

Figure F.4: horizontal color slice and contour slice of the pressure together with the map of the HIRLAM domain

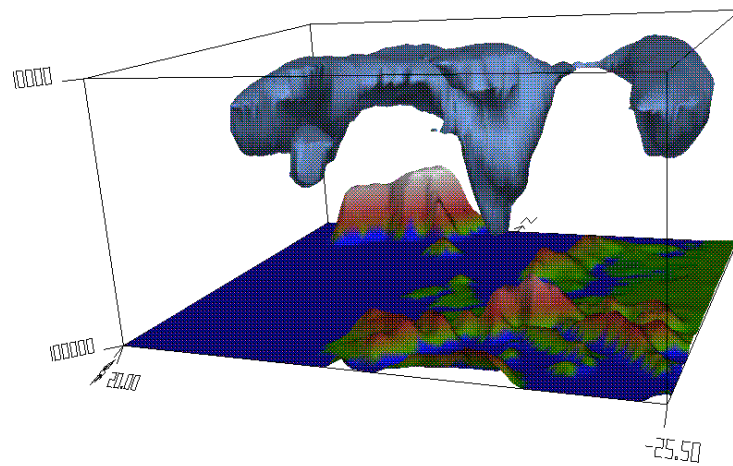


Figure F.5: iso surface of the wind at 40 knots together with the orography of the HIRLAM domain

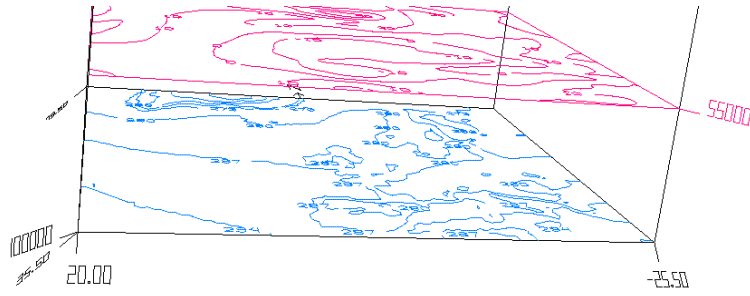


Figure F.2: contour surface of a 2-D field (without handle) and a 3-D field (with handle)

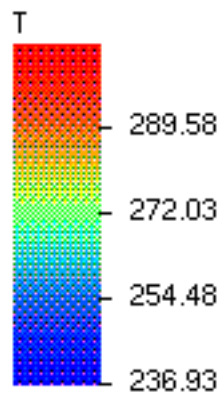


Figure F.3: legenda of the temperature

## Appendix F

# Example Images

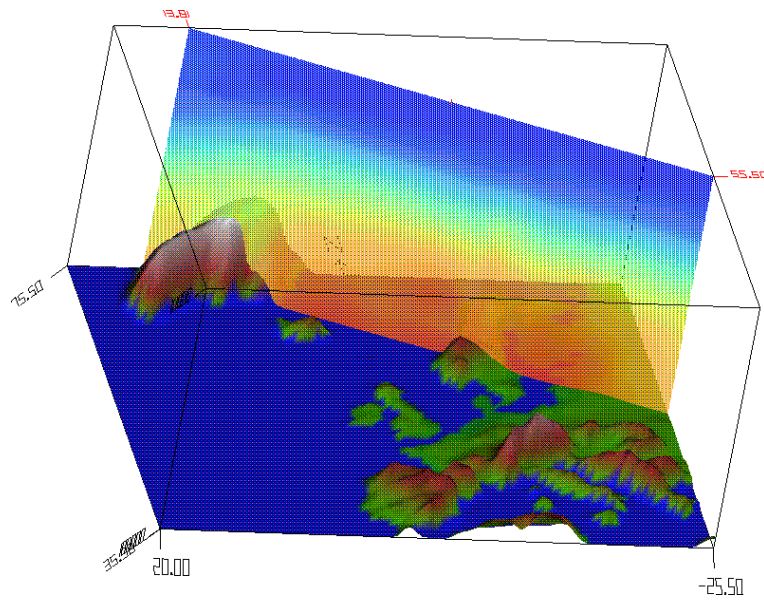


Figure F.1: vertical color surface of the temperature together with the orography of the HIRLAM domain

**Note:** This routine doesn't yet work in the VIS-5D distribution of VOGL. It is probably obsolete as the line style is not device dependent any more. The linestyle must be used when scan converting lines.

For full details the reader is referred to the X11 driver. The file `X11.c` contains the complete specification of a VOGL device driver.

**Note:** This routine is not available in the original VOGL distribution.

It is the responsibility of the device to approximate 24-bit color when a frame buffer of only 8-bits is available.

`DEV_mapcolor(int LUTindex, int r, int g, int b):`

Changes a colormap index to a given RGB value.

`DEV_font(char *fontfile):`

Sets up a hardware font. This should also set `vdevice.hwidth` and `vdevice.hheight`, which are the width and height of the current hardware font in pixels. VOGL assumes that hardware text is of a fixed width.

`DEV_char(char c):`

Prints a character of hardware text. This routine must do any necessary moving to make sure the current drawing position on the device is correct, and it must leave the device in graphics mode.

`DEV_string(char s[]):`

Prints a string of hardware text. This routine must do any necessary moving to make sure the current drawing position on the device is correct, and it must leave the device in graphics mode.

`DEV_fill(int n, int x[], int y[]):`

Draws filled polygons, devices which do not support this should just do an outline.

**Note:** This routine is obsolete in the VIS-5D distribution of VOGL. It is never called but included to retain backward compatibility.

`DEV_backbuf():`

Initializes double buffering by selecting the back drawing buffer and performing any other initializations. If no double buffering is supported, then `-1` should be returned.

**Note:** The double buffering used is in fact pseudo-double buffering. Drawing is done in a backbuffer which is copied into the front buffer when requested.

`DEV_swapbuf():`

Swaps the front and back buffer. What it really does in most device drivers is copy the back buffer into the front buffer.

`DEV_frontbuf():`

Switches drawing into the front drawing buffer.

`DEV_setlw(int w):`

Sets the line width.

**Note:** This routine doesn't yet work in the VIS-5D distribution of VOGL. It is probably obsolete as the line width is not device dependent any more. The linewidth must be used when scan converting lines.

`DEV_setls(int lss):`

Sets the line style of line drawings.

There must also be a function in the device driver to copy the device driver structure in the global (used) device entry. This function must be called conditionally from `drivers.c` as should the device name print statement (the device are printed if VOGL is used, but no device is set).

The nineteen functions required for a complete device driver are:

`DEV_init()`:

Device initialization routine. A routine which enables graphics on the device, sets the default color map, and sets `vdevice.maxS[xy]` and `vdevice.minS[xy]` to the window size in pixels.

**Note:** In the VIS-5D distribution of VOGL this routine also initializes the palette to colors distributed in the RGB space (when the frame buffer is only 8 bits).

`DEV_exit()`:

Does the necessary cleaning up to allow VOGL to exit leaving the device in a usable form.

`DEV_draw(int x, int y)`:

Draws a line from the current device position to a point (x, y) in vogle device coords - note these assume that (0, 0) is the bottom left hand corner. This routine must update the current device position.

**Note:** This routine is obsolete in the VIS-5D distribution of VOGL. It is never called but included to retain backward compatability.

`DEV_point(int x, int y)`:

Draws a single pixel in the current device color.

**Note:** This function is not available in the original VOGL distribution.

`DEV_getkey()`:

Gets a single character of input from a device capable of providing it.

`DEV_locator(int *wx, int *wy)`:

Returns the mouse position for the device in VOGL device coordinates (returned in the arguments) and returns a bit pattern giving which buttons were down at the time of the call.

`DEV_clear()`:

Clears the current viewport to the current background color.

`DEV_color(int LUTindex)`:

Changes the current device color to the specified LUT index.

`DEV_RGBcolor(int r, int g, int b)`:

Changes the current device color to the specified RGB value.

```
void lmbind(short target, short index)
```

`lmbind` binds one of the eleven light controlling resource of the GL to a resource defined by `lmdef`. `target` can be one of `MATERIAL`, `BACKMATERIAL`, `LIGHT[0-7]` or `LMODEL`.

```
void shademodel(long model)
```

Sets the shading model. Two shading models are supported: `FLAT` and `GOURAUD`.

```
long getsm(void)
```

Returns the current shading model. The value returned can be either `FLAT` or `GOURAUD`.

### E.1.4 Depthcueing functions

#### Private functions:

```
void calc_depthcue(Vertex *v)
```

Calculates the color of the given vertex `v` using the Screen Z-coordinate of `v` and the current RGB and Z-range.

#### Public functions:

```
void depthcue(int onoff)
```

Turns depth-cueing on or off. If `onoff` is zero, depthcueing is turned off. Otherwise it is turned on.

```
void lRGBrange(short rmin,gmin,bmin,rmax,gmax,bmax,long znear,zfar)
```

Specified the RGB color and Z range to be used for depth-cueing. The values `rmax`, `gmax` and `bmax` are mapped to `znear`.

## E.2 VOGL device driver structure

VOGL device drivers contain the following information:

- Three character pointers giving:
  - The name of the device,
  - The name for the small hardware font,
  - The name for the large hardware font.
- 19 function pointers giving the functionality of the device. If a device is not capable of some function (eg. color changing) a no-op function should be provided which has a return value of `-1`.

**LMC\_COLOR:** RGB color commands will set the current color. If a color is the last thing sent before a vertex, the vertex is colored. If a normal is the last thing sent before a vertex, the vertex will be lighted. **LMC\_COLOR** is the default mode.

**LMC\_EMISSION:** RGB color commands will set the EMISSION color property of the current material.

**LMC\_AMBIENT:** RGB color commands will set the AMBIENT color property of the current material.

**LMC\_DIFFUSE:** RGB color commands will set the DIFFUSE color property of the current material. Alpha, the fourth color component specified by RGB color commands will set the ALPHA property of the current material.

**LMC\_SPECULAR:** RGB color commands will set the SPECULAR color property of the current material.

**LMC\_AD:** RGB color commands will set the DIFFUSE and AMBIENT color property of the current material. Alpha, the fourth color component specified by RGB color commands will set the ALPHA property of the current material.

**LMC\_NULL:** RGB color commands will be ignored.

IMPORTANT: `lmcolor` does NOT change the data structures holding the material definitions, but is provided to support fast changes to the current material, without changing the definition of the currently bound material. Thus `lmcolor` changes are lost whenever a new material is bound. only the **MATERIAL** properties can be changed, not those of the **BACKMATERIAL**.

```
void n<len><type>(type normal)
```

These functions are named identical as the color functions (prefixed with an `n` instead of an `c`, and are used to specify vertex normals. `len` can currently only be 3. `type` can be `s` (short), `i` (long), `d` (double) or `f` (float). Vertex normals must be normalized in order for the shading calculation to be meaningful, because the dot product of the vectors is used as a measure of the angle between the vectors. See also the function `nmode()`.

```
void nmode(long mode)
```

Tell the GL when normal vectors must be normalized. `mode` can be:

- **NAUTO**, normalize vectors only when the current ModelView matrix is not orthonormal.
- **NNORMALIZE**, always normalize vectors, disregarding the current ModelView matrix.

Even in hardware, re-normalization can have an adverse effect on the performance.

```
void lmdef(short deftype, short index, short np, float props[])
```

Defines or modifies a material, light source, or lighting model. `deftype` can be one of **DEFMATERIAL**, **DEFLIGHT** and **DEFLMODEL**. `index` is the index of the resource to define or modify (each resource type has a unique array of definitions). `np` is the number of entries in the `props` array, `props` contains the various attributes appropriate to the type of resource. The kinds of attributes available for each resource type were listed in section 3.1.3.



```
void ccall(short r, short g, short b, short a)
```

Generic color processing functions. All public RGB color functions call this function. The 4 arguments are clipped to a limit of 255. Then depending on the *color mode*, see section 3.1.3, the current color is set or a specific material characteristic is set.

#### Public functions:

```
void cpack(unsigned long argb)
```

Specify a true red, green, blue color using the packed ARGB format (the 4 color components are packed into one unsigned long as 0xAARRGGBB). The A component is the alpha (transparency) component and is currently not used in VOGL.

```
void RGBcolor(short r, short g, short b)
```

Equal to `cpack()` only specifies no alpha component and red, green and blue separately.

```
void c<len><type>(type colors)
```

These are various color specification functions which all take a vector (an array) as input. `<len>` determines the length of the vector expected and can be 3 (array contains red, green and blue values) or 4 (alpha component also included). `<type>` determines the type of the array and can be s (short), i (long) or f (float). Colors are clipped to a maximum of 255. If the float variant of the function is used, values are expected to be in the range 0.0 - 1.0. For the other functions the range is 0 - 255.

### E.1.3 Lighting functions

The following functions are implemented for GL lighting support:

#### Private functions:

```
void calc_shade(Vertex *v)
```

Calculates color of vertex `v`. The supplied `Vertex struct` should contain the transformed vertex position in Eye coordinates. When lighting is on, the color is calculated using the vertex normal and lighting model. Otherwise the vertex is colored (using the current color).

```
void init_lightmodule(void)
```

Initializes the light module by binding all eleven lighting controlling resources to zero. The shading model, lighting and normalization modes are also initialized.

#### Public functions:

```
void lmcolor(long mode)
```

Specifies the color mode. The color mode determines the effect of color commands like `cpack()` and the `c` calls. The following color modes are possible:

**Private functions:**

`zbuftype WtoSz(Vector v)`

Returns the Screen Z-coordinate of the Vector *v* (which must be in Clip coordinates).

`int zbuffer_condition (Scoord x, Scoord y, zbuftype z)`

Eight zbuffer test functions which all return TRUE and update the Z-buffer if *condition* is TRUE, which can be *never, less, equal, lequal, greater, notequal, gequal* and *always*. Selection of the functions is done by `zfunction()`, which sets a function pointer to one of the eight functions (a case statement is probably too slow because the selected function must be called for every pixel in every geometry).

`void zbuffer_swapbuffers(void)`

Swaps the Z-buffers (only when double-buffering is supported and activated).

**Public functions:**

`void zbuffer(Boolean onoff)`

Turns Z-buffering on or off.

`void zfunction(int type)`

Specifies one of eight Z-buffer comparison functions that determine if a pixel must be updated or not. Possible values for *type* are `ZF_NEVER`, `ZF_LESS`, `ZF_EQUAL`, `ZF_LEQUAL`, `ZF_GREATER`, `ZF_NOTEQUAL`, `ZF_GEQUAL` and `ZF_ALWAYS`. For instance, if `ZF_LESS` is chosen, a pixel is updated if the Z-value of the incoming pixel is *less* than the existing Z-value in the Z-buffer.

`void lsetdepth(unsigned long near, unsigned long far)`

Sets the range of Z-buffer values that values in clip coordinates are mapped to.

`void zclear(unsigned long zfar)`

Clears the Z-buffer to the specified value.

`void czclear(unsigned long color, unsigned long zfar)`

Clears both the Z-buffer and the color frame buffer. This function has been provided by the GL because clearing both buffers at once can often be done in hardware faster than clearing the buffers separately using the functions `clear()` and `zclear()`. This is not the case in VOGL.

**E.1.2 RGB color functions**

Since VOGL only supported colormap mode, the following functions had to be implemented in order to enable the specification of RGB colors:

**Private functions:**

## Appendix E

# VOGL implementation details

This appendix describes the enhancements made to VOGL and the structure of VOGL device drivers. Of the private functions, only the most important are discussed. The reader is referred to the source code for further detail. The public functions are all discussed.

### E.1 VOGL enhancements

The list of `typedefs` of VOGL have been extended with the following new `typedefs`:

- `zbuftype`. This is the type used for each entry in the Z-buffer. It can be changed, but then also the `#define` for `MAXZ` must be changed. `MAXZ` represents the maximum number that fits in the type `zbuftype`.
- `Vertex`. This is a structure which holds information about vertices. It contains the following members:
  - `v`, vertex position in Object coordinates (as specified by the user),
  - `eye`, vertex position in Eye coordinates,
  - `clip`, vertex position in Slip coordinates,
  - `sx`, vertex *x* position in Screen coordinates,
  - `sy`, vertex *y* position in Screen coordinates,
  - `sz`, vertex *z* position in Screen coordinates,
  - `r`, `g`, `b`, `a`, vertex color and transparency when in RGB mode,
  - `color`, vertex color when in colormap mode (currently not used).

#### E.1.1 Z-buffer functions

The following functions have been implemented to support Z-buffering:



When the interface of `extract_vslice()` is altered, it is of course necessary to change all the function calls as well. The changed vertical slice routine is not enough however to visualize wind slices correctly. The wind vectors are scaled with a constant factor in the function `calc_hwind()` and `calc_vwind()`. The scaling must be changed when pressure levels are used.

`calc_hwind()` and `calc_vwind()`

- Change the scale factor for the W component of the wind.

## D.4 VOGL related modifications

In order to use VOGL with `vis5d` the following modifications must be made.

(`x.h`)

- In `x.h` the VOGL include file `vogl.h` must be included.

(`graphics.c`)

The graphics module contains all 3-D graphics library specific code. Each function in the graphics module contains system specific parts for the various graphics libraries used by VIS-5D. In order to use VIS-5D with VOGL, some functions must be extended with a code section specific to VOGL, while other functions can use the same code as the SGI and IBM systems does. The last code sections are enclosed by `sgi_or_ibm_or_VOGL` preprocessor statements and can be easily identified. Function which are different for VOGL with respect to the SGI are:

- `init_graphics`. Initialization is split in a system specific part and a VOGL specific part. The first part initializes the variables `ScreenWidth` and `ScreenHeight` to the size of the screen, and the second part sets variables which indicate the functionality that is available in VOGL such as High Quality rendering and screendoor transparency.
- `make_window`. This function creates the 3-D window and initializes the graphics module. VOGL must be instructed to use the window created in this function. This is done by calling the (X11 driver specific) function `vo_xt_window()`. Furthermore, VOGL doesn't support screendoor transparency, which doesn't have to be initialized.
- `set_color()`. Sets the current color and transparency. The latter is not set for VOGL.
- `polytrinorm()`. Due to strangeness in the IBM hardware it is necessary to break polytriangle strips into individual triangles and test to make sure the normal vectors are correct. Also triangles with zero area must be culled out. Apparently, VOGL suffers from the same behaviour, because the same precautions are necessary when using VOGL.
- Other changes are necessary to disable calls to transparency functions.

**grid.c:**`(read_old_grid)`

- Set the `LevelType` variable to `HEIGHT_LEVELS`.

`(read_new_grid)`

- Set the `LevelType` variable to `HEIGHT_LEVELS`.

Note that in the function `read_varNL_grid()` the `LevelType` is read from the file. This is not conditional, the leveltype is always available in this type files (ID = 0x80808085).

`(load_dataset_grid)`

- Depending on the leveltype read from the file (or set to the default value), initialize the variable pairs `BottomCoordinate`, `TopCoordinate` and `BottomBound`, `TopBound` to the appropriate values. For height levels, the coordinate pairs are equal, i.e. `BottomCoordinate = BottomBound` equals the height of level 0 and `TopCoordinate = TopBound` equals the height of level `MaxNL - 1`. For pressure levels, `BottomBound` must be set to 0 instead and `TopBound` to `MaxNL`. A warning is printed if an unknown leveltype is encountered and `HEIGHT_LEVELS` is used instead.

`(get_grid_value)`

- Implementation of the same type interpolation as described in section 5.2.1.

**render.c:**`(draw_box)`

- When real domain bounds must be printed near the box corners, use the variables `BottomCoordinate` and `TopCoordinate` instead of the variables `BottomBound` and `TopBound` used previously.

`draw_slice_tick()`

- Use `BottomCoordinate` and `TopCoordinate` to calculate the level values to place at slice ticks.

**work.c:**`extract_hslice()`

- Use linear interpolation for `HEIGHT_LEVELS` and other levels (default). When `LevelType` is `PRESSURE_LEVELS` however, interpolation must be performed using the *log* of the level pressures. The pressure of the level to extract is calculated using linear interpolation between the two adjoining levels, and the resulting pressure is used to calculate new interpolation fractions, as described in section 5.2.1.

`extract_vslice()`

- Logarithmic interpolation of vertical slices is currently not yet implemented, but the places are marked where modifications must be made. A method that can be used is described in section 5.2.2.

## D.3 Logarithmic interpolation modifications

The C preprocessor statements have the following form:

```
#ifdef LEVELTYPES
    <newcode>
#else
    <oldcode>
#endif
```

A new global variable `int LevelType` is declared to contain the leveltype of the data. Currently `LevelType` can have the following values:

**HEIGHT\_LEVELS:** Metres are used as unit of the vertical coordinate. This is the default leveltype and used for old COMP files. VIS-5D should behave as before for data defined at height levels.

**PRESSURE\_LEVELS:** Pascal is used as unit of the vertical coordinate. If data with this leveltype is visualized, vertical interpolation is performed using the logarithm of the vertical pressure coordinate.

Furthermore, because the vertical coordinate stored in the `Height[]` array doesn't have to be the height of the levels in metres anymore, these values can't be used for scaling of the topography. Two new variables are used: `BottomCoordinate` and `TopCoordinate` represent the bottom and top of the domain in units of the vertical coordinate. The already existing variables `BottomBound` and `TopBound` represent the top and bottom of the domain in kilometres. The first pair of variables is used to calculate values to place at slice ticks i.e. the real vertical coordinates. The second pair of variables is used to scale the height of the topography, etc. to the correct height.

The following changes have been made to `vis5d`:

### `globals.h`:

- Declare variables
  - `extern int LevelType,`
  - `extern float BottomCoordinate,`
  - `TopCoordinate.`
- and preprocessor constants
  - `HEIGHT_LEVELS,`
  - `PRESSURE_LEVELS.`

### `globals.c`:

Idem as `globals.h` but only the variables and without the `extern` class.

```
static int compress_iso_verts2(vx, vy, vz,
                              cvx, cvy, cvz, verts, numlev)
int numlev;
```

This function is obsolete and only changed to preserve consistency.

(extract\_hslice)

- New interface for `extract_hslice()`:

```
static float* extract_hslice( grid, level, colmajor, numlev )
int numlev;
```

where `numlev` is the number of levels of the parameter for which to extract a slice. It is used as an upper bound for the level to extract.

(calc\_hslice)

- Modified function call to `extract_hslice`, passing the extra parameter `NL[parm]`.

(calc\_vslice)

- Set the number of rows in the vertical slice to the number of levels of the parameter.

(calc\_chslice)

- Modified function call to `extract_hslice`, passing the extra parameter `NL[parm]`.

(calc\_cvslice)

- Set the number of rows in the vertical slice to the number of levels of the parameter.

(calc\_hwindslice)

- Modified function call to `extract_hslice`, passing the extra parameter `WindNL` (for  $U$ ,  $V$  and  $W$ ).
- Calculate the Z-coordinate of the horizontal wind slice using `WindNL`.

(calc\_vwindslice)

- Set the number of rows in the vertical slice to the number of windlevels, `WindNL`.

(calc\_traj)

- Convert from grid to graphics coordinates using `WindNL` (and check if `WindNL` is zero).



- The new slice position is scaled using `MaxNL`, but clipped to the extra parameter `maxlevel`.

(`move_vslice`)

- New interface for `move_vslice()`:

```
static int move_vslice(numlevels,
  curx, cury, r1,c1, r2,c2, corner)
  int numlevels;
```

where `numlevels` is a new input parameter and holds the number of levels of the parameter for which a vertical slice is moved.

- Set the number of levels of the vertical slice to move to the supplied value `numlevels`.

(`move_slice`)

- Call `move_hslice()` and `move_vslice()` with the new supplied parameter (6 times).

**traj.c:**

(`trace`)

- Set `ql` (the vertical box dimensions) to the number of levels of the wind parameters, `WindNL`.

**work.c:**

(`calc_surface`)

- Call `main_march()` with the correct number of levels, `NL[parm]`.
- Pass an extra parameter to `compress_iso_verts`. The number of levels must be used in that function to scale the Z-coordinates of the calculated iso-surface.

(`compress_iso_verts`)

- New interface for `compress_iso_verts()`:

```
static int compress_iso_verts(vx, vy, vz, vpts,
  cvx, cvy, cvz, verts, numlev)
  int numlev;
```

where `numlev` holds the number of levels of the parameter for which the iso-surface generated must be compressed. If `numlev` is one, a warning is given. This should never occur because iso-surfaces cannot be calculated for single-level fields.

(`compress_iso_verts2`)

- New interface for `compress_iso_verts2()`:

(draw\_chslices)

- If a horizontal color slice is drawn for a single-level field, initialize the Z-coordinate of the slice to **Zmax** instead of doing the division (which will be a division by zero).
- Disable the drawing of slice ticks for single-level fields (color slices).

(draw\_hwind)

- If a horizontal wind slice is drawn for a single-level field, initialize the Z-coordinate of the slice to **Zmax** instead of doing the division (which will be a division by zero).
- Disable the drawing of slice ticks for single-level fields (wind slices).

(draw\_slice\_tick)

- Calculate the slice tick Z-position from **MaxNL** (and make sure that **MaxNL** is greater than 1 (**MaxNL** could be 1 if only single-level fields are in the data set)).
- Idem as above, now to calculate the slice tick level value.

**save.c:**

(save)

- Calculate the correct number of bytes to save using **NL[ip]**.
- Write the correct number of **ga**, **gb** and grid values using **NL[ip]**.

(restore)

- Read the correct number of **ga**, **gb** and grid values using **NL[ip]**.

**slice.c:**

(distance\_to\_hslice)

- Calculate the normalized slice level using **MaxNL** and check if **MaxNL** is greater than one.

(find\_nearest\_slice)

- Skip single-level field slices when searching for the slice nearest to a given cursor position. These slices have no slice tick, so they can't be moved.
- Do the same for color slices.
- Do the same for wind slices.

(move\_hslice)

- A new interface:

```
static int move_hslice(curx, cury, level, corner, maxlevel)
int maxlevel;
```

with a new variable which holds the upper bound on the slice movement.

- Use `NL[ip]` instead of `N1` as upper bound for the for-loops.

(decompress)

- The modified `decompress()` interface:

```
static int decompress( ip, data, compdata, ga, gb )
int ip;
```

with an added `ip` parameter to use as index into the `NL []` array.

- Use `NL[ip]` as upper bound for the for-loop.

**gui.c:**

(create\_widgets)

- Create no buttons for single-level fields for iso-surfaces and vertical slices.

(add\_button\_row)

- When a new button row is created (for clone and analysis variables) disable the iso-surface and vertical slice buttons if the variable has only one level.

(init\_new\_var)

- Initialize the horizontal slice level to  $(NL[newvar]-1)/2$ . This will be zero for single-level fields, which is correct.
- Do the same for horizontal color slices.

**main.c:**

(initialize2)

- Initialize the horizontal contour and color slice levels of all physical parameters to  $(\#levels)-1)/2$ , as in `gui.c`.
- Initialize the horizontal wind slice levels to  $(WindNL-1)/2$ .
- Initialize the 3-D cursor position to the middle of the box  $(MaxNL-1)/2$ .

**render.c:**

(init\_box)

- Initialize the scale of the box using `MaxNL` instead of `N1`.

(draw\_box)

- When plotting grid dimensions at the corners of the box, `MaxNL` should be used.

(draw\_hslices)

- If a horizontal contour slice is drawn for a single-level field, initialize the Z-coordinate of the slice to `Zmax` instead of doing the division (which will be a division by zero).
- Disable the drawing of slice ticks for single-level fields (contour slices).

- Use a new (more difficult) calculation to find the file address in a McIDAS GRID file. This is more difficult because the grids can vary in size. First the total size of one time step is calculated. Then the offset in the current time step is found and is added to `timestepnumber*timestepsize`. Returned by the function is the sum of the value calculated and the size of the header (which is calculated in `read_variableNL_grid`).

`(read_from_disk)`

- Add the new file format: When a variableNL type COMP file is used, it is possible to read the original data from the McIDAS GRID file on disc.
- Use the number of levels of the current parameter to read the `ga`, `gb` values and to allocate the correct grid size.
- Read the correct number of bytes from the file, the offset is calculated by `grid_address()`.

`(get_grid)`

- Allocate a buffer of the right size (using `NL[ip]`, `ip` was already available as a parameter, thankfully).
- Get the address of the `ga`, `gb` values for the grid to compress using the new macros.
- Decompress the data, using a modified version of the function `decompress()`. The parameter number `ip` is passed as an extra parameter.

`(release_grid)`

- Deallocate is called with the modified grid size as an extra parameter.

`(get_grid_value)`

- If the 'lev' value to find a value at exceeds the total number of levels available for the requested parameter, return a missing value.
- Modify the upper bound check of `k0` (the level grid index of the point to obtain), and if the parameter has only one level available, set both level grid indices to zero. Interpolation in a cube is not possible in this case.

`(allocate_clone_variable)`

- Initialize the `NL[]` array entry for the new clone variable to the value of the original variable (the one being cloned).
- Use the new number of levels as upper bound in the for-loop.

`(allocate_analysis_variable)`

- Set the `NL[]` array entry for the new variable to `-1` because no levels are yet calculated by the external analysis function.

`(install_new_grid)`

- Use the macro `COMPsize` to calculate the grid size.

(read\_old\_grid)

- When the program is compiled with the preprocessor symbol `VARIABLE_NL` defined, it must still be possible to read the old COMP formats (see appendix A.4 for a description of the various file formats VIS-5D can handle). Therefore the old functions must be patched in the following way:
  - Read the single value `tempNL` from the file, and copy it into every (used) entry of `NL[MAXPARMS]`. Also initialize `MaxNL` to `tempNL`. Initialization of `WindNL` is delayed until the parameters are scanned for existence of the  $U, V$  and  $W$  parameters (later in `grid.c`).
  - Allocate the buffers for `ga` and `gb` (changed in size, use `MaxNL` instead of `NL`).
  - When reading the `ga` and `gb` values from the file, use `NL[ip]` as upper bound of the for-loop instead of `NL` (because `NL` no longer exists).
  - Allocate a buffer for the grid data (size has changed, now uses the `COMPSIZE` macro).
  - Read the grid from the file (size has changed, now uses the `COMPSIZE` macro).

(read\_new\_grid)

- Read the single value `tempNL` from the file, and copy it into every (used) entry of `NL[MAXPARMS]`. Also initialize `MaxNL` to `tempNL`. Initialization of `WindNL` is delayed until the parameters are scanned for existence of the  $U, V$  and  $W$  parameters (later in `grid.c`).
- The new grid format contains a Height array of values given the height values of the grid levels. The size of this array is `MaxNL`.

The rest of the changes to the function `read_new_grid()` are analog to the changes to `read_old_grid()`, from the `ga`, `gb` buffer allocation onwards.

(read\_variableNL\_grid)

- A new function is written to read the new format COMP files, described in appendix A.4. This function is just a straightforward implementation that reads the new file format. The functions `read_old/new/variableNL_grid` are much alike.

(load\_data\_set)

- This is the main function that calls one of the functions `read_old_grid()`, `read_new_grid()` or `read_variableNL_grid()`. Prior to loading the data set, the whole `NL[MAXPARMS]` array is set to `-1`. This can make possible bugs go real wrong and is intended to spot possible bugs more easy.
- Another `if (id==...)` is inserted to call the new `read_variableNL_grid()` function. The `FileFormat` variable is given the value 4 (the first unused value).
- After reading the data set, the variable `WindNL` is set to the minimum of the number of levels of the parameters  $U, V$  and  $W$  (if they are available). If none of them is available, `WindNL` will have the value `-1`, again to make bugs more apparent.
- Set the `TopBound` (of the domain) using `MaxNL` instead of `NL`.

(grid\_address)

where `numlevels` is the new parameter and is used seven times in an upper bound on the number of iterations of for-loops.

(`get_mcgrid`)

- The changed `get_mcgrid()` interface:

```
get_mcgrid( file, grid, data, numlevels )
int numlevels;
```

where `numlevels` is the number of levels to read from the McIDAS GRID file. This value is used to verify the read operation, to check if the correct number of levels is read.

**globals.c / globals.h:**

- Declaration of the following global variables:

```
int NL[MAXPARMS], MaxNL, WindNL;
```

These are described earlier in this section.

**grid.c:** The following COMP (or grid) formats are mentioned:

*old\_grid format* : equal number of levels for each grid, grid by grid compression.

*new\_grid format* : equal number of levels for each grid, layer (level) by layer compression.

*VariableNL\_grid format* : variable number of levels for each grid, layer (level) by layer compression.

- In the grid module buffers are created to store the `ga` and `gb` values of each level (for all parameters and all time steps). Entries from this buffer are set and obtained by two macros, which are modified to the following:

```
#define GA(IT,IP,LEV) Ga[(((IT) * MAXPARMS + (IP)) * MaxNL + (LEV))]
#define GB(IT,IP,LEV) Gb[(((IT) * MAXPARMS + (IP)) * MaxNL + (LEV))]
```

where `IT` is the time step, `IP` the parameter and `LEV` the level to obtain or set a `ga` or `gb` value for. The array contains gaps because for every parameter `MaxNL` values are stored to simplify the indexing. Because every grid has its own size, it is impossible to use the variable `CompSize` (which was initialized to  $((Nr*Nc*NL+3)/4)*4$ ), the number of floats in a grid). Therefore a new macro is defined:

```
#define COMPSIZE(ip) ( ((Nr*Nc*NL[ip]+3)/4)*4 )
```

This macro must be used instead of the old variable `CompSize`. If the parameter number `ip` is not available it must be provided by modifying function interfaces.

```

    INTEGER FUNCTION VARNLUSERFUNC( OUTGRID, MAXNL, OUTNL,
*                               INGRID, SUMNL,
*                               NR, NC, NL, NLIDX, NP,
*                               NAMES, DATE, TIME,
*                               SOUTH, WEST, BOTTOM,
*                               DELTALAT, DELTALON, DELTAHGT )

C    ARGUMENTS :
    INTEGER*4 MAXNL, OUTNL, SUMNL, NR, NC, NP
    INTEGER*4 NL(NP), NLIDX(NP)
    REAL*4    OUTGRID(NR,NC,MAXNL)
    REAL*4    INGRID(NR,NC,SUMNL)

    CHARACTER*8 NAMES(NP)
    INTEGER*4 DATE, TIME
    REAL*4 SOUTH, WEST, BOTTOM
    REAL*4 DELTALAT, DELTALON, DELTAHGT

```

References to the output grid are not changed: `outgrid(ir, ic, il)` references grid location row `ir`, column `ic`, level `il` as normal. References to the input grid are changed in the following way. The old (intuitive) manner of referencing `ingrid` to read grid location `(ir, ic, il)` of parameter `ip`:

```
ingrid(ir, ic, il, ip)
```

now becomes

```
ingrid(ir, ic, NLIDX(ip) + il)
```

which is only slightly less intuitive.

- The compress routine `compress_grid()` must know the number of levels to compress, so this value is passed as an extra integer parameter. The new interface is described below in this module.
- The protocol of the output grid sent back to `vis5d` has changed. First the number of levels in the output grid is sent, then the grid data. The number of `ga` and `gb` values sent back also changed because there is one such value for each level in the output grid.

(`compress_grid`)

- The changed `compress_grid()` interface:

```
compress_grid( data, comp, min, max, levga, levgb, numlevels )
int numlevels;
```

- The receive protocol is changed. The external functions sends the number of levels in the grid that it has computed. Dynamic allocation of two buffers (one for the uncompressed grid received, the other for a compressed grid calculated) is done using this value. After receiving the grid, the compression values `ga` and `gb` are received, one of each for every level in the output grid.

**cursor.c:**

(cursor\_event)

- Scale the cursor Z-coordinate using `MaxNL`, the size of the box expressed in number of levels.

**extmain.c:** This module is separated from `vis5d` in that it is not linked with `vis5d` but is linked instead with every external analysis functions. The external analysis function is called from `extmain.c`.

- Declare the following new global variables:

```
int NL[MAXPARMS],      Number of levels for each parameters.
    MaxNL,             Largest number of levels in NL[].
    SumNL,             Sum of all levels in NL[].
    NLindex[MAXPARMS]; Cumulative sum of levels in NL[].
```

(call\_user\_function)

- Declare a new integer variable to pass to the external analysis function. After execution of the external analysis function it contains the number of levels in the output grid.
- Receive an array of `NL`'s (one for each parameter) instead of one. These values are stored in `NL[MAXPARMS]` and `MaxNL`, `SumNL` and `NLindex[]` are calculated during the reception of the `NL[]` array.
- Allocate storage for all the input grids of one time step and the output grid. A total time step has `Nr*Nc*SumNL` floats. The output grid must be allocated worst case, ie. the largest number of levels, `MaxNL`, because it is not known in advance how many levels the external analysis function will compute. Also worst case buffers are allocated for the `ga` and `gb` values and the compressed grid.
- Storing of the input grids received from `vis5d` is done using `NLindex[]`. If the input buffer of size `Nr*Nc*SumNL` is seen as an array of levels, then `NLindex[]` gives the start index for each parameter in this array.
- Read a grid from a McIDAS GRID file using the function `get_mcgrid`, which now takes the number of levels to read as an extra parameter (change described below in this module).
- The interface of the external analysis functions has been changed. It now looks like



## D.2 2-D fields modifications

The C preprocessor statements have the following form:

```
#ifdef VARIABLE_NL
    <newcode>
#else
    <oldcode>
#endif
```

The following (extra or new) global variables are declared:

- `int NL[MAXPARMS]`, to hold the number of levels for each parameter. Note that the lowercase 'l' in the old single integer `NL` has been changed deliberately to make the compiler generate a lot of errors, which has helped in finding the places where modifications had to be made.
- `int MaxNL`, contains the *maximum* of all the numbers of levels of all physical variables. `MaxNL` represents the size of the viewbox in levels and must be used in every operation that needs the height of the box.
- `int WindNL`, contains the *minimum* number of levels of the three parameters,  $U$ ,  $V$  and  $W$ , which are used for wind trajectories and wind slices. This variable is introduced for safety only. In practice, the user could present VIS-5D with a file containing different sized  $U$ ,  $V$  and  $W$  grids. Using the `WindNL` variable, VIS-5D will only use the levels available for all three parameters.

The following list of modification descriptions is alphabetically ordered by module, and the modifications made to each module are listed from top to bottom. Line numbers have been omitted, because these are subject to change. Instead modified functions are listed between parentheses.

### analysis.c:

(`compute_analysis_variable`)

- Send a `NL` value (number of levels) for each parameter instead of just one at the beginning to the external analysis functions. The code that handles reception of the data can be found in the module `extmain.c`.
- Send a variable sized grid to the external analysis function. The size (in floats) is given by the formula  $Nr * Nc * NL[ip]$ , where `Nr` is the number of rows of the grid, `Nc` the number of columns and `NL[ip]` the number of levels of the current parameter (`ip`).
- Declare a new integer variable to receive an extra integer from the external function: the number of levels in the grid it will send thereafter.

- The appropriate `MAKECOLOR` macro must be defined. For HP systems this the same macro as for SGI and IBM systems.

**extmain.c:**

- On HP systems the `signed` keyword is invalid in the typedef for `int_1`.

**globals.h:**

- Define the symbol `SINGLE_TASK`.
- Define a new type `int_1` as `char`.

**graphics.c:**

`graphics.c` contains all 3-D graphics library related code. Most changes are related to VOGL, see appendix D.4. Currently the following parts are HP specific:

- Initialization of integers holding the current screen size,
- Setting of file format flags VIS-5D can save (X-Windows dump for HP),

**main.c:**`main()`

- The symbol `RLIMIT_CORE` is not defined for HP. Therefore the core dump disabling has been turned off for HP.
- The `-font` command line option is available on HP, so notify the user when the command line options are printed.
- Scan the command line arguments for the `-font` option.
- Call the main processing function `main_loop()` for HP, because HP is a single-threaded system.

**matrix.h:**

- Include `x.h` to define the `Matrix` type.

**x.h:**

- Include the graphics library and X-Windows header files.

## Appendix D

# VIS-5D modifications

This appendix contains high level descriptions of the modifications made to VIS-5D. The accent lies on the *reasons* for the modifications. For details about the implementation of the changes, the reader is referred to the source code of `vis5d`. All changes have been marked by preprocessor statements for easy finding of the changes and backward compatibility (an old version of VIS-5D without this modifications can easily be restored this way should problems arise due to the modifications). The changes are ordered alphabetically by module name. The name of the modified function is listed between parentheses.

### D.1 HP port modifications

#### D.1.1 LUI changes

`lui.c`:

(`LUI_MoveResizeWindow`)

- Set `XSizeHints` of the window by calling `XSetNormalHints` and `XSetStandardProperties`. The position and size must be set, but also the maximum size of the window. Otherwise all VIS-5D windows can be resized to full screen size.

(`LUI_ResizeWindow`)

- Same as the change for `LUI_MoveResizeWindow`, but only the size and maximum size have to be set, not the position.

#### D.1.2 `vis5d` changes

`cw.c`:



### C.2.6 grib2vis

**process\_cmdline\_arguments:** Parse the command line arguments and store the results in global variables. If a variable is not set, a default value will be taken if appropriate, otherwise an error is reported.

**init\_grid\_header:** Allocation and initialization (where possible) of the GRID file header.

**write\_grid\_header:** Write the GRID file header to a file.

**write\_grid:** Write a grid to the current output file.

**print\_grid\_info:** Print selected contents of a specified grid information structure.

**count\_timesteps:** Determine the number of time steps in a specified time range.

### C.2.4 `cutil`

The headers of the `cutil` functions can be found in the header file `cutil.h`.

**yymmdd2centuryday:** Convert the date in `yymmdd` format to century day.

**centuryday2yymmdd:** Convert the century day to the date in `yymmdd` format.

**hhmm2minutes:** Convert time in `hhmm` format to minutes since midnight.

**minutes2hhmm:** Convert the minutes since midnight to the time in `hhmm` format.

**ialloc:** Allocate float buffer and terminate if error occurred.

**ifree:** Free float buffer and warn if error occurred.

**icheck:** Check if pointer equals NULL and warn when this is the case.

**reorder\_matrix:** Reorder the input matrix from row-major to column-major.

**mirror\_matrix:** Mirror the matrix around the horizontal axis.

**sort:** Sort an array of integers in ascending or descending order.

**reportminmax:** Calculate the minimum and maximum of data and report them.

**getstring:** Read a string enclosed in double quotes from buffer.

### C.2.5 `futil`

These are functions that are not in the KNMI libraries or are a small section of functions from the KNMI libraries.

**calcq:** Calculate the relative humidity.

**iiydd:** Conversion from century day (days since 1 januari 1900) to the date in `yydd` format.

**setfortrancommon:** Set the variables to indicate the presence or absence of physical parameters. This function should first be called before using the interpolation routine `intpre`. Physical variables not present will be calculated by `intpre` if possible.

**conphys:** Define the physical constants used by the interpolation routines.

**abort:** A routine to simulate the abort function (which is not known to the HP FORTRAN compiler).

**getarg:** A routine to simulate the command line argument processing function `getarg` (not known to the HP FORTRAN compiler).

### C.2.2 grib

The headers of the grib functions can be found in the header file `grib.h`.

**grib\_parnumvalid:** Private test whether a passed physical variable is within the range of the current parameter table size. Returns TRUE when the parameter number is valid, FALSE otherwise.

**grib\_shortcode:** Return pointer to the short parameter name if the parameter number is valid. Otherwise a pointer to "N/A" (Not Available) is returned.

**grib\_unit:** Return pointer to the unit of the passed parameter if the parameter is within range. Otherwise a pointer to "N/A" is returned.

**grib\_parname:** Return pointer to the full parameter name if the parameter number is valid. Otherwise a pointer to "N/A" is returned.

**grib\_listparameters:** Print a list of physical variables as defined in section 2 of the grib definition.

**grib\_parnumfromname:** The parameter name that is passed invoking the function is returned as the parameter number if it exists.

### C.2.3 interpol

The headers of the interpol functions can be found in the header file `interpol.h`. For more information about coordinate systems see section C.2.6.

**interpolate\_calcmoleparameters:** Calculate the half and full model level parameters and the  $\eta$  pressure half and full model level parameters.

**interpolate\_calcindexarray:** Calculate index field that maps output gridpoints to input gridpoints with the help of the KNMI routine `defind`.

**interpolate\_init:** Initialize all global variables used for the interpolation. The information about the gridsize is passed in the Asimof Information structure. The parameters initialized are the data grids of the surface pressure, the surface temperature and the full level parameters temperature,  $U$ ,  $V$  and the specific humidity. Also the model level parameters are calculated. These parameters are used by the interpolation.

**interpolate\_destroy:** Free all allocated blocks and unset all variables set by the routine `interpolate_init`.

**interpolate\_pressure:** Interpolate a 3-D data grid to pressure levels according to the passed Asimof Variable structure input grid and output grid with the help of the KNMI routine `intpre`.

### C.2.1 `asimof`

The information needed for calling the functions `loadfd` and `getfd` is stored in the Asimof Variable (`AsimofVars`) structure. The grid information from an ASIMOF file is stored in the Asimof Information (`AsimofInfo`) structure. The exact headers of the `asimof` routines in this section can be found in the file `asimof.h`. The following routines are implemented in the module `asimof.c`:

**`asimof_allocvars`:** Allocate and initialize an Asimof Variable structure with the parameter values. If an input parameters is `-1`, the default value from `asimof.h` is used. The output is a pointer to an allocated and filled `AsimofVars` structure.

**`asimof_freevars`:** Free all blocks used by the Asimof Variable structure and memory used by the structure itself.

**`asimof_open`:** Open the ASIMOF grib database with the help of the function `loadfd` from the KNMI port library and fill the Asimof Variable structure with all relevant data from the ASIMOF file. Returns the errorcode as returned by `loadfd`.

**`asimof_close`:** Close the ASIMOF grib database connected to the device described by the Asimof Variable structure with the help of the function `asimhc`. Returns the errorcode as returned by `asimhc`.

**`asimof_getfield`:** Get a field from the ASIMOF database file with the help of the function `getfd`. Returns the errorcode as returned by `getfd`.

**`asimof_imdi`:** Returns the “key entry not used” indicator.

**`asimof_setkey`:** Fill the given key with `imdi` (key entry not used) values and insert values given as routine parameters. The function has a variable number of arguments.

**`destagger`:** Destagger data to a grid defined by the Asimof Variable structure. Destaggering of the data is necessary when the grid coordinates are not the same as the grid data coordinates used by `vis5d`.

**`asimof_getdata`:** Extract data fields from the ASIMOF file. The actual reading of the data is done with the function `asimof_getfield`. The parameter and level type of the physical variable to extract can be specified with the function parameters. When no level number is specified, this function extracts all data matching the parameter and level type. When more than one level number is found, the levels are sorted from bottom level to top level. Also the `vis5d` grid information structure is filled with values, known at that point. When necessary, the data array is destaggered with the function `destagger` (see section C.2.6). Optionally the data array is reordered to row major and is mirrored across the horizontal axis (see section C.2.6).

**`asimof_getcoordinates`:** Calculate the GRIB coordinates of an input field using the integer section 2 with the help of the function `gbg2fc`. Returns the number of grid points found.



**fpint:** Vertical interpolation of geopotential from a hybrid model level to pressure levels.

**intpre:** Organization of the interpolation to pressure levels during post-processing.

**omint:** Vertical interpolation of OMEGA (vertical velocity).

**postpp:** Main routine for post-processing to pressure levels and to model levels. This routine also handles the extraction of surface parameters. It calls at about all vertical interpolation routines described in this library. Because the many parameters involved in calling this function, we preferred to call the separate interpolation functions.

**tpint:** Vertical interpolation of the temperature to pressure levels.

**vineta:** Vertical interpolation from one  $\eta$ -level into another  $\eta$ -level.

**xpint:** Vertical interpolation from model levels to a pressure level.

**xtreta:** Organization of the extraction of model level data during post-processing.

### C.1.6 grw1

**as2ddr:** Open an ASIMOF file and construct the DDR from it.

**grclos:** Interface for the function `asimhw`.

**gread:** Read block1 and block2 from an ASIMOF file and print the DDR.

**gropen:** Interface to the routine `as2ddr` and ASIMOF.

**gwclos:** Interface for the routine `asimhc`.

**gwoopen:** Load block1 and block2 from an ASIMOF file and construct the DDR.

**gwrite:** Prints the DDR for a GRIB coded HIRLAM field file and writes the blocks to an ASIMOF file.

**priddr:** Prints the DDR for a GRIB coded HIRLAM field file.

## C.2 RUL libraries

Information and definition of the GRID file header and the 3-D grid information block can be found in the header file `grid5d.h`.

**gribex:** Coding and decoding of GRIB file format messages.

**minmax:** Calculate minimum and maximum values from an array of floating point numbers.

### C.1.3 util

**cd2dat:** Converts century day (number of the day in this century) to day/month/year format (ddmmyyyy).

**convdt:** Converts day/month/year format (ddmmyyyy) to century day (dddd).

**diasim:** Diagnose an ASIMOF file. **diasim** can be run with two options. The option **-h** will print an extensive header and **-s** will print the statistics of the field data. No option will just print the GRIB section 2 values of all variables.

**gregor:** Converts Julian day number into Gregorian (normal) date (year, month, day).

**ic2ymd:** Converts century day to day, month and year.

**idat2c:** Converts a given day, month and year into century day.

### C.1.4 vari

**gb2gcc:** Transform coordinates described in input section 2 (GRIB format) to coordinate system described in output section 2.

**gb2llc:** Transform from one latitude/longitude grid to another. This function is mainly used to transform from or to the geographical system.

**gb2lpc:** Transform from geographical grid to polar stereographical grid.

**gb2plc:** Transform from polar stereographical grid to latitude/longitude grid.

**gb2slc:** Shift pole of regular latitude/longitude grid.

**gb2tlc:** Stretch regular latitude/longitude coordinates toward the north pole.

### C.1.5 prpo

**defind:** Definition of a pointer field to be used for the extraction of grid-point values in a sub-area from a grid-point field defined in a larger area.

**destag:** Destaggering of wind components to fit the analysis program conventions. This program is used when the grid-points of the wind components are not the same as the grid-points of the system.

**etaeta:** Vertical interpolation from one  $\eta$ -level into another  $\eta$ -level.

# Appendix C

## Libraries

In this appendix the most important routines from the KNMI and RUL libraries are listed. For more detail the reader is referred to the inline documentation of the libraries.

### C.1 KNMI libraries

#### C.1.1 port

**asimhc:** Close an ASIMOF file.

**asimhr:** Read an array from the ASIMOF file. If the key passed along does not exist in the database an error is returned.

**asimhw:** Write an array into the ASIMOF file. If the key passed along is not unique an error is returned.

**getfd:** Get a coded field from the ASIMOF database and decode a field from GRIB. This function calls the functions **asimhr** and **degrib** (see appendix C.1.2).

**loadfd:** Open a direct-access file as GRIB database. If the file exists the contents will be extracted, else the file will be opened with an empty table of contents.

**putfd:** Convert a field into GRIB format and place it in an ASIMOF database. This function calls the functions **asimhw** and **engrib** (see appendix C.1.2).

#### C.1.2 gcod

**degrib:** Decoding of data or identification sections (section 1, 2, and 3) from GRIB format.

**encode:** Encoding of data in GRIB format.



**get5d:** Read a GRID file to obtain the grid info structures.

**iggt3d:** Read a data grid from a GRID file, returning the number of data points read. This function is used from within `vis5d`.

For a description of the VIS-5D utilities, see section 2.4.1.

## B.4 Little User Interface (LUI) library

The LUI library contains the following modules:

**browser:** File browser module.

**button:** Provides functions to create and maintain all sorts of buttons, like toggle buttons, one-shot buttons, etc. Buttons can be grouped in button pads.

**dial:** Provides functions to create and maintain dials.

**dialog:** Provides functions to create dialog boxes which can contain a message and OK and Cancel buttons.

**event:** Event dispatcher. Each window created by the LUI library can specify events and a call back function. This call back function is called only for the specified events.

**label:** Provides text labels and type-ins.

**lui:** Main LUI module, that contains the LUI initialization and most functions dealing with windows. It also contains various color transformation functions.

**pixmaps:** Contains pixmap definitions for radio widgets.

**popup:** Handles pop-up menus.

**slider:** Slider support. The user can use sliders to adjust a value used by the program.

INTEGER*4 DATE	Datestamp of variables in HHMMSS format i.e. hours*10000 + minutes*100 + seconds.
INTEGER*4 TIME	Timestamp of variables in YYDDD format i.e. year*1000 + century day.
REAL*4 SOUTH, WEST, BOTTOM	Geographic location of grids (SOUTH latitude in degrees, WEST longitude in degrees, BOTTOM coordinate in kilometres).
REAL*4 DELTALAT, DELTALON, DELTAHGT	Grid increments. DELTALAT and DELTALON in degrees, DELTAHGT in kilometres.

A user function should return 0 if completed successfully and a non-zero value if an error occurred.

The `extmain` module contains code to receive the input data from `vis5d` through a socket and send the output grid back to `vis5d`. The user functions is called from `extmain`.

### B.3 Utilities

The most important utility modules include the following:

**comp5d:** Conversion of McIDAS GRID files to COMP files, the compressed `vis5d` file format. There are several `comp5d` programs, one for each type of COMP file (see appendix A.4). The structure of the `comp5d` programs is as follows:

- Read the first time step grid info structures, checking that every physical variable has the same number of levels, etc.
- Read the following time steps grid info structures, checking if the same physical variables are encountered as in the first time step.
- Allocate memory to hold input grids,
- Read input grids from input GRID file(s),
- Compress grid data,
- Write grids to output COMP file,
- Write header of output COMP file.

Note that the time steps must be ordered chronologically and the order of the physical variables must be the same for each time step.

**compinfo:** Read and display contents of COMP files.

The rest of the programs can only handle the McIDAS GRID file format. The most important subroutines are:

**slice:** Slice handling functions. This module processes all events that occur when `vis5d` is in slice mode. The most important function is to find the correct slice and moving it, when the user drags from a specific position in the 3-D window.

**socketio:** Provides functions to send and receive data through a socket. These functions are used to send to and receive from the external analysis functions.

**sync:** Provides semaphore functions for systems that allow multi-threaded execution.

**topo:** Functions for reading and drawing the topography.

**traj:** Trajectory module. Trajectories are traced using this module.

**vtmcp:** Module dedicated solely to the calculation of iso-surfaces.

**work:** This module contains the actual routines to extract contour, color and wind slices, an interface to the iso-surface and trajectory module.

**zcont:** Contouring module. Given an input slice of data and a contour interval, it calculates the position of contours on the slice and outputs vector arrays containing the contour positions.

The modules `gui.c` and `cw.c` contain all the user interface functions (making use of the LUI library). For the direct reading of McIDAS GRID files, `vis5d` uses the McIDAS GRID file library.

## B.2 External User functions

The Fortran modules must all contain a function called `USERFUNC` with the following interface:

```

INTEGER FUNCTION USERFUNC( OUTGRID, INGRID, NR, NC, NL, NP,
*                          NAMES, DATE, TIME,
*                          SOUTH, WEST, BOTTOM,
*                          DELTALAT, DELTALON, DELTAHGT )

REAL*4 OUTGRID(NR,NC,NL)      Output grid calculated by the function.
REAL*4 INGRID(NR,NC,NL,NP)   Input grids, all physical variables,
                              one time step.

INTEGER*4 NR                  Number of latitude points (rows) in
                              input grids.

INTEGER*4 NC                  Number of longitude points (columns) in
                              input grids.

INTEGER*4 NL                  Number of levels in input grids.
INTEGER*4 NP                  Number of physical variables.
CHARACTER*8 NAMES(NP)        Names of variables. Padded with spaces.

```

function of `vis5d` needs a data grid, it must be requested from the grid module. Grids are stored in a compressed form in memory and are decompressed when requested. The decompression routine uses two values which are also stored in a compressed grid file: `ga` and `gb`. These values are used to scale and map floating point values from/to 8-bit integers. This conversion cannot be done without losing precision. `vis5d` is able to read the original GRID files with the floats as well (see also appendix A.4).

**gui:** The GUI (Graphical User Interface) module takes care of the creation and maintenance of all windows, except for the 3-D window which is created in the graphics module. GUI processes all user input from the mouse and the keyboard and updates appropriate global variables contained within the globals module which determine how the data set is displayed. GUI makes extensive use of the User Interface Library, LUI, discussed in section B.4.

**main:** Entrypoint of `vis5d`. After initialization, processing command line parameter, reading of the data set, etc. an eternal loop is executed processing user input.

**map:** Contains functions to read a map file, draw the map in the 3-D window and updating the display, if necessary. A map file usually contains a number of line segments defining land-sea borders (and country outlines). The elevation is calculated from the topography.

**matrix:** `vis5d` uses its own library routines of matrix operations to create and modify the transformation and projection matrices used to map object coordinates into the 3-D window (see section 3.1.1). Matrix multiplication, concatenation, copying, inversion, vector normalization and vector-matrix multiplication is provided.

**memory:** Memory management module. `vis5d` functions needing large amounts of memory should call one of the functions from this module. `vis5d` tries to retain all requested graphics in memory. When allocation of a requested block fails however, the memory management module will try to create enough free memory by deallocating the least-recently viewed graphics. A number of functions is provided. Memory allocated using `pallocate` (permanent allocate) will never be deallocated, contrary to memory allocated by `allocate`.

**queue:** Implementation of a task queue. When only one processor is available, the requested graphic will be calculated immediately. On systems with multiple processors, multi-threaded execution is possible. Each requested graphic is put into the task queue. Processors will calculate the graphics in the queue multi-threaded. Access to graphics is made exclusive through the use of semaphores.

**render:** The render module contains the drawing functions, which are all based on the graphics module. The box, slices, trajectories, slice ticks, clock, etc., are drawn from within the render module.

**save:** VIS-5D allows the user to save the current state of the 3-D window to a file. Data saved includes topography colors, labels and graphics with their colors and positions. Such a file can later be restored.



## Appendix B

# VIS-5D source code structure

### B.1 VIS-5D main source code

`vis5d` version 3.1 is split into several modules. The most important modules include (in alphabetical order):

**analysis:** Activation of activate external user analysis functions, which can calculate new grids from the available data. Analysis programs must reside in a special directory, which is scanned by `vis5d`. When an analysis function is activated, all available data (one time step at the time) is sent through the socket to the analysis program. The analysis program can calculate a new grid of data, derived from the existing data. This new grid is then sent back through the socket to the analysis module of `vis5d`. When successful, the new grid is installed and can be visualized as usual.

**extmain:** Wrapper for `vis5d` external analysis code. It is linked separately with every external analysis function. It receives the grid description and data through the socket connection with `vis5d` and calls the user analysis function. The grid description and data are passed as parameters to the analysis function. The interface is described in section B.2.

**globals:** Contain declarations of globally accessible variables and constants. The most important variables include the grid dimensions and the grid data pointers.

**graphics:** All 3-D graphics library related code is contained in the graphics module. It contains functions to make the 3-D window, initialize the graphics module/library, draw polylines, text, triangle meshes, quadrilateral meshes, etc. All functions in the graphics module are based upon functions from the 3-D graphics library (XFDI for the Stellar, GL for the SGI and IBM). Functions from other modules call functions from the graphics module if they want something drawn in the 3-D window.

**grid:** All grid management functions are contained in the grid module. `grid` will determine the file format of an input file and read the data sets contained within the file. If a

```

int offset          Offset in 4-byte ints from start of file.
int length          Length of segment data in 4-byte ints.

```

- Coordinate information for each segment. The position and length of the data is obtained from the directory structure. The data consists of an array of  $length/2$  ( $lat, lon$ ) pairs in  $10^{-5}$  degrees.

## A.6 VIS-5D topography file format

A VIS-5D topography file contains the surface elevation and land/sea mask of a specific part of the earth surface. Each file starts with the following header:

```

char id[40]         ID string "TOP0".
int westlon         West longitude in 1/100's of degrees.
int eastlon         East longitude in 1/100's of degrees.
int northlat        North latitude in 1/100's of degrees.
int southlat        South latitude in 1/100's of degrees.
int rows (nlat)     Number of rows (latitude).
int columns (nlon)  Number of columns (longitude).

```

Positive latitude degrees are north, positive longitude degrees are west.

After the header the topography data follows:

```
short data[rows][columns]
```

Row [0] corresponds to the north edge of the topography, Row [ROWS-1] corresponds to the south edge, Column [0] corresponds to the west edge and Column [COLUMNS-1] corresponds to the east edge. The `shorts` of the topography data consists of the surface elevation in metres multiplied by 2 and with 1 added if the location is water.

int	GRIDTIMES	Maximum number of time steps.
int	NumTimes	Number of time steps.
int	NumParms	Number of parameters.
int	Nr, Nc	<i>Horizontal</i> grid dimensions (global i.e. for all grids!).
int	NL[NumParms]	A number of levels for every parameter in the file.
float	LatInc, LonInc	Horizontal grid increments in latitude and longitude direction.
int	LevelType	Type of vertical coordinate.
float	Height[NL]	Levelvalues (unit depends on leveltype).
char	ParmName[GRIDPARMS][8]	Names of parameters.
float	MinVal[NumParms]	Minimum values of parameters.
float	MaxVal[NumParms]	Maximum values of parameters.
int	TimeStamp[GRIDTIMES]	Timestamp of time steps.
int	DayStamp[GRIDTIMES]	Daystamp of time steps.
float	NorthLat[GRIDTIMES]	North latitude of domain.
float	WestLon[GRIDTIMES]	West longitude of domain.

The data grids vary in size. The size can be calculated using the NL values from the array and are again rounded up to a multiple of four.

```

for each time step:
  for each parameter:
    int McFile      McIDAS GRID file number with original data.
    int McGrid      Grid number in GRID file.
    float ga[NL[ip]]
    float gb[NL[ip]]  Decompression values for each level. ip is
                      the parameter index.
    float griddata[*] Data for this grid consisting of Nr*Nc*NL[ip]
                      floats rounded up the nearest multiple of 4.

```

## A.5 VIS-5D map file format

A VIS-5D map file holds information of land/sea borders etc. as a number of continuous polylines (map segments). A map file has the following structure:

- Number of map line segments:

```
int nsegs
```

- Directory structure containing for *every* segment:

```
int d0, d1, d2, d3      Not used.
```

int	GRIDTIMES	Maximum number of time steps.
int	NumTimes	Number of time steps.
int	NumParms	Number of physical variables.
int	Nr,	Number of rows (latitude points).
	Nc,	Number of columns (longitude points).
	Nl	Number of levels.
float	LatInc, LonInc	Horizontal grid increments in latitude and longitude direction (in degrees).
float	Height[Nl]	Heights of levels (in km).
char	ParmName[GRIDPARMS][8]	Names of physical variables.
float	MinVal[NumParms]	Minimum value of physical variables.
float	MaxVal[NumParms]	Maximum value of physical variables.
int	TimeStamp[GRIDTIMES]	Timestamp of time steps (HHMMSS format).
int	DayStamp[GRIDTIMES]	Daystamp of time steps (YYDDD format).
float	NorthLat[GRIDTIMES]	North latitude of domain (in degrees).
float	WestLon[GRIDTIMES]	West longitude of domain (in degrees).

There is now slightly more data for each grid than in the case of the old style format, due to the extra decompression values:

```

for each time step:
  for each parameter:
    if (ID=0x80808083)
      int McFile      McIDAS GRID file number with original data.
      int McGrid      Grid number in GRID file.
    fi
    float ga[Nl], gb[Nl] Decompression values for each level.
    float griddata[*]  Data for this grid consisting of Nr*Nc*Nl
                      (rounded up to the nearest multiple of 4) floats.

```

#### A.4.3 Variable NL format

This is the new style format made to enable the visualization of single-level fields. The header looks like:

- Variable NL format, new style format with layer by layer compression, GRID file read, and a variable number of levels for each parameter.

#### A.4.1 Old style format

Grid compression is done grid by grid (entire grid at the time). The maximum number of time steps (GRIDTIMES) is 300 for ID=0x80808080 and 400 for ID=0x80808081. The maximum number of parameters (GRIDPARMS) is 20 for ID=0x80808080 and 30 for ID=0x80808081.

The following type-lengths are used: **char** is 1 byte, **int** is 4 bytes, **float** is 4 bytes.

The header looks like:

int	NumTimes	Number of time steps.
int	NumParms	Number of physical variables.
int	Nr,	Number of rows (latitude points).
	Nc,	Number of columns (longitude points).
	Nl	Number of levels.
float	LatN, LonW	Geographic location in degrees.
float	HeightTop	Height of highest grid level in km.
float	LatInc, LonInc	Horizontal grid increments in latitude and longitude direction in degrees.
float	HeightInc	Vertical grid increment in km.
int	DayStamp[GRIDTIMES]	Daystamp of time steps in YYDDD format. Note that the file always contains GRIDTIMES DayStamps, despite the number of time steps.
int	TimeStamp[GRIDTIMES]	Timestamp of time steps in HHMMSS format.
char	ParmName[GRIDPARMS][8]	Names of parameters.

Then the data follows, in the same order as the time stamps and variable names:

```
for each time step:
  for each parameter:
    float ga, gb      Grid decompression values
    float griddata[*] Data for this grid consisting of Nr*Nc*Nl
                      (rounded up to the nearest multiple of 4) floats.
```

#### A.4.2 New style format

Compression of the grids is done layer by layer and extra information is contained within the file to retrace the origin of the grids (the McIDAS GRID file and grid number in the file) to enable `vis5d` to read the original uncompressed grid data (only for ID=0x80808083). The header looks like:

coordinate unit to `vis5d`. Therefore the field `pad4[]` is *replaced* by:

```
int  leveltype  Indicator of unit of vertical coordinate system on
                    which grid is defined.
int  pad4[30]   Reserved (now one integer less).
```

The following values for the `leveltype` are currently valid:

```
HEIGHT_LEVELS (105)   This is the default leveltype and set for old
                        style COMP files. The unit of the vertical
                        coordinate is km.
PRESSURE_LEVELS (100) Indicates equidistant pressure levels. Vertical
                        interpolation between levels will be done using the
                        logarithm of the pressure. The unit of the vertical
                        coordinate is Pa.
```

The contents of the `top` and `topinc` entries must be changed. These will contain the pressure level of the highest level and the pressure level increment both multiplied by 1000 if the `leveltype` is `PRESSURE_LEVELS`. Otherwise the contents will be the same as before (height level \* 1000 and height increment \* 1000).

The data grids are stored as 4-byte floating points numbers. A data value *greater* than  $1e30$  indicates *missing* data (usually the value  $1e35$  is used).

## A.4 COMP file format

The COMP (compressed) file format is the only format that can be read directly into `vis5d`. VIS-5D is supplied with skeleton example programs for converting data into COMP file format and utilities for converting the GRID file format (see appendix A.3) into COMP files. During this conversion all data points in the grids are compressed from 4-byte floating points values to 8-bit integers, hence the name of the format. Because this compression results in loss of accuracy it is possible to read grids from the uncompressed GRID files when they are needed for calculations by `vis5d`. The compression uses two floating point values:  $g_a$  and  $g_b$ . These values are equal to  $g_a = \frac{250}{(g_{max} - g_{min})}$  (the range) and  $g_b = -g_a * \frac{(g_{max} + g_{min})}{2}$  (the offset) of the input grid data. New compressed data values are then calculated using the formula  $g_{new} = g_a * g_{old} + g_b$ .

There are several COMP file formats, which are identified by an ID field in the first 4 bytes in the file. The following ID fields are valid:

- Old style format with grid by grid compression (ID's 0x80808080 and 0x80808081),
- New style format with layer by layer compression and GRID file read,

and the new format

char	ident[32]	File description/identification.
int	nproj	Project number?
int	creation_date	Date created in YYDDD format.
int	maxsize	Number of data points in largest 3-D grid.
int	numgrid	Number of 3-D grids in this file.
int	firstgrid	Location of first grid's data as offset in 4-byte words from start of file.
int	pad[51]	Pad to 256 bytes.

The gridinfo structures contain the following fields each:

int	size	Number of data points / words of data (=nr*nc*nl).
int	nr	Number of rows (latitude points).
int	nc	Number of columns (longitude points).
int	nl	Number of levels.
int	iword	Location of data as offset in 4-byte words from start of file.
int	date	Grid date stamp in YYDDD format.
int	time	Grid time stamp in HHMMSS format.
int	pad1	Reserved.
char	name[4]	4-character name of physical variable.
char	units[4]	4-character unit description.
int	pad2[11]	Reserved.
int	itype	Always 4.
int	latn	North latitude * 10000.
int	lonw	West longitude * 10000.
int	latinc	Latitude increment * 10000.
int	loninc	Longitude increment * 10000.
int	pad3[4]	Reserved.
int	ihtype	Always 1.
int	top	Top height * 1000.
int	topinc	Height increment * 1000.

Because the grid info structure contains the location of the grid data in the file, it is possible to create files with a gap between the grid info structures and the data. This way space can be reserved for future headers.

For an original McIDAS GRID file, the gridinfo structures are padded to 256 bytes in length by adding:

```
int pad4[31] Reserved.
```

In order to make `vis5d` perform the correct type of vertical interpolation depending on the coordinate system automatically, it must know the `leveltype`, i.e. the unit of the vertical coordinate on which the grids are defined. Because the conversion from GRIB to COMP takes the indirect route via GRID (to retain the functionality of the utilities supplied with VIS-5D) it is necessary to modify the GRID format to pass information about the vertical

predetermined bit map provided by the centre that made the GRIB message. The rest of the section contains the bit map. This section is optional.

**Section 4:** The binary data section starts with three octets describing the length of the section. The next octet contains a flag setting the representation of the binary data. Octet 5-6 contain the scale factor for the binary data. Octet 7-10 are set to the minimum of the binary data. The following octet has the number of bits containing each packed value. Then from octet 12 on, the actual data is described, depending on the flag values in octet 4.

**Section 5:** The end section, ending the GRIB message, is always 4 octets long and contains the character coded message 7777.

## A.2 ASIMOF file format

The ASIMOF file format was designed as a database file system at KNMI, as a standard interface between their numerical models and presentation systems. It was adopted as the standard for the reference HIRLAM system. It is used at KNMI; many other HIRLAM institutes use it or are considering its use. For more detailed information about the format, see the inline documentation of the ASIMOF routines of the HIRLAM system. The ASIMOF file format is computer dependent.

## A.3 GRID file format

The GRID file format is used by the McIDAS system (VIS-5D is a stand alone subsystem of McIDAS, see [Smit75]). GRID files cannot be read directly by `vis5d` but they can be converted to the COMP file format (appendix A.4) which is currently the only format `vis5d` can read. There exists a compile time option Read-On-Demand that enables `vis5d` to read GRID files but even then the header of the COMP file that is made from the GRID file(s) must be available.

The global structure of a GRID file is:

1. header (size is 256 bytes).
2. `numtimes*numparms` gridinfo headers (size of each is 256 bytes). `numtimes` is the number of time steps, `numparms` is the number of physical variables.
3. space for future headers.
4. `numtimes*numparms` data grids (each data point is a 4-byte float).

The following type-lengths are used: `char` is 1 byte, `int` is 4 bytes, `float` is 4 bytes.

The McIDAS GRID file header consists of the following entries:



<i>Octet No.</i>	<i>Contents</i>	<i>Description</i>
1-3	46	Number of octets in section 2
4	0	No vertical parameters
5	255	No list present
6	10	Rotated latitude/longitude grid
7-8	92	Points on parallel
9-10	81	Points on meridian
11-13	-24500	Latitude first grid point
14-16	-25500	Longitude first grid point
17	128	Direction increments given Earth assumed spherical Positive U and V in easterly and northerly directions
18-20	15500	Latitude last grid point
21-23	20000	Longitude last grid point
24-25	500	Latitude direction increment
26-27	500	Longitude direction increment
28	64	First grid point left down bottom
29-32	0	
33-35	-30000	Latitude southern pole
36-38	0	Longitude southern pole
39	0	Angle of rotation
43-46	0	

Table A.3: GRIB format Section 2 *example* values for HIRLAM

<i>Octet No.</i>	<i>Contents</i>
1-3	Length of section
4	NV — Number of vertical coordinate parameters
5	PV — Location of list of vertical coordinate parameters, if present; or PL — Location of list of numbers of points in each row (if octet 4 > 0), if present; or 255 if neither are present
6	Data representation type (grid type)
7-8	Number of points along a parallel
9-10	Number of points along a meridian
11-13	Latitude of first grid point (in millidegrees)
14-16	Longitude of first grid point (in millidegrees)
17	Resolution and component flags
18-20	Latitude of last grid point (in millidegrees)
21-23	Longitude of last grid point (in millidegrees)
24-25	Latitude direction increment (in millidegrees)
26-27	Longitude direction increment (in millidegrees)
28	Scanning mode – direction of increments
29-32	Reserved – set to zero
33-35	Latitude of southern pole (in millidegrees)
36-38	Longitude of southern pole (in millidegrees)
39	Angle of rotation
43-52	Reserved for stretched latitude/longitude grid
PV	List of vertical coordinate parameters
PL	List of numbers of points in each row

Table A.2: GRIB format Section 2 – grid description section for rotated latitude/longitude grid type as used by HIRLAM

<i>Octet No.</i>	<i>Contents</i>
1-3	Length of section
4	Version number, currently 1
5	Identification of centre
6	Generating proces identification number
7	Grid definition number according to Volume B of publication WMO - No. 9
8	Flag signaling if section 2 is included (bit 1 = 1) and if section 3 is included (bit 2 = 1)
9	Indicator of parameter
10	Indicator of leveltype
11-12	Height, pressure, etc. of levels
13	Year of century
14	Month
15	Day
16	Hour
17	Minute
18	Indicator of unit of time range
19	Number of time units
20	Same as octet 19 or time interval, undergoing averaging or accumulation
21	Time range indicator
22-23	Number included in average (if octet 21 = 3 or 4) otherwise set to zero
24	Number missing from averaging or accumulation
25	Century of reference time of data
26	Reserved – set to zero
27-28	Units decimal scale factors
29-40	Reserved for future use – may not be present
41-nn	Reserved for originating centre use

Table A.1: GRIB format Section 1 – product definition section

**Section 0:** The indicator section contains 8 octets where the first 4 octets are character coded as GRIB, the next 3 octets contain the length of the entire GRIB message and the last octet represents the GRIB edition number.

**Section 1:** The product definition section starts with three octets, describing the length of this section. The rest of this section contains the information of when, where, by whom, etc. the GRIB message has been made. More specific information about this section can be found in table A.1.

While working with HIRLAM data, we encountered the following level types (octet 10):

- 100 – Isobaric surface: Grid values are given at a fixed pressure level. Octet 11-12 contain the pressure in hPa.
- 102 – Mean sea level: Grid values are given at mean sea level. Octet 11-12 are set to zero.
- 103 – Specified altitude: Grid values are given at a specific altitude. Octet 11-12 contain the altitude in metres.
- 105 – Specified height level (above ground): Grid values are given at on a specific height above the ground. Octet 11-12 contain the height in metres.
- 109 – Hybrid level: Grid values are given at model levels (see section 4.3). Octet 11-12 contain the level number.

**Section 2:** The grid description section starts with three octets indicating the length of this section. The rest of this section describes the grid as set in octet number 6 of section 2. Because HIRLAM uses a rotated latitude/longitude grid (octet 6 = 10) octet 7-52 contain specific information for that type of grid. More information about section 2 can be found in table A.2. For HIRLAM, an example section 2 is given in table A.3.

The rotated latitude/longitude coordinate system that is used by HIRLAM, formed by a general rotation of the earth, can be described by three parameters:

- The geographic latitude in degrees of the southern pole ( $\alpha_p$ );
- The geographic longitude in degrees of the southern pole ( $\beta_p$ );
- The rotation of the new polar axis (positive in easterly direction). This new polar axis can be obtained by first rotating the sphere through  $\beta_p$  degrees about the geographical polar axis and then rotating through  $(90 + \alpha_p)$  degrees so that the southern pole moved along the (previously rotated) Greenwich meridian.

**Section 3:** The bit map section starts with three octets, used for the length of this section. Octet 4 is used for giving the number of unused bits at the end of section 3. The next two octets are the table reference: if they are zero, a bit map follows, else it refers to a

# Appendix A

## File formats

In this appendix all relevant file formats are explained. HIRLAM produces files in the GRIB file format, which is the world standard format for storage of meteorological data. The ASIMOF file format is used at KNMI to store several GRIB fields in one file. The remaining file formats are related to VIS-5D (GRID, COMP, map and topography file formats).

### A.1 GRIB file format

GRIB (gridded binary) as described in FM 92-IX Ext.-GRIB [WMO91] is a file format for processed data in the form of grid-point values expressed in binary form. In this paragraph the GRIB edition 1 format will be explained.

In GRIB the data are represented in a sequence of an even number of octets (1 octet = 8 bits). This representation is independent of any particular machine representation. The octets of a GRIB message are grouped in 6 sections:

- Section 0: Indicator section
- Section 1: Product definition section
- Section 2: Grid description section
- Section 3: Bit map section
- Section 4: Binary data section
- Section 5: End section

Every section contains an even number of octets. If not, the section is appended with bits set to zero.



# Bibliography

- [Baas93] Baas, S.A.; Meteorological Visualization using apE (a toolkit for visualization), Internal report Leiden University, 1993.
- [GLman] Fisher, S., M. Heinrich; Graphics Library programming guide, document version 2.0, Silicon Graphics Incorporated, 1990.
- [Gour71] Gouraud, H.; Continuous shading of curved surfaces, IEEE Computer Graphics and Applications C-20(6), pp. 60-69, 1971.
- [Hibb85] Hibbard, W.L., R. Kraus, J.T. Young, 3-D Weather Displays Using McIDAS, Preprints, ICHPSMOH, Los Angeles, American Meteorological Society, 153-156, 1985.
- [Hibb89] Hibbard, W., D. Santek; Visualizing large data sets in the earth sciences, IEEE Computer Graphics and Applications 22(8), pp. 53-57, 1989.
- [Hill90] Hill Jr., F.S.; Computer Graphics, Maxwell MacMillan International Editions, 1990.
- [LoCl87] Lorensen, W.E., H.E. Cline; Marching Cubes: a high resolution 3D surface construction algorithm, Proceedings Siggraph '87, Computer Graphics 21(4), pp. 163-169, 1987.
- [Smit75] Smith, E.A., The McIDAS System, IEEE Transactions on Geoscience Electronics, GE-13, 123-134, 1975.
- [WMO91] Secretariat of the World Meteorological Organization; Manual on Codes, Volume I (Annex II to WMO Technical Regulations), International Codes, Part B - Binary Codes, 1988 edition, Supplement No. 3, Geneva, Switzerland, 1991.

- It would be more realistic to insert “missing” values below the orography instead of extrapolating the data values. This is not appropriate for all variables. In general all full model level fields should be substituted with “missing” values below the orography, but not the pressure at at mean sea level.
- The vertical coordinate of the topography should be converted from metres to Pascal. Because the pressure is not constant at the earth’ surface, this implies a change to `vis5d` to calculate a different topography every time step.
- Logarithmic interpolation of vertical slices is not yet implemented.
- Enhance VIS-5D in such a way that the user can determine the place of each colormap legends, in the same way as the labels can be positioned.
- For HP hardware platforms at Leiden University, the Starbase Graphics Library can be tested for use with VIS-5D. Starbase is a HP specific library and is probably faster than VOGL (but not portable).



## Chapter 8

# Future work

The recommendations listed in this chapter can be used as a guide to future work. They include required features (see section 1.2 for a list) that are not yet implemented.

- Extend the 1-D point probe in VIS-5D to a 2-D curve probe. A 2-D curve probe should enable the user to manipulate a (spline) curve in the domain being visualized. The value of the physical variables are then plotted as a function of the displacement along the curve.
- In meteorology wind vectors are often visualized using shafts and various attachments. The shafts (line-segment) are all equal in length and point in the opposite direction of the wind vector. The speed of the wind is visualized by attaching several line segment and triangles to the shaft (120 degrees rotated). This can easily be added to VIS-5D.
- When multiple vertical coordinate units are supported, it is recommended to print the unit in the 3-D window near the height axis to avoid misinterpretation of the data. A unit for every physical variable would be useful too.
- A visualization technique not currently supported by VIS-5D is volume rendering. Future versions of VIS-5D will include this technique, but will use transparency blending features from the graphics library. These are currently not supported by VOGL and should be implemented (if the speed of VOGL allows it).
- The visualization of 2-D fields is currently restricted in the way that every 2-D slice is placed in the bottom of the box. Slices can be visualized for arbitrary height, it is the placement that is always at the bottom. The 2-D fields of the KNMI not positioned at the earth' surface include the 2 metre temperature and 10 metre wind vectors. With the current vertical resolution of the vertical slices it is not useful to modify VIS-5D to draw the fields at the correct place, moreso because only one equidistant set of vertical coordinates can be used (all physical variables must be defined on the same grid points).
- The lighting model implemented in VOGL must be enhanced. For instance, spotlights are not supported at the moment.

on any hardware platform as long as a C compiler is available. Note however, that in order to use VOGL with VIS-5D other system requirements (see section 2.2) must also be satisfied.

## Chapter 7

# Conclusions

In order to visualize the output of a numerical weather forecasting model (HIRLAM), the visualization package VIS-5D was chosen. VIS-5D is suitable, although a number of modifications were necessary. Based on our experiences with VIS-5D, the following conclusions can be drawn:

Advantages of VIS-5D are:

- VIS-5D has a very intuitive user interface and enables the user to combine all supported visualization techniques and all available physical variables easily into one image,
- VIS-5D is free (under the GNU licence),
- the source code is supplied with VIS-5D,
- good support. For the (growing) list of users, a mailing-list<sup>1</sup> and e-mail to Bill Hibbard (whibbard@vms.macc.wisc.edu) or Brian Paul (bpaul@vms.macc.wisc.edu) if you would like to be added to the VIS-5D mailing list is available and the authors are very responsive when presented with questions and suggestions,
- VIS-5D is still in development, which means that future versions will have more functionality and keep up to date with respect to new visualization techniques.

A disadvantage of VIS-5D is:

- VIS-5D relies heavily on the existence of a 3-D graphics library. If such a library is not available, VIS-5D cannot be used. This problem has been solved for HP 700 Series by extending the library VOGL (software implementation of the SGI GL libraries) with the functionality needed by VIS-5D. However, a software implementation of a hardware library can never match its speed, so VIS-5D is a bit slow (the reader is referred to chapter 6 for concrete timing values) when using VOGL. VOGL by itself can be used

---

<sup>1</sup>S

11000 lines of code, and ca. 3000 lines were added.

Images showing some of the capabilities of VOGL are included in appendix F.

Because VOGL is quite slow, a number of speed tests were executed on a HP 700/RX. The drawing of color slices is tested because these take the longest time to render. The following times resulted:

Rendered graphics	Time [seconds]
Empty box	ca. 0,04
One horizontal color slice	ca. 6,5
Two horizontal color slices	ca. 11,6

To determine how much time each operation performed by VOGL costs, the operations are split into the following categories:

- drawing (the drawing of pixels on the screen and setting of colors),
- scan conversion (calculation of which pixels to draw),
- polygon clipping,
- miscellaneous (the remaining operations).

The same speed tests were run without execution of one of the listed categories, which resulted in the following:

VOGL categories:	Time relative to normal:
Normal operation	100%
No drawing	66%
No scan conversion, no drawing	50%
No clipping	66%
No clipping, No scan conversion	10%

From this results the conclusion can be drawn, that drawing and clipping are the categories that should be considered first to speed up VOGL.

Colormap legends are implemented. The only restriction is currently the space of the 3-D window. Depending on the width of the values printed next to the legends, up to ca. 7 legends can be displayed. An image displaying a colormap legend is given in appendix F.

## Chapter 6

# Results

This chapter presents the results obtained by using VIS-5D for the visualization of the HIRLAM output.

The required visualization techniques (see section 1.2) are all possible with VIS-5D. To obtain a discretized color mapping, the user must form the colored bands by changing the color map in the RGB widget window of VIS-5D. Wind vector slices can only be visualized if the  $U$  and  $V$  wind components are present (if the vertical wind component  $W$  is present it is used, otherwise it is substituted with zeros). To visualize wind trajectories, all three wind components ( $U$ ,  $V$  and  $W$ ) must be present *and* more than one time step must be available. Currently HIRLAM does not calculate the vertical wind component  $W$  (although the equations to do so are present) which means that wind trajectories cannot be used for the HIRLAM data.

Of the additional required features (see section 1.2) only the 2-D data probe is not yet available, but will be included in future versions of VIS-5D.

The HIRLAM data can be visualized using pressure for the vertical coordinate. This is realized by interpolating the HIRLAM data from HIRLAM model levels to pressure levels and writing the resulting data to a file that can be further processed by VIS-5D. Due to the exponential decrease of pressure with height logarithmic interpolation must be used. For large pressure values (10000 - 100000 Pa) the linear and logarithmic fractions are nearly equal however.

The VIS-5D modifications described in chapter 5 are all implemented successfully (except for the logarithmic interpolation of vertical slices). The source code of `vis5d` consists of ca. 35000 lines of code. The modifications add ca. 1800 lines of code (all modifications, including the graphics module modifications). This number of lines is exaggerated because the old code is retained and every modification is embedded within C preprocessor statements for backward compatibility.

VIS-5D is ported to the HP hardware platform for use at Leiden University. VOGL, a software implementation of the SGI Graphics Library is used for all graphics related functions. VOGL is enhanced with functionality necessary for VIS-5D. The original VOGL consisted of ca.

the gravitational acceleration at the earth's surface. The resulting geopotential height corresponds with the geometrical height (height above mean sea level). The land/sea mask (variable 81, leveltype 105, height 0) can be extracted directly from the HIRLAM climate file. The values must be rounded to the nearest integer (0 is land, 1 is sea).

VIS-5D is adapted to display the box height coordinates in Pascal, but the vertical coordinate of the topography is still in metres. If the vertical coordinate should be changed to pressure coordinates, `vis5d` should also be enhanced to calculate a new topography every time step, because it depends on the non-constant surface pressure.

## 5.5 Map conversion

`vis5d` has a built-in function to display the map of the world. This map consists only of coast lines, not the borders between countries. The data are extracted from the default map file `OUTLSUPW`, which is supplied with VIS-5D.

This map file has the same grid problems as the VIS-5D topography file, as described in section 5.4: the data grid points are in a latitude/longitude format while the HIRLAM grid points are in a rotated latitude/longitude format. Because the climate file does not contain a map of the HIRLAM area, the default map file had to be converted to the correct grid.

In the standard VIS-5D package the map module handles reading and drawing of the map. This module is the starting point to construct a function that makes a map of the HIRLAM area in the correct grid format.

The conversion of grids is possible with the ASIMOF library routine

```
gb2gcc(pxold, pyold, pxnew, pynew, &k, kb2i, pb2i, kb2o, pb2o)
```

where `pxold` and `pyold` denote the grid values in the longitude and latitude direction respectively, `pxnew` and `pynew` the converted longitude and latitude grid values, `k` is the number of longitude and latitude values, `kb2i` and `pb2i` are the values of the integer- and real-block of the GRIB section 2 of the input grid and `kb2o` and `pb2o` are the section 2 integer and real blocks of the output grid.

The HIRLAM map can be made by typing at the commandline:

```
hirlammap
```

or

```
hirlammap <input map file> <output map file>
```

Running `hirlammap` without commandline arguments assumes `OUTLSUPW` as the input map file and `HIRLAM.MAP` as the output map file.

is visualized on pressure levels logarithmic interpolation must be used instead. The change necessary to `vis5d` uses the same logarithmic fraction as derived for the horizontal slice extraction (see section 5.2.1).

### 5.3 Scaling vertical wind component

When a vertical coordinate system different from metres is used, the scaling of the vertical wind component  $W$  must be adjusted. Currently VIS-5D scales  $W$  from the unit m/s to gridboxes/s. The unit of the vertical wind component (if calculated by HIRLAM) is Pa/s instead of m/s which VIS-5D assumes. The required scaling can easily be performed by dividing the values of the vertical wind component by the height of a grid box in Pascal.

These modifications must be performed in the `trajectory` and `work` modules, for the trajectory calculation and horizontal and vertical wind slice calculation, respectively.

### 5.4 Topography conversion

VIS-5D has a built-in function to display the topography of the world. The VIS-5D topography consists of two items for every grid point:

1. Surface elevation (height above sealevel),
2. Land/sea mask.

The word topography assumes that there is more information than just the height and the land/sea mask available, for instance the flow of rivers, roads and the use of land. The correct word should be orography but since VIS-5D refers to topography when meaning the orography this will be used in the remainder of this section.

The topography file format is described in appendix A.6. The grid of the default topography file `EARTH.TOPO` is in a standard latitude/longitude format whereas the HIRLAM grid is a rotated latitude/longitude grid. With VIS-5D a sample file `maketopo.c` is supplied that enables the user to make his own topography data file. This sample file has been adjusted to make a correct HIRLAM topography. The topography file can be made with the program `hirlamtopo` by simply typing:

```
hirlamtopo <input-climatefile> <outputfile>
```

The information necessary to create the topography (in the correct grid format) can be extracted from the HIRLAM climate file. The height above the ground is calculated by extracting the geopotential (variable 6, leveltype 105, height 0) and dividing by  $G_0 = 9.80665m/s^2$ ,

To introduce logarithmic interpolation into the iso surface calculation, the following equation must be modified:

$$f = \frac{isovalue - nodeval_1}{nodeval_2 - nodeval_1}, \quad (5.3)$$

where *isovalue* is the value for which the iso surface is calculated, *nodeval*<sub>1</sub> and *nodeval*<sub>2</sub> are the data values at the edge nodes between which is linearly interpolated and *f* is the fraction, i.e. the position of *isovalue* within the interval [*nodeval*<sub>1</sub>, *nodeval*<sub>2</sub>].

In order to use logarithmic interpolation a three-step calculation must be performed. First the linear fraction *f* is calculated using formula 5.3. Then the pressure value corresponding with *isovalue* in the given interval can be found by using the following formula for *f*:

$$f = \frac{\ln(P_1) - \ln(P_{iso})}{\ln(P_1) - \ln(P_2)}, \quad (5.4)$$

where *P*<sub>1</sub> and *P*<sub>2</sub> are the pressure levels for the edge nodes. *P*<sub>iso</sub> must be calculated, because this value determines the location of *isovalue* in the interval [*nodeval*<sub>1</sub>, *nodeval*<sub>2</sub>] when interpolating using the logarithm of the pressure levels. Solving equation 5.4 using equation 5.3 yields:

$$P_{iso} = e^{(\ln(P_1) - f \cdot (\ln(P_1) - \ln(P_2)))} \quad (5.5)$$

Third and last, the values of *P*<sub>iso</sub>, *P*<sub>1</sub> and *P*<sub>2</sub> are used to calculate the following fraction:

$$f_{iso} = \frac{P_1 - P_{iso}}{P_1 - P_2} \quad (5.6)$$

Fraction *f*<sub>iso</sub> corresponds to the logarithmic intersection point between the two nodes.

The modifications necessary to `vis5d` use the above equations in a new macro `LOGcalcNode`. This macro is used only to calculate intersections of vertical cube edges.

## 5.2.4 Wind trajectories

To calculate a smooth wind trajectory, VIS-5D linearly interpolated in a 4-D box (three spatial and one time dimension). The interpolation in the vertical coordinate should be performed logarithmically if pressure levels are used. The change necessary to `vis5d` uses the same logarithmic fraction as used for the horizontal slice extraction (see section 5.2.1).

A second modification is necessary to the trajectory to scale the vertical wind component correctly, see section 5.3.

## 5.2.5 Data probe

The data probe is used to obtain single values from the data grids. If the data probe is positioned between two levels, `vis5d` linearly interpolated between the levels. When data



where

$P_{above}$  = the pressure of the upper level of the adjoining pair,

$P_{below}$  = the pressure of the lower level of the adjoining pair,

$P_{wanted}$  = the pressure of the level that must be computed.

The new data values are now calculated in the normal way using the logarithmic fraction. The pressure value  $P_{wanted}$  is calculated from the slice position and  $P_{above}$  and  $P_{below}$  by linear interpolation.

### 5.2.2 Vertical slices

A vertical slice is calculated by horizontal interpolation of the data values nearest to the slice for every level of data available. No interpolation between the levels is done, so a vertical slice has the same number of data points in the vertical direction as there are levels. There are several ways to introduce the logarithmic interpolation between the levels:

- Extract the slice as normal. For a contour slice, the contouring algorithm must be modified so that intersections of edges spanning different levels are calculated by logarithmic interpolation. For a color slice, the graphics library must interpolate the colors between the vertices logarithmically. For the wind vector slices no change is necessary, because only the data values extracted from the grid points are used.
- Extract extra levels in between the levels. Calculate these extra data values in between by the same type interpolation as described in section 5.2.1. This way the contouring algorithm doesn't have to be modified, provided the vertical resolution is high enough. The color slices can be rendered as normal using linear interpolation between the vertices.

The first method cannot be used, because the GL library does not support logarithmic color interpolation (the GL man pages refer to possible future implementation of this feature though, see [GLman]). Only the second option can be used, but has the disadvantage that more data points are extracted which increases the time needed to calculate and render the slices.

Implementation of this method is not yet finished.

### 5.2.3 Iso surfaces

The iso surfaces are calculated using the Marching Cubes algorithm [LoCl87]. The data grid is divided into cubes with eight data values at the corners of each cube. For every edge of a cube iso surface intersections are determined. The iso-surface intersects an edge of the cube if the values at the cube nodes lie on opposite sides of the iso value (the data value for which the iso-surface is calculated). The intersections of all twelve edges together determine the part of the iso surface that lies within the cube. A 3-D polygon is generated.

The intersections of vertical cube edges must be interpolated logarithmically, whilst the intersections of horizontal cube edges must be interpolated linearly, as normal.

- “Floating levels” (2-D fields at a certain altitude) are not allowed. All single level fields are assumed to be surface fields.
- The levels must (still) be equidistant.
- Each physical variable must have the same number of levels in each time step.

For a detailed description of the changes, see appendix D.2.

## 5.2 Logarithmic interpolation

The coordinate system chosen for visualization uses the pressure as the vertical coordinate unit (see section 4.3 for a description of possible coordinate systems). Because in nature the pressure decreases exponentially as a function of the altitude, vertical interpolation of the physical variables (between the data levels) must be performed using the logarithm of the pressure level values. Currently `vis5d` can only interpolate linearly. Because the correct interpolation method to use depends on the coordinate system, an extra entry in the `vis5d` COMP and GRID file format is necessary: an integer `leveltype`. (The GRID and COMP file formats and the modifications are described in appendix A.4). The `leveltype` variable indicates what kind of vertical coordinate system is used. Currently these can be height and pressure values. In the following sections the changes to `vis5d` are described. Only the calculation of graphics and the data probe, i.e. the extraction of data from the grids has to be modified. In the following sections, the modifications are explained.

### 5.2.1 Horizontal slices

When the user selects a horizontal slice for a level that is not in the data grid but between existing levels, `vis5d` calculates the values on the requested slice by interpolation of the data values of the two adjoining levels. The current interpolation function of `vis5d` calculates linear fractions  $f$  and  $(1.0 - f)$ , which are used to calculate the data values on the requested slice:

$$data_{value} = f \cdot data_{below} + (1 - f) \cdot data_{above} \quad (5.1)$$

where

$data_{below}$  = data value on lower adjacent slice,

$data_{above}$  = data value on upper adjacent slice.

This formula is evaluated for every grid point on the horizontal slice.

The most easy way to perform the logarithmic interpolation is to add an alternative way to compute the fractions. The logarithmic interpolation can be expressed using the following formula for the logarithmic fraction:

$$f_{log} = \frac{\ln(P_{below}) - \ln(P_{wanted})}{\ln(P_{below}) - \ln(P_{above})} \quad (5.2)$$

- No redundant data.

**Disadvantages:**

- Only a color mapping can be visualized, no contour or wind vector slices.
- Modification of VIS-5D is necessary.
- The 2-D fields must be stored in separate files.
- Recoloring the topography with a physical variable can be confusing to the user, who loses his perception of the actual height of the topography.

5. Modify VIS-5D to eliminate the restriction of the trivial (height) dimension.

**Advantages:**

- No redundant data.
- All visualization techniques (as far as useful) are available for each physical variable. Visualization techniques that are not applicable to 2-D fields can be turned off in the control window.
- User-friendly. Everything concerning grid dimensions can be hidden from the user, but the user can never forget the number of levels, due to the restricted slice movement etc. (`vis5d` can simply prohibit operations that are illegal for 2-D fields).
- All combinations of physical variables can be visualized simultaneously.
- Each field has an unique button row in the control window and can thus be easily identified.

**Disadvantages:**

- Requires the most work of all available solutions.
- The complexity of VIS-5D increases.

After examination of the source code of VIS-5D it was decided to choose solution 5, because:

- It is the most elegant and user-friendly method.
- No redundant data needs to be stored. This is quite important because `vis5d` already needs a lot of memory, it is wise to use it sparingly.
- Various changes to VIS-5D were already necessary for other reasons, see section 5.2.

The amount of work was limited in the following ways:

- Only the trivial dimension height (the number of levels) is eliminated. In other words, all data grids must still have the same number of rows and columns. The number of rows and columns must each be at least 2.

- Modification of VIS-5D is not necessary.

**Disadvantages:**

- The user has to keep track of where the 2-D fields are positioned in the 3-D grid. A horizontal slice must be positioned exactly on the real 2-D field, otherwise the 2-D field is interpolated with a neighbouring 2-D field which is incorrect. This cannot be solved by inserting dummy levels with missing data indicators between each pair of levels, because the slice interpolation will then return missing data values for a slice adjacent to the missing value slice.
  - Vertical slices and iso-surfaces are incorrect, but can still be visualized.
  - For each time step dummy values must be used to fill the 3-D grid in which the 2-D fields are stacked if there are not enough 2-D fields to fill the 3-D field completely.
2. A separate physical variable is used for every 2-D field and all levels in a 3-D grid are set to the same values as in the 2-D field.

**Advantages:**

- All physical variables can be viewed simultaneously.
- Modification of VIS-5D is not necessary.
- The slice position is not relevant (all positions will result in the correct values being obtained).

**Disadvantages:**

- A lot of redundant data.
  - Vertical slices and iso-surfaces are still meaningless and incorrect.
3. Split the physical variables to visualize into two separate COMP files, one with the 3-D fields and another one with the 2-D fields. The 2-D fields are duplicated so that they are two levels high, eliminating the trivial height dimension of one. Visualizing of the 3-D fields constructed this way can be done as normal.

**Advantages:**

- “Not many” redundant data (all 2-D fields are “only” duplicated).
- Vertical slices and iso-surfaces are incorrect, but less confusing because of the reduced volume of data.
- Modification of VIS-5D is not necessary.

**Disadvantages:**

- Combinations of 3-D and 2-D fields cannot be visualized.
4. Color the topography using one of the required 2-D fields. The 2-D fields are stored in a separate file.

**Advantages:**

## Chapter 5

# VIS-5D modifications

In this chapter the modifications to VIS-5D are discussed which were necessary to make VIS-5D able to visualize the HIRLAM output. The following modifications are discussed:

- 2-D field visualizations (VIS-5D could only cope with 3-D fields),
- logarithmic interpolation of the vertical coordinate when pressure levels are used,
- creation of topography and map files of the (current) domain of HIRLAM,
- creation of colormap legends.

The 1-D probe modifications have not yet been studied.

### 5.1 Visualization of 2-D fields

In order for VIS-5D to be useful for visualizing the HIRLAM data, a solution must be found to visualize 2-D fields. VIS-5D has been written specifically for the visualization of 3-D data fields. Fields with only one row, column or level cannot be visualized. Another restriction is that every physical variable must have the same number of points in all four dimensions.

Possible solutions to visualize 2-D fields with VIS-5D are:

1. Stack all 2-D fields which must be visualized in 3-D data fields and fill the remaining levels with (arbitrary) dummy values.

#### **Advantages:**

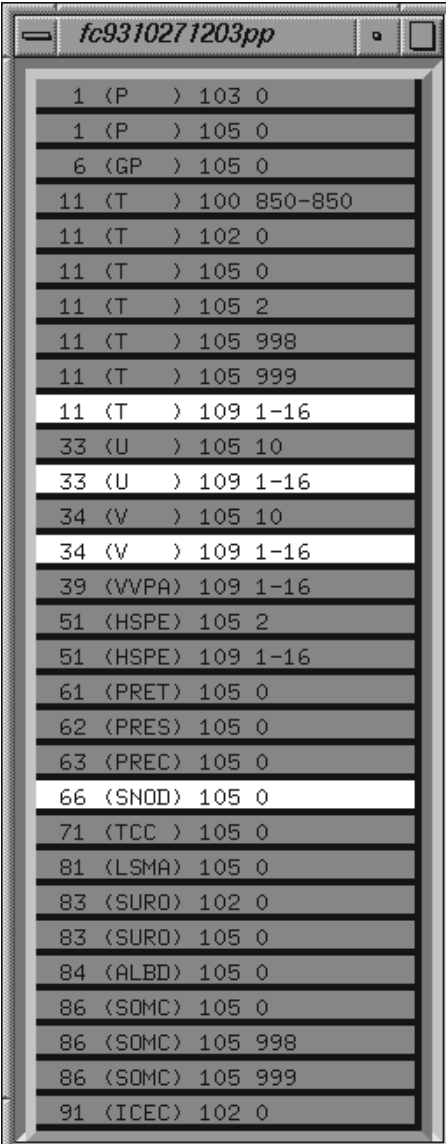
- All physical variables can be viewed simultaneously (remember that multiple visualizations of one physical variable are possible by cloning the variable using the NEWVAR option).



## 4.6 GRID to COMP conversion

In order to visualize the GRID files created by `xg2v`, they must be converted to COMP files which can be read by `vis5d`. This can be done with the utility `comp5d` which is supplied with VIS-5D. Due to the changes to VIS-5D which will be described in chapter 5 the standard `comp5d` program can not be used anymore, because it terminates with an error if the data grids in the input GRID file(s) do not have equal dimensions. To import the new GRID files, a new program `varNLcomp5d` has been derived from the existing `comp5d` program with the following modifications:

- When reading the first time step, the number of levels and height of each grid in the time step is stored in an array and kept for later use.
- When reading the remaining time steps, the number of levels and height of each grid is compared with the appropriate value from the arrays.
- The compression is modified to handle variable sized grids.



The image shows a window titled "fc9310271203pp" containing a list of data fields. Each field is represented by a row with a number in parentheses, a code in parentheses, and a value. The rows are as follows:

1	(P)	103	0
1	(P)	105	0
6	(GP)	105	0
11	(T)	100	850-850
11	(T)	102	0
11	(T)	105	0
11	(T)	105	2
11	(T)	105	998
11	(T)	105	999
11	(T)	109	1-16
33	(U)	105	10
33	(U)	109	1-16
34	(V)	105	10
34	(V)	109	1-16
39	(VVPA)	109	1-16
51	(HSPE)	105	2
51	(HSPE)	109	1-16
61	(PRET)	105	0
62	(PRES)	105	0
63	(PREC)	105	0
66	(SNOD)	105	0
71	(TCC)	105	0
81	(LSMA)	105	0
83	(SURO)	102	0
83	(SURO)	105	0
84	(ALBD)	105	0
86	(SOMC)	105	0
86	(SOMC)	105	998
86	(SOMC)	105	999
91	(ICED)	102	0

Figure 4.3: window with list of data fields



Starting `xg2v` will open the main data file window (see figure 4.1). In the top of this window the following buttons are available:

**Options:** Opens or closes the options window (see figure 4.2). In this window the default conversion options can be modified.

**Create:** Create the output file as selected by the user.

**Exit:** Exit `xg2v`.

Under this row of buttons, the window displays a list of files found in the specified search directories and matching the specified filename pattern.

The filenames are sorted in chronological order, which results in the time steps being written to the output file in the correct order. Each data file has a separate button, which are all turned off by default. The left mouse button can be used to select and deselect data files. A highlighted button will mean that the corresponding data file is included in the output file. The middle mouse button is used to open/close a new window displaying the contents of the corresponding data file as a list of data fields (see figure 4.3). The names of the data fields consist of the following information:

- GRIB variable number (according to WMO specifications),
- A short name of the variable,
- Leveltype number (according to WMO specifications),
- Levelrange.

Fields of equal variable number and leveltype are grouped under one button. Using the left mouse button, data fields can be selected and deselected. The data fields are read from the file, the first time the user clicks on a data file button in the main window.

`xg2v` will make sure that the selection of files and fields remains consistent by:

- Prohibiting the selection of new data files that do not contain all the selected data fields,
- Prohibiting the selection of new data fields that are not present in every selected data file.

If such a selection is attempted by the user, `xg2v` will report this in the standard output window.

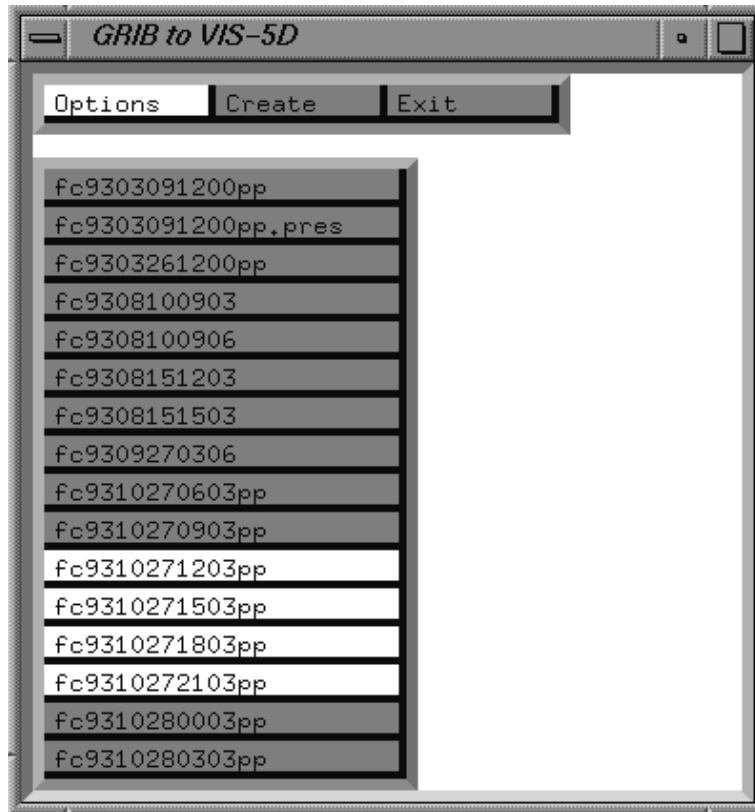


Figure 4.1: main window of xg2v

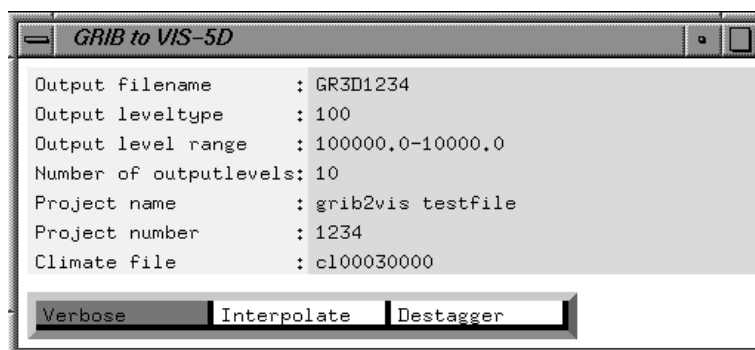


Figure 4.2: options window of xg2v

More information about the RUL libraries can be found in appendix C.2.

The RUL library functions were implemented in C (except for the functions in `futil.f`), for the following reasons:

- C provides dynamic memory allocation.
- C provides better string handling than FORTRAN77.

The routines in `futil.f` are provided because a FORTRAN `common` block cannot be referenced from C.

In order to call FORTRAN functions from a C program, the following should be noted:

- FORTRAN passes the address of every function parameter. Therefore when calling a FORTRAN function from C, of every parameter that is not a pointer the address must be passed.
- FORTRAN has no string termination character. When a character string is passed to a FORTRAN function expecting a `CHARACTER*(*)`, the string length must be passed as an extra parameter, after the list of remaining formal parameters.
- Arrays in C are row major opposed to arrays in FORTRAN being column major.
- The index of the first element in a C array is 0, in FORTRAN this is 1.

### 4.5.3 Conversion program

The program `xg2v` is responsible for the conversion of GRIB fields stored in ASIMOF files to McIDAS GRID files. `xg2v` must be started with the command:

```
xg2v [options]
```

The following options are available:

<i>Option</i>	Description	Default
<code>-file</code>	Specify (wild-carded) input filename pattern	"fc*"
<code>-path</code>	Specify comma-separated list of directories	.
<code>-out</code>	Override the default output filename	"GR3D1234"
<code>-climatefile</code>	Override the default climate file	"cl00030000"
<code>-projectname</code>	Override the default project name	"grib2vis testfile"
<code>-projectnumber</code>	Override the default project number	1234
<code>-outputleveltype</code>	Override the default output level type	100
<code>-outputlevelrange</code>	Override the default output level start and end, separated by a minus (-) sign	100000.0-10000.0
<code>-numoutputlevels</code>	Override the default number of output levels	10
<code>-v</code>	Increases the amount of information printed to the screen during conversion.	

- util:** Provides routines for date conversion and ASIMOF information programs,
- vari:** Provides conversions from one coordinate system into another, as well as routines to convert U and V to several coordinate systems,
- prpo:** Provides routines used for postprocessing (vertical interpolation and destaggering of the wind components) of the data,
- grw1:** Provides functions that are used as an interface to ASIMOF files. The DDR (data description record) is the same as the keys of a GRIB message (block1 and block2). The parameters are kept in a common FORTRAN block.

More information about the KNMI libraries can be found in appendix C.1.

Using the KNMI libraries not only reduces the work necessary, but also ensures future compatibility if the KNMI libraries are changed or extended.

#### 4.5.2 RUL libraries

The RUL libraries were created to provide an interface between the KNMI libraries and the main GRIB to VIS-5D conversion program `xg2v` (X-Windows GRIB to VIS-5D) for the following reasons:

- The KNMI libraries use very cryptic function and variable names,
- The function interfaces of the KNMI libraries are sometimes very complex (88 function parameters is no exception). The RUL libraries hide unnecessary detail but still retain enough flexibility to provide the same functionality as the KNMI libraries. The parameters that need not be altered are placed in a C `struct` which is passed to the functions in the RUL libraries.

The following RUL modules are implemented:

- asimof:** Handles the opening and closing of ASIMOF files and the extraction of data from these files.
- grib:** Provides functions to convert between GRIB physical variable names and numbers. Also a unit and short name is defined for every physical parameter, because the GRID file format reserves only 4 characters for the names of the physical variables.
- interpof:** Provides functions to interpolate GRIB data from  $\eta$ -levels to pressure levels.
- cutil:** Provides general purpose utility C-functions.
- futil:** Provides general purpose utility FORTRAN-functions.
- xg2v:** Main program module providing a X-Windows based user interface for the conversion.

level upwards. This means that when a physical variable is extracted with more than one level from an ASIMOF file, the levels must be sorted before writing them to an output GRID file. Because each GRIB message contains one level, it is most easy to extract the levels in the correct order. This has the additional advantage that it is easy to extract fields in *reverse order* which is necessary for vertical coordinates such as pressure, where the positive coordinate direction is downwards. The HIRLAM model levels are also defined downwards (level 1 is the top level).

For each field read from an ASIMOF file, two additional reordering operations must take place. First the array has to be converted from column major to row major, because the field is read by a FORTRAN function and is further processed by a C function. The second operation is a mirror operation around the horizontal axis, because the origin of the HIRLAM domain is the south-west-bottom grid point while VIS-5D expects the origin to be the north-west-bottom grid point.

Not all physical variables within HIRLAM are calculated on the same grid points. Some variables are translated a half grid point distance with respect to the grid for which the surface geopotential is defined (this grid is used as a reference grid for the remaining fields because the topo is derived from it and used as reference by the user). Because VIS-5D can only cope with variables defined on equal grid points, the physical variables will be linear interpolated to the “reference” gridpoints, read from the surface geopotential field. This process is called *destaggering*.

After the data fields are extracted and converted, they are written to a GRID data file. The grid info structure (which contents is completely known after the conversions) is written also.

Since single level fields cannot be vertically interpolated, they will be written to the GRID file without being interpolated (but reordering, mirroring and destaggering are performed on *all* fields).

Variables that are already interpolated to pressure levels will also be directly written to the GRIB file.

## 4.5 Conversion from GRIB to GRID

### 4.5.1 KNMI libraries

The KNMI has several libraries available that provide functions to manipulate the ASIMOF files, to perform various coordinate transformations, to transform time and date formats, etc. The following libraries are available:

**port:** Provides routines used for the ASIMOF file handling,

**gcod:** Provides GRIB related routines, such as encoding and decoding, retrieving sections from GRIB messages, printing these sections and checking the parameters of the sections etc.,

- The pressure decreases exponentially with the height and therefore cannot be interpolated linearly in height.
- The vertical velocity must be converted from Pa/s to m/s in order to keep the trajectory module functioning correctly.
- Extrapolating variables below the orography takes special care.

Considering the advantages and disadvantages of these three coordinate systems, the chosen vertical coordinate representation in VIS-5D is the pressure level coordinate system because it is the representation that corresponds best with the meteorological practice. As a result this means that levetypes other than pressure levels will have to be interpolated to pressure levels.

#### 4.4 ASIMOF to GRID conversion specification

In order to be able to interpolate variables to pressure levels, the following physical variables *must* be present in *every* ASIMOF file:

- Pressure at ground level,
- Temperature at ground level,
- Temperature at model full levels,
- $U$  component of the wind at model full levels,
- $V$  component of the wind at model full levels,
- Specific humidity at model full levels.

In addition, the geopotential at ground level (variable number 6, level type 105, height 0) must be read from a *climate file*, which is an ASIMOF file and contains the average of several variables over a long period (usually a year) of time.

If one of these variables is not available, it is impossible to interpolate to pressure levels and the conversion program will stop with an error message.

Furthermore the full model vertical level parameters A and B are needed to interpolate correctly. A and B represent the contribution of the orography and pressure to the  $\eta$ -coordinate, respectively. Between the full model levels of HIRLAM half model levels are defined. There is one half level more than there are full levels. The lowest and highest half levels correspond with the bottom and top of the domain of HIRLAM, respectively. The half model level parameters can be calculated from the full model level parameters. From the model half and full level parameters, the pressure at  $\eta$  half and full model levels can be derived.

The ASIMOF file format imposes no ordering on the GRIB fields contained within the file. However, VIS-5D expects the grids in the GRID and COMP files to be given from the bottom

- The largest change in pressure per height are in the lowest levels of the model. This results in the highest resolution in the area of most interest.
- Pressure levels correspond best with meteorological practice.

**Disadvantages:**

- The orography (i.e. the surface pressure) is not constant in time.
  - The vertical interpolation of variables must be done in the logarithm of the pressure because of the exponential decrease of the pressure with height.
  - The positive coordinate direction is downwards.
  - Special care must be taken to extrapolate variables below the orography.
2. *Model level numbers* (The HIRLAM model levels are used directly as an equidistant coordinate system).

**Advantages:**

- The most simple of all coordinate systems.
- The levels are equidistant so no initial interpolation is needed.
- No orography is needed (flat orography) because the bottom model level follows this orography.
- Best representation for a model behaviour study.

**Disadvantages:**

- The vertical velocity is defined as the change in pressure (Pa/s). This has to be transformed to a velocity expressed in model levels per second to be able to use the trajectory module.
  - The positive coordinate direction is downwards.
  - Vertical interpolation of the physical variables cannot be done linear, but must be done in the logarithm of the pressure.
  - The effects of the orography are being masked, because the used orography is flat.
3. *Geopotential height* (levels with the same geopotential i.e. the same potential energy of an air particle).

**Advantages:**

- Easy to calculate using the HIRLAM post processing package (`postpp`).
- The orography is constant and defined as the geopotential height of the surface.
- All variables (except the pressure) can be interpolated in height.
- The positive coordinate direction is upwards.

**Disadvantages:**

- The levels are not equidistant. The highest resolution is given and needed at the surface of the earth. Interpolation to equidistant levels, will result in loss of essential information in the most interesting part of the atmosphere.

- Only a single conversion step, resulting in quicker conversion and less disc space usage.
2. Two-step conversion: First convert GRIB data fields to a GRID file, then convert the GRID file to a COMP file with the already existing VIS-5D utilities. This has the following advantages:
- The VIS-5D package includes several utilities that allows the user to perform various operations on GRID files, such as interpolation between different time steps, resampling of the grid data, etc. (see section 2.4.1 for an overview of available utilities).
  - The `vis5d` program includes preprocessor statements (`RD_ON_DEMAND`) which enables the read-on-demand file access feature. `vis5d` will read the COMP file header information as usual but when a data grid is requested, the data is read from the GRID file with the original uncompressed data if possible. The advantage of this feature is that the data can be visualized without the loss of accuracy that happens when converting GRID to COMP (data values are compressed from `float` to `one-byte integer`, see also appendix A.4).

Considering the two possibilities, the choice was made to convert the HIRLAM output to the GRID file format, because in that way the functionality of the VIS-5D utilities and direct GRID file access remains available.

### 4.3 Vertical coordinate

The vertical coordinate used in the HIRLAM system is the so called  $\eta$ -coordinate. The  $\eta$ -coordinate system (also referred to as hybrid coordinate system) combines the orography (surface elevation) with pressure levels. Close to the ground, the levels follow the orography. Going upward the relative contribution of the orography decreases while the relative contribution of the pressure increases. The relative amounts of each contribution are given for each level by the HIRLAM model vertical parameters. Furthermore, the levels of the  $\eta$ -coordinate systems are not equidistant. The largest resolution is close the ground, which is the part of the atmosphere of most interest. In order to visualize the HIRLAM data with VIS-5D a coordinate system must be used with equidistant levels, because VIS-5D can not cope with non-equidistant levels.

In general, possible vertical coordinate systems for visualization are:

1. *Pressure levels* (levels given at specific pressure values).

#### Advantages:

- Pressure levels can easily be calculated from the surface pressure and the vertical model parameters, which are both directly available from the GRIB messages.
- The vertical velocity is correctly defined in Pa/s.



## Chapter 4

# Interfacing HIRLAM with VIS-5D

### 4.1 HIRLAM output

The HIRLAM output data files are available in the ASIMOF file format (see appendix A.2). ASIMOF files consist of one or more GRIB messages (see appendix A.1), with a directory structure giving information about the location and characteristics of the GRIB messages. The GRIB messages in the file contain the data. Each GRIB message can contain information and data of one level field of a physical variable. Information such as time and date stamp of the field, level type and value are included.

The ASIMOF data (or history) files have a name like **fc9303261200pp**, where:

<b>fc</b>	file type (fc: forecast, an: analysis, in: initialised analysis),
<b>93032612</b>	start of forecast (year, month, day, hour),
<b>00</b>	length of forecast,
<b>pp</b>	status identifier (pp: postprocessed, not postprocessed).

In the following sections the conversion from the ASIMOF file format to the COMP file format will be discussed.

### 4.2 Conversion to GRID versus COMP

To view the HIRLAM data using VIS-5D a conversion from the ASIMOF file format (the GRIB database file format) to the VIS-5D COMP file format is necessary. Two choices can be made for converting GRIB data to the COMP file format.

1. One-step conversion: Convert GRIB data immediately to the COMP file format. This has the following advantages:



Fortran function call a version with and one without an underscore attached to the function name is available.

- Due to differences in the X11 versions on the SGI/IBM/Stellar and the HP, it is necessary to set the (new) position and size of a window when it is opened, moved or resized (using X-Windows SizeHints).

The rest of the port is just a matter of following the instructions in the porting guide supplied with VIS-5D. See appendix D.1 for a description of HP port modifications.

### 3.1.5 Scan conversion

Because every draw operation must be performed on a per-pixel basis for correct functioning of the Z-buffer, lines and polygons must be scan converted by VOGL and for every pixel the Z-buffer must be checked. Two types of scan conversion are necessary: Line scan conversion and polygon scan conversion. Line scan conversion is implemented using the Bresenham line drawing algorithm (see [Hill90]). Two versions are supplied. The first is just a pure line drawing algorithm. The second takes extra color parameters for the start- and endpoint of the line. The line color is interpolated linearly between these values. Polygon scan conversion is more difficult. The algorithm used is a variant of the polygon scan conversion algorithm as described in [Hill90]. For each scanline the intersections with the polygon edges are found and sorted in increasing  $x$  values. Then for each pair of sorted  $x$  values a run of pixels is filled. The color of a run is determined by the color of the start- and endpixels of the run. Colors along the run are linearly interpolated between these values. Start- and endcolors of each run are found by linearly interpolated the vertex colors over the polygon edges.

Due to the manner of geometry specification of the GL (points and colors are specified by a sequence of function calls) a change to VOGL is necessary so that vertices are shaded as soon as they are specified, because the current color and normal (used for shading) are changed by the next color or normal specifying function call.

### 3.1.6 VOGL device drivers

The device dependent parts of VOGL are separated from the rest of the code into *device drivers*. A device driver gives VOGL access to certain device specific features such as drawing, keyboard and mouse read operations, device initialization and clean up. The use of device drivers makes VOGL *very* portable, because only the device driver needs to be altered when porting VOGL to a new system. The structure of a VOGL device driver is given in appendix E.2.

## 3.2 Miscellaneous problems

The following miscellaneous problems were solved when porting VIS-5D to HP:

- The C preprocessor used at Leiden University (release 8.05) does not understand `#elif`, which has been changed to `#else ... #if` statements.
- The HP FORTRAN compiler does not understand the type-length combination `INTEGER*1`, the type `BYTE` must be used instead.
- For compilers on some systems, it is necessary to attach an underscore to function names of Fortran functions that are called from C. This is not the case for HP 700 Series systems. A new preprocessor symbol has been used (`underscored`) and of each

**Specular:**

$$I_{sp} = I_s r_s (u_r \bullet u_v)^f,$$

- where
- $I_{sp}$  = intensity of specular reflected light,
  - $I_s$  = intensity of light source,
  - $r_s$  = material specular reflectance coefficient,
  - $u_r$  = normal vector pointing in the direction of reflected light (**R**),
  - $u_v$  = normal vector pointing in the direction of the vertex to viewpoint vector.
  - $f$  = shininess coefficient.

Because the cosine of the angle between two vectors is proportional to the dot product between their normalized versions, the latter is used to avoid cosine calculations. This has the consequence that all vectors used in the calculations must be normalized. Normalization of normal vectors is necessary if the supplied normal vectors do not have unit length or if the current ModelView matrix (which is used to transform the normal vectors to eye coordinates before doing lighting calculations) is not orthonormal. The GL can be notified by the user when to normalize normal vectors.

Two shading models have been added to VOGL: FLAT and GOURAUD shading [Gour71]. When FLAT shading is used the shade of the last vertex of a geometry determines the shade of the whole geometry. When GOURAUD shading is used, each vertex of a geometry is shaded and colors are interpolated linearly between the vertices.

All lighting functions have been implemented in a new module called light. The functions of the light module are listed in appendix E.1.3.

**3.1.4 Depth-cueing**

Depth-cueing has been added to VOGL, because while working with `vis5d` without depth-cued lines in combination with orthogonal projections, it became apparent that it is very hard to determine the orientation of the box. After implementation of the Z-buffer, the only modifications necessary were to implement functions to set the depthcueing color and Z range, and to turn depthcueing on or off. These are described in appendix E.1.4

To simplify the implementation of the depthcueing, there are currently two restrictions to the depthcueing. First, only lines can be depth-cued. Second, lighting and depthcueing are mutually exclusive, that is when depth-cueing is on, lighting commands are ignored (In fact, they are not ignored, but the colors calculated by the lightmodel are overwritten by the depthcueing color).

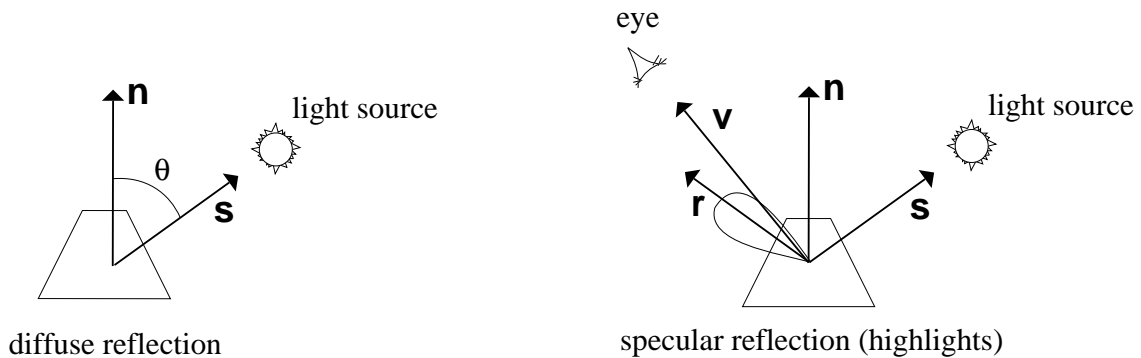


Figure 3.2: Vectors in the lighting calculations

- Light source color, scaled by the material diffuse reflection and a function of the angle between the vertex (surface) normal and the vertex-to-light source vector.
- Light source color, scaled by material specular reflectance and a function of the angle between the vertex-to-viewpoint vector and the reflected light vector.

For each light source two types of reflection due to the light source incident light are usually computed:

- Diffuse reflection (scattering). This occurs when some of the incident light slightly penetrated the surface and is reemitted uniformly in all directions.
- Specular reflection. These kind of reflections are more mirrorlike and highly directional. These are the reflections that make the surface look shiny.

The various vectors used for lighting calculations are depicted in figure 3.2.  $n$  is the surface normal vector,  $s$  is the vertex-to-light source vector,  $v$  is the vertex-to-viewpoint vector, and  $r$  is the direction of reflected (specular) light.

Diffuse and specular reflection can be expressed using the following formulas:

#### Diffuse:

$$I_d = I_s r_d (u_s \bullet u_n),$$

- where
- $I_d$  = intensity of diffuse reflected light,
  - $I_s$  = intensity of light source,
  - $r_d$  = material diffuse reflectance coefficient,
  - $u_s$  = normal vector pointing in the direction of  $s$ ,
  - $u_n$  = normal vector pointing in the direction of  $n$ .

A typical GL lighting session goes as follows:

- Define properties for each material, light and lighting model wanted,
- Activate (by binding to one of the eleven resources) the definitions,
- Draw the scene, providing the surface normals and vertices for all primitives to be lighted.

The reader is referred to [GLman] for more details. In the following sections the enhancements made to VOGL are discussed.

### 3.1.2 Z-buffering

The use of a Z-buffer has a great impact on the current structure of VOGL. The calculation has to be done on a per-pixel basis, because *every* pixel of a geometry can be behind or before the existing geometry. VOGL uses high-level drawing primitives to draw lines and filled polygons, which are implemented in the device drivers. Drawing of these primitives must now be done on a per-pixel basis so the device driver routines cannot be used anymore. Instead, a new device driver routine must be added to draw a single pixel (this routine did not exist, a single pixel was drawn by drawing a line with equal start- and endpoints).

The Z-coordinate used for Z-buffering is the *screen* Z-coordinate of the pixel. This value is obtained by a new function, which maps a vertex in clip coordinates to screen Z-coordinates using the current Z-buffer depth-range. A new type `typedef zbuftype` is added to VOGL. This type is used for each entry in the Z-buffer. Currently an 8-bit Z-buffer is used, but this can be changed easily by changing the `zbuftype typedef` and the define for `MAXZ` (the maximum possible value to store in the Z-buffer).

The reader is referred to appendix E.1.1 for a description of implemented functions.

### 3.1.3 Lighting

GL lighting calculates the color at each lighted vertex by summing the total ambient light (scaled by the material ambient reflectance), the material emitted light, and the contributions of each light source.

The total ambient light is the sum of the ambient light associated with each light source and the ambient light associated with the scene, as given by the lighting model.

The contribution of each light source is the sum of:

- Light source ambient color, scaled by the material ambient reflectance.

GL lighting performs the shading calculation on vertices in eye coordinates. This has the consequence that GL lighting calculations are only possible in *multi matrix mode* and not in *single matrix mode*.

GL lighting is controlled by defining various instances of three types of resources:

- materials (surface properties),
- light sources,
- lightmodel (environment properties).

These definitions can then be bound to one of eleven *light controlling resources* (a material, a backmaterial, up to eight light sources, and a lighting model). GL lighting is inactive if no material *or* no lighting model has been bound.

A material definition consists of the following properties:

- transparency,
- ambient reflectance,
- diffuse reflectance,
- material emitted light,
- specular reflectance,
- specular exponent (shininess).

A light source definition consists of the following properties:

- ambient light associated with the light source,
- light source color,
- light source position,
- spotlight direction (if the light source is a spotlight),
- exponent and spread of spotlight cone.

A lighting model definition consists of the following properties:

- ambient light associated with the entire scene,
- various attenuation factors to reduce direct light on objects and make the light intensity vary proportional to the object-light source distance.



### Z-buffering

A Z-buffer is a set of integers associated with pixels in the framebuffer. The integers represent the distance of the pixel to the viewpoint. When Z-buffering is activated, for each pixel  $(x, y)$  to draw the Z-buffer is checked to see if the new pixel is closer to the eye than the pixel already present at position  $(x, y)$ . If the new pixel is closer, the pixel is drawn and the Z-buffer is updated. VOGL does not support Z-buffering rendering.

### Depthcueing

When looking at objects in real life, the eye's ability to perceive depth gives awareness of the 3-D nature of objects and judgement of the relative distance of objects. The illusion of depth perception can be created on 2-D screens by *depth-cued* images. Depth-cueing modifies an object's color based on the distance to the viewer. The screen Z-coordinate is used to determine the depth of an object. This value is translated to a corresponding Z-buffer value. By specifying a color range and depth range, the user can determine what color the Z-values maps to. For a proper depthcueing effect, objects farther from the viewpoint should be colored darker. Depthcueing is not supported by VOGL.

### RGB and colormap mode

The GL supports two colormodes: RGB and colormap mode. In colormap mode a Color LookUp Table (LUT) is used. Colors are specified by entries in the LUT (single integer values). The LUT contains the red, green and blue color values. In RGB mode true color red, green and blue values are specified directly. Not all operations of the library are possible in colormap mode, the most powerful of the two is RGB mode. Other operations need special actions. For example, when drawing depthcued lines in colormap mode, the LUT must first be loaded with a color ramp before the line can be displayed correctly. VOGL only supports colormap mode, but VIS-5D needs RGB mode, because all colors in VIS-5D are represented as 24-bit color.

### GL lighting

In GL lighting, as in the real world, the characteristics of the light source determine the color and intensity of incident light on a surface. The characteristics of the object geometry and its surface material determine the direction, intensity and color of reflected light. How light is reflected depends on the interaction between the incident light and the surface material.

The interaction between light and matter is far more complicated than can be simulated in real-time. Lighting in the GL achieves a balance between realistic appearance and real-time drawing speed. The GL achieves this balance by performing lighting calculations *only at geometry vertices*, rather than calculating lighting for each pixel rendered.

The *eye coordinate system* is the result of *transforming* (through matrix multiplication) the geometry coordinates by the contents of the modeling and viewing (*ModelView*) matrix. The eye coordinate system is the system in which lighting calculations are performed internally.

Points in eye coordinates are transformed to *clip coordinates* by matrix multiplication with the *Projection* matrix. The next system is the *Normalized coordinate system* which is the result of limiting  $x, y$  and  $z$  to the range  $-w \leq x, y, z \leq w$  (clipping) and dividing  $x, y$  and  $z$  by  $w$ . The result are normalized coordinates in the range  $-1 \leq x, y, z \leq 1$ . The space of normalized coordinates is called the 3-D unit cube. The  $x$  and  $y$  coordinates of this 3-D unit cube are scaled directly into the *Window coordinate system*. The pixel at the lower-left corner has window coordinates  $(0, 0)$ .

Window coordinates, modified by a window offset that represents the window's location on the screen, represents the *Screen coordinate system*, which corresponds to pixel values. Screen coordinates are typically thought of as 2-D, but in fact all three dimensions of the normalized coordinates are scaled. The screen Z-coordinate can be used for hidden-surface removal and depth-cueing.

The coordinate systems are all present in VOGL, except for the *eye* coordinate system (this is skipped because the ModelView and Projection matrices are concatenated). The eye coordinate system is necessary for lighting calculations. Furthermore the incorrect term *World coordinates* is used instead of *Object coordinates*. In fact, if the ModelView matrix would be separated in a Model matrix and a View matrix, the coordinate system between these matrices would be correctly referred to as the *World coordinate system*.

## Transformation matrices

The graphics system of the GL maintains three transformation matrices: the ModelView, Projection and Texture matrices. The Modelview and Projection matrices are already discussed. The Texture matrix transforms texture coordinates directly from object to clip coordinates. Its transformation is unrelated to that specified by the ModelView and Projection matrices.

Operations on the matrices are performed depending on the current *matrix mode*. Four matrix modes are available. In *single matrix mode* only one transformation matrix is maintained which transforms points directly from object to clip coordinates. This is the only matrix mode available in VOGL, but cannot be used when lighting calculations are necessary, because these must be done in eye coordinates. The other three matrix modes are *viewing, projection and texture* matrix mode, which specify the matrix that is modified by operations such as `loadmatrix()`. Other functions will always affect the Projection matrix, disregarding the current matrix mode. Examples are `perspective()` and `ortho()` which are used to define the view volume that is mapped to the 3-D unit cube. Multi-matrix mode (not available in VOGL) is necessary to obtain objects in eye coordinates which are used for lighting calculations. A more detailed description is beyond the scope of this report, the reader is referred to [GLman] for details.

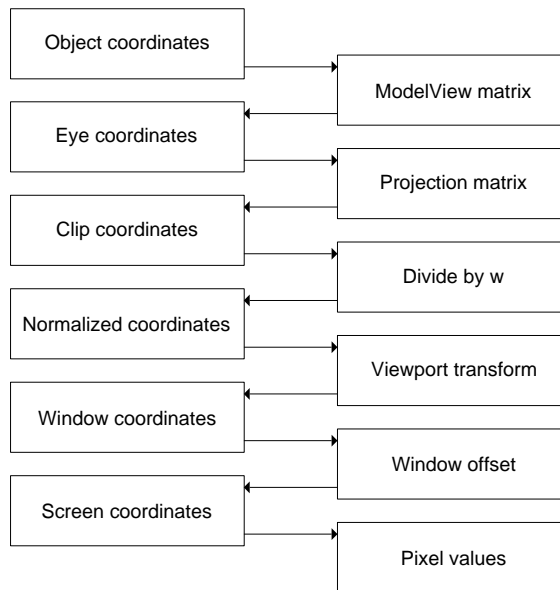


Figure 3.1: Coordinate systems used by the SGI GL

### 3.1.1 An introduction to the SGI GL

In this section parts of the existing SGI Graphics Library (GL) are discussed that are important in relation to the modifications and enhancements that are necessary to VOGL.

#### Geometry specification

Specification of the geometry to draw is done by issuing function calls, that each send a vertex position, vertex normal or vertex color to the GL. For instance, when a line is drawn, the GL must first be set to line drawing mode. After that separate function calls are issued to set the color, supply normals and to supply vertices, which are used to draw and color or shade the line. After the line is completely specified, the line drawing mode must be unset. Geometries drawn this way include lines, triangle meshes and quadrilateral meshes.

#### Coordinate systems

The coordinate systems used by the GL are important because each function acting upon geometry does so in one of the available coordinate systems. The coordinate systems used by the GL are displayed in figure 3.1. It starts with a 3-D system defined in right-handed cartesian floating point coordinates, called the *object coordinate system*. Geometry vertices specified by the user as  $(x, y, z)$  triplets are in this coordinate system. There are no limits to the size of coordinates in this system (other than the largest legal floating point value).

be written providing functions to draw lines, text, set color, etc. For a description of a VOGL device driver, the reader is referred to appendix E.2.

- VOGL resembles the SGI GL (the subset implemented), so modifications to VIS-5D are hardly necessary.
- VOGL 1.2.5 (February 26, 1993) has enough functionality to give an adequate display of contour slices and wind-trajectories.
- VOGL is free and can be modified and used for every purpose, as long as the original COPYRIGHT notice remains intact.

Disadvantages of VOGL are:

- Slow (compared with the hardware SGI GL). The speed of the hardware GL can of course never be matched by VOGL. This is made worse by the trade-off between portability and speed. Every geometry is drawn by device driver calls. To keep VOGL portable, device dependent optimizations should only be used in the device drivers and not in the main code section.
- Not all GL functionality used by VIS-5D are provided by VOGL. GL functionality used by VIS-5D but not available in VOGL include:
  - Z-buffering,
  - shading,
  - depthcueing,
  - transparency,
  - textures.

Z-buffering and shading are vital. The others are not, but can enhance the rendered images and increase the ease with which the images are interpreted by the viewer.

It was decided to use VOGL for porting VIS-5D for the following reasons:

- Portability, which is a great advantage because VIS-5D must be ported to HP *and* DEC. Using VOGL will probably also reduce the work necessary for future ports of VIS-5D.
- The resemblance to the SGI GL makes changes to VIS-5D hardly necessary, which reduces complexity of VIS-5D. Also, in cooperation with the KNMI, we had access to the manuals and documentation of the SGI GL.
- The subset of the GL already implemented reduces the amount of work to do.

The modifications necessary to `vis5d` for using VOGL are described in appendix D.4. In the following sections the enhancements made to VOGL are discussed, after a short introduction to the SGI GL which is necessary to understand why the modifications to VOGL were necessary.

## Chapter 3

# Porting VIS-5D

VIS-5D does not work on HP 700 Series and DEC hardware platforms (see section 2.2 for a list of system requirements). VIS-5D had to be ported to HP systems to be able to use VIS-5D at Leiden University. The KNMI is interested in a port of VIS-5D to a DEC Alpha station. The 3-D graphics library requirement is the most serious requirement for a port to HP and DEC. VIS-5D uses the XFDD library for Stellar and the SGI Graphics Library (GL) for SGI and IBM hardware platforms (see [GLman]). The XFDD library uses very high-level drawing primitives like `XFDDnormPolyTri(...)` which draws a poly-triangle strip (a polygonal mesh built of triangles) using normals for lighting calculations. The GL uses low-level primitives and draws a poly-triangle strip by separately sending the normal vectors and vertex positions to the GL. The function calls that supply the normals and vertices must be enclosed by function calls that notify the GL that a poly-triangle strip must be drawn. These two examples are fairly representative for the remaining part of the libraries. The 3-D library is further discussed in the next section.

### 3.1 3-D graphics library

In order to port VIS-5D to HP and DEC systems, a 3-D graphics library is necessary for both systems. An ideal solution would be a *portable* graphics library, which can be used for both target systems. A low-cost solution is available in the form of VOGL<sup>1</sup>, which stands for Very Ordinary GL like Graphics Library. VOGL is a software implementation of a *subset* of the (hardware) SGI Graphics Library (GL).

Advantages of VOGL are:

- VOGL is portable. The device dependent source code is separated from the main source code into device drivers. To use VOGL with a new device only a device driver has to

---

<sup>1</sup>VOGL was written by Eric H. Echidna at the University of Melbourne (echidna@munnari.OZ.AU). It may be freely used and modified as long as the original COPYRIGHT notice remains intact.

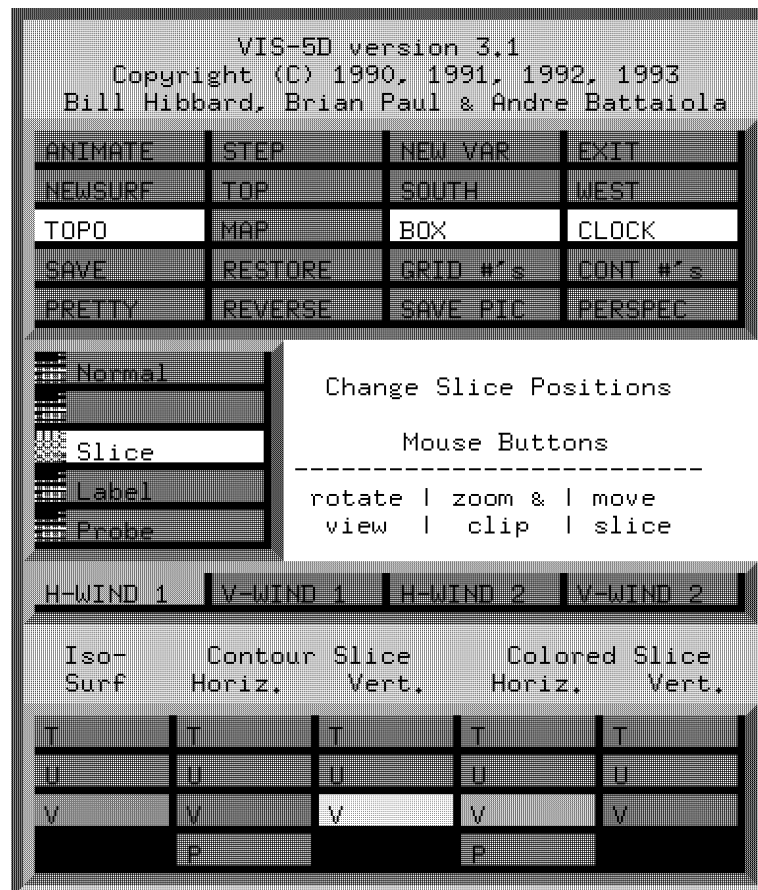
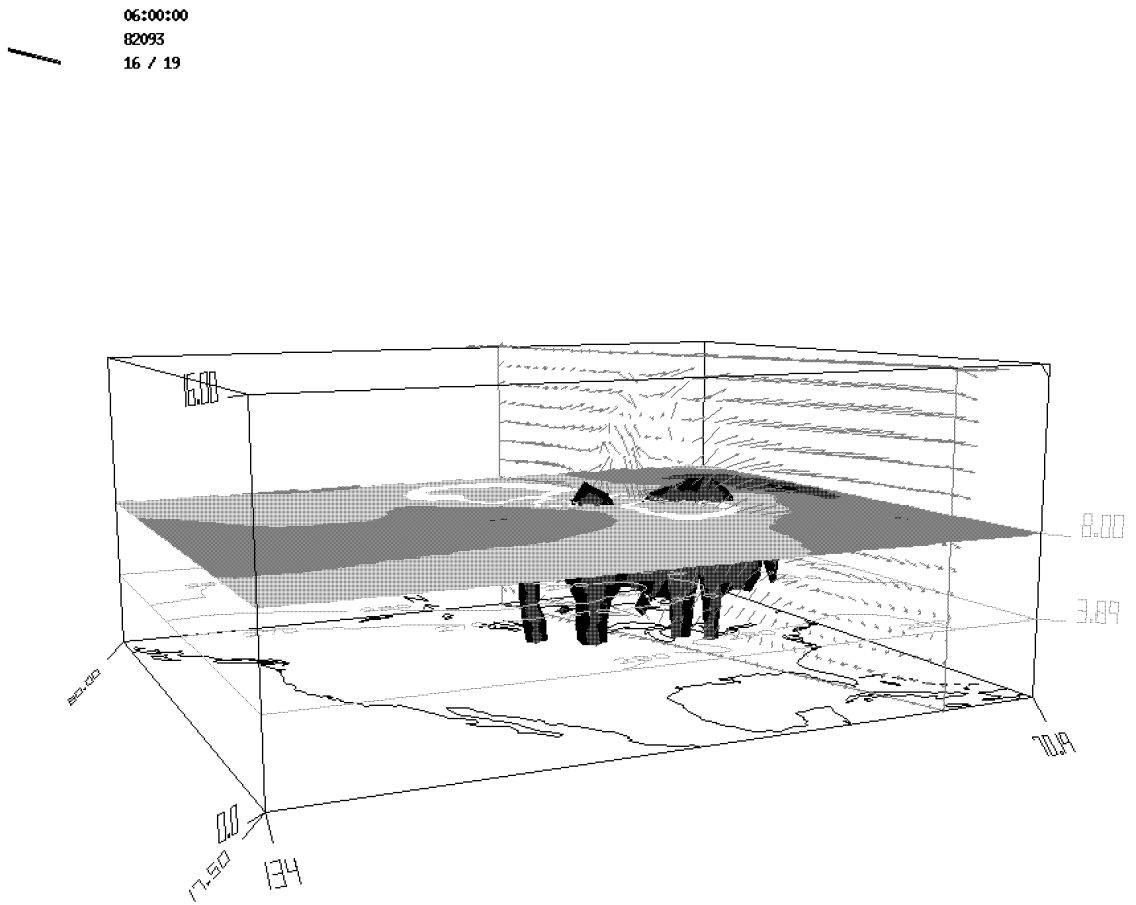


Figure 2.2: Control window



VIS-5D

Figure 2.1: Example contents of the VIS-5D 3-D window

- External user functions (directory `userfuncs`). One way of deriving new physical variables from existing ones is to have an external user function calculate the new variable. In the `userfuncs` directory the user can install Fortran modules which are all compiled and linked into separate programs. `vis5d` scans the `userfuncs` directory and the user can execute them from within `vis5d`. More information about external user functions is provided in appendix B.
- Utilities for managing and analyzing five-dimensional data grids and converting data into VIS-5D file format (directory `util`). In the VIS-5D package, several utilities (see section 2.4.1) are included for various operations on GRID and COMP files (see appendix A.3 and A.4). Skeleton example programs reside also in the utilities directory. These programs show how to import data into `vis5d`. The `util` directory also contains a library of utility functions (McIDAS library for reading McIDAS GRID files). The main code of `vis5d` makes use of the McIDAS library for the direct reading of GRID files.
- LUI (Little User Interface) library (directory `lui2`). The LUI library is based upon X-Windows and provides convenient functions for all sorts of manipulation and interaction needed in a window environment. VIS-5D uses this library for all graphical user interface functions.

The main program `vis5d` relies on the LUI library for all X-Windows based functions and the McIDAS library functions for reading GRID files. For a more detailed overview of the available source code, the reader is referred to appendix B.



isosurfaces, horizontal contour slices, vertical contour slices, horizontal colored slices, and vertical colored slices of the data. The buttons are arranged in a matrix. Each row corresponds to a physical variable in the data set. Each column corresponds to one type of graphic listed below.

To display the graphic, click the left mouse button on the widget button. A small control window will appear that is different for each type of graphic. The following graphic types are supported:

**3-D contour surface (isosurface):** The isosurface control window that appears is used to set the iso-level value for the contour surface. The desired value can be selected by dragging the slider to the correct place. When the correct value is selected, click on the “NEW SURF” button. The right mouse button can, besides to change the colors of the surface, be used to change the transparency. The transparency value can slide between 0.0 (invisible) and 1.0 (opaque).

**Contour slice:** After selecting a contour slice with the left mouse button, a contour slice control window containing a type-in widget will appear. This type-in is used to specify the interval between contour lines. These intervals can be typed in while pointing the mouse to the widget. A RETURN at the end will change the values. When a negative value is entered, all contour lines with a negative value will be dashed and all positive values will be drawn in solid lines. Optionally, after the interval value a range of values (**a**, **b**) can be specified which will cause only contour lines between **a** and **b** to be drawn.

**Color slice:** The color slice control window that turns up contains a function graph for red, green and blue that maps data values to colors. To change the red, green and blue function, press the left, middle and right mouse button, respectively, and drag the mouse to draw a new function. Also the keyboard cursor (arrow) keys can be used to change the function curves. Press right/left to move the function right/left. Press up/down to change the shape of the curve.

## 2.5 VIS-5D source code organization

In order to successfully make modifications to VIS-5D, a clear understanding of the source code is necessary. The source code, which is written in both C and FORTRAN77 (total ca. 35000 lines), can be split into four parts:

- **vis5d** main source code (directory **src**). The source code of **vis5d** is written in a modular fashion. The first systems **vis5d** was programmed for included the Silicon Graphics and Stellar Stardent using multi-threaded execution. The parts of the program that modify exclusive resources are enclosed within semaphore statements. Single-threaded execution is also possible. Due to the speed of these systems, the source code is easy to understand, because it is programmed in a very straightforward way. This makes modifications and enhancements to **vis5d** more easy.

bigger or smaller, respectively. When pressed and moved left or right, the clipping plane (i.e. plane of invisibility) moves towards or away from the user, respectively. When the right mouse button is pressed the box in the window can be translated.

- **Trajectory.** This mode is used for creating and displaying wind trajectories. This is only possible if the data set contains  $U$ ,  $V$  and  $W$  wind components and more than one time step. First, a position for the cursor must be selected. This can be done with the right mouse button. Then a time step must be selected with the STEP button. The trajectory will be traced forward from the chosen time step to the last time step and backward through time to the first time step. Finally, the middle mouse button must be pressed to activate the trajectory. Wind trajectories can be displayed in two ways: as line segments (2-D) and as ribbons (3-D). The trajectory window also contains two buttons labeled STEP and LENGTH. The STEP button is used to control the step size used in the trajectory tracing algorithm. The LENGTH value is used to control the length of the trajectories.
- **Slice.** This mode is used to reposition horizontal and vertical slices (contour slices, colored slices and wind vector slices). To move a slice, point the mouse at any slice corner and drag the slice to its desired position while holding the right mouse button. Horizontal slices can be positioned in any way as long as they are parallel to the bottom of the bounding box. Vertical slices can have any orientation as long as they remain vertical. They can be moved parallel by pointing to the middle of the top or bottom edge and then dragging to the new position while holding the right mouse button.
- **Label.** This mode is used to create and edit text labels in the 3-D window. First position the mouse pointer in the 3-D window and press the left mouse button. A vertical bar cursor will appear where the text can be typed in. The middle mouse button can be used to move the text to another position. To delete a text, point the cursor at the text and press the right mouse button.
- **Probe.** Used to inspect individual grid values by moving a 3-D cursor through the 3-D grid. For each physical variable, the value is printed along the left edge of the 3-D window.

### Wind slice buttons

If the data set contains  $U$  and  $V$  wind component variables there will be a row of four wind slice buttons (if the  $W$  wind component is present it is used, otherwise the  $W$  component is assumed to be zero for all grid points).

The location of the slice plane can be changed with the mouse while in “Slice” mode. The color of the slice can be changed by selecting the widget button with the right mouse button. The wind slice control window contains two type-in widgets to control the scale and density of the wind vectors. The scale parameter is used to multiply the length of the vectors drawn. The density parameter controls how many wind vectors are being displayed.

### Display buttons

The last set of widget buttons in the control window is used to control the display of 3-D

ANIMATE	Toggle animation on or off. Left and middle mouse button will result in forward animation, right mouse button in backward animation.
STEP	Step ahead one time step (left mouse button), go to first time step (middle mouse button) or step backward one step (right mouse button).
NEW VAR	Duplicate physical variables or invoke external analysis functions. Two kinds of new variables can be added: Clone variables (copies of existing variables) and external function variables (calculated as a function of the existing variables, see also appendix B.2).
EXIT	Exit the program. The decision must be verified.
NEW SURF	Compute a new 3-D contour surface.
TOP	Set 3-D window to a top view (left and middle mouse button) or to a bottom view (right mouse button).
SOUTH	Set 3-D window to a south view (left and middle mouse button) or to a north view (right mouse button).
WEST	Set 3-D window to a west view (left and middle mouse button) or to a east view (right mouse button).
TOPO	Toggle display of topography. Right mouse button will edit the topography color. The commandline option <code>-topo topographyfile</code> can be used to load a topography file. Default file is <code>EARTH.TOP0</code> .
MAP	Toggle display of map. The commandline option <code>-map mapfile</code> can be used to load a map file. The default map is <code>OUTLSUPW</code> .
BOX	Toggle the display of the 3-D box.
CLOCK	Toggle clock on or off.
SAVE	Save current graphics and colors (left mouse button) or only save colors (right mouse button).
RESTORE	Restore the data set last saved with the SAVE button.
GRID #s	Toggle bounds along the edges of the box in latitude, longitude and height coordinates or grid coordinates.
CONT #s	Toggle contour labeling on or off.
PRETTY	Toggle the 'pretty' rendering option on or off.
REVERSE	Toggle white box on black background to black box on white background.
SAVE PIC	Save picture in the 3-D window to a file.
PERSPEC	Toggle between perspective and orthogonal viewing projections.

#### Mouse behaviour buttons

This group of radio buttons (only one may be selected at the time) is used to determine the effect of pressing a mouse button while the mouse pointer is positioned in the 3-D window. To the immediate right of these buttons the mouse button legend is displayed. The following mouse behaviour buttons are available:

- **Normal.** This mode is used to view the graphics in the 3-D window. By pressing the left mouse button and dragging the mouse, the 3-D image can be rotated. The center mouse button has two functions. When pressed and moved up or down, the box gets

## 2.4.2 Graphical user interface

When starting `vis5d` it opens two windows: a 3-D window on the right of the screen and a control panel on the left.

The 3-D window is used to view and interact with the data. In its upper left corner an analog/digital clock is displayed which indicates the current time step. In the center of the window a box is drawn (when selected by the user) which depicts the domain of the 3-D data. All graphics will be displayed in this box. To the left side on the bottom, the values of all physical parameters are displayed when the Probe button is selected. In the right bottom corner, the name of the program is shown. An example is shown in figure 2.1.

The control window contains several groups of buttons that are used to control various functions. The graphical user interface is an X-Windows based interface that is device independent. Another possibility is to open the 3-D window with maximum size (full screen) and to open the control window and all other windows on a second terminal.

The control window (see figure 2.2) is used to control any graphic by clicking on its widget button with the left mouse button. A small control window will appear. To bring up the graphics control window without toggling the display, use the middle mouse button. To change the color of the graphic, click with the right mouse button on the widget button.

Some functions may not be present on the control window. This can happen because the function is not available on the hardware platform, the button does not apply to the data set or the button conflicts with a command line option that is set.

The buttons of the control window can be divided into four groups (see figure 2.2):

- control buttons,
  
- mouse behaviour buttons,
  
- wind slice buttons,
  
- display buttons.

### Control buttons

These buttons are used to toggle, activate or control various functions.

**vis5d command line options**

After producing a compressed file, the data can be visualized with the command

```
vis5d compfile [options]
```

The following command line options are available:

<code>-map mapfile</code>	Use a mapfile other than the default mapfile file <code>OUTLSUPW</code> . This default file is a world map in a normal latitude/longitude grid. The reader is referred to appendix A.5 for a description of the map file format.
<code>-topo topographyfile</code>	Use a topography file other than the default topography file <code>EARTH.TOP0</code> . This default file is a world topography in a normal latitude/longitude grid with heights and a land/water mask. The reader is referred to appendix A.6 for a description of the topography file format.
<code>-box x y z</code>	Specify the aspect ratio of the 3-D box in the 3-D window. The default ratios of this box are 2 2 1.
<code>-font fontname [height]</code>	Set the font used in the 3-D window. On the SGI system the height of the font can also be specified.
<code>-mbs n</code>	Override the assumed system memory size of 64 Mb.
<code>-rate ms</code>	Change the default animation rate to <i>ms</i> microseconds between frames. The default animation rate is 100 <i>ms</i> .
<code>-full</code>	Open the 3-D window as a borderless, full-screen sized window.
<code>-wdpy xdisplay</code>	Make the control window appear on a different X display.
<code>-alpha</code>	Use alpha blending instead of screen door transparency. This feature is only available on SGI systems.
<code>-wide w</code>	Set width of line segments in pixels. The default width is 1.0 pixels.

An overview of the command line options is given on the screen by running `vis5d` without arguments.

```
-lat XLATS XLATN
```

which sets the latitude range bounds (south, north) and

```
-lon XLONE XLONW
```

which sets the longitude range bounds (east, west) and

```
-hgt XHGTT XHGTT
```

which set the height range bounds (bottom, top) for the destination grid.

Before viewing the data set, the GRID files must be converted to a compressed file structure (COMP) file. This can be done using the command

```
comp5d N M F
```

where **N** is the first 3-D grid file number, **M** is the number of grid files in the data set and **F** is the name of the compressed grid file.

If the data set contains wind vector components (**U**, **V** and **W**) the horizontal wind (**SPD**) can be included in the COMP file using the command

```
comp5d N M F -wind SPD U
```

This will result in including **SPD** and **U** in the COMP file (not **V** and **W**).

To create 3-D grid file number **N** which allows 3-D grids of up to **M** points each, use the command

```
igu3d make N M
```

To get information about the COMP file made, use the command

```
compinfo COMP-file
```

that prints the grid dimensions, number of time steps, number of variables, etc. of the COMP file.

The command

```
help NAME
```

can be used to get a quick overview of the parameter format of the VIS-5D utilities.

where grid number  $K$  is produced by interpolating between grid numbers  $I$  and  $J$ , all in 3D grid file number  $N$ . Grid number  $K$  will be assigned day  $D$  (in YYDDD format) and time  $T$  (in HHMMSS format). The relative weighting of grids  $I$  and  $J$  is calculated from this time and date, assuming linear time interpolation. If grid  $K$  is not between grids  $I$  and  $J$  in date and time, `igg3d` prints an error message.

The second way to interpolate in time is to use the command

```
igg3d int I D T -setdel S M -lag U V -gr3df N
```

This will put a grid in the next empty slot of 3-D grid file number  $N$ , assigned to day  $D$  (in YYDDD format) and time  $T$  (in HHMMSS format). This grid will be interpolated from a sequence of grids, all in file number  $N$ , at grid numbers  $I$ ,  $I+S$ ,  $I+2S$ , ...,  $I+(M-1)S$ . This sequence of grids should be ascending in date and time. `igg3d` will search the sequence and linearly interpolate between the two consecutive grids from the sequence which bracket day  $D$  and time  $T$ . Furthermore, the interpolation will be done in a coordinate system moving at constant velocity  $(U, V)$ , where  $U$  and  $V$  are in m/s, with  $V$  positive for motion from south to north and  $U$  positive for motion from west to east. The two bracketing grids from the sequence will be shifted in latitude and longitude to their positions at day  $D$  and time  $T$ , and the result interpolated between these two spatially shifted grids. Furthermore, if the grids in the sequence are identified in their directory entries with variable name “ $U$ ” or “ $V$ ”, then the corresponding component of the velocity  $(U,V)$  will be subtracted from the grid values.

There is a built-in possibility to resample the GRID file. This is done with the commands

```
gg3d samp N M I J
```

and

```
gg3d ave N M I J
```

where a  $N*M$  grid is resampled to a  $I*J$  grid. The `samp` option will linearly interpolate between source grid points and is appropriate for increasing resolution. The `ave` option averages multiple source grid points and is used to decrease the resolution of a grid.

Starting `gg3d` without keywords will do a straight copy operation. `gg3d` can be invoked with the keyword

```
-size NLATS NLONS NHGTS
```

which will cause the destination grid dimensions to differ from the source grid dimensions. `NLATS`, `NLONS`, `NHGTS` are the destination grid latitude, longitude and height dimension, respectively. Other keywords that can be used are

file format (see appendix A.4). 3-D GRID file names consists of **GR3Dxxxx** where **xxxx** is the 3-D GRID *file number* (the number is assumed to be 4 digits). A 3-D GRID file consists of one or more 3-D data grids, which are numbered in the order in which they appear in the GRID file (starting with 1). Grids in a particular GRID file are referenced by the utilities using this *grid number*.

### VIS-5D utility commands

The directory information of the grids in the GRID files can be viewed with the command

```
igg3d list I J -gr3df N
```

where **N** is the 3-D grid file number and **I** and **J** give the range of grid numbers to list.

A short overview of the data in the GRID files such as the minimum and maximum values, the mean and the standard deviation and the number of grid points with missing data can be viewed with the command

```
igg3d info I J -gr3df N
```

with **I**,**J** and **N** the same as above.

The **igg3d** command also provides options for copying and deleting 3-D grids and interpolating 3-D grids in time. Sequences of 3-D grids are copied using the command

```
igg3d get N I J M K
```

where **N** is the source 3-D grid file number, **I** and **J** are the range of source grid numbers, **M** is the destination grid file number and **K** is the starting grid file number.

A single grid can be copied using the command

```
igg3d copy I J -gr3df N
```

where **N** is the 3-D grid file number, **I** is the number of the source grid and **J** is the number of the destination grid. The same way a number of grids can be deleted with the command

```
igg3d del I J -gr3df N
```

There are two ways to interpolate in time. The first way is use the command

```
igg3d ave K I J D T -gr3df N
```



The reader is referred to chapter 3 for information about porting VIS-5D to other hardware platforms.

## 2.3 Capabilities

`vis5d` can handle physical variables in single time steps and multiple time steps. It is also possible to produce a new variable as a function of the existing variables. All these data can be easily visualized, manipulated and looked at from every possible viewpoint.

`vis5d` is capable of visualizing data in the following ways:

- 3-D contour surface (isosurface): a surface that shows the 3-D volume, bounded by a particular isovalue. This is the 3-D equivalent of a 2-D contour line,
- contour slice: a slice with lines on which values in the data set are equal,
- colored slice: a slice with data values mapped to colors,
- wind vector slice: a slice that depicts the motions of the wind by drawing arrows which point in the direction of the wind. The length of each line segment indicates its magnitude. The tail of the line segments are all anchored within a horizontal or vertical plane through the 3-D box,
- wind trajectories: tracing of the motion of air through the 3-D volume much like the trailing of line smoke in a wind tunnel,
- probe: inspection of individual grid values.

All these visualization techniques can be shown simultaneously during one or more time steps. All graphics can be manipulated interactively, even during animation.

With equal ease, text labels can be put in the 3-D window and a window dump can be made from this window and saved to a file or printed.

## 2.4 User interface

In VIS-5D there are two ways of controlling the data. The first way of doing this is using command line options when starting `vis5d` or making and analyzing the data with the utility commands. The second way is with the use of the graphical user interface within the program.

### 2.4.1 Command line interface

With `vis5d` additional programs are supplied that enable the user to make and analyse the data. The grids are assumed to be in 3-D GRID file format (see appendix A.3) or in COMP

VIS-5D is offered under the terms of the GNU General Public License. There is no warranty for the VIS-5D package.

VIS-5D is being distributed as a stand alone system free of charge. However, it is also a subsystem of a much larger (ca. two million lines of FORTRAN code) system named McIDAS (Man-computer Interactive Data Access System, see [Smit75] and [Hibb85]).

## 2.2 System requirements

VIS-5D version 3.1 works on the following hardware platforms:

### **Stardent GS-1000 or GS-2000 with:**

true-color display (i.e. 32 planes),  
32MB RAM,  
optional SpaceBall is supported,  
Stellix version 2.3 or later is suggested.

### **Silicon Graphics with:**

24-bit color and Z-buffer (or 8-bit Indigo),  
32MB RAM,  
IRIX version 4.0.1 or higher,  
multiple processors are used when present.

### **IBM RS/6000:**

model 320H or higher,  
requires 24-bit color and Z-buffer (GTO desirable),  
32MB RAM,  
AIX version 3 or later is suggested.

To be able to use VIS-5D on a specific target machine, the following resources should be available:

- C Compiler,
- FORTRAN Compiler,
- X-Windows,
- 3-D graphics library with Z-buffer.

And preferable:

- 32 MB of memory,
- 24-bit color display with high performance 3-D graphics hardware.

## Chapter 2

# Overview of VIS-5D

This chapter gives an overview of VIS-5D, as originally distributed by ftp<sup>1</sup>. The latest available version of VIS-5D (3.1, August 1993) is used as the basis of this chapter. The modifications made in order to visualize the HIRLAM data with VIS-5D are described in chapter 5.

### 2.1 Background

VIS-5D is a software system for visualizing data made by numerical weather models and similar sources. VIS-5D works on data in the form of a five-dimensional rectangle. That is, the data are real numbers at each point of a “grid” or “lattice” which spans three space dimensions, one time dimension and a dimension for enumerating multiple physical variables. VIS-5D works perfectly well on data sets with only one variable or one time step (i.e. no time dynamics). However, the data should have some depth in all three spatial dimensions. This makes visualization of 2-D fields impossible.

The VIS-5D package includes the `vis5d` visualization program, several programs for managing and analyzing five-dimensional data grids, and instructions and sample source code for converting your data into its file format. The VIS-5D source code is also included so it can be modified or extended. There are also sample data sets from the LAMPS (Limited Area Meso-Scale Prediction System) model and from Robert Schlesinger’s numerical thunderstorm model available.

VIS-5D version 1.0 was written by Bill Hibbard and Dave Santek of the University of Wisconsin Space Science and Engineering Center, supported by the NASA Marshall Space Flight Center, and by Marie-Francoise Voidrot-Martinez of the French Meteorology Office. Later version enhancements were written by Bill Hibbard, Brian Paul, and Andre Battaiola. Dave Kamins and Jeff Vroom of Stellar Computer, Inc. provided substantial help and advice in using the Stellar software libraries.

---

<sup>1</sup>VIS-5D can be obtained via anonymous ftp from `vis5d.ssec.wisc.edu` in the directory `/pub/vis5d`.

- VIS-5D is restricted to height as the vertical coordinate,
- VIS-5D doesn't support the creation of color map legends,
- VIS-5D supports only a 0-D (point) probe, instead of a 1-D (curve) probe.

Solving these problems requires a relatively small effort compared to the effort required to implement a complete visualization package.

Furthermore, the following features should be supported:

- support for various vertical coordinate systems,
- colormap legends (for color mappings, a legend should be (optionally) displayed giving information about the correspondence between colors and data values),
- data probe (to inspect the value of one or more physical variables along a path through the data domain).

### 1.3 Visualization tools

To visualize the output of HIRLAM a visualization tool is needed. Visualization programs can be divided into two categories. First, there are programs written with a specific purpose in mind. These kind of programs contain specific routines to calculate and display an (often restricted) range of images. Secondly, there are visualization toolkits. These consist of a collection of modules all containing only a small part of the total visualization process. By combining modules the user can construct larger applications suited to his visualization needs. Advantages of a toolkit are flexibility and the ease with which the toolkit can be enhanced with new modules. For this reasons the required visualizations were first attempted using apE III, a toolkit for visualization. Due to the large number of shortcomings and bugs (see [Baas93]), apE was abandoned in favour of another package called VIS-5D. Chapter 2 will give an overview of VIS-5D.

VIS-5D was chosen for the following reasons:

- VIS-5D is written specifically for the visualization of meteorological data,
- free (under the GNU licence),
- the source code is available,
- support (an active VIS-5D mailing-list is available),
- VIS-5D is written with portability in mind,
- the KNMI evaluated VIS-5D and are very enthusiastic about it.

To make VIS-5D able to visualize the HIRLAM output as described in section 1.2, the following problems had to be solved:

- VIS-5D only runs on SGI, Stellar Stardent and IBM RS/6000 systems,
- VIS-5D can't read the HIRLAM output data format,
- VIS-5D can't visualize 2-D data sets,

## 1.2 Project description

Currently, the physical variables produced by HIRLAM include:

- wind direction and -speed,
- temperature,
- (surface) pressure,
- air density,
- humidity.

For the purpose of visualization, the output variables can be divided into three categories:

- 2-D scalar fields (surface pressure),
- 3-D scalar fields (temperature, air density, humidity),
- 3-D vector fields (wind).

The wind is actually a 3-D field of 2-D vectors because the vertical component of the wind vectors (physical variable  $W$ ) is currently not present (it is not calculated by HIRLAM, but this can easily be changed).

The following visualization techniques, stationary or as time loops, are required:

- For 2-D scalar variables and 2-D slices taken from 3-D scalar variables:
  - contour images (lines connecting the gridpoints where the variable has the same value, the contours labeled with corresponding values),
  - color mappings (data values are mapped to colors, according to a specified colormap),
  - discretized color mappings (the data is first mapped into intervals using a step function, so that colored bands are formed).
- For 3-D scalar variables:
  - iso-surfaces (3-D equivalent of a contour line).
- For 3-D vector variables:
  - 2-D slices with arrows representing wind vectors.
  - wind trajectories (streaklines, depicting the path of particles through a stationary point, moved by the wind).

# Chapter 1

## Introduction

### 1.1 Project background

In collaboration with the KNMI (the Royal Dutch Meteorological Institute), the High Performance Computing Division of the Department of Computer Science, Leiden University (RUL) researches efficient implementations of a numerical weather forecasting model on various parallel computer systems. This model, called HIRLAM (High Resolution Limited Area Model), is the result of a cooperative project between Denmark, Finland, Iceland, Ireland, The Netherlands, Norway and Sweden. The purpose of the HIRLAM project is to design a state-of-the-art weather forecasting system. HIRLAM is already used by some of the participating countries to make predictions of the weather. The horizontal domain of HIRLAM as used at KNMI is displayed in figure 1.1.



Figure 1.1: Horizontal domain of HIRLAM

The output of the atmospheric circulation model of HIRLAM consists of large volumes of four-dimensional data (currently several time steps each consisting of ca. 100 by 100 by 16 calculated data points) representing several physical variables. Visualization is a natural way to obtain insight in these data.

C.2.5	futil . . . . .	82
C.2.6	grib2vis . . . . .	83
<b>D</b>	<b>VIS-5D modifications</b>	<b>85</b>
D.1	HP port modifications . . . . .	85
D.1.1	LUI changes . . . . .	85
D.1.2	vis5d changes . . . . .	85
D.2	2-D fields modifications . . . . .	87
D.3	Logarithmic interpolation modifications . . . . .	97
D.4	VOGL related modifications . . . . .	99
<b>E</b>	<b>VOGL implementation details</b>	<b>101</b>
E.1	VOGL enhancements . . . . .	101
E.1.1	Z-buffer functions . . . . .	101
E.1.2	RGB color functions . . . . .	102
E.1.3	Lighting functions . . . . .	103
E.1.4	Depthcueing functions . . . . .	105
E.2	VOGL device driver structure . . . . .	105
<b>F</b>	<b>Example Images</b>	<b>109</b>



<b>5</b>	<b>VIS-5D modifications</b>	<b>43</b>
5.1	Visualization of 2-D fields . . . . .	43
5.2	Logarithmic interpolation . . . . .	46
5.2.1	Horizontal slices . . . . .	46
5.2.2	Vertical slices . . . . .	47
5.2.3	Iso surfaces . . . . .	47
5.2.4	Wind trajectories . . . . .	48
5.2.5	Data probe . . . . .	48
5.3	Scaling vertical wind component . . . . .	49
5.4	Topography conversion . . . . .	49
5.5	Map conversion . . . . .	50
<b>6</b>	<b>Results</b>	<b>51</b>
<b>7</b>	<b>Conclusions</b>	<b>53</b>
<b>8</b>	<b>Future work</b>	<b>55</b>
<b>A</b>	<b>File formats</b>	<b>59</b>
A.1	GRIB file format . . . . .	59
A.2	ASIMOF file format . . . . .	64
A.3	GRID file format . . . . .	64
A.4	COMP file format . . . . .	66
A.4.1	Old style format . . . . .	67
A.4.2	New style format . . . . .	67
A.4.3	Variable NL format . . . . .	68
A.5	VIS-5D map file format . . . . .	69
A.6	VIS-5D topography file format . . . . .	70
<b>B</b>	<b>VIS-5D source code structure</b>	<b>71</b>
B.1	VIS-5D main source code . . . . .	71
B.2	External User functions . . . . .	73
B.3	Utilities . . . . .	74
B.4	Little User Interface (LUI) library . . . . .	75
<b>C</b>	<b>Libraries</b>	<b>77</b>
C.1	KNMI libraries . . . . .	77
C.1.1	port . . . . .	77
C.1.2	gcod . . . . .	77
C.1.3	util . . . . .	78
C.1.4	vari . . . . .	78
C.1.5	prpo . . . . .	78
C.1.6	grw1 . . . . .	79
C.2	RUL libraries . . . . .	79
C.2.1	asimof . . . . .	80
C.2.2	grib . . . . .	81
C.2.3	interpol . . . . .	81
C.2.4	cutil . . . . .	82

# Contents

<b>Preface</b>	<b>i</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Project background . . . . .	1
1.2 Project description . . . . .	2
1.3 Visualization tools . . . . .	3
<b>2 Overview of VIS-5D</b>	<b>5</b>
2.1 Background . . . . .	5
2.2 System requirements . . . . .	6
2.3 Capabilities . . . . .	7
2.4 User interface . . . . .	7
2.4.1 Command line interface . . . . .	7
2.4.2 Graphical user interface . . . . .	12
2.5 VIS-5D source code organization . . . . .	15
<b>3 Porting VIS-5D</b>	<b>19</b>
3.1 3-D graphics library . . . . .	19
3.1.1 An introduction to the SGI GL . . . . .	21
3.1.2 Z-buffering . . . . .	25
3.1.3 Lighting . . . . .	25
3.1.4 Depth-cueing . . . . .	27
3.1.5 Scan conversion . . . . .	28
3.1.6 VOGL device drivers . . . . .	28
3.2 Miscellaneous problems . . . . .	28
<b>4 Interfacing HIRLAM with VIS-5D</b>	<b>31</b>
4.1 HIRLAM output . . . . .	31
4.2 Conversion to GRID versus COMP . . . . .	31
4.3 Vertical coordinate . . . . .	32
4.4 ASIMOF to GRID conversion specification . . . . .	34
4.5 Conversion from GRIB to GRID . . . . .	35
4.5.1 KNMI libraries . . . . .	35
4.5.2 RUL libraries . . . . .	36
4.5.3 Conversion program . . . . .	37
4.6 GRID to COMP conversion . . . . .	41



# Preface

This is a master thesis about the visualization of meteorological data on behalf of Leiden University and the KNMI (Royal Dutch Meteorological Institute). We would like to thank a number of people who helped us with the many problems that arose during the research.

In the first place we would like to thank our supervisor Lex Wolters (Leiden University) for his support during the research and the time he has spent in correcting this thesis.

Special thanks goes to Gerard Cats (KNMI), Toon Moene (KNMI) and Ben Wichers Schreur (KNMI) for their help in explaining the sometimes complex working of meteorology. Furthermore we would like to thank Nies Huijsmans (Leiden University) for his advice on the subject of computer graphics.

# Meteorological Visualization using VIS-5D

Simon Baas    Hans de Jong  
Leiden University

August 1993