TRULY RANDOM NUMBER GENERATOR FROM OCEAN EVENTS
N476 Netto, Gustavo Marques
Truly random number generator from ocean events / Gustavo Marques Netto. – Niterói, RJ : [s.n.], 2016.
51 f.

Trabalho (Conclusão de Curso) – Departamento de Computação, Universidade Federal Fluminense, 2016.
Orientador: Leandro Augusto Frata Fernandes.


CDD 006.42
TRULY RANDOM NUMBER GENERATOR FROM OCEAN EVENTS

Work submitted to the Computer Science Bachelor Course at Universidade Federal Fluminense as a partial requirement to obtain the title of Bachelor in Computer Science.

Advisor: Prof. Leandro Augusto Frata Fernandes

Niterói-RJ
2016
GUSTAVO MARQUES NETTO

TRULY RANDOM NUMBER GENERATOR FROM OCEAN EVENTS

Work submitted to the Computer Science Bachelor Course at Universidade Federal Fluminense as a partial requirement to obtain the title of Bachelor in Computer Science.

Approved by:

[Signature]
Prof. LEANDRO AUGUSTO FRATA FERNANDES, D.Sc. - Advisor
UFF

[Signature]
Prof. MARCOS DE OLIVEIRA LAGE FERREIRA, D.Sc.
UFF

[Signature]
Prof. JOSÉ RICARDO DE ALMEIDA TORREÃO, Ph.D.
UFF

Niterói-RJ
2016
Thanks

I would like to thank those who helped during this work, in special professor Leandro A. F. Fernandes. I would also like to thank Mads Haahr from RANDOM.ORG for supplying data for the tests.
Abstract

Random number are constantly present in our daily lives. They can be used in the lottery, making you rich, or in encryption algorithms, responsible for protecting billions of bytes of data every day. However, generating this numbers is not a simple process. A machine like a computer is deterministic by nature and random numbers are supposed to be the opposite, unpredictable. In this work, we propose a technique that generates truly random numbers from natural phenomena. In this case, images from the movement of the ocean water captured by a camera. We make the assumption that the wind influence over the water creates enough randomness, so the generation passes a set of statistical tests that assure the quality of the generated numbers. In fact, the normal vectors of the water surface are used to generate the numbers. We employ geometric reconstruction to minimize the image distortions introduced by to the perspective of an area of the water surface. The reconstruction also results in the viewer direction for the rectified image. This is used in a new approach for Shape from Shading. In this technique, based on the work of Shah and Ping [1], we remove the necessity for the illumination direction as we deal with outdoor scenes in this work.

Keywords: Random Numbers, Shape from Shading, Truly Random Number Generation, Geometric Reconstruction, Surface Reconstruction.
Contents

Abstract vi

List of Figures viii

1 Introduction 1
  1.1 Related Works .................................................. 3

2 Theoretical Concepts 5
  2.1 Pinhole Camera Model ........................................... 5
  2.2 Shape from Shading ............................................. 6
  2.3 Concepts of Statistics ......................................... 7
    2.3.1 Chi-square Distribution .................................. 7
    2.3.2 Chi-square Test (Goodness to fit) ....................... 8

3 Surface Reconstruction 9
  3.1 Results on Water Surface Reconstruction .................... 16

4 Random Number Generation 23
  4.1 Methodology for Testing Random Number Generation ........... 24
    4.1.1 Mathematical Tests ....................................... 25
    4.1.2 NIST Tests ............................................... 27
  4.2 Results on Random Number Generation ........................ 37

5 Final Remarks 40
  5.1 Limitations .................................................... 40
  5.2 Future Works .................................................. 41

Bibliography 42
List of Figures

1.1 Overall view of the technique ........................................... 4

3.1 Flow diagram of the calibration process to get the transformation from world to image coordinates ........................................... 10

3.2 Warped image positioned in the reconstructed world, in scale .......... 11

3.3 A screen-shot of the tool used to calibrate the horizon line in the image ........................................... 12

3.4 Vanishing line in the image, the region of the image that will be projected and the region projected ........................................... 14

3.5 Flow diagram of the shape from shading proposed ........................................... 15

3.6 Comparison showing the effect of the sky color on the water ................. 15

3.7 Set of images with reconstructions with average waves ........................ 18

3.8 Normal vectors of Figure 3.7d represented as a colored image ............... 19

3.9 Set of images with reconstructions with high waves .......................... 19

3.10 Normal vectors of Figure 3.9d represented as a colored image ............... 20

3.11 Set of images with reconstructions with short waves .......................... 20

3.12 Normal vectors of Figure 3.11d represented as a colored image ............... 21

3.13 Set of images with reconstructions with blobs .............................. 21

3.14 Normal vectors of Figure 3.13d represented as a colored image ............... 22

4.1 Flow diagram of the generation of random numbers from the normal vectors of an image ........................................... 24
Chapter 1

Introduction

In our daily lives, we deal with random numbers in many situations, Video games, simulations, gambling and cryptography are some examples. These numbers are supposed to represent unique events. Their combinations are unpredictable and most of the times non-reproducible. A simple example would be the lottery. It consists in generating numbers and the person who guesses them takes the jackpot. One of the widest known lottery game is Powerball. Its prize can reach more than one hundred million dollars and it includes almost all states of the USA. The need for random numbers is clear here and with this prize, the process that generates them has to be reliable. Otherwise, the whole game could be compromised.

An important application is in the security area. Usually, if we want to send a private information we use algorithms to encrypt data. If anyone tracks the message, this person will not be able to understand the content. One of the most used algorithms is the RSA, developed by R. Rivest, A. Shamir, and L. Adleman in 1978. It uses two large prime numbers as a core part of it, which means that knowing these numbers the encryption can be broken. In addition, proving that an integer is prime is much simpler than factoring numbers to craft prime numbers [2]. Therefore, the method to create these numbers is basically generating numbers and checking if they are prime, if not, the process restarts.

Since random values are so important, the way that they are created becomes a problem. In most cases, a computer is using this numbers and are generating them, too. However, computers are deterministic in its core. Unless an external event happens, the computer is supposed to follow the same set of instructions, always in the same order and manner. Additionally, algorithms that generate numbers still depend on an external seed.
The seed is a number responsible for defining the behavior of the generator. So, a pair
generator/sequence will always generate the same sequence of pseudo-random numbers.
These generators are known as Pseudo-Random Numbers Generators (in this work, they
will be mentioned as PRNG).

PRNGs apply a series of transformations to remove the dependence among previ-
ously generated numbers and the subsequent ones. They also tend to have better statis-
tical results as they force the independence among numbers. On the other hand, knowing
the seed implies in breaking the non-determinism. So the generation process becomes de-
terministic. In this case, they still depend on a Truly Random Number Generator (they
will be mentioned as TRNG in this work).

TRNGs need to come from really random sources. Thus, most of them come from
nature, as it will be presented in this manuscript. Natural events have so many variants
that they become unpredictable. For instance, some software asks the users to make
movements with the mouse to create randomness. Even if the user tried, he/she would
not make the same pattern twice because people do not have that much control over their
muscles. In this work, our approach makes use of images of water movement as a source
of randomness, resulting in a TRNG.

We propose a technique that generates random numbers from the angles between
pairs of direction that are perpendicular to the water surface. For example, given a surface,
for every point there is a direction perpendicular to it. We can write these directions as
vectors and they are called normal vectors. By assuming that the water movement is
random, the normal vectors will be too, as well as the angles between them. To estimate
the water surface, we use a single image from the water. So, from a single image, we are
capable of generating $n$ angles, where $n$ is the number of points (or pixels, in this context).
By applying a threshold over the angles, we produce bit values, one per angle. We build
random integers numbers by binding the random bits.

The two main problems become: (i) getting the normal vectors from the image of
the water surface and (ii) generating random numbers from the normal vectors. Figure 1.1
illustrates the overall process. First, to estimate the normal vectors we deal with a problem
known as Shape-from-Shading. It consists on deriving a 3-D scene description from one
or more 2-D images. We attack this problem by using projective geometry to select a
region of the water surface and then we rectify this part, mitigating the distortion caused
by the perspective view. To perform this, we need to reconstruct geometrically the mean water plane, this also results in getting the viewer direction from any pixel of the rectified image. This is possible because we become able to get the approximate coordinates of the camera and the points on the water plane. With this information, we are able to apply a variation of the Shape from Shading problem proposed by Ping and Shah [1] (Chapter 3). This technique refines the estimation of the water heights (a discrete height map) by iterating over a function whose output is the height map. The process tries to minimize the difference between the input image and the estimated reflectance with the estimated height. In our case, it deals with uncertainties about illumination that an outdoor scene presents. Hence, for each image a discrete height map is reconstructed. We then get the angles between pairs of normal vectors calculated from the height map (Chapter 4). We convert these angles to random bits applying a threshold. In order to validate our approach, we present two sets of tests that show the randomness and we compare it against the performance of other generators (Chapter 4).

For a better understanding of the proposed technique, we introduce in Chapter 2 concepts that are going to be used in this work. In Chapter 5, we present our final remarks, limitations of the work and some opportunities for improvement.

1.1 Related Works

The technique employs two different steps that were previously approached by other authors. The first is Shape from Shading to reconstruct surfaces from images of water. Although, the general algorithm used here was defined by Ping and Shah [1], different approaches, even more general, were developed afterward. The second is the generation of truly random numbers given a measurement of a natural event.

Li et al. [3] used Shape from Shading directly as the first step to generate artificial water surfaces from a single video sequence. Not only the reconstruction is used but also optical flow as flow divergence is employed to compute the change in surface height. However, in terms of geometric reconstruction from a single image, details about the illumination and other assumptions were not mentioned in their paper, making the reproduction of the work challenging. Another approach was developed by Quan [4]. It assumes Phong’s Reflection Model for the water and perpendicular illumination direc-
Figure 1.1: The technique takes a picture from a water surface, the Guanabara bay, (b) applies geometric transformations to minimize the effect of perspective, (c) reconstructs the water surface from the shading, (d) compares pairs of normal vectors from the surface and (e) applies a threshold over the angles to get random bit values.

On the other hand, the illumination assumption and the direct dependence on the image derivatives create poor initial reconstructions.

On the generation of truly random numbers, the main examples come from particulars behaviors of natural phenomena. The first is Hotbits [5], an on-line service designed by John Walker in 1986. The generators use the randomness of the decay of Caesium-137 to generate the numbers. Even though it is possible to obtain an expected time to decay into Barium-137, the real moment is unpredictable. Consequently, it compares if the last decay time was greater than the previous one. If it was, then a certain bit value is generated. The source is specific, and the hardware necessary for such activity is, at least, inaccessible to most users. Another on-line service is RANDOM.ORG [6]. It is probably the one with more resources and information about. It uses atmospheric noise captured by radio devices to generate the numbers. The statistical tests applied are the same that were employed in this work. In terms of the algorithm or how the bits are extracted, neither the company or any publication make it publicly available.
Chapter 2

Theoretical Concepts

In this chapter, we introduce concepts that will be used throughout this work. First, Section 2.1 presents the pinhole camera model, it allows us understanding how the 3-D view is projected into the 2D image plane. This will be used to reduce the perspective distortions over the water surface and to estimate the viewer direction on Chapter 3. After this, Section 2.2 presents a classical solution for the Shape from Shading problem from a single image [1]. This work presents important topics that will be used in Chapter 3. Finally, Section 2.3 presents a probabilistic distribution, know as Chi-square distribution. Associated with it, there is an important statistical test, widely used by us in Chapter 4.

2.1 Pinhole Camera Model

Considering the 3-D space of the world and a point in world coordinates and an Euclidean coordinate system to express the 3-D space. The pinhole camera model applies the following transformation to map \( W \) to the image plane

\[
G = KR[I| - C]W = PW, \tag{2.1}
\]

where \( W = [X,Y,Z]^T \), \( G = [x,y,z]^T \) are the coordinates of the mapped point in 2-D (homogeneous coordinates, a third coordinate is included to indicate the homogeneous plane). \( P \) is the camera matrix, which is given by \( KR[I| - C] \) [7]. It maps world to image coordinates. The \( 3 \times 3 \) matrix \( K \) has the intrinsic parameters of the camera (camera parameters that
are internal and fixed to a particular setup) and it is written as

\[
K = \begin{bmatrix}
a_x & s & p_x \\
0 & a_y & p_y \\
0 & 0 & 1
\end{bmatrix},
\] (2.2)

where \(a_x\) is the multiplication of the camera focal distance and the ratio of pixels by distance on the \(x\)-axis of the camera’s sensor. The same is applied to the \(y\)-axis, for \(a_y\).

For the remaining variables, \(s\) is the skew of the pixel grid. Additionally, \(p_x\) and \(p_y\) are the offset between the origin of the image and the principal point (the intersection between the line passing through the camera center and the image plane), in pixels. The matrices \(R\) and \([I \mid -C]\) are responsible for centering the world on the center of the camera and for aligning the world axis to the camera’s axis. So they apply respectively a translation and a rotation to do it. \(R\) is a \(3 \times 3\) matrix and it is a typical 3-D rotation matrix, aligning the camera axis to the world axis. For the translation, the matrix is \(3 \times 4\), \([I \mid -C]\), where \(I\) is a \(3 \times 3\) identity sub-matrix and \(C\) is a \(3 \times 1\) vector expressing the cartesian location of the center of the camera in world coordinates.

### 2.2 Shape from Shading

Shape from Shading is a problem that consists on recovering the height map from surfaces using 2-D images. In this case, we focus on the solution proposed by Shah and Ping [1]. This technique uses a single image as input. It iterates over an equation that estimates the discrete height map of the surface.

The iterative process occurs over this equation, in this case the previous discrete height \((Z^{n-1}(x,y))\) starts as 0 for every position and it defines the next estimation, \(Z^n(x,y)\).

\[
Z^n(x,y) = Z^{n-1}(x,y) - \frac{f(Z^{n-1}(x,y))}{\partial f(Z^{n-1}(x,y))},
\] (2.3)

and

\[
f = I - R,
\] (2.4)

where \(I\) is the input image and \(R\) is the calculated reflectance (the ration of light reflected from the surface to the light incident upon it). According to [1], the derivative in
Equation (2.3) is
\[
\frac{df(Z^{n-1}(x,y))}{dZ(x,y)} = \frac{(\frac{\partial Z^{n-1}}{\partial x} + \frac{\partial Z^{n-1}}{\partial y})(\frac{\partial Z^{n-1}}{\partial x} i_x + \frac{\partial Z^{n-1}}{\partial y} i_y + 1)}{\sqrt{(1 + \frac{\partial Z^{n-1}}{\partial x}^2 + \frac{\partial Z^{n-1}}{\partial y}^2)^3}} \sqrt{1 + i_x^2 + i_y^2}\]
(2.5)
where Z is the height map, n indicates the iteration. This technique assumes Lambertian surfaces, i.e., surfaces whose luminance is isotropic, uniform in all directions. It means that the lighting is diffuse, with no highlights or reflexes on the surface. The final reflectance depends only on the angle between the illumination and the normal vector of the surface.
The illumination direction is modeled as \( \tau \in [0, 2\pi] \), the tilt (azimuth) angle, and \( \delta \in [0, \pi/2] \), the slant angle (angle between the vector and \([0,0,1]^T\)). The illumination direction in Cartesian coordinates is defined by
\[
[i_x, i_y, i_z]^T = [\sin \delta \cos \tau, \sin \delta \sin \tau, \cos \delta]^T.
\]
(2.6)
Hence, the reflectance is defined as basically the dot product between the unitary illumination direction and the unitary normal vector
\[
R = \frac{[\cos \tau \sin \delta, \sin \tau \cos \delta, \cos \delta] \cdot [\frac{\partial Z^{n-1}}{\partial x}, \frac{\partial Z^{n-1}}{\partial y}, 1]}{\sqrt{1 + \frac{\partial Z^{n-1}}{\partial x}^2 + \frac{\partial Z^{n-1}}{\partial y}^2}},
\]
(2.7)
where \( \frac{\partial Z}{\partial x} \) is \( Z^{n-1}_{x,y} - Z^{n-1}_{x-1,y} \) for every \( x \in [1, 2, \cdots, n-1] \) and \( \frac{\partial Z}{\partial y} \) is \( Z^{n-1}_{x,y} - Z^{n-1}_{x,y-1} \) for every \( y \in [1, 2, \cdots, n - 1] \).

2.3 Concepts of Statistics

2.3.1 Chi-square Distribution

Given \( n \) independent random variables with standard normal distribution, \( u \). The sum of their squares has a Chi-square distribution with \( n - 1 \) degrees of freedom,
\[
\sum_{i=1}^n u_i^2 = q \sim \chi^2_{n-1}.
\]
(2.8)
The probability density function in this case is defined as
\[
f_q(x; k) = \begin{cases} 
  \frac{x^{(k/2-1)}e^{-x/2}}{2^{k/2}\Gamma\left(\frac{k}{2}\right)}, & x > 0 \\
  0, & \text{otherwise}
\end{cases}
\]
(2.9)
where $\Gamma (x) = \int_0^\infty s^{x-1}e^{-s}ds$ is known as the gamma function and $k = n-1$. The cumulative distribution function of the distribution is defined as

$$F_Q(x,k) = \frac{\gamma\left(\frac{k}{2}, \frac{x}{2}\right)}{\Gamma\left(\frac{k}{2}\right)} = p\left(\frac{k}{2}, \frac{x}{2}\right),$$

(2.10)

where $p$ is the incomplete gamma function

$$p(a,x) = \frac{1}{\Gamma(a)} \int_0^x t^{a-1}e^{-t}dt$$

(2.11)

and $\gamma$ is the lower incomplete gamma function

$$\gamma(a,x) = \int_0^x t^{a-1}e^{-t}dt.$$  

(2.12)

### 2.3.2 Chi-square Test (Goodness to fit)

This test is also known as Pearson’s Chi-square test and it evaluates how the analyzed set of samples fits a probability distribution. According to Pearson’s theorem [8], given a set of samples $p_1, p_2, ..., p_n$ with frequencies $m_1, m_2, ..., m_n$, we can measure the deviation of the sample from an expected value. The normalized sum of squared deviations approximates to a Chi-square distribution with $n-1$ degrees of freedom (2.13), and it is called test statistics ($X^2$):

$$X^2 = \sum_{i=1}^n \frac{(m_i - np_i)^2}{np_i} \rightarrow \chi^2_{n-1}.$$  

(2.13)

So, the test tries to confirm a null hypothesis ($H_0$, the samples fitting the distribution), the opposite will be the alternative hypothesis ($H_1$, the samples not fitting the distribution). The result of (2.13) is a statistic value that compared to the cumulative Chi-square distribution with $n-1$ degrees of freedom tells how likely that value or an even more extreme value can happen. This comparison uses the following equation:

$$p\text{-value} = Pr(X_p \geq X^2|H_0),$$

(2.14)

where $p$-value is the probability of such extreme event and $X_p$ is the random variable which $X^2$ (the test statistic) is a value of. This probability can be calculated with the formula defined in (2.10), with $n-1$ degrees of freedom and $X^2$ as the value of the variable. If the $p$-value is less than a level of significance (typically 1%), $H_0$ is rejected.
Chapter 3

Surface Reconstruction

The first step of our random number generation procedure consists of the geometric reconstruction of the water surface from an image. We capture the image of the water. However, in this image we define the $x$ and $y$ world axis parallel to the mean water plane, consequently, the $z$ axis corresponds to the normal vector of the plane, pointing to the sky. The camera is positioned over the mean water plane. We define the origin of the coordinates system of the world as the resulting orthographic projection of the camera on the plane. This implies that $C = [0,0,h]^T$, where $h$ is camera height. This setup is presented on Figure 3.2, the rectangle selected and presented in this figure is just a piece of the image.

The model used is explained more in depth the Section 2.1. The main objective is to define a transformation that mitigates the effects of perspective on a region of the water surface. To achieve this, we have to define a matrix $P$ that takes world coordinates and maps to image coordinates (see Equation (2.1)). This matrix allows us to obtain the direction from each point on the water surface to the camera. The steps of this process can be seen on Figure 3.1. At this moment, the water plane is considered to have no variation in height (it is the mean water plane). In our experiments, the matrix $K$ with the intrinsic parameters of the camera is built as:

$$K = \begin{bmatrix} fm_x & 0 & u_0 \\ 0 & -fm_y & v_0 \\ 0 & 0 & 1 \end{bmatrix},$$

(3.1)

where $f$ is the focal length (in centimeters), $m_x$ and $m_y$ are the ratio between the sensor...
and the image measurements (width and height, in centimeters for the sensor and in pixels for the image), $u_0$ and $v_0$ are the center of the image, in pixels. Comparing to the matrix on Equation (2.2), we assume the skew of the pixel as 0, as the pixels of our camera are square.

The matrix $K$ takes the coordinates from the camera coordinates system into the homogeneous image coordinates. In order for us to orient the camera on world coordinate system it is necessary to include the horizon line, according to the Figure 3.3. This results on the the rotation, $R$, if the line has some inclination, a rotation has to be applied, assuming that the horizon should not have any slope. We created an interactive tool where we can mark points over the horizon. An image of this process is in Figure 3.3. Applying linear least squares to fit a line through the points will return the line in the image, the normal direction of this line is the normal of the plane.

In image coordinates, we can consider the horizon as a line. Consequently, this line will have the coefficients $\vec{l} = [a, b, c]^T$. This vector multiplied by the transpose of $K$ ($\vec{n} = K^T \vec{l}$), will be in the camera coordinates. We will call it $\vec{n} = [n_x, n_y, n_z]^T$ since it represents the normal of the mean water plane. We can craft a rotation matrix by just getting three perpendicular vectors. This transformation that rotates the axis to the camera ones (Section (2.1)), $R$, is defined by

$$R = \left[ \begin{array}{c} \vec{n} \\ [1,0,0] \times \vec{n} \\ \vec{n} \times ([1,0,0] \times \vec{n}) \end{array} \right] ,$$

where $\times$ denotes the cross-product.
According to Section 2.1, \( P = KR[I - C] \), where \([I - C]\) is an identity matrix concatenated with \(-C\). The matrix \( P \) defines a relation between world coordinates and image coordinates. Figure 3.2 shows the projection of an area of the water surface, considering the distance in centimeters. In addition, we can retro-project a point on the image plane. This consists of defining the line that has all the points in the world that would project into \( G \). This line is a parametric function given by:

\[
W(\lambda) = P^T(PP^T)^{-1}G + \lambda C,
\]

where \( W(\lambda) \) is the parametric function that returns the position (a point \( W \)) on the line according to the parameter \( \lambda \in \mathbb{R} \), \( W \) is the point coordinates in the world, \( C \) is the center of the camera and \( P \) defined by Equation (2.1). If we intersect this line with the mean water plane, \( \lambda \) can be fixed because we will have the exact distance from the camera.

In our case, the horizon line rotation and the height of the camera provide information to reconstruct the water plane. Substituting the coordinates obtained by the parametric Equation (3.3), in the plane equation defined by the normal of the water plane,
we obtain an equation that will depend only on one parameter, $\lambda$. This equation is:

$$\begin{bmatrix} n_x, n_y, n_z \end{bmatrix} \begin{bmatrix} W(\lambda)_x \\ W(\lambda)_y \\ W(\lambda)_z \end{bmatrix} = 0, \quad (3.4)$$

where $[n_x, n_y, n_z]^T$ is the vector normal to the water plane and $[W(\lambda)_x,W(\lambda)_y,W(\lambda)_z]^T$ is a point in the line defined at the parametric equation (3.3). After solving this equation for $\lambda$, we get lambda and the point on the water surface. The formula to define a point becomes

$$W = (-C_z/W(\lambda)_z)W(\lambda) + C, \quad (3.5)$$

where $W(\lambda)$ is the parametric function with $\lambda$ defined to generate a normalized vector.

As a result, given an area of the water plane on the image we are able to geometrically reconstruct it. We ask the user to select a point, $G_1$, on the original image and a size of a rectangle (height and width in world unit length, e.g. centimeters). This point is retro-projected from the image to the world which results in the point $W_1$. The other three vertexes of the rectangle are obtained by dislocating the first point in the world axis, according to the predefined size, calculating $W_2, W_3$ and $W_4$. These other three point are projected back in the image resulting in $G_2, G_3$ and $G_4$. Then, we create a homography
that maps from the vertexes of the rectangle on the original image to the rectangle on the world coordinate that takes the points over the mean water plane and rectifies them,

\[
\begin{bmatrix}
  x_{g1} & y_{g1} & 1 & 0 & 0 & -x_{g1}x_{w1} & -y_{g1}x_{w1} & -x_{w1} \\
  0 & 0 & 0 & x_{g1} & y_{g1} & 1 & -x_{g1}y_{w1} & y_{g1}y_{w1} & -y_{w1} \\
  x_{g2} & y_{g2} & 1 & 0 & 0 & 0 & -x_{g2}x_{w2} & -y_{g2}x_{w2} & -x_{w2} \\
  0 & 0 & 0 & x_{g2} & y_{g2} & 1 & -x_{g2}y_{w2} & y_{g2}y_{w2} & -y_{w2} \\
  x_{g3} & y_{g3} & 1 & 0 & 0 & 0 & -x_{g3}x_{w3} & -y_{g3}x_{w3} & -x_{w3} \\
  0 & 0 & 0 & x_{g3} & y_{g3} & 1 & -x_{g3}y_{w3} & y_{g3}y_{w3} & -y_{w3} \\
  x_{g4} & y_{g4} & 1 & 0 & 0 & 0 & -x_{g4}x_{w4} & -y_{g4}x_{w4} & -x_{w4} \\
  0 & 0 & 0 & x_{g4} & y_{g4} & 1 & -x_{g4}y_{w4} & y_{g4}y_{w4} & -y_{w4}
\end{bmatrix}
\begin{bmatrix}
  h_1 \\
  h_2 \\
  h_3 \\
  h_4 \\
  h_5 \\
  h_6 \\
  h_7 \\
  h_8 \\
  h_9
\end{bmatrix} = 0,
\]

where \(x_{g/w_i}\) and \(y_{g/w_i}\), are the components of each point. The matrix formed by \(h_i\) in Equation (3.6) is the homography matrix. It maps image to world coordinates as

\[
\begin{bmatrix}
  x_w \\
  y_w \\
  z_w
\end{bmatrix} =
\begin{bmatrix}
  h_1 & h_2 & h_3 & x_g \\
  h_4 & h_5 & h_6 & y_g \\
  h_7 & h_8 & h_9 & 1
\end{bmatrix}
\]

where \([x_w,y_w,z_w]\) are the coordinates in the world and \([x_g,y_g]\) in the image. We will call the rectified image of the region from the water plan as warped image, \(W\).

In addition to the warped image, we calculate the view direction in the world, \(\vec{v}\), for every point in the warped image. This is possible by subtracting the camera coordinates, \(C\), from points coordinates in the world system of coordinates. This entire process is done only one time because for every image we are always selecting the same region to be warped and the camera is fixed.

The next steps use the warped image. Figure 3.5 shows the flowchart of the process to estimate the normal vectors of the water surface, the green step is responsible for estimating the discrete height map. In the warped image, each pixel corresponds to a centimeter square. We decided to get the normal vector by approximating the geometry of the water with an adaptation of Shape from Shading (Section 2.2), in a similar fashion that was employed by Li et al. [3]. The use of this technique requires considering the water as a Lambertian surface and a unique pointwise light source for the scene. As depicted in Figure 3.6, the water reflects the sky depending on the incident light direction. Hence,
Figure 3.4: The first image is the original picture with the vanishing line and selected region that will be rectified in orange. In this case, the original image has 4000 × 6000 pixels. The second image maps each square centimeter of the selected region to a pixel. In this example, a 2700 × 4000 centimeters area in the world becomes an image of the same size in pixels.

the water is not a Lambertian surface. In addition, even if we ignore other sources of light inside the scene because it is an outdoor one, we can consider the existence of diffuse light coming from almost all directions.

To solve this issue, we take concepts from the reflectance law [9]. Given a partially reflective surface, part of the incident light will be reflected and part will be refracted by a semi-transparent material. To simplify, we only assume that the water will be reflecting a single point in the sky. The complete reflectance is the integral of all the lights that could reach that point. The result is a simplified Fresnel equation [9]

\[ c_{Fresnel} = c_{refract}(1 - (1 - \vec{v}.\vec{d})^k) + c_{reflect}(1 - \vec{v}.\vec{d})^k, \]  

(3.8)

where \( \vec{v} \) is the viewer direction, \( \vec{d} \) is the estimated normal vector of the surface and \( k \) is called Fresnel power (typically 5). It is worth pointing that \( \vec{n} \) corresponds to the normal vector of the mean water plane, on the other hand, \( \vec{d} \) is the normal vector of the waves on the water. \( c_{refract} \) is the color when the light is completely refracted and \( c_{reflect} \) when it is totally reflected. For a gray scale image, the \( c_{refract} \) is considered to be the minimum intensity in the image, \( R_{min} \), in a scenario that the Guanabara bay absorbs almost all light. For \( c_{reflect} \), we consider that the maximum intensity is the maximum value in the image, \( R_{max} \). We consider \( k = 5 \), as it is typically used in the Fresnel equation. The
Figure 3.5: Flow diagram with the steps used to reconstruct the water surface height. So, given an image, we already have calculated that transformation and the viewer direction of the area on the water surface. With this information we can iterate our Shape from Shading approach.

Figure 3.6: Comparison showing the effect of the sky color on the water.

result is the reflectance taking the form:

\[ R = R_{max}(1 - \vec{v}.\vec{d})^5 + R_{min}(1 - (1 - \vec{v}.\vec{d})^5). \]  

(3.9)

In our implementation of Shape from Shading we use the equation above as the reflectance model, instead of \( R = \vec{i}.\vec{d} \), where \( \vec{i} \) is the illumination direction and \( \vec{d} \) is the normal vector. The 3D reconstruction allows us to get the viewer direction, as we have the camera position and the point coordinates. The other difference to [3] is the derivative defined in Equation (2.5) which becomes

\[ \frac{df(Z^{n-1}(x,y))}{dZ(x,y)} = (R_{max}(1 - \vec{v}.\vec{d})^4 + R_{min}(1 - (1 - \vec{v}.\vec{d})^4)) \frac{df(Z^{n-1}(x,y))}{dZ(x,y)} . \]  

(3.10)

This is possible because the classic approach assumes the reflectance as the Equation (2.7). So basically, it is a dot product. Deriving \( f(2.4) \) by \( Z^{n-1} \), as it is defined in the classic approach, is deriving the negative reflectance because the input image is constant. Therefore, when we derive \( f(2.4) \) with Equation (3.9) as the reflectance model
and applying the chain rule we obtain
\[
\frac{df(Z^{n-1}(x,y))}{dZ(x,y)} = (\max(1 - \vec{v} \cdot \vec{d})^4 + \min(1 - (1 - \vec{v} \cdot \vec{d})^4)) \frac{dg(\vec{v}, \vec{d})}{dZ(x,y)}, \quad \text{where } g(\vec{v}, \vec{d}) = -\vec{v} \cdot \vec{d}.
\] (3.11)

As in the case of the classic approach the derivative is a dot product, but instead of \(\vec{i}\), for the illumination direction, we have \(\vec{v}\) for the viewer direction. Considering this similarity, we use the Equation 2.5 with \(\vec{v}\) substituting \(\vec{i}\), to calculate \(\frac{dg(\vec{v}, \vec{d})}{dZ(x,y)}\). A detail in this case is that both \(\vec{d}\) and \(\frac{dg(\vec{v}, \vec{d})}{dZ(x,y)}\) are calculated using the normalized \([\frac{\partial Z^{n-1}(x,y)}{\partial x}, \frac{\partial Z^{n-1}(x,y)}{\partial y}, 1]\).

The final equation to calculate the estimated is:
\[
Z^n(x,y) = Z^{n-1}(x,y) - W - R \frac{df(Z^{n-1}(x,y))}{dZ(x,y)},
\] (3.12)
where the derivative Equation (??) was defined in the previous paragraph and \(W - R\) is the difference between the warped image and the calculated reflectance. The previous equation is part of an iterative process, as it happens on [1].

After the iterations, \(Z(x,y)\) will have the discrete height map. The components of the normal vector, called here as \([d_x, d_y, d_z]\), are calculated according to \(Z\). The components \(d_x\) and \(d_y\) are calculated using the Sobel filter. We use this filter because it mitigates the effects of the discretization as the filter considers the adjacent regions, a \(3 \times 3\) region centralized on the pixel. The component \(d_z\) is assumed to be 1. Then, we normalize the normal vector as:
\[
[d_x, d_y, d_z] = \frac{[d_x, d_y, 1]}{\sqrt{d_x^2 + d_y^2 + 1}}.
\] (3.13)

### 3.1 Results on Water Surface Reconstruction

In this section, the results from the geometric reconstruction of the water surface will be presented. We could not define a ground truth for the reconstruction, as it is impossible for us to acquire the true normal vectors of the water without a scanner. An option would be using a tool to create a simulation of the water. However, time constraints and the skill necessary to create a similar scene to what this work is dealing with made us avoid this approach.

We propose a qualitative analysis that compares the similarity between the image that we use as input and the output of the algorithm. We show the normal vector direction
and how it aligns to our assumptions previously mentioned in this chapter. As examples, here we present four images, in different situations in terms of the amount of movement of the water. In Figures 3.7, 3.9, 3.11 and 3.13, we tried to select four different situations regarding normal, high, short and noisy waves, respectively.

The original image has 4000 × 6000 pixels. The projection considered an area of 2700 × 4000 cm² and the height of the camera was 2390 cm above the sea level. Then we resize the image by a factor of \( \frac{1}{3} \). The result of this geometric transformation is in each letter (c) of Figures 3.7, 3.9, 3.11 and 3.13. The reconstruction processes have two thresholds to stop the iterations. The first is 12 iterations and the second is a maximum height above 1000 (on the whole surface). The second usually limited the process in five iterations. A Gaussian filter with variance of 0.7 is applied over the image before the algorithm to smooth the surface, minimizing the effects of high-frequency oscillations. Additionally, after the reconstruction a median filter, with the window size of 21, removes outliers that may appear. The algorithm tends to create small bumps with significant height. The median filter removes this problem as it does with salt-and-pepper noise. Considering the axis \( x \), \( y \) and \( z \) where the \( z \) represents the height, the projection on the plane \( xy \) are the second image on the second column of the Figures 3.7, 3.9, 3.11 and 3.13. The reconstruction was plotted with gray-scale colors and a pointwise illumination positioned on the infinite, trying to emulate the lighting effect on the original image. As we are not reconstructing with the water BRDF function, texture and other details, the appearance is more similar to plastic or metal.

The results in Figures 3.7, 3.9, 3.11 and 3.13, letters (b) and (d), are similar to the input images. However, when the heights are considered, they are still not similar to the real ones. Typically, the maximum height after the reconstruction does not reach one. This is clearly not the correct height. In fact, the algorithm does not converge to a specific height. In order to deal with this issue, we decided to scale the height as they stay in a specific range to a more suitable one. The results displayed in this section are multiplied by 10.

Figures 3.8, 3.10, 3.12 and 3.14 present the vectors components as colors. Thus, the \( x \), \( y \) and \( z \) components of the derivative will be mapped respectively to the red, green and blue color components of the RGB image. The normal vectors follow the assumption that the components directed to the camera will be darker, both in the color image as
Figure 3.7: An average wave image is captured. Image (a) presents the selected area that will be rectified, while (b) shows the reconstructed surface in a similar point of view of (a). The rectified image is presented at (c) and the reconstructed surface at (d). The green rectangles correspond to regions where the reconstruction was satisfactory comparing (a) to (b) and (c) to (d). The regions highlighted by the red frames depicts regions where the reconstruction was not satisfactory.

well as in the normal vector image, as they will not be reflecting the sky but refracting the light.

Although the reconstruction can be similar, it is not entirely correct. Some details, as oscillations similar to wrinkles, are not reconstructed. Instead, they tend to disappear. Spume also becomes “false waves” and gain a height that does not match the reality. Figure 3.13 depicts this situation as the blob on the lower right corner becomes a bump. That structure is not observed in the original image.
Figure 3.8: The normal vectors from Figure 3.7d, where $||\vec{d}|| = 1$ and $\frac{[d_x, d_y, d_z]^T + 1}{2}$ corresponds to red, green and blue, respectively, in the colored image.

Figure 3.9: An high wave image is captured, (a) presents the selected area that will be rectified, (b) has the reconstructed surface in a similar point of view of (a). The rectified image is presented at (c) and the reconstructed surface at (d). The green rectangles correspond to regions where the reconstruction was satisfactory comparing (a) to (b) and (c) to (d). The red are the opposite, regions where the reconstruction does not approximately match the reality.
Figure 3.10: The normal vectors from Figure 3.9d, where $\|\vec{d}\| = 1$ and $\begin{bmatrix} d_x, d_y, d_z \end{bmatrix}^T + \frac{1}{2}$ corresponds to red, green and blue, respectively, in the colored image.

![Figure 3.10](image)

Figure 3.11: An short wave image is captured, (a) presents the selected area that will be rectified, (b) has the reconstructed surface in a similar point of view of (a). The rectified image is presented at (c) and the reconstructed surface at (d). The green rectangles correspond to regions where the reconstruction was satisfactory comparing (a) to (b) and (c) to (d). The red are the opposite, regions where the reconstruction does not approximately match the reality.

![Figure 3.11](image)
Figure 3.12: The normal vectors from Figure 3.11d, where $\|\vec{d}\| = 1$ and $\frac{[d_x, d_y, d_z]^T + 1}{2}$ corresponds to red, green and blue, respectively, in the colored image.

Figure 3.13: An image with noise, probably foam, is captured, (a) presents the selected area that will be rectified, (b) has the reconstructed surface in a similar point of view of (a). The rectified image is presented at (c) and the reconstructed surface at (d). The green rectangles correspond to regions where the reconstruction was satisfactory comparing (a) to (b) and (c) to (d). The red are the opposite, regions where the reconstruction does not approximately match the reality.
Figure 3.14: The normal vectors from Figure 3.13d, where $\|\vec{d}\| = 1$ and $
abla [d_x, d_y, d_z]^T + 1$
2
corresponds to red, green and blue, respectively, in the colored image.
Chapter 4

Random Number Generation

The generation of random numbers depends directly on the normal vectors. However, the technique used to acquire the normal vectors is ignored by this part. Figure 4.1 presents a flow diagram with each step of the generation.

With the reconstructed geometry (with a normal vector associated with each pixel) for an image, we sample the normal vectors map using a predefined set of points, $L$. Actually, they are pairs of points that will give two normal vectors, so

$$L = \{(i,j) | (i,j) \text{ are indexes of normal vectors}\}. \quad (4.1)$$

The angle between each pair is calculated using Equation (4.2). However, any sampling pattern in this part could result in some correlation between the location and the angle. To avoid this, we use a weaker random number generator (one that fails in the tests defined at Section 4.1). The angles are calculated accordingly to

$$\theta = \cos^{-1}(\vec{d}(i) \cdot \vec{d}(j)), \quad (4.2)$$

where $i, j$ are indexes from the weak set of random numbers $L$, defined at Equation (4.1).

The final step consists on turning the angles into binary numbers. This turns out to be critical, as we have to force the realization of a Bernoulli variable, which takes 1 with a certain probability and zero with another. For a uniform distribution this probability is equal to 0.5. We do this by setting a threshold as the median of the set of angles (that are different for each image),

$$b = \begin{cases} 
1, & \text{if } \theta > \tilde{\theta} \\
0, & \text{otherwise}
\end{cases}, \quad (4.3)$$
where $b$ is the bit, $\tilde{\theta}$ is the median and $\theta$ the angle. After enforcing this, the probability mass function becomes

$$p_b(b) = \begin{cases} 
\frac{1}{2}, & \text{if } b = 1 \\
\frac{1}{2}, & \text{if } b = 0 \\
0, & \text{otherwise}
\end{cases} $$

(4.4)

where $p_b$ is the probability mass function over the random variable $b$, that is the bit, generated on (4.3).

Although the pairs used to sample the normal vectors come from a weak random generator (Equation 4.1), using them for every image still caused bias on the generator. To avoid this, each image is sampled by a constant set of pairs combined with the random numbers generated by the previous image. This combination is basically summing and making sure that the index is a valid one (taking the module by the maximum index):

$$L = ([i + r_{\max \#d}], [j + t_{\max \#d}]),$$

(4.5)

where $r, t$ are random numbers generated by the previous image and $\max \#d$ is the maximum number of random values generated by one image.

### 4.1 Methodology for Testing Random Number Generation

The validation is directly related to the final result, so, to the generated random numbers. Consequently, testing them define how good they are randomly. In terms of
tests, we used two sets. The first one is a set that we will call as mathematical tests. It tries to identify less subtle deviations on the generation with statistical and geometric tests. The second set is composed of just statistical tests defined by the National Institute of Standards and Technology, from the USA [10]. However, first it is necessary to define some concepts from statistics that will be used in the tests and what a random number is. In addition, how these tests are able to assure these feature.

A random phenomenon is one that it is impossible to predict the result, regardless of the total number of possible results. Thus, considering a computer, that only deals with binary numbers, it is an event with a given probability of generating an 1 or 0. In a statistical point of view, the uniform distribution is expected when the amount of generated numbers tends to increase. Another characteristic is unpredictability, i.e. the next output of the generator should be independent of any previous results.

As randomness is a statistical property. Different statistical tests can be used to define patterns on a sequence. Actually, the tests try to prove a null hypothesis, that would be, the sequence is made of random numbers. It is possible that an infinity of tests exists, as there might exist a distribution that a generator will start to fit in at some point. As a result of this, no set of tests is considered complete.

In spite of this, our set was defined to join a group comprised of basic tests and a group of more complex tests. The former can be considered shallower, but they can quickly exclude a non-random number. The later looks for a more official and standardized assurance. So it employs more detailed and descriptive tests, besides coming from a recognized institution (National Institute of Standards and Technology).

4.1.1 Mathematical Tests

The first set of tests includes five mathematical tests and the source code is available in the page of Fourmilab [11]. This test was used for another well known random number generator that comes from atomic decay, Hotbit [5].

**Entropy** - Given an random variable $f$ on the finite alphabet denoted by $A = \{a_1,a_2,\ldots,a_{|A|}\}$.

The entropy, $H$, is defined over the discrete probability distribution of the elements from this alphabet.

$$H(f) = \sum_{a \in A} p(f = a)\log(1/p(f = a))$$
In this case, this function has \( \log \) in base 2 as we are dealing with binary information and we will be counting the entropy in a byte. So the maximum entropy will be eight because of the 256 possible combinations of bits.

**Chi-square Test** - This test is the same presented in Section 2.3.2. It considers 256 elements in the alphabet, so the compared distribution is uniform, having one of probability \( \frac{1}{256} \) for each possible outcome. As a result, the degree of freedom is 255 and the level of confidence is 1%. It also establishes more limits for the percentage, if it is between 1% and 5% it is considered suspect and up to 10% it considers it “almost suspect”.

**Arithmetic Mean** - This is less a test and more a useful information to indicate some closeness to randomness. For an infinity amount of numbers generated randomly, their mean value should get closer to 127.5. The test calculates the mean and compares it to 127.5.

**Monte Carlo Value of \( \pi \)** - The test uses the fact that the ratio between the area of a circle inscribed inside a square and the area of the square is \( \frac{\pi}{4} \). The area of the square is \((2r)^2\) and the circle is \(\pi r^2\). Consequently, for \(N\) selected points, \(N\pi/4\) of them will fall inside the circle.

The test consists of selecting groups of 6 bytes from the data to be analyzed. The first three corresponds to \(X\) and the last three to \(Y\) coordinate. If the points fall inside the circle then \(X^2 + Y^2 < r^2\) and it is considered a hit. We can approximate \(\pi\) as \(\frac{4\#\text{Hits}}{N}\). The test points the error between the calculated and the value of \(\pi\) defined as a constant.

**Serial Correlation Coefficient** - Given two sets of quantities \((U,V)\) the correlation coefficient between them has the following formula [12]:

\[
C = \frac{n \sum_{i=1}^{n} (U_i V_i) - \sum_{i=1}^{n} (U_i) \sum_{i=1}^{n} (V_i)}{\sqrt{(n \sum_{i=1}^{n} U_i^2 - \sum_{i=1}^{n} U_i^2)(n \sum_{i=1}^{n} V_i^2 - \sum_{i=1}^{n} V_i^2)}}.
\]

The correlation lies between \([-1,1]\] and the sets are totally uncorrelated for 0 and linearly dependent in the limits. In the case of this test, the idea is to compare the element with the next one in the sequence of random numbers generated. The first set will have numbers from \([x_0,x_{n-1}]\) and the second \([x_1,x_n]\).
4.1.2 NIST Tests

The second set of tests has a very detailed description in the report [10]. However, here the tests will be described to the point where it is possible to implement them. For this purpose, some formulas and explanations were omitted.

**Frequency Test** - This test focuses on the proportion of zeros and ones. In a random sequence, they should happen with similar frequency. Considering the sequence of bits $\varepsilon = \varepsilon_1, \varepsilon_2, ..., \varepsilon_n$, the test applies the transformation $X_i = 2\varepsilon_i - 1$ to the sequence. After this, all the elements are summed, $S_n = \sum_{i=1}^{n} X_i$. The next step is based on the De Moivre-Laplace theorem, for a large number of trials, the probability of the binomial sum, normalized by $\sqrt{n}$, named here as $S_n$, approximates to the cumulative normal distribution (defined here as $\phi$).

For an normal distribution with $\sigma^2 = 1$, where $\sigma^2$ is the variance. The complementary error function ($erfc(\frac{X}{\sqrt{2}})$) gives the probability that a measurement $X$ with normally distributed errors has a distance less than $X$ from the mean value $2\phi(z) - 1$. So, if $X$ is $S_n$, the probability will indicate how likely that value can happen with the mean equal to 0. The complementary function is defined as.

$$erfc(z) = \frac{2}{\sqrt{\pi}} \int_{z}^{\infty} e^{-u^2} du$$

In this test, if the probability is too small, i.e., less than 0.01, it means that $S_n$ is too large. In such a case, the sequence is considered not random.

**Frequency Test with a Block** - This test tries to identify parts of the sequence that deviate from the ideal proportion of 50% of ones and zeros. The data is split in $N$ non-overlapping sequences, each with $M$ elements. For each separate sequence, the number of ones is calculated and the distribution is compared to the ideal of 0.5 using the Chi-square test. The Chi-square statistic assumes the formula,

$$X^2 = 4M \sum_{i=1}^{N} [\pi_i - \frac{1}{2}]^2.$$ 

where $\pi$ is the observed frequency in each sequence of size $M$. With the statistic we can apply the Chi-square test (section 2.3.2). The number of subsequences defines the degree of freedom.
The probability from the incomplete gamma function greater than 0.01 indicates that the null hypothesis is not false (i.e., the sequence is random).

**Run Test** - The term “Run” here is defined as an uninterrupted sequence of the same value, i.e., zeros or ones. This test tries to define if the switching between the two values is satisfactory. Initially, the proportion of ones is defined as \( \pi = \frac{\sum_j \varepsilon_j}{n} \), where \( n \) is the total number of elements and \( \varepsilon_j \) is an element of the sequence to test.

The function to consider is

\[
erfc\left(\frac{|V_n(\text{obs}) - 2n\pi(1 - \pi)|}{2\sqrt{2n\pi(1 - \pi)}}\right).
\]

\( V_n \) is defined as

\[
V_n = \sum_{k=1}^{n-1} r(k) + 1 \begin{cases} 
  r(k) = 0, & \text{if } \varepsilon_k = \varepsilon_{k+1} \\
  r(k) = 1, & \text{otherwise}
\end{cases}
\]

For the limit where \( n \to \infty \) the function \( \frac{|V_n(\text{obs}) - 2n\pi(1 - \pi)|}{2\sqrt{2n\pi(1 - \pi)}} \) goes to the cumulative standard normal distribution.

The decision rule is applied to the complementary error function \( (erfc) \) at 1%. For an error less than 1% the test fails. \( V_n \) increases with the variation of bits. So extreme values of the function are penalized. For more information, see Section 3.3 of the report by NIST [10].

**Longest Run of Ones in a Block** - This test consists in breaking the sequence, \( \varepsilon \), of \( n \) bits in \( N \) blocks of size \( M \). The length of the block can assume three different values (8, 128, 10000) depending on the sequence.

The idea is to compare the frequency of the longest run within a sequence against the expected one. The Chi-square test is applied in this case. Initially, the frequencies are tabulated in the following table.
Table 4.1: Different cells according to the size of each block. For every cell an specific frequency or a range (in the extremes).

<table>
<thead>
<tr>
<th>$v_i$</th>
<th>M=8</th>
<th>M=128</th>
<th>M=10000</th>
</tr>
</thead>
<tbody>
<tr>
<td>$v_0$</td>
<td>≤1</td>
<td>≤4</td>
<td>≤10</td>
</tr>
<tr>
<td>$v_1$</td>
<td>2</td>
<td>5</td>
<td>11</td>
</tr>
<tr>
<td>$v_2$</td>
<td>3</td>
<td>6</td>
<td>12</td>
</tr>
<tr>
<td>$v_3$</td>
<td>≥4</td>
<td>7</td>
<td>13</td>
</tr>
<tr>
<td>$v_4$</td>
<td></td>
<td>8</td>
<td>14</td>
</tr>
<tr>
<td>$v_5$</td>
<td>≥9</td>
<td></td>
<td>15</td>
</tr>
<tr>
<td>$v_6$</td>
<td></td>
<td></td>
<td>≥16</td>
</tr>
</tbody>
</table>

The statistical measurement to test is

$$\chi^2(\text{obs}) = \sum_{i=0}^{K}(v_i - N\pi_i)^2 \div N\pi_i.$$

The principle is to compare the frequency against the expected concentration of ones within the block. So, $\pi_i$ is the same $\pi_{\square}$ defined for the run test, but inside a block. A large value defines a nonrandom sequence (with the presence of clusters).

Finally, the incomplete gamma function yields the tested value. So, for the Chi-square function (as explained in the second test) the degree of freedom is $K/2$ and $x = \chi^2/2$. A value less than 0.01 indicates a random sequence. More details about this test is presented in Section 3.4 of [10].

**Binary Matrix Rank Test** - This test focus on finding out linear dependencies among subsets of bits with length $n$. The sequence $\varepsilon$ is broken in $O$ disjoint matrices of $P$ rows and $Q$ columns, both sets with size 32.

The main idea is to apply the Chi-square test comparing the observed value with the expected number of ranks, assuming randomness in the original sequence. Each matrix will have its binary rank defined. According to it, they will be grouped into three categories: $F_M$ for full rank matrices, $F_{M-1}$ for matrices of full rank -1 and $N - F_M - F_{M-1}$ for the remaining cases.

The test statistic is defined as
\[ \chi^2_{(obs)} = \frac{(F_M - 0.2888N)^2}{0.2888N} + \frac{(F_{M-1} - 0.5776N)^2}{0.5776N} + \frac{(N - F_M - F_{M-1} - 0.1336N)^2}{0.1336N}. \]

The definition of these constants multiplying the number of matrices can be found in Section 3.5 of the report by NIST [10].

Finally, the critical value is calculated using the incomplete gamma function with \(N/2\) degrees of freedom and Chi-square value divided by 2. Results of less than 0.01 indicate that the null hypothesis is false, i.e. the sequence is not random.

**Fourier Transform Test** - This test focuses on the peaks of the Fast Fourier Transform of the sequence. Periodic features exceeding 95% of the maximum value should not surpass significantly 5% of the sequence frequencies. Initially, a sequence of bits, \( \varepsilon \), is transformed into the domain \(+1, -1\) by the function \(2\varepsilon - 1\), resulting in the sequence \(\varepsilon\Box\). The Discrete Fourier Transform will produce complex variables representing the periodic components of the frequency. The peaks of the \(n/2\) elements of the sequence \(\varepsilon\Box\) are the modules of the complex variables.

The 95% threshold is given by \(\tau = \sqrt{(\log \frac{1}{0.05})n}\). In a random sequence \(N_0 = 0.95n/2\) of the elements shout not exceed \(\tau\). This value is compared to the total number of peaks less than \(\tau\), named as \(N_1\).

The complementary error function of \(d = \frac{N_1 - N_0}{\sqrt{n 0.95 \, 0.05^2}}\) over \(\sqrt{2}\), returns the probability of \(d \in [-d, d]\). If it is less than 0.01 then the sequence is considered not random, as the mean value would be 0. The strategy is similar to the one used on the frequency test. Please, refer to Section 3.6 of [10] for details.

**Non-overlapping Template Test** - Given a set of templates with size \(m\), the sequence of bits \(\varepsilon\) of size \(n\) is tested. The main idea is to compare the observed number of matches with the expected one in a random sequence. To perform this test the entire sequence is broken in \(N\) strings of \(M\) bits each.

A template \(B\), with size \(m\), is compared against a window in the block. If there is a hit then the window slides to the next bit that was not matched. On the other hand, it just advances a single bit. Assuming randomness the mean and variance of the hits are, respectively,
\[ \mu = \frac{(M - m + 1)}{2^m}, \quad \sigma^2 = M\left(\frac{1}{2^m} - \frac{2m - 1}{2^{2m}}\right). \]

In this work, there will not be any further explanation about this values. More information can be found in Section 3.7 of [10]. The statistic test is the Chi-square test. Defining the number of hits as \( W \),

\[ \chi^2(\text{obs}) = \sum_{j=1}^{N} \frac{(W_j - \mu)^2}{\delta^2}. \]

As it happened for the previous tests, the critical value is defining by calculating the incomplete gamma function of half the number of blocks and the statistic value. For a value less than 0.01 the sequence is not random.

**Overlapping Template Test -** This test, like the previous one, tries to detect sequences of templates in the string \( \varepsilon \) of size \( n \). The difference is the treatment of the hit. Here, it advances one bit each time instead of jumping to the one right after the template.

For this test, the entire sequence is again broken into \( N \) non-overlapping sequences of size \( M \). For each sequence, given a template \( B \), a window (with size \( m \)) slides and the number of hits is counted. Five counters are kept, \( v_0, v_1, v_2, v_3, v_4, v_5 \), they keep track of the number of matches of each template in each sequence. For example, if in the sub string 1 the template 011 happens three times, then \( v_3 \) will be incremented by one.

These counters are compared against the theoretical expected value for an random sequence. The Chi-square test is again used. However, here expected values will be assumed as constants,

\[ \chi^2(\text{obs}) = \sum_{i=0}^{5} \frac{(v_i - N\pi_i)^2}{n\pi_i}, \quad \pi_0 = 0.364091, \pi_1 = 0.185659, \pi_2 = 0.139381, \pi_3 = 0.100571, \pi_4 = 0.070432, \pi_5 = 0.139865. \]

The critical value is calculated with the incomplete gamma function as in previous tests. So, a large statistic value results in a small critical value. The threshold is again 0.01. The definition of them and more details about this test can be found in Section 3.8 of [10].
Maurer’s “Universal Statistical” Test - This test focuses on how much the sequence can be compressed. According to its author, the test is designed “to be able to detect one of the very general classes of statistical defects that can be modeled by an ergodic stationary source with finite memory”.

The test splits the sequence $\varepsilon$, of $n$ bits, into two sets of non-overlapping blocks. The first one has size $Q$ bits and is an initialization sequence. The other one, with size $K$ bits, is the one to be analyzed. Each block has size $M$.

With the initialization sequence an table is created, it relates the last occurrence of each sub sequence of size $M$.

The test goes through all blocks in the testing sequence and replaces the last occurrence of each possible block. In addition, a variable accumulates the $\log_2$ distances to the last occurrence of the current block. The statistical test is then defined by this sum divided by $K$ (the total number of blocks that are tested).

The critical value is given by calculating the complementary error function of

$$\rho = \text{erf} \left( \frac{|ln-\text{expectedValue}(M)|}{\sqrt{2\delta}} \right),$$

where the critical value. In this work, the explanation of this formula as the ones for the expected value (table 4.2) and variance ($\sigma^2 = \sqrt{\text{variance}(M)}$), $c = 0.7 - \frac{0.8}{M} + (4 + 32 \frac{M}{K} - 3M^{15})$ will be omitted. More information can be found in the section 3.9 of [10].
Table 4.2: Expected values and variances for every possible length of $L$

<table>
<thead>
<tr>
<th>$L$</th>
<th>expectedValue</th>
<th>variance</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>5.2177052</td>
<td>2.954</td>
</tr>
<tr>
<td>7</td>
<td>6.1962507</td>
<td>3.125</td>
</tr>
<tr>
<td>8</td>
<td>7.1836656</td>
<td>3.238</td>
</tr>
<tr>
<td>9</td>
<td>8.1764248</td>
<td>3.311</td>
</tr>
<tr>
<td>10</td>
<td>9.1723243</td>
<td>3.356</td>
</tr>
<tr>
<td>11</td>
<td>10.170032</td>
<td>3.384</td>
</tr>
<tr>
<td>12</td>
<td>11.168765</td>
<td>3.401</td>
</tr>
<tr>
<td>13</td>
<td>12.168070</td>
<td>3.410</td>
</tr>
<tr>
<td>14</td>
<td>13.167693</td>
<td>3.416</td>
</tr>
<tr>
<td>15</td>
<td>14.167488</td>
<td>3.419</td>
</tr>
<tr>
<td>16</td>
<td>15.167379</td>
<td>3.421</td>
</tr>
</tbody>
</table>

In this test, if the critical values differ significantly from the expected values, then the sequence is random if the critical value is greater than 0.01.

**Linear Complexity Test** -  Linear feedback shift register is a type of electronic circuit known for producing pseudo-random sequences. However, the register generates a repetition of a sequence of bits, so the longer the sequence the more similar to a random generator.

Given a sequence $\varepsilon$, of length $n$, $N$ sub sequences of $M$ bits are created. For each block, the Berlekamp-Massey algorithm calculates the shortest linear feedback shift register sequence that generates the bits at $M$. The idea is that by summing, in a certain manner, previous bits modulo 2 should generate the next element of the sequence.

The Chi-square measurement is given by $\sum_{i=0}^{k} \frac{(v_i - N\pi_i)^2}{N\pi_i}$, where $\pi_0 = 0.010417$, $\pi_1 = 0.03125$, $\pi_2 = 0.125$, $\pi_3 = 0.5$, $\pi_4 = 0.25$, $\pi_5 = 0.0625$, $\pi_6 = 0.02083$.

The variable $v_i$ is defined according to the result of the function $T_i = (-1)^M(L_i - \mu) + \frac{2}{5}$, where $\mu$ is the theoretical mean $\mu = \frac{M}{2} + \frac{9 + (-1)^{M+1}}{36} - \frac{M}{2} + \frac{2}{9}$. So, if
$T_i \leq -2.5$ Increment $v_0$ by one
$-2.5 < T_i \leq -1.5$ Increment $v_1$ by one
$-1.5 < T_i \leq -0.5$ Increment $v_2$ by one
$-0.5 < T_i \leq 0.5$ Increment $v_3$ by one
$0.5 < T_i \leq 1.5$ Increment $v_4$ by one
$1.5 < T_i \leq 2.5$ Increment $v_5$ by one
$T_i \geq 2.5$ Increment $v_6$ by one.

The critical value is the Chi-square probability with 6 degrees of freedom. A critical value lower than 0.01 would indicate a deviation from the expected in a random sequence. More details about these constants and the test, in general, can be found at Section 3.10 of [10].

**Serial Test** - The focus of the serial test is to check whether the frequency of all possible $2^m$ m-bits overlapping patterns match the expected value for a random sequence. In a random sequence, the frequencies should be uniform.

The measure that defines how well the $m$-bits frequency match the expected frequencies is defined as

$$\nabla^2 \psi^2_m = \psi^2_m - \psi^2_{m-1}, \text{ and } \nabla^2 \psi^2_m = \psi^2_m - 2\psi^2_{m-1} + \psi^2_{m-2},$$

where

$$\psi^2_m = \frac{2^m}{n} \sum_{i_1 \ldots i_m} (v_{i_1 \ldots i_m} - \frac{n}{2^m})$$

$$\psi^2_{m-1} = \frac{2^{m-1}}{n} \sum_{i_1 \ldots i_{m-1}} (v_{i_1 \ldots i_{m-1}} - \frac{n}{2^{m-1}})^2$$

$$\psi^2_{m-2} = \frac{2^{m-1}}{n} \sum_{i_1 \ldots i_{m-2}} (v_{i_1 \ldots i_{m-2}} - \frac{n}{2^{m-2}})^2,$$

where $v_{i_1 \ldots i_m}$ is the frequency of a specific bit pattern of size $m$.

The reference distribution for the test is the Chi-square. So, as it was defined in previous tests, the incomplete gamma function defines the critical value.
\( \rho_1 = i_{gamc}(2^{m-2}, \nabla \psi^2_m) \), and \( \rho_2 = i_{gamc}(2^{m-3}, \nabla^2 \psi^2_m) \),

where \( \rho \) is the critical value. Both values are compared to 0.01. If any of them is larger then it means a non-uniformity. Please refer to Section 3.11 of [10] for more details.

**Approximate Entropy Test** - This test compares the frequency of overlapping blocks of two consecutive/adjacent lengths \( (m \) and \( m+1 \)) against the expected for a random sequence. Given a sequence of length \( n \), the sequence is augmented by \( m-1 \) bits to create \( n \) overlapping sequence of \( m \) bits. Considering two sizes of sequence \( m \) and \( m+1 \). For each of them, the incidence of each possible sequence is accumulated. Considering the incidence of, for example, \( m = 3 \), there will be 8 accumulators, we will call each one \( \pi_i \).

The test statistic is computed as \( \chi^2 = 2n [\log 2 - ApEn(m)] \), where \( ApEn(m) = \varphi^{(m)} - \varphi^{(m+1)} \) and \( \varphi^{(m)} = \sum_{i=0}^{2^m-1} \pi_i \log \pi_i \).

The chi-square test applied over this statistic with \( 2^{m-1} \) degrees of freedom. This can be calculated by \( P-value = i_{gamc}(2^{m-1}, \chi^2/2) \), where \( i_{gamc} \) is the incomplete gamma function, and the \( P-value \) is the critical value that is compared to 0.01. If the variable id greater than 0.01, than it is considered random.

**Cumulative Sums Test** - The test checks if the cumulative sums of the partial sequences deviate from the expected value, 0. In a sequence, called here as \( \varepsilon \) and with size \( n \), composed by 1’s and -1’s, a random walk should generate a small cumulative sum.

Given \( \varepsilon \), the partial sums of successively larger sequences \( (S_1, S_2, S_3, \ldots) \) is computed in the forward and backward mode.

So in the forward mode

\[
S_1 = X_1 \\
S_2 = X_1 + X_2 \\
\vdots \\
S_n = X_1 + \ldots + X_n
\]

and in the backwards mode \( S_1 = X_n \)

\( S_2 = X_n + X_{n-1} \)
$S_n = X_n + \ldots + X_1$

The maximum absolute value between these two will be used to calculate the critical value. Here, we will just present the formula to calculate it. More details can be found in the Section 3.13 of the NIST report [10].

$$\rho = 1 - \sum_{(\frac{n}{2}+1)+4}^{\frac{n}{2}+1} \left[ \Phi\left( \frac{(4k+1)z}{\sqrt{n}} \right) - \Phi\left( \frac{(4k-1)z}{\sqrt{n}} \right) \right] + \sum_{(\frac{n}{2}-3)+4}^{(\frac{n}{2}-1)+4} \left[ \Phi\left( \frac{(4k+3)z}{\sqrt{n}} \right) - \Phi\left( \frac{(4k+1)z}{\sqrt{n}} \right) \right],$$

where $\Phi$ is the standard cumulative normal probability distribution function and $\rho$ is the critical value.

For critical values less than 0.01, the sequence is not random.

**Random Excursion Test Variant** Given a bit sequence $\varepsilon$, 18 possible states of a cumulative random walk are defined, $[-9, -8, \ldots, 8, 9]$. Applying the function $2\varepsilon_i - 1$ over the sequence yields a sequence of only -1 or +1, the new sequence will be called $X$. The sum, in a random walk, should be always close to 0. The cumulative sums are calculated in the following way and it always starts and ends on 0.

$$S_1 = X_1$$
$$S_2 = X_1 + X_2$$

$$\vdots$$
$$S_n = X_1 + \ldots + X_n$$

An accumulator $\xi(x)$ counts the occurrence of each possible state. For each state the complementary error function test, defined below, will indicate the probability of that sum, considering a random walk.

$$\rho = \text{erfc}\left( \frac{\left| \xi(x) - J \right|}{2J(4|x|-2)} \right),$$

where $J$ is the number of times that the sequence is equal to 0 and $\rho$ is the critical value. A large deviation from randomness is indicated by $\rho < 0.01$. 
4.2 Results on Random Number Generation

We applied the tests defined in the previous section to compare our generator with three others. The first is the generator provided by the function rand(), on Visual Studio 2015. The second one is the rand() function provided by MATLAB. The third is the rand() function provided by GCC. The last is the generator provided by RANDOM.ORG [6]. The one from Visual Studio changes from C++ compiler to another. Unfortunately, we could not define the pseudo-random generator used, the GCC uses the Linear congruential generator. We directly used the MATLAB generator. The default configuration uses the Mersenne Twister generator with seed equals to 0. It also assures a uniform distribution for the generation. The last generator is from the RANDOM.ORG [6], it uses atmospheric noise to create the bits. It is stated that the generator passed the tests defined in Section 4.1.2.

For MATLAB, GCC and C++ we simply created 10 million bits and saved then in a file. For RANDOM.ORG, the website administrators keep a set of 1MB of bits, generated every day. We combined bits from two different days to obtain the necessary 10 million. For our approach, we used a video, which resolution is $1020 \times 1920$ and frame rate of 30 frames per second. We sampled this video taking one image every 50 frames. The idea was to minimize the effects of similar images on the generation, given that the speed in which the water changes is slower than the frame water, at least visually. Each frame generates 20000 bits, therefore it was necessary 500 frames or approximately 7 minutes of video to generate 10 million bits.

We split the 10 million bits into 10 sets. The table below shows the worst result for every generator in each test from the Mathematical Test, defined on Section 4.1.1. For example, in the Arithmetic Mean row, the result more distant to 127.5 was selected. In the serial Correlation Coefficient, this value is 0. The only different is the Chi-Square Test, where we defined a threshold of 1%. If the probability tested is less than this, we consider the sequence not random. The table has the number of sequences that passed this test. The Visual Studio generator was the only one that failed this test and it was the one used to select the pairs of normal directions in our approach.

The NIST tests are defined in Section 4.1.2. We decided to count how many times each sequence passed each test. While mostly we applied one test per set of a million bits each, resulting in 10 tests. For the Cumulative Sums and Serial Test, there are actually
Table 4.3: Mathematical tests results using four different generators. Given a set of 10 million numbers, we split into 10 subsets of 1 million elements each. We selected the worst result in each test, the exception is the chi-square test. In this specific one, we counted the amount of sets that passed. The Visual C++ and the GCC generator failed a test, all the chi-square tests.

<table>
<thead>
<tr>
<th></th>
<th>Our</th>
<th>MATLAB</th>
<th>C++</th>
<th>GCC</th>
<th>RANDOM.ORG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chi-Square Test</td>
<td>10</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>Arithmetic Mean</td>
<td>126.899</td>
<td>127.076</td>
<td>126.525</td>
<td>126.775</td>
<td>127.265</td>
</tr>
<tr>
<td>Monte Carlo Value of $\pi$</td>
<td>3.169</td>
<td>3.135</td>
<td>3.178</td>
<td>3.181</td>
<td>3.134</td>
</tr>
<tr>
<td>Serial Correlation Coefficient</td>
<td>0.006</td>
<td>-0.005</td>
<td>-0.003</td>
<td>-0.004</td>
<td>-0.003</td>
</tr>
</tbody>
</table>

2 tests. So for each sub-sequence two tests are applied and we summed the times they passed. The threshold for each type of test is 8 out of 10 sequences passing the tests.

The Non-Overlapping Template consists of 148 tests, one for each template. In Table 4.4, instead of summing them all we present a number of tests that passed, considering a minimum of 8 of 10. Each template will have its own 10 different tests. If it fails more than two times, then the generator failed for that specific sequence of bits. For example, the C++ generator failed for 5 templates whereas the other generators passed them all.

The last one is the Random Excursions Variant test, because it requires a minimum value of cycles (more information on Section 4.1.2), not all sequences are suitable for it. However, the fact that the only generator not able to be tested was the same that failed other tests indicates a possible failure.
Table 4.4: It has the results of the NIST tests after 10 sets of one million bits each. Tests with * were calculated differently, Non-Overlapping Template basically applies a test for each template and Random Excursions Variant depends on the bits to apply the test (the set can allow less than 10 tests). Again the Visual C++ generator failed completely on 5 tests. In addition, the GCC failed on six tests.

<table>
<thead>
<tr>
<th>Test</th>
<th>Our</th>
<th>MATLAB</th>
<th>C++</th>
<th>GCC</th>
<th>RANDOM.ORG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>10/10</td>
<td>10/10</td>
<td>0/10</td>
<td>1/10</td>
<td>10/10</td>
</tr>
<tr>
<td>BlockFrequency</td>
<td>10/10</td>
<td>10/10</td>
<td>10/10</td>
<td>10/10</td>
<td>10/10</td>
</tr>
<tr>
<td>CumulativeSums</td>
<td>20/20</td>
<td>20/20</td>
<td>0/20</td>
<td>2/20</td>
<td>20/20</td>
</tr>
<tr>
<td>Runs</td>
<td>10/10</td>
<td>10/10</td>
<td>0/10</td>
<td>0/10</td>
<td>10/10</td>
</tr>
<tr>
<td>LongestRun</td>
<td>10/10</td>
<td>10/10</td>
<td>10/10</td>
<td>10/10</td>
<td>9/10</td>
</tr>
<tr>
<td>Rank</td>
<td>10/10</td>
<td>10/10</td>
<td>10/10</td>
<td>9/10</td>
<td>10/10</td>
</tr>
<tr>
<td>FFT</td>
<td>10/10</td>
<td>10/10</td>
<td>10/10</td>
<td>10/10</td>
<td>10/10</td>
</tr>
<tr>
<td>NonOverlappingTemplate*</td>
<td>148/148</td>
<td>148/148</td>
<td>143/148</td>
<td>143/148</td>
<td>148/148</td>
</tr>
<tr>
<td>OverlappingTemplate</td>
<td>9/10</td>
<td>10/10</td>
<td>0/10</td>
<td>10/10</td>
<td>10/10</td>
</tr>
<tr>
<td>Universal</td>
<td>10/10</td>
<td>10/10</td>
<td>10/10</td>
<td>0/10</td>
<td>10/10</td>
</tr>
<tr>
<td>ApproximateEntropy</td>
<td>10/10</td>
<td>10/10</td>
<td>0/10</td>
<td>0/10</td>
<td>10/10</td>
</tr>
<tr>
<td>RandomExcursionsVariant*</td>
<td>10/10</td>
<td>7/7</td>
<td>-/-</td>
<td>1/1</td>
<td>6/6</td>
</tr>
<tr>
<td>Serial</td>
<td>20/20</td>
<td>20/20</td>
<td>18/20</td>
<td>20/20</td>
<td>20/10</td>
</tr>
<tr>
<td>LinearComplexity</td>
<td>10/10</td>
<td>10/10</td>
<td>10/10</td>
<td>10/10</td>
<td>10/10</td>
</tr>
</tbody>
</table>
Chapter 5

Final Remarks

We developed techniques to approach the generation of the normal vectors from the water surface and the generation of the numbers from the normals. It uses geometric reconstruction and a modified version of Shape from Shading [1] to estimate the discrete height map. We then calculate the angles between vectors and make the information binary so we can get the numbers.

As stated before, even though natural events are often used as source for random number generators, the ones mostly used for TRNGs are hard to observe and even dangerous. For example, either Hotbit or RANDOM.ORG require specific hardware. Consequently, they are not as simple to implement as one that requires just a camera as a sensor. Secondly, we extended a classic approach for the Shape from Shading problem to work on images with natural and outdoor illumination, with distant objects and non-Lambertian surfaces. Even though the results qualitatively are not as precise as expected, we have a reconstruction that is able to generate random numbers from the variation of the normal directions of the water surface. In addition, it still estimates surfaces with detailed level of similarity to the observed one.

5.1 Limitations

As a consequence of depending on a camera to capture images, the random numbers are generated only the time with daylight. The shape from shading technique also assumes that we will be dealing with just water, with no presence of foam, boats, birds and many other probable objects. We can notice the influence of this “noise” in the fourth example.
of Section 3.1. The number of bits generated by a single image is another issue. In this work, the maximum number of bits was 20000. We could generate more, meaning that we would sample more normal vectors from each image. However, at some point this could result in a spatial dependence between the bits, consequently, the generator could fail some tests. This problem happens when we did not use previously generated (weak) random indexes to scramble the pairs used in each iteration (see Chapter 3).

A specific test fails, the Fast Fourier Transform, this happens because by using the same pairs, a pattern starts to emerge as the quantity of generated numbers increase. Another limitation is the processing time, to process an image with MATLAB in an Intel i5, 2.4 GHz, it takes approximately 15 seconds. With our current implementation, a larger image would demand more processing time (if we desire to keep the same generation time). Therefore, for an expansion of the number of random numbers created, it is necessary to optimize the implementation of the proposed technique.

### 5.2 Future Works

The processing time is a critical problem to be approached. We understand that most of the 15 seconds come from the Shape from Shading part of the technique. Consequently, improvements on this part is a key factor. New techniques for Shape from Shading still have to be researched to improve this results. Another area open for improvement is the use of previously generated numbers on the next steps. Although this is not affecting the results, as the tests are passing, a better approach may exist. Perlin noise [13] employs hashing techniques which can be a solution for this problem.
Bibliography


