

# 3D Visualization of Weather Radar Data

Examensarbete utfört i datorgrafik av

Aron Ernvik

LITH-ISY-EX-3252-2002  
januari 2002

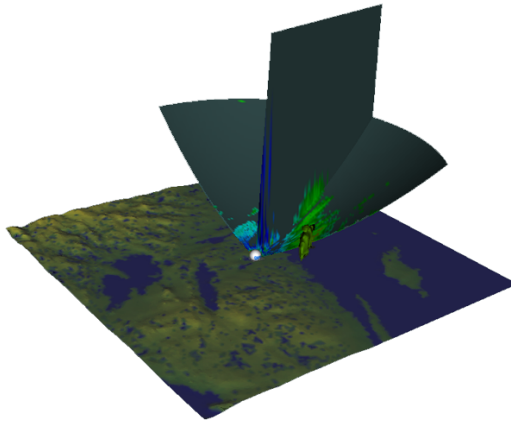


# *3D Visualization of Weather Radar Data*

*Thesis project in computer graphics at Linköping University*

*by*

*Aron Ernvik*




LITH-ISY-EX-3252-2002

*Examiner: Ingemar Ragnemalm*

*Linköping, January 2002*



	<b>Avdelning, Institution</b> Division, Department  Institutionen för Systemteknik 581 83 LINKÖPING	<b>Datum</b> Date 2002-01-15
<b>Språk</b> Language Svenska/Swedish X Engelska/English	<b>Rapporttyp</b> Report category Licentiatavhandling X Examensarbete C-uppsats D-uppsats Övrig rapport _____	<b>ISBN</b>  <b>ISRN</b> LITH-ISY-EX-3252-2002  <b>Serietitel och serienummer</b> <b>ISSN</b> Title of series, numbering      _____
<b>URL för elektronisk version</b> <a href="http://www.ep.liu.se/exjobb/isy/2002/3252/">http://www.ep.liu.se/exjobb/isy/2002/3252/</a>		
<b>Titel</b> Tredimensionell visualisering av väderradardata Title      3D visualization of weather radar data  <b>Författare</b> Aron Ernvik Author		
<b>Sammanfattning</b> Abstract There are 12 weather radars operated jointly by SMHI and the Swedish Armed Forces in Sweden. Data from them are used for short term forecasting and analysis. The traditional way of viewing data from the radars is in 2D images, even though 3D polar volumes are delivered from the radars. The purpose of this work is to develop an application for 3D viewing of weather radar data.  There are basically three approaches to visualization of volumetric data, such as radar data: slicing with cross-sectional planes, surface extraction, and volume rendering. The application developed during this project supports variations on all three approaches. Different objects, e.g. horizontal and vertical planes, isosurfaces, or volume rendering objects, can be added to a 3D scene and viewed simultaneously from any angle. Parameters of the objects can be set using a graphical user interface and a few different plots can be generated.  Compared to the traditional 2D products used by meteorologists when analyzing radar data, the 3D scenes add information that makes it easier for the users to understand the given weather situations. Demonstrations and discussions with meteorologists have rendered positive reactions. The application will be installed and evaluated at Arlanda airport in Sweden.		
<b>Nyckelord</b> Keyword visualisering, väder, radar, datorgrafik, 3d, visualization, weather, graphics		



# Abstract

There are 12 weather radars operated jointly by SMHI and the Swedish Armed Forces in Sweden. Data from them are used for short term forecasting and analysis. The traditional way of viewing data from the radars is in 2D images, even though 3D polar volumes are delivered from the radars. The purpose of this work is to develop an application for 3D viewing of weather radar data.

There are basically three approaches to visualization of volumetric data, such as radar data: slicing with cross-sectional planes, surface extraction, and volume rendering. The application developed during this project supports variations on all three approaches. Different objects, e.g. horizontal and vertical planes, isosurfaces, or volume rendering objects, can be added to a 3D scene and viewed simultaneously from any angle. Parameters of the objects can be set using a graphical user interface and a few different plots can be generated.

Compared to the traditional 2D products used by meteorologists when analyzing radar data, the 3D scenes add information that makes it easier for the users to understand the given weather situations. Demonstrations and discussions with meteorologists have rendered positive reactions. The application will be installed and evaluated at Arlanda airport in Sweden.

---

# Table of contents

1	Introduction	1
	1.1 Background	1
	1.2 Purpose	1
	1.3 Order of work	1
	1.4 Outline of this thesis	2
2	Introduction to weather radars	5
	2.1 Background	5
	2.2 Radar hardware	5
	2.2.1 The transmitter	6
	2.2.2 The antenna	7
	2.2.3 The waveguide	8
	2.2.4 The transmit/receive switch and the receiver	8
	2.3 Radar parameters	8
	2.4 The radar equation	10
	2.5 Doppler velocity measurements	12
	2.6 Radar data	13
	2.7 Range, height, and distance	15
3	RAVE - current and future use	19
	3.1 Background	19
	3.2 System design	19
	3.2.1 Graphical user interface	20
	3.3 Products from 3D radar data	21
	3.3.1 The PPI product	21
	3.3.2 The CAPPI product	22
	3.3.3 The PCAPPI product	23
	3.3.4 The RHI product	24
	3.3.5 The VAD product	25
	3.4 RAVE users	26
4	Visualization of volumetric data	29
	4.1 Introduction	29
	4.2 Polar and cartesian volumes	30
	4.2.1 Gaussian splatting	30
	4.2.2 Shepard's method	30

---



---

4.3	Different approaches	31
4.4	Interpolation	32
4.4.1	Nearest neighbour interpolation	32
4.4.2	Trilinear interpolation	32
4.5	Slicing techniques	33
4.6	Surface rendering techniques	34
4.6.1	The marching cubes algorithm	34
4.7	Volume rendering techniques	36
4.7.1	Object-order techniques	37
4.7.2	Image-order techniques	38
4.8	Shading	41
4.8.1	Polygon shading	42
5	Implementation	45
5.1	Visualization software	45
5.1.1	Visual programming systems	46
5.1.2	The Visualization Toolkit, VTK	49
5.1.3	3D graphics APIs	49
5.2	Choice of software	49
5.3	Software design	50
5.4	Visualization objects	52
5.4.1	The topography object	52
5.4.2	The radar, bounding box and axis objects	52
5.4.3	The elevation surface object	53
5.4.4	The RHI object	54
5.4.5	The CAPPI object	55
5.4.6	The glyphs object	56
5.4.7	The winds object	57
5.4.8	The VAD object	57
5.4.9	The isosurface object	58
5.4.10	The volume rendering object	59
5.4.11	The radar ray object	61
6	Evaluation and conclusions	63
6.1	Introduction	63
6.2	Evaluation	63
6.2.1	Usefulness	63
6.2.2	Performance	65
6.3	Future improvements	66
6.4	Conclusions	67
7	Resources	69
7.1	Books and articles	69
7.2	Internet	70
A	Class Diagrams	71

---

---

# List of figures

Figure 1.	This is what a radar looks like from the outside. This one is positioned near the southern tip of Sweden's largest island, Gotland.	6
Figure 2.	Simple block diagram of a radar. (Adapted from Rinehart, 1991)	7
Figure 3.	The surface generated by a radar scan when the elevation angle is held constant and the azimuth angle varies from 0 to 360 degrees.	14
Figure 4.	Range, height, and distance.	16
Figure 5.	The basic system design in RAVE.	20
Figure 6.	A PPI image and the main graphical user interface in RAVE.	22
Figure 7.	Generating a CAPPI image. The output image is orthogonal to the paper.	23
Figure 8.	A CAPPI (left) and a PCAPPI (right) image, showing data from the same location, time, and altitude (2 500 metres).	24
Figure 9.	Generating a RHI image. The output image is parallel to the paper.	24
Figure 10.	VAD circles.	25
Figure 11.	Velocity/azimuth display of the Doppler velocity along a VAD circle.	26
Figure 12.	Pixels and voxels.	29
Figure 13.	Nearest neighbour interpolation.	32
Figure 14.	Trilinear interpolation.	33
Figure 15.	The marching cubes.	36
Figure 16.	Object-order volume rendering (forward mapping). The volume is transformed and mapped into image space.	37
Figure 17.	Image-order volume rendering (inverse mapping). The image plane is transformed and mapped onto the volume.	39
Figure 18.	Perspective ray casting.	40
Figure 19.	Block diagram of a typical visualization system.	46
Figure 20.	IRIS Explorer user interface.	48
Figure 21.	Basic system design.	50
Figure 22.	Main GUI screenshot when a few objects have been added.	51
Figure 23.	Topography, radar, bounding box, and height axis objects.	53
Figure 24.	Creating an elevation surface. Radar as seen from above.	54
Figure 25.	A range/height indicator, or RHI, object, and its manager.	55
Figure 26.	A semitransparent CAPPI object and its manager.	56
Figure 27.	A glyphs object.	57
Figure 28.	A winds object, as seen from above.	58

---

---

Figure 29.	A VAD object.	58
Figure 30.	Two isosurface objects, with isovalues 14.5 and 18.5 dBZ for outer and inner surface, respectively.	59
Figure 31.	The volume rendering manager.	60
Figure 32.	A volume rendering object.	61
Figure 33.	A radar ray object and its manager.	61
Figure 34.	Graphical user interface class diagram.	72
Figure 35.	Visualization objects class diagram.	75

---



---

# List of tables

Table 1.	Radar bands and their frequencies and wavelengths. From <i>Rinehart</i> (1991). Visible light wavelegths range from 400 to 700 nm.	9
Table 2.	Two different computer configurations tested with RAVE.	65

---



## Introduction

---

### 1.1 Background

Swedish Meteorological and Hydrological Institute, SMHI, runs a network of 12 weather radars together with the Swedish Armed Forces. Data from them are used for short term weather forecasting and analysis; for detecting wind speeds and directions as well as rain fall amounts. The radars deliver data in the form of 3D polar volumes four times an hour, day and night, every day of the year. In order to meet the demand for a user friendly tool to visualize and analyse the data, the development of a software system called RAVE (which is an abbreviation for Radar Analysis and Visualization Environment) was initiated in 1996. From the beginning RAVE was designed to, among other things, “create and visualize arbitrary 2D products based on 3D data” and to be able to “render 3D views of polar and cartesian volume data with and without backdrops” (*Bolin & Michelson*, 1996). During 1997 a first version of RAVE was implemented that included basic functionality but it was not able to render any 3D views - neither is the current version of RAVE. This project aims to fill this hole.

### 1.2 Purpose

The purpose of this work is to design and implement a generic, object-oriented application for 3D visualization of weather radar data. The application should be integrated with existing software.

### 1.3 Order of work

This project spans 20 weeks, starting September 2001. During this period of time, the following tasks should be completed:

- 
1. Literature studies. Searching for and reading some of what has been written about 3D visualization of volumetric data in general, and radar data in particular.
  2. User interviews. Finding out what features the users of the application want.
  3. Data analysis. Investigating the nature of weather radar data.
  4. Platform comparison. Comparing some software platforms and deciding which one is most suitable for the purpose.
  5. Experimental programming. Learning to develop software using the chosen software environment.
  6. Implementation of basic features. A standard PC workstation should be used and the performance needs to be reviewed before any advanced features, such as rendering sequences of radar data, are implemented.
  7. Implementation of advanced features, given the limited time of the project.
  8. Evaluation.

## **1.4 Outline of this thesis**

Chapter 2, *Introduction to weather radars*, treats basic theory of weather radars for the reader who is not familiar with the subject. It describes the nature of the data delivered from a radar.

Chapter 3, *RAVE – current and future use*, includes an overview of the functionality in the current version of RAVE. There are also excerpts from the original design, in which it was stated what 3D graphics capabilities RAVE should have. Some common (2D) products that RAVE is able to generate are presented, as well as some that are added with this project.

Chapter 4, *Visualization of volumetric data*, describes some different methods and aspects of visualization of 3D data in a 2D image.

Chapter 5, *Implementation*, starts with a description and comparison of some conceivable software platforms. A platform is chosen and some design and implementation details follow.

Chapter 6, *Evaluation and conclusions*, finally evaluates the work and suggests areas of future research.



---

---

There is also an appendix: *Appendix A, Class diagrams*. This appendix contains a couple of class diagrams and textual descriptions of these.

---

---

---

# 2

## Introduction to weather radars

---

### 2.1 Background

Radar is an acronym for *radio detection and ranging*. Radar devices transmit electromagnetic waves, receive echoes from targets in the surroundings and determine various things about the targets from the characteristics of the received echo signals. Radar technology was developed primarily before and during the Second World War. Back then, the main goal was to scan the air for enemy airplanes. Radar technology is still being used for aircraft detection but nowadays radar is also used for a variety of other purposes - one of which is for detection of severe weather, including heavy thundershowers, hail, tornadoes, hurricanes and strong wind storms.

This chapter will give a brief overview of the components and important parameters a radar system and an equally brief discussion of the radar equation. This equation describes the amount of power that is received back at the radar in terms of wavelength, particle size and a few other things. Finally, something is told about the data delivered from a typical radar in the Swedish network.

### 2.2 Radar hardware

From the outside, a radar looks as shown in Figure 1. Its appearance is dominated by the protective dome, or *radome*, which contains the antenna and the tower or foundation. Typical radius for the radome is 2-3 metres. Inside it, and (typically) in a shed at the base of the tower, are several subsystems. A simple block diagram of a radar is shown in Figure 2.

---

**Figure 1.**

This is what a radar looks like from the outside. This one is positioned near the southern tip of Sweden's largest island, Gotland.

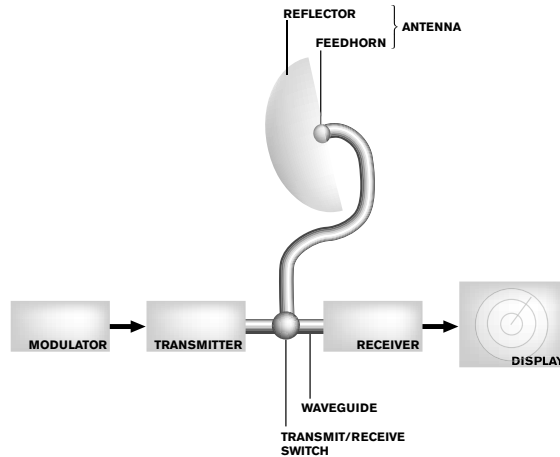


### **2.2.1 The transmitter**

The transmitter is the source of electromagnetic radiation. It generates a high frequency signal which leaves the radar through the antenna. The transmitter is controlled by the modulator. The purpose of the modulator is to switch the transmitter on and off and to provide the correct waveform for the transmitted pulse. Typical pulse durations range from 0.1 to 10  $\mu$ s. The number of pulses per second is denoted PRF (pulse repetition frequency) and this typically ranges from 200 to 3 000 Hz. Every pulse travels from the radar's antenna at the speed of light and if it hits any target along its path, some of its energy will be reflected back toward the radar. If the energy of this reflection is large enough it will be detected by the radar. By measuring the time between the sending and receiving of the pulse, the distance to the target can be easily calculated.

**Figure 2.**

Simple block diagram of a radar. (Adapted from *Rinehart, 1991*)



### 2.2.2 The antenna

The antenna directs the radar's signal into space. For any given electromagnetic frequency, a larger directional antenna will give a smaller antenna beam pattern and thus better angular resolution. An antenna that sends the radiation equally in all directions is called an *isotropic antenna*, but this type of antennas is not used in radars. However, an isotropic antenna is often the antenna against which radar antennas are measured to compare them to each other (via the *antenna gain* parameter, treated in Section 2.3 on page 8). The radiation from an isotropic antenna is much like the radiation from a candle whereas the radiation from a radar antenna is more like the radiation from a flashlight.

Radars have an antenna and a reflector. The antenna transmits the radar signal through the feedhorn toward the reflector which then reflects and directs the signal away from the radar. Most weather radars have reflectors which are parabolic in cross-section and circular when viewed from the front or back. The beam pattern formed by such a circular parabolic reflector is usually quite narrow, typically one degree in width for the mainlobe of the pattern. The combination of the feedhorn and reflector is often collectively referred to as the "antenna", in this report and otherwise. (*Rinehart, 1991*, pp 11)

---

### 2.2.3 The waveguide

The conductor connecting the radar transmitter and antenna is called a *waveguide*. This is usually a hollow, rectangular, metal conductor whose interior dimensions depend on the wavelength of the signals being carried. Coaxial cables are too lossy for these purposes at the high frequencies used by radars. (*Ibid.*)

### 2.2.4 The transmit/receive switch and the receiver

The purpose of the transmit/receive switch is to protect the receiver from the high power of the transmitter. Most radars transmit from a few thousand watts to 1 MW of power, but the receiver is designed to detect powers of  $10^{-10}$  W, i.e. 0.1 nW, or less. At transmission, the transmitter is connected to the antenna and the receiver is disconnected. As soon as a pulse has been transmitted, the receiver is connected until the next pulse should be transmitted. The receiver is designed to detect and amplify the very weak signal echoes received by the antenna. Most receivers in weather radars are of the so-called *superheterodyne* type in which the received signal is mixed with a reference signal at some frequency which is different from the transmitted frequency. This mixing converts the signal to a much lower frequency (30 to 60 MHz) at which it can be more easily processed. (*Ibid.*)

## 2.3 Radar parameters

One of the most important parameters of any radar is the wavelength (or frequency) for which it is designed. The transmitter and antenna must both be specifically designed for the same wavelength. Different wavelengths are useful for detecting objects with different shapes and sizes. Short wavelength radars more effectively detect small particles, such as drizzle drops or cloud droplets. However, using shorter wavelengths, a larger part of the energy in the waves is absorbed by the reflecting particles. This makes it difficult to accurately measure the back-scattered energy for more distant targets that lie beyond the range of closer targets. The damping process is known as *attenuation*.

Using longer wavelengths, the attenuation becomes less effective. This means that a distant thunderstorm behind a closer thunderstorm will appear on the radar screen with a more accurate intensity. Wavelength and frequency of electromagnetic waves are related through the well-known equation

$$f = \frac{c}{\lambda} \quad (\text{Eq 1})$$

where  $f$  is the frequency,  $c$  the speed of light, and  $\lambda$  the wavelength. The frequencies used by radars range from 100 MHz through 100 GHz, and within this frequency range different frequency bands have been defined as seen in Table 1. All 12 radars operated in Sweden use frequencies within the c band. Generally shorter wavelengths mean smaller and less expensive equipment.

**Table 1.**

Radar bands and their frequencies and wavelengths. From *Rinehart* (1991). Visible light wavelengths range from 400 to 700 nm.

Band designation	Nominal frequency	Nominal wavelength
HF	3-30 MHz	100-10 m
VHF	30-300 MHz	10-1 m
UHF	300-1000 MHz	1-0.3 m
L	1-2 GHz	30-15 cm
S	2-4 GHz	15-8 cm
C	4-8 GHz	8-4 cm
X	8-12 GHz	4-2.5 cm
K <sub>u</sub>	12-18 GHz	2.5-1.7 cm
K	18-27 GHz	1.7-1.2 cm
K <sub>a</sub>	27-40 GHz	1.2-0.75 cm
mm	40-300 GHz	7.5-1 mm

Another important parameter is the size of the reflector. Typical diameters for weather radar reflectors range from 30 cm to as much as 10 m. Yet another measure of importance to radar antennas is the *antenna gain*. The gain ( $g$ ) of an antenna is the ratio of the power that is received at a specific point in space to the power that would be received at the same point from an isotropic antenna. This is usually measured logarithmically in decibels and is then written as

$$G(\text{dB}) = 10 \log_{10} \frac{p_1}{p_2} \quad (\text{Eq 2})$$

where  $p_1$  and  $p_2$  represent the power of the two antennas. Typical antenna gains range from 20 dB to 45 dB. A fourth important param-

eter is the antenna beamwidth. Antenna gain and antenna beamwidth are related as given by

$$g = \frac{\pi^2 \cdot k^2}{\theta \cdot \varphi} \quad (\text{Eq 3})$$

where  $\theta$  and  $\varphi$  are the horizontal and vertical beamwidths, respectively, and  $k^2$  depends on the kind and shape of the antenna. For circular reflectors,  $k^2 = 1$  and  $\theta = \varphi$ . The beamwidths are the angular distances across the beam pattern at the point where the power is reduced to one half of the peak power. (*Ibid.*)

## 2.4 The radar equation

In this section, the radar equation will be described for two different conditions: for point targets and for distributed targets. The two equations both describe the amount of energy reflected towards the radar by particles in the air, in terms of the size of the particles and antenna properties. Details about the radar equation can be found in *Rinehart* (1991), chapters 4 and 5, from which the facts in this section and subsections have been compiled.

For an isotropic antenna, the area covered by a single, expanding wave is equal to the area on the surface of a sphere at the corresponding distance. The power density  $S$ , i.e. power per unit area, can thus be written as

$$S = \frac{p_t}{4\pi r^2} \quad (\text{Eq 4})$$

where  $p_t$  is the transmitted power, and  $r$  is the distance from the radar. To get the power  $p_\sigma$  received by a target with area  $A_\sigma$  when a non-isotropic antenna with gain  $g$  is used, all that needs to be done is to multiply  $S$  with  $A_\sigma$  and  $g$  - assuming the target is positioned along the center of the radar beam axis:

$$p_\sigma = \frac{p_t g A_\sigma}{4\pi r^2} \quad (\text{Eq 5})$$

This power is usually reradiated isotropically back into space. Some of this reradiated energy will be received back at the radar. The amount of energy detected by the radar is given by



$$p_r = \frac{p_i \sigma A_e}{4\pi r^2} = \frac{p_i g A \sigma A_e}{(4\pi)^2 r^4} \quad (\text{Eq 6})$$

where  $A_e$  is the effective area of the receiving antenna. This area can be expressed in terms of the antenna gain and the wavelength  $\lambda$  of the radar:

$$A_e = \frac{g\lambda^2}{4\pi} \quad (\text{Eq 7})$$

One last refinement that will yield *the radar equation for point targets* has to do with the area of the target,  $A_\sigma$ . This area is not necessarily the size the target appears to the radar. Instead, a parameter called the *backscattering crosssectional area* of the target is used, and it is called  $\sigma$ .<sup>1</sup> The radar equation for a point target located on the center of the antenna beam pattern can thus be written as

$$p_r = \frac{p_i g^2 \lambda^2 \sigma}{64\pi^3 r^4} \quad (\text{Eq 8})$$

However, users of weather radars are usually not interested in point targets but in distributed targets; when a radar is aimed at a meteorological target, there are many raindrops, cloud particles or snowflakes within the radar beam at the same time. The above equation (Equation 8) is not applicable in these cases. Without going into details, the radar equation for distributed targets is given here:

$$p_r = \frac{\pi^3 p_i g^2 \theta \phi h |K|^2 l z}{1024 \ln 2 \lambda^2 r^2} \quad (\text{Eq 9})$$

As before,  $\theta$  and  $\phi$  are the horizontal and vertical beamwidths, respectively.  $h$  is the pulse length in space corresponding to the duration  $\tau$  of the transmitted pulse.  $|K|^2$  is the *dielectric constant of precipitation* and depends on the particles' water phase and the wavelength of the radar<sup>2</sup>. The parameter  $l$  describes attenuation, i.e. loss of power in travelling through a medium such as the atmosphere, clouds, rain or through the radome.  $l$  is always between zero and one, usually

- 
1.  $\sigma$  depends not only on the size, shape and kind of matter making up the target but also of the radar wavelength. For details, turn to *Rinehart* (1991).
  2. For the most commonly used radar wavelength,  $|K|^2$  for water is around 0.93 and for ice it's 0.20 - this difference surely affects  $p_r$ !
-

---

closer to 1 than to 0.  $l$  is often neglected in the calculations (i.e. it is considered to be equal to 1).  $z$ , finally, is called the *reflectivity factor* and this is calculated as

$$z = \sum D^6 \quad (\text{Eq 10})$$

where the sum is over all the particles in a unit volume and  $D$  is the radius of a particle (e.g. a typical raindrop).

Ignoring  $l$  and approximating  $|K|^2$ , the only unknown parameter in the equation above is the reflectivity factor  $z$ . Solving for  $z$  we get an indication of the characteristics of the precipitation in the area from which the received power  $p_r$  originates. Reflectivity factors, or simply *reflectivities*, can range from as small as 0.001 for fog or weak clouds to as much as 50 000 000 for very heavy hail. It is practical to describe  $z$  using logarithmic units, so a new parameter  $Z$  is introduced to do just that:

$$Z = 10_{10} \log z \quad (\text{Eq 11})$$

This new parameter  $Z$  is measured in units of dBZ<sup>1</sup>, whereas the linear parameter  $z$  is measured in mm<sup>6</sup>/m<sup>3</sup>. This gives us roughly -30 dBZ for fog and 77 dBZ for large and heavy hail. The reflectivity data files that are encountered in RAVE usually contain dBZ values, linearly transformed into the 8 bit [0, 255] interval.

## 2.5 Doppler velocity measurements

In 1842, Christian Doppler discovered that moving objects will shift their frequencies of sound in proportion to their speeds of movement. Weather radars do not use sound waves but electromagnetic waves, but these show the same behaviour: a moving target observed by a stationary weather radar will shift the frequency of the radar signal an amount depending upon its (radial, or parallel to the emitted ray) speed. Not all weather radars have the equipment to perform Doppler velocity calculations, but the 12 Swedish ones do.

For a single target at distance  $r$  from the radar, the radar wave will travel a distance of  $2r$  - to the target and back to the radar. Using

---

1. dBZ stands for “decibels relative to a reflectivity of 1 mm<sup>6</sup>/m<sup>3</sup>”.

---

wavelength  $\lambda$ , this corresponds to  $2r/\lambda$  wavelengths, or, since one wavelength equals  $2\pi$  radians,  $4\pi r/\lambda$  radians. So if the radar signal is transmitted with an initial phase of  $\phi_0$ , the phase of the received signal will be

$$\phi = \phi_0 + \frac{4\pi r}{\lambda} \quad (\text{Eq 12})$$

The time derivative of this phase, i.e. the change of phase from one pulse to the next, is given by

$$\frac{d\phi}{dt} = \frac{4\pi}{\lambda} \cdot \frac{dr}{dt} \quad (\text{Eq 13})$$

This time derivative, or angular frequency, is also equal to  $2\pi f$ . And the radial velocity of the target,  $v$ , is exactly the time derivative of  $r$ , included in Equation 13 above. Thus, the frequency shift caused by a moving target may be written as

$$f = \frac{2v}{\lambda} \quad (\text{Eq 14})$$

where  $v$  is the radial velocity of the object and  $\lambda$ , as before, is the wavelength of the radar waves. A Doppler radar maintains a constant transmitter frequency and phase relationship from one pulse to the next, and exploits Equation 14 in order to calculate radial velocities. It is important to realize just that: it is only *radial* velocities that can be measured using a Doppler radar. Objects travelling at a 90 degree angle to the radar beam will appear to have zero velocity. (*Rinehart*, 1991, pp 73)

## 2.6 Radar data

All weather radars basically operate the same way: First, the antenna is instructed to point at a certain *elevation angle*. This is the angle between the radar ray and the ground at the radar. Then the radar transmits a few pulses at this elevation and with a constant horizontal rotational velocity starting from due north. Having received the echoes from these pulses (if any), the transmitter and antenna proceed clockwise, increasing the *azimuth angle*, or *bearing*, in short steps from 0 to 360 degrees, using the same elevation angle. The azimuth angle is a direction in terms of the 360 degree compass; north is at 0 degrees, east is at 90 degrees, and so on. Then a new elevation angle is

used and the transmitter and antenna loop through the whole azimuth span of 360 degrees. The data received from one such revolution, with the elevation angle held constant, is shown in Figure 3. Here, the colours on the surface correspond to different reflectivities ( $z$  in the radar equation). The upper half of the surface is mostly clear air with few echoes. In this figure, the earth is assumed to be flat – hence the rays bend slightly upwards<sup>1</sup>.

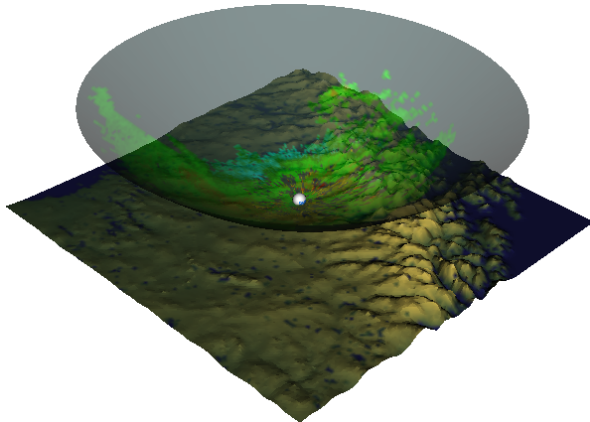
A typical Swedish radar uses 12 different elevation angles, from 0.5 to 40 degrees, and an azimuthal resolution of 0.9 degrees. Operating in non-Doppler mode the radars have a maximum range of 240 km and a resolution of 2 km along each ray. Using radar terminology to describe the radial resolution, we say the radars have 2 km *range bins*. A complete scan thus contains

$$12 \cdot \frac{360}{0.9} \cdot \frac{240}{2} = 576000 \quad (\text{Eq 15})$$

reflectivity samples and this collection of data is called a *polar volume*. It takes almost 15 minutes to collect data for a full polar volume, i.e. when one scan is completed, it is soon time to start a new one

**Figure 3.**

The surface generated by a radar scan when the elevation angle is held constant and the azimuth angle varies from 0 to 360 degrees..



1. The gaseous composition of the atmosphere also affects ray bending (according to Snell's law). See Section 2.7 on page 15.

---

The samples in a polar volume may be arranged in a three-dimensional array to implicitly associate each value with its origin in space:

```
data = float[elevationAngles][azimuthAngles][rangeBins]
```

In order to display this data in a 3D scene as in Figure 3, the (*elevation, azimuth, range bin*) coordinates need to be transformed into a cartesian (*x, y, z*) coordinate system. This transformation should take the ray bending into account, as discussed in the next section.

## 2.7 Range, height, and distance

If there were no atmosphere, a horizontal ray emitted from a radar would travel in a straight line. Since the earth is curved, its height above the earth's surface would increase with the distance from the radar. In other words: it would exhibit a relative curvature with respect to the earth's surface of  $1/R$ , where  $R$  is the radius of the earth:

$$\frac{d\phi}{dS} = \frac{1}{R} \quad (\text{Eq 16})$$

Here,  $\phi$  is the angle between the tangent of the circle arc and some fix axis and  $dS$  is a short distance along the arc.

But the earth does have an atmosphere. Temperature, as well as atmospheric and vapour pressure vary with the height in the atmosphere and this affects the refractive index. These parameters often change abruptly with the height as the atmosphere is divided into relatively distinct layers of different gaseous compositions, hence the refractive index changes with the height. Electromagnetic waves travelling in the atmosphere bend slightly when the refractive index changes, in accordance with Snell's law:

$$\frac{n - dn}{n} = \frac{\sin i}{\sin r} = \frac{u_i}{u_r} \quad (\text{Eq 17})$$

where  $n$  is the refractive index at some point in the atmosphere and  $dn$  is the change of  $n$  over some layer in the atmosphere. The parameters  $i$  and  $r$  are the angles of incidence and refraction, respectively;  $u_i$  and  $u_r$  are the speeds of electromagnetic radiation in the first and second layers, respectively.

For a radar ray travelling in a non-uniform atmosphere, the ray will bend more or less relative to the earth, depending on how much the refractive index changes with height. The curvature of a radar ray relative to the earth's surface is then

$$\frac{d\phi}{dS} = \frac{1}{R} + \frac{dn}{dh} \quad (\text{Eq 18})$$

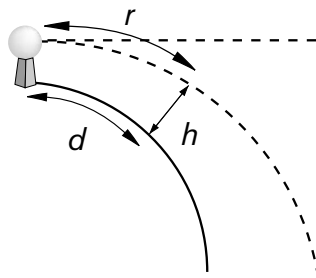
where the last term describes how the refractive index of the atmosphere varies with the height. Under normal circumstances, this derivative is negative - the refractive index is typically around 1.0003 at sea level and 1.0 in outer space. The effect is then that radar rays appear to bend downward and their curvature relative to the surface of the earth is less than  $1/R$ . It is a good approximation to use the following approximation of the curvature (Rinehart, 1991):

$$\frac{1}{R'} = \frac{1}{\frac{4}{3}R} \quad (\text{Eq 19})$$

Using this curvature, the *range*, *height*, and *distance* variables may be defined as shown in the grossly exaggerated Figure 4. The *range* of a certain radar sample is the distance from the radar to the sample, along the ray. The *distance* is the corresponding distance from the radar along the earth's surface, to the point directly beneath the sample. And the *height* of the sample is simply the sample's height above the surface of the earth.

**Figure 4.**

Range, height, and distance.



The range, height, and distance calculations are performed in order to calculate the cartesian coordinates of a sample. If the sample is along a ray with azimuth angle  $\alpha$  and elevation angle  $\epsilon$ , the  $x$ ,  $y$ , and  $z$  coordinates are simply given by:

---

---

$$\begin{aligned}x &= -d \cdot \cos \alpha \\y &= d \cdot \sin \alpha \\z &= h\end{aligned}\tag{Eq 20}$$

This relationship is used when the radar data is transformed into a cartesian coordinate system. This way, the radar's  $x$  and  $y$  coordinates are always  $(0, 0)$ , and its  $z$  coordinate depends on the radar's height above sea level.

---

---



---

# RAVE - current and future use

---

## 3.1 Background

The RAVE project was launched in 1996. The main goal was to develop an application for analysis and visualization of radar data at SMHI, and possibly for its international equivalents in the Baltic region<sup>1</sup>. Some basic work was done as a master's thesis by Håkan Bolin during the first six months of the project. Since then, the system has been extended with various plugins and functions to import data from different radar data formats, etc. But until now, there have been no 3D visualization possibilities, although it was stated from the beginning that there should be.

In this chapter, RAVE's current functionality is presented along with some discussions on the products that are generated, and on a few of the products that are added with this project. Then something is told about the current and future users of RAVE.

## 3.2 System design

The basic system design is shown in Figure 5. Radar data is sent via modems and telephone lines to a database server at SMHI. A RAVE application acts as a client to the database server and can fetch radar data via Internet and the http protocol.

RAVE is supported on Unix and Linux platforms. RAVE could quite easily be ported to the Windows or Macintosh platforms though,

---

1. There is a cooperation between weather radar operators in Sweden, Norway, Denmark, Finland, Germany, and Poland called BALTRAD and operated within the framework of The Baltic Sea Experiment: BALTEX. See *Raschke et al* (2001).

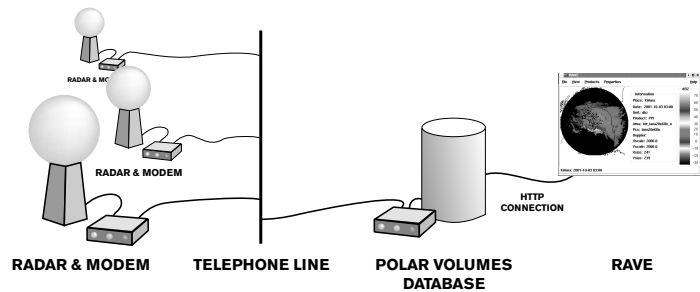
---

since Python is a platform-independent language. A few c modules would have to be recompiled.

As shown in Equation 15, a typical polar volume contains about half a million samples. The samples are all eight bits (i.e. one byte) each. Often, there are a lot of zero values - corresponding to clear air - in the volumes. A simple run-length encoding<sup>1</sup> is employed that typically renders file sizes of around 30 to 250 kilobytes, as compared to the half a megabyte size of an uncompressed file. The file sizes vary with the weather; lots of precipitation implies large file sizes!

**Figure 5.**

The basic system design in RAVE.



### 3.2.1 Graphical user interface

RAVE's main graphical user interface looks as shown in Figure 6 on page 22. There are a few drop-down menus at the top of the main window; the first one is the *File* menu. This menu provides options to open a radar data file from the radar data network database as well as from the local file system. Time sequences of polar volumes can also be opened this way. There are also options to save files using different formats and to quit "raving".

Once a polar volume has been loaded, a product can be generated through the *Products* drop-down menu. The possible products for a single polar volume are PPI, CAPPI and PCAPPI images (described below). Selecting one of these alternatives from the Products menu brings up a dialog where options for the desired product can be set. When this dialog is closed, the product is generated and displayed,

---

1. In a run-length encoding, each sequence of consecutive, equal values is replaced by the value and the sequence length, e.g. 0, 0, 0, 0, 0, 0, 2, 2, 2, 9 would be replaced by (0, 6), (2, 3), (9, 1).

---

occupying the larger part of the window. A new item, *Volume visualization*, is added to the products menu with this project, allowing 3D views of polar volumes. The 3D graphics is displayed in a new window. More on this later!

Using the *View* and *Options* menus, the user can choose what features should be included in the currently displayed window. Options include a background that shows the surrounding land and water, or a vector graphics overlay of coastlines, borders, lakes, and the like. On top of that, the user can select what colour legend to use, and whether the legend should be displayed along with the image or not. Different colour legends map the scalar values (i.e. wind or reflectivity) to different colours.

It is also possible to bring up an information window that displays information (geographical as well as image coordinates and scalar value) about the pixel currently below the mouse pointer.

When a product is generated, it is displayed in the main window as shown in Figure 6. It is possible to zoom in the image by clicking and dragging to create a rectangular region of interest.

In the following, the most common radar products will be presented.

### **3.3 Products from 3D radar data**

In this section, the most common 2D products generated from a single polar volume of radar data is presented. The ones included in RAVE before this project started are the PPI, CAPPI and PCAPPI products. Added with this project are the RHI and VAD products, where the latter is a 3D product from a wind volume. All the others are 2D products.

#### **3.3.1 The PPI product**

Considering radars in general, the PPI scope is by far the most common radar display. PPI is an abbreviation for *plan position indicator*. The PPI is one of the most common products used with weather radars. This product takes all data collected during a 360 degree azimuth scan using the same elevation angle and projects it down onto a plane. The result is a circular area with the radar positioned at the

---

centre of the circle. The radius of the circle depends on the elevation angle; larger elevation angles give smaller circles, and vice versa.

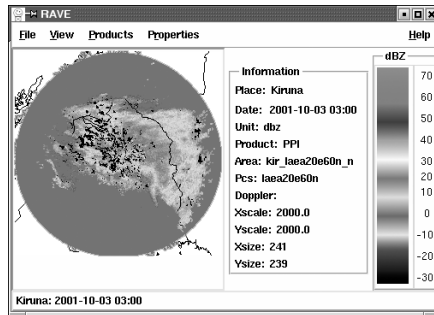
The PPI product is generated as an image with cartesian coordinates where the radar is positioned at the origin. For each  $(x, y)$  coordinate in the image, the polar coordinates are calculated as  $(r, \phi)$ . Here,  $r$  is the distance to the origin and  $\phi$  is the counter-clockwise angle from the positive  $x$  axis to the line drawn from the origin to the  $(x, y)$  point. Then  $r$  is used to determine which one of the typically 120 bins is closest, and  $\phi$  is used to determine which one of the typically 400 azimuth angles is closest. Nearest neighbour or bilinear interpolation may be used to calculate the value to store at position  $(x, y)$  in the output image.

An example PPI image is shown in Figure 6. The *elevation surface object*, discussed in Section 5.4.3 on page 53, adds height information to a PPI image and displays it in 3D.

---

**Figure 6.**

A PPI image and the main graphical user interface in RAVE.



### 3.3.2 The CAPPI product

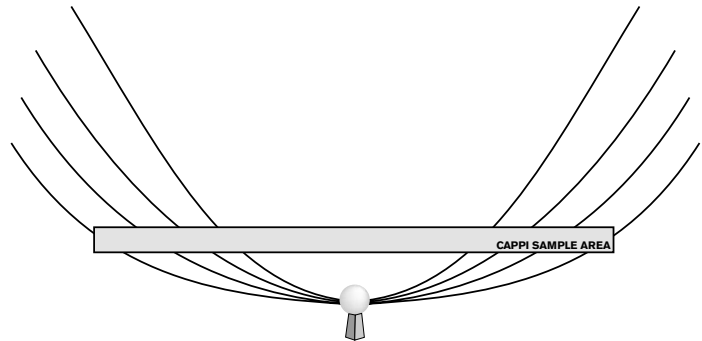
The “CA” in CAPPI stands for *constant altitude*. This is a *constant altitude plan position indicator*. The output from a CAPPI algorithm is a 2D image that is parallel to the surface of the earth; this image has been generated from an input polar volume. In Figure 7, a radar is shown with four of its elevation angles. When a CAPPI image is generated, the samples closest to the chosen altitude are projected down onto the output image plane, typically using nearest neighbour or bilinear interpolation.

---

At lower altitudes, there is data only in the region closest to the centre of the output image (which is straight above the radar), whereas at higher altitudes, there is *no* data available at the centre of the output image. There will be a “hole” with no data at the centre of the image for higher altitudes, and this hole grows with the altitude. This is understood by looking at Figure 7. There is a CAPPI image to the left in Figure 8. For this CAPPI, generated for an altitude of 2 500 m., the maximum range of the data is around 100 km.

**Figure 7.**

Generating a CAPPI image. The output image is orthogonal to the paper.



### 3.3.3 The PCAPPI product

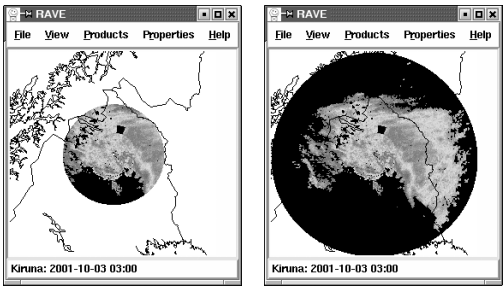
The “p” is for *pseudo*. This product also generates data for some constant altitude, but values are extrapolated to extend the image beyond the left and right boundaries of the CAPPI sample area in Figure 7<sup>1</sup>. The extrapolation is based on data from the lowest elevation, i.e. the closest available data. In Figure 8, all the data in the right image that is absent in the left one has been extrapolated from the lowest elevation angle. The CAPPI image is an exact subpart of the PCAPPI image; more specifically, the centers of a CAPPI and a PCAPPI image from the same height look the same.

It is PCAPPI images from low altitudes, typically around 500 metres, that are sometimes shown in the weather forecasts on Swedish television. The data shown on TV is quantized to around four levels, so a lot of information is stripped away from the eight bit polar volumes before the data enters the Swedish living rooms!

---

1. And beyond the near and far boundaries, not seen in the 2D figure! Remember, in Figure 7, the generated image is orthogonal to the paper.

**Figure 8.** A CAPPI (left) and a PCAPPI (right) image, showing data from the same location, time, and altitude (2 500 metres).



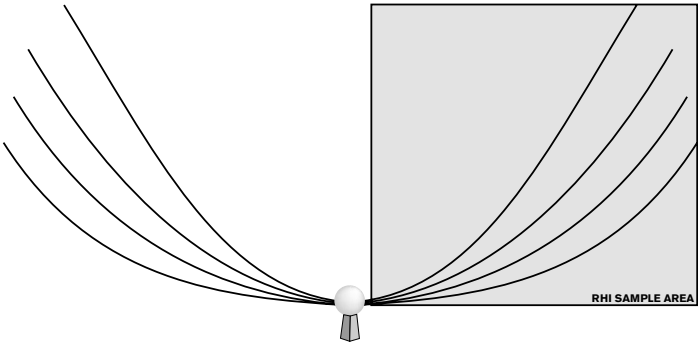
**3.3.4 The RHI product**

RHI is an abbreviation for *range/height indicator*. The RHI product is generated using data from a single azimuth angle and all elevation angles. The resulting image lies in a plane that is orthogonal to the surface of the earth. See Figure 9.

The RHI products provides important information about the altitude and height, or “thickness” of storms. Meteorologists can tell a lot about the current weather situation by looking at a single RHI image!

As with the previously described products, some interpolation method is used in order to fill the pixels in the output image that lie between elevations.

**Figure 9.** Generating a RHI image. The output image is parallel to the paper.



---

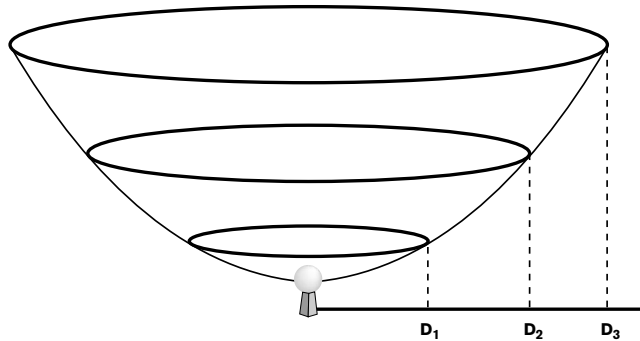
### 3.3.5 The VAD product

This abbreviation is for *velocity/azimuth display*. The VAD technique can be used to calculate the vertical wind profile at the radar site, using data from a Doppler PPI scan. Measuring the Doppler velocity along a circle at constant altitude, assuming the wind field is homogeneous, and plotting the radial wind velocity against the azimuth angle, one gets a sine-function like shape. In Figure 9, three VAD circles have been drawn for the three distances from the radar:  $D_1$ ,  $D_2$ , and  $D_3$ . These three distances obviously correspond to different altitudes and ranges along the radar ray, as discussed in Section 2.7, “Range, height, and distance”, on page 15. Data from one such circle is plotted in Figure 11. In the weather situation depicted in Figure 11, there was no precipitation in the azimuth interval between 60 and 160 degrees, i.e. north-east, east, and south-east from the radar.

---

**Figure 10.**

VAD circles.



The azimuth angle at the maximum of the sine-function corresponds to the horizontal wind direction; the radial wind velocity, measured by the radar, has its maximum value when it is parallel to the actual wind direction. It is important to realise, however, that the radial wind velocity is also affected by the speed at which the reflecting particles are falling. This portion of the radial velocity is independent of the azimuth angle and results in an offset of the sine curve. The horizontal wind velocity is the sine curve's amplitude with respect to the offset.

In Figure 11, the maximum value of a sine curve fitted to the points is in the clear air interval, somewhere around 80 degrees azimuth. As can be expected, the minimum value is about 180 degrees from there,

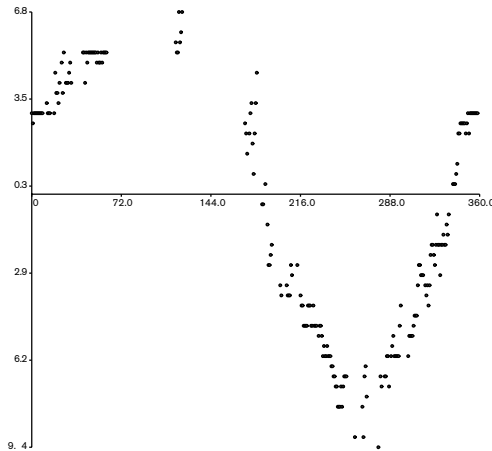
---

at some 260 degrees. This means that the winds are blowing from around east direction (ENE). The sine curve's vertical offset from zero is around -2.5 m/s, so the horizontal wind speed is around 7 m/s.

---

**Figure 11.**

Velocity/azimuth display of the Doppler velocity along a VAD circle.



The input to a VAD algorithm is data from one or several (with different elevation angles) Doppler PPI scans and the desired altitudes at which VAD circles should be generated. The output is either a graph like the one in Figure 11 or a set of horizontal vectors, one for each of the desired altitudes. Such an algorithm was already implemented in RAVE before the start of this project, but the only possible way to view the output from the algorithm was as lists of numbers. A 3D visualization object employing this algorithm has been added (see Figure 29).

### 3.4 RAVE users

RAVE, in its current form, is not being used in the daily forecast production at SMHI. There are other tools to view simple radar data products, possibly together with colocated satellite images, that are used instead. RAVE is more of a post storm analysis tool; researchers at SMHI can examine exceptional weather situations in greater detail using RAVE.



---

Built upon Python, it is possible to call every RAVE method and function by issuing commands in a Python interpreter. New algorithms can easily be added and tested and this openness is one of RAVE's greatest strengths. RAVE is a superior tool to perform research within weather radar analysis and visualization.

At the end of this project, the application will be installed at Arlanda airport outside Stockholm for evaluation. Meteorologists at airports make great use of weather radar data. It is one of their most important decision-making tools.

Meteorologists are used to viewing and analyzing 2D images like PCAPPI and RHI products. It is thus very important to include these types of objects in the 3D version of RAVE, as they may serve to bridge the gap between 2D and 3D visualizations. Having spoken to meteorologists at SMHI, their main opinion is that it will probably be of great use to them to be able to analyze radar data in 3D. As with all new technologies it will, however, take some time for them to get used to it. It is of great importance that the user interface be designed for easy use.

---

---

# Visualization of volumetric data

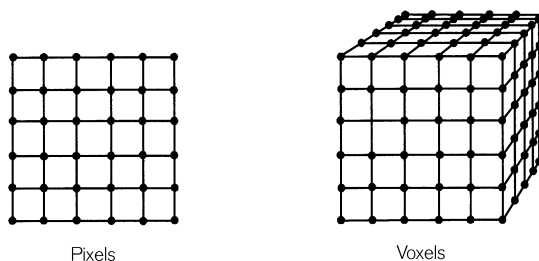
## 4.1 Introduction

This chapter aims to explain and compare the three main classes of methods for visualization of volumetric data: slicing, surface rendering and volume rendering techniques.

A digital image consists of a two-dimensional array of data elements. These represent colour or light intensity and are referred to as *pixels*, which is short for picture elements. Similarly, a volume of data is a three-dimensional array of data elements which possibly is the result of a radar scanning and sampling some volume of air. The acquired values are called *voxels*, or *volume elements*. A voxel is defined as a point without a size in three-dimensional space<sup>1</sup>. The locations of the voxels are usually confined to specific regular spacing on a rectangular grid, as in Figure 12, but this is not always the case.

**Figure 12.**

Pixels and voxels.



1. Another common definition is to consider a voxel to be a cube with some small size

---

## 4.2 Polar and cartesian volumes

In this chapter, it is assumed that the data is stored in a cartesian coordinate system, forming a rectilinear grid of data. However, as seen in Chapter 2, the nature of radar data is originally not cartesian but polar. Some products may be created directly from the polar volumes, e.g. vertical cross-sectional planes are easily created using all available data with one azimuth angle (see Section 3.3.4, “The RHI product”, on page 24, and Section 5.4.4, “The RHI object”, on page 54). Some products, e.g. isosurfaces, require that data be resampled into a cartesian grid first, though.

### 4.2.1 Gaussian splatting

One method to create a cartesian grid of data out of a dataset with another structure – completely unorganised or with some polar structure, for instance – is called *Gaussian splatting*. In this method, the input data points are “splatted” into the output dataset using some Gaussian ellipsoid as a weighting function. First, the dimensions of the output cartesian grid are set. Then, for each input point, its position in the output grid is calculated. The 3D Gaussian kernel is used as the weighting function when the input point’s value is distributed to the points in the output dataset that are closest to the calculated, exact output point. (*Schroeder et al*, 2001)

The most important parameter to the algorithm is the width of the Gaussian kernel. This kernel is precalculated; the same kernel is used for all input points.

### 4.2.2 Shepard’s method

As in the Gaussian splatting algorithm above, the first step is to decide the dimensions of the output dataset. For each point in the output dataset, the closest input points are weighted together to form the value that is stored in the output point. The weighting depends on the inverse distance to the input points; i.e. closer input points contribute more than points that are farther away from the current output point. The equation used is as follows:

$$F(x, y, z) = \sum_{i=1}^n w_i f_i \quad (\text{Eq 21})$$

---

where  $n$  is the number of input points,  $f_i$  are the values at the input points, and  $w_i$  are the weight functions assigned to each input point. One common way to define the latter is:

$$w_i = \frac{h_i^{-p}}{\sum_{j=1} h_j^{-p}} \quad (\text{Eq 22})$$

where  $p$  is a positive number called the *power parameter*. The larger the power parameter, the less input points far away from the current output point will affect it. Finally,  $h_i$  is the distance from input point  $i$  to the current output point, or

$$h_i = \sqrt{(x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2} \quad (\text{Eq 23})$$

(*Environmental Modeling Systems, Inc.: Inverse Distance Weighted Interpolation*)

### 4.3 Different approaches

The three basic options for displaying a three-dimensional data set on a two-dimensional computer screen are (Watt, 1999):

1. To slice the data set with a *cross-sectional plane*. This is the easiest option with a quite straightforward solution if the plane normal is parallel to one of the coordinate axes of the volume data set.
2. To extract an object that is “known” to exist within the data set and convert its surface into polygons, which are typically rendered by a hardware accelerated 3D renderer. If the whole data set is the torso of a human body then the liver may be extracted and displayed in isolation. The original volume data needs to be segmented first. This process is an example of *surface rendering*.
3. To assign transparency and/or colour to voxels in the data set, and then view the entire set from any angle. This is what is usually known as *volume rendering*.

The first option is already implemented in RAVE, but only for planes parallel to the earth’s surface. This and the two other options will soon be explained further, but first we need to know how to calculate the data value at an arbitrary point in the volume by using interpolation. This is useful not only during slicing, surface extraction or vol-

---

ume rendering, but also when a polar data volume should be resampled into a cartesian data volume, e.g. using Gaussian splatting or Shepard's method.

## 4.4 Interpolation

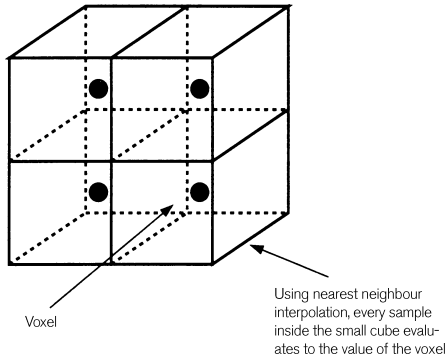
Voxels are usually addressed using three integer coordinates:  $x$ ,  $y$ , and  $z$ . But what if one needs to know the data value at, say, (8.8, 4.1, 4.0)?

### 4.4.1 Nearest neighbour interpolation

The easiest solution to the aforementioned problem is to simply use the value at the nearest voxel - in our case the value at position (9, 4, 4). This method is called *nearest neighbour* or *zero-order interpolation*. Using this method, there is a region of constant value around each sample in the volume, as seen in Figure 13. This interpolation scheme is the least time consuming, but also the least accurate.

Figure 13.

Nearest neighbour interpolation.



### 4.4.2 Trilinear interpolation

When *trilinear interpolation* is used, the data value is assumed to vary linearly between voxels along directions parallel to the major axes.

Figure 14 illustrates this. The point  $P$  has coordinates  $(x, y, z)$  within the cube cell defined by voxels  $A$  through  $H$ . Voxel  $A$  has coordinates (0, 0, 0) and a value of  $V_A$  and voxel  $H$  has coordinates (1, 1, 1) and a value of  $V_H$ . The value of  $V_P$  calculated using trilinear interpolation is then:

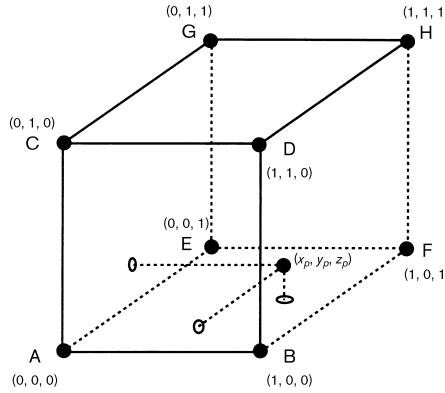
$$\begin{aligned}
V_P = & V_A \cdot (1-x)(1-y)(1-z) + V_B \cdot x(1-y)(1-z) + V_C \cdot (1-x)y(1-z) \\
& + V_D \cdot xy(1-z) + V_E \cdot (1-x)(1-y)z + V_F \cdot x(1-y)z + V_G \cdot (1-x)yz \\
& + V_H \cdot xyz
\end{aligned} \quad (\text{Eq 24})$$

In general,  $A$  will be at some location  $(x_A, y_A, z_A)$  and  $H$  will be at  $(x_p, y_p, z_p)$ . In this case,  $x$  in Equation 24 above would be replaced by  $(x - x_A)/(x_H - x_A)$  with similar substitutions made for  $y$  and  $z$ .

Trilinear interpolation is obviously more time consuming than is nearest neighbour, but it gives more accurate results. Even more accurate interpolation methods may be used<sup>1</sup> but the computational price is usually too high - especially when interpolation calculations should be performed for millions of voxels. Most of the times, trilinear interpolation gives good enough results.

**Figure 14.**

Trilinear interpolation.



## 4.5 Slicing techniques

A cross-sectional plane can be extracted from the volume data set. This operation is known as *slicing*. When the voxels are confined to a regularly spaced grid, the plane normal is parallel to one of the coordinate axes and the plane's offset from the origin is an integer, all that has to be done is to form an image using the voxels in the plane. When the plane's offset from the origin is not an integer, linear interpola-

1. Interpolation may be regarded as convolution using a kernel, where the optimal "kernel" is the infinite sinc function. Tricubic convolution is a common method. Here, cubic splines in three dimensions are used to approximate the sinc function. See *Danielsson et al* (2000).

---

tion may be used in order to calculate the values in the voxels. This technique may also be used when the plane normal is not parallel to one of the coordinate axes.

Slicing techniques effectively only display two dimensions of the data. However, this reduction from 3D to 2D may lead to images that are easily interpreted. And when a 3D environment is available, planes with different normals and offsets can be viewed simultaneously. If the user is allowed to adjust the parameters of the planes and to view the scene from arbitrary positions, the result may be some very revealing visualizations.

## 4.6 Surface rendering techniques

Surface extraction, in the case of visualizing weather radar data, would be to identify separate cloud formations and for each cloud create a computer graphics object to represent the cloud. These objects would be hollow with surfaces made of polygons (triangles).<sup>1</sup>

When volumetric data is visualized using a surface rendering technique, a dimension of information is essentially thrown away. However, the resulting images may be very useful anyway, and these are generally rendered faster than if volume rendering would be used<sup>2</sup>.

A surface can be defined by applying a binary segmentation function  $B(v)$  to the volumetric data.  $B(v)$  evaluates to 1 if the value  $v$  is considered part of the object, and 0 if the value  $v$  is part of the background. The surface is then the region where  $B(v)$  changes from 0 to 1. If no interpolation is used, this results in a surface consisting of all the rectangular faces between voxels with differing values of  $B(v)$ .

### 4.6.1 The marching cubes algorithm

An *isosurface* is the 3D surface representing the locations of a constant scalar value within a data volume. Every point on the surface has the same constant value; this value is the *isovalue*. The *marching cubes algorithm* can be used to generate isosurfaces from a volume. It examines each volume element of a chosen (small) size in the volume and deter-

---

1. Please note, however, that natural clouds usually do not exhibit as sharp edges as this approach may produce!  
2. Especially when polygon rendering is accelerated by dedicated graphics hardware.



---

mines, from the arrangement of vertex values above or below the isovalue, what the topology of an isosurface passing through this element would be. The small volume elements are all cubes of the same size and may contain one or more voxels. The values at the vertices (the corners of the cubes) are calculated using some interpolation method.

Once the values at each vertex of an element have been calculated, the second step of the algorithm is to look at each vertex value and determine if its scalar value is higher or lower than the isovalue of interest. Each vertex is then assigned a binary value - e.g., 0 if value is lower, 1 if value is higher than the isovalue. In this case, the binary segmentation function  $B(v)$  is a thresholding function where the isovalue is the threshold. If the eight vertices of an element are all classified as ones or zeros, the element will have no isosurface passing through it - it will either be completely inside or completely outside the surface. The algorithm then moves on to the next element. Since the eight vertices of each element are all assigned binary values, each element can be classified as belonging to one of  $2^8 = 256$  cases. A table is predefined to contain, for every one of these 256 cases, (a) how many triangles will make up the isosurface segment passing through the cube, and (b) which edges of the cube contain the triangle vertices, and in what order. This is illustrated in Figure 15: here, the black dots represent vertices that have been assigned 1 in the first step of the algorithm. The triangles that constitute the isosurface in the volume element are shown. The coordinates for the vertices of these triangles are calculated using linear interpolation; the value at every vertex should be exactly the sought-for isovalue. By symmetry and rotational symmetry the 256 possible cases reduce to the 15 shown in the figure.

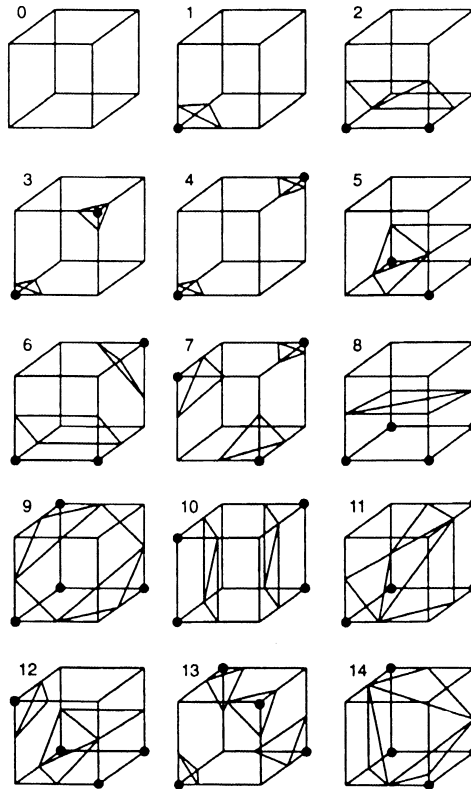
The marching cubes algorithm represents a high-speed technique for generating 3D isosurfaces. Part of the elegance and simplicity of the algorithm lies in its ability to generate surface polygons within each single volume element. The surface generation is almost completely failureproof<sup>1</sup>. (*Gallagher, 1995*).

---

1. There are, however, a few exceptional situations in which the isosurface polygons are discontinuous across two adjacent volume elements

**Figure 15.**

The marching cubes.



## 4.7 Volume rendering techniques

Representing a surface contained in volumetric data (as described above) can be useful in many applications, including visualization of radar data, but there are a few major drawbacks. First, the geometric primitives used - mostly triangles - can only approximate the actual surface in the original data. To make a good approximation, a lot of triangles need be used. The rendering complexity and memory requirements increase rapidly with the increased resolution. Second, much of the information in the original data is lost when hollow shells are created out of “solid” data volumes. Third, amorphous phenomena such as fire, fog or clouds cannot really be adequately represented using surfaces.

*Volume rendering* is an alternative approach which solves the mentioned problems. Transparency and colour values are assigned to all voxels in the data set based on their scalar values, and then the entire set may be viewed from any angle. Volume rendering can basically be achieved using an *object-order technique* or an *image-order technique*.

Object-order volume rendering techniques use a forward mapping scheme where the volume data is first transformed according to the desired rotation and then mapped onto the image plane. Image-order techniques, on the other hand, use a backward mapping where the image plane is transformed while the volumetric data is stationary. An imaginary ray is cast from each pixel in the image plane into the volume data. Figure 16 and Figure 17 illustrate the difference.

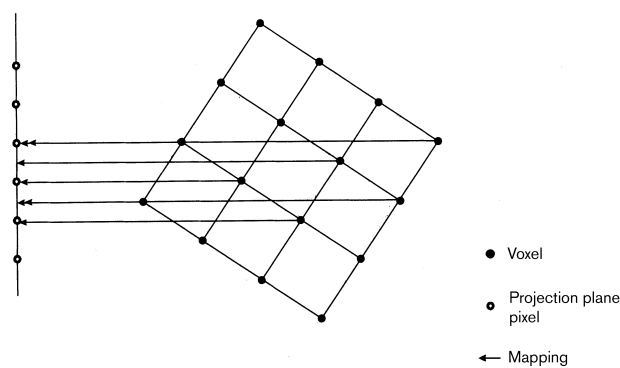
**4.7.1 Object-order techniques**

Using an object-order technique, the voxels are to be projected onto the image plane. If two voxels project to the same pixel, the one that was projected later will prevail. The most intuitive solution is to loop through and project the voxels in a back-to-front order. This way, the voxels closer to the image plane will also appear closer to the image plane since they are drawn “on top of”, erasing, the voxels behind them.

The volume is transformed before rendering, so that the  $z$  axis is perpendicular to the  $(x, y)$  image plane. Voxels far away from the image plane have large  $z$  values, while voxels closer to the image plane have smaller  $z$  values. This transformation is computationally expensive.

**Figure 16.**

Object-order volume rendering (forward mapping). The volume is transformed and mapped into image space.



---

The voxels can be looped through in a front-to-back order using a so-called *Z-buffer* with as many data elements as there are pixels. The first voxel is projected to pixel  $(x, y)$  in the image plane. At the same time, the voxel's distance to the image plane is stored at position  $(x, y)$  in the *Z-buffer*. If another voxel projects to position  $(x, y)$ , by looking in the *Z-buffer* it is possible to determine that there is no need to process it further and draw it since it would be hidden by the first voxel. With a front-to-back method, once a pixel value is set, its value remains unchanged.

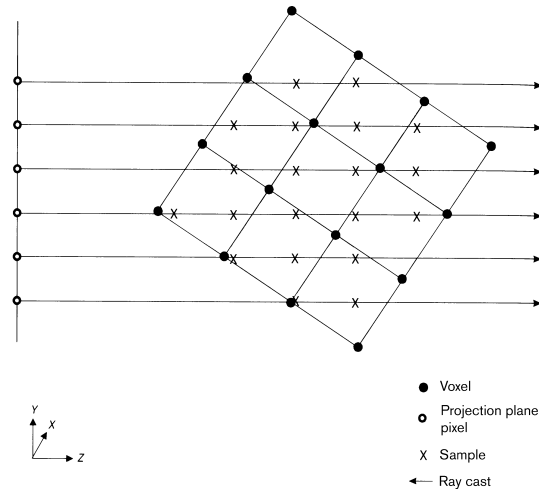
Clipping planes parallel to the image plane and clipping planes orthogonal to the three major axes may easily be added when back-to-front or front-to-back algorithms are used. Voxels beyond the clipping planes are simply ignored when traversing the volume. This way, different parts of the data set can be explored.

#### **4.7.2 Image-order techniques**

Image-order volume rendering techniques are fundamentally different from the object-order techniques described above. Instead of determining how a voxel affects the pixels in the image plane, in an image-order technique, for each pixel it is calculated which voxels contribute to it. This is done by casting an imaginary ray from each pixel into the data volume. Samples are taken (at regular intervals) along the ray. The contributions from the samples are weighted together, possibly taking transparency, colour and distance to the image plane into account, into a value that is stored in the pixel from which the ray originates.

**Figure 17.**

Image-order volume rendering (inverse mapping). The image plane is transformed and mapped onto the volume.



### Binary ray casting

One of the first image-order volume rendering techniques, called binary ray casting<sup>1</sup>, was developed on a machine with only 32 kilobytes of RAM. It was developed to generate images of surfaces contained within binary data volumes without the need to explicitly perform boundary detection and hidden-surface removal. In order to provide all possible views, the volumetric data is kept in a fixed position while the image plane is allowed to move; if the volume should be viewed from another direction, the image plane is moved accordingly. For each pixel in the image plane, a ray is sent from that pixel and the algorithm determines if it intersects the surface contained within the data. The projection may be parallel or with some perspective. A two-dimensional example of perspective projection is shown in Figure 18. To determine the first intersection along a ray, samples are taken with regular intervals. This algorithm works with binary data volumes, i.e. a value of 0 corresponds to background or “nothing” and a value of 1 corresponds to the object. If an intersection occurs,

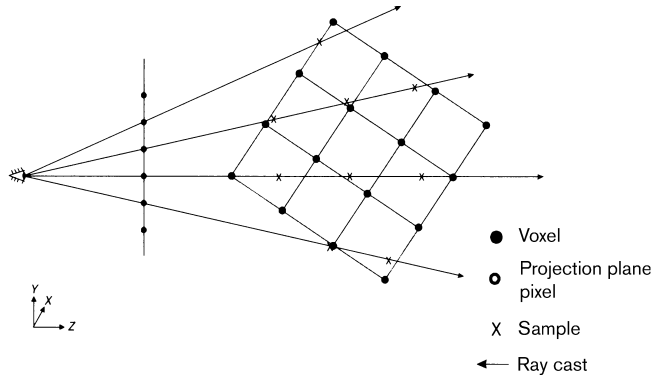
1. Please note that this is not the same thing as *ray tracing*, even though the two techniques are similar. In ray tracing, each ray is followed when it bounces on shiny objects. Very realistic scenes can be rendered using this technology.

---

shading calculations are performed (see Section 4.8 on page 41) and the resulting colour is given to the pixel from which the ray originates.

---

**Figure 18.** Perspective ray casting.



#### **Modern ray casting**

The computers of today typically have 256 megabytes of RAM memory and hard drives to store several gigabytes of data. The display units are able to display 24-bit colour at high resolutions and in one second, a typical processor can perform some billion floating point operations. Thus, it is no longer necessary to confine to visualizing small, binary data volumes<sup>1</sup>. Modern algorithms for ray casting basically work the same way as the one above: rays are cast from every pixel in the image plane and into the volumetric data. Parallel or perspective projection may be used. The main difference is the way that the pixel colour is calculated from several (if not all) voxels encountered along the ray. A few alternative approaches follow.

We could define a *threshold value* and stop at the first intersection with a voxel whose value is greater than the threshold. This value, possibly together with some shading technique, is used to colour the pixel. This approach is similar to the binary ray casting above. If two thresholds are used (a lower and an upper), surfaces similar to the iso-surfaces produced by the marching cubes algorithm can be generated.

---

1. Still, computer hardware performance is the major bottleneck of volume rendering as the data sets to be visualized tend to grow.

---

Another approach is to follow the ray through the whole data volume, storing in the image plane pixel the maximum value encountered along the ray. This is called *maximum intensity projection* (MIP) and by using this technique, internal features of the data may be revealed that would otherwise be hard to see.

Yet another approach is to store the *sum of the values* encountered along the ray. This is essentially how X-rays work. Or the *average of the values* along the ray could be calculated and stored at the current pixel.

The most generic technique involves defining an opacity and colour for each scalar value, then accumulating intensity along the ray according to some compositing function. The compositing function may take not only opacities and colours of the samples into account, but also the distance from the samples to the image plane.

## 4.8 Shading

When viewing any real object, a viewer subconsciously uses the information provided by shades on the object in order to understand how the object is shaped. This effect can be used with good results in 3D visualizations on a computer screen; the shades help the user a great deal in understanding the different shapes and relations of objects.

There could be one or several light sources in a 3D graphics scene. Volume rendering algorithms may employ different illumination models to calculate lighting effects in order to make the data more intuitively understandable.

The Z-buffer may be used for shading - pixels that correspond to voxels far away from the image plane can be darkened, for example. This technique is called *depth cueing*. More accurate shades are obtained by passing the Z-buffer, which essentially is a 2D image, to a gradient-shader. The gradient at each  $(x, y)$  pixel location is evaluated:

$$\nabla z = \begin{bmatrix} \frac{\partial z}{\partial x} \\ \frac{\partial z}{\partial y} \\ 1 \end{bmatrix} \quad (\text{Eq 25})$$

---

This estimates surface orientations; the gradient is parallel to the surface normal. The partial derivatives can be approximated using backward, forward or central differences; here is a central difference approximation of the first component above:

$$\frac{\partial z}{\partial x} \approx \frac{D(x+1, y) - D(x-1, y)}{2} \quad (\text{Eq 26})$$

$D(x, y)$  is the depth associated with the pixel at  $(x, y)$ . When gradients, approximating surface normals, have been calculated for all pixels in the output image, the gradient shader takes these and the distance(s) from the light source(s) into account to produce the final, shaded image. Angles between the normals and vectors from the surfaces to the light source(s) affect the output shade. (*Gallagher, 1995, pp 178*)

#### 4.8.1 Polygon shading

A typical basic illumination model, used to calculate light intensities, should be fast and deliver reasonably accurate results. The lighting calculations are typically based on the optical properties of the surface, the background (ambient) lighting, and the positions, directions, colours, and intensities of the light sources.

In order to make sure that surfaces not exposed directly to a light source are still visible, a background or ambient light is defined for every 3D scene. This light has no spatial or directional characteristics. Thus, the amount of ambient light incident on each surface is equal for all surfaces.

On top of this ambient lighting, point sources of light contribute to the light intensity of a surface. This contribution depends on the angle between the surface normal,  $\mathbf{N}$ , and a normalized vector from a point on the surface to the light source,  $\mathbf{L}$ . Calling this angle  $\theta$ , the intensity contribution from a point source of light is written as

$$I = k_d I_l \cos \theta \quad (\text{Eq 27})$$

where  $k_d$  is a parameter between 0 and 1 describing the surface's shininess, and  $I_l$  is the intensity of the light source. A surface is illuminated by a point source only if the angle of incidence,  $\theta$ , is in the range 0 to 90 degrees (i.e.  $\cos \theta$  is between 0 and 1).



---

Using an illumination model as the one described briefly above, with a few extensions, light intensities can be calculated for each vertex in every triangle. Then, the light intensity can be linearly interpolated over the triangle in order to make the shading appear softer, especially for curved surfaces that have been approximated with polygons. This technique is known as *Gouraud shading*, and it can be performed by dedicated hardware in the 3D graphics accelerator.

Even better-looking shading can be achieved using *Phong shading*. In this case, the surface normals are linearly interpolated over the triangle, the  $\theta$  angle is calculated for each normal, and the expression in Equation 27 is evaluated.

Details on polygon shading can be found in any book dealing with 3D graphics, see for instance *Hearn & Baker* (1997).

---

---

---

# Implementation

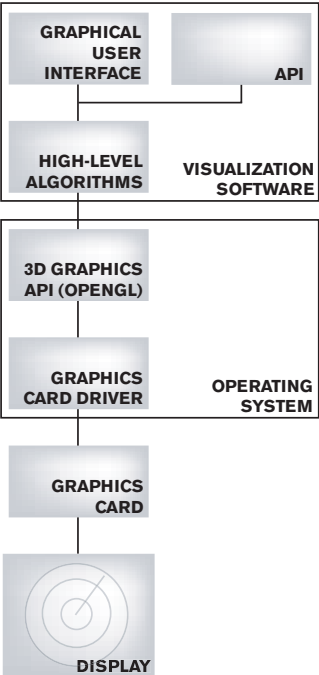
---

## 5.1 Visualization software

The last few decades computer power has developed at an incredible pace; a decade ago, 3D scientific visualization was something that could only be accomplished using super computers. Nowadays the same task is feasible using only a standard PC - even the cheapest graphics cards contain some 3D acceleration hardware. This evolution has lead to the development of quite a few visualization software environments. Some of the most popular of those will be described briefly and their eligibility for incorporation into RAVE will be discussed. Before that, we take a look at the structural units of a 3D visualization system.

A typical 3D visualization system is shown in Figure 19. Starting with the two bottom boxes, images are to be displayed on a display unit connected to the graphics card on a computer. The graphics card contains hardware to render 2D and 3D graphics. The 3D graphics may contain light sources and shaded lines and polygons with different opacities and reflection properties. Rendering commands to the hardware are dispatched from the graphics card driver, which is part of the operating system. This, in turn, receives its commands through a graphics API, such as OpenGL, DirectX or QuickDraw. Such an API serves as a convenience layer for the programmer, with relatively highlevel operations that can be used in overlying applications, such as OpenDX, AVS/Express or IRIS Explorer. These contain implementations of even higher-level graphics operations, such as the marching cubes algorithm or different ray casting algorithms, as described in Chapter 4. Finally, as a front end to the user there is either a graphical user interface, or an application programming interface, or both, as shown by the top boxes. Applications (e.g. radar data visualization software) are built using this front end.

**Figure 19.** Block diagram of a typical visualization system.



**5.1.1 Visual programming systems**

There are several visual programming systems available. In visual programming systems, a data-flow approach to visualization and analysis is used. Data-flow is a way of decomposing data processing tasks into a sequence of self-contained steps, which are implemented as modules. The data-flow is directed by connecting the modules together, hereby creating *visualization pipelines*. Each module's parameters can be adjusted through a mouse-driven graphical user interface. There are modules for manipulating, transforming, processing, rendering and animating data. Most modules have one or more input pipes and one or more output pipes. The pipeline typically starts with a module that can read some data from a file. The following modules alter the data in different ways. The last module in the pipeline may be one that writes the modified data to a file or sends drawing commands to the underlying 3D graphics API.

---

Using a visual programming system, little or no programming knowledge is needed to create advanced visualizations of complex processes. A screenshot from IRIS Explorer, which is representative for this category of visualization systems, is shown in Figure 20.

#### **AVS/Express**

This is a software environment for visualization developed by Advanced Visual Systems, avs. Development and research started in 1988. Avs claims that more commercial applications have been developed with Express than with any other visualization tool. Express includes a comprehensive set of visualization modules. Express is highly modular, hierarchical and extensible - it is possible to write custom modules in c++ and include them in the system.

In its standard version, Express contains more than 850 visualization objects. Visualization of 3D data volumes can be made using ray casting techniques, surface extraction techniques, or both, and geographical data can be added easily. Express from avs can certainly be used for visualization of data from Doppler radar systems. It runs on Linux, Unix and Windows platforms but it is a very expensive system that has no bindings to Python, the programming language that RAVE is built upon. (*Advanced Visual Systems: Software*)

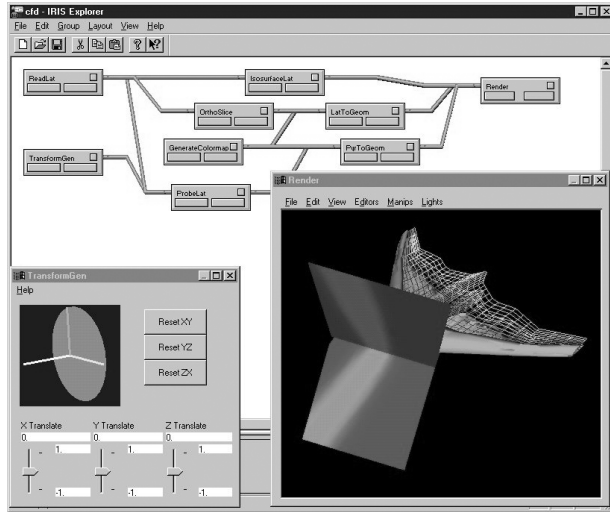
#### **IRIS Explorer**

IRIS Explorer was originally developed and distributed by Silicon Graphics, Inc. (SGI). Interest in the system grew over the years and SGI decided to move IRIS Explorer to an external independent software house. In 1994 the Numerical Algorithms Group (NAG) took over the development, porting, marketing, sales and support. IRIS Explorer runs on Linux, Unix and Windows platforms. Just like avs, it is an extendable but expensive system that lacks support for Python. (*IEC - Iris Explorer Center*)

Figure 20 shows a screenshot from an IRIS Explorer project.

**Figure 20.**

IRIS Explorer user interface.



### **IDL, The Interactive Data Language**

IDL is another system whose main difference from the two described above is that more of the development is textual programming, using a special high-level scripting language. There is a GUI but this is not as heavily relied upon as in the previous two systems. IDL includes routines for rendering isosurfaces and for volume rendering. It runs on many platforms, including different versions of Unix, Linux and Windows. The software is not in the public domain. (*IDL - The Interactive Data Language*)

### **OpenDX, Open Visualization Data Explorer**

OpenDX is the open source version of IBM's Visualization Data Explorer product. OpenDX is based on the last release of Visualization Data Explorer, set up to reflect an open source project. There is a comprehensive GUI with a lot of different modules that can be connected to form a visualization pipeline, as in IRIS Explorer or AVS/Express. Own modules can also be programmed in c++ and added to the pipeline. As in many other open source projects, there is an extensive and enthusiastic user community (*Open Visualization Data Explorer*). OpenDX is a free, open source product with high quality and great functionality. Randal Hopper has developed Python bindings for OpenDX, but only basic control is supported; DX executives

---

can be started up and communicated with using Python (*Py-OpenDX*).

#### **VIS5D**

This free package supports rendering of higher dimension data sets. Isosurfaces, volume renderings, coloured slices etc. are possible operations that can be performed on these data sets. The software runs on several platforms and was originally developed to aid in the visualization of numerical simulations of the earth's atmosphere and oceans. There is no support for Python. (*Vis5D Home Page*)

### **5.1.2 The Visualization Toolkit, VTK**

The Visualization Toolkit (VTK) is a free, open source, software system for 2D and 3D image processing, computer graphics, and visualization. VTK was originally developed as part of a book with the same name. VTK consists of a C++ class library and interpreted interface layers for Tcl/Tk, Java, and Python. VTK supports scalar, vector, tensor, texture, and volumetric methods for visualization and data manipulation operations such as polygon reduction, mesh smoothing, cutting, contouring, Delaunay triangulation, etc. VTK basically works as the systems mentioned above: a visualization pipeline is created by connecting different modules to each other, but the top box in Figure 19 is absent - there is no graphical user interface. Instead, modules are created as objects, typically with `SetInput()` and `GetOutput()` methods.

#### **5.1.3 3D graphics APIs**

Another possible approach is to skip a software layer and use a 3D graphics API such as OpenGL or DirectX directly. This, however, would greatly increase development time as the wheel would need to be reinvented many times, so to speak - several algorithms needed are already implemented and tested in all the available visualization packages. This approach is not very appealing!

## **5.2 Choice of software**

The Visualization Toolkit is free, contains a few hundred well implemented, well documented and tested modules and has working Python bindings. The whole package is open source, under constant development, and it is certainly possible to add custom modules as needed. There is an extensive user community with people using VTK

in a vast range of different fields. Not one of the other packages described above or known to the author has all these advantages. The choice seems simple.

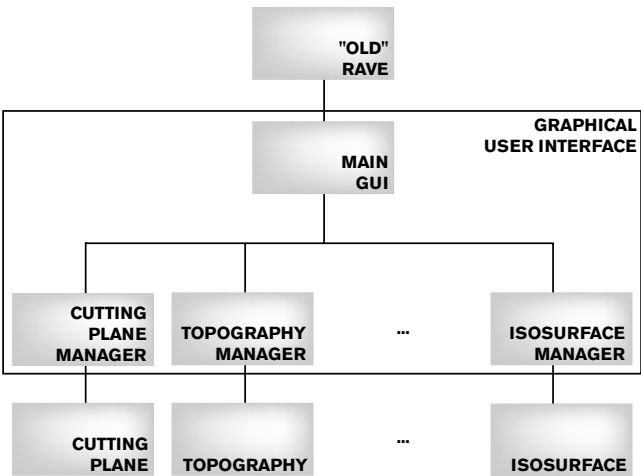
### 5.3 Software design

As discussed in Chapter 4, several different approaches are possible when volumetric data, such as radar data, should be visualized. Iso-surfaces, cutting planes and vAD profiles (see Chapter 3) are examples of objects that should all be possible to include in a visualization scene - simultaneously. The user should be able to add, manipulate and remove objects through a graphical interface.

Python is a language that supports object-oriented programming. By using object-oriented techniques, the basic design is quite straightforward. It is shown in Figure 21.

Figure 21.

Basic system design.



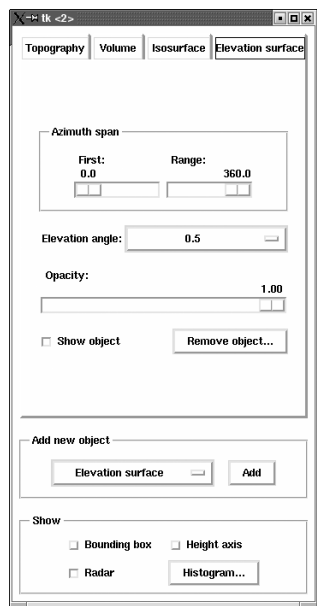
There should be a main graphical user interface class. This is basically a window containing the necessary controls to add new objects and to adjust global properties (e.g. whether to display height axis, radar etc.). This class, called `raveVtkMain`, is also responsible for communication with the “old RAVE”, shown at the top of Figure 21. Polar data volumes will be opened as before, through existing interfaces in RAVE. On top of that, it generates a few basic VTK objects from the



polar volume. These are used by the visualization objects described below.

For each new object, a tab is created in the main window in which object specific properties, such as colour, opacity, or elevation angle, can be altered through different buttons, menus and sliders. These GUI components along with event handlers are encapsulated in special manager classes - one class for each kind of object. Each manager object also has a reference to an actual object, e.g. an isosurface or a cutting plane. These objects in turn have references to `vtk` objects and contain object specific methods (e.g. `getIsovalue` in an isosurface object) used by the manager, typically in response to mouse events. The `vtk` objects are collected in a rendering window (not shown in Figure 21). A screenshot from the main GUI is shown in Figure 22. The GUI is created through a Python interface, called *Tkinter*, to a GUI package called *Tk*. In the following, the different objects are described briefly.

**Figure 22.** Main GUI screenshot when a few objects have been added.



---

## 5.4 Visualization objects

All visualization object classes are named `raveVtkObjectName`, e.g. `raveVtkIsosurface`, and they all inherit some basic properties and methods from an abstract class simply called `raveVtkObject`. This class includes methods to get and set object description, position, visibility, opacity, and colouring, among a few other generic operations. All objects get their input from the single `raveVtkMain` instance.

### 5.4.1 The topography object

Precipitation is heavily affected by the underlying terrain. It is thus important that data from a weather radar can be related to the topography below it.

An *elevation map* is a two-dimensional image picturing some geographic region where each pixel value is the (average) height above sea level in the corresponding geographic region. Elevation maps have been created for all SMHI radars. These maps are squares, each side being twice the radar range in length (i.e. 480 km) and with the radar positioned at the centre of the map. Using an elevation map together with the `vtkWarpScalar` module, a so-called *carpet plot* can be generated. Polygons are generated so that points with higher scalar values are raised more than are points with lower values, thus creating a three-dimensional terrain map.

For each radar there is also a binary land/water map with the same size and resolution as the elevation map. This map is used to colour the three-dimensional surface blue where there is water. Where there is land, the colour depends on the height above sea level.

In the topography manager, there are controls to change the topography's colouring ("default", which is a green-yellow-white colour scale, or grayscale; in both cases colour varies with the height above sea level), to set its opacity and to show or hide it. There can be only one topography object in a scene. The topography object for the surroundings of the Kiruna radar is shown in Figure 23, together with three other objects presented below.

### 5.4.2 The radar, bounding box and axis objects

These three objects have no manager classes. Their visibility can be controlled through checkboxes in the main GUI, and as in the case of the topography, there can be only one of these objects at a time.

---

The radar object is simply a model of a radome with a foundation, shown at the position of the real radar.

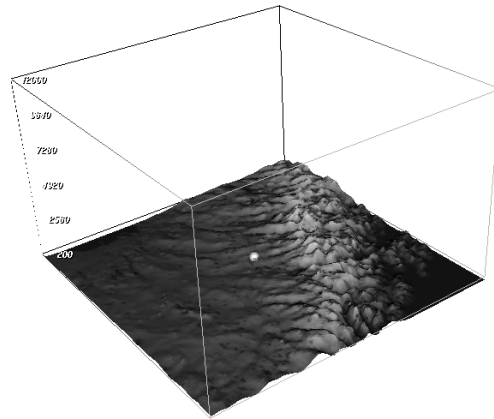
The bounding box makes it easier to understand the position of the radar data when rotating, zooming and panning.

The height axis provides important height information. This object has been designed so that arbitrary axes can be added to the scene, should the user want to.

---

**Figure 23.**

Topography, radar, bounding box, and height axis objects.



#### 5.4.3 The elevation surface object

This is essentially a PPI image (see Section 3.3.1 on page 21) with height information added to it: samples from a single elevation angle are shown at their positions in space. An example image with an elevation surface object is shown in Figure 3 on page 14. The surface is generated by forming consecutive triangle strips where the vertices in each triangle are three output samples from the radar, as shown in Figure 24. The figure shows a small part of an elevation surface as seen from above. In the figure there are twelve samples (black dots): four from three different azimuth angles. The first triangle has vertices 1, 2 and  $n+2$ , where  $n$  is the number of samples along a radar ray. All samples in a ray, typically 120, have the same elevation and azimuth angles.

A `vtkPolyData` object can be used to represent geometric structures consisting of vertices, lines, polygons, and triangle strips. Point

---

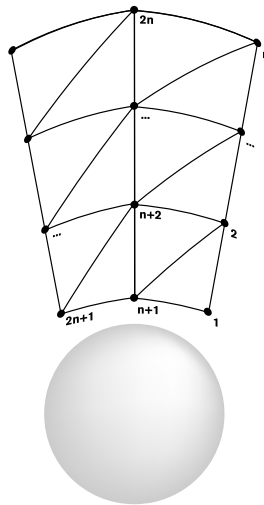
attribute values, e.g. scalars and/or vectors, are also represented. In this case, a scalar value for reflectivity is associated to each point and this is used for colouring. Colours are interpolated, by the graphics hardware, between the vertices. The surface is not discontinuous where there is clear air (zero reflectivity return).

The manager of this object, shown as the currently selected object tab in Figure 22 on page 51, contains controls to select what elevation angle and what azimuth angles (start and range) to use. Note that the azimuth span does not have to be 360 degrees - a smaller slice, or sector, can be shown instead of the whole revolution. These objects can be made more or less transparent using an opacity slider.

---

**Figure 24.**

Creating an elevation surface. Radar as seen from above.

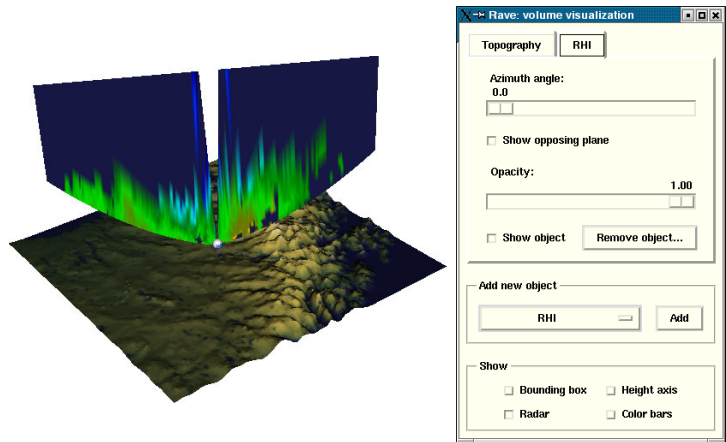


#### 5.4.4 The RHI object

As discussed in Chapter 3, one of the most common 2D products of radar data is the CAPPI image. Another very common product, although not implemented in RAVE until now, is the *range/height indicator*, or RHI. An example RHI is shown in Figure 25.

**Figure 25.**

A range/height indicator, or RHI, object, and its manager.



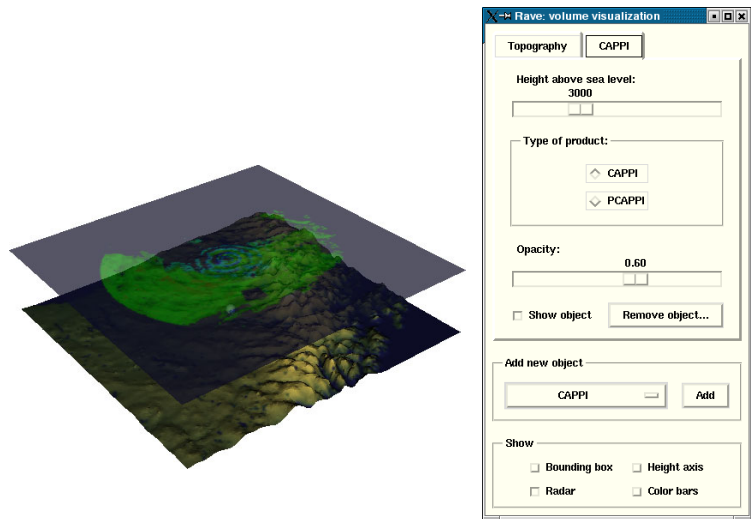
The elevation surface displays data for a single elevation and different azimuth angles. This object performs the complementary operation: it displays data for a single azimuth and different elevation angles. The very same technique is used to generate the actual polygon data: triangle strips are created using three neighbouring points to form each triangle.

The manager for this type of objects contains controls to select what azimuth angle to use and whether or not the RHI 180 degrees from this azimuth angle should be displayed together with it. As with most other objects, opacity can be controlled using a slider.

#### **5.4.5 The CAPPI object**

This object is a plane parallel to the surface of the earth. Technically, it is a texture mapping of a CAPPI or pseudo-CAPPI image, described in Chapter 3, to a plane at the CAPPI height. The manager contains a slider to select height above sea level and a pair of radio buttons to select whether a CAPPI or a pseudo-CAPPI should be displayed (see Chapter 3). There is also an opacity slider. The colouring of these objects is the same as in the case of RHI:s or elevation surfaces: it depends on the reflectivity. A CAPPI object that has been made semi-transparent is shown in Figure 26.

**Figure 26.** A semitransparent CAPPI object and its manager.



#### 5.4.6 The glyphs object

This technique represents reflectivity data by using symbols, or glyphs. A glyph is drawn for each reflectivity sample in the volume. The colouring depends on reflectivity in the particular point.

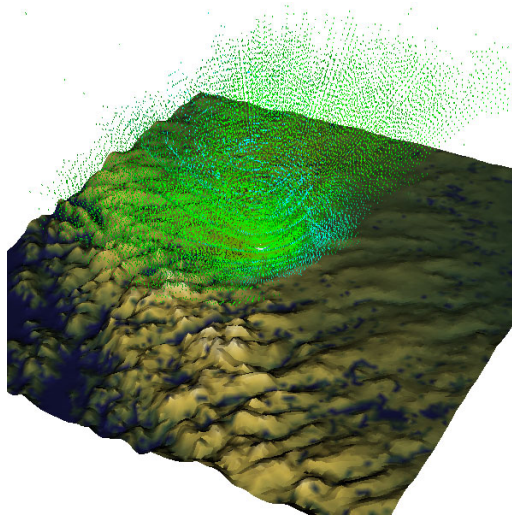
Using controls in the manager, the look of the glyphs can be changed. Currently, the user can choose from using vertical or horizontal lines or 3D crosses, but new glyphs are easily added. Upper and lower thresholds for the glyphs can be set; glyphs are drawn only in those points where the reflectivity value is between the thresholds. It is also possible to choose to display every  $n$ :th glyph, where  $n$  can be set using a slider.

Some glyphs are shown in Figure 27. A still image like this one (especially when printed!) is difficult to interpret. The user has to rotate and pan the image in order to get an idea of the arrangement of samples. This goes for most of the objects.

---

**Figure 27.**

A glyphs object.



#### **5.4.7 The winds object**

This is basically a glyphs object, but for Doppler wind volumes only. A cone is used for glyphing. The cones are scaled according to the wind speeds, and they are directed along the different rays to adequately depict the radial winds that the data represents. The colour of the glyphs can be set to vary with reflectivity (taken from the corresponding reflectivity Doppler volume, if available) or with the wind speed. As in the case of the glyphs object above, the wind glyphs can be scaled, thresholds can be set and the user can choose to show every  $n$ :th glyph. An example is shown in Figure 28. Glyphs are shown for wind speeds between -6 and 6 m/s.

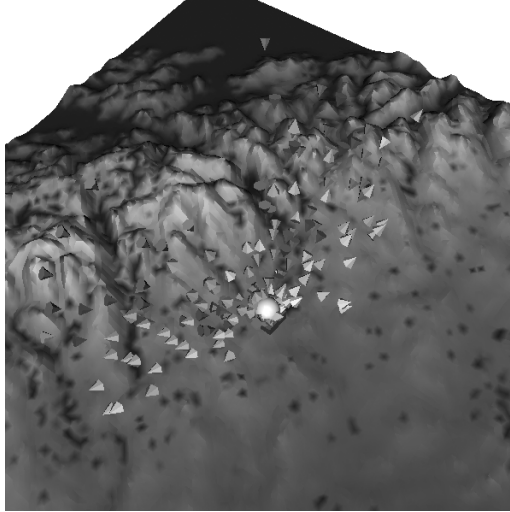
#### **5.4.8 The VAD object**

The VAD product is described in Section 3.3.5 on page 25. It uses data from a Doppler wind scan to create horizontal wind vectors for different altitudes. Using this visualization object, the user can choose the height interval between the vectors and whether they should be coloured according to the wind speed or according to the reflectivity. The vectors are represented as arrows or cones, drawn straight above the radar's position. An example VAD object is shown in Figure 29.

---

**Figure 28.**

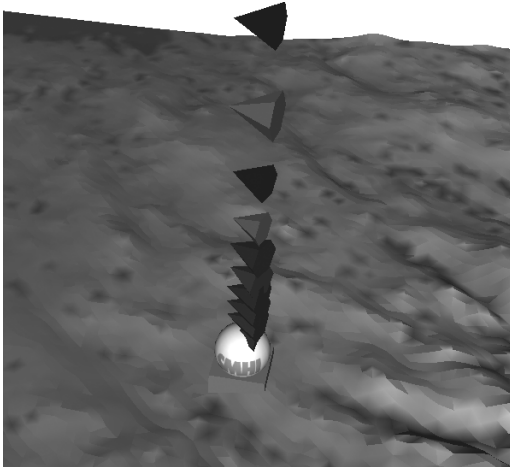
A winds object, as seen from above.



---

**Figure 29.**

A VAD object.



#### **5.4.9 The isosurface object**

Isosurfaces are generated using the marching cubes algorithm, discussed in Section 4.6.1 on page 34. The colouring of the isosurfaces can either be set to vary automatically with the isovalue, which can be adjusted using a slider, or it can be set manually. This is useful when several isosurfaces with similar isovalues (and thus similar colours) are

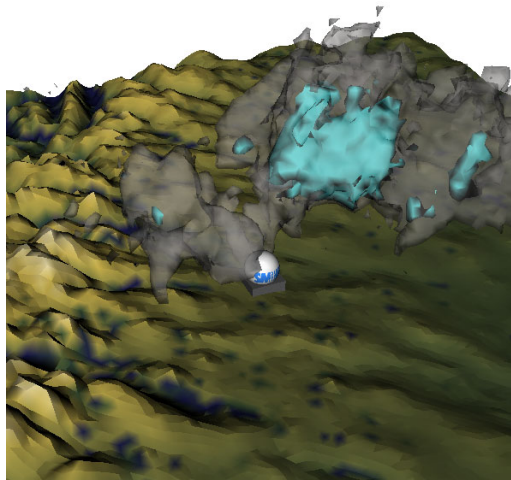


---

displayed simultaneously. The opacity of an isosurface can be set using a slider - otherwise only the surface with the lowest isovalue would be visible. Isosurfaces may look something like the ones in Figure 30. Here, two isosurfaces are shown, the inner one (brighter) has an isovalue of 18.5 dBZ, and the outer one is at 14.5 dBZ and has its opacity set to 0.3 in order to reveal the inner one.

**Figure 30.**

Two isosurface objects, with isovalues 14.5 and 18.5 dBZ for outer and inner surface, respectively.



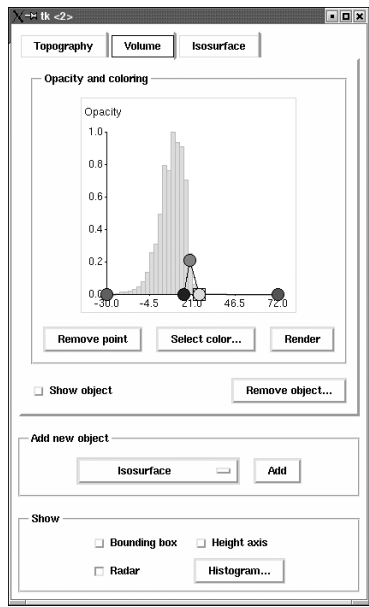
#### **5.4.10 The volume rendering object**

Volume rendering is discussed in Section 4.7 on page 36. This object makes it possible to combine image order volume rendering with the different planes, surfaces and glyphs discussed above.

The object is based on a compositing raycaster, where transfer functions can be specified for colour and opacity variations with reflectivity. The manager of a volume rendering object, shown in Figure 5.4.11, contains an editable graph, where these piecewise functions can be easily altered. Here, a histogram is shown in the background. The histogram describes the number of samples with reflectivities in different intervals. This may be helpful when designing the opacity and colour functions. Opacity varies along the  $y$  axis. Each breakpoint not only defines an opacity through its  $y$  coordinate, but it also defines a colour. Opacities and colours are linearly interpolated between the

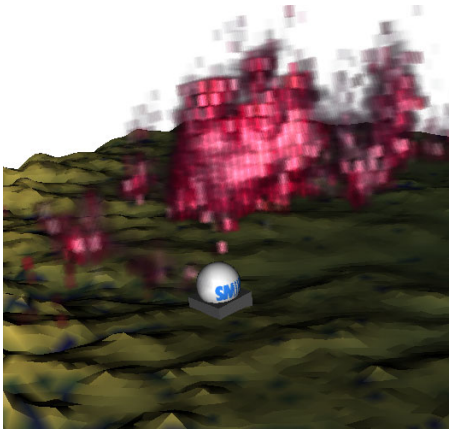
breakpoints. The user can add breakpoints by clicking at the position where the points should be added. Once added, a point can be dragged or its colour can be changed by first selecting the point and then clicking the “Select colour” button. This brings up a colour chooser.

**Figure 31.** The volume rendering manager.



Volume rendering is very slow on computers running at around 500 MHz. This makes it difficult to navigate in the scene, because it takes several seconds to update the display area. Since one ray is cast for each pixel, using a smaller window means that not as many rays need be cast, thereby reducing rendering time. As the computers keep getting faster, the object will be easier to work with in the future. An example of a volume rendering object is shown in Figure 32.

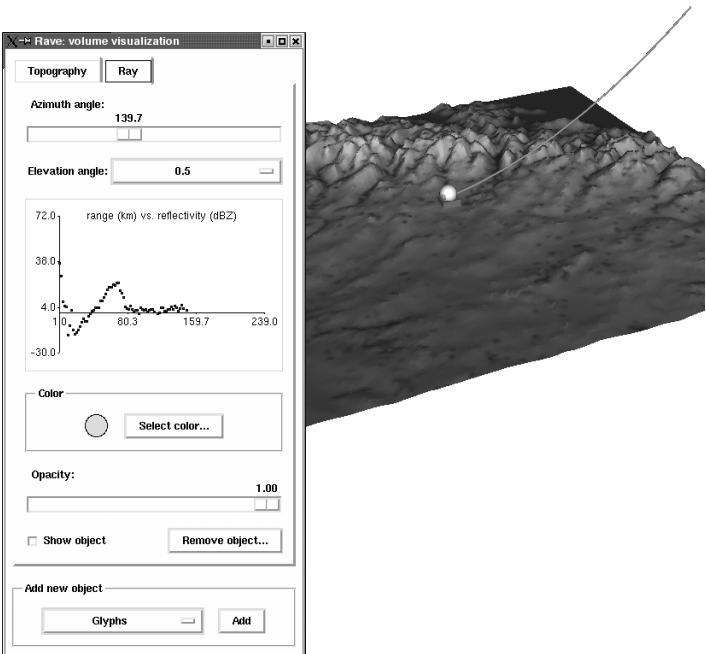
**Figure 32.** A volume rendering object.



**5.4.11 The radar ray object**

The radar ray object draws a ray with elevation and azimuth angles as set by the user. The user can choose to view a plot of the reflectivity variation along the ray. An example ray is shown in Figure 33.

**Figure 33.** A radar ray object and its manager.



---

---

---

# 6

## Evaluation and conclusions

---

### 6.1 Introduction

In the beginning of this project it was very unclear what the final result would be like. Little research has been made within the field of 3D visualization of radar data so it was difficult to know what to expect. Thus, different ideas were discussed, different kinds of objects were tested in isolation, and the ones that were judged as useful have been included in the application. Informal evaluations have been performed during the development and these have gradually set directions for the following design. In this chapter, some of the thoughts and opinions that have arisen during discussions with researchers and meteorologists are presented, along with some conclusions that can be drawn from the work that this paper has presented. Areas of future research and/or extensions are also here.

### 6.2 Evaluation

In this section, the application's usefulness and performance are discussed.

#### 6.2.1 Usefulness

Apart from the use as a research tool for the analysis of exceptional weather situations, one of the areas where this application would be useful is for weather forecasting around airports. At the end of this project, the application will be installed at Arlanda airport, outside Stockholm. Weather radars are used extensively around airports as one of the most important decision-making tools when creating local, short-term forecasts. Both heavy precipitation and strong winds heavily affect the traffic around airports; as we have seen, both may be

---

analyzed using radar technology. The application has been demonstrated for a meteorologist working at Arlanda and she believed that it offers the possibility to extract features from the current weather situation that are very difficult, if not impossible, to see using the existing 2D visualization systems. Some of her thoughts about a few objects are presented below. Her main thoughts on the application were very enthusiastic, and this has led to the installation of it at her workplace.

#### **Range/height indicators**

The height of precipitation clouds is important information when aircraft should be routed around airports. This includes the altitude of the clouds' precipitation kernels and their thicknesses. The range/height indicator is a common 2D product in radar visualization systems used to investigate just that. In this implementation, the main strength not offered by plain 2D systems lies in the ability to relate the RHI geographically to topography, and to other objects (including other RHIs) in an intuitive way. In the case of plain 2D systems, the user has to create a 3D environment him/herself and this is not easy! Using 3D technology, the computer takes over the time-consuming, difficult task of building the 3D environment.

Basically being a 2D object, its orientation and location in the 3D scene provides important extra information.

#### **Isosurfaces**

This is an object that takes some time to get used to, as it is a true 3D object that has no equivalent in 2D visualization of radar data<sup>1</sup>. It is useful in detecting and displaying the kernels, the areas of the most dense precipitation, in clouds. By slowly decreasing the isovalue, the isosurface expands and it is possible to see the extent of different precipitation densities.

For users at SMHI and Arlanda, this is a completely new way of examining weather radar data and the users are enthusiastic.

#### **Radar rays**

The radar ray object displays a ray in the 3D view and it is possible to display a plot to show how the reflectivity varies along the ray. Choos-

---

1. The 2D equivalent of isosurfaces is the isoline. This is not used in radar visualization, though.

---

---

ing elevation and azimuth angles so that the ray corresponds to a path followed by a landing aircraft or one taking off, it is possible to see what the aeroplane has to go through in terms of reflectivities.

Nothing similar has been available to meteorologists at Arlanda before, and there is good hope that this object, too, will be of great usefulness to them.

**Volume rendering**

In a volume rendering, the user basically has the possibility to view the whole echo picture at once. The big problem here is the speed; it just takes too long to render an image for the object to be useful. When this problem is overcome, there will probably be great use for it. In the meantime, the glyphs object can be used instead to provide similar products.

**6.2.2 Performance**

The application performs well, mainly thanks to VTK being a well-written C++ library, and to hardware accelerated 3D graphics renderers. The 3D graphics are rendered at interactive speeds (at least ten frames per second) in windows that are some 1 000 x 1 000 pixels in size when a modern, though not state-of-the-art, graphics card is used. The application has been tested with the two configurations shown in Table 2.

---

**Table 2.**

Two different computer configurations tested with RAVE.

	System A	System B
Processor	500 MHz Intel Pentium 3	600 MHz AMD K7
Graphics chipset	nVidia TNT Ultra 2, 32 MB	nVidia GeForce 2 Ti, 64 MB
Memory	256 MB	384 MB

The main difference between these two systems is the graphics chipset used; in System A, it is based on the three-year-old TNT technology, while System B has the GeForce 2 Titanium chipset, released during the second half of 2001 from the same manufacturer. The difference in performance is quite astounding! Using System A, the application is certainly usable, with typical rendering speeds of around ten frames per second in large windows, but it is even more so

---

when using System B; rendering speeds are at least two times the ones achieved with System A.

There are two cases when the application shows significant processing delays: the first one has been mentioned several times already, and it has to do with volume rendering. The calculations are performed completely by the CPU; there is no dedicated hardware for this<sup>1</sup>. Using the systems above, there is however a notable difference between System A and System B (i.e. the latter is faster and performs better). As of this writing, a new computer has a clock speed of around 1.5 GHz and this difference would probably be significant. This exciting test remains to be executed.

The second bottleneck originates from Python being terribly slow at looping through numeric arrays. This operation is performed the first time a `vtkPolyData` object is generated; data is read from a Python array that is created when RAVE opens a polar volume, the values are linearly transformed, and stored in a `vtkPolyData` object. This loop takes around ten seconds - an eternity in front of a computer screen - to perform. The good news is that this loop only has to be performed once for each type of data (i.e. reflectivity, Doppler reflectivity, or wind) and session. The same phenomenon appears when data from a whole polar volume needs to be looped through at other times, e.g. when creating a histogram. Again, faster computers will partly solve this problem. There is also a possibility that future versions of Python will replace the unnecessarily slow array handling routines with faster ones.

### 6.3 Future improvements

During this project, some ideas of future research and development have arisen. The most important are presented briefly below:

- Spatial animations. The user should be able to create animations within a single polar volume, e.g. to set first and last angle of an RHI, and to watch it “spin”.
- Temporal animations. The user should be able to open several polar volumes from the same location but with different timestamps, and

---

1. Actually, VTK supports a line of (expensive) volume rendering accelerators. These would certainly solve the problem of the slow volume renderings, but due to their high prices, they have not been considered as an option.



---

to manually and automatically step forward and backward in the sequence.

- Saving options. The user should be able to save settings; this may include saving camera and object settings to a file, but it may also include the possibility to store camera positions temporarily during a session, i.e. “camera bookmarks”. The user should also be able to save the current view as a still image.
- Composites. The user should be able to view data from several radars simultaneously.

## **6.4 Conclusions**

The purpose of this work, as stated in Section 1.2 on page 1, was “to design and implement a generic, object-oriented application for 3D visualization of weather radar data”. The application developed during this project is a complete, generic, ready-to-use tool for 3D visualization and analysis of weather radar data.

Compared to the traditional 2D products used by meteorologists when analyzing this kind of data, the 3D scenes add information that makes it easier for the user to understand the actual depicted weather situation.

As of this writing, no formal evaluation of the application has taken place. Such an evaluation will be conducted in the near future. The first demonstrations and discussions with meteorologists have rendered positive reactions and it is the hope of the author that the application will be of great use during many years to come.

---

---

---

# Resources

---

## 7.1 Books and articles

Bolin, Håkan & Daniel B. Michelson: *The Radar Analysis and Visualization Environment: RAVE*. Preprints, American Meteorological Society: 28th Conference on Radar Meteorology (1997), pp 228-229.

Danielsson, Per-Erik, Olle Seger & Maria Magnusson-Seger: *Bildbehandling 2000*. Linus & Linnéa, Linköping, 2000.

Gallagher, Richard S.: *Computer Visualization - Graphics Techniques for Scientific and Engineering Analysis*. CRC Press, Inc., USA, 1995.

Hearn, Donald & M. Pauline Baker: *Computer Graphics - C Version, 2nd edition*. Prentice-Hall PTR, Upper Saddle River, New Jersey, USA, 1997.

Lichtenbelt, Barthold, Randy Crane & Shaz Naqvi: *Introduction to Volume Rendering*. Prentice-Hall PTR, Upper Saddle River, New Jersey, USA, 1998

Raschke et al: *The Baltic Sea Experiment (BALTEX): A European Contribution to the Investigation of the Energy and Water Cycle over a Large Drainage Basin*. Bulletin of the American Meteorological Society, vol. 82 (2001), no 11, pp 2389-2413.

Rinehart, Ronald E.: *Radar for Meteorologists*. Department of Atmospheric Sciences, University of North Dakota, Grand Forks, USA, 1991.

Schroeder, Will, Ken Martin & Bill Lorensen: *The Visualization Toolkit, 2nd edition*. Prentice-Hall PTR, Upper Saddle River, New Jersey, USA, 1998.

---

Schroeder, Will et al: *The Visualization Toolkit User's Guide*. Kitware, Inc., 2001.

Watt, Alan H.: *3D Computer Graphics, 3rd edition*. Addison-Wesley Pub Co, USA, 1999

## 7.2 Internet

Advanced Visual Systems: *Advanced Visual Systems: Software*. [http://www.avis.com/software/soft\\_t/avsxps.html](http://www.avis.com/software/soft_t/avsxps.html). Visited 2001-09-11.

Environmental Modeling Systems, Inc.: *Inverse Distance Weighted Interpolation*. [http://www.ems-i.com/smshelp/Scatter\\_Module/Scatter\\_Scattermenu/Inverse\\_Distance\\_Weighted.htm](http://www.ems-i.com/smshelp/Scatter_Module/Scatter_Scattermenu/Inverse_Distance_Weighted.htm). Visited 2001-12-19.

Hopper, Randall: *Py-OpenDX*. <http://people.freebsd.org/~rhh/pyopendx>. Visited 2001-09-12.

The Numerical Algorithms Group Ltd.: *IEC - Iris Explorer Center*. [http://www.nag.co.uk/Welcome\\_IEC.html](http://www.nag.co.uk/Welcome_IEC.html). Visited 2001-09-12.

OpenDX.org: *Open Visualization Data Explorer*. <http://www.opendx.org>. Visited 2001-09-12.

Research Systems: *IDL - The Interactive Data Language*. <http://www.rsinc.com/idl>. Visited 2001-09-12.

Space Science and Engineering Center: *Vis5D Home Page*. <http://www.ssec.wisc.edu/~billh/vis5d.html>. Visited 2001-09-12.

BALTEX Radar Data Centre home page. <http://www.smhi.se/brdc>. Visited 2001-12-18.

---

# Appendix A

## Class diagrams

---

### A.1 Introduction

An overview of the basic design of the application developed during this project was given in Section 5.3 on page 50. As described in that section, there are two basic subsystems: the graphical user interface and the visualization objects. In this appendix there are two class diagrams describing the relations, methods and parameters of the different classes within these two subsystems.

Three software packages have been used extensively: *The Visualization Toolkit*, which consists of some shared object libraries (written in c++) along with Python bindings to access these; *Tkinter*, which is a Python interface to the GUI package Tk; and *Python MegaWidgets*, or PMW, which is based on Tkinter and provides higher-level graphical user interface widgets, such as tabbed panes and grouped buttons.

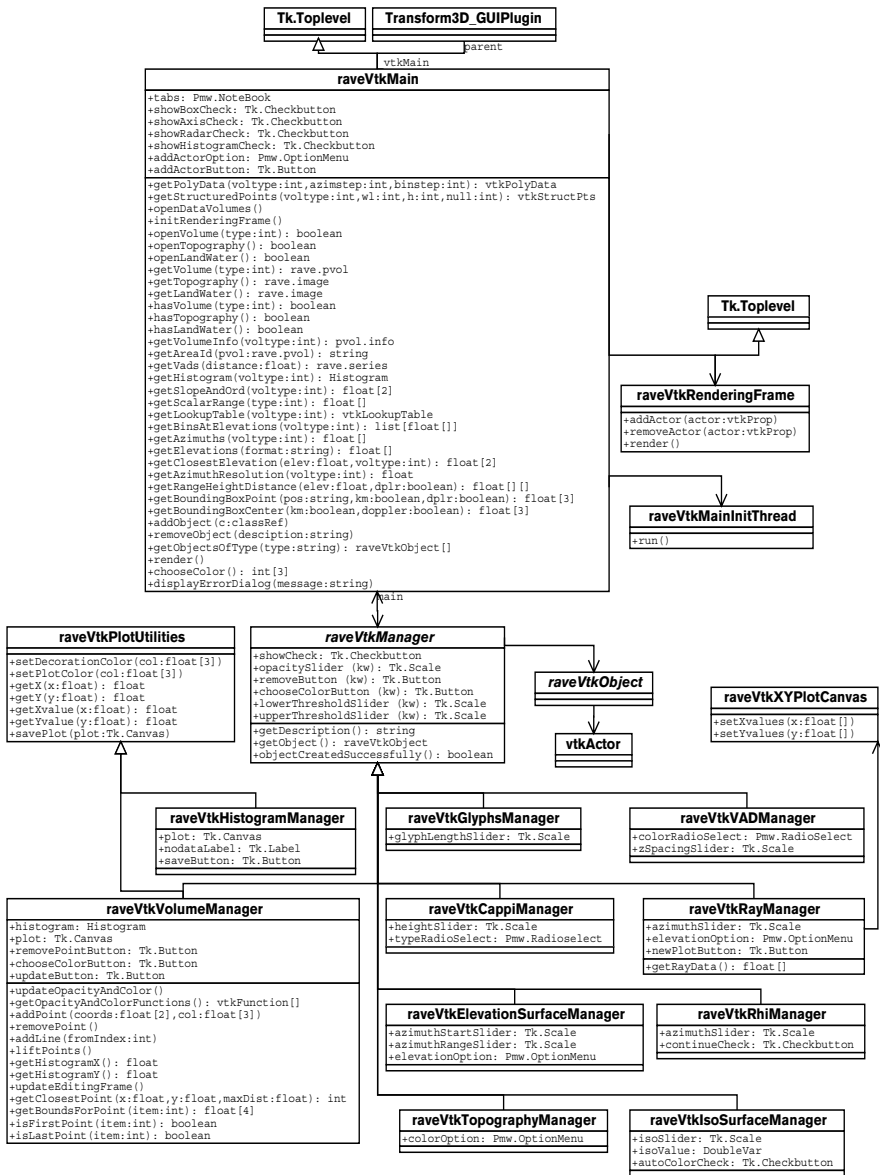
All classes developed during this project are written in the Python language. The software runs on Linux but could easily be adapted to run on Unix platforms as well.

### A.2 Graphical user interface

The class diagram for the graphical user interface classes is shown in Figure 34. The `raveVtkMain` class inherits from the `TkToplevel` class. This is an ordinary window. A `raveVtkMain` instance is created from within a `Transform3D_GUIPlugin`, which in turn is created when the RAVE user chooses to create a PPI, CAPPI, PCAPPI, or Volume visualization product from the Products menu in the main GUI (see Section 3.2.1 on page 20).

In the constructor of `raveVtkMain`, as well as in every other constructor in this system, instance variables are set to default values or to values specified as arguments to the constructor.

**Figure 34.** Graphical user interface class diagram.



When the instance variables have been set, some files are opened: polar volumes with reflectivity, Doppler reflectivity, and wind values; and elevation and land/water maps for the topography. A thread is

---

created and started (instance of `raveVtkMainInitThread`) which prepares some common data: `vtkPolyData` and histograms, for instance. The prepared data is handed over to the `raveVtkMain` instance, and this initialization thread destroys itself.

Then the actual graphical user interface is created: buttons and labels are added to the window, as well as a tabbed pane to contain the managers that are added with each new object. Event callbacks are associated with every clickable item. The definitions of the event callbacks are in the `raveVtkMain` class itself or in the manager class to which their corresponding widgets belong (they are not included in the class diagram, though). Their main function is usually to pass on the call to some other method (e.g. `addObjectClicked()` calls `addObject()` with the appropriate arguments).

Next, a `raveVtkRenderingFrame` instance is created. This is a new window to display the 3D graphics. It makes use of a `vtkTkRenderWindow` instance. This is a Tk compliant object included with VTK's Python bindings. This class, too, as can be seen in the class diagram, inherits from the `Tk Toplevel` class.

Once the rendering frame has been created, a few objects - topography, radar, bounding box, and axis - are created. When the topography object is created, a topography manager object (instance of `raveVtkTopographyManager`) is created and a new tab is added to the tabbed pane in the `raveVtkMain` window.

Every time a new manager is created, a reference to the `raveVtkMain` object is passed along to the manager's constructor. This way, the managers are able to use the many methods for generic data handling included in the `raveVtkMain` class. For instance, it is `raveVtkMain` that creates some fundamental VTK objects useful to many of the different visualization objects (the most important objects are accessed through `getPolyData()` and `getStructuredPoints()`). And since it is `raveVtkMain` that is responsible for file opening, information about the opened files can be requested via calls to some specific methods.

### **A.2.1 Managers**

There is a manager class for each visualization object class. The manager is responsible for creating widgets in the `raveVtkMain`

---

instance's tabbed pane. A reference to a newly created tab is passed to the manager's constructor when this is called from `raveVtkMain`. The constructor initializes instance variables and widgets.

There is an “abstract” class<sup>1</sup> called `raveVtkManager` from which all the concrete manager subclasses inherit some basic properties. The constructors in the concrete manager classes all begin with a call to `raveVtkManager`'s constructor, supplying different *keyword arguments* to it. Keyword arguments is a powerful Python construct which gives the user the possibility to supply different, optional arguments to a method, function, or constructor. In this case, the technique is used to make it possible for different managers to add different, common widgets to themselves. For instance, a widget that is included in several managers is a “Select colour” button along with a small oval displaying the currently selected color for the object. This widget is added to a manager by calling the `raveVtkManager` constructor with an argument like `colorChooser=4`, meaning that the colour selection widget should be added at the manager's grid row number four. Several different options like this one are available. This makes the sizes of the managers' implementations quite small, and it is easy to create new managers that include some of the most common buttons and menus.

Event callbacks to the common widgets are all in the `raveVtkMain` class, but these may be overridden by definitions of methods with the same names in the concrete subclasses (so-called *dynamic binding*). Event callbacks to widgets specific for a certain manager class are defined in that class.

Each manager creates, at the end of its constructor, an object to be associated with it. A `raveVtkIsosurfaceManager` creates a `raveVtkIsosurface` instance, and so on. These objects are described in the next section.

---

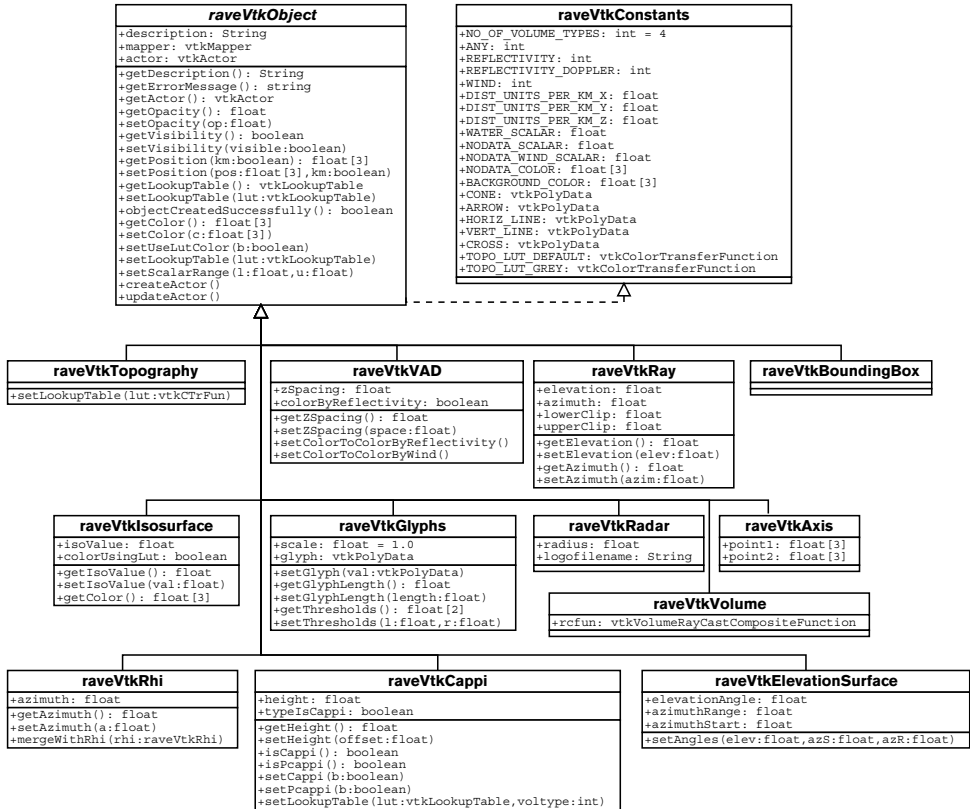
1. Actually, there is no support for abstract classes in Python. It is always possible to create instances of every class, but in the case of `raveVtkManager`, it would make no sense to do so - in its own, it is useless!



### A.3 Visualization objects

This is the second group of classes and the class diagram is shown in Figure 35. Again, there is an “abstract” superclass of all objects, simply called `raveVtkObject`, that contains some attributes and methods that are common to all concrete subclasses. These include, for instance, the `setPosition()` and `setColor()` methods.

**Figure 35.** Visualization objects class diagram.



All object classes serve as containers for some VTK objects. These objects are connected to form a visualization pipeline for each different type of object. The pipeline typically starts with data from `raveVtkMain`'s `getPolyData()` or `getStructuredPoints()` methods, and it always ends with an instance of `vtkActor`, or one of its subclasses, and one of `vtkMapper`'s subclasses.

---

---

One of the simplest possible visualization pipelines can be found in the `raveVtkIsosurface` object. It starts with the output from `raveVtkMain.getPolyData()`, adds a `vtkMarchingCubes` object and a mapper and actor. When the object has been created, the isovalue can be set using the `raveVtkIsosurface.setIsovalue()` method, and this is the method that is called when the user adjusts the isovalue using the isovalue slider in the manager. This method, in turn, calls the `SetValue()` method of the `vtkMarchingCubes` instance. All objects basically operate this way: the managers issue calls to the objects due to user interactions with the different sliders, menus, and buttons, and the objects pass the calls on to some underlying `vtk` object. Finally, a rerendering is requested from the `vtkTkRenderwidget` instance in order to reflect the changes.



På svenska

Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under en längre tid från publiceringsdatum under förutsättning att inga extra-ordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida <http://www.ep.liu.se/>

In English

The publishers will keep this document online on the Internet - or its possible replacement - for a considerable time from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: <http://www.ep.liu.se/>

© Aron Ernvik