

MASTER

Real-time realistic radar simulation

Wagener, P.

Award date:
2012

[Link to publication](#)

Disclaimer

This document contains a student thesis (bachelor's or master's), as authored by a student at Eindhoven University of Technology. Student theses are made available in the TU/e repository upon obtaining the required degree. The grade received is not published on the document as presented in the repository. The required complexity or quality of research of student theses may vary by program, and the required minimum study period may vary in duration.

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain

Eindhoven University of Technology
Department of Mathematics and Computer Science

Real-time Realistic Radar Simulation

Paul Wagener

Master's Thesis
November 2011

Supervisors: Huub van de Wetering, Andrei Jalba

Abstract

The screen of a radar system is complicated and subject to many artifacts. Many physical aspects of the world need to be simulated in order to faithfully reproduce it. In this thesis we introduce an algorithm that can output a realistic image of a radar screen and update it in real-time. This algorithm is designed to take advantage of the graphics hardware to allow for a realtime simulation. It works by rendering an image from the viewpoint of the radar antenna and using the depth map of this image as an analogue to the distance-measuring performed by a real radar system. In this thesis we compare the results of our algorithm with the results of existing radar simulations as well as real radar systems. Our radar simulation is currently the only one known to us that can simulate in *real-time* almost all of the features of a real radar system.

4

Contents

1	Introduction	5
2	Radar	7
2.1	Overview	7
2.2	Radar basics	7
2.2.1	Screen	7
2.2.2	Antenna	9
2.2.3	Signal strength	12
2.2.4	Radiation pattern	13
2.2.5	Antenna gain	14
2.2.6	Horizontal beamwidth	14
2.2.7	Vertical beamwidth	15
2.2.8	Pulse length	15
2.2.9	Pulse frequency	16
2.3	Radar effects	17
2.3.1	Occlusion	17
2.3.2	Echo scaling	17
2.3.3	Sea clutter	19
2.3.4	Rain clutter	21
2.3.5	Side lobes	21
2.3.6	Specularity	23
2.3.7	Specular reflections	24
2.3.8	Second Trace	24
2.4	Screen	26
2.4.1	Gain Control	26
2.4.2	Sea Clutter Control	27
2.4.3	Rain Clutter Control	28
2.4.4	Additional information	29
3	Radar Simulation	31
3.1	Input	31
3.2	Output	32
3.3	Related Work	33
3.3.1	Naive algorithm	33
3.3.2	Radar Signal Simulation	33
3.3.3	Radar Image Simulation Based on 3d Scene Database in Marine Simulator	37
3.3.4	Transas	37

3.3.5	In2Sim	37
3.3.6	Related simulations	39
4	Rendering-based simulation algorithm	41
4.1	Sending the signal	43
4.2	Render settings	45
4.3	Pulse length	45
4.4	Specularity	45
4.5	Sidelobes	46
4.6	Screen	48
4.6.1	Gain control	48
4.6.2	Rain control	48
4.6.3	Sea control	49
4.7	Specular Reflections	49
4.8	Finding surface normals	51
4.8.1	Solution 1: From depth map	51
4.8.2	Solution 2: By color coding	52
4.9	Complexity	54
5	Results	57
5.1	Occlusion	57
5.2	Side lobes	58
5.3	Pulse Length	60
5.4	Specularity	60
5.5	Specular Reflections	62
5.6	Gain Control	64
5.7	Rain Control	65
5.8	Sea	65
5.9	Simple Environment	67
5.10	Complex Environment	68
5.11	Complex Environment 2	68
5.12	Extra information	70
6	Conclusions & Future Work	73
6.1	Future Work	73
6.2	Evaluation	74
A	Depth Buffer	77
B	Radiosity	81
B.1	Global Illumination	81
B.2	Radiosity algorithm	81

Chapter 1

Introduction

Since the second world war radar (short for Radar Detection And Ranging) systems have been the single most important navigation instrument on ships[1]. Today it is mandatory equipment for every motorized ship on everything ranging from tugboats to ocean tankers[4]. A radar system allows mariners to see an overview of the surroundings of their ship (see figure 1.0.1). The radar screen presents them with a map on which coastlines and neighbouring ships are highlighted allowing them to plot a course and navigate without collisions.

A radar system allows ships to navigate in heavy fog or at night when visibility outside is low. The picture on the radar screen will mostly remain unaffected by outside visibility. Consequently, busy ports such as Rotterdam may remain in operation for 24 hours a day, regardless of weather conditions. Without radar this would be impossible.

Radar is however not a magical map of the environment. It can only see the surroundings which are in direct line of sight of the ship (plus a little more as explained in section 2.3.7). And there are many effects that clutter and distort the radar screen. Because of this it takes special training to correctly interpret a radar image and to ignore misleading information. This training used to take place on a real ship behind a real radar system. But due to high cost, trainers have started to use radar simulators.

These radar simulators (see figure 1.0.2) run a computer simulation of a ship faring in a river or on the ocean. The trainee is required to navigate with no outside image. They can only use the radar image for navigation, simulating a zero visibility condition. The trainee must correctly identify obstacles and other ships on the radar and plot a course that avoids collisions. If they can correctly navigate by radar for 45 minutes the trainee has passed and gets a certificate indicating that he or she is qualified to use a radar system.

The simulation of the radar screen must be as life-like as possible, not only all the features of a modern radar should be present, but also all the limitations and distortions. On top of that the simulation should also be able to update the screen as fast as a real radar updates its screen. How to accomplish these two requirements will be the main focus of this thesis.



Figure 1.0.1: A modern radar system showing part of the Nieuwe Waterweg.



Figure 1.0.2: A radar training station at the Scheepsvaart Transport College in Rotterdam. The radar simulation is visible on the center-right monitor.

Chapter 2

Radar

When thinking about radar the image that comes to mind is a large green glowing circular display with dots that represent nearby ships. This is usually accompanied by a green glowing line, sweeping clockwise around the screen while updating the green dots underneath it. While the green glowing monitors have been replaced by high-resolution LCD screens, this image is still accurate. Even the most modern radars still have the sweeping line updating the contents of the screen[6]. Even though the technology of the radar has evolved since its first use in the Second World War, the principle behind it has stayed the same.

2.1 Overview

In this section we explain how a radar system constructs a radar screen with echo location. We will describe all the different parameters of a radar system that can influence the outcome, such as signal strength, radiation pattern, beam width, pulse length and pulse frequency. We also explain the most important artifacts that a radar screen can display, such as occlusion, echo scaling, sea clutter, rain clutter, side lobes and second trace echoes. And conclude with how gain control, rain control and sea control can influence the radar image.

2.2 Radar basics

A radar system is made up of two parts:

- The *screen*. This is the display that shows the neighboring environment as a round map. Around this map the screen shows additional information and controls.
- The *antenna*. This is the detection device that is mounted on top of the ship and that feeds data about the environment to the screen.

2.2.1 Screen

A radar screen is the display that always shows a map of the surroundings of the radar antenna. The center of the map is the location of the radar antenna. The distance on the map from the center to the edge of the radar screen is known as the *range* (figure 2.2.1). The greater the range the more zoomed out the map on the radar display is. The range can be configured by a radar

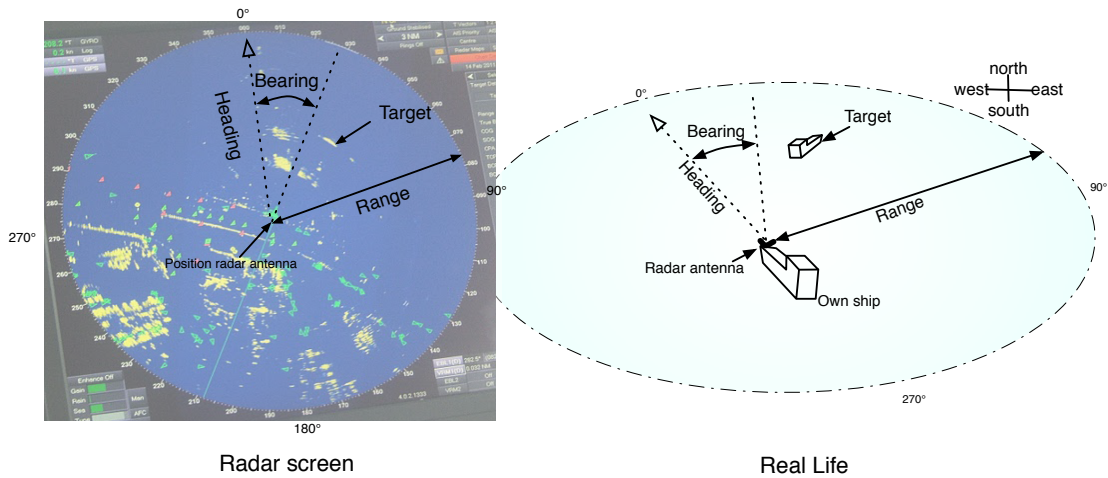


Figure 2.2.1: Heading, bearing and range on a radar screen.

operator depending on the situation, though it is limited by the maximum range of the radar antenna.

Most modern radar screens have a blue background and show in yellow the objects within the range. All highlighted objects on the screen that exist in real life and are displayed at the correct position are known as *targets*. Everything else that is highlighted is known as *clutter*. Clutter distracts from the real objects on the screen. The radar operator wants to reduce clutter as much as possible. An overview of all the different types of clutter is given in section 2.3.3.

A direction on the radar screen is known as a *bearing* and is measured in degrees. A bearing is relative to the bow of the ship. A bearing of 0° points straight ahead while a bearing of 90° points to the right side of the ship. The bearing is used by navigators to identify ships and to calculate if another ship might be on a collision course. Around the radar screen all bearings that are a multiple of 10 are displayed. The *heading* of a ship is the direction in which the bow of the ship is pointing relative to the magnetic north. If a ship has a heading of 0° , it is going towards the north pole, and a ship that has a heading of 90° is going east.

To paint the image the screen gets information from the antenna about the environment. However the antenna is directional and can only give information about one direction at a time. The screen can only update a tiny area of the screen in the direction that the radar antenna is pointing to (figure 2.2.2). We call this area that can be updated a *slice*. The screen is made up of a finite number of slices, equal to the number of bearings the radar can send pulses to in a single revolution.

This is a problem as one usually wants the entire screen to get updated with new information from the antenna. In order to achieve this, the antenna rotates in a clockwise direction constantly sending information to the screen about each direction it points to. This gives the screen the chance to update the entire image as the antenna makes one full revolution. This also gives rise to the typical radar effect where a clockwise sweeping line seems to update the contents of the screen.

The information that the antenna sends to the screen is a continuous analog stream that encodes how far the targets are (which unfortunately includes also clutter as introduced in section 2.3.3). To detect targets, the antenna sends out a pulse of radio waves and listens for

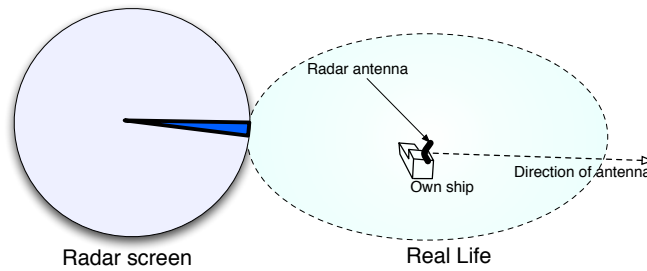


Figure 2.2.2: Slice of the screen that can be updated given the bearing of the radar antenna in real life.

returning signals (see section 2.2.2 for a detailed overview). In figure 2.2.3 an example situation is given with two targets and an antenna that sends three pulses.

In this example there are two targets roughly ahead of the ship. The radar should display this on the screen as two yellow areas on top of a blue background. To do this it makes for each pulse a one dimensional image strip that it paints blue with yellow on top for the parts where the antenna detected a target. To decide if something is a target the user can configure a threshold. Every value above the threshold will be colored yellow, while every value below it will remain blue. This threshold is influenced by the gain, rain and sea control, and is further explained in section 2.4.1.

Each image strip is drawn as a pie-shaped slice on the radar screen. If for example the pulse from the antenna was sent to bearing 5° the information from that pulse would be drawn in a slice pointing to 5° on the radar screen (figure 2.2.4). We have given here an example where the antenna only sends a pulse every 5 degrees of rotation. In reality, radar antennas can send pulses as fast as one pulse per degree of rotation[6].

We can summarize this with the following algorithm, where PPR is the amount of pulses the antenna sends during one revolution (section 2.2.9) and $bearing$ is the direction the antenna is pointing relative to the bow of the ship:

Algorithm RADAR-ALGORITHM

1. $bearing \leftarrow 0$
2. **while true**
3. $bearing \leftarrow (bearing + \frac{360}{PPR}) \bmod 360$
4. point antenna to $bearing$
5. send pulse
6. $signal \leftarrow$ receive pulse
7. color $image$ blue
8. color $image$ yellow where $signal$ above threshold
9. Fill slice[$bearing$] on screen with $image$

2.2.2 Antenna

The principle behind radar detection is based on the same principle a bat uses to detect his surroundings and has everything to do with echo location. To detect a distant target with echo location, a signal is sent in the direction of the target (see figure 2.2.5). The signal travels to the target, reflects against it and returns to the sender. The time it took for the signal to travel to

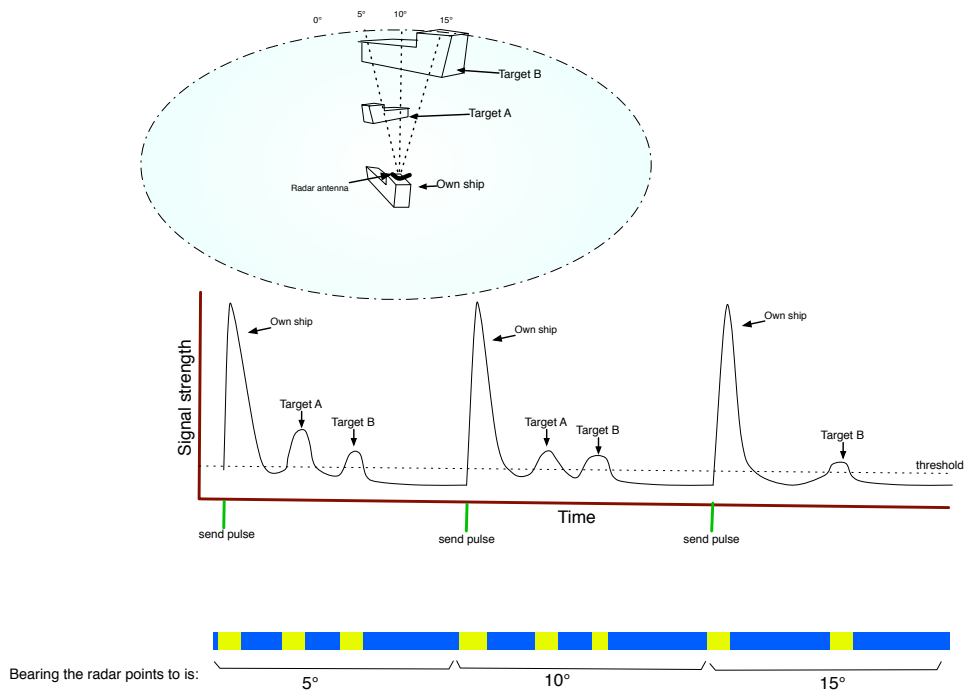


Figure 2.2.3: Example of information stream from antenna with radar configured to send information every 5 degrees. Below the colors based on the threshold.

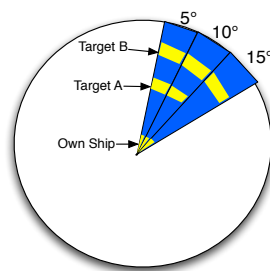


Figure 2.2.4: Information from figure 2.2.3 reconstructed onto the radar screen.

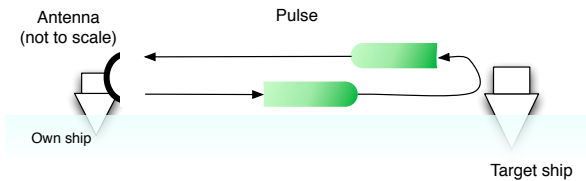


Figure 2.2.5: Radar waves sent by the radar antenna bounce against a target and travel back to the antenna.

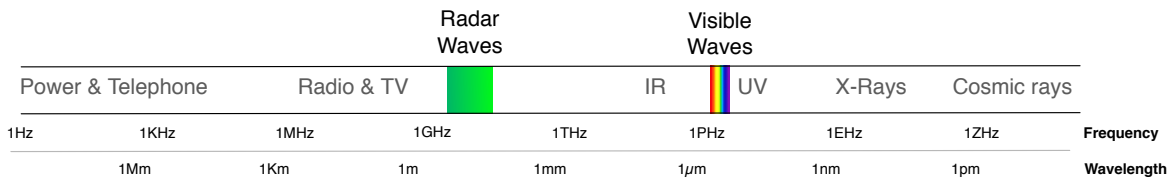


Figure 2.2.6: Location of radar waves in the electromagnetic spectrum.[8]

and from the target gives the distance between sender and target. The longer the signal travels the farther away the target is.

Whereas a bat uses sound, a radar uses radio waves¹. This has the advantage that the signals travel significantly faster (the speed of light instead of the speed of sound, approximately a million times as fast). Because of this many more signals per second can be sent.

Radar uses radio waves which form a specific band of the electromagnetic spectrum. The radio waves sent out by radar antennas have a frequency of around 1.55 to 5.2 GHz[8] (see figure 2.2.6)². These frequencies are high enough not to be hindered by smoke and fog but small enough to reflect against most solid objects[5].

In physics the term *light* refers to the band of the spectrum that can be seen by the naked eye. In this thesis this term will also be used to refer to the radar waves band. This is technically a misnomer but it should aid comprehension if the term 'light' is used instead of the term 'electromagnetic radiation'. While radar waves are not visible to the human eye they behave in the same way as visible waves do.

The radar antenna continually sends and receives pulses. A *pulse* is a short flash of light. The pulse travels away from the antenna in an expanding shell of light. Most of the light vanishes into the sky or scatters away in a random direction. But a small amount is reflected back into the antenna. The same antenna that sent out the pulse now receives the pulse and accurately keeps track of how much light returns and at what time it returned. This data is sent to the screen which processes it into a picture (section 2.4).

The antenna has to wait for the previous pulse of light to have either returned or dissipated into the environment before it can send. If the antenna sends out a new pulse while light from a far away target is still in transit back to the antenna an ambiguous situation can occur. Figure 2.2.7 depicts a situation where there is one target in front of the antenna. It is however not clear how far away this target is. The signal from this target could either be reflected from a far away

¹Machines using echo location with sound also exist and are used in environments where radio waves cannot differentiate between signal and noise such as deep water. These machines are called SONAR.

²Machines using echo location with electromagnetic radiation near the visible spectrum also exist and are called LIDAR. These are typically used for short range application such as archeology, 3d scanning and speed guns.[16]

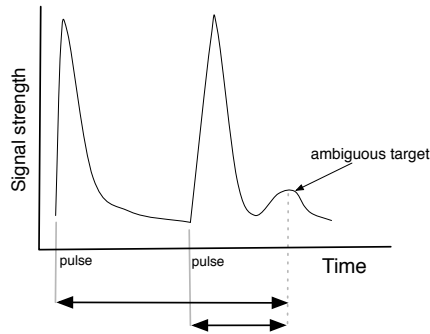


Figure 2.2.7: Pulses are too close together which makes the distance to the target ambiguous.

target from the first pulse or from a close target from the second pulse. To avoid this situation enough time has to be put in between pulses. Generally it is safe to send a new pulse after the amount of time it takes for light to travel to the furthest detectable target and back.

2.2.3 Signal strength

All the light the antenna sends in a pulse originates at the radar antenna and expands out in a sphere of light (the light intensity on this sphere is not uniform as will be explained in the next section). As the light expands in every direction the energy of the light in a fixed area on the sphere gets weaker with the inverse square of the distance. A target that is close can return 1.000.0000 times as much light back to the antenna than a target that is far away[15]!

While far away targets should not be shown 1.000.000 times as dim on the radar screen. Therefore, before the antenna passes the information to the screen it applies a filter that amplifies the incoming signal. To determine how much the signal needs to be amplified one needs to consider how weak the signal is as it is returning from a given distance. As the light expands across the environment the energy (E) reduces with the reciprocal of the square of the distance (r), i.e.,

$$E = \frac{1}{r^2} \quad (2.2.1)$$

That means that the strength of the signal when measured at a target at 2 kilometers is 1/4 the strength of the signal at 1 kilometer. At 3 kilometer the strength is 1/9 and at 4 kilometers the strength is 1/16. As the light hits the target it again expands as a sphere originating from the target (assuming the target is reasonably diffuse), further weakening the signal. The loss of signal from the target back to the antenna also follows the inverse-square law. Thus the total loss of energy of the signal (E') from the antenna to a target at distance r and back to the antenna is

$$E' = \frac{1}{r^2} \cdot \frac{1}{r^2} = \frac{1}{r^4} \quad (2.2.2)$$

That means that the received energy at the antenna follows the inverse fourth-power law with respect to the distance of the target. The energy from a target at 2 kilometers is 1/16th that of a target at 1 kilometer. At 3 kilometers it is 1/81 of that and at 4 kilometers it is 1/256. To double the range of a radar, one would have to send 16 times as much energy! To compensate for this, the antenna itself amplifies the incoming signal by a factor of t^4 before it is sent to the

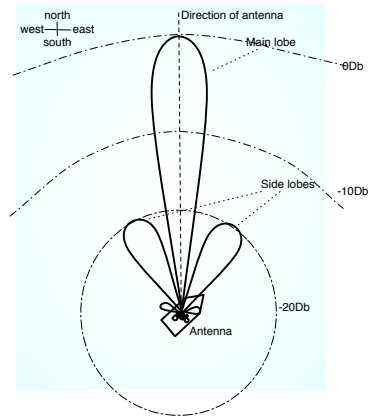


Figure 2.2.8: Example of a radiation pattern sent by a radar. The radial distance from the center represents signal strength relative to signal strength in the main direction. Plot is a cross section of the radiation pattern at antenna height. The signal is the strongest in the direction the antenna is pointing at (the main lobe) and moderately strong in two directions to each side (side lobes).

screen, where t is the time that past since the last send pulse. Note that we can use time t here, as range r and time t are related by the equation

$$t = r/C \tag{2.2.3}$$

where C is the speed of light.

2.2.4 Radiation pattern

As the antenna is directional, it tries to send the light in a specific direction. In the ideal case all the energy of the radar would be concentrated in a single vertical plane so that only targets that are directly in front of the antenna are detected. Unfortunately, it is a fundamental limitation of antennas that they leak radiation in all direction. The pattern in which they leak radiation is called the *radiation pattern* and depends on the physical characteristics of the antenna. The pattern is characterized by a series of lobes stretching from the antenna. (see figure 2.2.8).

The signal from the antenna is strongest in the direction it is pointed at. Moving radially away from this point the signal gradually gets weaker until it hits a point where no energy is sent at all due to light interference. This point is called the *null gap*. Moving further the signal gets stronger again. These areas are called *lobes*. The signal cycles between lobes and null gaps for 360° . The strongest lobe is in the direction the antenna is pointing at and is called the *main lobe*. All other lobes are called *side lobes*. The further a lobe is from the main lobe the weaker it is, with each lobe decreasing an order of magnitude in strength. Side lobes can cause adverse effects on the radar screen (see section 2.3.5). In this thesis we will only consider the two side lobes directly adjacent to the main lobe. All other lobes can be ignored due to their extremely weak energy output. Note that the radiation pattern determines the *strength* of the signal; the *speed* is constant in all directions (the speed of light).

The radiation pattern describes how much energy the antenna sends in each direction. But simultaneously also describes how sensitive the antenna is for picking up a signal from a direction. The antenna is for example much more sensitive for light coming from the direction the antenna

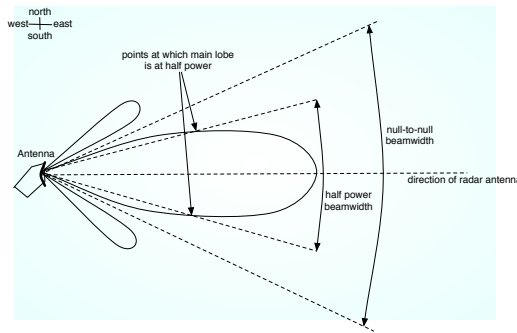


Figure 2.2.9: Top view of radiation pattern with two ways to measure horizontal beamwidth.

is pointing at, than for the same amount of light coming from the side or the back of the antenna. How sensitive the antenna is for a direction is perfectly described by its radiation pattern. The radiation pattern can be described by three parameters: antenna gain, horizontal beam width and vertical beam width.

2.2.5 Antenna gain

The gain of the antenna (denoted by G) is how far an antenna can send radiation compared to an isotropic antenna that radiates in all directions equally. This is expressed as an equation where for a given signal strength D in decibel, R is the distance that the signal from the directional antenna can travel before being reduced to D , and R_i denotes the distance that the signal from the hypothetical isotropic antenna can travel before being reduced to D , i.e.,

$$G = \frac{R}{R_i} \quad (2.2.4)$$

This ratio is the same for all D . A larger gain is better for an antenna as it can concentrate more energy in a smaller area, allowing it to pick up signals from further away. A larger gain means that less energy is wasted on the lobes other than the main lobe. The gain of the antenna is independent of the amount of energy that is actually transmitted. The amount of energy transmitted refers to how much energy the antenna can send in a single pulse. This is measured in kilowatts. Radar systems send between 2 to 10 Kw per pulse depending on the radar system[22].

Radars today have a 'Gain Control' which adjusts the threshold for which targets are visible (see figure 2.2.3). It is important to note that this does not refer to the antenna gain. The name comes from the early days of radar, when it controlled the gain of an amplifier that crudely boosted the signal allowing more targets to be seen.

2.2.6 Horizontal beamwidth

The horizontal beamwidth is a measure of how narrow the main lobe is. The narrower the main lobe, the more accurate and well-defined the radar picture is. This property can be measured in two ways which are depicted in figure 2.2.9.

The *null-to-null beamwidth* defines the beamwidth in terms of the null gaps between the main lobe and the side lobes. If the null gaps are for example 5° to the left and right of the main lobe then the total null-to-null beamwidth will be 10° .

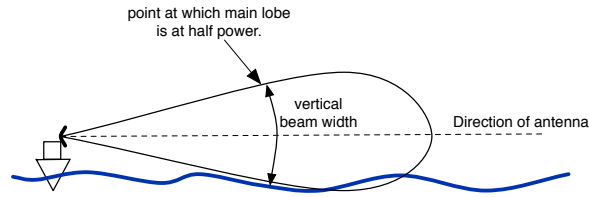


Figure 2.2.10: Definition of vertical beamwidth.

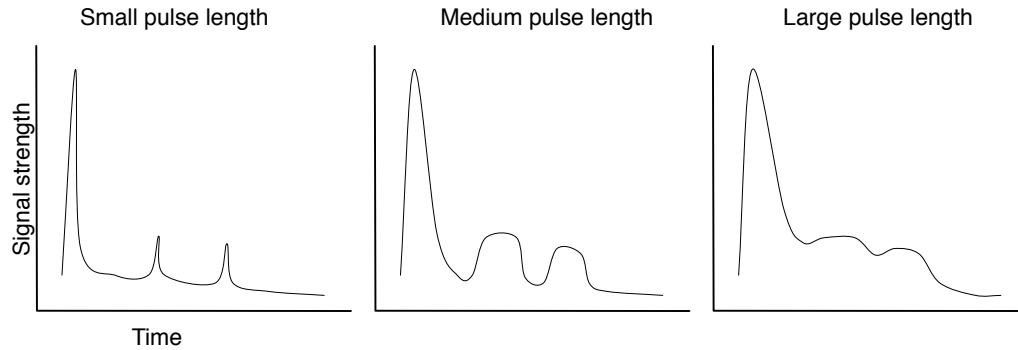


Figure 2.2.11: Difference of incoming signal with different pulse lengths in a situation where there are two targets at different distances in front of the radar.

The *half-power beamwidth* defines the beamwidth in terms of the power output of the main lobe itself. In the horizontal center of the main lobe the power output is at its highest; as one drifts away from this center towards a null-gap the power output decreases to zero. The half-power beamwidth is defined as the distance between the two points where the power output is exactly between full power and zero power.

The half-power beamwidth is the most common beamwidth representation. A typical radar has a half-power beamwidth of around 0.65° to 5° [10]. The beam width is important for simulating artifacts that are seen on a radar screen as will be explained in section 2.3.2.

2.2.7 Vertical beamwidth

Aside from its width the main lobe also has a height. This is called the vertical beam width of an antenna, and just like the horizontal beamwidth, it is defined by the half-power beamwidth (figure 2.2.10).

The vertical beamwidth is less important than the horizontal beamwidth as it does not introduce as many artifacts, to the radar image, as the horizontal beamwidth does. Most radar systems have a vertical beamwidth of around 15° to 30° [10].

2.2.8 Pulse length

A radar cannot send a light pulse that is infinitely short. The antenna always needs some time to warm up, stay on and then cool down. The time that the antenna is switched on, determines the thickness of the shell of light that leaves the antenna. This thickness is known as the pulse length

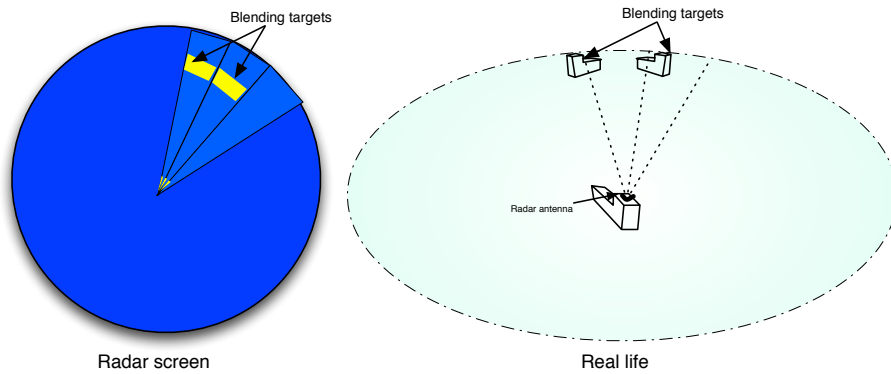


Figure 2.2.12: Due to a pulse frequency that is too low, the image on the screen has a resolution that is too low to differentiate between two little ships or one large ship.

of the radar. The larger the pulse length the more energy that is returned and the easier it is to detect targets. But unfortunately there is also a trade off as the pulse length simultaneously determines the effective screen resolution of the radar screen.

If two targets are behind each other, a radar pulse can differentiate them because the signal bounces off them at different distances. But, if the pulse length increases, the distance between the two return signals would decrease until the two overlap (see figure 2.2.11). At that point the two targets would appear as a single target on the radar screen. The pulse length can also not be set too small as then the energy returned to the antenna will not be sufficient for the radar to detect and display its target(s).

In practice the pulse length determines the effective resolution of the screen in the range direction. If for example the range is set to 1000 meters, and the pulse length is 10 meters, then one can get a resolution of 100 pixels on the screen. A typical pulse length on marine radar is 15 to 300 meter. [6].

2.2.9 Pulse frequency

The antenna can only send a finite amount of pulses in a given amount of time. To get the best image the radar should send as many pulses per revolution as possible. Typically each target should be illuminated by the radar beam at least more than once to increase the chance of detecting small targets. There are a number of different ways to express the pulse frequency; In this thesis the combination of revolution time (how many seconds it takes for the antenna to rotate 360°) and pulses per revolution (the amount of pulses the antenna transmits in one 360 degree revolution) will be used.

The pulses per revolution on radar systems vary wildly from 10 to 10000 pulses per revolution depending on the application that they are used for. Marine radar systems will vary between 300 to 600 [7].

If the pulse frequency is too low it will not only miss certain targets but also decrease the fidelity of the image. In exactly the same way that a pulse length that is too high will blend together ships that are close behind each other, a pulses-per-revolution that is too low will blend together ships that are next to each other (see figure 2.2.12).

In practice, the pulse frequency determines the resolution of the screen in the angular direction. The more pulses the antenna sends, determines the more detail the picture has in the angular direction.

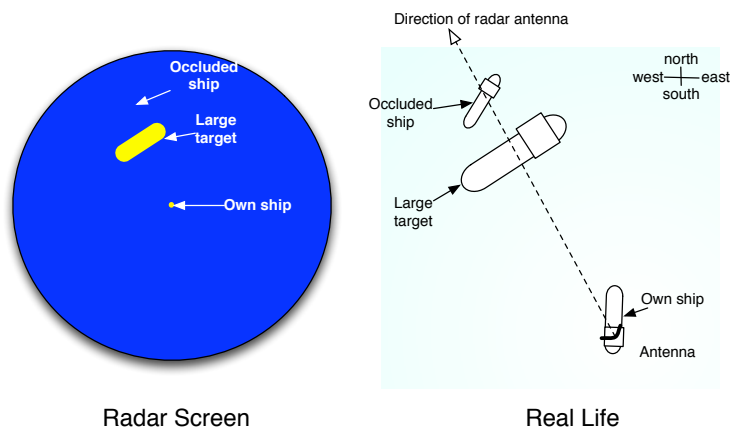


Figure 2.3.1: Small ship is occluded by a larger ship. The radar screen does not show the small ship.

2.3 Radar effects

A radar system provides mariners with a map of the environment; but it comes with limitations. Radar operators are trained in how to correctly interpret a radar image and ignore any clutter [21]. This section describes the limitations and clutter that a radar screen has.

2.3.1 Occlusion

Just like regular light, radar light cannot pass through solid objects. This is a problem if there are targets behind large solid objects. The radar light cannot reach those targets and thus they do not show up on the radar screen. Those targets are *occluded* from the radar.

In figure 2.3.1 a small ship is behind a large freight ship. As the light from the radar cannot penetrate the large ship and reach the small ship, it will not be detected by the radar. The occlusion effect is one of the most important effects of a radar screen.

Occlusion can also occur due to the objects on the ship carrying the radar. Often there is a large mast with communication equipment near the radar system that is within the "line of sight" of the radar antenna. If a target is exactly behind the mast from the point of view of the antenna, the target may not appear on the radar screen. The areas on the screen that are occluded by structures on the ship itself are called *shadow sectors* (figure 2.3.2).

2.3.2 Echo scaling

Targets appear to be slightly bigger on the radar screen than they are in real life. This is due to the fact that the radar beam can hit a target even before the radar is exactly oriented towards it. Due to the radiation pattern the beam itself has some width and can reflect on a target that is not directly in the view line of the radar. This causes the radar to already receive echoes from a target before the radar is directly facing the target. On the radar screen this results in targets being already drawn before the radar is facing them. This causes the target to appear a bit bigger than it actually is.

The amount of echo scaling is dependent on the horizontal beamwidth of the radar. If the beam is wider the echo will appear to be bigger. Consider for example the situation in figure

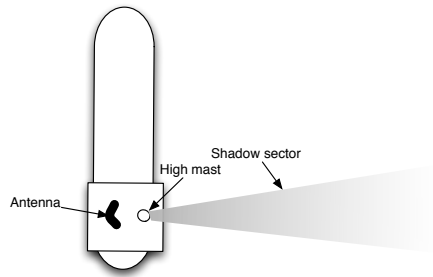


Figure 2.3.2: Area invisible to the radar due to a mast obstructing the view of the antenna.

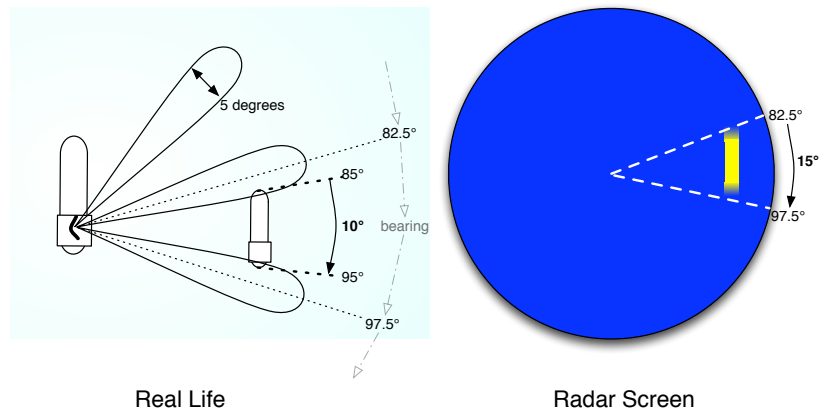


Figure 2.3.3: Ship in real life is visible from 85° to 95° due to the echo scaling effect. In real life the radar detects it only from 82.5° to 97.5° .

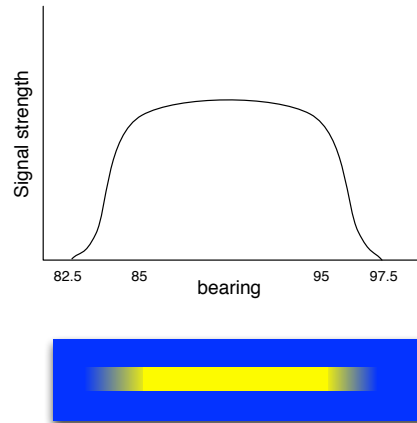


Figure 2.3.4: Signal strength per bearing from example in 2.3.3, with resulting echo on radar screen which has fuzzy edges.

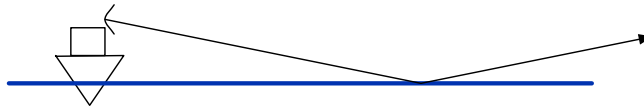


Figure 2.3.5: Light bounces off of the sea.

2.3.3, where the beam has a horizontal beam width of 5° and there is a target that starts at bearing 85° and ends at bearing 95° . The beam will start to intersect the target when the radar is rotated at 82.5° and will stop to intersect at 97.5° . The echo on the screen will appear to occupy $97.5 - 82.5 = 15^\circ$ of the screen, while in reality it should only occupy $95 - 85 = 10^\circ$ of the screen.

The scaling term is linearly-dependent on the horizontal beam width. If the horizontal beam width is x degrees then the target will appear to be x degrees bigger than it really is: it will appear to be $x/2$ degree bigger on its left side and $x/2$ degree bigger on its right side.

The extra echoes will not be sharply defined, see figure 2.3.4. The amount of light that returns when the beam barely intersects the target is considerably less than when it fully intersects the target. This results in the echo appearing to 'fade in' as more light is received from the target as the beam intersects more with the target. The same holds when the target rotates out of view and the beam will slowly stop intersecting with the target.

On a real radar screen this gradient effect is hard to observe because of the limited amount of pulses the radar sends. A target a kilometer away is typically only hit by 5 to 7 pulses [6]. While the scaling effect will still be noticeable, the fuzzy edges effect will often not be noticed due to only being as large as 1 or 2 pixels on the radar screen.

2.3.3 Sea clutter

Radar light reflects on everything that it hits, including the surface of the sea itself. Obviously one would not want to see the sea itself on the radar screen as it would color almost the entire radar screen yellow, making it impossible to see the targets.



Figure 2.3.6: Light bounces on a nearby wave.

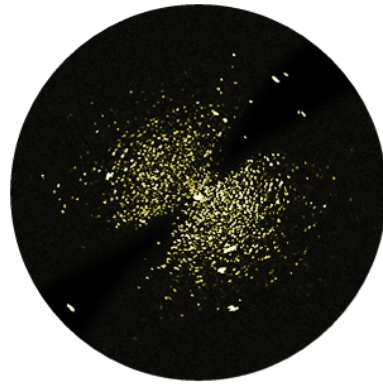


Figure 2.3.7: Sea waves being visible on an old radar screen.[27]

Luckily the sea hardly reflects radar light to the antenna. The surface of the sea is highly specular, usually very flat and always looked at from a shallow angle. This causes most of the light to bounce off on it and not back to the antenna (figure 2.3.5). In essence the surface of the sea acts like a giant mirror.

If the conditions are more windy, the surface of the sea will have waves on it. If the surface of a wave is steep enough, radar light can be reflected by the wave, back to the antenna. These echoes are known as *sea clutter* and appear as a ring of clutter around the center of the radar screen (figure 2.3.7). The effect is the most visible near the center of the radar screen and extends outward depending on the roughness of the sea. The surface of most distant waves is too shallow for radar light to be reflected to the antenna.

The direction of the wind strongly influences the sea clutter. The sea clutter appears more prominently in the direction that the wind is blowing. The reason for this is that the wind influences the shape of the waves (figure 2.3.8).

The wind pushes the wave in one direction, which creates a steep surface on one side and a more shallow surface on the other side. The steep side reflects radar light much better to the

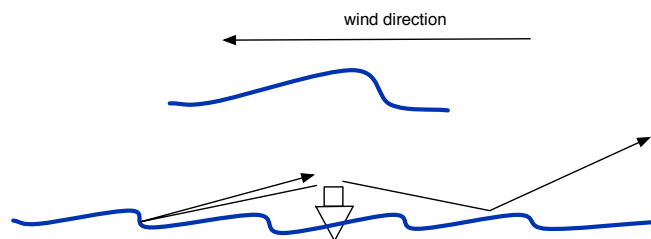


Figure 2.3.8: Wind influence on wave shape and radar reflectivity

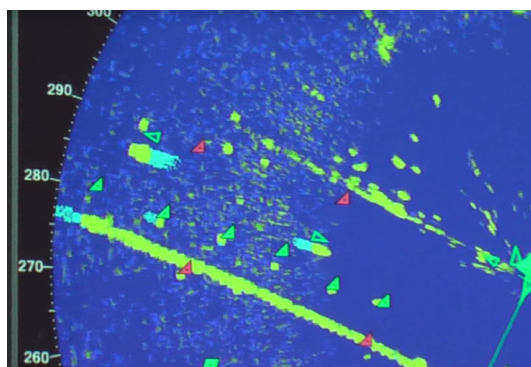


Figure 2.3.9: A haze of clutter indicating a local rain shower

radar antenna making those waves appear more prominent on the radar screen.

2.3.4 Rain clutter

Not only water in the sea can reflect radar light, but also water in the air. Rain droplets have a very small surface area and reflect only a minimal amount of radar light. But if they appear in large enough numbers (for example during a rainshower), they can collectively reflect a significant amount of radar light back to the antenna - enough to be seen on a radar screen, see figure 2.3.9.

Rain clutter on the radar screen can be recognized by a constantly changing haze of echoes that obscures the real targets. While the ability of radar to pick up rain droplets is used in other radar applications [23] it is undesirable for a marine radar system. The effect can be partly suppressed by the rain clutter reduction feature on these systems (see section 2.4.3).

2.3.5 Side lobes

The distinctive shape of the radiation pattern (section 2.2.4) also influences the radar screen. Most of the echoes that come back to the radar antenna come from the main lobe. This is the largest part of the radiation pattern and it is the direction the radar antenna puts most of its energy in, see section 2.2.4).

Like all antennas, the radar antenna cannot be purely directional. It will radiate light in all directions. Most of the light that does not go into the main lobe will go into the side lobes. These are the two direction to the left and right of the radar antenna that also radiate light, although with a much lesser intensity than the main lobe. Between the side lobes and the main lobe, there is a small area where no light is radiated, called the *null gap*. This is the part of the radiation pattern where the light interferes with itself and cancels itself out.

The intensity of the side lobes is so low that the amount of light that comes from a side lobe and echoes back is negligible and has almost no effect on the radar screen. Under certain situations however, the existence of side lobes do have an effect. If a target is sufficiently large, reflective and close, the light reflected from the side lobes will start to have an effect.

As the radar rotates, the side lobes will be the first to pick up echoes of a target, well before the main lobe even intersects the target. After the radar rotates further, the target will be in between the side lobe and the main lobe, in the null gap of the radiation pattern. At this point no light hits the target and nothing is drawn on the radar screen. As the radar rotates even further, the main lobe hits the target, and the real echo is drawn on the radar screen.

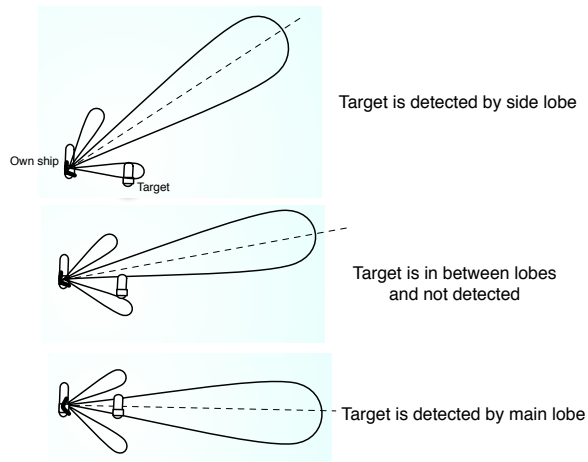


Figure 2.3.10: Effect of side lobes on a real radar system.

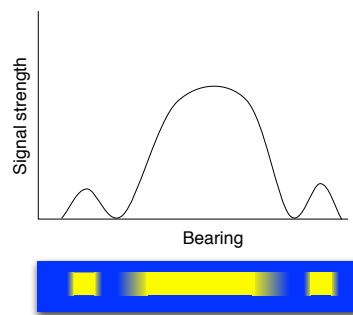


Figure 2.3.11: Amount of light returned as radar rotates past a close target.

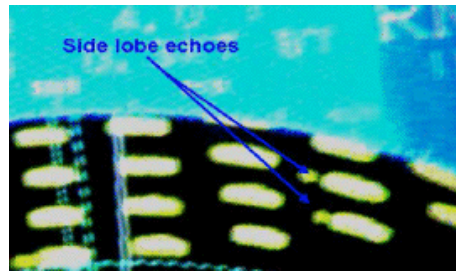


Figure 2.3.12: Effect of side lobes on a real radar system.[28]

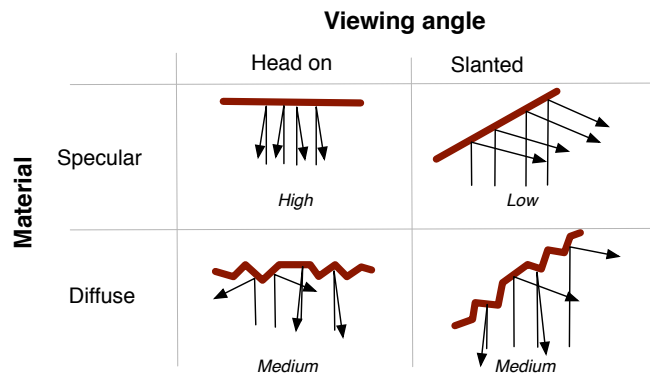


Figure 2.3.13: Light reflectivity of an object depends on the viewing angle and specularity of the object.

The effect on the radar screen is that a target appears to be surrounded by little ghost targets, to its left and right (figure 2.3.12). If the amount of pulses from the radar is too low, the radar might miss the pulse where the target is exactly inbetween the side lobe, and thus, the main lobe and the ghost targets will blend together with the real target, creating an echo that is larger than the real target.

2.3.6 Specularity

The reflectivity of an object does not only depend on its size but also on the material that its made of and the viewing angle. If an object is *specular*, it tends to deflect all incoming light from a single light source in a single direction. If the surface of an object is oriented towards a light source it will reflect most of the light back to the source, whereas, if it is viewed from a slanted angle it will reflect most of the light away from the light source. If an object is not specular, it is considered *diffuse*. Diffuse objects tend to scatter incoming light in all directions, regardless of the position of the light source or the angle of the material.

The level of visibility of an object in the radar image depends on its material and the viewing angle. A ship (which is likely a specular object) can appear more prominent than a similar sized diffuse object when viewed head on. But can at the same time appear less prominent than the diffuse object when viewed from a slanted angle (figure 2.3.13).

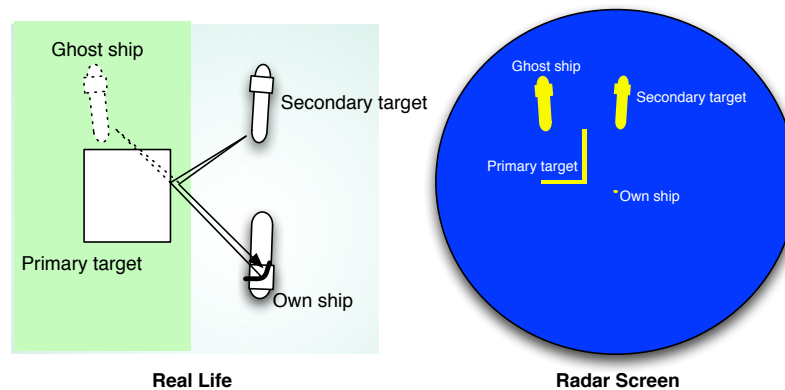


Figure 2.3.14: The radar detects a ghost ship due to radar energy bouncing from a primary target to a secondary target and back.

2.3.7 Specular reflections

So far the only type of radar light reflection considered is the *direct reflection* where the radar light from the antenna bounces against a target and returns directly back to the antenna. If the light hits a target that is large and sufficiently specular, it can bounce via this target to a secondary target and back again to the antenna via the primary target. This type of reflection is called *specular reflection*. The secondary target is viewed via the primary target. The primary target functions as a mirror for the radar light (figure 2.3.14).

On the radar screen, the secondary target will appear behind the primary target, while in reality it is not there (figure 2.3.14). This only occurs in specific situations. The primary target must reflect a large amount radar light for the secondary target to be detectable. To do this it must be large, otherwise there is not enough surface area for the radar light to bounce against. It must be relatively specular, otherwise all the energy hitting the primary target will be scattered in all directions, and not enough light will reach the secondary target. Further, it must be close, so that a relatively large amount of light can strike the primary target. In addition the secondary target has to be relatively large or reflective to reflect enough radar light back to the antenna.

2.3.8 Second Trace

As shown in figure 2.2.7, there can be ambiguous targets on the screen, if the pulse frequency is chosen too high. To resolve this, the radar only sends out a new pulse when it can reasonably assume it will not detect any more targets from the last pulse. The fact that the earth is round puts a fundamental limit on how far a radar can "see". If a target is below the horizon, it is unlikely for light from the radar to go to that target and back, as it is not in direct line-of-sight of the antenna (see figure 2.3.15).

There are exceptional circumstances where light from the antenna can detect targets that are under the horizon. In normal atmospheric conditions the air is relatively warm near sea level and gets colder the higher it is. However, the reverse condition may exist. When two weather fronts collide cold air can get temporarily trapped beneath a layer of warm air. This situation is called an *inversion*. When light travels along the cold layer of an inversion, it can get trapped beneath the warm layer. The warm layer has a higher refractive index and bends the light back down towards the cold layer. This way light can travel further than it normally would, and it

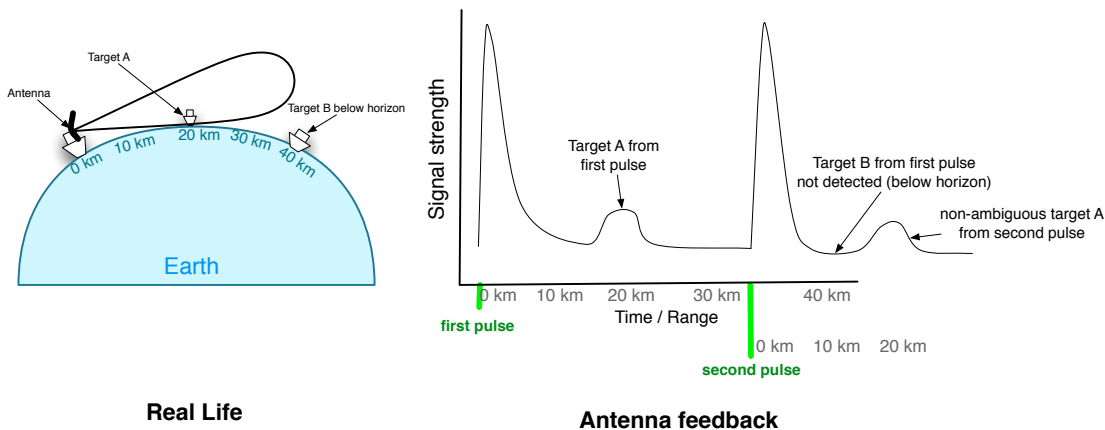


Figure 2.3.15: Effect of horizon on the pulse frequency. After 30 kilometers the horizon blocks the line-of-sight to almost all targets. After waiting for the time it takes for light to travel to and from a target at 30 km ($0.0002\text{seconds} = \frac{2 \cdot 30\text{km}}{C}$, where C is the speed of light, 299.792.458 seconds per meter), it is safe again to send a new pulse.

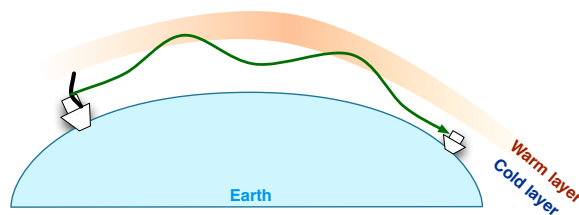


Figure 2.3.16: Atmospheric ducting refracts the radar light back down again, allowing it to reach a target over the horizon.

can easily go over the horizon and back. This phenomenon is known as *atmospheric ducting* (see figure 2.3.16).

During an atmospheric inversion, targets that normally should be out of range are suddenly detected by the antenna. And because of their great distance, they are detected after the next pulse has already been sent by the antenna. Signals that return from the previous pulse are called *second trace echoes* and give a distorted image of targets that are beyond the usual maximum range of the radar.

Because the radar interprets the signal as being from the last pulse sent, it draws targets that are at distance $MAX + D$ at distance D , where MAX is the maximum range of the radar, resulting in a distorted image (see figure 2.3.16). Second trace echoes mostly occur if there are very reflective objects out of range such as large cliffsides or hills. It is unusual that enough light is returned from ships or similar-sized objects to be visible in the second trace.

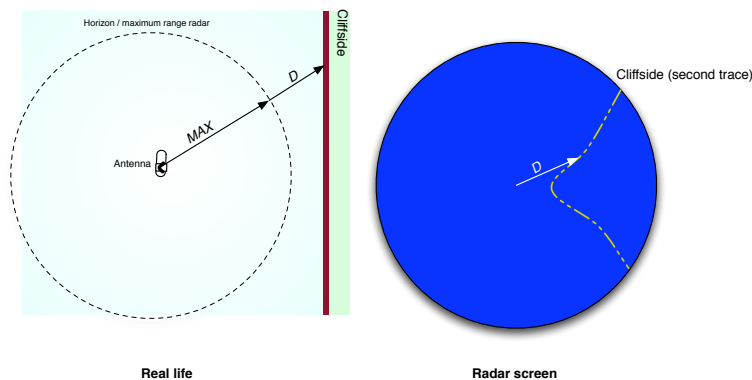


Figure 2.3.17: The cliffside is picked up by the radar, but only after its next pulse has already been sent, thus tricking the radar into thinking the cliffside is closer than it actually is.



Figure 2.4.1: The three controls that can adjust the radar image.

2.4 Screen

A radar operator does not just have to look at the radar screen: he can also adjust it himself. A radar system comes with numerous options that draw important overlay information on top of the radar image that aids the operator in avoiding traffic collisions and improves the usability. While most options only add overlays or apply trivial filters, such as brightness or night colors, there are three controls that adjust how the radar image itself is processed. These are the Gain, Rain and Sea controls.

2.4.1 Gain Control

In the early days of radar there was an extra amplifier between the antenna and the screen that could boost the signal to allow more targets and clutter to be seen. The gain (amplification) of this amplifier could be controlled by the radar operator to get a satisfactory image that would show all the targets, but with as little clutter as possible.

In modern radar systems the amplifier has been replaced by digital signal processing but the gain control remains. The name is now only an artifact as the traditional amplifier has disappeared. It should not be confused with the gain of an antenna (section 2.2.5) which is a similarly-named, but otherwise unrelated concept.

Today it controls the threshold used for determining if a pixel should be colored blue (nothing) or yellow (target). As discussed in section 2.4, the screen gets a feed from the antenna with information about how far the targets are. The screen then has to determine which signals are strong enough to be drawn on the radar screen. This is done by means of a threshold. All values above the threshold are drawn as a target on the screen while all values below it are not.

With the gain control a radar operator can adjust the threshold to allow fewer or more targets to become visible (figure 2.4.2). The higher the gain, the lower the threshold will become. On the radar screen the setting of the gain control is represented as a rectangle that fills up as the

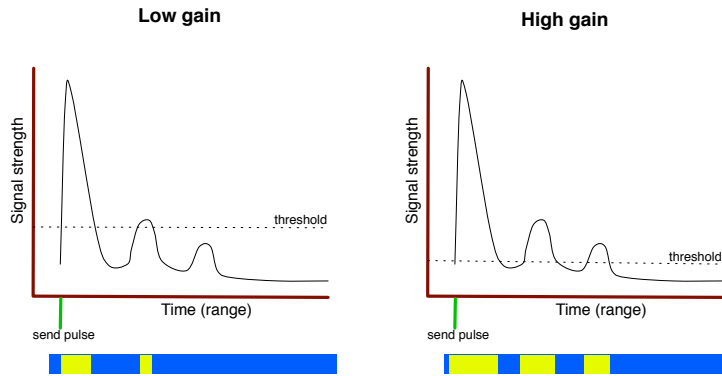


Figure 2.4.2: A high gain (lower threshold) allows weak targets and clutter to be more visible.

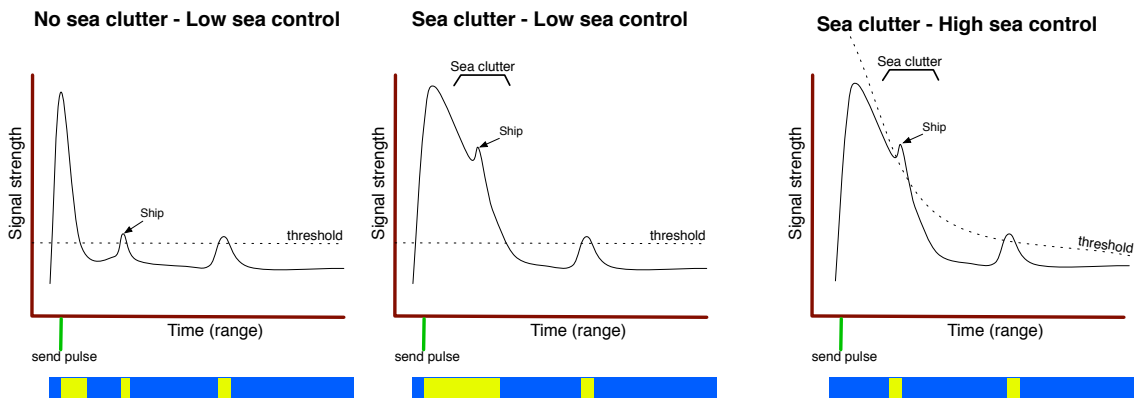


Figure 2.4.3: A higher sea control allows targets among the sea clutter to be seen again.

gain is set higher (figure 2.4.1). The minimum and maximum gain settings are unknown and differ from radar system to radar system. A high gain makes the screen 'more yellow', as more and more targets and clutter are above the threshold. Radar operators strive for a gain setting where other ships are clearly visible, but where random clutter is still too weak to be above the threshold.

2.4.2 Sea Clutter Control

As discussed in section 2.3.3 sea clutter represents reflections from nearby waves that are strong enough to be displayed on the radar screen. These reflections can be strong enough that they obscure any real targets such as ships and coastlines. With the sea clutter control, a radar operator can remove most of this clutter and allow only real targets to be displayed. The defining feature of sea clutter is that it appears strong near the antenna itself and quickly diminishes with distance. This fact is used by the sea clutter control to reduce only echoes that are close to the radar. Like the gain control, the sea control adjusts the threshold line that determines if a signal is drawn on the screen. But instead of adjusting the height of the entire line, the change in height is dependent on the range. (figure 2.4.3).

The curve applied by the sea clutter control is the inverse exponential where the control

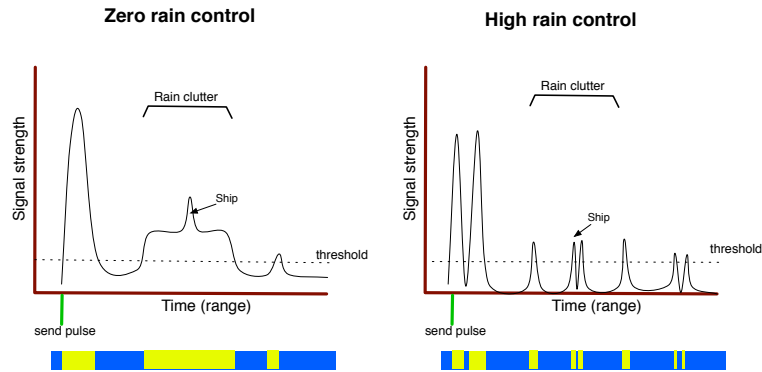


Figure 2.4.4: Taking the absolute derivative of the signal can reveal targets among the clutter.

influences the exponent. It can be summarized with the equation:

$$T = C \frac{t}{R^s} \quad (2.4.1)$$

where T is the new value of threshold t at time/range R , given a sea control setting s . C is a constant that differs per radar system and determines how strong the effect is. The minimum sea-control setting is usually 0, equating to no change in the threshold value. The maximum setting differs per radar system.

2.4.3 Rain Clutter Control

Rain clutter (section 2.3.4) is characterized as an area on the screen where there are, on average, more reflections that obscure the real targets. This typically happens if there is a local rain shower that reflects radar light. The screen will show rain showers as haze obscuring the real targets in the area (figure 2.3.9).

The gain control is often difficult to apply in this situation. A lower gain would reveal real targets in a rain shower, but can simultaneously make very weak targets outside the rain shower invisible. Preferably the gain should only be lower inside the rain shower itself. The rain control tries to resolve this problem by taking the absolute value of the derivative of the signal. Given that the original threshold is defined by the function $f(x)$, the threshold with rain control applied $f_r(x)$ is

$$f_r = |f'(x)| \quad (2.4.2)$$

This derivate curve shows peaks whenever the original curve goes up or down and stays zero whenever the original curve remains constant (example in figure 2.4.4). The result is that large areas of uniform clutter only show up when they start and end. And that targets within the clutter can be clearly seen again.

The control itself can be configured from fully on to fully off. Every value inbetween is linearly interpolated between the original curve and the derivate curve. Though the rain control can be useful, it also has its disadvantages. Applying the rain control can make very large echoes or echoes that only gradually increase in intensity appear to be weak. For these reasons it is in practice not used as much as the gain and sea control.

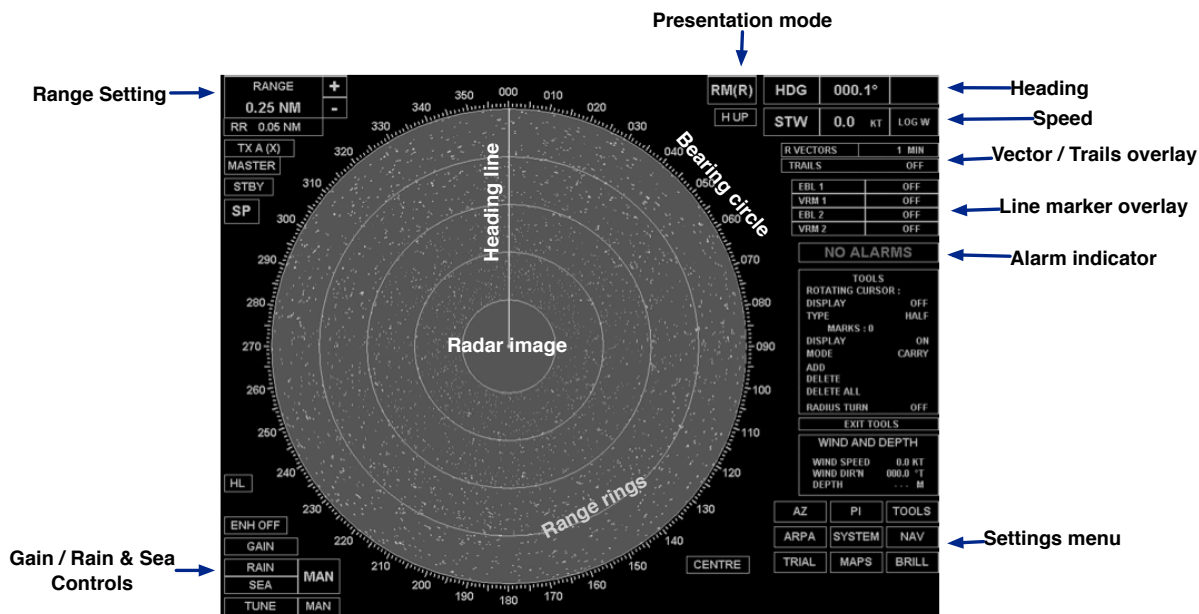


Figure 2.4.5: Additional information on radar screen

2.4.4 Additional information

A radar image is round, while the radar screen is rectangular. This leaves some extra room for radar systems to display important additional information, see figure 2.4.5.

Range Setting Allows the operator to indicate how 'zoomed in' the picture should be. Indicated as the distance (in nautical miles) between the center of the screen and the edge of the screen. 1 nautical mile = 1852 meters.

Presentation mode On a radar screen the location of the radar antenna is typically fixed in the center, while the coastline and other targets move relative to it. This mode is known as Relative Motion (RM). Most screens can also be configured for True Motion (TM); in this mode the coastline and other stationary targets remain fixed on the screen while ones own ship moves across it. If the ship is almost off of the edge of the screen, the image will re-center on it. Most radar operators prefer Relative Motion. Another presentation aspect is how the screen should be rotated. An operator can choose between Head Up and North Up mode. In Head Up mode the top of the screen always corresponds to the direction the ship is heading, while in North Up the top of the screen is the direction to True North.

Heading - The heading is the course the ship is currently on, with respect to the true north. This information is not calculated from the radar image, but comes from a gyrocompass situated elsewhere on the ship.

Speed - Speed indicates how fast the ship is moving with respect to the ground (Speed Over Ground, SOG), or with respect to the water (Speed Through Water, STW). As with the heading, this information comes not from the image but is fed from a GPS or Doppler device.

Vectors - For a radar operator it can be difficult to assess from a static image if another ship is on a collision course. Only by analyzing sequential radar images a guess can be made on the speed and course of another ship. All radar systems today have a feature that does this analysis and can show this information about targets. The vector overlay shows lines extending out from

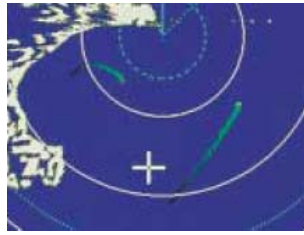


Figure 2.4.6: Radar image with trails indicating course of other ship.

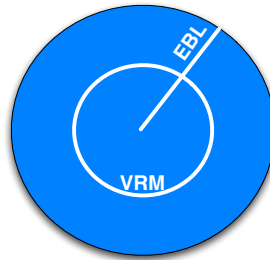


Figure 2.4.7: Electronic Bearing Line and Virtual Range Marker help to find the distance and bearing to locations on the screen.

the ship whose directions indicate the ships heading and whose length indicates the ships speed.

Trails - Past positions of another ship are often a good indicator of where it will be in the future. The trails feature can show up to a few minutes of past images beneath the current radar image. The trails images are a darker color to differentiate them from the current image.

Line marker overlay Sometimes it is necessary for a ship to stay a minimum distance away from something, or be within a certain angle of something. In these cases it can be handy to overlay helper lines on the radar image that can be configured on certain distances or angles. The Virtual Range Marker (VRM) displays a circle on the screen centered on the antenna at a given distance, while the Electronic Bearing Line (EBL) shows a line extending from the antenna to a given bearing (figure 2.4.7).

Chapter 3

Radar Simulation

A radar simulator is an algorithm that takes as input all the information about the radar that it is simulating (both the radar itself and its environment) and outputs a visualization of the radar screen. There are two important goals to this:

- *Realistic.* The visualization of the radar screen should be as close as possible as what a real radar would show. While it is not possible to make a pixel-perfect replica due to the often random nature of rain and sea clutter it should be within a reasonable margin. A trained radar operator should not be able to tell the difference.
- *Real-time.* A real radar updates the contents of its screen every few seconds. A radar simulator should be able to update its contents at least as fast as that on a high end machine.

We can summarize the concept of a radar simulator as follows: A radar simulator is an algorithm that provides a realistic real-time visualization of a real radar screen. Given identical inputs and parameters a radar simulator and the radar it simulates must give reasonably comparable outputs.

3.1 Input

The input for the algorithm will be all the data that a real radar antenna also has access to. These can be split in three different categories: Environment, Radar & Ships (figure 3.1.1).

Environment

The environment is a three-dimensional mesh that includes everything that a radar antenna could reasonably pick up. This includes coastlines, buildings, islands, piers and anything else that is solid that could be picked up by the radar. For every surface in the mesh the specularity is included as it can have a significant impact on the final visualization (section 2.3.6).

The environment is static, it is something that will not change during the course of the simulation.

Ships

The ships are all the vessels that traverse the environment. Each ship has a specific location, heading, bearing, course and speed which the simulator has to keep track of to realistically move each ship along its course. Every ship also has a 3d mesh associated with it that influences how light is reflected against it.

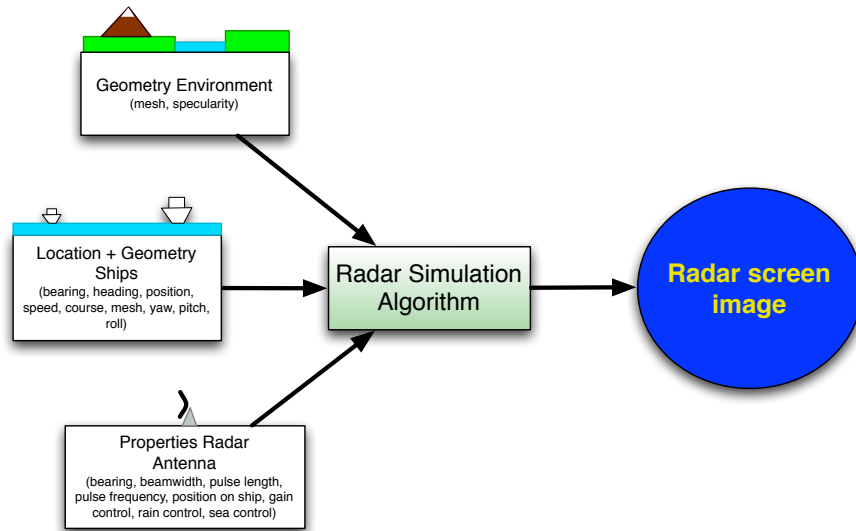


Figure 3.1.1: Overview of the input and output of a radar simulation algorithm

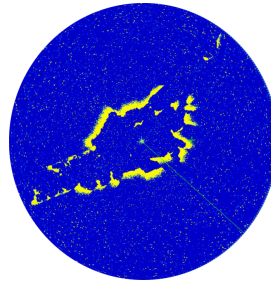


Figure 3.2.1: Example of a visualization to output.

Radar

The last thing that is needed to simulate a radar screen is the parameters of the radar system itself. The most important thing is its location and direction within the environment but also the properties of the antenna are required, such as gain, beam width, pulse length and pulse frequency (section 2.2). These also include the parameters that a radar operator can adjust for himself: gain control, rain control and sea control.

3.2 Output

The output of the radar simulation algorithm should be the round radar screen as seen on a real radar system (figure 3.3.1). Like in real life it should continually update itself in a sweeping clockwise direction across the screen to reflect changes in the position of the radar antenna in the environment as well as changes in the location of the surrounding ships.

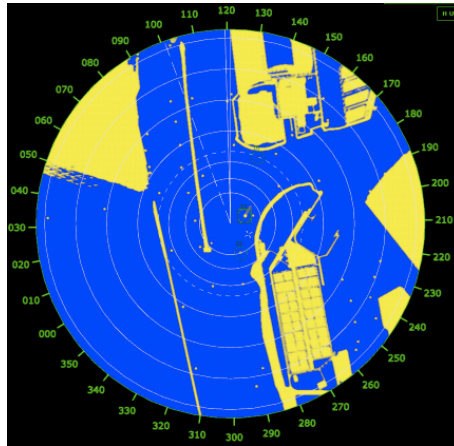


Figure 3.3.1: Naive radar unrealistically displaying a map.

3.3 Related Work

Several attempts at making a realistic real time radar simulation have already been made. In this section each simulation will be given a description along with their limitations.

3.3.1 Naive algorithm

The simplest way to simulate a radar image is to simply draw the geometry directly with a yellow color on a black background. Or if speed is really of the essence one can display a prerendered map of the environment.

This is the most simple and fast way but has little to do with actually simulating a radar system. As all it does is display a circular area of a map. It is still included in this thesis as it is actually used commercially to simulate radar systems. In situations where only the textual information from a radar is of real importance or there is limited computational power available this option is usually chosen. Naturally the naive algorithm supports none of the radar effects. Effort has been made to make it more realistic by at least pretending to have the occlusion effect.

One can take the image from the naive algorithm and apply an edge detection filter on it (figure 3.3.2). The edge detection on the original image would not work as yellow buildings on yellow land would not produce any edge where they should in real life. But when used on the height map the buildings stand out quite clearly from the surrounding landmass (figure 3.3.2). With only the edges visible the image will no longer show an echo on the inside of the building. Though it still erroneously display the backside of the building. This problem is not as noticeable in cases where the backside is either very far away (coastline of landmasses) or very nearby (ships). The algorithm fails to produce a correct image in many situations and is not suitable if the goal is to create a realistic radar image.

3.3.2 Radar Signal Simulation

To recreate the image that a radar produces one needs to simulate more accurately what the radar antenna does. It sends out a pulse of light and then listens for any of it to return as they reflect and scatter against the surfaces in the environment. The time it takes for the light to return can be used to estimate the distance to targets in the environment. A radar simulation

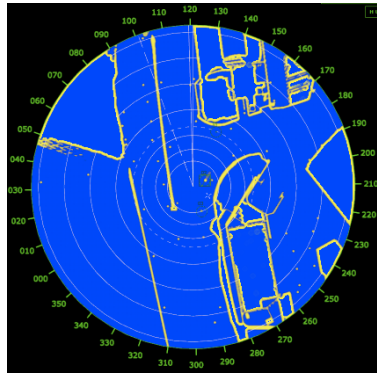


Figure 3.3.2: Edge detection on naive radar allows for a slightly more realistic result.

algorithm should accurately mimic this situation and all the effects that are a consequence of it (section 2.3).

One algorithmic possibility is radiosity, which is a global illumination algorithm in computer graphics that is well known for its realistic results [26]. The radiosity algorithm was at the basis for the Radar Signal Simulation algorithm[3]. The reason this was chosen is because illuminating a three dimensional scene is very much like creating a radar image. In the radiosity algorithm the environment to be lighted is divided into small patches that can send and receive light. At the start of the algorithm a few patches are designated as 'light sources', these will constantly radiate light to other patches. After that all patches will receive and send light to eachother in a number of passes. The end result is that each patch will contain the amount of light it should have when compared to real life (a more detailed explanation of the radiosity algorithm is given in Appendix B). We can use the radiosity algorithm in the same way for a radar simulator. The light patch will be the radar antenna and the environment will be the surrounding ships, coastlines, houses and all other objects that can reflect light.

After running the radiosity algorithm with these settings the lighting information from the radiosity algorithm needs to transform back into data that can be drawn as a radar image. In the final pass of the radiosity algorithm the returned light is collected back into the antenna patch. But instead of only collecting the amount of light also the distance from the originating patch is taken into account. This distance together with the amount of light of the originating patch can be collected as an antenna feed like in figure 2.2.3. The end result of running the radiosity algorithm is therefore only one slice of the radar image. To make a complete radar image the radiosity algorithm has to be executed for every direction.

Specularity Using the radiosity algorithm as a basis for lighting the scene has a few disadvantages. The most important feature missing is specularity [11]. While most coastlines and houses are fairly diffuse reflectors, the metal ships are often highly specular. The hull of a ship is much more visible on a radar screen if it is seen face on, when light bounces back to the antenna in the specular direction, then when it is seen from an angle. Also large flat features such as the flat frontside of an office building tend to exhibit specular behavior [9].

To compensate for this the Radar Signal Simulation algorithm extends the radiosity algorithm with features from raytracing. When a patch sends radiation to another patch it is not only added to the light value of the patch but also the direction it came from is stored. This ray of radiation is called a *signal* in the algorithm. As a signal is sent out again from a patch it divides itself into multiple directions with the most prominent direction being the specular direction (figure 3.3.3).

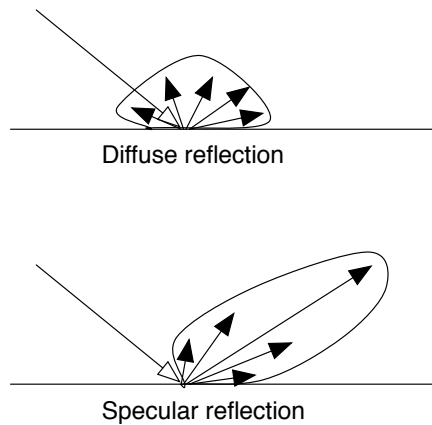


Figure 3.3.3: Outgoing signals for two patches that have a different specularity level.



Figure 3.3.4: Top view of a collection of antenna patches in the shape of the radiation pattern; patches with a warmer color give off more light. [3]

Each patch can have a specularity value that determines if it is very specular (in which case almost all radiation is sent out in the specular direction) or not specular at all (in which case it behaves exactly as a diffuse reflector like the original radiation algorithm).

Occlusion Using the radiosity algorithm ensures that shadows are calculated in a realistic fashion. If an area is behind another object from the point of view of the antenna light source, the radiosity algorithm will make sure that no light can travel to that area directly. But shadows are only as detailed as the size of the patches. If the patch size is chosen too big the radar image becomes inaccurate with light bleeding into areas that should be occluded on the radar screen.

A tradeoff has to be made between accurate shadow areas (small patches) and fast execution (large patches).

Radiation pattern In order to create a realistic radar image the radiation pattern (section 2.2.4) has to be taken into account. This is done in the Radar Signal Simulation algorithm in the most realistic way possible. Instead of modeling the antenna as a single antenna patch it is modeled as several dozen patches in the shape of the radiation pattern. Each patch in this radiation pattern transmits the appropriate amount of light; for example, a patch at the peak of the radiation pattern transmits more light than the radiation that goes to the sides (see figure 3.3.4). The same patches are also used when receiving the light but in reverse, patches that transmitted a large amount of light are now more sensitive to incoming light than average. This simulates the fact that a radiation pattern of an antenna also defines how sensitive the antenna is for incoming signal from each direction.

The effect of sidelobes can also be accurately be simulated with this. Sidelobes are nothing more than parts of the radiation pattern, and they can be modeled in the same way (figure 3.3.5).



Figure 3.3.5: Top view of a radiation pattern with side lobes.

Limitations The addition of signals has a significant impact on the *running time* of the algorithm. For every direction the radiosity algorithm has to be executed, which itself is a computationally expensive operation. The total complexity of the radiosity algorithm depends on a the following factors:

- N , The number of patches in the scene
- P , The amount of passes of the algorithm

For every pass one needs to send the light of every patch to every other patch. In practice patches will only be able to send light to the other patches that it can see. This set is in most circumstances significantly smaller than the set of all patches. There are however worst case scenarios where every patch can see every other patch, for example if they are arranged on the inside of a sphere. The sending of light itself can be done in constant time. The worst case running time is therefore

$$O(P \cdot N^2). \quad (3.3.1)$$

While the algorithm is quadratic in the number of patches, it can still be run in real time when using a clever implementation of the radiosity algorithm[11]. The addition of signals to simulate specular reflections introduces a few new variables to the algorithm:

- BF , the branching factor. The maximum amount of outgoing signals an incoming signal is allowed to be split up in.
- H , the amount of hops. The maximum amount of patches a signal is allowed to bounce on before it returns to the radar.
- $minP$, the minimum amount of light a signal has to carry. All signals below this threshold are immediately discarded.

These variables are all limiting factors on the amount of signals allowed at a time. Because incoming signals break up into multiple outgoing signals the number of signals tends to grow very fast with every pass of the radiosity algorithm. Assuming that every patch can send light to at least BF other patches the amount of signals after P passes will be

$$BF^P. \quad (3.3.2)$$

Similar to the real world, each patch sends its light to every other patch within its field of view, which in the worst case is N when every patch can see every other patch. For a patch to be able to send to every other patch, the signal would need to be split into N outgoing signals. For this $BF = N$ needs to hold. In that case the worst case running time is degraded to

$$O(N^P) \quad (3.3.3)$$

As N is large even for simple geometries, due to splitting triangles into smaller patches, and P has to be 2 or larger for the radiosity effect to work at all, one quickly comes to a running time that is unreasonably high for realtime calculation to still be possible.

To alleviate this a few mitigating factors have been incorporated into the Radar Signal Simulation algorithm. For example signals that fall below $minP$ are discarded from the simulation on the premise that the effect they have on the final radar image is negligible. Another is that signals can only bounce against H patches before returning back to the antenna. This prevents situations where a signal can become stuck bouncing back and forth between two patches without adding anything to the final image. A final mitigating factor is that signals that have travelled too far and would fall off the radar screen if they would ever arrive back at the antenna are removed from the simulation. While all these optimizations help speed up the algorithm, they do not prevent that it runs exponentially in the number of passes.

In chapter B.2 it was discussed that the radiosity algorithm is view-independent and that it only needs to be run once in order to view the three dimensional scene from every possible angle with the correct lighting. The running time would not matter if it was possible to precompute the lighting information once and use it for all subsequent executions of the algorithm. Unfortunately one can not use this property advantageously. As the antenna moves to a different position with the ship, it is not only the *view* that is moving but also the *light source*. One can not reuse results of the algorithm where the light source was at a different position. On top of that the results of the radiosity algorithm can also not be reused if the environment changes. In the real world there are plenty of moving objects in the environment, most notably other ships. The high running time of the Radar Signal Simulation algorithm prevents it from being used as a real radar simulator in practice.

3.3.3 Radar Image Simulation Based on 3d Scene Database in Marine Simulator

A paper by Hongxiang, Kelun and Yicheng [13] try to solve the same problem with the '3d scene database' method. Like Radar Signal Simulation it takes as input the full 3d geometry of the environment and outputs a round radar screen (see figure 3.3.6).

The main drawback with this method is that it does not take any specular reflection effects into account. Only the direct reflections are considered in this paper. Reflection area, reflection angle, distance and material properties are simulated correctly. As with Radar Signal Simulation the algorithm uses line-triangle intersection to find all triangles that are in line of sight of the radar. The paper proposes an optimization to the line-triangle intersection algorithm that works in their specific case allowing them to simulate an environment of 50.000 triangles in under three seconds. The algorithm does not support echo scaling, side lobes effect, second trace effect, gain control, rain control, pulse length effect or specularity.

3.3.4 Transas

The Transas Radar Simulator (figure 3.3.7) is an example of a commercial radar simulator. It is part of the Navi-Trainer Professional 5000 Navigational Simulator [19]. As it is a closed source commercial application. Little is known about what radar effects it supports. Only the specularity effect is given as a bullet point feature.

From the publicly disclosed screenshots it can be inferred that the Transas Radar Simulator supports gain, rain and sea controls. As well as realistic sea clutter and the occlusion effect.

3.3.5 In2Sim

The In2Sim Radar/ARPA trainer (figure 3.3.8) is another commercial radar simulator. As with the Transas Radar Simulator little is known about what radar effects it supports as most public

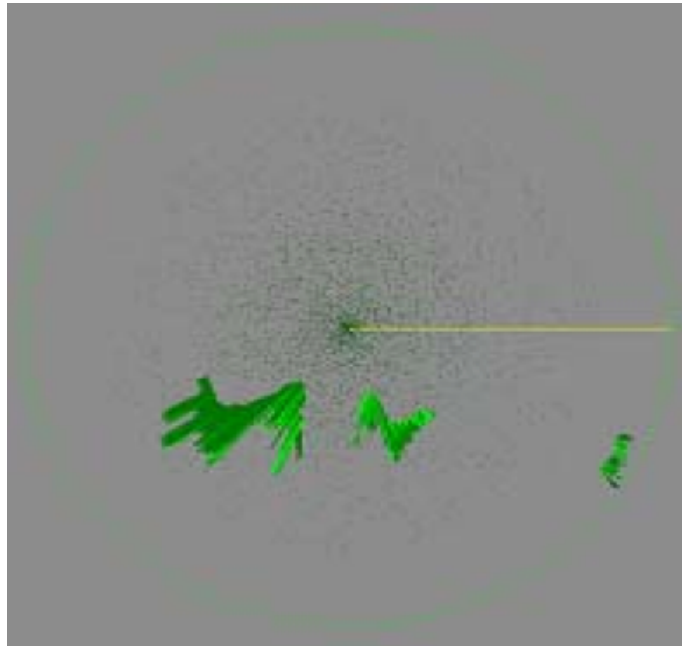


Figure 3.3.6: Screenshot showing simulated radar screen with the 3d scene database method.[13]

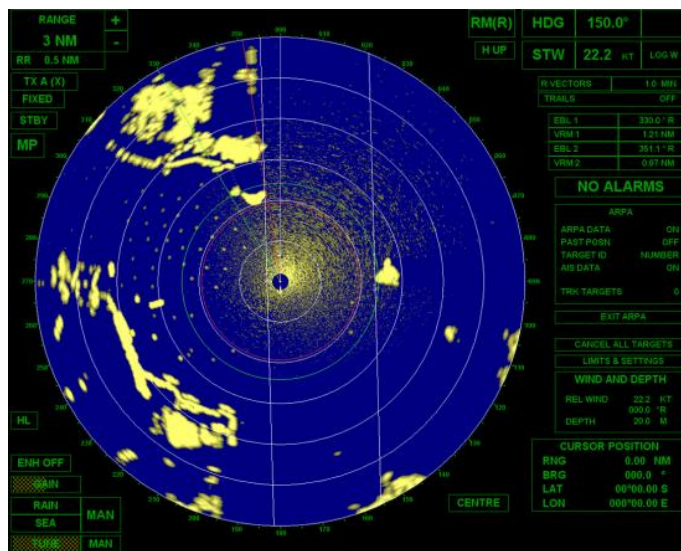


Figure 3.3.7: Screenshot of the Transas Radar Simulator [19]

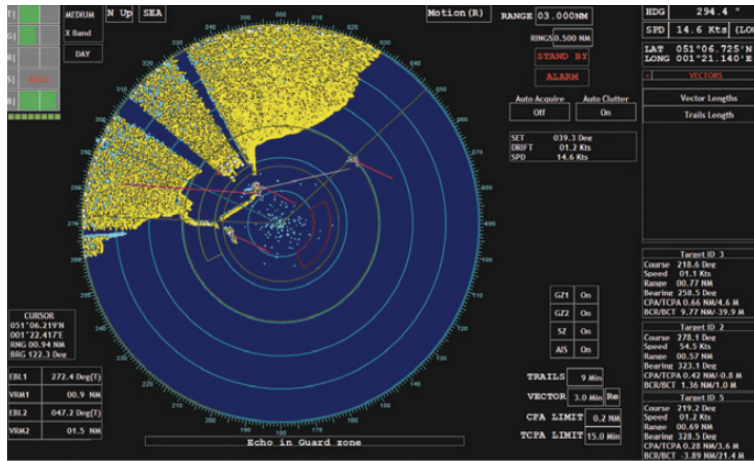


Figure 3.3.8: Screenshot of the In2Sim Radar Simulator [20]

information focuses on the information around the radar image and the vector overlays instead of the radar image itself[20].

The only feature that can be inferred from the screenshots is that it supports gain, rain and sea controls.

3.3.6 Related simulations

Another radar simulator paper called 'Bistatic Radar Imaging of the Marine Environment Part II: Simulation and Results Analysis' [14] discusses in detail how to correctly simulate a radar in a marine environment. Unfortunately the paper does not refer to the marine radars that this thesis is about. Instead they simulate *bistatic radars*. These are radar systems that have their transmitter and receiver located at different locations. A marine radar, in contrast, is a *monostatic radar*. Bistatic radars are typically used across large distances for remote sensing applications. Most methods described in this paper unfortunately hinge on the fact that the scale is larger and that the radar is bistatic. Other related research includes papers that model and simulate radar signals at the wavelength levels. These focus on correctly simulating correct noise, doppler effect, phase

Chapter 4

Rendering-based simulation algorithm

While the Radar Signal Simulation produces an accurately-simulated radar image its main weakness is that the running time is large even for simple geometries. A single radar image can take several seconds to a few minutes to complete. This is unacceptable in a ship simulator where the crew must have an always up-to-date image on the radar screen. What is needed is an algorithm that can simulate all the radar effects (see section 2.3) in real time so that it can be used in a real ship simulator environment. To do this, a few corners have to be cut on some of the radar effects.

In this thesis we will introduce what we call the "Real-time Rendering-based Realistic Radar Simulator". Our simulator can display a radar screen that is both real-time and realistic. Our rendering method works by rendering the environment around the radar antenna. This can be done fast on most GPUs. After rendering we read back the depth buffer, which contains most information needed to create a realistic radar screen.

Our rendering method simulates a real antenna by first rendering the scene from the viewpoint of the radar antenna (figure 4.0.1). That is, it places a virtual camera at the exact spot where the radar antenna is located, which points in the same direction as the radar antenna. From this viewpoint the environment is rendered to the frame buffer. After rendering is complete, the color buffer is completely ignored. Every pixel in the depth buffer now contains the distance to the camera (color buffer and depth buffer are explained in Appendix A). Using the depth buffer, we can count for every distance how many pixels in the buffer are at that distance. This gives us a histogram that contains by how many pixels each distance is represented in the image. This will be called the *signal buffer*. This signal buffer can then be interpreted as an antenna feed like in figure 2.2.3.

Our algorithm takes the following data as input:

- gain
- rain
- sea
- totalWatt (total amount of energy radar can send per pulse)
- startingThreshold (the value used to determine whether target is visible, usually dependent on totalWatt)

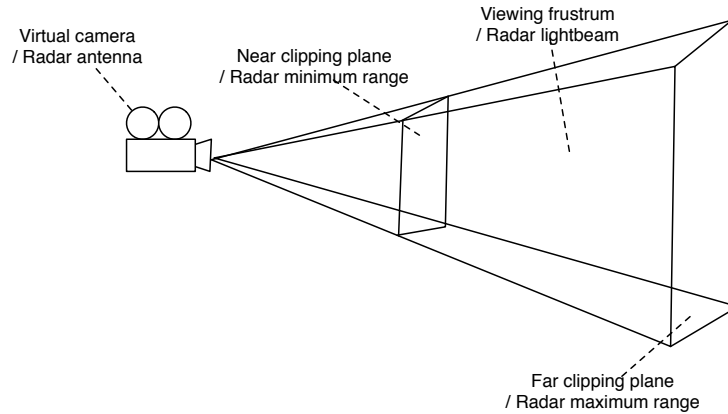


Figure 4.0.1: Analogy of a virtual camera with a radar antenna.

- range
- horizontal beamwidth
- vertical beamwidth
- pulse length
- pulsesPerRevolution.
- shipPosition

We will assume that these variables are globally present and remain constant. Except for gain, rain and sea which the user should be able to control. The output of the algorithm is a continuously-updated radar image.

The main simulation loop is as follows:

Algorithm *simulation()*

Input: Nothing

Output: Continuously-updated radar image

1. rotation $\leftarrow 0$
2. **while true**
3. **do** rotation $\leftarrow (\text{rotation} + 1) \bmod \text{pulsesPerRevolution}$
4. signalBuffer $\leftarrow \text{SENDSIGNAL}(\text{ROTATION})$
5. T $\leftarrow \text{startingThreshold}$
6. T, signalBuffer $\leftarrow \text{APPLYEFFECTS}(T, \text{signalBuffer})$
7. DRAW SIGNAL BUFFER(T, signalBuffer, rotation)

The signalBuffer is a one-dimensional array from 0 to n that is analogous to the histogram that a radar keeps track of, introduced in section 2.4. signalBuffer[1] represents the energy returned from the closest point the antenna can receive energy. The value from this index will be displayed in the center of the radar screen. signalBuffer[n] represents the very edge of the screen at the maximum range of the radar. Each value represents the total amount of energy that returned from that range. The amount of indices should be *range/pulse length* at a minimum but can be chosen greater.

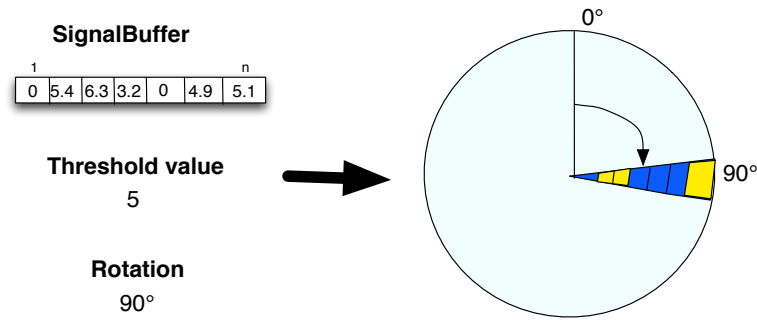


Figure 4.0.2: Simple example of drawing the signal buffer as a slice on the radar screen

Drawing the signalBuffer as a slice on the screen is done by means of a threshold value T . Every value above the threshold value is colored yellow while every value below it remains blue, see figure 4.0.2 and algorithm DRAW SIGNAL BUFFER.

Algorithm *DrawSignalBuffer*(T , *signalBuffer*, *rotation*)

Input: Threshold T , the signalBuffer and the rotation

Output: signalBuffer drawn on screen as slice.

1. slice \leftarrow texture of width length(signalBuffer), and height 1
2. **for** $i \leftarrow 1$ **to** length(signalBuffer)
3. **do if** signalBuffer[i] $\geq T$
4. **then** slice[i][0] \leftarrow yellow
5. **else** slice[i][0] \leftarrow blue
6. start \leftarrow rotation / pulsesPerRevolution $\cdot 2\pi$
7. end \leftarrow (rotation + 1) / pulsesPerRevolution $\cdot 2\pi$
8. render triangle with slice texture and coordinates (0, 0), (sin(start), cos(start)), (sin(end), cos(end)).

The gain, rain and sea filters are applied after the signalBuffer has been computed, but before it is drawn on screen (algorithm APPLYEFFECTS). The gain, rain and sea effects are applied sequentially and influence both the threshold and the signalBuffer. These are elaborated on in section 4.6.1, 4.6.2 and 4.6.3.

Algorithm *applyEffects*(T , *signalBuffer*)

Input: Threshold T and the signalBuffer

Output: Threshold T and signalBuffer both updated for gain, rain and sea effects

1. $T \leftarrow$ APPLYGAINEFFECT(T)
2. $T \leftarrow$ APPLYSEAEFFECT(T)
3. signalBuffer \leftarrow APPLYRAINEFFECT(signalBuffer)
4. **return** T , signalBuffer

4.1 Sending the signal

The SENDSIGNAL function is responsible for filling the signal buffer. It does this task by rendering the scene to a frame buffer and then converting that frame buffer to a signal buffer.

Algorithm *sendSignal*(rotation)

Input: Rotation

Output: signalBuffer is filled with result from single simulated pulse

1. Clear depthbuffer and signalBuffer
2. Move camera to *shipPosition*
3. Rotate camera to *rotation*
4. RENDERSCENE()
5. ADDDEPTHBUFFERTOSIGNALBUFFER()

The camera position and orientation before rendering should be exactly the orientation of the radar antenna. After the rendering the addDepthbufferToSignalbuffer fills the signal buffer.

Algorithm *addDepthbufferToSignalBuffer()*

Input: Access to the frame buffer

Output: Values from depth buffer are added to the signalBuffer

1. depthpixels \leftarrow READDEPTHBUFFER()
2. **for** $P \leftarrow$ depthpixels
3. **do** $D \leftarrow$ UNDODEPTHSCALING(P_{depth})
4. $D \leftarrow$ APPLYPULSELENGTHEFFECT(D) //section 4.3
5. $E \leftarrow$ totalWatt / framebufferWidth / framebufferHeight
6. $E \leftarrow E / \frac{1}{D^2}$
7. $E \leftarrow$ APPLYRADIATIONPATTERNEFFECT(E, P) //section 4.5
8. $E \leftarrow$ APPLYSPECULARITYEFFECT(P, E, Camera) //section 4.4
9. signalBuffer[D] \leftarrow signalBuffer[D] + E

The algorithm iterates over all pixels in the frame buffer, we assume that we can for each pixel P read out its corresponding value in the depth buffer P_{depth} . We calculate for each pixel a distance D in meters and the returned energy E in watt. In real life the signal follows the inverse fourth-power law (section 2.2.3). $\frac{1}{D^2}$ of the total energy reaches the target and from that only $\frac{1}{D^2}$ reaches back to the antenna. A real radar compensates for this by amplifying the incoming signal by a D^4 term.

In a perspective camera projection objects get smaller the further they are. Like a radar this follows the inverse square law. But unlike a radar we do not have to worry about 'energy' being lost from the object back to the camera. Thus in our algorithm we only have to compensate for the inverse square law (line 6).

Precision in the depth buffer are not linearly distributed. Precision close to the near clipping plane is many orders of magnitude greater than close to the far clipping plane. To convert this logarithmic value to a linear one in world space meters, we can apply the following algorithm: [12]

Algorithm *undoDepthScaling(D)*

Input: Value from the depth buffer D

Output: Value D scaled linearly between near and far clipping plane

1. $z \leftarrow 2 \cdot D - 1$
2. **return** $\frac{-2 \cdot far \cdot near}{z \cdot (far - near)} - far + near$

where D is the logarithmic distance obtained from the depth buffer and *far* and *near* are the distances of the clipping planes. The energy E is derived from totalWatt (a property of the radar that determines how much energy it sends out in a single pulse) divided by the width and height of the frame buffer. This gives the energy value per pixel. For each pixel we add energy E at index D in the signalBuffer.

4.2 Render settings

The horizontal beam of a radar is typically very narrow. A standard radar can have a horizontal beamwidth of 3° while the vertical beamwidth is 25° . This should be reflected in the field of view settings of the camera. We have chosen the horizontal field of view to run from the null gap left from the left side lobe to the null gap right from the right side lobe. With this setting we can fully emulate the side lobe effect (section 4.5). The vertical field of view is chosen to be the null-to-null vertical beamwidth.

The dimensions of the frame buffer is not as important as the field of view but should not be chosen too small. We found that a vertical size of 1024 pixels is sufficient for most situations. Care should be taken with determining the near and far clipping plane of the camera. The far clipping plane can be set equal to the maximum range of the radar. The distance of the near clipping plane has a strong influence on the precision of the depth buffer due to its logarithmic scaling. In our implementation we have set the near clipping plane to be 1/10th that of the far clipping plane.

4.3 Pulse length

The pulse length defines the duration of a signal sent by the antenna. The larger the pulse length, the longer the signal is and the larger the echo on the screen becomes. If for example the pulse length is 0.03μ seconds (10 meter) one expects that the echo on the screen is 10 meters long. To simulate this we add a random distance to each pixel equal to at most the pulse length:

Algorithm *applyPulseLengthEffect(D)*

Input: Distance D

Output: D randomized within pulse-length area.

1. **return** $D + \text{RANDOM}(0, \text{pulse-length})$

This randomizes the depth of each pixel within the pulse length. The effect of this is that each echo will appear to be exactly the size of the pulse length, which is the effect that we are after.

4.4 Specularity

Up to this point we have considered each surface to be diffuse. Because we have the special situation that the light source and the camera are the same vector, diffuse objects all reflect the same amount of energy per pixel (given equal distances). While many objects behave in a diffuse fashion, some objects behave in a specular fashion. The amount of light a specular surface returns depends on the position of the light, the position of the camera, the specularity of the surface and the orientation of the surface. To calculate the specularity we will use the Phong Illumination Model[18], i.e.,

$$E' = K_d \cdot E \cdot (N \cdot L) + K_s \cdot E \cdot (-C \cdot \text{reflect}(L, N))^s \quad (4.4.1)$$

Where E' is the reflected energy, E is the initial energy, K_d is the diffuse reflection factor, K_s is the specular reflection factor. L is the normalized light vector, C is the normalized camera vector, N is the normalized surface normal and s is the shininess parameter. All vectors are illustrated in figure 4.4.1.

The reflection function takes a vector L and a surface normal N and returns the vector that is vector L reflected off of a surface with normal N . It can be implemented with the following algorithm:

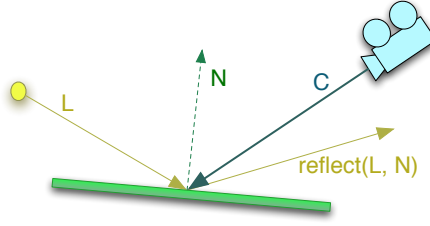


Figure 4.4.1: Relevant vectors for calculating the returned amount of light for a specular object.

Algorithm REFLECT(L , N)

1. **return** $-2N(L \cdot N) + L$

The *shininess* in the Phong Illumination Model determines how spread out the reflected light is. As the shininess increases, more light will be sent in the specular direction, and less light is sent in all other directions. There is no limit to how high the shininess variable can get. As it approaches infinity almost all light will be sent in the specular direction. The diffuse and specular reflection factors K_d and K_s determine how specular or diffuse the material is. Because we have no emissive or ambient reflection factor in our model we give each surface only a specular reflection factor and calculate the diffuse reflection factor with equation 4.4.2.

$$K_d = 1 - K_s \quad (4.4.2)$$

In the radar simulator the vector that indicates the antenna direction must be used for both the light vector L and the camera vector C . The radar antenna is both a transmitter and a receiver and so acts as both the camera and the light source. The surface normal, the specularity and the shininess of a surface can be retrieved from the method described in section 4.8.2.

Algorithm *applySpecularityEffect*(E , P , C)

Input: Energy E , pixel P , Camera vector C

Output: Energy E adjusted for specularity of surface

1. $T \leftarrow P_{triangle}$
2. **return** $E \cdot (1 - T_{specularity}) + T_{specularity} \cdot E \cdot (-C_{vector} \cdot \text{REFLECT}(C_{vector}, T_{normal}))^s$

4.5 Sidelobes

In section 2.2.4, we mentioned that the strength of the echo depends on the incoming direction relative to the direction in which the antenna is pointing. When we calculate the energy returned from a pixel in the depth buffer, we need to take into account the location of this pixel in the image. A pixel that is in the exact center of the image should have the highest energy value as it is the exact point the radar antenna is pointing to. Pixels further from the center should have less energy.

To exactly simulate the radiation pattern, we can use the radiation pattern formula[17] to determine the amount of energy returned from a specific direction. In the vertical direction the strength S drops off slow first and fast as it approaches null gap. This can be modeled as equation 4.5.1.

$$S = |\cos(P_y)| \quad (4.5.1)$$

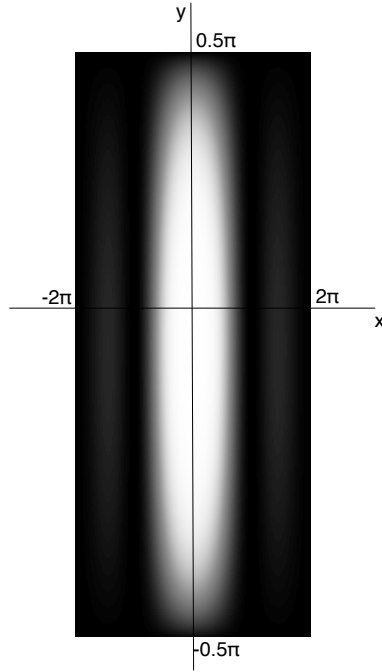


Figure 4.5.1: Overlay on rendered image to simulate radiation pattern sensitivity. (Sidelobes weakly visible to the left and right of the center main lobe)

where P_y is assumed to be normalized between -0.5π and 0.5π . In the horizontal direction the strength can be modeled with equation 4.5.2.

$$S = \left(\frac{\sin(P_x)}{P_x} \right)^2 = \text{sinc}^2(P_x) \quad (4.5.2)$$

where we assume that P_x has been normalized between -2π and 2π . This exactly captures the main lobe and the two side lobes. We could capture more lobes but as lobes become increasingly weaker the minimal accuracy gain is not worth the cost. We combine the vertical strength and horizontal strength by multiplying them as shown in equation 4.5.3

$$S = \left(\frac{\sin(P_x)}{P_x} \right)^2 |\cos(P_y)| \quad (4.5.3)$$

The formula is undefined for $P_x = 0$ due to the division by P_x . However the limit for P_x as it approaches 0 is 1 (with $P_y = 0$). In our implementation we make an exception for $P_x = 0$ and return the result from equation 4.5.1 instead. The energy value is multiplied by the strength to get the energy that is corrected for the radiation pattern. Equation 4.5.3 is plotted in figure 4.5.1, with [0-1] mapped to [black-white]. In `APPLYRADIATIONPATTERN EFFECT` the equation is worked out as the implemented algorithm.

Algorithm *applyRadiationPatternEffect*(E, P)

Input: Energy E , pixel p

Output: Energy E adjusted for radiation pattern effect

1. $x \leftarrow 4\pi P_x - 2\pi$

2. $y \leftarrow \pi P_y - 0.5\pi$
3. **if** $|P_x| \leq \epsilon$
4. **then return** $E|\cos(y)|$
5. **else return** $E|\cos(y)| \left(\frac{\sin(x)}{x}\right)^2$

An unimplemented optimization would be to precompute all possible values in a texture or look up table. The radiation pattern energy multiplier can then be looked up in a single operation instead of recomputing the same multiplier every time. The radiation pattern model that we described can easily be interchanged by other models by modifying the APPLYRADIATION-PATTERN-EFFECT method.

4.6 Screen

After all effects have been applied we have a signal buffer that we have to convert to a blue / yellow image. On a radar system this is done by means of a threshold, which is described in section 2.4. All signal values above the threshold are colored yellow, while all values below it are blue. This threshold can be configured by the gain, rain and sea control.

4.6.1 Gain control

The gain is a dimensionless variable that can be configured from 0 to 1. Where at 0 only the strongest signals remain visible while at 1 almost all signals are visible on the screen. There is no literature available that gives an accurate description of what the minimum or maximum values mean or how the gain exactly influences the threshold. All that is known is that a higher gain lowers the threshold and a lower gain increases the threshold. We have implemented a formula that we believe is plausible for a realistic radar system.

Algorithm *applyGainEffect*(T)

1. **return** T / G

Where T is the threshold value and G is the gain from 0 (exclusive) to 1. With this formula the threshold remains the same if the gain is 1 and becomes arbitrarily high as the gain drops to 0. This reflects the real life radar system where the gain is normally configured around 0.5. While it has a large influence between 0 and 0.1 when almost no targets remain on the screen. Note that with this setup we can never allow the gain to become exactly zero, due to division-by-zero.

4.6.2 Rain control

As described in section 2.4.3, the rain control is a special filter in that it adjusts the energy in the signal buffer instead of the threshold value. As with the gain, the rain control R is a dimensionless control that can be configured from 0 to 1. At 1 it applies a discrete time derivative to the energy value E . Instead of using the energy value E itself it uses the absolute difference of the current energy and the previous energy value E_{prev} to compare to the threshold.

But the rain control is not a simple on/off switch; it is a range from fully off to fully on. All the settings in between are interpolated between off and on. Thus we can calculate the final energy value to be $(1 - R)$ times the original energy and R times the derivative value as seen in line 3.

Algorithm *applyRainEffect*(signalBuffer)

Input: signalBuffer

Output: signalBuffer adjusted for rain effect

1. **for** $i \leftarrow -1$ **to** $n - 1$
2. **do** $\text{diff} = \text{abs}(\text{signalBuffer}[i+1] - \text{signalBuffer}[i-1])$
3. $\text{signalBuffer}[i] = \text{signalBuffer}[i] * (1 - R) + \text{diff} \cdot R$
4. **return** signalBuffer

4.6.3 Sea control

The sea control S controls how visible nearby targets are. As the sea control increases to 1, close targets disappear, while far away targets remain visible. To achieve this, the sea control adjusts the threshold value per range. With the sea control at 0, the threshold will be the same at every range. As the sea control increases to 1, the threshold value for close ranges increases dramatically. To achieve this we use the following equation,

$$T = \frac{T}{R^S} \quad (4.6.1)$$

where T is the threshold value in watts, R is the range and S is the sea control setting from 0 to 1. This leaves the dimension of the range unspecified. If we interpret the range to be in meters (as is the case in our implementation), then everything further than 1 meter will have a lower threshold and everything closer will have a higher threshold. Sea clutter can however appear up to hundreds of meters away. Only being able to increase the threshold within a range of 1 meter makes no practical sense.

To get a more practical curve, we introduce a baseline range b around which the curve pivots. Every range further than b has a lowered threshold while every range closer has an increased threshold. At b the threshold remains the same. Dividing range R by b has the desired effect on the curve:

Algorithm *applySeaEffect*(T, R)

Input: Threshold T , range R , sea control S

Output: Threshold T adjusted for sea effect.

1. **return** $(T/\frac{R}{b})^S$

There have been no references in the literature as to what the baseline value is. Presumably the value differs from radar system to radar system. In our implementation we have used a value of 2500 meters, as it seems to be comparable to a real radar system.

4.7 Specular Reflections

In section 2.3.7 it was explained that a large surface can act like a mirror for the radar signal and show ghost targets on the screen (figure 2.3.14). Implementing this effect with the rendering method does not come as natural as with the Radar Signal Simulation algorithm that works with rays instead of pixels. In our implementation we solved the problem by finding the largest flat surface S in the frame buffer and use only that surface to reflect light on. We render the scene a second time, but with the camera mirrored over the largest surface. The result from this rendering pass is then added to the signal buffer, resulting in both the surface itself as well as the targets it should reflect appearing as targets in the signal buffer.

The following algorithm replaces `sendSignal()` introduced in section 4.1 and has support for the specular reflections effect:

Algorithm SENDSIGNALSPECULAR()

1. Clear color, depth, stencil and signal buffer
2. RENDERSCENE()
3. ADDDEPTHBUFFERToSIGNALBUFFER()
4. $S \leftarrow$ largest rendered surface
5. Clear color & depth buffer
6. render S in stencil buffer
7. Mirror camera in S and set clipping plane to S
8. RENDERSCENE() only in stencil buffer area
9. ADDDEPTHBUFFERToSIGNALBUFFER()
10. **return** signalBuffer

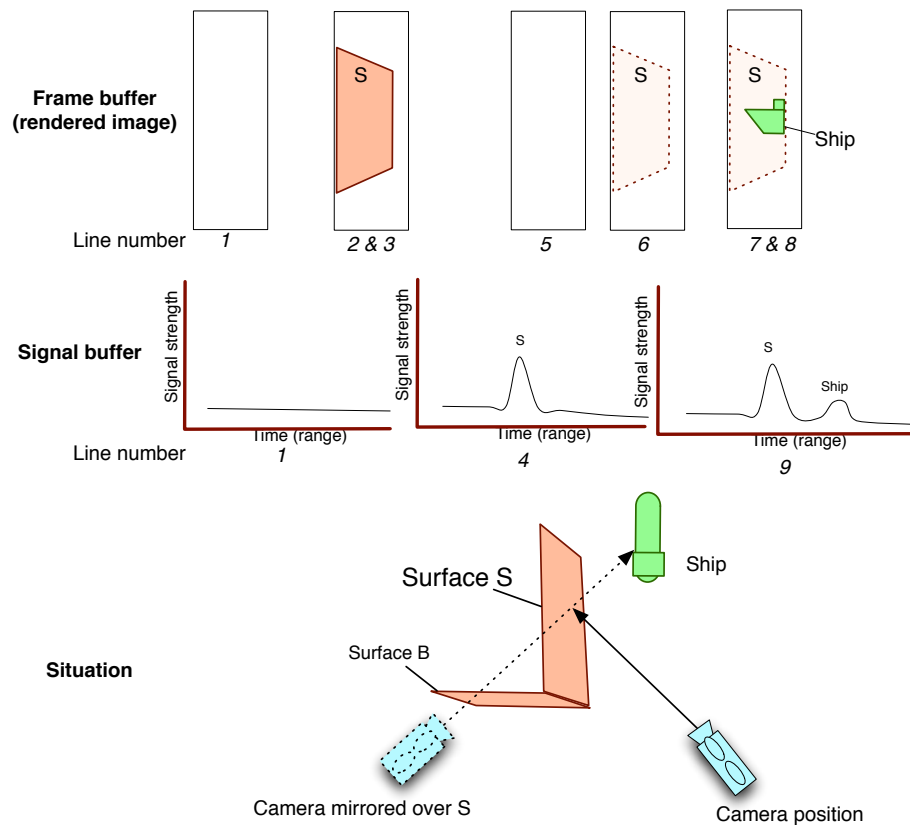


Figure 4.7.1: All the steps of algorithm sendSignalSpecular() applied to a specific situation where the surface S acts as a specular surface and the signal can bounce to a nearby target

The primary effect of specular reflections is that extra targets appear on the screen that do not exist in reality. If the antenna is close enough to a large flat surface that surface can bounce the signal in another direction. This can return signals from a target that is not in the direction the antenna is pointing to, which show up as false targets on a radar system.

To achieve this effect we render the scene twice. First we clear the frame buffer and signal buffer (line 1), render the scene (line 2) and then add the frame buffer to the signal buffer (line 3). This part is identical to the sendSignal() algorithm in section 4.1.

Instead of ending the algorithm here, we find the largest rendered surface S in the frame buffer and use that as the one surface that reflects light one pass further. Note that in real life every surface would reflect light, not just the largest. The Radar Signal Simulation algorithm simulates this accurately, while our algorithm has to take a shortcut here to maintain real time computation. The largest surface is determined by the amount of pixels it takes up in the frame buffer.

To make sure only the pixels of surface S can reflect light, we employ the use of the stencil buffer. After clearing all the buffers once more (line 5) we draw surface S only into the stencil buffer. In this way, we can force the second rendering pass to only render in the pixels occupied by surface S . This prevents the rendering of objects that could never be detected through surface S .

After this, move the camera to perspective mirrored over S (figure 4.7.1). Before rendering, a clipping plane has to be enabled that prevents any objects from rendering before surface S . For example in figure 4.7.1 surface B blocks the view from the mirrored camera position to the target. Obviously this surface should not interfere with a signal in real life that goes from the target to surface S and to the antenna. To prevent surface B from rendering, we set up a clipping plane that goes exactly through surface S and only allows objects beyond that surface to be rendered. Because of back face culling we do not have to worry that surface S itself blocks the view. After rendering the second time we add the result of this frame buffer also to the signal buffer (line 9). This results in a signal buffer containing both surface S itself and the targets it reflects.

The major drawback of this extension of the rendering method is that it doubles the running time. For every antenna rotation we have to perform two rendering steps and two frame buffer read operations, which are two of the most expensive operations in the algorithm. Another drawback of this method compared to the result of the Radar Signal Simulation algorithm, is that it can only handle one reflection pass. In real life, if two surfaces are sufficiently parallel and close to each other, a signal can bounce between them multiple times resulting in multiple ghost targets. This situation cannot be realistically simulated in real time using the rendering method.

An unimplemented extension of this algorithm is to repeat it also for the second largest triangle, the third largest triangle and so on. While this comes at a severe running time penalty, it allows for all specular reflection effects to be simulated accurately. While testing we have found that one reflection pass is sufficient for almost all prominent ghost targets.

4.8 Finding surface normals

For all flat surfaces it is possible to draw a vector that points exactly away from the surface, perpendicular to the plane of the surface. Such a vector is called a *normal*, and it is very important if we want to know at what angle a surface is being viewed. For some solutions for specularity and specular reflections, the algorithm needs the normal of a surface. Getting this information requires a tricky workaround.

The problem comes from the fact that we cannot tell from the depth map at what angle each pixel is (it only gives information about the depth of each pixel). However, there are several ways around this problem.

4.8.1 Solution 1: From depth map

For each individual pixel, in isolation, we cannot tell the normal of its surface. However, the surrounding pixels can give a hint.

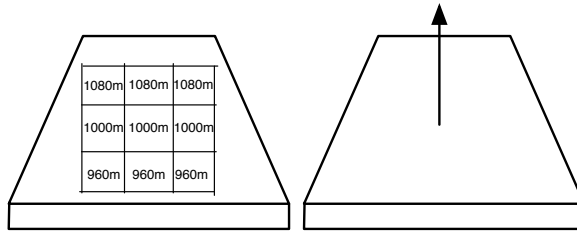


Figure 4.8.1: The normal for the middle pixel can be inferred from the distances of the surrounding pixels

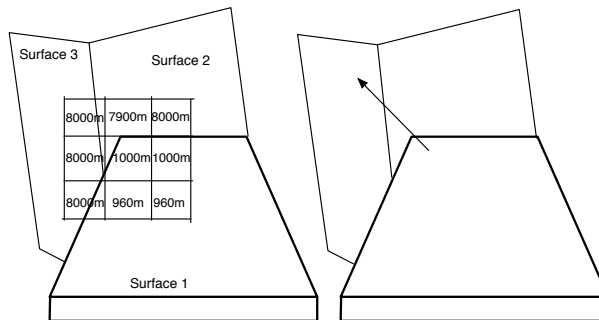


Figure 4.8.2: Inference uses values not belonging to the surface whose normal we are trying to determine. This computes an incorrect normal vector.

If we assume all surrounding pixels belong to the same surface, we can infer which way the normal is pointing. If for example all surrounding pixels have the same distance value, we know that the normal is pointing straight towards the camera. If the pixels above are further away than the pixels below we know that the normal must be pointing upward. And if the pixels to the left are further away than the pixels on the right we know that the vector must be pointing to the left (from the viewpoint of the camera). Using the depth values of the surrounding pixels we can precisely calculate the orientation of the normal (figure 4.8.1).

As stated above, this method only works if all surrounding pixels render the same surface. If a pixel is on the border of a surface it will use values from pixels that are from surfaces in the back or in front and it will compute an incorrect normal (figure 4.8.2).

While this problem is unavoidable, it can be mitigated by choosing a high enough resolution for the rendered image. When the resolution increases the number of pixels that are near a surface border will be relatively lower when compared to the number of pixels that are not near a border. This will decrease the impact of faulty surface normals on the radar screen. The resolution can only be increased so much before performance is going to be a problem. Fortunately, there exists a better solution.

4.8.2 Solution 2: By color coding

So far we have only used the depth buffer for calculations, ignoring the color buffer entirely. This is a waste because for every pixel we can store 32 bit of extra information about it in the color buffer (normally, the color buffer is shown to the user and is supposed to be a pretty picture).



Figure 4.8.3: $(0.707, 0.707, 0)$ normal encoded as a color

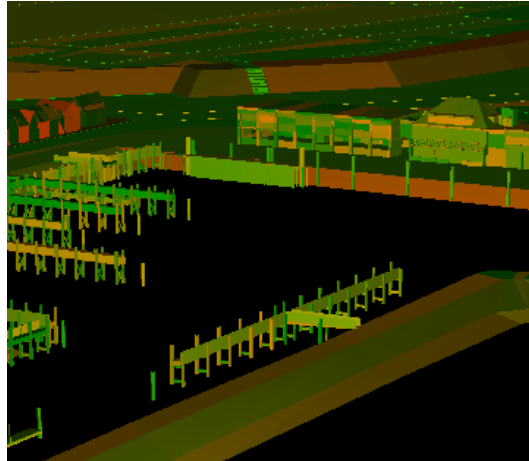


Figure 4.8.4: Color buffer with every surface color-coded with a unique color.

But in our case it is never shown to the user).

A pixel in the color buffer is made up of four components: red, green, blue and alpha. Each of these components have 8-bit precision, meaning that we can have 255 different red values, 255 different green values and so on.

Let us consider for example a surface that has a normal vector of $(0.707, 0.707, 0)$ and a pixel that is occupied by this surface. Instead of giving the pixel any undefined color we encode the information of the x, y and z components of the normal vector into the red, green and blue color channels of the pixel. This creates a color that is 70.7% red and green and contains no blue. In the color buffer the surface will appear to have a brownish color (figure 4.8.3)

The normal can be reconstructed back from the color by using the reverse process. Namely by interpreting the red, green and blue as x, y and z coordinates. This solution was used in our implementation, until we needed to encode other information such as material specularity and shininess. 32 bits of color information was too little for us to encode all the surface properties we needed to know about each pixel. The solution to this was to store each surface in an array in main memory and encode the *index* of the surface into the color.

Each pixel in the color buffer has 32 bits of information. 8 bits for red, green, blue and alpha in that order. We want to encode the surface index into this pixel, but chances are high that the index is higher than 255 meaning that we cannot use just the 8 red bits for this. To get around this we stop viewing the color as 4 separate values of 8 bits and start viewing it as one large 32 bit number. This gives us a total of $2^{32} = 4.294.967.296$ different indices we can store per pixel. While this does introduce an upper bound on the number of indices one can store, and thus on the number of surfaces one can identify by color. It is however safe to assume that most environments can be defined using less than four billion surfaces.

Getting the normal from the color is done using the reverse process. First we interpret the color as a 32 bit number. Then we use this number as an index to look up the surface in an array

containing the properties of all surfaces and find the normal information at the correct index. The number 0 (black) is reserved for pixels that are not part of any surface (for example the sky). This is also the method that we used in the final implementation of our radar simulation. Figure 4.8.4 shows an example of what the color buffer looks like with this process.

4.9 Complexity

The complexity of our algorithm is dependent on three variables: the number of pixels in the frame buffer p , the number of triangles t and the size of the signalBuffer n . We will define the complexity in terms of how long it takes for the radar to complete a single revolution.

To complete a revolution the algorithm has to draw an amount of slices equal to $pulsesPerRevolution$. Each slice is simulated by a call to SENDSIGNAL, a call to APPLYEFFECTS and a call to DRAWSIGNALBUFFER. Thus the complexity of the algorithm is,

$$O(pulsesPerRevolution \cdot (SENDSIGNAL + APPLYEFFECTS + DRAWSIGNALBUFFER))$$

In SENDSIGNAL the depth buffer is cleared, which is the size of the number of pixels in the frame buffer p . And the signalBuffer is cleared, whose size n depends on the resolution of the radar. Both actions take $O(p)$ and $O(n)$ respectively. After clearing the camera is moved and rotated which is a constant operation. After that the scene is rendered, which is linearly dependent on the number of triangles t , and a call to ADDDEPTHBUFFERTO SIGNALBUFFER is made. The complexity of SENDSIGNAL is,

$$O(p + n + t + ADDDEPTHBUFFERTO SIGNALBUFFER)$$

In ADDDEPTHBUFFERTO SIGNALBUFFER the value from the depth buffer is read back which is of size n . After that for each pixel the functions UNDODEPTHSCALING, APPLYPULSELENGTHEFFECT, APPLYRADIATIONPATTERNEFFECT and APPLYSPECULARITYEFFECT are called. These are all constant time functions. Which means the complexity here is just

$$O(p)$$

In the function APPLYEFFECTS three other effects are called. APPLYGAINEFFECT, APPLYSEAEFFECT and APPLYRAINEFFECT. The gain and sea function can run in constant time. APPLYRAINEFFECT iterates over the signalBuffer and runs in $O(n)$ time. Thus the complexity for the effects is

$$O(n)$$

DRAWSIGNALBUFFER iterates over all the values in the signalBuffer once. Filling and rendering the texture on screen can be done in constant time. Thus this function has a complexity of

$$O(n)$$

Combining all the functions we have a total complexity of

$$O(pulsesPerRevolution \cdot (p + n + t))$$

where p is the number of pixels in the frame buffer, n is the size of the signalBuffer and t is the amount of triangles to be rendered. We see that for a single pulse the algorithm is linear in both the number of pixels, the resolution of the display and the number of triangles. This would suggest that the number of triangles is the biggest bottleneck in our algorithm, but this is not true. The only time the triangles are iterated over is on the GPU when they are rendered. As this piece of hardware is optimized for rendering triangles the actual bottleneck is in the number of pixels of the frame buffer.

Chapter 5

Results

In this chapter we test our implementation of the Real-Time Realistic Radar Simulation algorithm. We will compare its results to the Radar Signal Simulation algorithm and see if it can simulate the same effects. All tests have a single contrived environment designed to highlight a single specific radar effect. The test is passed if our algorithm can simulate the effect both realistically and in real-time. We say it is realistic, if the effect can clearly be recognized in the image. And we say it is in real-time if a single rotation can be generated in under three seconds. The computer that we used to run these tests is a MacBook Air 2010 model which has the following specification:

Note that this system is fairly low end in terms of computation and graphics performance by today standards. Our radar simulation implementation is mostly bound by CPU performance, due to the high cost of processing each pixel on the CPU. There is a lot of performance to gain over the time measurements in our results by running the same tests on high-end hardware. Our implementation can be configured for a wide range of radar systems. The radar system that we simulate in our tests has the following specification and settings: (unless otherwise specified)

There will also be a discrepancy in timing information. Our tests run on a machine as described in table 5.1. But the same tests for the Radar Signal Simulation algorithm were performed on a quadcore Intel I5 3.33 Ghz CPU, which performs significantly better.

5.1 Occlusion

The occlusion effect is one of the most important effects, as it significantly impacts what is visible on the radar image and what is not. In this test we have an environment where a large vessel blocks the line of sight to a smaller vessel that is right behind it (figure 5.1.1).

The expected result is that the reflection of the large vessel shows up prominently on the

Table 5.1: Machine specification

CPU	Intel Core 2 Duo
Clockspeed	2.13 Ghz
RAM	4 GB
Graphics Card	NVIDIA GeForce 320M
VRAM	256 MB
OS	Mac OS X 10.7 Lion

Table 5.2: Radar specification

Horizontal Beamwidth	3°
Vertical Beamwidth	25°
Pulses Per Revolution (PPR)	360
Pulse length	0.03 μ s (10 meter)
Range	2778m (1.5 nm)
Gain control	50%
Rain control	0%
Sea control	0%



Antenna

Figure 5.1.1: Top view of the situation for the occlusion test. A small vessel hidden behind a large vessel from the point of view of the radar antenna.

radar screen while that of the small vessel remains hidden because it cannot be seen from the position of the antenna.

When we compare the result of our implementation with that of Radar Signal Simulation, we notice that the two are quite different. This difference can be explained in that the two algorithms try to simulate different types of radar signals. The Radar Signal Simulation simulates classic radar systems that have a low resolution and green-on-black displays. Our implementation tries to simulate the more modern systems as seen in figure 2.3.10 or figure 1.0.1. Comparing with the result of the Radar Signal Simulation, we see that both correctly hide the location of the small vessel and prominently show the location of the large vessel.

5.2 Side lobes

To test if side lobes are correctly simulated, we use an environment with a single ship (figure 5.2.1) and an enlarged horizontal beam width. If we would use the 3° horizontal beamwidth, the resulting side lobes and the gaps between the side lobes and the main lobe would be too small to see in the radar image. To test if they still correctly appear we increased the horizontal beam width to 15°. Note that such a setting is never used in real life due to excessive echo scaling.

Table 5.3: Result of occlusion test

Model	Triangles	CPU Time (our simulator)
TwoShips.rof	134	2.5 seconds

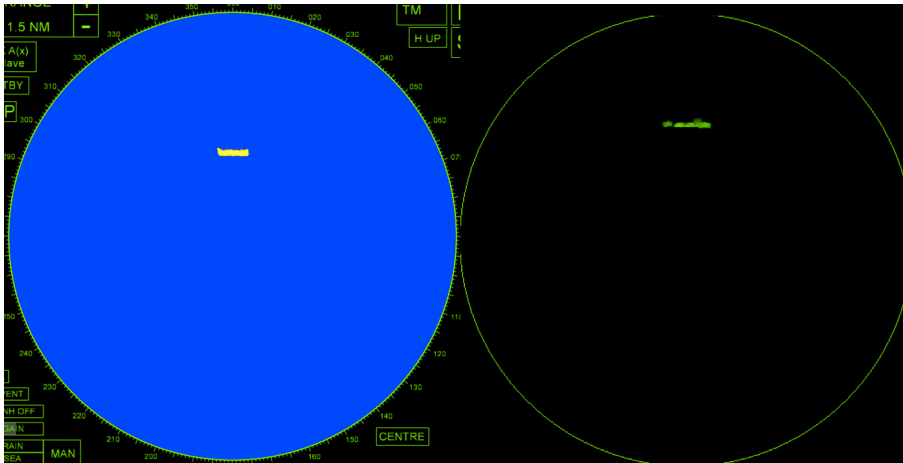


Figure 5.1.2: Result of the occlusion test. Both our simulator (left) and the Radar Signal Simulation (right) do not show the little vessel behind the visible large vessel.

—

•Antenna

Figure 5.2.1: Situation for the side lobe test: A single small ship ahead of the radar antenna

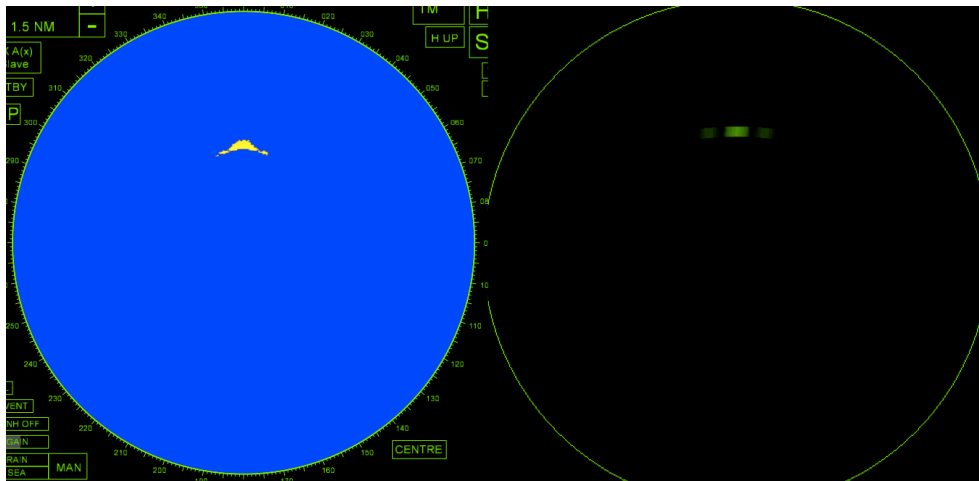


Figure 5.2.2: Result of the side lobe test. Left: Result of our implementation. Right: result of Radar Signal Simulation

Table 5.4: Result of side lobe test

Model	Triangles	Time
SimpleShip.rof	42	9.7 seconds



Figure 5.2.3: Less detail in side lobes due to reduced dimensions in depth buffer.

With the enlarged horizontal beam width, we see that the radar image shows a realistic depiction of the side lobe effect. Next to the main echo from the vessel, there are two small extra echoes that come from the side lobes.

Perceptually the two seem quite different. The result of Radar Signal Simulation shows a perfectly symmetric clean low resolution of the side lobes effect, while our result looks more jaggy and random. This is a result of the randomness in our algorithm. This was included because, similar to real life, no two sequential radar images are exactly alike.

Another thing we notice is a dramatic increase in the time it takes to generate this image compared to our last test. Even though the amount of triangles is three times smaller, the time it takes to generate the image is three times as long. The reason for this is that the increased horizontal beam width has increased the dimensions of the depth buffer. The larger dimensions increase the time for the depth buffer to be loaded back into memory and to analyze its pixels. When we explicitly half the dimensions of the depth buffer while maintaining its aspect ratio we get a time of 3.5 seconds for one image. This comes at the cost of reduced detail in the final image (figure 5.2.3).

5.3 Pulse Length

Increasing and decreasing the pulse length results in larger and more visible targets on the screen. This comes at the cost of detail and resolution in the image. To test the pulse length we use the same environment as in the previous test (figure 5.2.1).



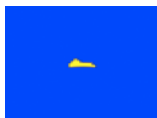
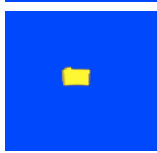
The expected result, as described in section 2.2.8, is that the size of the echo on the screen will be equal to the size of the pulse length. We will choose different pulse lengths, each with a different order of magnitude.

In table 5.3 we can see that the pulse length has a significant effect on the final result. As the pulse length increases by a factor of 10, the echo will increase with the same amount in the vertical direction. If we choose an absurdly large pulse length of 1000 meter and a maximum range of 3000 meters (1.6 nautical miles), we see that the echo takes up exactly 1/3 of the range (figure 5.3.1).

5.4 Specularity

In figure 2.3.13 the concept of specularity was illustrated with the concept of a slanted plate and a straight plate that reflected different amounts of light depending on the material. In this test we will use these same situations. We will use two environments: One with a slanted plate and

Table 5.5: Result of the pulse length tests

PL (seconds)	PL (meters)	Result	Time
0 μs	0		2.5 seconds
0.003 μs	1		2.5 seconds
0.03 μs	10		2.5 seconds
0.3 μs	100		2.5 seconds

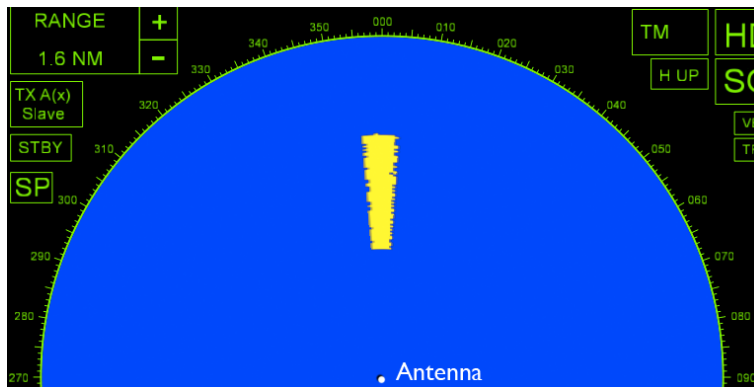


Figure 5.3.1: Result of a pulse length of 1000 meters and a range of 3000 meters. Echo is 1000 meters long.

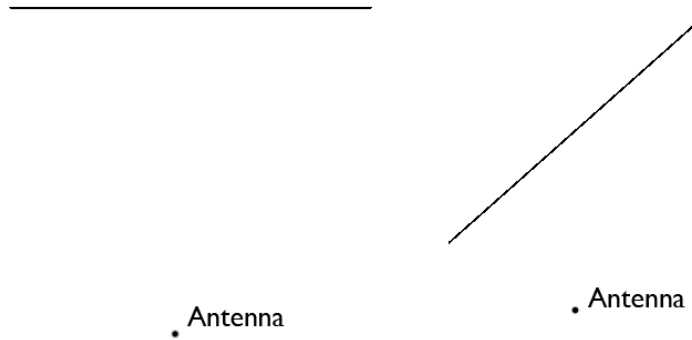


Figure 5.4.1: Situation for specularity test. Left: SinglePlate.rof. Right: SinglePlateSlanted.rof

Table 5.6: Result of specularity test

Table 5.7: Result of the pulse specularity test. The diffuse plate always is completely visible while the specular plate only reflects where the surface normal is pointing to the camera.

Model	Diffuse	Specular (shininess: 50)	Specular (shininess: 100)
SinglePlate.rof			
SinglePlateSlanted.rof			

one with a straight plate (figure 5.4.1). For both situations we will test three conditions: one condition where the plate is completely diffuse, one where the plate is specular with a shininess of 50 and one where the plate is specular with a shininess of 100.

In table 5.6 we see the expected result for a plate of diffuse material. When the specularity increases the visible part of the wall decreases. Only the part of the wall where the light does not bounce away on an angle becomes visible.

For the slanted wall we see that point *A* straight ahead of the antenna is only detectable when the wall is diffuse. A specular wall is too slanted at point *A* to return any signal back to the antenna.

5.5 Specular Reflections

In section 2.3.7, we gave an example where a large wall is next to a nearby small ship. When this situation occurs in real life, a ghost target of the little ship would appear behind the wall. The wall acts as a reflector and bounces the signal to the target and back. For this test we created the situation in figure 2.3.14 as an environment (figure 5.5.1).

In figure 5.5.2 we see that behind the wall at location *A*, a ghost ship appears that does not

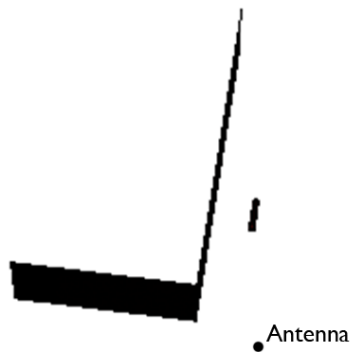


Figure 5.5.1: Situation used to test specular reflections: a large wall that should reflect the nearby ship.

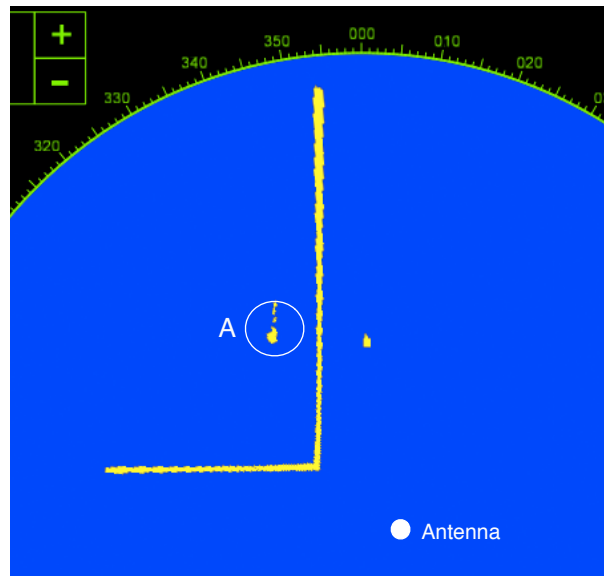


Figure 5.5.2: Result of specular reflection test. Both the real ship and a reflected ghost ship (location *A*) are visible.

Table 5.8: Result of specular reflection test

Model	Triangles	Time
BuildingReflection.rof	60	16.4 seconds

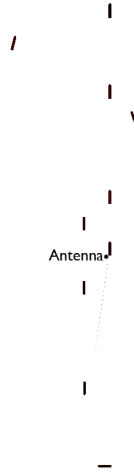


Figure 5.6.1: Environment for the gain test. Antenna surrounded by a collection of ships

Table 5.9: Result of gain test

Model	Triangles	Time
Ships.rof	560	3.1 seconds

exist in the original situation. This target is a result of the specular reflection extension. It correctly appears at the exact mirror position of the real ship. The ghost ship is even larger in this situation because it is viewed at an angle while for the real ship only the back can be seen.

Enabling the specular reflection extension has a severe impact on the running time of the algorithm. The time nearly quadruples when compared to the same situation, but with specular reflection turned off. The reason for this is that several extra frame buffer drawing and reading calls need to be made when the extension is enabled. As these are one of the most expensive operations, the running time increases significantly.

5.6 Gain Control

To test the gain, we will use a situation where the antenna is surrounded by a collection of other ships (figure 5.6.1).

The gain is a control that can weed out weaker targets and clutter from the picture. In this situation we have targets that are viewed from the side and targets that are viewed from the back or the front. Targets that are viewed from the side can reflect more energy to the antenna than the targets that are viewed from the front. As we decrease the gain we expect these smaller echoes to diminish and eventually disappear while the larger echoes remain prominent. We will test with three gain settings: 5%, 50% and 100%.

From figure 5.6.2 we see with the gain set at just 5%, the weak targets *A* and *B* have completely disappeared from the image, due to the fact that they are viewed from a head-on angle and are weaker on the screen. Targets *C* and *D* that are at the same distance as *A* remain visible, as they are viewed from the side and are stronger targets. As the gain increases, all targets increase in brightness until all targets are visible at 50% and become bigger at 100%.

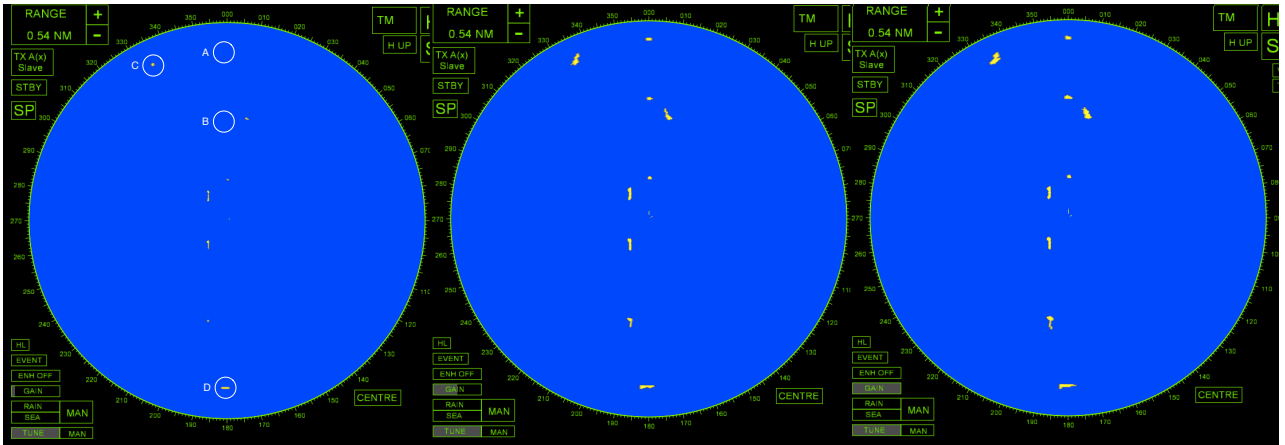


Figure 5.6.2: Result of gain test. Left: Gain at 5%. Middle: Gain at 50%. Right: Gain at 100%.

Table 5.10: Result of rain control test

Model	Triangles	Time
Ships.rof	560	3.2 seconds

5.7 Rain Control

The rain control can make targets visible within a large area of noise by using the difference in energy instead of the energy itself (section 2.4.3). To test this, we add a large amount of noise at every range simulating a severe rain shower. The amount of noise we add is a random value that is almost equal to threshold value, with a maximum deviation of 50% the threshold value. This makes the entire screen random garbage with the real targets hidden among them peaking above the threshold value. The rain control should in this situation be able to reduce this noise while retaining the echoes of the real ships.

The environment will be the same as the previous test (figure 5.6.1)

In the situation without the rain filter the screen is filled with noise from the rain shower. The real targets are hard to recognize in this image. When we apply the rain filter the noise becomes subdued while the real targets become more visible. Thus the rain control can successfully separate the noise (which has little energy variation) from the real targets (that have a lot of energy variation).

5.8 Sea

The sea control can adjust the threshold depending on distance. There are situations where nearby clutter such as waves can obscure targets. In these cases one wants to adjust the threshold but only for nearby targets (section 2.4.2).

In this test we will use the same situation as for the gain and the rain test (figure 5.6.1). As we increase the sea clutter we expect nearby targets to get fainter while targets further away remain visible. There will be three settings tested: Sea control at 0%, at 50% and at 100%.

In figure 5.8.1 we see that as the sea control increases, targets become increasingly fainter.

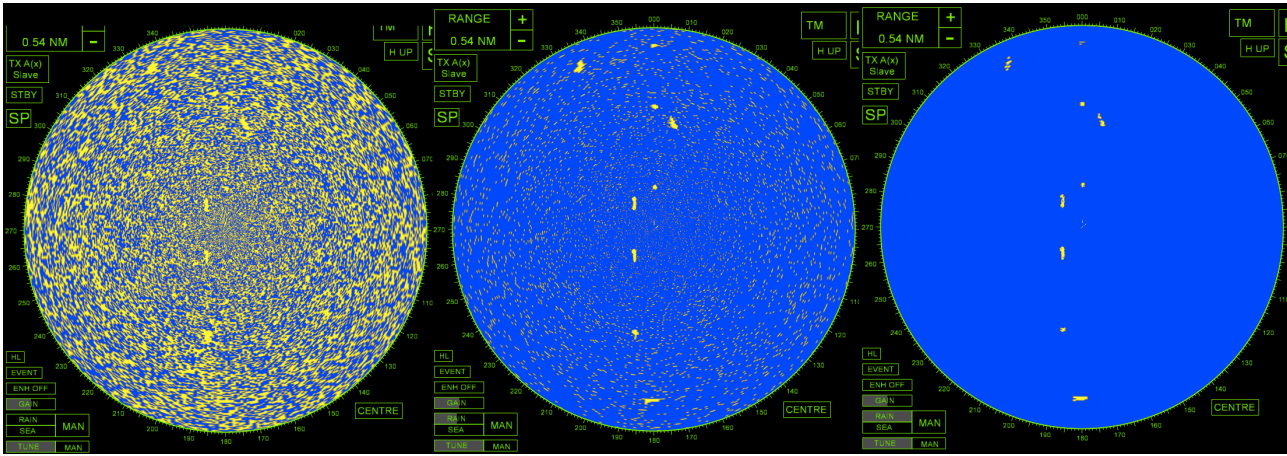


Figure 5.7.1: Result of rain control test. Left: Rain control at 0%. Middle: Rain control at 50%. Right: Rain control at 100%.

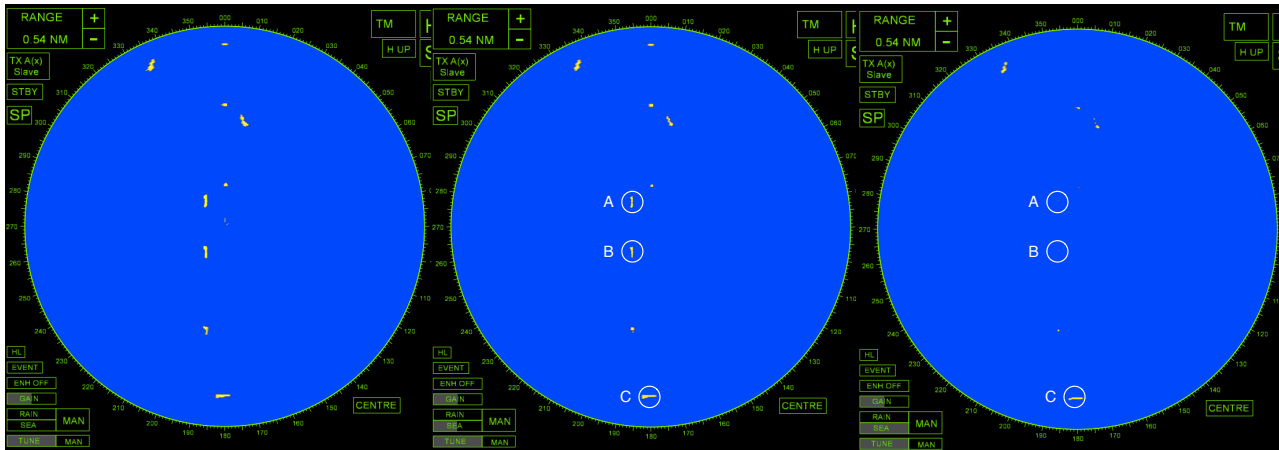


Figure 5.8.1: Result of sea control test. Left: Sea control at 0%. Middle: Sea control at 50%. Right: Sea control at 100%.

Table 5.11: Result of sea control test

Model	Triangles	Time
Ships.rof	560	3.0 seconds

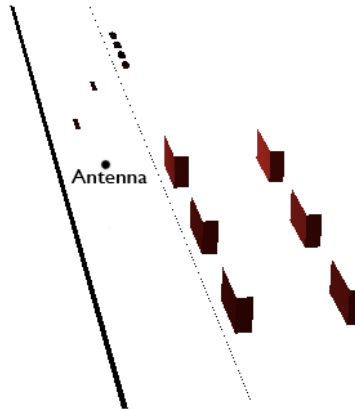


Figure 5.9.1: Situation used for simple environment test. Semi-realistic environment featuring two shores, two ships, six apartment buildings and four houses.

Table 5.12: Result of simple environment test

Model	Triangles	Time	Radar Signal Simulation
SimpleEnvironment2.rof	190	3.9 seconds	9.7 seconds

With a maximum sea control setting, nearby targets such as ship *A* and *B* disappear completely despite their relatively strong echo. Distant targets such as ship *C* remain visible on the screen and are relatively unaffected by the sea control.

5.9 Simple Environment

In the Radar Signal Simulation thesis, the most complex environment that was tested is a semi-realistic depiction of a river. This environment consists of 190 triangles and depicts two shorelines, two ships and 10 buildings (figure 5.9.1).

This environment is interesting because it closely resembles an environment often encountered by marine radar in real life. The shore line should be accurately portrayed as a straight line on the screen and the skyscrapers should occlude each other depending on the viewpoint.

When we compare our result to that of Radar Signal Simulation (figure 5.9.2), we see that both correctly occlude parts of the environment. The ship *A* correctly hides a part of the top left shore line *B* and the top left skyscraper hides skyscraper *C*. Another effect that is correct in both implementation, is that the the shoreline becomes less defined, the further it is away from the radar. The shore line that is far away is viewed from a grazing angle which can reflect less energy back to the antenna per m^2 than the shoreline close.

The running time of our application is also significantly lower than that of the Radar Signal Simulation, even though it runs on an inferior machine. The increased amount of triangles only has a small effect on the total time when compared to test 5.1. As 190 triangles is a trivial amount of triangles to render for a GPU, the decrease in speed from 2.5 seconds to 3.9 seconds can mostly be attributed to the fact that in this environment there is something to analyze and draw in every direction instead of only straight ahead.

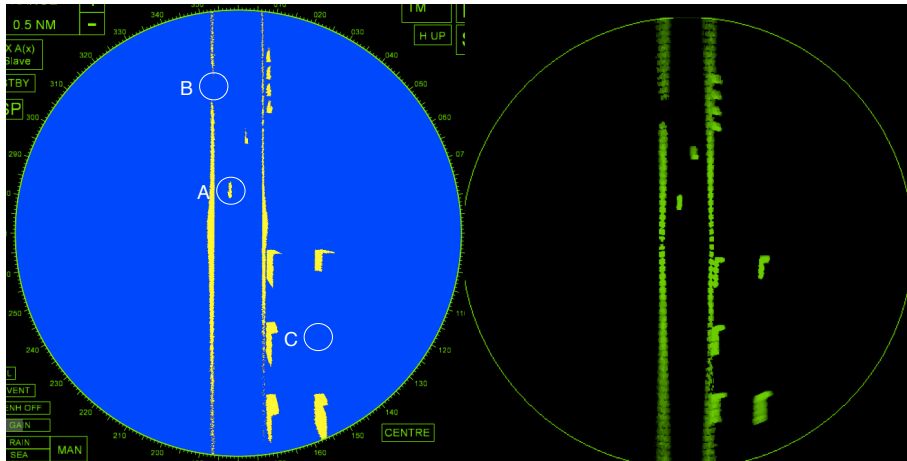


Figure 5.9.2: Result of simple environment test. Left: Result of our implementation. Right: Result of Radar Signal Simulation.

Table 5.13: Result of complex environment test

Model	Triangles	Time
Dover.x	136237	4.7 seconds

5.10 Complex Environment

In real life ship simulators environments are typically more complex than 190 triangles. Thanks to Ship Simulator developer V-STEP we have had the opportunity to test our algorithm on their Dover model which consists of 136.237 triangles. This environment consists of both the Port of Dover and the Port of Calais, as well as accurate surrounding terrains from England and France. The radar antenna is situated in the center of Port Dover (figure 5.10.1).

This environment tests as many aspects of our radar simulation as possible. It consists of both large and small objects and contains smooth terrain gradients, steep cliffs and rectangular structures. Most importantly it has a large number of triangles known to be a bottleneck for the Radar Signal Simulation.

In our result we see that a number of things are correctly simulated. The large vertical cliffs are the most prominent feature on the screen as they would be in real life. Their large surface area reflects a significant amount of energy back which makes them stand out from the other features. It can also be seen that the docks in the upper right region, occlude each other and that some of the far away terrain is visible as a smudge on the screen.

Even though the amount of triangles is now orders of magnitudes greater than in test 5.9, the running time has only increased with one second.

5.11 Complex Environment 2

During our research we had the opportunity to see a real radar system in action in Rotterdam on a pilot ship. We made a picture of the radar screen while it was stationary in the Berghaven. As V-STEP was also kind enough to supply a 3d model of the port of Rotterdam, we are able

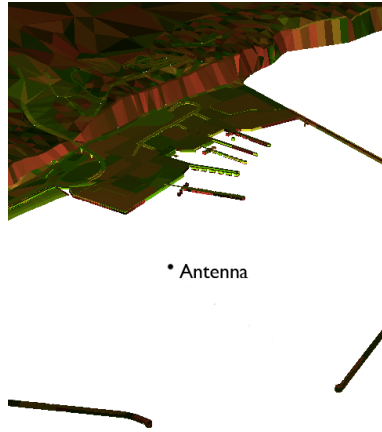


Figure 5.10.1: The antenna for the complex environment test is in the center of Port Dover.

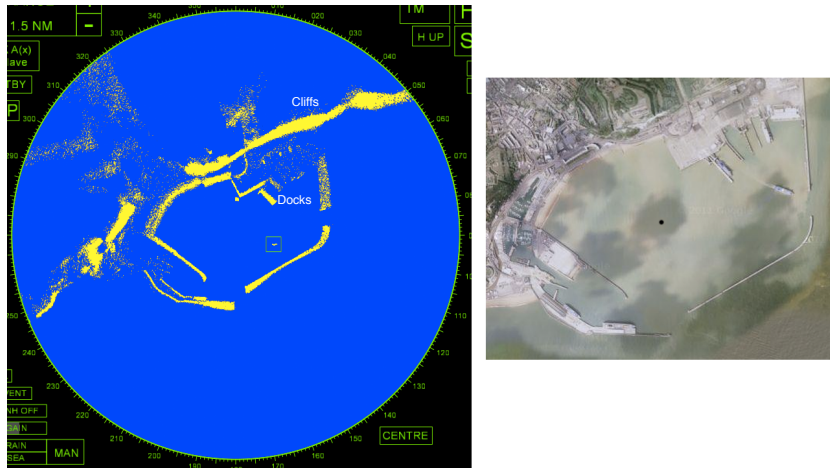


Figure 5.10.2: Result of complex environment test. Left: Result of our implementation. Right: Satellite view of Port Dover

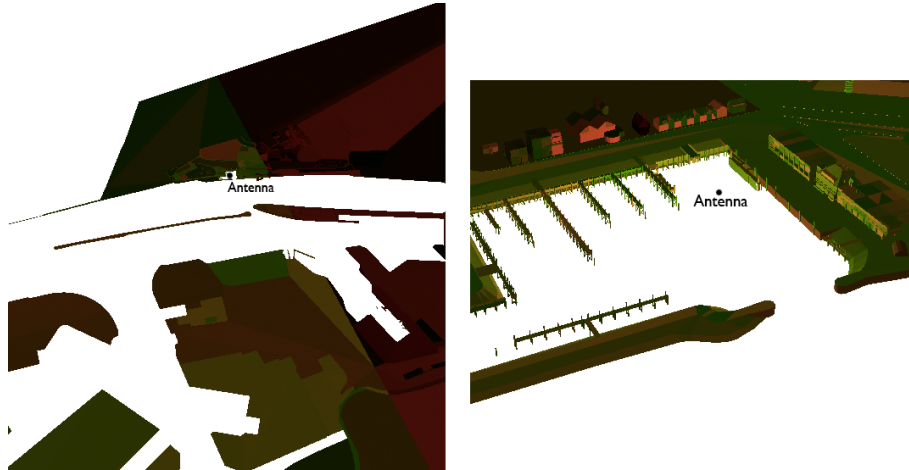


Figure 5.11.1: Left: 3d environment for the complex environment 2 test. Right: Close-up of buildings surrounding antenna.

Table 5.14: Result of complex environment 2 test.

Model	Triangles	Time
Rotterdam.x	46.772	3.8 seconds

to make a direct comparison between a real radar screen and our simulation with the same environment.

The environment consists of multiple shorelines and dams that occlude each other (figure 5.11.1). The two landmasses in the middle of the river are fairly low and should not impede line of sight with objects behind it. The immediate surroundings of the antenna contain buildings that obscure line-of-sight. Large parts right of the river should remain out-of-sight.

When we compare the real radar screen to our result (figure 5.11.2) it is easy to see that in the circled areas the real radar screen contains many more targets. This is because the environment that we use does not contain the many industrial storage buildings, cranes and docked vessels that were present in the real world. When we focus on the features that are present in both environments we see that they correspond fairly well. The dam in the center and the immediate surroundings are represented the same as well as most of the distant shorelines.

Another difference is that the real radar screen shows red and green dots. The red dots represent buoys, while the green represent AIS targets (Automatic Identification System, a system where every ship sends out its own GPS location). These are not represented on our simulator as we did not have the exact same data.

5.12 Extra information

To add a bit of realism to the simulator we spent some extra time to simulate the information that surrounds the radar image (figure 5.12.1).

The layout of this information is based on the Vision BridgeMaster, which is a popular series of radar hardware. Many features are currently working including the EBL and VRM overlays, heading, the target collision / speed information, own position, cursor position, vectors, trails

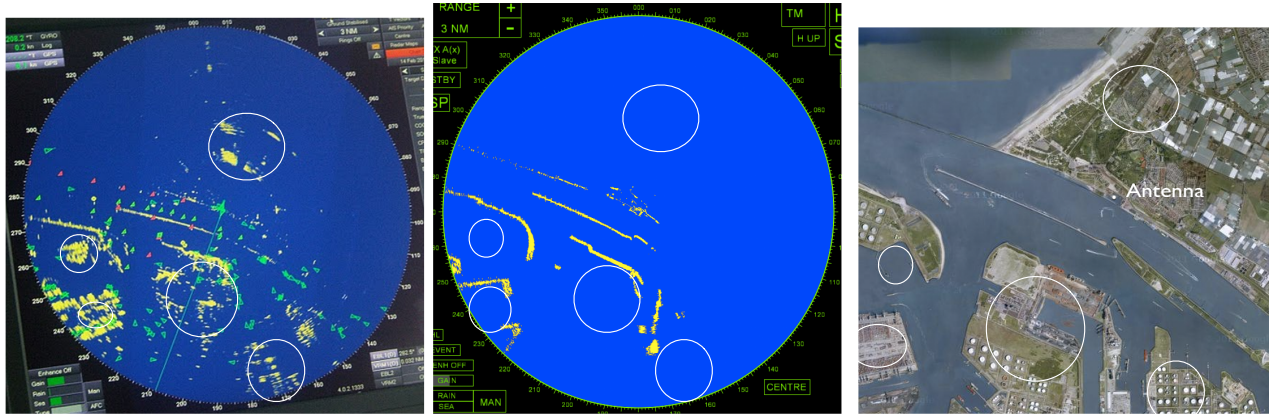


Figure 5.11.2: Result of complex environment 2 test. Left: Real radar screen. Middle: Result of our implementation. Right: Satellite view of area

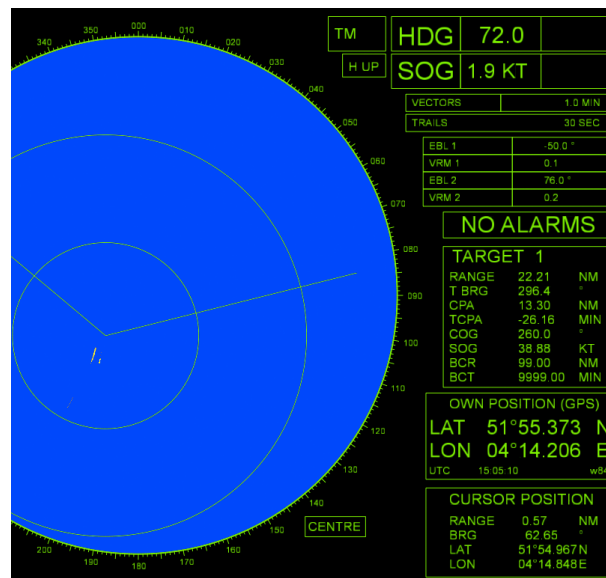


Figure 5.12.1: Extra information on the right displaying relevant information for a radar operator.

and the true motion / relative motion mode.

Chapter 6

Conclusions & Future Work

The goal of this project was to create a real-time realistic radar simulator. In the results section we demonstrated that the simulation is able to handle most situations in real-time. Most screenshots were generated by the simulator within 5 seconds. A real radar system can make a revolution every two seconds, which means that there is still need for improvement. We've tested the environment presented in section 5.11 on a computer with more high end specifications (Intel Core 2 Quad CPU@2.40GHz CPU, NVidia GTX 280 GPU). On this machine the time it took to simulate a single revolution was halved from four to two seconds, making the simulation as fast as a real radar.

We have also demonstrated that the running time of the algorithm does not heavily depend on the complexity of the environment. Increasing the number of triangles by a few order of magnitudes only increased the running time by a few seconds. Modern GPU's are optimized to render a large amount of triangles without any problem. As the environment we want to render is for the most part static, we can use environments that are as complex as modern game worlds. That means that the simulation can support environments up to millions of triangles [25].

Instead, the running time depends on the size of the rendered image. Through experimentation we have found that the most expensive part of the code is reading back the depth buffer to main memory and analyzing every pixel of it. By rendering a smaller picture the running time can be decreased at the cost of accuracy. A smaller rendered picture also means less detail in echoes as demonstrated in the side lobes test (section 5.2).

With regards to realism we have implemented the most important occlusion effect as well as a number of secondary effects. Occlusion (section 2.3.1) and echo scaling (section 2.3.2) both are an inherent property of the rendering technique as they come from the z-buffer and the width of the rendered picture respectively. Other effects such as specularity, side lobes and specular reflections needed to be added with some work-arounds, with varying degrees of realism.

When comparing the results of our implementations with that of the Radar Signal Simulation we can get comparable results, despite the radically different algorithm behind our method.

6.1 Future Work

There is one effect that we have described but that we have not implemented yet, which is the second trace effect (section 2.3.8). It should be possible to include this effect in by doubling the distance of the far plane and overlaying the distant half very weakly on top of the radar image. We decided not to spent the time implementing this due to time constraints and the relative rarity of the effect.

Another thing we have not implemented is many possible optimizations. The environment is for example always drawn as a static mesh from a draw buffer. This results in a lot of overdraw. A more efficient way to draw would be to use a binary partition space tree. This would only draw the relevant portion of the environment and would allow for more complex environments.

Another optimization would be to run the algorithm entirely on the GPU. The current implementation renders the image on the GPU and then copies the depth buffer back to main memory. But the act of copying something from the GPU to main memory is a large bottleneck. In a traditional graphics pipeline instructions only flows from the CPU to the GPU, allowing the GPU to run a few frames behind in executing commands. This allows it to run as efficiently as possible. But if we copy data back from the GPU, the CPU is forced to wait for all buffered existing commands to execute before it can send that data back. This is a very inefficient way to utilize the GPU. If it would be possible to run the parts that the CPU is currently processing also on the GPU (analyzing the depth buffer pixels and writing the result back to the texture) a large speed increase should be obtained. We have looked into this but have found the additional research and code complexity to be not worth the possible speed increase.

6.2 Evaluation

During this project we had almost immediate interest from V-STEP to integrate this technique in their professional ship simulator. Outside of this project we implemented our algorithm in their software. There it currently is the default radar simulation replacing a professional commercial radar simulation package from In2Sim. We think that this is a good indication that our algorithm is currently one of the best radar simulators.

Bibliography

- [1] Radar Origins Worldwide: History of Its Evolution in 13 Nations Through World War II. *By Jr. Raymond C. Watson*
- [2] Sorensen's Guide to Powerboats, Second Edition. *Erik Sorensen*. McGraw-Hill 2008
- [3] Radar Signal Simulation *Corné van der Pol TU/e* 2010
- [4] International Convention for the Safety of Life at Sea (SOLAS). Chapter V - Safety of Navigation. Section 2.3.2.
- [5] The Use of Radar at Sea. *Yehuda Amichai*. Page 124.
- [6] Radar and ARPA Manual: Radar and Target Tracking for Professional Mariners. *Alan G. Bole, Alan Wall, W. O. Dineley*. Page 9. Page 33.
- [7] Understanding Radar Systems. *Simon Kingsley, Shaun Quegan*. Page 28.
- [8] Modern Dictionary of Electronics. *Rudolf F. Graf*. Page 612.
- [9] Radar reflectivity of Land and Sea. *Maurice W. Long*. Page 224.
- [10] Radar Navigation and Maneuvering Board Manual. *Griffes, ProStar Publications, Incorporated*. Page 5.
- [11] A two-pass solution to the rendering equation: A synthesis of ray tracing and radiosity methods. *John R. Wallace, Michael F. Cohen, Donald P. Greenberg*. SIGGRAPH '87
- [12] Optimal depth buffer for low-cost graphics hardware. *Eugene Lapidous, Guofang Jiao*. HWWS '99 Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware.
- [13] Radar Image Simulation Based on 3d Scene Database in Marine Simulator. *Ren Hongxiang, Wang Kelun, Jin Yicheng*. Computational Intelligence and Software Engineering, 2009. CiSE 2009.
- [14] Bistatic Radar Imaging of the Marine Environment Part II: Simulation and Results Analysis. *Arnold-Bos, A.; Khenchaf, A; Martin, A.*. Geoscience and Remote Sensing, IEEE Transactions on November 2007.
- [15] Integrated Bridge Systems Volume 1: RADAR and AIS. *Dr. Andy Norris*. Page 7.
- [16] Computer Processing of Remotely-Sensed Images: An Introduction. *Paul Mather, Magaly Koch*. Page 320.

- [17] Pattern limitations in multiple-beam antennas. *W. White*. Antennas and Propagation, IRE Transactions on, 1962.
- [18] Illumination for computer generated pictures. *BT Phong*. Communicatinos of the ACM, 1975.
- [19] Transas Radar/ARPA training. http://www.transas.com/products/simulators/sim_products/navigational/conventional/arpa/
- [20] In2Sim Radar Simulator. http://www.nautissim.com/public/files/products/downloads/21/Nautis_Radar-ARPA_Brochure_v5_online.pdf
- [21] Modern Radar Systems. *Hamish Meikle*. Page 370.
- [22] Advanced Marine Electrics and Electronics Troubleshooting: a Manual for Boatowners and Marine Technicians. *Edwin Sherman*. Page 185.
- [23] Weather radar: Principles And Advanced Applications. *Peter Meischner*.
- [24] Computer graphics: theory into practice. *Jeffrey J. McConnell*. Page 197.
- [25] The Application of Games Technology Within Business. *Kane Forrester*. Page 20.
- [26] Rendering synthetic objects into real scenes: bridging traditional and image-based graphics with global illumination and high dynamic range photography. *Paul Debevec*. SIGGRAPH '08 ACM SIGGRAPH 2008 classes.
- [27] Image obtained from <http://www.radartutorial.eu/11.coherent/co04.en.html> and used per the GNU Free Documentation License as available on <http://www.gnu.org/licenses/fdl.html>
- [28] Image obtained from http://www.dft.gov.uk/mca/mcga07-home/shipsandcargoes/mcga-shipsregsandguidance/mcga-windfarms/offshore-renewable_energy_installations/mcga_north_hoyle_windfarm_report/mcga_north_hoyle_windfarm_report_section6_7.htm and used with permission per the Copyright Maritime and Coastguard Agency 2010.

Appendix A

Depth Buffer

A buffer is a two dimensional array of values that is used by the graphics card to help in the rendering process. The two most important buffers are the color buffer and the depth buffer.

The *color buffer* is the final composite image that is displayed on the screen to the user. It is constructed by drawing every triangle one by one in the color buffer. The order in which this is done is important. The last triangle that is drawn will appear to be in front of all other triangles, even if that triangle is actually behind them in the original scene. In order for this scheme to work we need to first sort all triangles from furthest to closest to the camera before we draw them to the color buffer. This algorithm is known as the Painter's algorithm [24].

There are a number of disadvantages to the painters algorithm. The main problem is that it is slow to sort all the triangles in the correct order. Modern graphics cards have to be able to render millions of triangles per frame. It would be too costly for them if they should have to order them from furthest to closest for every frame. Another problem is that sometimes it is not clear whether a triangle will be in front or behind another triangle. Two triangles might intersect or they might overlap in a manner in which it is impossible to assign a correct order to render them in.

To solve this problem modern graphics card employ the use of a depth buffer. This is an invisible buffer the same size as the color buffer that defines for each pixel how far away it is from the camera.

The depth buffer is initialized with every pixel at an infinite depth. When a triangle is drawn to the color buffer it is checked for every pixel if it is behind or in front of the existing drawn picture. If a pixel is behind the existing depth in the depth buffer it is discarded. If it is in front is filled in with the color of the triangle and the depth buffer is updated to the new depth.

The depth buffer is always in sync with the color buffer. That means that if all triangles are

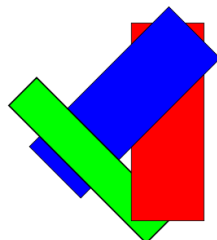


Figure A.0.1: Instance where the painters algorithm will fail.

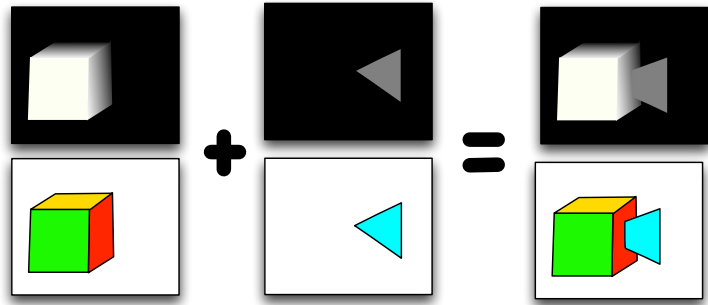


Figure A.0.2: Adding a triangle to the scene with the help of a depth buffer.

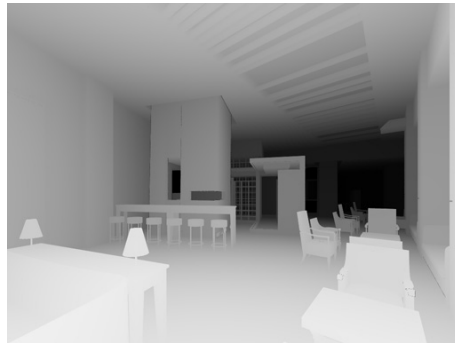


Figure A.0.3: Visualization of a depth buffer. The whiter a pixel is the closer it is to the camera.

drawn and the color buffer is displayed the depth buffer will contain full information on how far away each pixel is. Normally the depth buffer is erased to be used for the next frame, however for our rendering method this data will prove to be invaluable.

Appendix B

Radiosity

B.1 Global Illumination

In the real world lights illuminate the world around us allowing us to see it. Objects that are directly illuminated by a light source appear very bright, however objects that are in the shade and not directly illuminated, do not appear to be totally black. Light still reaches it because they can reflect off of nearby objects that are directly illuminated. This technique is called *global illumination* and it can greatly enhance the realism of a rendered three dimensional scene.

B.2 Radiosity algorithm

Radiosity is a rendering algorithm that renders a scene with global illumination. It works by dividing the scene into small *patches* (usually squares or triangles) and using each patch in the three dimensional scene as both a light receiver and light transmitter for multiple passes.

Light sources in the scene are also modeled as patches. These behave in a special way in that they always transmit light regardless of whether they have received any light.

In the first pass only the light source patch radiates light towards the other patches, in the second pass the light source and all patches that received light in the previous pass radiate light to all patches. In the third pass all patches again radiate light to all other patches. This process repeats for a fixed number of passes.

The amount of light a patch radiates to another patch depends on a few factors. If patch a wants to radiate light to patch b but there are patches in between that block the line of sight between them then no light will be radiated from a to b .

If there is line of sight then the amount of light that is radiated between a and b depends on the size, distance and their orientation of the patches. If the patches are for example part of the same wall and facing the same direction then it is impossible for patch a to radiate light to patch b as they can not see each other.

All patches have a *light value* associated to them. This is how much light they can radiate in the next pass. In the first pass all patches have a light value of 0 and are completely dark. Only the light source patches have a non-zero light value. The light value of light source patches will remain constant throughout the passes of the algorithm and typically have a much higher value than the other 'normal' patches.

Each pass each patch a will radiate away all the light that it has towards all other patches. The new light value of a depends completely on the light that is radiated towards each patch a ,

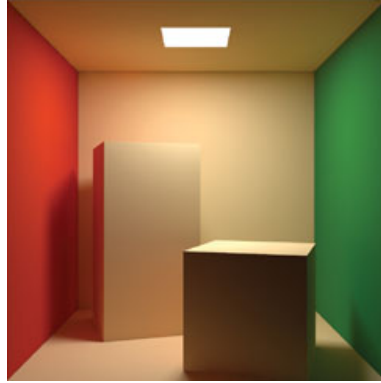


Figure B.1.1: Scene rendered with global illumination. The light indirectly illuminates the shadow areas cast by the cubes via the walls, causing the color to bleed to these areas.

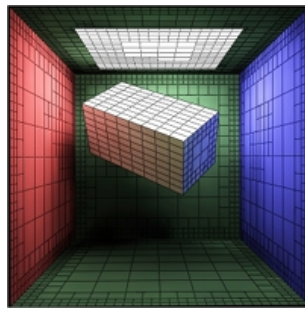


Figure B.2.1: Dividing a simple geometry into a large number of small patches

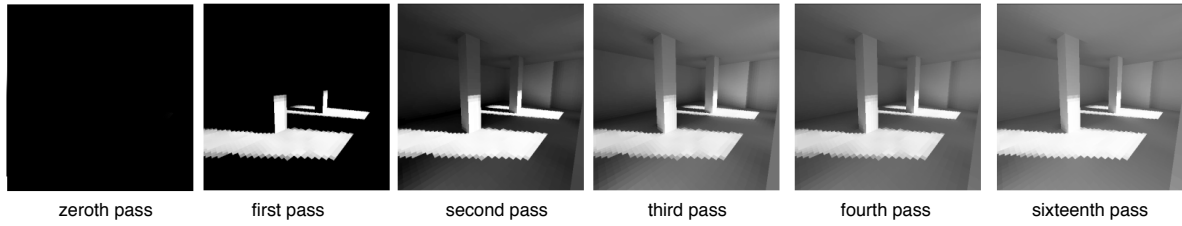


Figure B.2.2: Passes of the radiosity algorithms. The patch acting as a light source is offscreen outside the window.

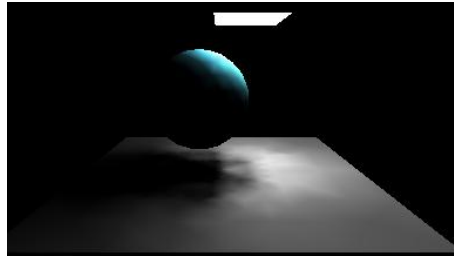


Figure B.2.3: Using patches that are too big results in undetailed and aliased shadows

the old light value is not relevant for the new value as it was all radiated away towards other patches.

As more light enters the scene from the light source patches the average light value of a patch will increase, and the scene will appear to be brighter and brighter. After a few passes all patches are bright enough to be seen.

Advantages The major advantage of radiosity is that it is *view-independent*. The algorithm has to be run only once on the scene to be able to render it from any angle. The amount of light of each patch in the final pass can be stored and used when the scene is drawn without running the algorithm again. This is in contrast with view-dependent algorithms like raytracing where pixels are used as the basis of the algorithm instead of patches. Every time the virtual camera moves around, the raytracing algorithm has to be run again because each pixel now casts a ray in a slightly different direction. The radiosity algorithm does not have to be run again because moving the camera does not impact the light value of each patch.

Disadvantages The main disadvantage of radiosity is that the quality of the image is directly related to size of the patches. If the patches are too big then a lot of artifacts will occur. At each pass each patch can only have a constant value of light that it receives. If there is a lot of shadow detail on a single patch then that detail will be lost.

In order to get a realistic lighting and detailed shadows the size of each patch should be as small as possible. Ideally, each patch will not appear larger than a single pixel when rendered from most viewpoints. The typically large number of patches in a scene severely increase the running time of the algorithm. The size of the patches is a trade-off, if the number of patches is too low shadows will appear to be undetailed while if the number of patches is too high the running time will be too long.

Another disadvantage is that the result of the radiosity algorithm is dependent on the configuration of the scene and the position of the lights. We are free to move the camera around without running the algorithm again but we cannot move a light source or a wall around. While

Radiosity	
Advantages	View independent Realistic diffuse lighting
Disadvantages	High complexity Lighting dependent No specular lighting

Table B.1: Radiosity algorithm advantages & disadvantages

this does not happen in most cases where the radiosity algorithm is used it turns out to be a critical disadvantage for use in a radar simulation.

Finally the radiosity algorithm is only able to light a scene with diffuse lighting. All light that a patch receives is all added to its light value, the direction from which the light came is not stored. As a patch transmits its light it does so in a perfectly diffuse manner, it sends an equal amount of light in each direction. It cannot do any specular reflection as the algorithm would need to know from which direction most light came, something which is explicitly not stored.