Environnement Canada Environment Canada

Canada

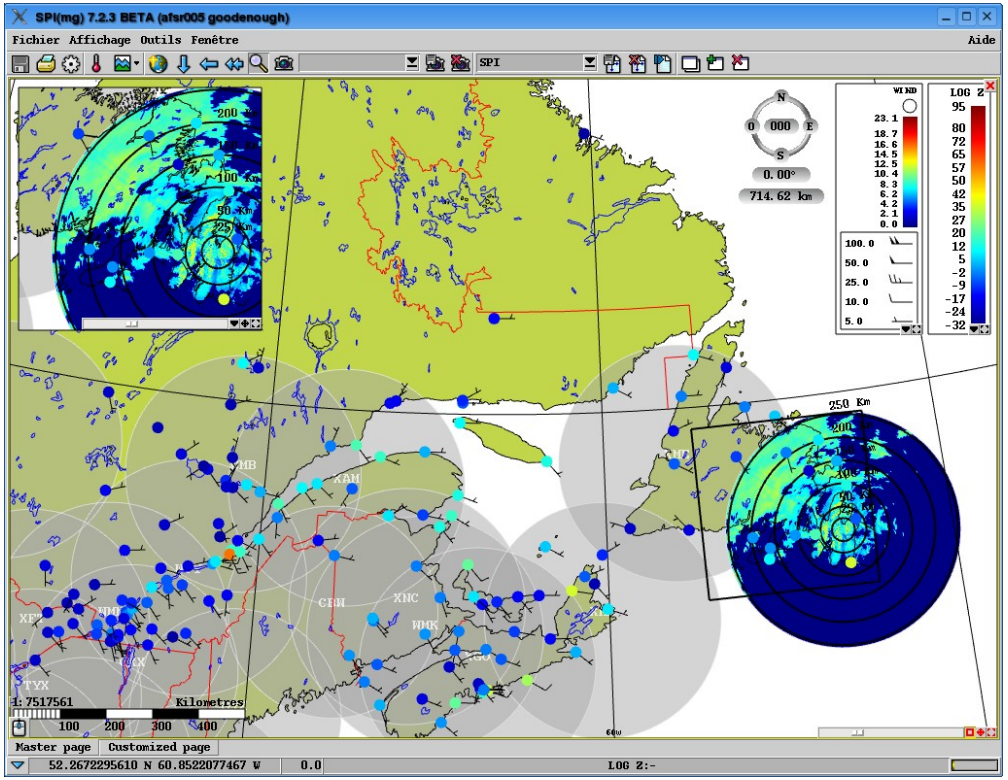
SPI (et API)

aperçu,

nouveautés

et

exemples



Le plan !

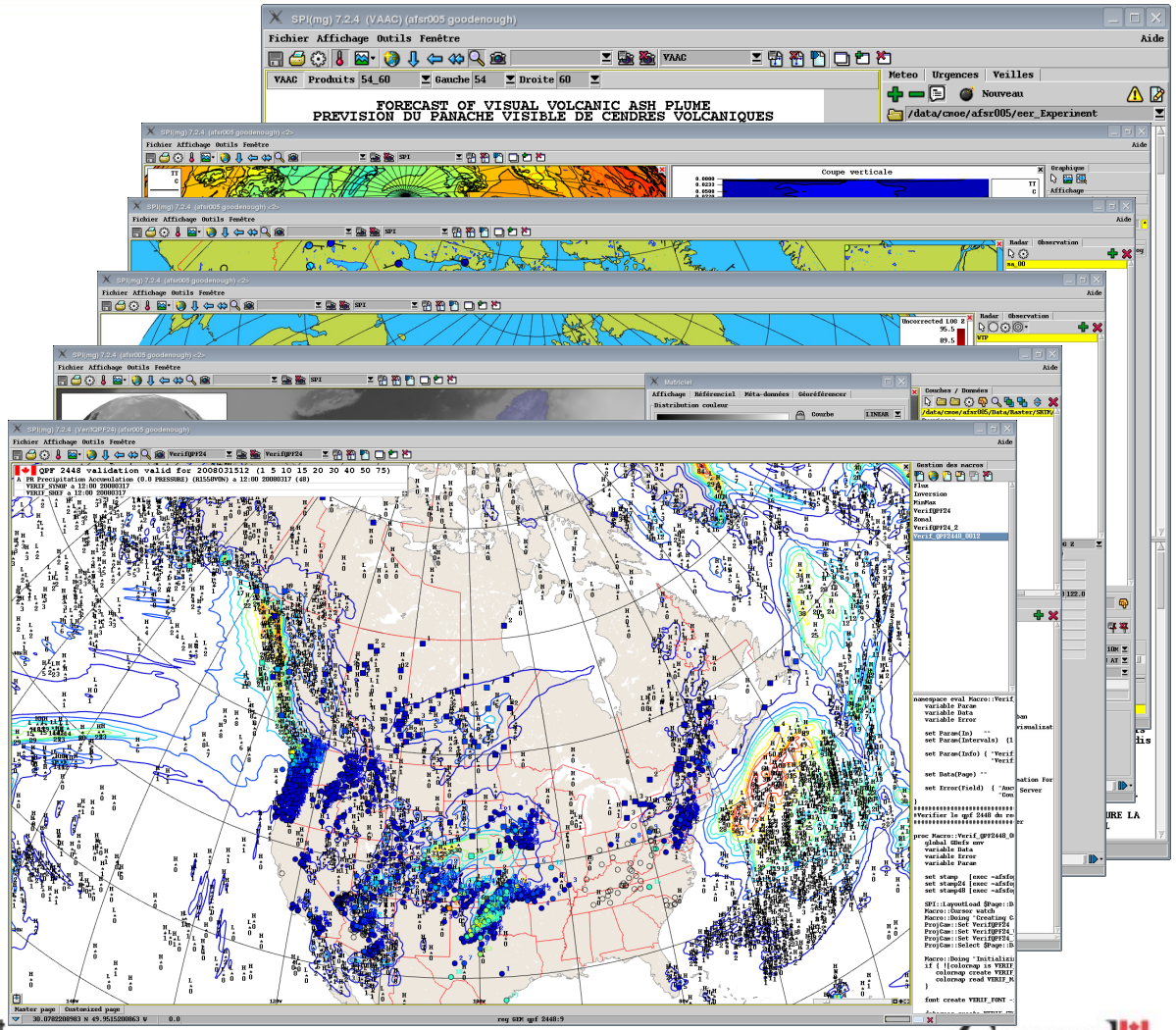
- SPI
  - C’est quoi
  - Exemples d’utilisations
  - Hands on
  - Quoi de plus que …
- API
  - Description
  - Exemples
- Projets connexes
- Suite …

SPI: C'est quoi ?

- Globe virtuel + Données scientifiques + GIS + EER + + +
- Visualisation 2D – 3D – 4D
- Performances (>20fps)
- Formats de données (RPN, BURP, Radar, GIS, …)
- API de traitement et manipulation de données
- Analyse (graphes, statistiques, manipulations, …)
- Extensible (Macro, Outils, …)
- Configuration (Projet, script, ~/.eer_ToolDefs/eer_Default)
- Intégration (visualisation, prétraitement, analyse, validation, produit, automatisation, …)
- Script / Automatisation
- Développement par EER pour EER, mais …

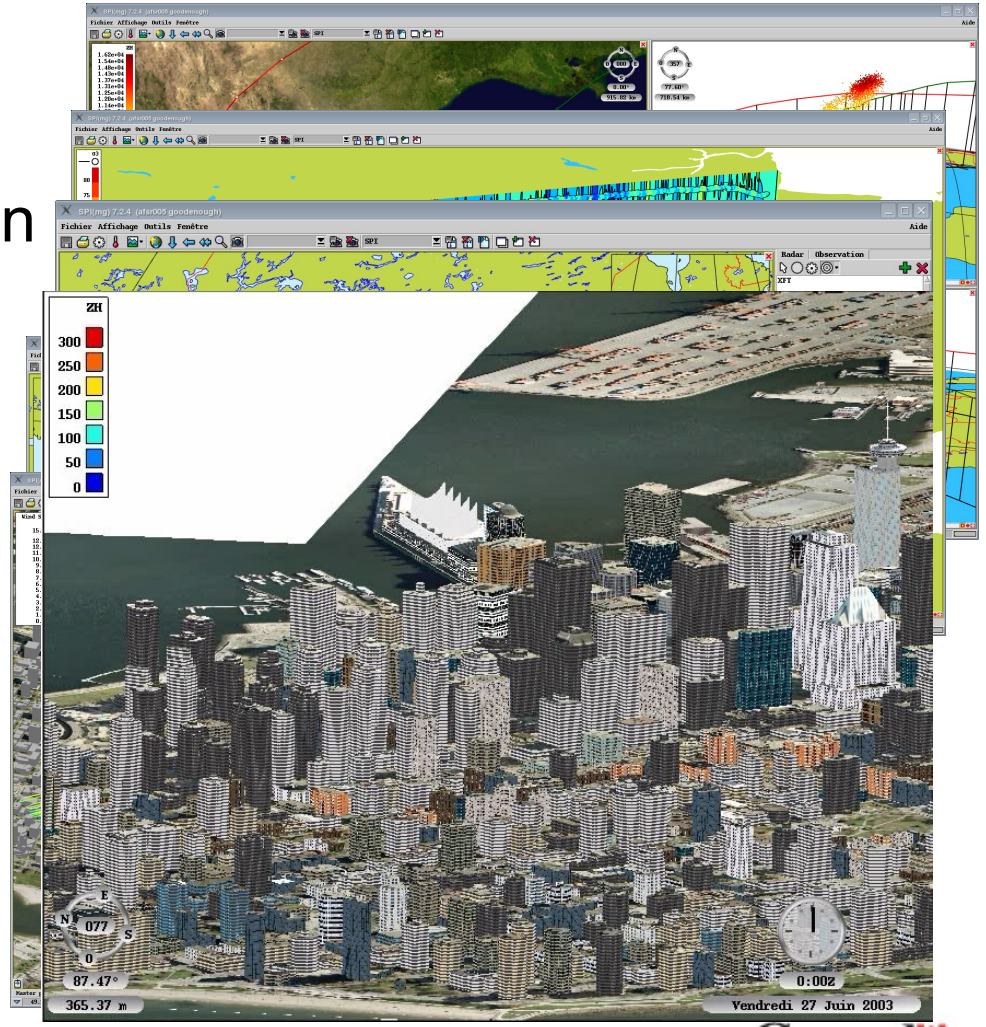
SPI: Utilisations typiques

- EER
- Données météo (FSTD)
- Données d'observations (BURP,BUFR)
- Données radar (URP)
- Données GIS (ShapeFile, GeoTIFF,…)
- Automatisation



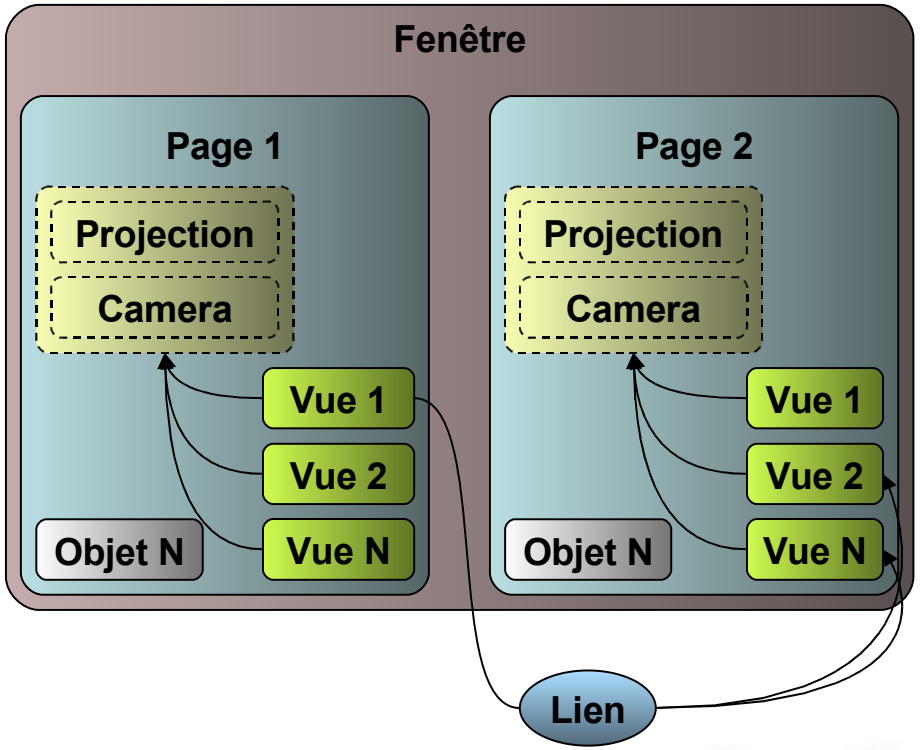
SPI: Utilisations ++

- Multiple point de vue dynamiques
- Visualisation de coupe en 3D
- Volume scan de radar
- Intégration de données GIS et modélisations
- Visualisation de modélisation de dispersion urbaine
- Animation « Flyby»



SPI: Hands on

- Principes / Vocabulaire
  - Fenêtres / Pages / Vues / Projections / cameras
- Aperçu
  - Manipulation (2D/3D)
  - Objets
  - Paramètres
  - Outils
- Exemples
  - Produit / Projet
  - EER
  - Fichier standards
  - Urbain

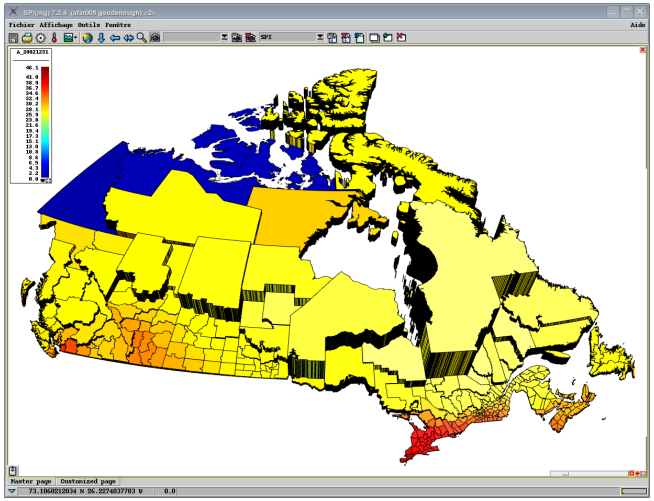


SPI: Quoi de plus que les autres

- Multiples pages dynamiques
- Multiples vues par pages
- Combinaisons objets / vues / pages = produits
- Capacité de lire plusieurs formats de données
- 3D gratos
- Performances ↑ / Ressources ↓
- Flexibilité / Extensibilité
- Script, mode batch
- Développement / débogage accéléré
- Unique API / code base

API (Application Programming Interface)

- IO plusieurs formats (RPN,BURP,BUFR,URP,GIS,…)
  - OGR: http://www.gdal.org/ogr/ogr_formats.html
  - GDAL: http://www.gdal.org/formats_list.html
- Calculs sur les matrices (>80 opérateurs)
  - Statistiques, mathématiques, géographiques, filtres, logiques, slicer, …
- Traitements / Interpolations / Transformations
  - Données: matrice, observation, vectoriel,
  - Méthodes: linéaire, cubique, moyenne, conservatif, krigage, rastérisation, …
- Analyse de données / validation
- Génération de produits graphiques
- Génération de produits GIS



API: Exemple 1 (Interpolation de champs)

```
#!/bin/sh
# the next line restarts using tclsh \
exec $SPI_PATH/tclsh "$0" "$@"

package require TclData

#----- Ouvrir les fichiers d'entree (1) sortie (2)

fstdfile open IN read  Data/2005102612_012
fstdfile open OUT write ./FSTD_InterpHorizontal.fstd

#----- Creer un champs sur grille PS de 229x229 en specifiant les IG1 IG2 IG3 et IG4

fstdfield create GRID 229 229 1
fstdfield define GRID -GRTYP N 115.0 300.0 25000.0 350.0

#----- Boucler sur les champs a interpoler

foreach var { TT UU GZ } {
   foreach fld [fstdfield find IN -1 "" -1 -1 -1  "" $var] {
      fstdfield read FROM IN $fld
      fstdfield gridinterp GRID FROM
      fstdfield write GRID OUT -32 True 1
   }
}

fstdfile close IN
fstdfile close OUT
```

Démarer Tcl

Inclure les API SPI dans Tcl

Ouvrir les fichiers standard d'entrée et sortie

Créer une grille PS

Boucle sur tous les champs TT, UU et GZ

Lire le champs, l'interpoler et l'écrire

Fermer les fichiers standards

API: Exemple 2 (Utilisation de shapefile)

```
#!/bin/sh
# the next line restarts using tclsh \
exec $SPI_PATH/tclsh "$0" "$@"

package require TclData

proc TimeZoneCheck { Lat Lon } {

   set tz 999

   if { [llength [set idx [ogrlayer pick TZ [list $Lat $Lon]]]] } {
      set tz [ogrlayer define TZ -feature $idx ZONE]
   }
   return $tz
}

#----- Open the TZ file

set layer [ogrfile open OGRFILE read Data/timezone.shp]
eval ogrlayer read TZ [lindex $layer 0]
ogrfile close OGRFILE

#----- Get the TZ for a latlon

puts [TimeZoneCheck 65.10 -123.3]
```

Démarer Tcl

Inclure les API SPI dans Tcl

Sélectionner le polygone à la lat-lon, en extraire la valeur du champs ZONE

Ouvrir le shapefile et lire la couche

Récupérer la zone horaire

API: Exemple 3 (Calculs sur données raster)

```
...

#----- Ouverture d'un fichier DEM
set bands [gdalfile open FILE read Data/srtm_n045w074_badmedian3x3]
gdalband read BAND $bands

#----- Calcul de la pente en degree
vexpr SLOPE dslopedeg(BAND)
gdalband configure SLOPE -desc "Slope in degree"

#----- Calcul de la pente en pourcentage
vexpr SLOPE100 dslope100(BAND)
gdalband configure SLOPE100 -desc "Slope in percent"

#----- Calcul de la courbature du profile
vexpr PCURV dprofcurve(BAND)
gdalband configure PCURV -desc "Profile curvature"

#----- Calcul de la courbarture tangentielle
vexpr TCURV dtangcurve(BAND)
gdalband configure TCURV -desc "Tangential curvature"

#----- Calcul de l'aspect (angle de la pente en XY)
vexpr ASPEC daspect(BAND)
gdalband configure ASPEC -desc "Aspect in degree"

#----- Calcul de la derivee partielle de deuxieme ordre dxy
vexpr DXX   ddxysecond(BAND)
gdalband configure DXX -desc "Second order partial dxy derivative"

#----- Sauvegrader les resultata
gdalfile open FILEOUT write Data/GDAL_Slope.tif "GeoTIFF"
gdalband write { SLOPE SLOPE100 PCURV TCURV ASPEC DXX } FILEOUT { COMPRESS JPEG PROFILE GeoTIFF TFW YES }
gdalfile close FILEOUT
```

Ouvrir le DEM et lire la bande

Effectuer les calculs et définir la description

Écrire les resultats en Geotiff compressé avec worldfile

API: Exemple 4 (Transformation, UTM à LL)

```
...

package require TclData

#----- Create UTM referential

georef create UTMREF
georef define UTMREF -projection
    {PROJCS["WGS_1984_UTM_Zone_14N",GEOGCS["GCS_WGS_1984",DATUM["D_WGS_1984",SPHEROID["WGS_1984",6378137.0
    ,298.257223563]],PRIMEM["Greenwich",0.0],UNIT["Degree",0.0174532925199433]],PROJECTION["Transverse_Mer
    cator"],PARAMETER["False_Easting",500000.0],PARAMETER["False_Northing",0.0],PARAMETER["Central_Meridia
    n",99.0],PARAMETER["Scale_Factor",0.9996],PARAMETER["Latitude_Of_Origin",0.0],UNIT["Meter",1.0]]}

#----- Reproject to Latlon

set ll [georef project UTMREF [lindex $argv 0] [lindex $argv 1]]

#----- Print LatLon

puts "[lindex $ll 0] [lindex $ll 1]"
```

Créer un géoréférentiel UTM 14N (Nomenclature OGC:WKT)

Transformer la coordonnée XY en LatLon

Écrire les lat lon

API: Exemple 5 (Données radar)

```
...

#----- Open the radar file

set scans [radarfile open RADARSITE read Data/200803121320~~CONVOL:URP:XFT:RADAR:IRIS]

puts "Available scans : $scans"

#----- Read the first scan

radarscan read SCAN1 RADARSITE 0

puts "Radar type    : [radarscan define SCAN1 -TYPE]"
puts "Date          : [radarscan define SCAN1 -DATE]"
puts "Site ID       : [radarscan define SCAN1 -SITEID]"
puts "Azimuth (Deg) : [radarscan define SCAN1 -AZIMUTHRESOLUTION]"
puts "Bin (Km)      : [radarscan define SCAN1 -BINRESOLUTION]"
puts "Sweep angles  : [radarscan define SCAN1 -SWEEPANGLE]"

radarscan stats SCAN1 -level 1


puts "Projected lat lon (sweep and bin): [radarscan stats SCAN1 -coordpoint 45.28 -74.74]"
puts "value (lat lon): [radarscan stats SCAN1 -coordvalue 45.28 -74.74]"
puts "value (sweep and bin): [radarscan stats SCAN1 -gridvalue 75.3750167173 107.193774955]"
```

Ouvrir le fichier radar

Lire le premier volume scan

Afficher quelques informations

Sélectionner le premier balayage

Effecuter des reprojections, et récupération de valeur en lat-lon

API: Exemple 6 (GeoPhysX: masque vectoriel)

```
proc GeoPhysX::AverageMaskCANVEC { Grid } {

   …
   set limits [georef limit [fstdfield define $Grid -georef]]
   set lat0 [lindex $limits 0]
   set lon0 [lindex $limits 1]
   set lat1 [lindex $limits 2]
   set lon1 [lindex $limits 3]

   GenX::GridClear $Grid 0.0

   #----- Loop over files
   foreach file [GenX::CANVECFindFiles $lat0 $lon0 $lat1 $lon1 { HD_1480009_2 } ] {
      GenX::Trace "   Processing file $file" 2
      ogrfile open CANVECFILE read $file
      ogrlayer read CANVECTILE CANVECFILE 0
      fstdfield gridinterp $Grid CANVECTILE ALIASED 1 0
      ogrfile close CANVECFILE
   }

   #----- Use whatever we have for US
   ogrfile open USLAKESFILE read $GenX::Path(Various)/mjwater.shp
   ogrlayer read USLAKES USLAKESFILE 0
   fstdfield gridinterp $Grid USLAKES ALIASED 1.0
   ogrfile close USLAKES

   vexpr $Grid 1.0-clamp($Grid,0.0,1.0)

   if { [llength [set idx [fstdfield find GPXOUTFILE -1 "" 1200 -1 -1 "" "ME"]]] } {
      fstdfield read GPXME GPXOUTFILE $idx
      vexpr $Grid ifelse($Grid>0.0 && GPXME==0.0,0.0,$Grid)
   }
   …
}
```

Get the grid limits in lat - lon

Loop on every water(HD_1480009_2) layer intersecting the grid limits

Rasterize the polygon by the percentage of grid cell covered

Do the same with US lake coverage

Invert the percentage since we want land percentage

Check consistency with topography

API: Exemple 7 (GeoPhysX: Large TIFF)

```
proc GeoPhysX::AverageVegeCORINE { Grid } {

   …
   #----- Pour la conversion des classes CORINE vers les classes RPN
   vector create FROMCORINE { 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
    33 34 35 36 37 38 39 40 41 42 43 44 }
   #----- Correspondance de Janna Lindenberg de decembre 2007
   vector create TORPN  { 21 21 21 21 21 21 24 24 24 14 20 20 20 16 18 18 18 18 18 18 18 18 5 5 25 14 14 14 14 14 24
    10 10 2 11 11 1 1 1 1 1 1 1 1 }

   fstdfield copy GPXVG $Grid
   fstdfield copy GPXVF $Grid
   GenX::GridClear [list GPXVG GPXVF $Grid] 0.0

   #----- Open the file
   gdalfile open CORINEFILE read $GenX::Path(CORINE)/lceugr100_00_pct.tif

   #----- Loop over the data by tiles since it's too big to fit in memory
   for { set x 0 } { $x<[gdalfile width CORINEFILE] } { incr x $Data(TileSize) } {
      for { set y 0 } { $y<[gdalfile height CORINEFILE] } { incr y $Data(TileSize) } {
         GenX::Trace "   Processing tile $x $y [expr $x+$Data(TileSize)] [expr $y+$Data(TileSize)]" 2
         gdalband read CORINETILE { { CORINEFILE 1 } } $x $y [expr $x+$Data(TileSize)] [expr $y+$Data(TileSize)]
         gdalband stats CORINETILE -nodata 255

         vexpr CORINETILE lut(CORINETILE,FROMCORINE,TORPN)
         fstdfield gridinterp GPXVF CORINETILE NORMALIZED_COUNT $Data(VegeTypes) Fa
      }
   }
   fstdfield gridinterp GPXVF - NOP True

   #----- Calculate Dominant type and save
   GeoPhysX::DominantVege $Grid
   …
}
```

Create the lookup table to go from CORINE dataset to RPN vege types

Make copies of the grid for VG and VFs and clear to 0

Read the dataset by tile, being to big to fit in memory (67000x58000)

Apply the lookup table for class conversion

Count the values by type. This will expand NK to the number of types

Divide the counting by the total valid number of values

API: Exemple 8-1 (Macro VerifQPF24)

```
proc Macro::Verif_QPF2448_0012::Execute { } {
   global GDefs env
   variable Data
   variable Error
   variable Param

   set stamp   [exec ~afsfops/S/r.fnom]
   set stamp24 [exec ~afsfops/S/r.fnom 1]
   set stamp48 [exec ~afsfops/S/r.fnom 2]

   SPI::LayoutLoad $Page::Data(Frame) VerifQPF24
   Macro::Cursor watch

   ProjCam::Set VerifQPF24      { 0.0 0.0 1.0 } { 0.0 0.0 2.0 } { 0.0 1.0 0.0 } 2.420 0 0 1 0 0 0 51.845 -101.926
   ProjCam::Set VerifQPF24_WEST { 0.0 0.0 1.0 } { 0.0 0.0 2.0 } { 0.0 1.0 0.0 } 5.443 0 0 1 0 0 0 49.327 -118.581
   ProjCam::Set VerifQPF24_EAST { 0.0 0.0 1.0 } { 0.0 0.0 2.0 } { 0.0 1.0 0.0 } 6.729 0 0 1 0 0 0 46.805 -71.947
   ProjCam::Select $Page::Data(Frame) $Page::Data(Frame) VerifQPF24

   Macro::Doing "Initializing config"
   if { ![colormap is VERIF_MAP] } {
      colormap create VERIF_MAP
      colormap read VERIF_MAP $GDefs(DirEER)/eer_Map/REC_Col.std1.rgba
   }

   font create VERIF_FONT -family courier -size -12 -weight bold

   dataspec create VERIF_SPECSYNOP
   dataspec configure VERIF_SPECSYNOP -desc "SYNOP (${stamp}_pcp)" -size 10 -icon SQUARE \
      -color black -colormap VERIF_MAP -mapall False -rendertexture 1 -rendercontour 1 \
      -rendervalue 1 -font VERIF_FONT -intervals $Param(Intervals)

   dataspec create VERIF_SPECSHEF
   dataspec configure VERIF_SPECSHEF -desc "SHEF (${stamp}_)" -size 10 -icon CIRCLE \
      -color black -colormap VERIF_MAP -mapall True -rendertexture 1 -rendercontour 1 \
      -rendervalue 1 -font XFont12 -intervals $Param(Intervals)
```

Appel externe pour obtenir les noms de fichiers

Lire et instaurer la mise-en-page

Créer les cameras (points de vues)

Créer les polices et palettes

Créer les objets de configurations

API: Exemple 8-2 (Macro VerifQPF24)

```
   Macro::Doing "Reading precip field"
   fstdfile open VERIF_REGFILE read /data/gridpt/dbase/prog/regdiag/${stamp48}_048
   fstdfield read VERIF_REGFIELD VERIF_REGFILE -1 "" -1 48 24 P PR
   set valid [fstdstamp toseconds [fstdfield define VERIF_REGFIELD -DATEV]]

   fstdfield configure VERIF_REGFIELD -desc "reg GEM qpf 2448" -factor 1e3 \
      -colormap VERIF_MAP -font VERIF_FONT -color black -intervals $Param(Interval
      -rendertexture 0 -rendercontour 2 -mapall True -rendervalue 2 -value INTEGER



   Macro::Doing "Reading surface obs shef"
   metobs create VERIF_SHEF /data/ade/dbase/surface/shef/${stamp}_
   metobs define VERIF_SHEF -VALID $valid False
   metmodel define [metobs define VERIF_SHEF -MODEL] -items { { 0 0 13023 { } } } -spacing 10
   metmodel configure [metobs define VERIF_SHEF -MODEL] 13023 -dataspec VERIF_SPECSHEF

   Macro::Doing "Reading surface obs synop"
   metobs create VERIF_SYNOP /data/ade/dbase/surface/synop/${stamp}_pcp
   metobs define VERIF_SYNOP -VALID $valid False
   metmodel define [metobs define VERIF_SYNOP -MODEL] -items { { 0 0 13023 { } } } -spacing 10
   metmodel configure [metobs define VERIF_SYNOP -MODEL] 13023 -dataspec VERIF_SPECSYNOP



   Macro::Doing "Creating product"
   Viewport::UnAssign $Page::Data(Frame) $Viewport::Data(VP)
   Viewport::Assign   $Page::Data(Frame) $Viewport::Data(VP) { VERIF_REGFIELD VERIF_SYNOP VERIF_SHEF }

   Macro::Doing ""
   Macro::Cursor left_ptr
}
```

Lire le champs de précip

Configurer l'affichage du champs

Lire les observations et indiquer l`heure de validitée désirée

Configurer l'affichage des modèles et de leurs composantes

Vider la liste d'affichage de la vue courante dans la page courante

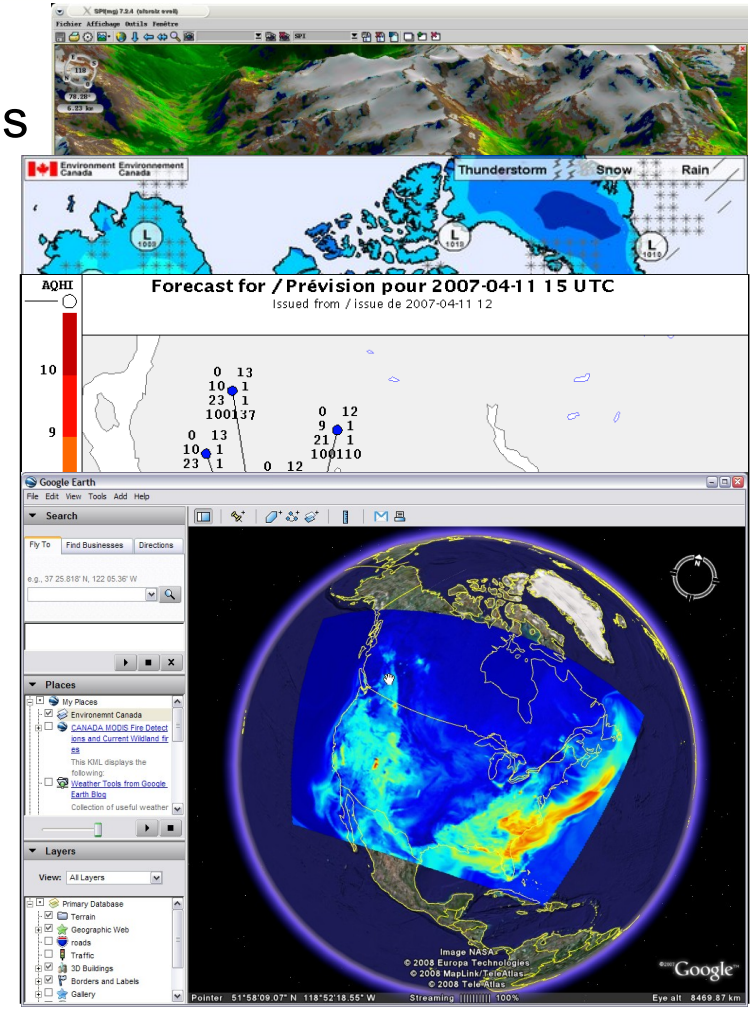
Assigner les données a la liste d'affichage de la vue courante

API: Quoi d'autre ?

- Pré et Post traitements aux modèles de prévisions.
- Calcul du flux le long d'une coupe verticale
- Calcul de la densité de drainage avec des données vectorielles
- Interpolation d'observations de vol d'avion dans une coupe verticale de l'atmosphère
- Interpolations des émissions en conservant la masse
- Calcul de concentration moyenne par région politiques / géographiques
- Classification urbaine a partir de données vectorielles et/ou satellitaires

SPI et API: Projets connexes

- GenPhysX - GenBioGenX
  - Calculs des champs géophysiques pour GEM et biogéniques pour GEM-MACH
- Carte JetMap sur wxoffice
- Passe opérationnelles AQ
  - Première utilisation officielle dans les passes CMOI
- Modélisation urbaine
  - Classification à 5m
- Serveur WMS
  - Diffusion des données de modélisations pour web mapping, google earth et autres.



La suite …

- Pour l'utiliser
  - export SPI_PATH=~afsrops/eer_SPI-7.2.4a
  - ~afsrops/eer_SPI-7.2.4a/SPI (Dev)
  - ~afseeer/eer_SPI-7.2.4a/SPI (Ops)
  - Tiny weeny NVIDIA ou –soft
- Documentation
  - http://iweb.cmc.ec.gc.ca/~afsr005/SPI/
- Lien externe
  - http://eer.cmc.ec.gc.ca/index_e.php?page=s_software/spi/spi_e.html
- Liste de distribution
  - spi-users@cmc.ec.gc.ca

. . .

- Rapport de bug
  - bugzilla.cmc.ec.gc.ca
- Support, Maintenance
  - Dans le contexte des mandats opérationnels et de R&D de la section EER
- Intéressés aux autres applications potentielles
  - Enjeux ressources à considérer