n");
# OSNI values are in decimetres (10cm)
        print(OUTFILE "multiplier:\t0.1\n");

# We have definitely started reading a block now so can set all these variables
        $readingblock = 1;
        $rasterrowindex = 0;
        $rastercolumnindex = 0;
        $readingheader = 0;
      }

    }
}
```

B.3 Combined DEM

The following Perl script was written to combine height data from various sources into one sites file with an additional smoothing attribute, suitable for interpolation in GRASS using s.surf.rst with the variable smoothing option.

combineheights.pl

```
#!/usr/local/bin/perl
# Cmd-line arguments are: contour file, spotheight file, re-sampled OSNI DTM file
# Line below contains the offset values for the elevations in each input file
# (i.e. average amount below or above the spot heights)
@offsets=(0, 0, 0);
# Line below contains the smoothing precision values for each input file
#@suffixes=(6.53, 0.1, 6.96);
#@suffixes=(9.52, 0.1, 8.86);
@suffixes=(10, 7);
open(OUTFILE, ">smoothheights");
# For each input data file (there are 3)
for $whichfile(0..1)
{
# Use $whichfile counter as an index into the command-line arguments
  $filename=$ARGV[$whichfile];
  open(FILE, $filename) || die($filename." not found.\n");
  # Read in the whole file into atts array (one line per array element)
  @heightdata = <FILE>;
  close(FILE);
# For each height value in the input file
  foreach $heightline(@heightdata)
  {
    chomp($heightline);        # Remove new line character
# If the first character of the line is a number, i.e. this is a data line
    if (substr($heightline,0,1) =~ /[0-9]/)
    {
      ($easting, $northing, $attributes) = split(/\|/, $heightline, 3); # Split line on |
      @elevvalues = split(/%/, $attributes); # Split line on %
# Copy the line to the output file (with elevation value appropriately
# adjusted) followed by an additional attribute
# representing the smoothing value for this height point
      print(OUTFILE $easting."|".$northing."|%");
# Use of -1 index gives last value in the array. Thus is OK if there was more
# than one numeric attribute in the site list; it will take the last one. This
# is useful if the file has been s.out.ascii/s.in.ascii'ed to reduce the
# number of points to those only in the current region.
      print(OUTFILE $elevvalues[-1]-$offsets[$whichfile]." %");
      print(OUTFILE $suffixes[$whichfile]."\n");

    }
  }
}
close(OUTFILE);
```

Appendix C

3-D Visualisation in the GRASS GIS

C.1 Line-of-sight 'viewable wedge' creation

The following shell script can be run within GRASS and will use the r.mapcalc command to generate a map 'losmask' containing the viewable cells (1) and hidden cells (0).

```
#!/bin/sh
# Camera location co-ordinates
xl=352052.866
yl=334763.923
# Centre of view co-ordinates
xc=352131.583
yc=334702.293
# Camera horizontal field of view in degrees
fov=57.8
r.mapcalc "losmask = eval(covheading = atan($xc - $xl, $yc - $yl), \
heading = atan(x() - $xl, y() - $yl), min = covheading - $fov / 2, \
max = covheading + $fov / 2, min = if(min < 0, min + 360, min), \
max = if(max > 360, max - 360, max), if(heading > min && heading < max))"
```

C.2 SG3d Modifications

The following diff file shows the changes made in terms of C source code to SG3d to enable camera modelling and co-ordinate dump functionality. Note that only the changed lines are shown.

```
diff -u SG3d/Dcell.c SG3d.new/Dcell.c
--- SG3d/Dcell.c        2002-03-25 12:17:20.000000000 +0000
+++ SG3d.new/Dcell.c    2003-03-11 21:10:32.751473058 +0000
@@ -473,19 +473,24 @@
      y = FROM_TO[FROM][Y];
      z = FROM_TO[FROM][Z];

-     if(x < X_Max + 20. && x > X_Min - 20. &&
-           y < Y_Max + 20. && y > Y_Min - 20. &&
-           z < Z_Max + 20. && z > Z_Min - 20.){
-       nearclip = 0.01;
+     if(x < X_Max + 20. && x > X_Min - 20. &&       /* Viewer is within the */
+           y < Y_Max + 20. && y > Y_Min - 20. &&  /* region covered by    */
```

```
+            z < Z_Max + 20. && z > Z_Min - 20.){    /* the data              */
+/*      nearclip = 0.01; */
+         /* PK Focal length for NVDS15B Camera 3.45mm */
+         nearclip = 0.00345;
          farclip = 1500.0;
      }
-     else{
-        nearclip = 20.0;
-        farclip = 4000.0;
+     else{                                          /* Viewer is far away;  */
+        nearclip = 20.0;                           /* zoom in with a long  */
+        farclip = 4000.0;                          /* focal length         */
      }

      getviewport (&left, &right, &bottom, &top);
-     aspect = (float) (right-left) / (top-bottom);
+    printf("left: %d right: %d bottom: %d top: %d\n", left, right, bottom, top);
+     aspect = (float) (right-left+1) / (top-bottom+1);
+     /* PK Correct for non-square pixels */
+/*       aspect = 1.394666667; */

      if(Aortho->val){
         Osize = 1 + 2000 * Aheight->val;
@@ -501,7 +506,10 @@
         ortho(-Osize*aspect,Osize*aspect,-Osize,Osize, nearclip, farclip);
      }
      else
+      {
+         printf("Calling perspective(fovy %d, aspect %f, near %f, far %f)\n",
+                persp, aspect, nearclip, farclip);
         perspective (persp, aspect, nearclip, farclip);
+      }

      loadmatrix(ID_matrix);

diff -u SG3d/externs.h SG3d.new/externs.h
--- SG3d/externs.h      2003-02-20 09:45:34.000000000 +0000
+++ SG3d.new/externs.h  2003-03-13 21:50:02.000154051 +0000
@@ -344,7 +344,7 @@
 GLOBAL Actuator *Avwidth1, *Avwidth2, *Adrape, *Aflat, *Avcolor, *Anewvect;
 GLOBAL Actuator *Afocus, *Abgcolor, *Anozero, *Anofocus;
 GLOBAL Actuator *AvCcset, *AbgCcset, *AstCcset, *AscaleCcset, *AlabelCcset;
-GLOBAL Actuator *Adotype, *Acenter, *Astcolor, *Adump, *Aglobe;
+GLOBAL Actuator *Adotype, *Acenter, *Astcolor, *Adump, *Aglobe, *Acoorddump;
 GLOBAL Actuator *Anoclear, *Aclear;

 GLOBAL Actuator *AcCpal, *Accred, *Accgrn, *Accblu, *Accok, *Accbox;
Only in SG3d.new: externs.h~
diff -u SG3d/focus.c SG3d.new/focus.c
--- SG3d/focus.c        1999-12-29 15:12:23.000000000 +0000
+++ SG3d.new/focus.c    2003-04-30 17:04:31.460796805 +0100
@@ -21,6 +21,8 @@
 void reset_fromto ();
 void draw_x ();

+int QUB_viewportx, QUB_viewporty, QUB_dumpflag=0;
+
 void
 keep_focus()
 {
@@ -126,6 +128,16 @@
                            show_what(surf_pt[X]/XYscale + c.west,
                              surf_pt[Y]/XYscale + c.south,
                              surf_pt[Z]/Z_exag + Zoff);
+                           /* Print extra information (distance to point) to
+                            * the screen when a 3-D reverse look-up is done */
+                           fprintf(stderr,"Distance: %.2f\n",
+                          sqrt( (surf_pt[X]/XYscale - fr_to[FROM][X]/XYscale)*
+                              (surf_pt[X]/XYscale - fr_to[FROM][X]/XYscale)
+                             + (surf_pt[Y]/XYscale - fr_to[FROM][Y]/XYscale)*
+                              (surf_pt[Y]/XYscale - fr_to[FROM][Y]/XYscale)
+                             + (surf_pt[Z]/Z_exag - fr_to[FROM][Z]/Z_exag)*
+                              (surf_pt[Z]/Z_exag - fr_to[FROM][Z]/Z_exag) )
+                                    );
                            fprintf(stderr,"\nleft button : What's here? \n");
```

```
                            fprintf(stderr,"right button : Quit \n");
                        }
@@ -429,6 +441,104 @@
     cpack(0xFFFFFF);
 }

+/* PK 13 June 2002 */
+/* Function asks for a file name for the co-ordinate dump and checks that it
+ * was completed successfully. Copied from save_window_img() in writeimg.c */
+
+void
+save_coord_dump()
+{
+char filename[80];
+
+    fprintf(stderr,"Enter name of ASCII file for co-ordinate dump: ");
+    gets(filename);
+    fprintf(stderr,"\n");
+
+    if(filename[0] != '\0'){
+       if(0 > write_coord(left,bottom,right,top,filename))
+           fprintf(stderr,"Unable to save %s.\n", filename);
+       else
+           fprintf(stderr,"%s saved.\n", filename);
+    }
+    else
+       fprintf(stderr,"<request cancelled>\n");
+
+}
+
+/* Function scans through all the pixels in the image, performs a 3-D reverse
+ * look-up using get_los() and los_intersect() on each one and writes the
+ * results to a text file. Loosely based on write_rgb() in writeimg.c */
+
+write_coord(l,b,r,t,name)
+Screencoord l, b, r, t;
+char *name;
+{
+
+    int xsize, ysize;
+    FILE *fileptr;
+    float  fr_to[2][4], surf_pt[3];
+    struct Cell_head c;
+
+    QUB_dumpflag = 1; /* Now in get_los() x and y values will be taken from
+                       * this function and not set by mouse clicks */
+
+    xsize = r - l + 1;
+    ysize = t - b + 1;
+
+
+    if(NULL == (fileptr = fopen(name, "w")))
+       return (-1);
+
+    QUB_viewporty = 0;
+    QUB_viewportx = 0;
+/* Observer location */
+    get_los(fr_to);
+    if (los_intersect(surf_pt,fr_to)){
+      G_get_set_window (&c);
+      /* First line in file contains observer easting, northing & elevation */
+      fprintf(fileptr,"%.2f %.2f %.2f\n",
+         fr_to[FROM][X]/XYscale + c.west,
+         fr_to[FROM][Y]/XYscale + c.south,
+         fr_to[FROM][Z]/Z_exag + Zoff);
+    }
+
+    for(QUB_viewporty=0; QUB_viewporty<ysize; QUB_viewporty++) {
+
+       for(QUB_viewportx=0; QUB_viewportx<xsize; QUB_viewportx++){
+
+           get_los(fr_to);
+           if (los_intersect(surf_pt,fr_to)){
+               G_get_set_window (&c);
+               /* row, column, easting, northing, elevation, distance */
```

```
+                fprintf(fileptr, "%d %d %.2f %.2f %.2f %.2f\n",
+                                QUB_viewporty,
+                                QUB_viewportx,
+                                surf_pt[X]/XYscale + c.west,
+                                surf_pt[Y]/XYscale + c.south,
+                                surf_pt[Z]/Z_exag + Zoff,
+                                sqrt(
+                                  (surf_pt[X]/XYscale - fr_to[FROM][X]/XYscale)*
+                                  (surf_pt[X]/XYscale - fr_to[FROM][X]/XYscale)
+                                + (surf_pt[Y]/XYscale - fr_to[FROM][Y]/XYscale)*
+                                  (surf_pt[Y]/XYscale - fr_to[FROM][Y]/XYscale)
+                                + (surf_pt[Z]/Z_exag - fr_to[FROM][Z]/Z_exag)*
+                                  (surf_pt[Z]/Z_exag - fr_to[FROM][Z]/Z_exag) )
+                        );
+
+            }
+            else
+                /* Use -1000 as null value to signal invalid image data (e.g.
+                 * sky or sea) */
+                fprintf(fileptr, "%d %d -1000 -1000 -1000 -1000\n", QUB_viewporty, QUB_viewportx);
+
+        }
+
+    }
+    fclose(fileptr);
+    QUB_dumpflag = 0;
+    return(0);
+}
+
 int
 get_los(vect)
 float  vect[2][4];
@@ -453,13 +563,24 @@

         getorigin(&ox, &oy);
         getviewport (&left, &right, &bottom, &top);
-        aspect = (float) (right-left) / (top-bottom);
+        aspect = (float) (right-left+1) / (top-bottom+1);
         ox += left;
         oy += bottom;
         pix_w = right - left + 1;
         pix_h = top - bottom + 1;
-        x = getvaluator(MOUSEX) - ox;
-        y = getvaluator(MOUSEY) - oy;
+    /* By PK 13 June 2002 */
+    /* If we are doing a co-ordinate dump, get pixel values from global
+     * variables set in write_coord(), else use mouse interactively */
+        if (QUB_dumpflag == 0)
+        {
+           x = getvaluator(MOUSEX) - ox;
+           y = getvaluator(MOUSEY) - oy;
+        }
+        else
+        {
+           x = QUB_viewportx;
+           y = QUB_viewporty;
+        }
         if(x < 0 || x > pix_w || y < 0 || y > pix_h)
             return(0);
         pix_dx = x - pix_w/2;
diff -u SG3d/panel.c SG3d.new/panel.c
--- SG3d/panel.c        1999-12-29 15:12:24.000000000 +0000
+++ SG3d.new/panel.c    2003-03-13 21:50:08.840107149 +0000
@@ -142,6 +142,7 @@
 extern void set_real_to();
 extern void put_scale();
 extern void save_window_img();
+extern void save_coord_dump();
 Panel * my_initscriptpanel ();

 #define PTOGGLE(x) ((x) = !(x))
@@ -799,6 +800,15 @@
     Adump->y = -1.5;
     pnl_addact (Adump, panel);
```

```
+     /* PK 13 June 2002 */
+     /* Add Panel GUI button for co-ordinate dump */
+     /* Coord Dump */
+     Acoorddump=pnl_mkact (pnl_wide_button);
+     Acoorddump->label = "Coord Dp";
+     Acoorddump->x = 3.2;
+     Acoorddump->y = -2.0;
+     pnl_addact (Acoorddump, panel);
+
      /* Surface */
      Asurface=pnl_mkact (pnl_toggle_button);
      Asurface->label = " Surface Only";
@@ -1961,6 +1971,11 @@
         save_window_img();
         return (0);
      }
+     if (a == Acoorddump)
+     {
+        save_coord_dump();
+         return (0);
+     }
      if (a == Afast1 || a == Afast2)
      {
         slow_start(a, Afast1, Afast2, &fast_res, 10);
```

C.3 Co-ordinate dump file manipulation

The following MATLAB scripts were created to manipulate the co-ordinate dump file output by SG3d and merge the results with image data prior to interpolation or database storage.

readin.m

```
filename=['qhcoordapr24smothin.txt'];
clear grass;
frame=0;
clear row;
clear column;
clear east;
clear north;
clear elev;
clear observer;
% Dimension arrays for speed; 416017 = 721 * 577
row=zeros(416017,1);
column=zeros(416017,1);
east=zeros(416017,1);
north=zeros(416017,1);
elev=zeros(416017,1);
dist=zeros(416017,1);
vec=fopen(filename,'rt');        % Open as a text file for read-only with
                                 % handle 'vec'
% First line contains observer location
observer=str2num(fgetl(vec));
while ~feof(vec)                 % While not at end of file
  frame=frame+1;                 % Read data for a new frame
  grass=str2num(fgetl(vec));
% GL row ordering works in opposite direction to MATLAB and starts at 0
  row(frame) = int16(577 - grass(1));
% Column ordering also starts at 0 so add 1 here as well
  column(frame) = int16(grass(2) + 1);
  east(frame)=grass(3);
  north(frame)=grass(4);
%  elev(frame)=grass(5)./100;  % When using GRASS 4.3 elev values were in cm
  elev(frame)=grass(5);
  dist(frame)=grass(6);
end
fclose(vec);                     % Close vector.txt file
```

writergb.m

```
origimage=corrected;

numpoints=size(east);

% 1st Pass
% Need to project every point into a grid

rfile = fopen('qhapr24coordsmothin.r','w');
gfile = fopen('qhapr24coordsmothin.g','w');
bfile = fopen('qhapr24coordsmothin.b','w');
for count=1:numpoints
    origrow=row(count);
    origcolumn=column(count);
    easting = east(count);
    if (easting > -1000 & origrow < 577 & origcolumn < 721 & validpels(origrow,origcolumn))
      northing = north(count);
      fprintf(rfile, '%.2f|%.2f|%%%.0f\n', easting, northing, double(origimage(origrow,origcolumn,1)));
      fprintf(gfile, '%.2f|%.2f|%%%.0f\n', easting, northing, double(origimage(origrow,origcolumn,2)));
      fprintf(bfile, '%.2f|%.2f|%%%.0f\n', easting, northing, double(origimage(origrow,origcolumn,3)));
    end
end
fclose(rfile);
fclose(gfile);
fclose(bfile);
```

C.4 Improvements to s.surf.idw

The following unified diff file shows the improvements made to s.surf.idw in terms of C source code. Only the changed lines are shown. The diff files may also be browsed at http://freegis.org/cgi-bin/viewcvs.cgi/grass/src/sites/s.surf.idw/cmd/ .

```
diff -u cmd.old/main.c cmd/main.c
--- cmd.old/main.c      2004-05-01 19:41:20.953183074 +0100
+++ cmd/main.c  2003-04-24 17:03:52.356256063 +0100
@@ -1,41 +1,63 @@
+#include <stdlib.h>
+#include <math.h>
 #include "gis.h"
 #include "site.h"

 int search_points = 12;

-int npoints = 0;
-int npoints_alloc = 0;
+long npoints = 0;
+long **npoints_currcell;
 int nsearch;
+static int i;

 struct Point
 {
     double north, east;
     double z;
+};
+struct list_Point
+{
+    double north, east;
+    double z;
     double dist;
 };
-struct Point *points = NULL;
-struct Point *list;
-
+struct Point ***points;
+struct Point *noidxpoints = NULL;
+struct list_Point *list;
+static struct Cell_head window;
+static struct Flag *noindex;
```

```
+void calculate_distances(int, int, double, double, int*);
+void calculate_distances_noindex(double, double);
+/* read_sites.c */
+void read_sites(char *, int);

 int main(int argc, char *argv[])
 {
     int fd, maskfd;
     CELL  *mask;
     DCELL *dcell;
-    struct Cell_head window;
     struct GModule *module;
     int row, col;
+    int searchrow, searchcolumn, pointsfound;
+    int *shortlistrows=NULL, *shortlistcolumns=NULL;
+    long ncells;
     double north, east;
-    double dx,dy;
-    double maxdist,dist;
-    double sum1, sum2;
-    int i,n,max, field;
-    void read_sites();
-    char errmsg[200];
+    double dist;
+    double sum1, sum2, interp_value;
+    int n, field;
     struct
     {
-       struct Option *input, *npoints, *output, *dfield;
+        struct Option *input, *npoints, *output, *dfield;
     } parm;
+    struct cell_list
+    {
+        int row, column;
+        struct cell_list *next;
+    };
+    struct cell_list **search_list, **search_list_start;
+    int max_radius, radius;
+    int searchallpoints = 0;
+

     parm.input = G_define_option() ;
     parm.input->key        = "input" ;
@@ -66,108 +88,396 @@
     parm.dfield->multiple = NO;
     parm.dfield->required = NO;
     parm.dfield->description = "which decimal attribute (if multiple)";
+
+    noindex = G_define_flag();
+    noindex->key = 'n';
+    noindex->description = "Don't index sites by cell (for very large regions;"
+                           " uses less memory)";

     G_gisinit(argv[0]);

     module = G_define_module();
     module->description =
                     "Surface interpolation from sites data by Inverse "
-                    "Distance Weighted algorithm.";
+                    "Distance Squared Weighting.";

     if (G_parser(argc, argv))
         exit(1);

+    fprintf(stderr, "%s:\n", G_program_name());
+
     if (G_legal_filename(parm.output->answer) < 0)
     {
-       fprintf (stderr, "%s=%s - illegal name\n", parm.output->key, parm.output->answer);
-       exit(1);
+        fprintf (stderr, "%s=%s - illegal name\n", parm.output->key, parm.output->answer);
+        exit(1);
     }
```

```
     if(sscanf(parm.npoints->answer,"%d", &search_points) != 1 || search_points<1)
     {
-       fprintf (stderr, "%s=%s - illegal number of interpolation points\n",
-               parm.npoints->key, parm.npoints->answer);
-       G_usage();
-       exit(1);
+        fprintf (stderr, "%s=%s - illegal number of interpolation points\n",
+                parm.npoints->key, parm.npoints->answer);
+        G_usage();
+        exit(1);
     }

     sscanf(parm.dfield->answer,"%d", &field);
     if (field < 1)
-    {
-      sprintf (errmsg, "Decimal attribute field 0 doesn't exist.");
-      G_fatal_error (errmsg);
-    }
+      G_fatal_error ("Decimal attribute field 0 doesn't exist.");

-    list = (struct Point *) G_calloc (search_points, sizeof (struct Point));
+    list = (struct list_Point *) G_calloc (search_points, sizeof (struct list_Point));


+/* get the window, dimension arrays */
+    G_get_window (&window);
+
+  if(!noindex->answer)
+  {
+    npoints_currcell = (long **)G_malloc(window.rows * sizeof(long));
+    points = (struct Point ***)G_malloc(window.rows * sizeof(struct Point));
+
+
+    for(row = 0; row < window.rows; row++)
+    {
+        npoints_currcell[row] = (long *)G_malloc(window.cols * sizeof(long));
+        points[row] = (struct Point **)G_malloc(window.cols * sizeof(struct Point));
+
+        for(col = 0; col < window.cols; col++)
+        {
+            npoints_currcell[row][col] = 0;
+            points[row][col] = NULL;
+        }
+    }
+  }
+
 /* read the elevation points from the input sites file */
     read_sites (parm.input->answer, field);

     if (npoints == 0)
     {
-       fprintf (stderr, "%s: no data points found\n", G_program_name());
-       exit(1);
+        fprintf (stderr, "%s: no data points found\n", G_program_name());
+        exit(1);
     }
     nsearch = npoints < search_points ? npoints : search_points;

-/* get the window, allocate buffers, etc. */
-    G_get_window (&window);
-
-    /*cell = G_allocate_cell_buf();*/
+  if(!noindex->answer)
+  {
+    /* Arbitrary point to switch between searching algorithms. Could do
+     * with refinement PK */
+    if( (window.rows*window.cols)/npoints > 400 )
+    {
+        /* Using old algorithm.... */
+        searchallpoints = 1;
+        ncells = 0;
+
+        /* Make an array to contain the row and column indices that have
+         * sites in them; later will just search through all these. */
+        for( searchrow=0; searchrow<window.rows; searchrow++)
```

```
+            for(searchcolumn=0; searchcolumn<window.cols; searchcolumn++)
+                if( npoints_currcell[searchrow][searchcolumn] > 0 )
+                {
+                    shortlistrows = (int *)G_realloc(shortlistrows,
+                                                     (1 + ncells)*sizeof(int));
+                    shortlistcolumns = (int *)G_realloc(shortlistcolumns,
+                                                     (1 + ncells)*sizeof(int));
+                    shortlistrows[ncells] = searchrow;
+                    shortlistcolumns[ncells] = searchcolumn;
+                    ncells++;
+                }
+    }
+    else
+    {
+        /* Fill look-up table of row and column offsets for
+         * doing a circular region growing search looking for sites */
+        /* Use units of column width */
+        max_radius = (int)(0.5 + sqrt(window.cols*window.cols +
+          (window.rows*window.ns_res/window.ew_res)*(window.rows*window.ns_res/window.ew_res)));
+
+        search_list = (struct cell_list **)G_malloc(max_radius * sizeof(struct cell_list));
+        search_list_start = (struct cell_list **)G_malloc(max_radius * sizeof(struct cell_list));
+
+        for(radius = 0; radius < max_radius; radius++)
+            search_list[radius] = NULL;
+
+        for(row = 0; row < window.rows; row++)
+            for(col = 0; col < window.cols; col++)
+            {
+                radius = (int)sqrt(col*col +
+          (row*window.ns_res/window.ew_res)*(row*window.ns_res/window.ew_res));
+                if (search_list[radius] == NULL)
+                    search_list[radius] =
+                      search_list_start[radius] = G_malloc(sizeof(struct cell_list));
+                else
+                    search_list[radius] =
+                      search_list[radius]->next = G_malloc(sizeof(struct cell_list));
+
+                search_list[radius]->row = row;
+                search_list[radius]->column = col;
+                search_list[radius]->next = NULL;
+            }
+    }
+ }
+
+/* allocate buffers, etc. */
+
     dcell=G_allocate_d_raster_buf();

     if ((maskfd = G_maskfd()) >= 0)
-       mask = G_allocate_cell_buf();
+        mask = G_allocate_cell_buf();
     else
-       mask = NULL;
+        mask = NULL;

-  /*fd = G_open_cell_new(parm.output->answer);
-    if (fd < 0)
-    {
-       fprintf (stderr, "%s: can't create %s\n", G_program_name(), parm.output->answer);
-       exit(1);
-    }
-  */
-    fd=G_open_raster_new(parm.output->answer,2);
+
+    fd=G_open_raster_new(parm.output->answer, DCELL_TYPE);
     if (fd < 0)
     {
         fprintf (stderr, "%s: can't create %s\n", G_program_name(), parm.output->answer);
         exit(1);
     }
+

     fprintf (stderr, "Interpolating raster map <%s> ... %d rows ... ",
-       parm.output->answer, window.rows);
```

```
+         parm.output->answer, window.rows);

      north = window.north + window.ns_res/2.0;
      for (row = 0; row < window.rows; row++)
      {
          fprintf (stderr, "%-10d\b\b\b\b\b\b\b\b\b\b", window.rows-row);

-         if (mask)
-         {
-             if(G_get_map_row(maskfd, mask, row) < 0)
-                 exit(1);
-         }
-         north -= window.ns_res;
-         east = window.west - window.ew_res/2.0;
-         for (col = 0; col < window.cols; col++)
-         {
-             east += window.ew_res;
-                 /* don't interpolate outside of the mask */
-             if (mask && mask[col] == 0)
-             {
-                 G_set_d_null_value(&dcell[col], 1);
-                 continue;
-             }
-                 /* fill list with first nsearch points */
+          if (mask)
+          {
+              if(G_get_map_row(maskfd, mask, row) < 0)
+                  exit(1);
+          }
+          north -= window.ns_res;
+          east = window.west - window.ew_res/2.0;
+          for (col = 0; col < window.cols; col++)
+          {
+              east += window.ew_res;
+                  /* don't interpolate outside of the mask */
+              if (mask && mask[col] == 0)
+              {
+                  G_set_d_null_value(&dcell[col], 1);
+                  continue;
+              }
+
+             /* If current cell contains more than nsearch points just average
+              * all the points in this cell and don't look in any others */
+
+              if (!(noindex->answer) && npoints_currcell[row][col] >= nsearch)
+              {
+                  sum1 = 0.0;
+                  for (i = 0; i < npoints_currcell[row][col]; i++)
+                      sum1 += points[row][col][i].z;
+
+                  interp_value = sum1/npoints_currcell[row][col];
+              }
+              else
+              {
+               if(noindex->answer)
+                 calculate_distances_noindex(north, east);
+               else
+               {
+                  pointsfound = 0;
+                  i = 0;
+
+                  if( searchallpoints == 1 )
+                  {
+                      /* If there aren't many sites just check them all to find
+                       * the nearest */
+                      for( n=0; n<ncells; n++)
+                          calculate_distances(shortlistrows[n], shortlistcolumns[n],
+                                              north, east, &pointsfound);
+                  }
+                  else
+                  {
+                      radius = 0;
+                      while(pointsfound < nsearch)
+                      {
+                          /* Keep widening the search window until we find
```

```
+                        * enough points */
+                       search_list[radius] = search_list_start[radius];
+                       while(search_list[radius] != NULL)
+                       {
+                           /* Always */
+                           if (row < (window.rows - search_list[radius]->row) &&
+                               col < (window.cols - search_list[radius]->column))
+                           {
+                               searchrow = row + search_list[radius]->row;
+                               searchcolumn = col + search_list[radius]->column;
+                               calculate_distances(searchrow, searchcolumn,
+                                                   north, east, &pointsfound);
+                           }
+
+                           /* Only if at least one offset is not 0 */
+                           if ((search_list[radius]->row > 0   ||
+                                search_list[radius]->column > 0 ) &&
+                               row >= search_list[radius]->row &&
+                               col >= search_list[radius]->column )
+                           {
+                               searchrow = row - search_list[radius]->row;
+                               searchcolumn = col - search_list[radius]->column;
+                               calculate_distances(searchrow, searchcolumn,
+                                                   north, east, &pointsfound);
+                           }
+
+                           /* Only if both offsets are not 0 */
+                           if (search_list[radius]->row > 0  &&
+                               search_list[radius]->column > 0 )
+                           {
+                               if (row < (window.rows - search_list[radius]->row) &&
+                                   col >= search_list[radius]->column )
+                               {
+                                   searchrow = row + search_list[radius]->row;
+                                   searchcolumn = col - search_list[radius]->column;
+                                   calculate_distances(searchrow, searchcolumn,
+                                                       north, east, &pointsfound);
+                               }
+                               if (row >= search_list[radius]->row &&
+                                   col < (window.cols - search_list[radius]->column))
+                               {
+                                   searchrow = row - search_list[radius]->row;
+                                   searchcolumn = col + search_list[radius]->column;
+                                   calculate_distances(searchrow, searchcolumn,
+                                                       north, east, &pointsfound);
+                               }
+                           }
+
+                           search_list[radius] = search_list[radius]->next;
+                       }
+                       radius++;
+                   }
+               }
+           }
+
+           /* interpolate */
+           sum1 = 0.0;
+           sum2 = 0.0;
+           for (n = 0; n < nsearch; n++)
+           {
+               if(dist = list[n].dist)
+               {
+                   sum1 += list[n].z / dist;
+                   sum2 += 1.0/dist;
+               }
+               else
+               {
+                   /* If one site is dead on the centre of the cell, ignore
+                    * all the other sites and just use this value.
+                    * (Unlikely when using floating point numbers?) */
+                   sum1 = list[n].z;
+                   sum2 = 1.0;
+                   break;
+               }
+           }
```

```
+                interp_value = sum1/sum2;
+            }
+            dcell[col] = (DCELL) interp_value;
+        }
+        G_put_d_raster_row(fd,dcell);
+    }
+    G_close_cell(fd);
+    fprintf (stderr, "done__________\n");
+    exit(0);
+}
+
+void newpoint ( double z,double east,double north)
+{
+    int row, column;
+
+    row    = (int)((window.north - north) / window.ns_res);
+    column = (int)((east - window.west) / window.ew_res);
+
+    if (row<0 || row>=window.rows || column<0 || column>=window.cols)
+        ;
+    else /* For now ignore sites outside current region */
+    {
+  if(!noindex->answer)
+  {
+        points[row][column] = (struct Point *) G_realloc (points[row][column],
+                  (1 + npoints_currcell[row][column]) * sizeof (struct Point));
+        points[row][column][npoints_currcell[row][column]].north = north;
+        points[row][column][npoints_currcell[row][column]].east  = east;
+        points[row][column][npoints_currcell[row][column]].z     = z;
+        npoints_currcell[row][column]++;
+  }
+  else
+  {
+        noidxpoints = (struct Point *) G_realloc(noidxpoints,
+                                (1 + npoints) * sizeof (struct Point));
+        noidxpoints[npoints].north = north;
+        noidxpoints[npoints].east  = east;
+        noidxpoints[npoints].z     = z;
+  }
+        npoints++;
+    }
+}
+
+void calculate_distances(int row, int column, double north,
+                         double east, int *pointsfound)
+{
+    int j, n, max;
+    double dx, dy, dist;
+    static double maxdist;
+
+                /* Check distances and find the points to use in interpolation */
+    for (j = 0; j < npoints_currcell[row][column]; j ++)
+    {
+                /* fill list with first nsearch points */
+        if (i < nsearch)
+        {
+            dy = points[row][column][j].north - north;
+            dx = points[row][column][j].east  - east;
+            list[i].dist = dy*dy + dx*dx;
+            list[i].z = points[row][column][j].z;
+            i++;
+
+            /* find the maximum distance */
+            if (i == nsearch)
+            {
+                maxdist = list[max=0].dist;
+                for (n = 1; n < nsearch; n++)
+                {
+                    if (maxdist < list[n].dist)
+                        maxdist = list[max=n].dist;
+                }
+            }
+        }
+        else
+        {
```

```
+
+            /* go thru rest of the points now */
+            dy = points[row][column][j].north - north;
+            dx = points[row][column][j].east  - east;
+            dist = dy*dy + dx*dx;
+
+            if (dist < maxdist)
+            {
+                /* replace the largest dist */
+                list[max].z = points[row][column][j].z;
+                list[max].dist = dist;
+                maxdist = list[max=0].dist;
+                for (n = 1; n < nsearch; n++)
+                {
+                    if (maxdist < list[n].dist)
+                        maxdist = list[max=n].dist;
+                }
+            }
+
+        }
+    }
+    *pointsfound += npoints_currcell[row][column];
+}
+
+void calculate_distances_noindex(double north, double east)
+{
+    int n, max;
+    double dx, dy, dist;
+    double maxdist;
+
+                /* fill list with first nsearch points */
             for (i = 0; i < nsearch ; i++)
             {
-                dy = points[i].north - north;
-                dx = points[i].east  - east;
+                dy = noidxpoints[i].north - north;
+                dx = noidxpoints[i].east  - east;
                 list[i].dist = dy*dy + dx*dx;
-                list[i].z = points[i].z;
+                list[i].z = noidxpoints[i].z;
             }
                 /* find the maximum distance */
             maxdist = list[max=0].dist;
@@ -179,14 +489,14 @@
                 /* go thru rest of the points now */
             for ( ; i < npoints; i++)
             {
-                dy = points[i].north - north;
-                dx = points[i].east  - east;
+                dy = noidxpoints[i].north - north;
+                dx = noidxpoints[i].east  - east;
                 dist = dy*dy + dx*dx;

                 if (dist < maxdist)
                 {
                         /* replace the largest dist */
-                    list[max].z = points[i].z;
+                    list[max].z = noidxpoints[i].z;
                     list[max].dist = dist;
                     maxdist = list[max=0].dist;
                     for (n = 1; n < nsearch; n++)
@@ -197,43 +507,4 @@
                 }
             }

-                /* interpolate */
-            sum1 = 0.0;
-            sum2 = 0.0;
-            for (n = 0; n < nsearch; n++)
-            {
-                if(dist = list[n].dist)
-                {
-                    sum1 += list[n].z / dist;
-                    sum2 += 1.0/dist;
-                }
```

```
-                else
-                {
-                    sum1 = list[n].z;
-                    sum2 = 1.0;
-                    break;
-                }
-            }
-        /*    cell[col] = (CELL) (sum1/sum2 + .5);*/
-            dcell[col] = (DCELL) (sum1/sum2);
-        }
-        G_put_d_raster_row(fd,dcell);
-    }
-    G_close_cell(fd);
-    fprintf (stderr, "done__________\n");
-    exit(0);
-}
-
-void newpoint ( double z,double east,double north)
-{
-    if (npoints_alloc <= npoints)
-    {
-        npoints_alloc += 128;
-        points = (struct Point *) G_realloc (points,
-                    npoints_alloc * sizeof (struct Point));
-    }
-    points[npoints].north = north;
-    points[npoints].east  = east;
-    points[npoints].z     = z;
-    npoints++;
 }
Only in cmd.old: read_cell.c
diff -u cmd.old/read_sites.c cmd/read_sites.c
--- cmd.old/read_sites.c        2004-05-01 19:41:40.753138614 +0100
+++ cmd/read_sites.c    2004-02-16 12:13:59.755938848 +0000
@@ -1,4 +1,5 @@
 #include <stdio.h>
+#include <stdlib.h>
 #include "gis.h"
 #include "site.h"
 #include "proto.h"
@@ -16,6 +17,7 @@
     FILE *fd;
     int n, c, i, d;
     Site *site;
+    extern long npoints;

     field -= 1;  /* field number -> array index */

@@ -48,10 +50,12 @@
     while (G_site_get(fd, site) >= 0)
     {
        newpoint(site->dbl_att[field],site->east,site->north);
+        if(!(npoints%1000))
+            fprintf(stderr,"%10ld sites\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b", npoints);
     }
     fclose (fd);
-    fprintf (stderr, "\n");
+    fprintf (stderr, "%10ld sites loaded\n", npoints);
 }
```

Appendix D

Position Estimation

D.1 GAMIT track module kinematic GPS operation

The following listing is a command file for the 'track' kinematic GPS processing program. Comments within the file explain how it should be used.

```
# Template command file for Kinematic GPS Analysis using GAMIT 'track'
# PK Feb 2003
# N.B. Every command *must* have a blank line after it
#
# Set mode to SHORT at bottom of this file and then Run program as
# track -f track.cmd | tee track.log
# then
# grep FINAL track.log > amb.in
# Edit amb.in and in 'Fixd' column change any 1s to 7s (floating -->
# force fix), then change mode to LONG at bottom of this file and do
# track -f track.cmd -a amb.in | tee track.log
# Final results in track.GEOD.????.LC
# After that
# cat track.GEOD.mobl.LC | awk 'NR>2 {printf "%s\t%s\t%s\t%f\n", $5, $4, $6, ($10/1000)}' > gpsoutput.txt
# Will give nice latitude, longitude, ellps height, rms error in an ASCII file
#
# Site name (4 chars can be anything), RINEX filename, F=fixed K=kinematic
# Max of 5 sites (including fixed) can be processed in one go
 obs_file
    base  base0220.03o  F
    mobl  mobl0220.03o  K

# SP3 file must be for the correct day or can use RINEX ephemeris file e.g.
# nav_file  t0000220.03n  NAV
 nav_file  igs12023.sp3  SP3

# Use largest sampling interval (secs) in all RINEX files
 interval 1

# Settings for Leica AT502 Antenna:
# (Get dUp offset for base station from RINEX file header)
#          Antenna Ref Pt          L1 Phase Ctr        L2 Phase Ctr
#          dN     dE    dUp      dN     dE    dUp      dN     dE    dUp
 ante_off
   base  .0000  .0000 1.3750  .0003  .0020  .0618 -.0014  .0018  .0654
   mobl  .0000  .0000  .0000  .0003  .0020  .0618 -.0014  .0018  .0654
#   t003  .0000  .0000  .0000  .0003  .0020  .0618 -.0014  .0018  .0654
# Values from RINEX file? below
# ante_off
#   t000  .0000  .0000 1.3750  .0000  .0000  .0683  .0000  .0000  .0712
#   t011  .0000  .0000  .0000  .0000  .0000  .0683  .0000  .0000  .0712
#   t012  .0000  .0000  .0000  .0000  .0000  .0683  .0000  .0000  .0712

# Kinematic unit roves more than 1 km from base; if less use
# mode SHORT
```

```
 mode LONG

 back_type SMOOTH

# Output Geodetic co-ords (WGS84?) and northing/easting/up offsets from base
 out_type GEOD+NEU
```

D.2 Tokin YPR Sensor software modifications

The following unified diff file shows the changes made to the supplied software to save the YPR data and bearing to a text file at the camera frame rate.

```
Index: main.cpp
===================================================================
RCS file: /indigo-disk2/cvs/yprsensor/main.cpp,v
retrieving revision 1.1
retrieving revision 1.6
diff -u -r1.1 -r1.6
--- main.cpp    19 Jul 2002 10:35:22 -0000      1.1
+++ main.cpp    23 Jan 2003 11:48:55 -0000      1.6
@@ -1,3 +1,4 @@
+#define  _WIN32_WINNT  0x0500  /* Needed for timer functions PK 17 Jan 2003 */
 #include <windows.h>
 #include <tchar.h>
 #include <iostream.h>
@@ -10,9 +11,25 @@
 #include "TkInput.h"
 #include "TkInputDef.h"

+#include <time.h>
+
+#define RAD_TO_DEG      57.29577951308232
+#define DEG_TO_RAD      .0174532925199432958

 int main( int argc, char *argv[] )
 {
+       /* PK 19 July 2002 file pointer and filename for ypr text file */
+       FILE *fileptr;
+       char filename[80];
+    register double current;
+       double start;
+       int timemillisecs;
+       HANDLE hTimer = NULL;           /* Variables for windows timer */
+       LARGE_INTEGER li = { 0 };
+       float partsecond, partincrement;
+       double yaw, pitch, roll, bearing;
+       char message[50];
+
     cout << "Initializing COM" << endl;

     if ( FAILED( CoInitialize( NULL )))
@@ -83,30 +100,89 @@

         TK3DISTATE Tk3dIState;

-        for( i=0; i<100; i++ )
-        {
-                Sleep(1);
-                ZeroMemory( &Tk3dIState, sizeof(TK3DISTATE) );
-                pTk3dInput->GetDeviceData( 0, &Tk3dIState, &retinfo );
-                printf("%03d:GetDeviceData(0):retonfo = 0x%x\n",i, retinfo);
-                if( retinfo == 0x00 )
-                {
-                        printf("         Alfa=%f,Beta=%f,Gamma=%f,Button=0x...")
-                        printf("         ubFlag=0x%x,wRoll=0x%x,wPitch=0x%x...")
-                        printf("wLeanRL=0x%x,wMoveFB=0x%x,wMoveRL=0x%x,uwB...")
-                }
-/*              ZeroMemory( &Tk3dIState, sizeof(TK3DISTATE) );
-                pTk3dInput->GetDeviceData( 1, &Tk3dIState, &retinfo );
-                printf("%03d:GetDeviceData(1):retonfo = 0x%x\n",i, retinfo);
```

```
-                if( retinfo == 0x00 )
-                {
-                        printf("          Alfa=%f, Beta=%f, Gamma=%f, Button=0x...")
-                        printf("          ubFlag=0x%x, wRoll=0x%x, wPitch=0x%x...")
-                        printf("wLeanRL=0x%x, wMoveFB=0x%x, wMoveRL=0x%x, wwB...")
+    /* PK 19 July 2002 Ask for filename */
+       fprintf(stderr,"Enter name of ASCII file for ypr values: ");
+    gets(filename);
+    fprintf(stderr,"\n");
+       /* Cancel if enter was pressed without typing anything */
+    if(filename[0] != '\0'){
+       /* but return with error if there is a problem opening the file */
+         if(NULL == (fileptr = fopen(filename, "w")))
+        return (-1);
+
+                /* Time to wait between polling ypr device
+                 * Modified to use Windows timer functions 17 Jan 2003 PK */
+                timemillisecs = 40;
+                li.QuadPart = -(timemillisecs*10000);
+                partincrement = (float)timemillisecs / 1000;
+                partsecond = 0;
+
+         /* Some of the code below is copied from the GPS DECCO8c.cpp file */
+
+                hTimer = CreateWaitableTimer(NULL, TRUE, NULL);
+                current = start = 0;
+                SetWaitableTimer(hTimer, &li, 0, NULL, NULL, 0);
+
+                while(1)
+                {   /*first do loop encompasses the whole program making it repeat */
+                        /* Wait for length of time set above in timemillisecs */
+                        WaitForSingleObject(hTimer, INFINITE);
+                        SetWaitableTimer(hTimer, &li, 0, NULL, NULL, 0);
+
+                        current += partincrement;
+
+                    ZeroMemory( &Tk3dIState, sizeof(TK3DISTATE) );
+                        pTk3dInput->GetDeviceData( 0, &Tk3dIState, &retinfo );
+                        if( retinfo == 0x00 )
+                        {
+                                /* Assume direction pointed forward by Y on white box is front */
+                                yaw = -Tk3dIState.Alfa*RAD_TO_DEG;
+                                pitch = Tk3dIState.Gamma*RAD_TO_DEG;
+                                roll = Tk3dIState.Beta*RAD_TO_DEG;
+                                bearing = Tk3dIState.Alfa_North;
+                                bearing = -(Tk3dIState.Alfa_North*RAD_TO_DEG + 90);
+                                if (bearing < -360)
+                                        bearing = bearing + 720;
+                                else if (bearing < 0)
+                                        bearing = bearing + 360;
+                                sprintf(message, "");
+                                if ( abs((int)yaw) > 180 )
+                                        sprintf(message, "Warning: Yaw range exceeded");
+                                if ( abs((int)pitch) > 60 )
+                                        sprintf(message, "Warning: Pitch range exceeded");
+                                if ( abs((int)roll) > 60 )
+                                        sprintf(message, "Warning: Roll range exceeded");
+                        /* PK 19 July 2002 Put simply yaw, pitch and roll angles in text file;
+                         * leave screen output as it is */
+                                fprintf(fileptr, "%.2f %.2f %.2f %.2f %.2f\n", current, yaw,
+                                        pitch, roll, bearing);
+                                printf("%03d degs %+.2f(Y) %+.2f(P) %+.2f(R) %s %.2f\n",
+                                       (int)(bearing+0.5), yaw, pitch, roll, message, current);
+                        }
+
+
+                        if( (long)current == (long)(start + 10) ) /*Should be 1200 for 20mins */
+                        {
+       /* PK 19 July 2002 Now saves every 10 seconds no effect on performance on
+       * fast PC and allows for laptop battery dying---will never lose more than 10
+       * seconds of data */
+       /* 9th Feb 99 closing stream so saves every 20 mins */
+                                fclose(fileptr);
+                    printf("\n Closed file then...");
+                                printf("\n definitely closed \n");
```

```
+                                   fileptr = fopen(filename,"a");
+                                   printf("opened again \n");
+                                   start = current;
+
+                           }
+
                    }
-*/
-           }
-
+
+
+
+       }
+       else
+                   /* Get here if enter was pressed without typing in a filename */
+            fprintf(stderr,"<request cancelled>\n");
+
           // The completion of driver
           pTk3dInput->CloseDevice( &retinfo );
           printf("CloseDevice : retinfo = 0x%x\n",retinfo);
```

D.3 Modified Hough Transform

Here is a listing of the MATLAB script the modified hough transform was implemented in.

```
points=[-0.4572 3.4728 0
0.4572 3.4728 0
-0.4572 4.3872 0
0.4572 4.3872 0
-0.4572 5.3016 0
0.4572 5.3016 0
-0.4572 6.216 0
0.4572 6.216 0
-0.4572 7.1304 0
0.4572 7.1304 0];
%-0.4572 8.0448 0
%0.4572 8.0448 0
%-0.4572 8.9592 0
%0.4572 8.9592 0
%-0.4572 9.8736 0
%0.4572 9.8736 0];
temp=size(points);
numpoints=temp(1);

impoints=[492   269
   492   529
   452   301
   451   502
   426   318
   426   484
   408   331
   407   473
   396   339
   395   464];
temp=size(impoints);
numimpoints=temp(1);

% x y z position of camera
% GIS co-ordinate system x is east y is north and z is up
%campos=[352583.536 ;336373.31; 17];
% rotations about x y z axes respectively
% Relative camera initially horizontal and pointing north (along y-axis)
% So order of angles actually is pitch (up +ve), roll (clckwse +ve), yaw
% (+ve to the left)
%camypr=[0 0 pi/2];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Camera Interior Orientation Global Variables
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
global imagesize ccdsize xypp fl
% Image size rows by columns (768 by 576 for PAL w/ square pixels)
imagesize=[576 768];
% CCD dimensions (1/3 inch is 3.2 by 2.4 mm)
ccdsize=[3.2e-3 2.4e-3];
% xy co-ordinates of camera principal point
xypp=ccdsize./2;
% focal length
fl=[3.45e-3];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

options=optimset('Display','off');
boundfact=2;
others=[0 0 0.8 0 0 0]
oldothers=others+[0.1 0.1 0.1 0.1 0.1 0.1];
accumsize=[50 50];
%absl b=[-2 -2 0 -0.2 -0.2 -0.2];
%absub=[2 2 3 0.2 0.2 0.2];
abslb=[-inf -inf -inf -inf -inf -inf];
absub=[inf inf inf inf inf inf];
validbound=0.05;

iters=0;
while( max(abs(others-oldothers)) > 0.01 )
   iters=iters+1
boundfact = boundfact.*0.9;
accumsize = accumsize+[10 10];

for count=4:6
  if others(count) > 2*pi others(count) = mod(others(count),(2*pi)); end
  if others(count) < -2*pi others(count) = mod(others(count),(-2*pi)); end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Project and Display %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%
%projanddisp

%%%%% x y approximations

x0=others(1:2);
sizex0=size(x0);
whichones = 1;
lb=x0 + [-5 -5].*boundfact;
ub=x0 + [5 5].*boundfact;

switch sizex0(2)
  case 1
    resol=(ub-lb)./accumsize(1);,...
    accum=zeros(accumsize(1));
  case 2
    resol=(ub-lb)./[accumsize(1) accumsize(2)];,...
    accum=zeros(accumsize(1),accumsize(2));
end
ptsfile=fopen('ptsfile.txt','w');
warning off
for gndpntcount=1:numpoints
  for impntcount=1:numimpoints

    x=fmincon('distance',x0,[],[],[],[],lb,ub,'zbuffer',options,whichones,...
      others,points(gndpntcount,:),impoints(impntcount,:));
    fprintf(ptsfile, '%.2f %.2f\n', x(1), x(2));
    switch sizex0(2)
      case 1
         accumindex = ceil((x-lb)./resol);
         if accumindex == 0 accumindex = 1; end
         if accumindex > accumsize(1) accumindex = accumsize(1); end
        accum(accumindex) = accum(accumindex) + 1;
      case 2
        accumindex = ceil((x-lb)./resol);
        if accumindex(1) == 0 accumindex(1) = 1; end
        if accumindex(2) == 0 accumindex(2) = 1; end
        if accumindex(1) > accumsize(1) accumindex(1) = accumsize(1); end
```

```
        if accumindex(2) > accumsize(2) accumindex(2) = accumsize(2); end
        accum(accumindex(1),accumindex(2)) = ...
                                    accum(accumindex(1),accumindex(2)) + 1;
    end
  end
end
warning on
fclose(ptsfile);
figure
switch sizex0(2)
  case 1
    plot(accum)
    [r]=find(accum==max(max(accum)));
  case 2
    mesh(accum)
    [r c]=find(accum==max(max(accum)));
end

others(1)=(mean(r)-0.5)*resol(1)+lb(1);
others(2)=(mean(c)-0.5)*resol(2)+lb(2);
for count=1:2
  if abslb(count)>others(count)
    others(count)=abslb(count);
  end
  if absub(count)<others(count)
    others(count)=absub(count);
  end
end
others
%projanddisp
%%%%% z approximations

x0=others(3);
sizex0=size(x0);
whichones = 2;
lb=x0 + [-5].*boundfact;
ub=x0 + [5].*boundfact;


%%%%%%%%% independent

switch sizex0(2)
  case 1
    resol=(ub-lb)./accumsize(1);,...
    accum=zeros(accumsize(1));
  case 2
    resol=(ub-lb)./[accumsize(1) accumsize(2)];,...
    accum=zeros(accumsize(1),accumsize(2));
end

warning off
for gndpntcount=1:numpoints
  for impntcount=1:numimpoints

    x=fmincon('distance',x0,[],[],[],[],lb,ub,'zbuffer',options,whichones,...
      others,points(gndpntcount,:),impoints(impntcount,:));

    switch sizex0(2)
      case 1
         accumindex = ceil((x-lb)./resol);
         if accumindex == 0 accumindex = 1; end
         if accumindex > accumsize(1) accumindex = accumsize(1); end
        accum(accumindex) = accum(accumindex) + 1;
      case 2
        accumindex = ceil((x-lb)./resol);
        if accumindex(1) == 0 accumindex(1) = 1; end
        if accumindex(2) == 0 accumindex(2) = 1; end
        if accumindex(1) > accumsize(1) accumindex(1) = accumsize(1); end
        if accumindex(2) > accumsize(2) accumindex(2) = accumsize(2); end
        accum(accumindex(1),accumindex(2)) = ...
                                    accum(accumindex(1),accumindex(2)) + 1;
    end
  end
end
warning on
```

```
figure
switch sizex0(2)
  case 1
    plot(accum)
    [r]=find(accum==max(max(accum)));
  case 2
    mesh(accum)
    [r c]=find(accum==max(max(accum)));
end

%%%%%%%%%%%%% end independent
%Converting from accumulator position back into z co-ordinate work all right?
others(3)=(mean(r)-0.5)*resol+lb;
for count=3:3
  if abslb(count)>others(count)
    others(count)=abslb(count);
  end
  if absub(count)<others(count)
    others(count)=absub(count);
  end
end
others
%projanddisp
end
```

The following two MATLAB functions, distance and zbuffer, are used within the above script.

distance.m

```
function [dist]=distance(curroptim, whichones, others, gndpoint, impoint);

global xypp fl ccdsize imagesize

campos = others(1:3)';
campypr = others(4:6);

switch whichones
  case 1 % x & y
    campos(1) = curroptim(1);, campos(2) = curroptim(2);
  case 2 % z
    campos(3) = curroptim(1);
  case 3 % rotx & roty
    campypr(1) = curroptim(1);, campypr(2) = curroptim(2);
  case 4 % rotz
    campypr(3) = curroptim(1);
end

sinypr=sin(campypr);
cosypr=cos(campypr);
rotmat=[cosypr(2)*cosypr(3) (sinypr(1)*sinypr(2)*cosypr(3) - ...
cosypr(1)*sinypr(3)) (cosypr(1)*sinypr(2)*cosypr(3) + sinypr(1)*sinypr(3));...
cosypr(2)*sinypr(3) (sinypr(1)*sinypr(2)*sinypr(3) + ...
cosypr(1)*cosypr(3)) (cosypr(1)*sinypr(2)*sinypr(3) - sinypr(1)*cosypr(3));...
-sinypr(2) sinypr(1)*cosypr(2) cosypr(1)*cosypr(2)];

% NB backslash matrix left divide as the camera ypr angles are relative to
% the world space but we are rotating the points into the camera space
xyzrel = rotmat\(gndpoint' - campos);

% position of point in CCD xy values with lower left as origin
xyim = xypp + (fl./xyzrel(2)).*[xyzrel(1) xyzrel(3)];

% convert to pixel co-ordinate system upper left origin, centre of first
% pixel at (1,1)
uv = [0.5 0.5] + ([(ccdsize(2) - xyim(2)) xyim(1)]./...
[ccdsize(2) ccdsize(1)]).*imagesize;

error = impoint - uv;
dist = error(1)^2 + error(2)^2;
```

zbuffer.m

```
function [c, ceq]=zbuffer(curroptim, whichones, others, gndpoint, impoint);

global xypp fl ccdsize imagesize

campos = others(1:3)';
campypr = others(4:6);

switch whichones
  case 1 % x & y
    campos(1) = curroptim(1);, campos(2) = curroptim(2);
  case 2 % z
    campos(3) = curroptim(1);
  case 3 % rotx & roty
    campypr(1) = curroptim(1);, campypr(2) = curroptim(2);
  case 4 % rotz
    campypr(3) = curroptim(1);
end

sinypr=sin(campypr);
cosypr=cos(campypr);
rotmat=[cosypr(2)*cosypr(3) (sinypr(1)*sinypr(2)*cosypr(3) - ...
cosypr(1)*sinypr(3)) (cosypr(1)*sinypr(2)*cosypr(3) + sinypr(1)*sinypr(3));...
cosypr(2)*sinypr(3) (sinypr(1)*sinypr(2)*sinypr(3) + ...
cosypr(1)*cosypr(3)) (cosypr(1)*sinypr(2)*sinypr(3) - sinypr(1)*cosypr(3));...
-sinypr(2) sinypr(1)*cosypr(2) cosypr(1)*cosypr(2)];

% NB backslash matrix left divide as the camera ypr angles are relative to
% the world space but we are rotating the points into the camera space
xyzrel = rotmat\(gndpoint' - campos);

% This is transformed so if zbuff is <= 0 the constraint is satisfied
c = fl - xyzrel(2);
ceq=[];
```

Appendix E

Colour and Illumination

E.1 Flattened Hue

This MATLAB function is used to create a flattened hue representation. The two versions of code shown in the file are also a good example of how MATLAB operations may be greatly speeded up by using vector notation rather than for loops.

flathue.m

```
%flathue=zeros(size(hue));
% New improved vectorised method 24/10/02 PK
% for hue = 2*pi floor(hue./pi) will be 2 instead of 1 but this doesn't matter
% as ((2(pi)-hue) will then be 0 so 2 by 0 is still 0
flathue=floor(hue./pi) .* ((2*pi)-hue) + ~floor(hue./pi) .* hue;
%for row=1:rows
%for column=1:columns
%if hue(row,column)>pi
%flathue(row,column) = (2*pi)-hue(row, column);
%else
%flathue(row,column) = hue(row,column);
%end
%end
%end
normflathue=uint8(round(flathue.*(255/pi)));
figure,imshow(normflathue)
```

References

Aber, J. S., Aber, S. W. and Pavri, F., 2002. Unmanned small-format aerial photography from kites for acquiring large-scale, high-resolution, multiview-angle imagery. In: ISPRS Commission I Mid-term Symposium.

Agouris, P., Mountrakis, G. and Stefanidis, A., 2000. Automated spatiotemporal change detection in digital aerial imagery. In: Proceedings of SPIE, Vol. 4054, pp. 2–12.

Ali, M. and Aggarwal, J. K., 1977. Automatic interpretation of infrared aerial color photographs of citrus orchards having infestations of insect pests and diseases. IEEE Transactions on Geoscience Electronics 15(3), pp. 170–179.

Anderson, K., 2000. An introduction to the facilities of the NERC equipment pool for field spectroscopy (NERC EPFS). In: Activities of the NERC EPFS in support of the NERC ARSF. ARSF Annual Meeting, Keyworth, Nottingham, UK. http://www.soton.ac.uk/~epfs/resources/pdf_pubs/EPFS.pdf.

Anderson, S. O., Simmons, R. and Goldberg, D., 2003. Maintaining line of sight communications networks between planetary rovers. In: Proceedings of the 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems, Vol. 3, pp. 2266–2272.

Anderson, T. K., 1991. William Alfred Green FRSAI: the man and his photographs. Ulster Local Studies.

Ballard, D. H., 1981. Generalizing the hough transform to detect arbitrary shapes. Pattern Recognition 13(2), pp. 111–122.

Bertozzi, M., Broggi, A. and Fascioli, A., 1998a. An extension to the inverse perspective mapping to handle non-flat roads. In: Proceedings of the IEEE Intelligent Vehicles Symposium, pp. 505–510.

Bertozzi, M., Broggi, A. and Fascioli, A., 1998b. Stereo inverse perspective mapping: Theory and applications. Image and Vision Computing 16(8), pp. 585–590.

Birchfield, S., 2003. KLT website. http://vision.stanford.edu/~birch/klt/.

Bossler, J. C., Goad, C., Johnson, P. and Novak, K., 1991. GPS and GIS map the nation's highways. GeoInfo Systems Magazine pp. 26–37.

Bouman, C. A. and Shapiro, M., 1994. Multiscale random field model for bayesian image segmentation. IEEE Transactions on Image Processing 3(2), pp. 162–177.

Bruzzone, L. and Cossu, R., 2003. An adaptive approach to reducing registration noise effects in unsupervised change detection. IEEE Transactions on Geoscience and Remote Sensing 41(11), pp. 2455–2465.

Burrough, P. A., 1986. Principles of Geographical Information Systems for Land Resources Assessment. Oxford University Press, Oxford.

Butler, D. R., 1994. Repeat photography as a tool for emphasizing movement in physical geography. Journal of Geography 93(3), pp. 141–151.

Cambridge University, 2004. Cambridge university collection of air photos (CUCAP). http://www.uflm.cam.ac.uk/library_page.htm.

Campbell, J. B., 2002. Introduction to Remote Sensing. Third edn, Taylor & Francis, London.

Cebecauer, T., Hofierka, J. and Šúri, M., 2002. Processing digital terrain models by regularized spline with tension: Tuning interpolation parameters for different input datasets. In: Proceedings of the Open source GIS - GRASS users conference.

Chatterji, G. B., Menon, P. K. and Sridhar, B., 1998. Vision-based position and altitude determination for aircraft night landing. Journal of Guidance, Control, and Dynamics 21(1), pp. 84–92.

Cheetham, G., 1965. Textbook of Topographical Surveying. Fourth edn, HMSO, London.

Ciurea, F. and Funt, B., 2003. A large image database for color constancy research. In: Proceedings of the Color Imaging Conference: Color Science, Systems, and Applications, pp. 160–164.

Commission Internationale de l'Eclairage, 1931. Proceedings of the 8th session.

Conrady, A. E., 1960. Applied Optics and Optical Design. Vol. 2, Dover, New York.

Cox, S. J. D., 1995. s.to.cell: a program to partially populate a grid with values from a site-list for use with the grass gis. CSIRO Exploration & Mining Report 237F.

Cross, P. and Ziebart, M., 2002. GPS attitude determination algorithm designed for real-time on-board execution. In: Proceedings of ION GPS-2002, Portland, Oregon, USA.

Diner, D. J., Beckert, J. C., Bothwell, G. W. and Rodriguez, J. I., 2002. Performance of the misr instrument during its first 20 months in earth orbit. IEEE Transactions on Geoscience and Remote Sensing.

Dungan, W., 1979. A terrain and cloud computer image generation model. ACM SIGGRAPH 13(2), pp. 143–150.

Evans, S., 2002. Personal discussions. Discussion on available OSNI datasets.

Foley, J. D. and van Dam, A., 1982. Fundamentals of interactive computer graphics. Addison-Wesley, Reading, Mass., USA.

Forsberg, R., Strykowski, G., Iliffe, J. C., Ziebart, M., Cross, P. A., Tscherning, C. C., Cruddace, P., Stewart, K., Bray, C. and Finch, O., 2002. OSGM02: A new geoid model of the British Isles. Kort & Matrikelstyrelsen, Denmark.

FreeGIS Project, 2003. http://freegis.org/.

Gerdes, D. and Brown, B., 1992. SG3d reference manual. http://grass.ibiblio.org/gdp/html_grass5/html/SG3d.ref.

Gershon, R., Jepson, A. D. and Tsotsos, J. K., 1987. From R, G, B to surface reflectance: Computing color constancy descriptors in images. In: Proceedings of the 10th International Joint Conference on Artificial Intelligence, pp. 755–758.

Gülch, E. and Müller, H., 2001. New applications of semi-automatic building acquisition. In: 3rd Workshop on Automatic Extraction of Man-Made Objects from Aerial and Space Images, pp. 103–114.

Habib, A. and Kelley, D., 2001. Single-photo resection using the modified Hough transform. Photogrammetric Engineering and Remote Sensing 67(8), pp. 909–914.

Habib, A. F. and Alruzouq, R. I., 2004. Line-based modified iterated hough transform for automatic registration of multi-source imagery. Photogrammetric Record 19(105), pp. 5–20.

Harmon, V. and Shapiro, M., 1991. GRASS Image Processing Tutorial. http://grass.itc.it/gdp/html_grass5/Postscript/imagery.ps.

Hart, R. H. and Laycock, W. A., 1996. Repeat photography on range and forest lands in the western united states. Journal of Range Management 49, pp. 60–67.

Helferty, J. P., Zhang, C., McLennan, G. and Higgins, W. E., 2001. Videoendoscopic distortion correction and its application to virtual guidance of endoscopy. IEEE Transactions on Medical Imaging 20(7), pp. 605–617.

Hirano, A., Welch, R. and Lang, H., 2003. Mapping from ASTER stereo image data: DEM validation and accuracy assessment. ISPRS Journal of Photgrammetry & Remote Sensing 57, pp. 356–370.

Holland, K. T., Holman, R. A., Lippmann, T. C., Stanley, J. and Plant, N., 1997. Practical use of video imagery in nearshore oceanographic field studies. IEEE Journal of Oceanic Engineering 22(1), pp. 81–92.

Hubel, P. M., 2000. The perception of color at dawn and dusk. Journal of Imaging Science and Technology 44(4), pp. 371–375,387.

International Telecommunication Union, 1995. Recommendation BT.601-5 (10/95): Studio encoding parameters of digital television for standard 4:3 and wide-screen 16:9 aspect ratios. ITU, Geneva, Switzerland.

K2VI, 2004. K2VI—Key to Virtual Insight software. http://www.k2vi.com/.

Kelly, P. D. and Dodds, G., 2002. Image processing and GIS integration for environmental analysis. In: Irish Signals and Systems Conference, pp. 121–126.

Kornfeld, C., 2003. Personal discussions. Discussion on synchronising two DV cameras.

Krinov, E. L., 1953. Spectral reflectance properties of natural formations. National Research Council of Canada Technical Translation TT-439. Translated by G Belkov. Originally published in Russian 1947.

Lake, M. W., Woodman, P. E. and Mithen, S. J., 1998. Tailoring GIS software for archaeological applications: An example concerning viewshed analysis. Journal of Archaeological Science 25, pp. 27–38.

Lambert, P. and Carron, T., 1999. Symbolic fusion of luminance-hue-chroma features for region segmentation. Pattern Recognition 32(11), pp. 1857–1872.

Land, E. and McCann, J. J., 1971. Lightness and retinex theory. Journal of the Optical Society of America 61, pp. 1–11.

Laussedat, A., 1854. Mémoire sur l'emploi de la chambre clair dans les reconnaissances topographiques. Mémorial de l'Officier du Génie.

Laussedat, A., 1898. Recherches sur les instruments, les méthodes et le dessin topographiques. Vol. 1, Gauthier-Villars, Paris.

Lu, C.-P., Hager, G. D. and Mjolsness, E., 2000. Fast and globally convergent pose estimation from video images. IEEE Transactions on Pattern Analysis and Machine Intelligence 22(6), pp. 610–622.

Lunetta, R. S. and Elvidge, C. D. (eds), 1999. Remote Sensing Change Detection: Environmental Monitoring Methods and Applications. Taylor and Francis, London.

Lutzeler, M., Dickmanns, E. D. and Gregor, R., 2001. EMS-Vision: Combining on- and off-road driving. In: Proceedings of SPIE, Vol. 4364, pp. 429–440.

Madhavan, R. and Durrant-Whyte, H. F., 2004. Natural landmark-based autonomous vehicle navigation. Robotics and Autonomous Systems 46(2), pp. 79–95.

Magee, J., 1991. A Journey through Lecale. Friar's Bush Press, Belfast.

Mai, W. J., Dodds, G. and Tweed, C., 2003. A PDA-based system for recognizing buildings from user-supplied images. In: HCI03–10th International Conference on Human–Computer Interaction, pp. 143–157.

McCauley, J. D., 1993. Grass tutorial on s.sample. http://grass.itc.it/gdp/sites/s.sample-tutorial.ps.gz.

McCauley, J. D. and Engel, B. A., 1995. Comparison of scene segmentations: SMAP, ECHO, and maximum likelihood. IEEE Transactions on Geoscience and Remote Sensing 33(6), pp. 1313–1316.

McMenemy, K., 2002. Photometric Evaluation of Aerodrome Ground Lighting. PhD thesis, Queen's University Belfast, Department of Electrical and Electronic Engineering.

McMenemy, K., Dodds, G. and Mullin, F., 2001. In-situ guidance lighting performance measurement. In: Illuminating Engineering Society of North America, Annual Conference, pp. 387–398.

McMenemy, K., Mullin, F. and Dodds, G., 2003. Calibration and use of video cameras in the photometric assessment of aerodrome ground lighting. In: Proceedings of SPIE, Vol. 5017, pp. 104–115.

Meneely, J. and Graham, C., 2002. Personal discussions. Discussion on available OSNI datasets.

Mercator GPS Systems, 1998. QUEST GPS tutor. http://www.mercat.com/QUEST/gpstutor.htm.

Mezei, L. M., 1990. Practical Spreadsheet Statistics and Curve Fitting for Scientists and Engineers. Prentice-Hall.

Mitas, L. and Mitasova, H., 1998. Distributed soil erosion simulation for effective erosion prevention. Water Resources Research 34(3), pp. 505–516.

Mitasova, H., 2002. Personal correspondence. Discussion on extending the 3-D visualisation capabilities of GRASS GIS.

Mitasova, H., Mitas, L., Brown, W. M., Gerdes, D. P., Kosinovsky, I. and Baker, T., 1995. Modeling spatially and temporally distributed phenomena - new methods and tools for GRASS GIS. International Journal of Geographical Information Systems 9(4), pp. 433–446.

Mitášŏvá, H. and Mitáš, L., 1993. Interpolation by regularized spline with tension: I. Theory and Implementation. Mathematical Geology 25(6), pp. 641–655. (http://skagit.meas.ncsu.edu/˜helena/gmslab/papers/lmg.rev1.ps).

Moellering, H., 1980. The real-time animation of three-dimensional maps. American Cartographer 7(1), pp. 67–75.

Morgenthaler, D. G., Hennessy, S. J. and deMenthon, D., 1990. Range-video fusion and comparison of inverse perspective algorithms in static images. IEEE Transactions on Systems, Man, and Cybernetics 20(6), pp. 1301–1312.

Mostafa, M. M. R., 2001. History of inertial navigation systems in survey applications (editorial article). Photogrammetric Engineering and Remote Sensing.

Munn, R. E. (ed.), 1979. Environmental Impact Assessment Principles and Procedures. Scientific Committee on Problems of the Environment (SCOPE).

Neteler, M. and Mitasova, H., 2002. Open Source GIS: A GRASS GIS Approach. Kluwer Academic Publishers, Dordrecht, Netherlands.

Newman, W. M. and Sproull, R. F., 1979. Principles of Interactive Computer Graphics. Second edn, McGraw-Hill Book Company, New York.

NorthWest Research Associates Inc., 2001. Video Metric Systems Argus Beach Monitoring Stations. http://www.planetargus.com/.

Novak, K., 1995. Mobile mapping technology for GIS data collection. Photogrammetric Engineering and Remote Sensing 61(5), pp. 493–501.

Ohta, Y., Kanade, T. and Sakai, T., 1980. Color information for region segmentation. Computer Graphics and Image Processing 13(3), pp. 222–241.

Ordnance Survey, 2002. A guide to coordinate systems in Great Britain. OSGB, Southampton. http://www.gps.gov.uk/guidecontents.asp.

OSi and OSNI, 2000. The Irish Grid: A Description of the Co-ordinate Reference System. OSi, Dublin and OSNI, Belfast. (http://www.osni.gov.uk/downloads/grid.pdf).

O'Sullivan, D. and Unwin, D., 2003. Geographic Information Analysis. John Wiley & Sons, Inc., New Jersey, USA.

Perkins, C. R. and Parry, R. B., 1996. Mapping the UK. Bowker-Saur, London.

Pointer, M. R., Attridge, G. G. and Jacobson, R. E., 2001. Practical camera characterisation for colour measurement. In: Final Program and Proceedings: IS and T's 54th Annual Conference, pp. 246–251.

RemoteSensing.org, 2004. PROJ.4 website. http://remotesensing.org/proj/.

Rogers, R. M., Wit, J. S., Crane, C. D. and Armstrong, D. G., 1996. Integrated INU/DGPS for autonomous vehicle navigation. In: IEEE PLANS, Position Location and Navigation Symposium, pp. 417–476.

Russ, J. C., 1994. The Image Processing Handbook. Second edn, CRC Press, Inc., Boca Raton, Florida, USA.

Sandmeier, S. R., 2000. Acquisition of bidirectional reflectance factor data with field goniometers. Remote Sensing of Environment 73(3), pp. 257–269.

Sangwine, S. J. and Horne, R. E. N. (eds), 1998. The Colour Image Processing Handbook. Chapman & Hall, London.

Schilling, R. J., 1990. Fundamentals of Robotics Analysis and Control. Prentice-Hall International, Englewood Cliffs, New Jersey, USA.

Schroeder, M., 2003. Grass r.proj manual page. http://grass.itc.it/gdp/html_grass5/html/r.proj.html.

Shapiro, M., 2003. Grass s.in.ascii manual page. http://grass.itc.it/gdp/html_grass5/html/s.in.ascii.html.

Sheng, Y., Gong, P. and Biging, G. S., 2003. True orthoimage production for forested areas from large-scale aerial photographs. Photogrammetric Engineering and Remote Sensing 69(3), pp. 259–266.

Shi, J. and Tomasi, C., 1994. Good features to track. In: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pp. 593–600.

Slaughter, D. C. and Harrell, R. C., 1989. Discriminating fruit for robotic harvest using color in natural outdoor scenes. Transactions of the ASAE 32(2), pp. 757–763.

Smith, S. W., 1997. The Scientist and Engineer's Guide to Digital Signal Processing. California Technical Publishing.

Snyder, J. P., 1987. Map projections: A working manual. U.S. Geological Survey Professional Paper 1395.

Sproson, W. N., 1978. Pal system I phosphor primaries: the present position. Proceedings of the Institution of Electrical Engineers 125(6), pp. 603–605.

Sproson, W. N., 1983. Colour Science in Television and Display Systems. Adam Hilger Ltd., Bristol.

Stauder, J., Mech, R. and Ostermann, J., 1999. Detection of moving cast shadows for object segmentation. IEEE Transactions on Multimedia 1(1), pp. 65–76.

Strausz, D. A., 2001. Application of photogrammetric techniques to the measurement of historic photographs. Student Research Paper, Oregon State University. http://oregonstate.edu/instruct/geo422/522nofig.pdf.

Tanaka, S. and Suga, Y., 1979. Landscape drawings from Landsat MSS data. Photogrammetric Engineering and Remote Sensing 45(10), pp. 1345–1351.

Tao, C. V., Chapman, M. A. and Chaplin, B. A., 2001. Automated processing of mobile mapping image sequences. ISPRS Journal of Photogrammetry & Remote Sensing 55, pp. 330–346.

The MathWorks, Inc., 1997. MATLAB Image Processing toolbox User's Guide. The MathWorks, Inc.

TheFreeDictionary.com, 2004. Encyclopaedia article on 'world file'. http://encyclopedia.thefreedictionary.com/World%20file.

Thompson, M. M. (ed.), 1966. Manual of Photogrammetry. Third edn, American Society of Photogrammetry, Falls Church, Virginia, USA.

Todd, R., 1990. Measurement of luminance to BS5480 Part 2. The Lighting Journal pp. 241–244.

Transmap Corp., 2003. Transmap mobile mapping solutions. http://www.transmap.com/.

Tsai, R. Y., 1987. A versatile camera calibration technique for high-accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses. IEEE Journal of Robotics and Automation RA-3(4), pp. 323–344.

Tseng, Y. and Wang, S., 2003. Semiautomated building extraction based on CSG model-image fitting. Photogrammetric Engineering and Remote Sensing 69(2), pp. 171–180.

Tsukada, M. and Ohta, Y., 1990. An approach to color constancy using multiple images. In: Proceedings of the 3rd International Conference on Computer Vision, pp. 385–389.

University of California, 2004. CIE standard datasets website. http://cvision.ucsd.edu/cie.htm.

Uren, J. and Price, W. F., 1994. Surveying for Engineers. Third edn, MacMillan Press, London.

Walter-Shea, E. A., Hays, C. J., Mesarch, M. A. and Jackson, R. D., 1993. Improved goniometer system for calibrating field reference-reflectance panels. Remote Sensing of Environment 43(2), pp. 131–138.

Weinhaus, F. M. and Devarajan, V., 1997. Texture mapping 3D models of real-world scenes. ACM Computing Surveys 29(4), pp. 325–365.

Welch, R., Jordan, T., Lang, H. and Murakami, H., 1998. ASTER as a source for topographic data in the late 1990s. IEEE Transactions on Geoscience and Remote Sensing 36(4), pp. 1282–1289.

Williams, S. B., Dissanayake, G. and Durrant-Whyte, H., 2002. Towards multi-vehicle simultaneous localisation and mapping. In: IEEE International Conference on Robotics and Automation, pp. 2743–2748.

Woebbecke, D. M., Meyer, G. E., Von Bargen, K. and Mortensen, D. A., 1995. Color indices for weed identification under various soil, residue and lighting conditions. Transactions of the ASAE 38(1), pp. 259–269.

Woo, M., 1997. OpenGL programming guide : the official guide to learning OpenGL. Addison Wesley, Reading, Mass.

Wood, J., 1996. The Geomorphological Characterisation of Digital Elevation Models. PhD thesis, University of Leicester, Department of Geography.

Wyszecki, G. and Stiles, W. S., 1982. Color Science. John Wiley & Sons, Inc., New York.

Zatari, A., McMenemy, K. and Dodds, G., 2003. Glare, luminance and illuminance measurements of road lighting using vehicle-mounted CCD cameras. In: Illuminating Engineering Society of North America, Annual Conference.