5-2016

Visualization of Three-Dimensional Models from Multiple Texel Images Created from Fused Ladar/Digital Imagery

Cody C. Killpack
Utah State University

VISUALIZATION OF THREE-DIMENSIONAL MODELS FROM MULTIPLE TEXEL IMAGES CREATED FROM FUSED LADAR/DIGITAL IMAGERY

by

Cody C. Killpack

A thesis submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Computer Engineering

Approved:

Dr. Scott E. Budge Major Professor	Dr. Jacob H. Gunther Committee Member
Dr. Don Cripps Committee Member	Dr. Mark R. McLellan Vice President for Research and Dean of the School of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2015

Abstract

Visualization of Three-Dimensional Models from Multiple Texel Images Created from Fused Ladar/Digital Imagery

by

Cody C. Killpack, Master of Science

Utah State University, 2015

Major Professor: Dr. Scott E. Budge
Department: Electrical and Computer Engineering

The ability to create three-dimensional (3D) texel models, using registered texel images (fused ladar and digital imagery), is an important topic in remote sensing. Offering both visual and dimensional accuracy, texel models are becoming invaluable tools for disaster management, situational awareness, and even automatic target recognition (ATR). These models are automatically generated by matching multiple texel images into a single common reference frame. However, rendering a sequence of independently registered texel images often provides challenges, and the problems that arise when viewing texel models will be discussed and corrected in this paper.

As a result of stretched triangles and sensor noise, standard rendering techniques are not suitable for texturing 3D texel models. Although accurately registered, the model textures are often incorrectly overlapped and interwoven. Forming a texture atlas (stitching images together) is impractical, due to the independent nature of the underlying data sets. Consequently, corrections must be done after all the primitives have been rendered by determining the best texture for any viewable fragment in the model. Determining the best texture is difficult, as each texel image remains independent after registration. The depth data is not merged to form a single 3D mesh, thus the individual image textures must be

rendered separately. It is therefore necessary to determine which textures are overlapping and how to best combine them during the render process. The best texture for a particular pixel can be computed using 3D geometric criteria, in conjunction with a real-time, view-dependent ranking algorithm. As a result, overlapping texture fragments can be hidden, exposed, or blended according to their computed measure of reliability.

The visual defects of a texel model are reduced when overlapping textures are rendered using the view-dependent algorithm discussed in this paper. Successful rendering examples using synthetic data sets are presented.

(74 pages)

Public Abstract

Visualization of Three-Dimensional Models from Multiple Texel Images Created from Fused Ladar/Digital Imagery

by

Cody C. Killpack, Master of Science

Utah State University, 2015

Major Professor: Dr. Scott E. Budge
Department: Electrical and Computer Engineering

The ability to create three-dimensional (3D) images offers a wide variety of solutions to meet ever increasing consumer demands. As popularity for 3D cinema and television continues to grow, 3D images will remain an important area of research and development. While there are a variety of ways to create a 3D model, *texel images* are quickly becoming the preferred solution for many visual applications. A texel image contains fused depth and color information that has been captured with a *texel camera*. The combination of multiple texel images taken around a scene can be used to form a *texel model*. Offering both visual and dimensional accuracy, texel models are becoming invaluable tools for disaster management, situational awareness, and even military applications. However, displaying a texel model often provides challenges, and the problems that arise when viewing texel models will be discussed and corrected in this paper.

Dedicated to Nicolette, Penelope, and Elizabeth

Acknowledgments

I would like to thank my advisor and major professor Dr. Scott E. Budge, whose knowledge and guidance has been unwavering throughout my academic career. I deeply appreciate his support, collaboration, and patience with me. I am indebted for the limitless time and support he continually gives us as research students. I would also like to thank my committee members, Dr. Jacob H. Gunther and Dr. Don L. Cripps, for their insights and instruction during my stay at Utah State University. I would also like to thank the ECE and CS Department faculty and staff, especially Dr. Kenneth Sundberg, Dr. Sanghamitra Roy, and Mary Lee Anderson.

I would like to thank my fellow colleagues at the Center of Advanced Imaging Ladar, Neeraj Badamikar, Xuan Xie, and Taylor Bybee, whose knowledge and advice has elevated my research experience. I would also like to thank the United State Air Force, for continuing to provide me with opportunities to learn and grow. I would also like to express my gratitude toward fellow software maintenance engineers and friends Bryce Pincock, Brandon Struthers, Troy Webster, and many more.

I am grateful for the support of my loving family, who I can never thank enough for their patience and understanding. I owe my success to the guidance and teachings of my parents, Kelly and Shelley Killpack, and to the love and support of my siblings, Katie and Shalee. I am grateful for the love and support of my wife, Nicolette, and the encouragement from my two beautiful children, Penelope and Elizabeth. It has been an incredible journey, and I look forward to sharing with you the many adventures yet to come.

Cody C. Killpack

Contents

List of Figures

Acronyms

3D	Three-Dimensional
API	Application Programming Interface
ATR	Automatic Target Recognition
BSP	Binary Space Partitioning
CAIL	Center for Advanced Imaging Ladar
CMR	Common Mode Rejection
COP	Center Of Projection
EO	Electro-Optical
FBO	FrameBuffer Object
FOV	Field Of View
FPS	Frames Per Second
GL	Graphics Library
GLEW	OpenGL Extension Wrangler
GLSL	OpenGL Shading Language
GLUT	OpenGL Utility Toolkit
GPU	Graphics Processing Unit
HSR	Hidden Surface Removal
ICP	Iterative Closest Point
ILF	Image Ladar Format
POV	Point Of View
SOP	Surface Operator
TOF	Time Of Flight
XLF	Extensive Ladar Format

Chapter 1

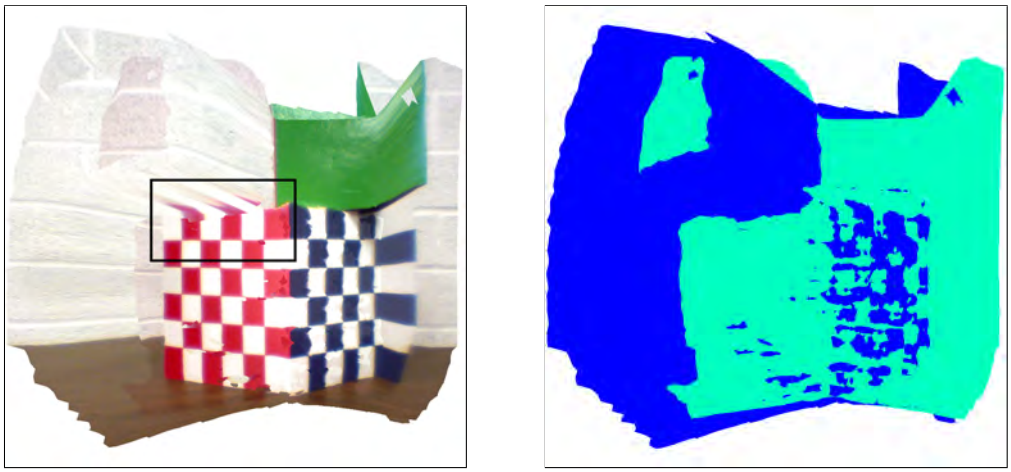
Introduction

The need for automatically generating 3D models exists in many technical fields of study. Creating a 3D model, with visual and dimensional accuracy, is an invaluable tool for disaster management, situational awareness, and even automatic target recognition (ATR). Building these models involves combining 3D data captured from various viewpoints around an object of interest. Once the images are matched with a registration algorithm, they can be viewed and explored in a vector graphics tool. The ability to apply and overlay accurate textures within this viewing program is an important step towards achieving a usable model.

The Center for Advanced Imaging Ladar (CAIL) at Utah State University uses texel images (fused ladar and digital imagery) to create 3D models. Texel images are generated with a texel camera, which houses a co-axial ladar array and electro-optical (EO) camera. The data produced by a texel camera is fused at the pixel level [1–3]. Researchers at CAIL have exploited this data fusion to register multiple texel images into a single 3D model [4]. The transformation details computed during registration are used to match individual texel images, bringing them into a single reference frame (common world coordinate system). The result is a 3D model composed of independent 2.5D texel images.

Rendering these 3D texel models continues to be an important research focus at CAIL. Rendering involves loading and viewing a 3D model. This process is currently being done at CAIL with a proprietary software package known as the *3D Viewer*. The *3D Viewer* provides the ability to magnify, transform, and explore a texel model in 3D space. However, viewing a sequence of independently registered texel images often provides challenges, and the problems that arise when using standard rendering techniques need to be corrected. Even when a 3D model is accurately registered, the individual textures assigned to it are often incorrectly overlapped and interwoven. This is caused by stretched triangles, noise,

and the rendering methods used by the *3D Viewer*, as shown in Figure 1.1. In Figure 1.1(a), the problems with stretched triangles caused from different viewpoints are clearly illustrated. Figure 1.1(b) shows the same registered scene with the textures replaced by a color assigned to each image, indicating where the rendered textures originate.



(a) Textured 3D Texel Model (b) Colored 3D Texel Model

Fig. 1.1: Registered 3D texel model with noise and distortion present. The individual texel images, or scans, were taken from two slightly different perspectives. (a) The combination of these two texel images in an independently registered and textured sequence. (b) The same scene found in (a) but with a unique color assigned to each image rather than a texture.

Forming a texture atlas (stitching textures together) is impractical due to the independent nature of the underlying data sets. Consequently, corrections must be done after all the primitives (triangles) have been rendered by determining the best texture for any given area of the model. Triangle based multi-texturing techniques have been proposed and define the best texture for a given area as a triangle labeling problem [5].

The best texture for a particular area of the model can be defined using a 3D geometric criterion. Additionally, this geometric criterion must be used in conjunction with a real-time, view-dependent ranking algorithm, as the virtual camera perspective also influences the suitability of a texture. The concept of view-dependent texture mapping has been proposed along with examples of real-time applications [6, 7]. Well known 3D geometric criteria have also been demonstrated, such as the angle between a triangle normal and the viewing frustum, distance to the camera, surface area of a triangle face, and even

photo-consistency metrics [5,8–10]. These methods have all been explored and ranked [11]. A combination of these metrics, referred to as the *reliability* of the texture, would allow overlapping texture fragments (pixels) to be hidden, exposed, or blended according to their computed measure of reliability. Methods for blending these textures have been proposed [12, 13]. Current techniques determining the visibility and rank of a texture triangle can also be found [13, 14].

The problem addressed in this paper is uniquely challenging because each texel image remains independent after registration. The depth data has not been merged, eliminating the need for a fused texture atlas. Therefore, the solution must determine which textures are overlapping, and how to best combine them dynamically while rendering. The OpenGL Shading Language (GLSL) enables textures to be combined on a fragment level (per-pixel) during the render process. Furthermore, the accuracy of a texture triangle can be interpolated, which results in greater uniform interaction among overlapping textures. As a result, the best texture assignment for a given area of the 3D texel model can be done on a per-fragment basis. This is achievable in real-time for applications that use texel images obtained from framing ladar sensors. Attempting real-time rendering solutions for models produced by commercial texel cameras becomes difficult due to the rapid increase of data and computation required. For this reason, high resolution solutions usually rely on point-based rendering (drawing an abundance of colored dots or small circles) to ease computation and neglect traditional 2D texturing altogether [15]. However, colored point-based rendering would waste large portions of digital imagery, and is therefore an unusable solution for the texel images created at CAIL. Ergo, investigating the model texture problem remains beneficial, as the goal of CAIL has always been to provide accurate and robust solutions for low budget consumers.

The remainder of this paper is presented as follows. A brief history of the previous work done at CAIL is presented in Chapter 2. Information regarding fundamental concepts, published research, and proprietary software will also be explained. The motivation behind model rendering research is further illustrated. Chapter 3 explores the generation

of synthetic ladar data, which is used to prove the feasibility of the proposed rendering method presented in this paper. Chapter 4 contains the research and theory behind the view-dependent multi-texturing render algorithm for registered texel images. The results can be found in Chapter 5, where examples using synthetic data sets are given. Chapter 6 concludes the paper, giving insight into the limitations and fallacies of the rendering algorithm defended. Future work is also discussed.

Chapter 2

Background

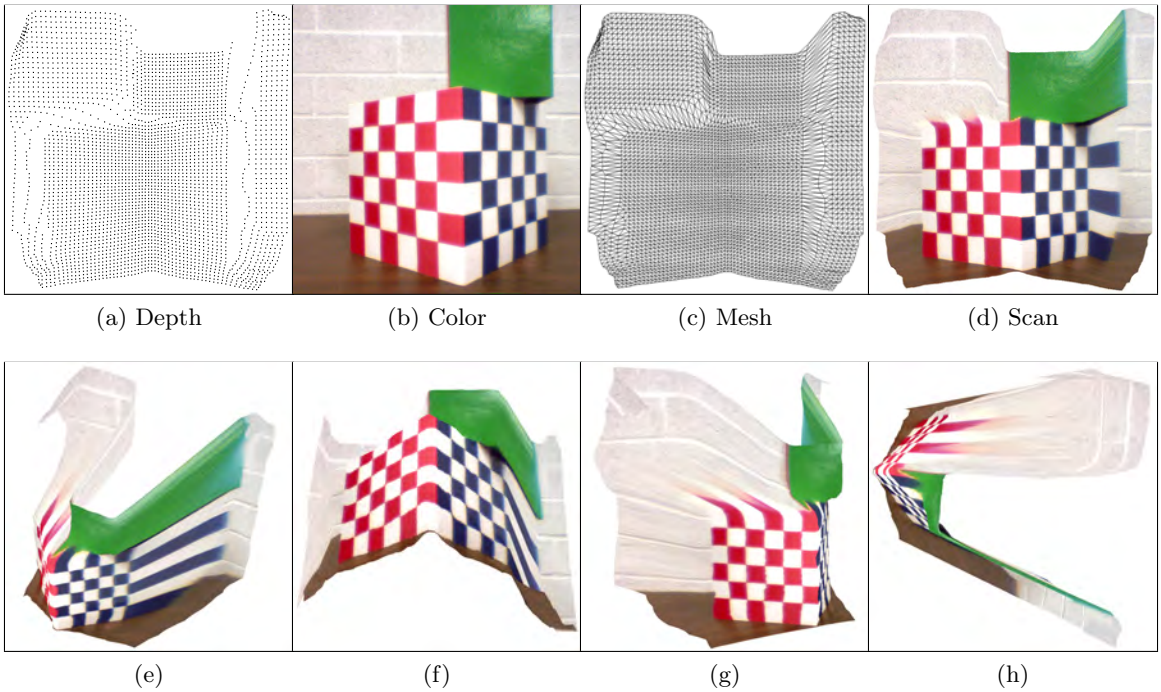
The motivation for improving the textures of 3D models stems from the previous work done at CAIL. This chapter provides insights into the foundational work, published research, and proprietary software developed by researchers at CAIL. Section 2.1 begins the chapter with an introduction into texel cameras, including the true definition of a texel image, which is often misunderstood in the literature. Next, novel developments in texel image registration are presented in Section 2.2. To conclude the chapter, Section 2.3 explores how texel images are currently being viewed at CAIL and reveals the unique challenges being faced during the rendering process. Additional insight for the concepts discussed in this chapter can be found in the cited CAIL publications.

2.1 Texel Images

Fused ladar and digital imagery constitute what is known as a texel image. These images are created from a data acquisition device called a texel camera, which houses both an electro-optical and ladar sensor. The EO sensor captures color information, while the ladar sensor captures range information. Combined during acquisition, these two data sets form a 2.5D colored surface. A typical texel camera uses the ladar and EO sensors in such a way that their optical pathways coincide via a technique known as coboresighting [16]. The pathway coinciding process is accomplished by integrating a cold mirror into the system. This unique implementation allows depth and color data to be fused at the pixel level. As a result of real-time fusion, there is no additional post-processing required. A correctly coboresighted texel camera eliminates parallax and allows both sensors to view objects from the same point of view (POV). This coboresighting technique is protected by a patent and has been the cornerstone of CAIL research and development [17]. Details concerning the

previous texel image research done at CAIL have been published [2, 18, 19]. In addition, further insights into texel cameras, coboresighting, and sensor calibration are also available [3, 16].

The depth and color components of a texel image become familiar when used within a vector graphics viewing tool. The depth data is simply a point cloud, consisting of vertices whose XYZ coordinates represent measured range values. The color data is a 2D digital image, used as a texture for a tessellated point cloud. When viewed together, the depth and color data form a 2.5D textured mesh, or texel image. This complete image is also referred to as a *scan*. Figure 2.1 illustrates these components with a texel image captured using a handheld texel camera developed at CAIL.

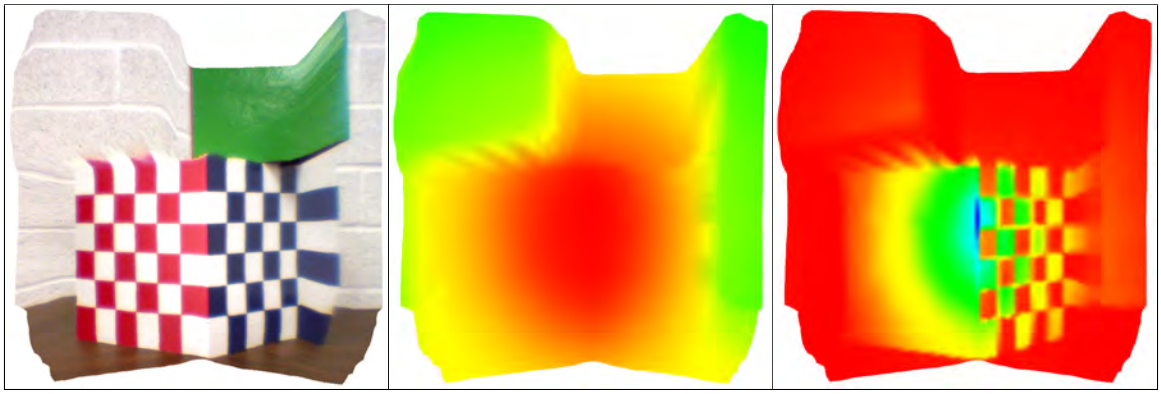


(a) Depth (b) Color (c) Mesh (d) Scan

(e) (f) (g) (h)

Fig. 2.1: Components of a texel image. The depth and color information was captured using a co-axial texel camera in a research laboratory at CAIL. (a) Raw depth data captured by the ladar sensor mounted on the handheld texel camera. (b) Raw color data captured by the EO sensor mounted on the handheld texel camera. (c) Triangulated mesh of (a) using a basic row-by-column tessellation method. Generating a triangle mesh allows textures to be easily applied to objects during rendering. (d) Combination of (a)–(c) to form a 2.5D texel image or scan. For reference, a variety of different perspective views for (d) can be seen in (e)–(h).

At a minimum, texel images generated at CAIL are required to have at least the following: a point cloud, a texture, and a set of UV coordinates providing a mapping between the two. It should be noted however, that additional per-vertex information may also be available, depending on the texel camera and acquisition software being used. Some examples include vertex range and vertex intensity. The range of a vertex (meters) represents the actual distance between the ladar center of projection (COP) and the sample point in 3D space. The intensity of a vertex represents the strength of the ladar light pulse returned (object reflectivity). A visual interpretation of range and intensity information can be seen in Figure 2.2. Intensity can be affected by external light sources and is often influenced by the textures and materials of the object being sampled. In this example, the right face of a checkered cube was painted using a high gloss white and a flat dark blue, and the reflective properties of these two paint finishes become noticeable when examining Figure 2.2(c).



(a) Image Texture (b) Image Range (c) Image Intensity

Fig. 2.2: Texel image range and intensity measurements. The sample point (vertex) values have been interpolated across the texel image canvas. (a) Texel image captured in a research laboratory at CAIL. (b) Interpolated range values of (a), displayed using a heat map coloring scheme. Warm colors represent close acquisition ranges while cooler colors represent sample points captured further away from the sensor. (c) Interpolated intensity values of (a), displayed using a heat map color scheme. Warm colors represent areas of low reflectivity and cooler colors represent sample points showing higher levels of reflectivity.

2.1.1 Image Sensors

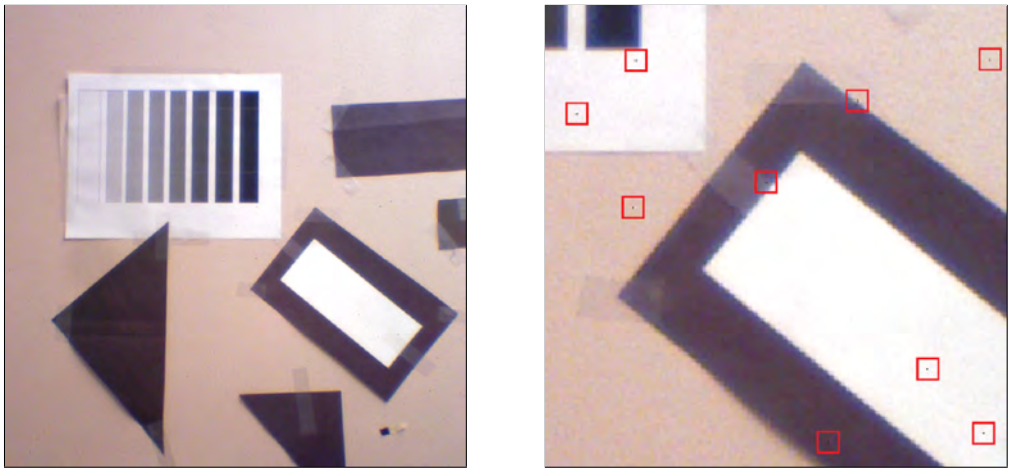
An image sensor, or Electro-Optical sensor, is a color sensing device used to capture the texture information of a given object or scene. CAIL currently uses a Micron 1280 x 1024 CMOS sensor, which has been encapsulated into a stand-alone development board. The board easily interfaces with proprietary CAIL acquisition software and can provide a real-time image feed to the operator. The camera captures RGB color, which is eventually exported as an image file. CAIL software currently generates bitmap image files, however any file format could potentially be used.

CAIL has made minor alterations to the Micron EO sensor, including changing the lens and altering the final resolution. Since the EO sensor was to be ultimately paired with a ladar sensor, it was necessary to match the fields of view (FOV) for both devices. The ladar sensor currently used by CAIL has a FOV of just under 50 degrees, and the EO sensor was given an appropriate replacement lens to match this requirement. In addition, the final resolution of the image needed to be altered as well. The original resolution of 1280 x 1024 has been resized to a 1024 x 1024 power-of-two dimension. This complies with the backwards compatibility requirements of older CAIL graphics programs, specifically those which use Open Graphics Library (OpenGL) versions below 2.0.

Figure 2.3 shows a sample color image acquired from the Micron EO camera. The image is a 24-bit RGB bitmap which has been resized to the required 1024 x 1024 dimensional format. Figure 2.3(b) shows a zoomed portion of Figure 2.3(a) to draw attention to dead pixel areas within the image. These black pixels represent comprised areas inside the EO sensor hardware. Unfortunately, heavy use and continual degradation will eventually render this particular sensor obsolete. CAIL is currently configuring a newer and higher resolution EO camera as a future replacement.

2.1.2 Ladar Sensors

Ladar, an acronym for LAser Detection And Ranging, is a remote sensing device used for measuring distance. Ladar is synonymous with lidar (LIght Detection And Ranging) and both terms are used interchangeably in the literature. A ladar emits pulses of light



(a) Original Image (b) Zoomed Image

Fig. 2.3: Digital image taken with a Micron CMOS color EO sensor. (a) 24-bit RGB bitmap (1024 x 1024) which has been captured and exported from a scene created in the CAIL laboratory. (b) Zoomed portion of (a) with attention drawn to dead pixel areas using graphical red box overlays. Dead pixels (black) represent compromised areas within the EO camera hardware.

to illuminate targets. The reflected light is then analyzed to extract range measurements. Computing the range is usually done in one of three ways: time-of-flight (TOF), frequency modulated continuous wave, or amplitude modulated continuous wave [20]. Although similar to radar, a ladar device operates at much higher frequencies, resulting in accurate measurements at the expense of distance [16].

CAIL currently utilizes a Canesta 64 x 64 CMOS TOF ladar sensor. The sensor emits a pulse of light and calculates range based on the time elapsed for the light pulse to return. The time expired is linearly dependent on the distance between the ladar sensor and the reflecting surface. The 64 x 64 array captures range information at multiple frames per second (FPS) resulting in 4096 total sample points. A collection of sample points, or vertices, represent a single point cloud in 3D space. A point cloud describes the shape, size, and distance of a real-world object or scene. An example data set, captured with a ladar sensor in the CAIL laboratory, is shown in Figure 2.4. The vertices of a point cloud are often connected together to form a mesh of triangles, or wireframe, as shown in Figure 2.4(b). Tessellating a point cloud allows colored images to be easily applied across its surface, an

important and fundamental vector graphics concept called *texturing*.

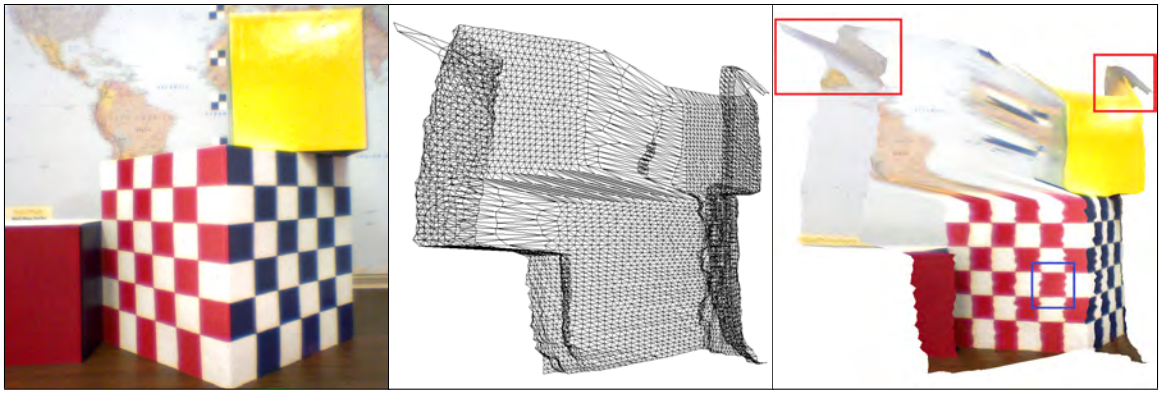
(a) Depth (b) Mesh (c) Color

Fig. 2.4: Depth image taken with a Canesta 64 x 64 CMOS TOF ladar sensor. (a) Depth data returned from the target scene. (b) Tessellated form of the point cloud given in (a), where a basic row-by-column tessellation method was used to generate a triangle mesh. (c) Image of the original laboratory scene, given for reference.

Even though ladar sensors have become sophisticated range acquisition tools, they continue to suffer from certain limitations. The most obvious is the restriction on reliable range of operation imposed by the use of higher frequency signals. In fact, the handheld Canesta camera currently used by CAIL is only accurate when positioned less than two meters away from a target scene. It should be noted that sensors capable of capturing data at greater distances are available, but their size and price rapidly increase.

Additional sensor limitations include noise and degradation. Noise is often seen in images taken by ladar sensors, especially when aimed at objects that are shiny or glossy. Various materials, fabrics, and paint finishes can cause unwanted timing effects when reflecting ladar light. The result is often disturbing, and causes various uneven and bumpy areas within the image. In addition to noise, degradation can also affect the quality of an image. For example, the Canesta camera currently used by CAIL has begun to lose capture quality as a result of excessive heat exposure and frequent use. The impact of noise and degradation on the ladar sensor can be seen in Figure 2.5. The two red box overlays in Figure 2.5(c) highlight the effects of degradation present in the point cloud. The blue box overlay in Figure 2.5(c) showcases the uneven depth data captured around the colored boxes

of the checkered cube. This distortion is primarily due to unreliable TOF measurements, which are the results of using glossy paints to cover the cube model with a checkerboard pattern. An uneven and bumpy point cloud can cause the texture laid across it to become stretched, pixelated, and unnatural.



(a) Color (b) Mesh (c) Scan

Fig. 2.5: Ladar acquired depth image with noise and degradation present. (a) Colored image of the laboratory scene taken with an EO sensor. (b) Tessellated point cloud captured in the CAIL laboratory using a Canesta ladar sensor. Notice the stretched mesh triangles in the upper left and upper right corners of the image. This is a common artifact of sensor degradation and calibration error. (c) Combination of (a)–(b), emphasizing the presence of noise.

It is important to note that certain measures can be taken to reduce the effects of noise and degradation against a ladar image. CAIL utilizes a variety of parameters when acquiring and processing ladar data sets. Shutter time, frame rates, common mode rejection ratios (CMR), modulating frequencies, and sensor light illumination power need to be taken into account when capturing depth data. Caveats and insights when using these parameters are available [18]. In addition, sensor calibration, lens adjustment, and mechanical mounting techniques also become critical factors for reducing noise and increasing data quality [16].

2.2 Texel Image Registration

Creating 3D texel models from registered 2.5D texel images continues to be an important research focus at CAIL. Recently, extensive progress has been made on this front, including new techniques involving image-based registration [4, 16] rather than exclusive

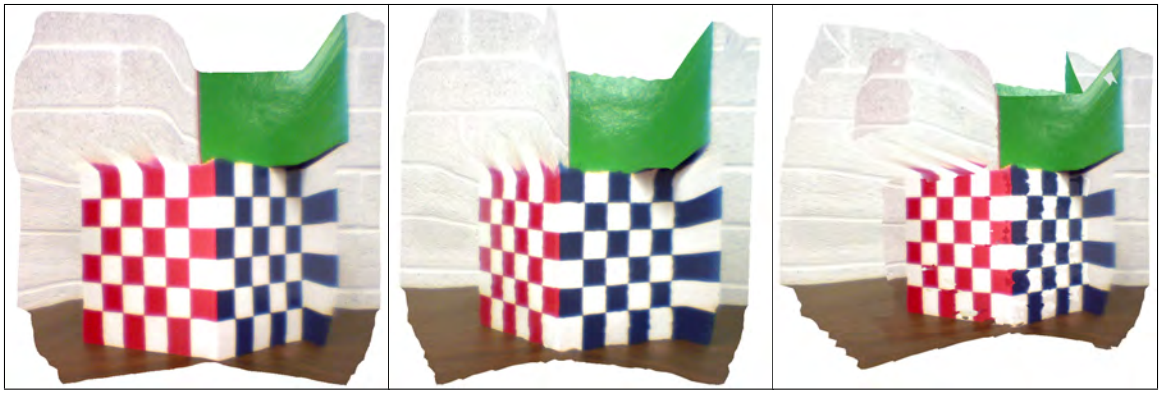
point cloud registration techniques, such as the Iterative Closest Point (ICP) algorithm and all its well know variants. An image-based matching approach is unique because it utilizes the color information of a texel image during the registration process. This is made possible due to the inherent nature of a texel camera, where the color and range information are fused at the pixel level. This means that processing can be done on either of the two data sets. If the point clouds are matched, it implies that the texture images will be matched as well and vice versa. Using the EO image, instead of the point cloud, exploits a critical caveat of texel cameras: EO sensor resolution will always be better than ladar sensor resolution. Even the most advanced and expensive ladar systems cannot compete with the high resolution color cameras so readily available today. Consider the handheld texel camera currently used by CAIL, which has an EO camera component with a resolution of 1024 x 1024 while the ladar component of the system has a resolution of 64 x 64. Consequently, using the EO image for registration instead of the point cloud, results in greater accuracy and reliability.

Figure 2.6 demonstrates two different examples of image-based registration. Figures 2.6(a)–(b) show two independent texel images, acquired from different vantage points within an identical scene, which have been matched together to form the 3D texel model shown in Figure 2.6(c). Similarly, Figures 2.6(d)–(f) show an additional example of the image-based registration process. An overview of the steps used to generate matched texel models is summarized in Algorithm 1 [16]. Notice that the first few steps of the algorithm rely solely upon the 2D color information of the given texel image. Once a matching model

Algorithm 1 Image-Based Registration of Independent Texel Images

1. Detect Harris Features
2. Find Putative Correspondences
3. Establish A Model That Fits The Corresponding Features
4. Generate An Optimal Estimation Of The Model
5. Estimate The Final Orientation Using Ladar Points

(a) Left Scan (b) Right Scan (c) Texel Model



(d) Left Scan (e) Right Scan (f) Texel Model

Fig. 2.6: Texel image registration. (a)–(b) Two slightly different perspectives of an identical scene created in the CAIL research laboratory. (c) Texel model generated from matching and registering the scans in (a)–(b). An additional registered texel model example is given in (d)–(f).

is found, the final 3D transformation is computed using the 3D ladar points. During this registration process, the transformation details for each texel image are stored in a unique Surface Operator (SOP) matrix.

2.2.1 Surface Operator Matrices

A SOP matrix is simply a transformation matrix. It is composed of a single translation matrix and three rotation matrices. Such a matrix allows (x, y, z, w) vertices in homogenous coordinates to be translated and rotated about the X, Y, and Z axis. The SOP matrix is expressed by multiplying translation matrix T, given by

$$T = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

and rotation matrices R_x, R_y, and R_z given as

$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha & 0 \\ 0 & -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad R_y = \begin{bmatrix} \cos\beta & 0 & \sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad R_z = \begin{bmatrix} \cos\gamma & \sin\gamma & 0 & 0 \\ -\sin\gamma & \cos\gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

The resulting matrix, $TR_xR_yR_z$, then takes the form

$$TR_xR_yR_z = \begin{bmatrix} \cos\beta\cos\gamma & \cos\beta\sin\gamma & \sin\beta & t_x \\ -\sin\alpha\sin\beta\cos\gamma - \cos\alpha\sin\gamma & -\sin\alpha\sin\beta\sin\gamma + \cos\alpha\cos\gamma & \sin\alpha\cos\beta & t_y \\ -\cos\alpha\sin\beta\cos\gamma + \sin\alpha\sin\gamma & -\cos\alpha\sin\beta\sin\gamma - \sin\alpha\cos\gamma & \cos\alpha\cos\beta & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Each texel image is assigned a unique SOP matrix during the registration process. These matrices allow texel images to be transformed from a model coordinate system to a world

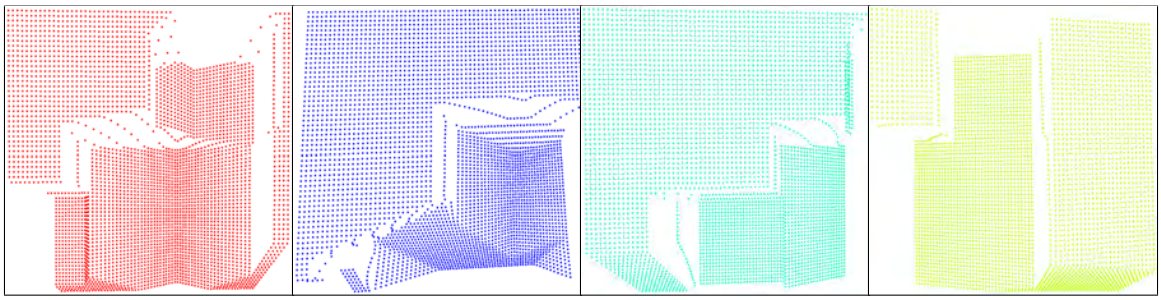
coordinate system. When rendered together, the result is a 3D model composed of independent texel images. The texel images are considered independent because their data sets are not combined after registration. The point clouds have not been merged and the textures are not stitched together. Instead, the texel images remain separate, containing only the SOP matrix generated during registration. The combination of a texel image and a SOP matrix is called a *scan*.

The registration process matches a *sequence* of scans around a centralized texel image, denoted as the *base*, which is usually the first image in the sequence. The remaining texel images are then matched to the coordinate system of the base image. Once complete, the corresponding SOP matrices will contain the appropriate transformations to rotate and translate all texel images into the same world coordinate system. Not surprisingly, the SOP matrix for the base texel image is an identity matrix.

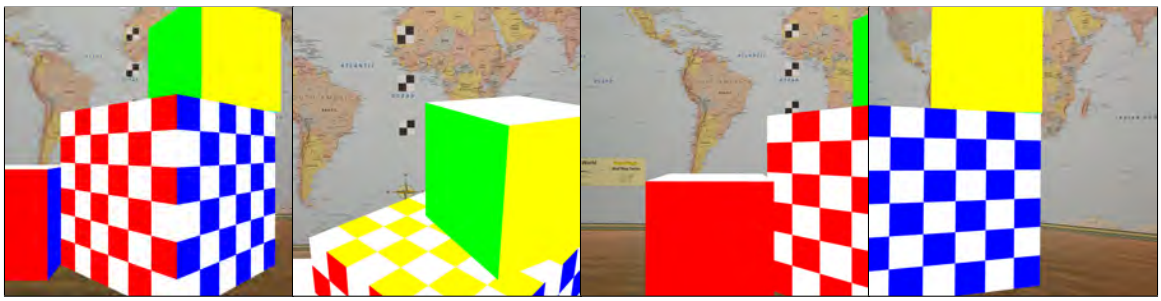
Figure 2.7 helps solidify this concept. In this example, synthetic ladar data has been used to better illustrate the process, however the same principles apply to real-world data sets. Figure 2.7(a) contains the base scan for an entire sequence of texel images. For clarity, the corresponding texture element of the base scan is shown in Figure 2.7(e). The scan sequence to be registered is shown in Figures 2.7(b–d). Again, companion Figures 2.7(f–h) offer the corresponding image textures for these scans. Using the SOP matrices generated during registration, the three sequence scans are translated and rotated into the coordinate system of the base scan during render time. The independent texel images now form a 3D texel model, as shown in Figure 2.7(i).

2.2.2 Extensible Ladar Format

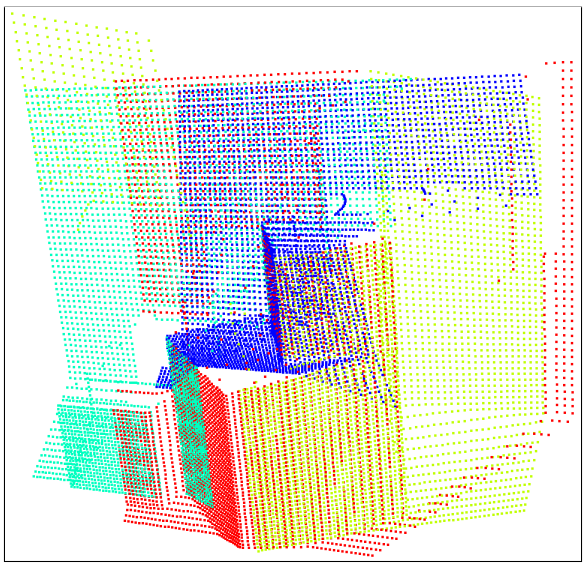
To better manage scan and sequence data, CAIL has developed a proprietary standard for encapsulating texel image information. This format provides a convenient way to access, manipulate, and modify the data contained in related sets of images. An Image Ladar Format (ILF) is an archive structure which contains header, description, SOP, and data files for a particular scan. The header file contains details about the type of camera used to capture the data, including orientation and field of view. The data files associated with



(a) Base Scan (b) Top Scan (c) Left Scan (d) Right Scan



(e) Base Texture (f) Top Texture (g) Left Texture (h) Right Texture



(i) Texel Model

Fig. 2.7: Matched synthetic texel model. (a) Base scan in the sequence. The model coordinate system of (a) becomes the common reference (world) coordinate system for all other scans in the sequence. (b)–(d) Three additional scans within the sequence, captured from a variety of scene perspectives. (e)–(h) Companion color textures for (a)–(d), shown for clarity. (i) 3D texel model created from rendering independent texel images where scans (b)–(d) have been translated and rotated into the coordinate system of base scan (a) using their corresponding SOP matrices.

the scan are also encapsulated and typically include:

1. xyz.dat (3D point cloud data)
2. texture.bmp (2D digital image)
3. uv.dat (image (u,v) coordinate-to-xyz-point mapping)
4. intensity.dat (ladar intensity data)
5. range.dat (vertex range measurements)
6. row.dat (vertex row coordinates)
7. column.dat (vertex column coordinates)

Only the first three files in the list are required. Any additional information is optional. The data can be in any format (binary integers, floating-point numbers, etc.) as defined in the header. Once a sequence of *ilf* files have been generated, they are encapsulated together into a single archive structure called an Extensive Ladar Format (XLF). The format of the *xlf* file provides significant flexibility and can be easily parsed using a compatible rendering program, such as the CAIL *3D Viewer* package. An overview of the *ilf* and *xlf* architecture is shown in Figure 2.8.

2.3 Default Model Rendering

Rendering involves loading and viewing a model. It also usually implies an ability to magnify, transform, and explore that model in 3D space. However, rendering and manipulating a sequence of independently registered texel images provides challenges, and the problems that arise when using standard viewing techniques must be corrected. The application programming interface (API) used by CAIL for 3D vector graphics rendering is the Open Graphics Library. To maintain backwards compatibility with older programs, version 3.0 is the current extension being supported for development. Combined with the companion libraries OpenGL Utility Toolkit (GLUT) and OpenGL Extension Wrangler Library (GLEW), interaction with the graphics processing unit (GPU) is accelerated. In addition, shader flexibility in the rendering pipeline is available with the OpenGL Shading Language

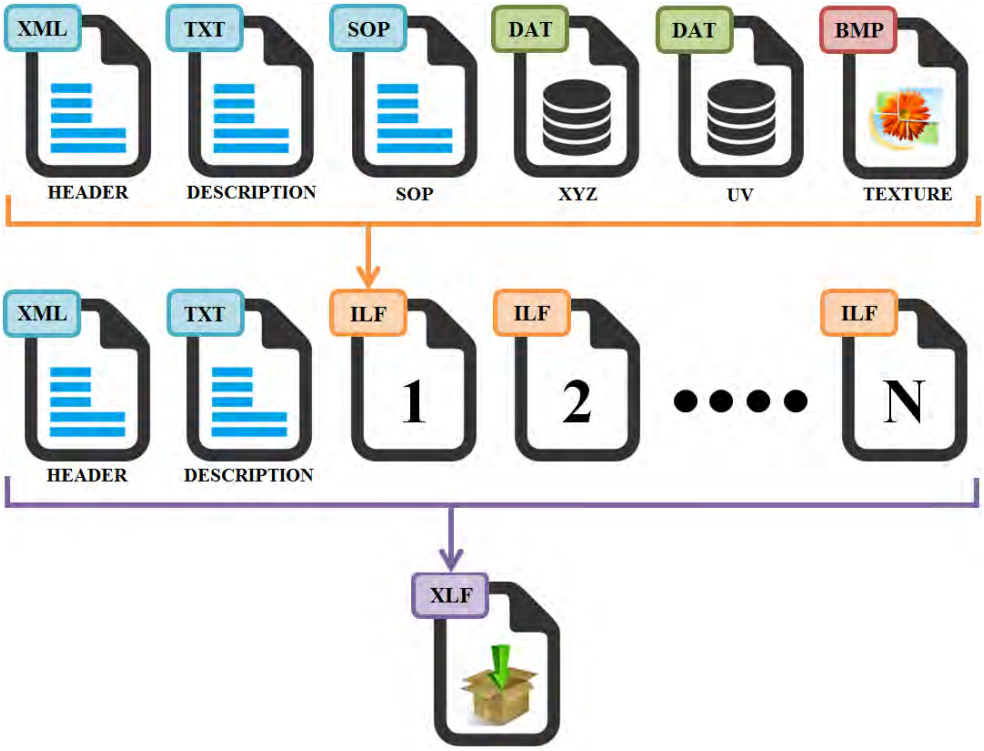


Fig. 2.8: Proprietary CAIL data encapsulation standard.

and enables nonconventional rendering techniques to be implemented.

2.3.1 *3D Viewer*

CAIL has developed a proprietary scan and sequence viewer called *3D Viewer*. Implemented with OpenGL, the *3D Viewer* is a powerful viewing tool that can texture and display a sequence of registered texel images stored in a *xlf* file. When rendering a sequence of scans, the *3D Viewer* currently uses depth buffering to solve scene visibility problems. The depth buffer, also called the z-buffer, stores a per-pixel depth value for a rendered object on screen. When another object occupying the same pixel space must be rendered, the depth values are compared, and the closer of the two is given preference. In many cases, z-culling is sufficient for rendering opaque textures, and the painter's algorithm [21], which paints distant objects first and closer objects second, seems like a reasonable choice.

Unfortunately, stretched triangles and sensor noise prove otherwise, as demonstrated previously in Figure 1.1. In this scene, two registered texel images are displayed in the *3D*

Viewer using standard rendering techniques. Although the images have been accurately transformed into the same world coordinate system, visible defects remain present. Figure 1.1(b) helps emphasize these rendering flaws by using color to distinguish where the two textures originate.

2.3.2 Motivation

Standard rendering techniques are not suitable for texturing 3D texel models. As a result, alternative and improved rendering methods merit further investigation. Proper texel model rendering is challenging, as each texel image remains independent after registration. The depth data is not merged to form a single 3D mesh, eliminating the possibility of generating and correcting a fused texture atlas. In addition, altering the data sets must be avoided, including any changes to the underlying UV mapping coordinates. It is therefore necessary to determine which textures are overlapping and how to best render them together in real-time [22].

Chapter 3

Synthetic Texel Images

Measuring the success of a potential texturing algorithm is often subjective and aesthetically rooted. While mathematical metrics do exist [11], the inherent nature of model rendering is simple: appearance is everything. However, evaluating undeveloped algorithms for texel images is difficult, especially when noisy and erroneous data is used. Consequently, using real-world models during initial algorithm development can mask flaws and introduce red herrings. Distinguishing the difference between technique limitations and inherent data errors is cumbersome, and ultimately impossible. To mitigate risk, noiseless (synthetic) texel data should be used during the early stages of research and development. Synthetic data facilitates testing, debugging, and evaluating a potential texturing algorithm. Once proven effective against noiseless data, the algorithm can then be matured to accommodate real-world data sets.

CAIL has developed a prototype software package for generating synthetic texel images known as the *Synthetic Scene Generator*. Using dimensionally accurate virtual environments, the tool creates sequences of noiseless images to form seamless 3D texel models. Details regarding the creation noiseless imagery can be found in this chapter and are presented as follows. Section 3.1 begins the chapter with a discussion on the fundamental principles surrounding virtual environments. Next, details concerning the acquisition and manipulation of synthetic color and depth data is addressed in Sections 3.2 and 3.3. Section 3.4 explores how synthetic imagery is correctly sampled. Section 3.5 contains the conventions for tracking and recording the transformation matrices needed for texel model registration. To conclude the chapter, Section 3.6 highlights the preliminary results produced by the *Synthetic Scene Generator* and showcases a variety of 3D texel model examples. For reference, a high-level block diagram for software flow is given in Figure 3.1.

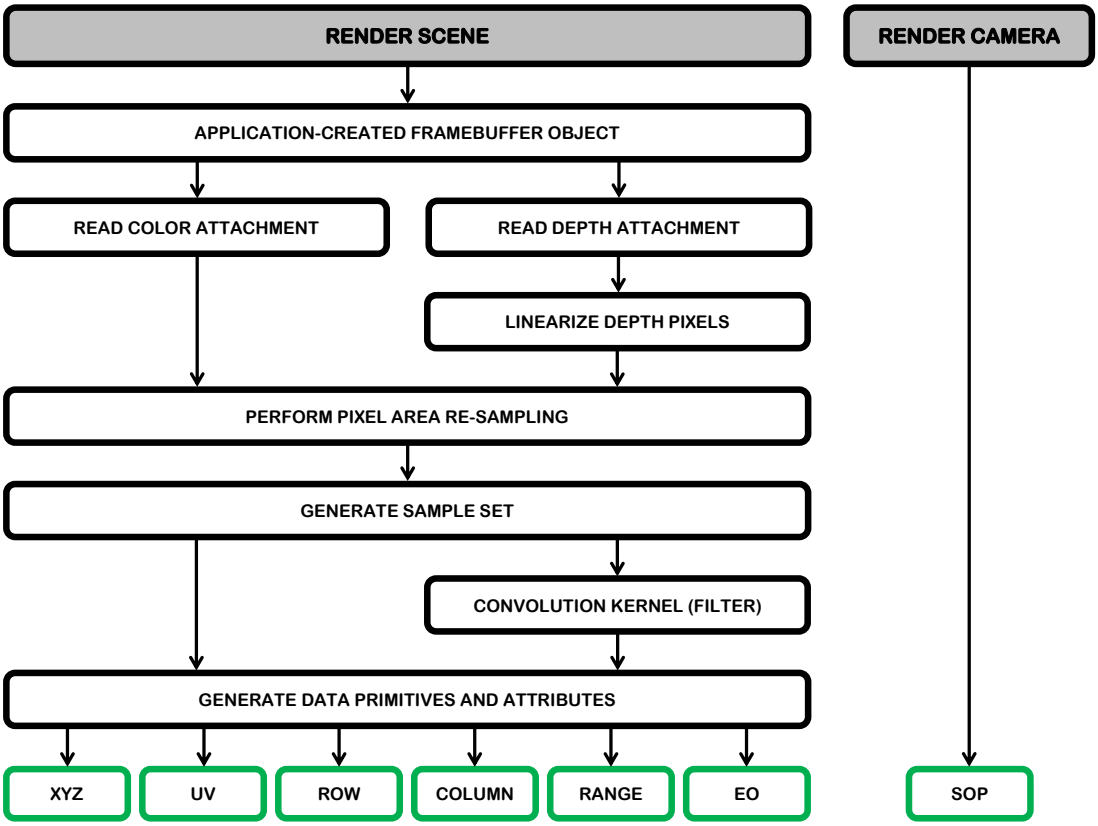
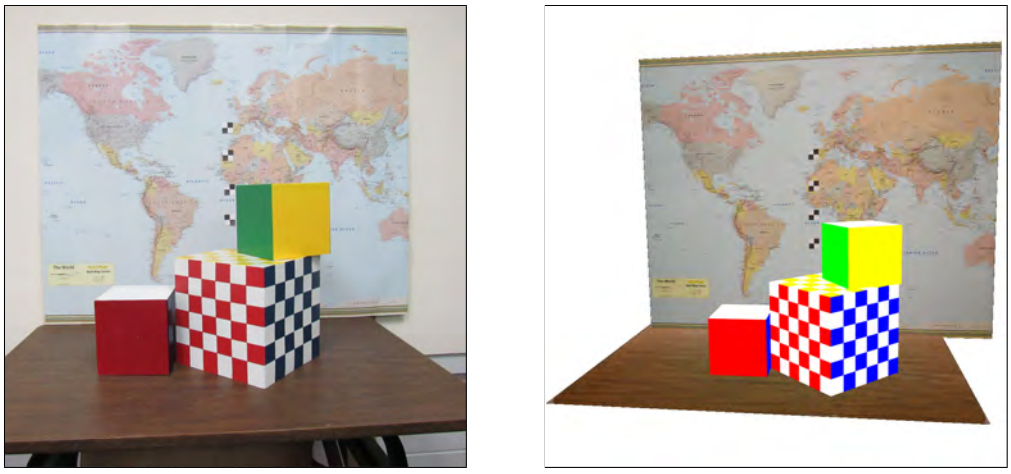


Fig. 3.1: *Synthetic Scene Generator* software flow chart.

3.1 Scene Fundamentals

At its core, the *Synthetic Scene Generator* is simply a vector graphics rendering tool capable of using geometric models and textures to replicate real-world environments, such as the one shown in Figure 3.2. OpenGL provides the means to sample synthetic environments with framebuffer objects, which store color and depth information for every pixel being rendered to the screen. When sampled correctly, the buffered per-pixel data can be used to emulate the appearance of an actual texel image. Synthetic scenes are designed to mimic authentic visual and dimensional accuracy, and must therefore be rooted in real-world coordinate systems. Coordinate conversion is derived from the virtual camera frustum, which must be tailored to the physical properties of the real texel camera being simulated. Unit conversion allows virtual models to be scaled down from real-world dimensions (meters) to virtual scene dimensions (unitless). Likewise, reversing the process allows the *Synthetic Scene Generator* to convert any sampled depth coordinates into useful measurements. For simplicity, a symmetric field of view is used during synthetic scene generation. Non-symmetrical viewing volumes may be used if the FOV width and the FOV height of a modeled texel camera are significantly different.



(a) Real Scene (b) Virtual Scene

Fig. 3.2: Virtual scene comparison. (a) Real-world scene created in the CAIL laboratory. (b) Replica scene of (a) created with the *Synthetic Scene Generator*. The virtual scene authentically simulates the dimensional and visual properties of its real-world counterpart.

3.2 Synthetic Color Data

A texel camera uses an electro-optical sensor to capture the RGB color information of a given scene. The current sensor being used at CAIL is a stand-alone development board produced by Micron. After accommodating for lens distortion and parallax, the final 1024 x 1024 resolution image is used as a mesh texture when being rendered. Fortunately, OpenGL provides the means to effectively mimic the characteristics of the Micron EO sensor. The steps for creating a virtual EO image are summarized in Algorithm 2 and elucidated in Sections 3.2.1 and 3.2.2 respectively.

Algorithm 2 Processing Synthetic Color Information

1. Read and store color attachment point.
2. Anti-alias the scene.

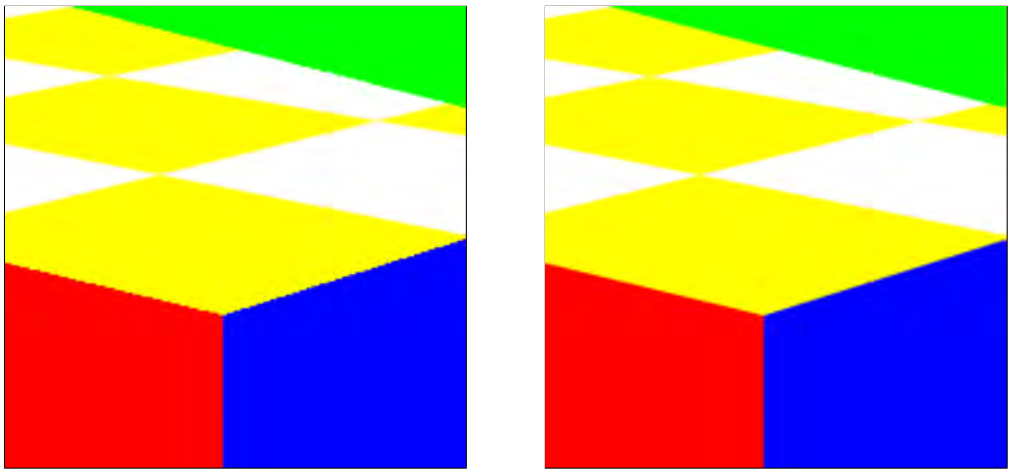
3.2.1 Pixel Acquisition

Reading the pixel data from an OpenGL buffer is relatively straightforward, regardless of whether a context or user-defined framebuffer object (FBO) is used. However, the *Synthetic Scene Generator* takes advantage of user-defined framebuffers, which allow scenes to be rendered off-screen at super resolutions, facilitating anti-aliasing. The process involves creating a renderbuffer to store colored pixel data. The renderbuffer is then assigned to the framebuffer via a color attachment point. After binding and checking for completeness, the FBO can now be used for off-screen rendering. Once the scene has been drawn, the pixel data can be read directly from the framebuffer and stored in any preferred data structure for further processing.

3.2.2 Scene Anti-Aliasing

Low resolution rendering of geometric shapes often introduces aliasing within a virtual scene. Aliasing often causes virtual scenes to look different from those captured with real EO sensors. While using high resolution monitors helps mitigate these effects, the problem

usually remains prominent, especially when rendering large shapes with sharp edges. The solution involves rendering the scene off-screen at a higher resolution, followed by resampling. As discussed previously, an alternative to the OpenGL context framebuffer (screen resolution dependent) is the creation of a user-defined framebuffer. A user-defined FBO allows rendering to non-default framebuffer locations without disturbing the main screen, and at higher resolutions. These oversized images can then be interpolated using pixel area relation (nearest-neighbor) and resized to the desired canvas size. The benefits of scene anti-aliasing are showcased in Figure 3.3. Once finished, the color canvas is written to an image file and is now ready to be used as the texture component within a synthetic texel image.



(a) 1X (b) 10X

Fig. 3.3: Anti-aliased virtual scene comparison. (a) Synthetic image captured at 1X resolution (1024 x 1024) with no enhancement. (b) Same scene found in (a) but captured at 10X resolution (10240 x 10240) then resized back to 1X resolution via interpolation.

3.3 Synthetic Depth Data

A texel camera uses a ladar sensor to capture the range information of a given scene. The current sensor being used at CAIL is a Canesta 64 x 64 CMOS TOF. Producing a total of 4096 pixels, these sampled vertices form what is known as a 2.5D point cloud. As a result of the sensor data being captured simultaneously, with no mechanical movement required,

the process of replicating the same result synthetically is greatly facilitated. OpenGL provides the means to effectively mimic the characteristics of the Canesta ladar sensor. The steps for creating a virtual ladar image are summarized in Algorithm 3. The procedures for pixel acquisition and scene anti-aliasing are identical to that of synthetic color data, and will not be revisited. Insights into the linearization and convolution steps of Algorithm 3 can be found in Sections 3.3.2 and 3.3.3 respectively. For additional reference, a brief OpenGL depth buffering overview is given in Section 3.3.1.

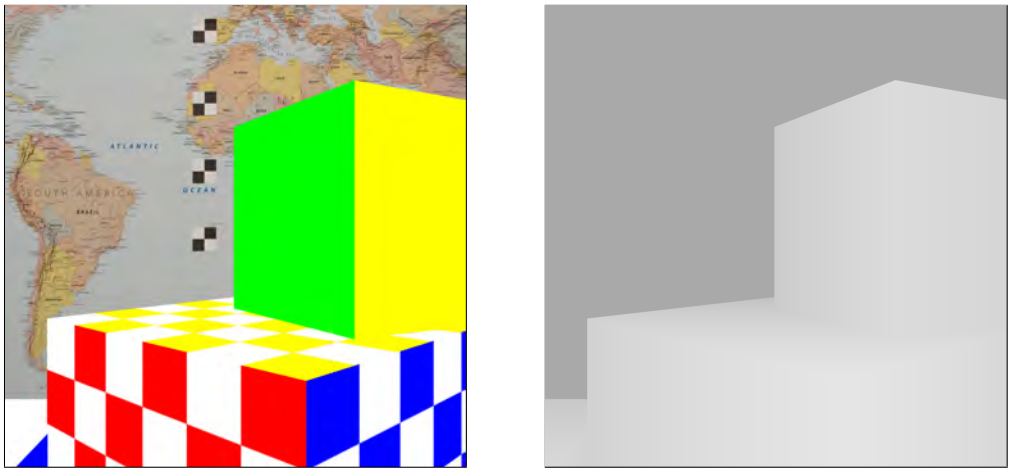
Algorithm 3 Processing Synthetic Depth Information

1. Read and store the depth attachment point.
2. Anti-alias the scene.
3. Linearize the depth buffer.
4. Convolve the depth image.

3.3.1 Scene Precision

In computer graphics, depth buffering is commonly used to manage the depth coordinates of a three-dimensional image. Primarily used as a solution for scene visibility problems, depth buffering decides which rendered primitives should be visible and which should be hidden from view. Decisions are made per-fragment and are based on the distance values contained within the depth buffer. A visual representation of the depth buffer can be seen in Figure 3.4, where the depth values are lighter when closer to the camera and darker when further away.

The OpenGL depth buffer, also referred to as the z-buffer, is a two-dimensional bitplane whose elements represent every pixel on the screen. When a scene is rendered, the depth of a generated pixel is stored in the z-buffer. When another object occupying the same pixel space must be rendered, the depth values are compared, and the closer of the two is given preference. This process, called z-culling, ensures that a closer object will hide a further one. The distance of a given pixel is derived from the z-coordinate system, but it is



(a) Color Buffer (b) Depth Buffer

Fig. 3.4: Depth buffer visualization. (a) Virtual scene created with the *Synthetic Scene Generator*. (b) Depth buffer corresponding to the image in (a) but visualized in gray scale to illustrate the concept of distance, where the monochrome color assignment spans the linear interpolation of white (near clipping plane) to black (far clipping plane).

also influenced by the viewing frustum. The resulting viewport transformation is a depth coordinate clamped to a specified range. The default range for OpenGL is [0, 1]. While this works for most applications, it is worth mentioning that any desired ranged may be specified by the programmer.

As previously mentioned, the viewing frustum (perspective projection matrix) also directly influences pixel range transformations. The near and far clipping planes (*zNear* and *zFar*) represent adjustments to the minimum and maximum values that the OpenGL depth buffer can store. Poor precision of the z-buffer is often a direct result of the values chosen for *zNear* and *zFar*. A lack of precision limits the ability to resolve the distance between two nearby objects and often results in zig-zag anomalies (z-fighting). Ideally, *zNear* and *zFar* should be close together, with *zNear* as far from the eye as tolerable to reduce the flickering that can occur when multiple primitives have similar values in the z-buffer.

An additional caveat regarding depth precision is the non-linear nature of the z-buffer. The values contained within the z-buffer are not spread evenly over distance. Values closer to the camera are more precise than values further away. While a non-linear buffer ensures

that closer objects are rendered properly, it does not accurately represent the way depth data is captured by the Canasta camera at CAIL. Therefore, in order to correctly generate synthetic ladar data, an additional linearization process must first be invoked.

3.3.2 Buffer Linearization

The non-linear nature of the z-buffer promotes additional manipulation before the canvas can be used as a ladar image facsimile. Once formatted, the range values can then be converted into meters using *scene unit conversion*. Fortunately, modifying transformed range values into useful depth measurements is a straight forward process. Equation 3.1 details the eye coordinate (*modelview* matrix) reference conversion that takes place on a per-pixel basis. The graphical illustration of this conversion is shown in Figure 3.5, where the red graph represents the standard z-buffer and the blue graph represents the z-buffer after linearization. Although buffer precision is slightly degraded during this process, well chosen *zNear* and *zFar* values help mitigate losses.

$$Depth = \frac{2.0 * zNear * zFar}{zFar + zNear - (2.0 * Depth - 1.0) * (zFar - zNear)} \tag{3.1}$$

3.3.3 Convolution Kernel

The final step in depth data manipulation involves a convolving average routine, which helps an image appear more realistic and natural. When a diverging beam of light from a ladar sensor strikes the edge of an object, the computed range value will lie somewhere between the nearest and furthest points of reflection. The texel camera currently used by CAIL returns a 64 x 64 depth array sampled from a 1024 x 1024 pixel canvas. Therefore, every depth sample returned is an average of 256 uniformly neighboring pixels. To better mimic real-world sensor sampling, a 16 x 16 operator (kernel) has been used as a linear filter, as shown in Figure 3.6.

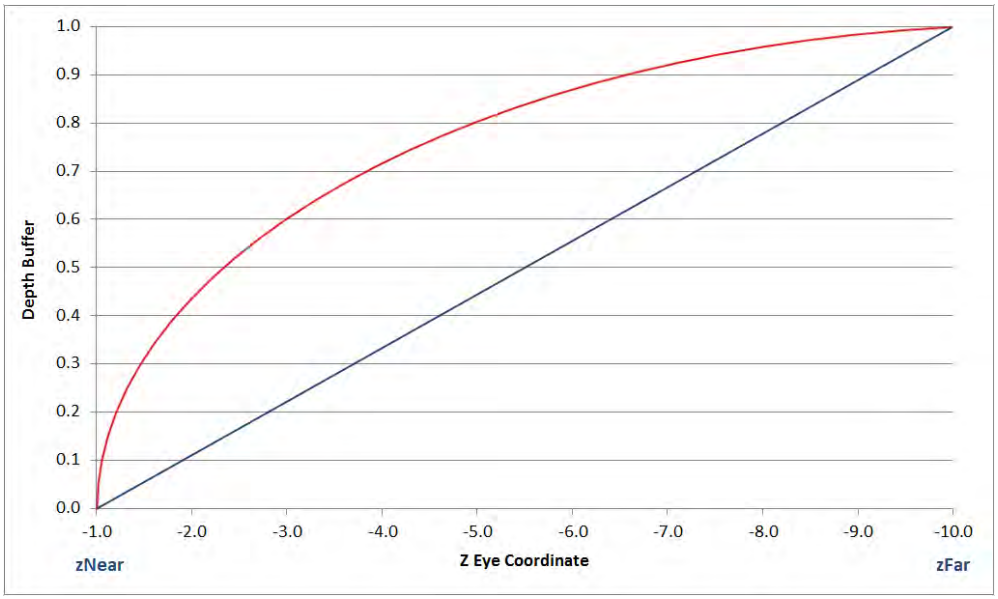
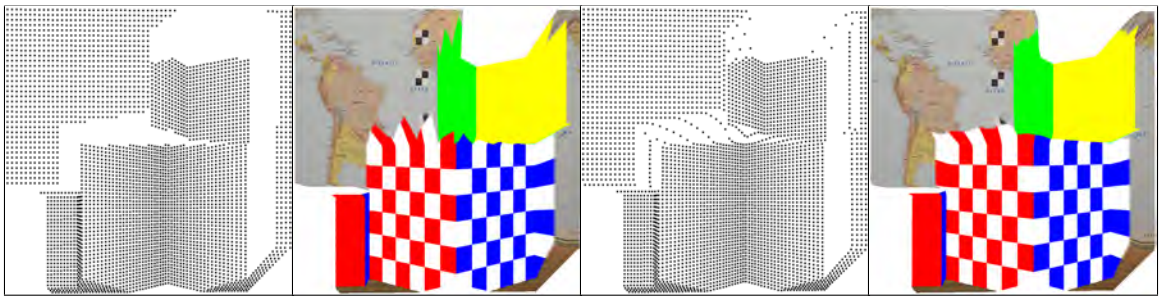


Fig. 3.5: Depth buffer linearization. The z values range from 0.0 to 1.0, spanning across the near and far clipping plane regions. The red graph represents the standard z-buffer, where the non-linear nature of the data ensures that objects closer to the camera have higher levels of precision during z-culling. The blue graph represents the z-buffer after linearization, which allows eye coordinates to be transformed into real-world coordinates using *scene unit conversion*.

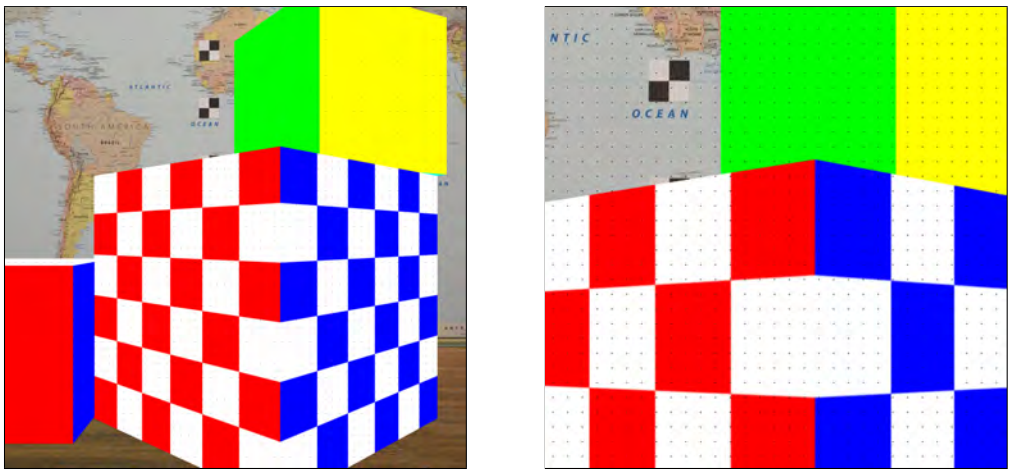


(a) (b) (c) (d)

Fig. 3.6: Convolution average routine. (a) Synthetic depth scene with no alterations. Stair-step effects can be seen along the top edge of the center cube, an artifact of evenly sampled rows being taken along an angled edge. (b) Textured form of (a), which clearly does not resemble a real texel image. (c) Corrected version of (a) using a convolution kernel. (d) Textured from of (c), giving a more natural appearance.

3.4 Image Sampling

Having completed the post-processing stage, the color and depth buffers are now ready to be sampled. Non-integer (sub-pixel access) division of the buffer dimensions (1024 x 1024) by the target point cloud size (64 x 64) produces the appropriate sample set, as shown in Figure 3.7. Sampled depth points are then used to compute the corresponding XY coordinates needed to compose a point cloud. After a normalized XY map is created at 1.0 meter, the map is applied across the entire span of sampled depth components. It should be noted that this process must be tailored to the ladar camera being replicated, as some systems use an *even angles* (uniformly divided FOV) method, while others employ an *even spaces* (uniformly divided distance) strategy during acquisition. Once the final XYZ data has been computed, range information is realized via the Pythagorean Theorem. UV coordinates are also generated, which allow the colored texture image to be applied across the point cloud during rendering. Additionally, row and column index data for the mesh may be exported if desired.



(a) (b)

Fig. 3.7: Division of a 1024 x 1024 canvas into a 64 x 64 sample set. Black pixels represent individual sample point locations within the buffer. (a) Original sample set image. (b) Zoomed portion of (a).

3.5 Model Matrices

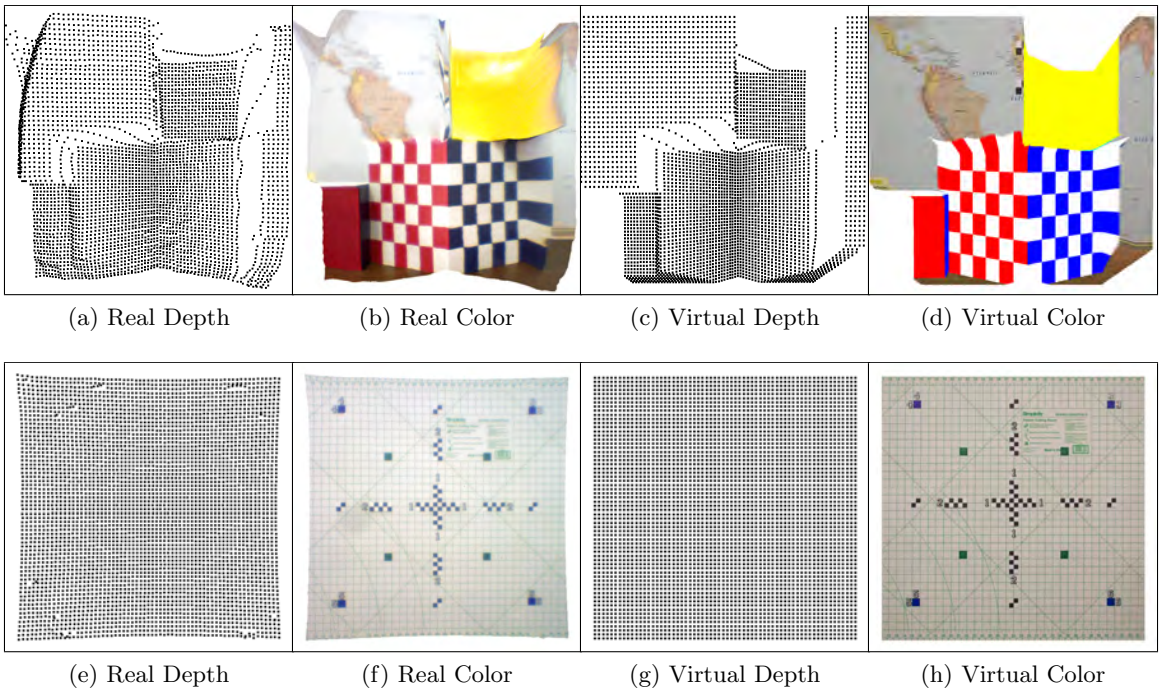
A surface operator matrix is an accumulated transformation matrix generated during texel image registration. When multiple images are matched to form a single 3D model, the unique transformation details for every texel image in the scene are stored in a SOP matrix. As a result, all underlying scans can be transformed from an individual (model) coordinate system to a common (world) coordinate system. Proprietary CAIL software, such as the *3DViewer*, uses SOP matrices to render 3D texel models comprised of independently drawn texel images. When used in this fashion, the SOP matrix of a scan can be thought of as a *model matrix*, which translates and rotates object vertices within the vertex shader.

Unlike the noise and distortion present in real texel image coordinate systems, perfect model matrices are achievable in the virtual world, providing a means for seamless registration. Forming a model matrix with a virtual camera is relatively straightforward, given that the position and orientation of the camera is inherently tracked. The *Synthetic Scene Generator* utilizes such a camera (*view matrix*) during the image acquisition process. The first (base) image captured in the sequence is assigned the identity matrix. All subsequent images taken thereafter are issued model matrices derived from changes in camera position and orientation (relative to the base scan) by taking the inverse of the view matrix. Once complete, the translation components of the model matrix are converted into meters using the unit conversion system of the virtual scene. Note that rotational components, which are independent of coordinate system units, remain unmodified.

3.6 Synthetic Results

Preliminary results are demonstrated in Figure 3.8, where two synthetically generated images are compared to their real-world counterparts. With the absence of noise, the virtual images created are extremely accurate and can be used with a high degree of confidence. Furthermore, accurate registration of multiple texel images is possible, and produces seamless 3D models, as shown in Figure 3.9. The presence of z-fighting seen in Figure 3.9(d) is a key indicator of true registration precision. Even extreme image perspectives can be accurately registered, and while vast distances (meters) between overlapping regions of texture

will eventually reveal discontinuity, these artifacts are a result of heavy interpolation and limited pixel resolution, not registration errors.

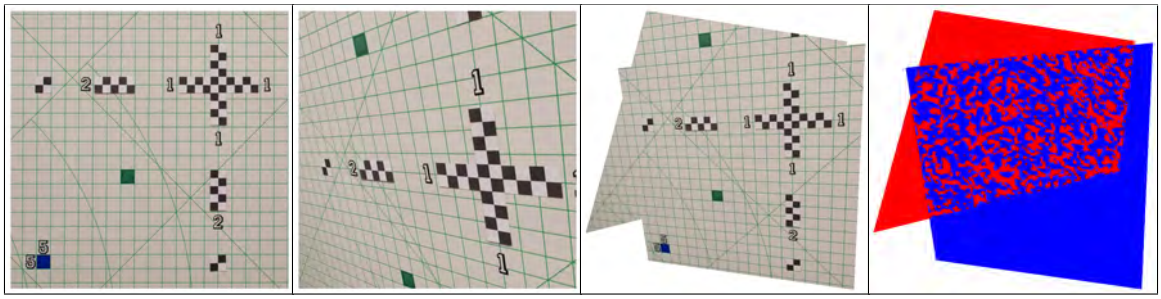


(a) Real Depth (b) Real Color (c) Virtual Depth (d) Virtual Color

(e) Real Depth (f) Real Color (g) Virtual Depth (h) Virtual Color

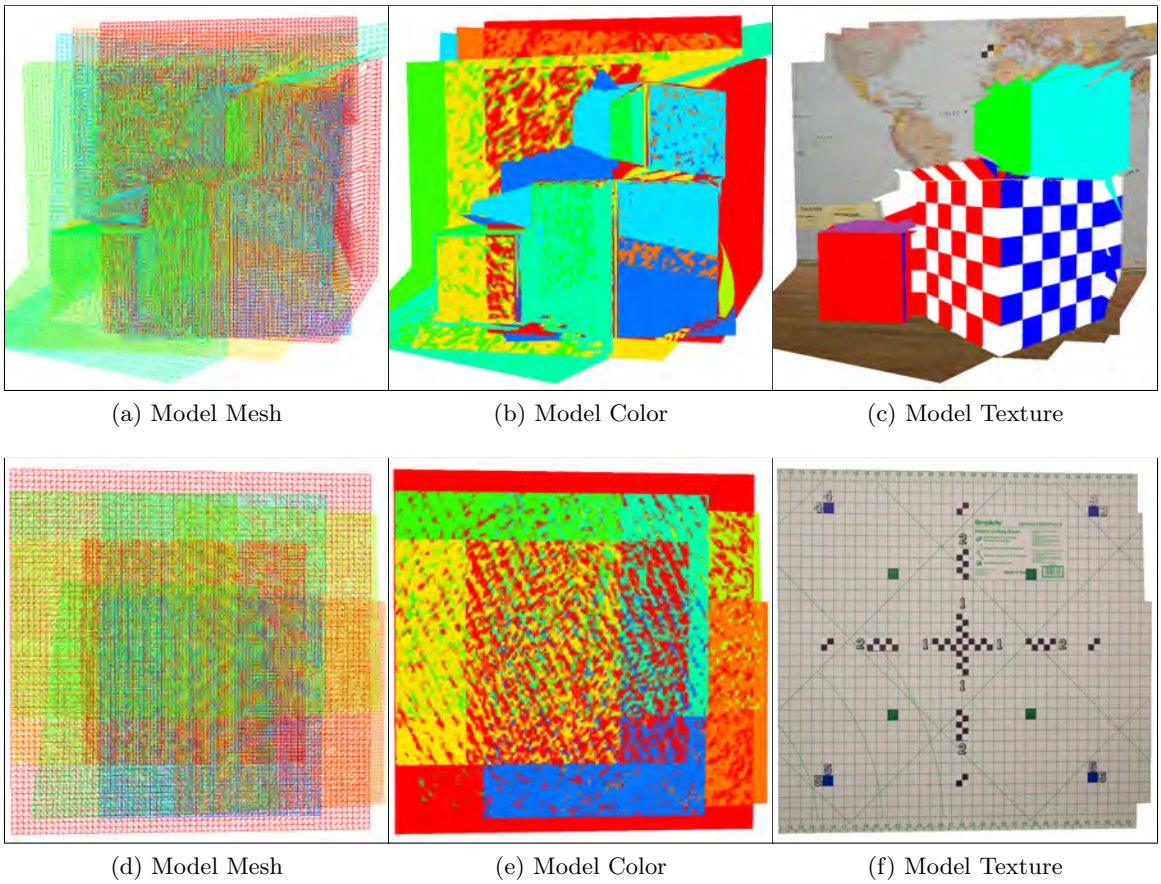
Fig. 3.8: Virtual image comparison. (a) Depth image captured with a Canesta 64 x 64 CMOS TOF ladar sensor. (b) Companion texture image for (a) captured with a Micron 1024 x 1024 EO image sensor. (c)–(d) Synthetically generated versions of (a)–(b) created with the *Synthetic Scene Generator*. (e)–(h) Calibration image comparison given for additional reference.

A final demonstration is given in Figure 3.10, where two sets of ten unique virtual texel images have been registered together to form 3D models. The underlying model images were taken in sequence and cover a wide range of perspectives. The final result is a complete texel model facsimile, certifying that the *Synthetic Scene Generator* is a reliable tool in noiseless data generation.



(a) First Image (b) Second Image (c) Textured Model (d) Colored Model

Fig. 3.9: Virtual registration accuracy. (a) Base scan. (b) Alternate perspective scan. (c) Registered form of (a) and (b), creating a seamless 3D model. (d) The same scene found in (c) but with a unique color assigned to each image rather than a texture.



(a) Model Mesh (b) Model Color (c) Model Texture

(d) Model Mesh (e) Model Color (f) Model Texture

Fig. 3.10: Synthetic 3D texel models. Both models were constructed using ten unique texel images captured from various viewpoints within a virtual scene. (a) Colored model mesh. (b) Colored model layers. (c) Textured 3D model. (d)–(f) 3D calibration texel model.

Chapter 4

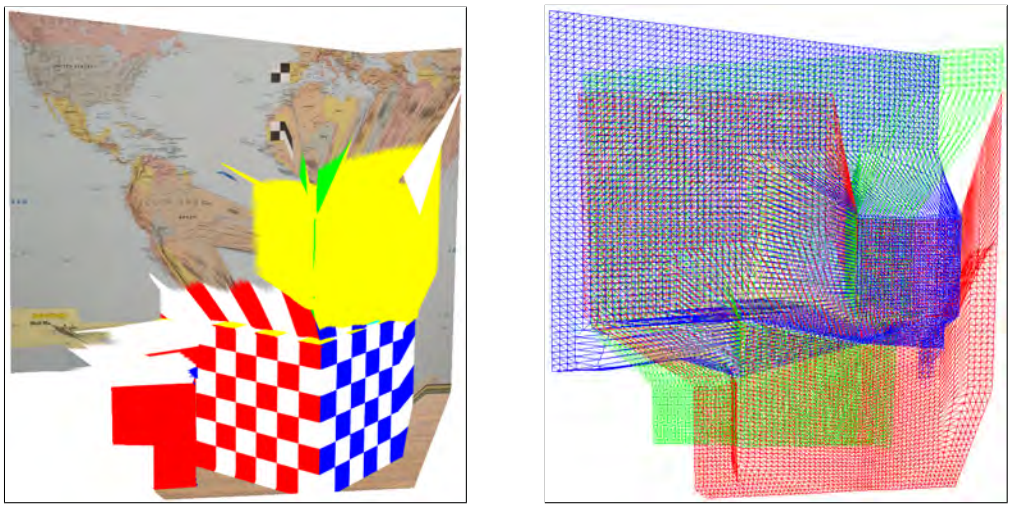
Texel Model Rendering

This chapter explores 3D texel model rendering techniques beyond the standard methods previously discussed. The ability to utilize the best texture from every scan within a model is presented, with the complete rendering procedure being done using OpenGL and GLSL extensions. Section 4.1 begins the chapter with a discussion on the limitations of traditional sorting and rendering methods. An improved technique for hiding, emphasizing, and blending textures using estimated levels of reliability is given in Section 4.2. To concluded the chapter, a proposed 3D texel model rendering algorithm is presented in Section 4.3.

4.1 Sorting Triangles

For a texel model composed of n scans, there may be up to n overlapping triangles for any given texture region. Overlapping textures can quickly become heavily congested, as demonstrated in Figure 4.1. Consequently, corrections must be done after all the triangles have been rendered by determining the most reliable textures for any given area of the model. Texture reliability is computed using a combination of geometric metrics, which allow overlapping textures to be hidden, exposed, or blended according to rank. However, as the number of scans within a model grows, sorting and ranking overlapping textures can become computationally intensive, potentially making real-time rendering difficult.

Fortunately, exploiting the alpha bitmaps of overlapped textures can facilitate real-time rendering, and becomes the main motivation behind triangle sorting and ranking. Alpha blending has low computational overhead and is an inherent feature of most shading languages. Adjusting the alpha component of a texture can render it opaque, invisible, or any combination in between. Using alpha bitmaps to blend object layers within a scene is



(a) Model Texture (b) Model Wireframe

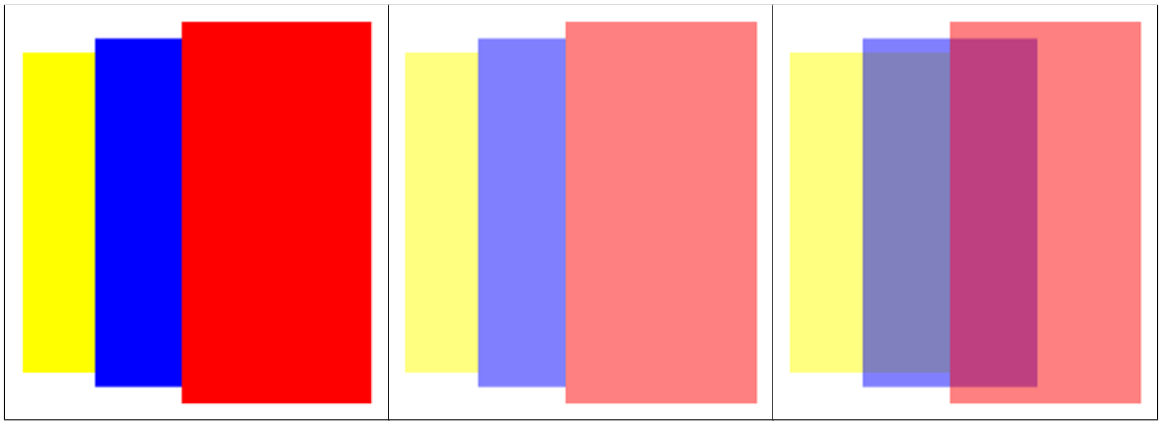
Fig. 4.1: Texture congestion. (a) Synthetic texel model composed of individual texel images. (b) Underlying wireframe architecture for (a), where up to four different textures (each uniquely colored) are found interacting across congested areas of the model.

traditionally done via transparency sorting.

4.1.1 Transparency Sorting

When OpenGL blending is enabled, the renderer will combine (mix) new colors being drawn to the screen with any pixels already present in the framebuffer. As a result, all opaque objects must be drawn before attempting to render translucent ones. Successfully implementing transparency sorting involves rendering object layers in the correct order. The rendering order depends on both the transparency and the depth of the object layer, as illustrated in Figure 4.2. In this example, Figure 4.2(a) contains three colored blocks which have been rendered at different depths. Since the layers are all opaque, the rendering order is irrelevant. However, once the alpha values for these layers begin to change, the need for ordered rendering becomes apparent, as shown comparing Figures 4.2(b)–(c). The difference between these two scenes is a direct result of the default context implementation used by OpenGL. Consequently, a higher level of algorithm complexity is needed to correctly render transparency sorted textures.

In addition to ordered rendering problems, transparency sorting techniques begin to



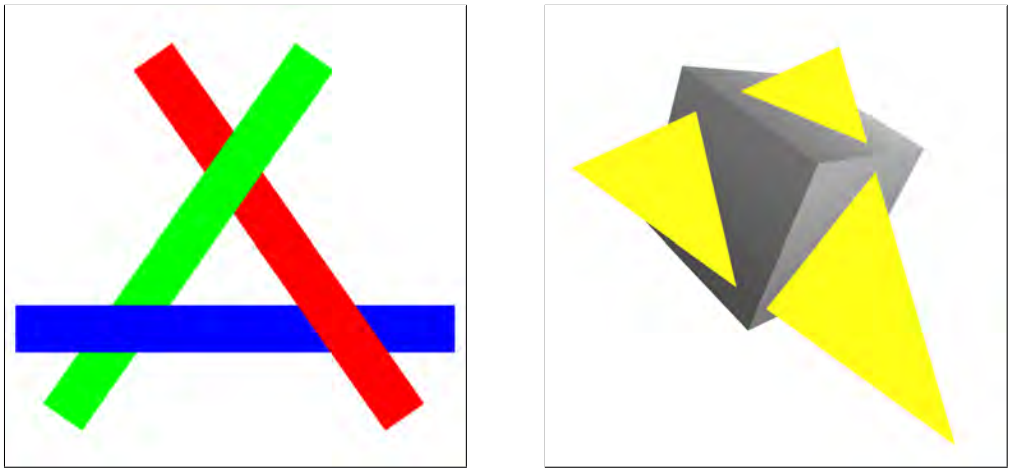
(a) Opaque Blocks (b) Front To Back (c) Back To Front

Fig. 4.2: Ordering principle of transparency sorting. (a) Opaque block objects rendered at different depths. (b) Transparent version of the blocks in (a) rendered *front to back*. (c) Identical scene in (b) rendered *back to front*.

break down with complex object interaction. At the root of these limitations, lies the *cyclic* and *intersecting object* problem cases, shown Figure 4.3. In Figure 4.3(a), the unique overlapping nature of these three objects prevents transparency sorting by hindering proper triangle ordering. All possible rendering orders produce conflicting results. The problem is therefore cyclic, and can only be corrected by *splitting* intersecting areas into additional objects. A similar problem is shown in Figure 4.3(b), where object intersection also prevents reliable transparency sorting and binary space partitioning (BSP) [23] must be used to sort and split static transparent polygons. Unfortunately, implementing a real-time sorting and splitting algorithm for large texel images would be difficult, and the use of these traditional techniques for texel model rendering is discouraged.

4.1.2 Fragment Sorting

The OpenGL Shading Language provides a powerful alternative to that of traditional transparency sorting techniques. Sorting and blending can be done per-fragment (per-pixel) using fragment shaders. Implementing multi-shader pipelines enables real-time processing techniques to be performed on individual texel images and allows for additional model processing on separate rendering passes. Sorting and blending fragments using GLSL is



(a) Cyclic Case (b) Intersecting Case

Fig. 4.3: Classical transparency sorting problem cases.

faster and more efficient than attempting traditional transparency sorting methods on triangle meshes. Utilizing the graphics processing unit to perform fragment level sorting and blending is discussed in detail in Section 4.3.

4.2 Accumulated Texture Reliability

An estimation of accumulated texture reliability for a single texel image, or scan, is done via 3D geometric criteria. The reliability of a texture represents an accumulated rankable value, which is computed using three specific criteria discussed in this section. The ability to determine rank among overlapping textures, based on their estimated reliability values, is a critical step for proper texel model rendering. Not only must these reliability ratings be accurate, but they must also adapt dynamically to satisfy changes in virtual camera perspective. For convenience, the values assigned by these criteria are clamped, falling between the range $[0.0, 1.0]$. A reliability rating of zero represents a discardable fragment (essentially unusable for rendering), while a reliability rating of one represents an ideal fragment. In general, a wide range of reliabilities will be found across the surface of a texel image mesh; however, only a limited number of these will ever be considered perfectly reliable.

The steps for computing the accumulated texture reliability of a single texel image,

viewed from the current virtual camera position, are summarized in Algorithm 4. This process is repeated until all visible scans in a texel model have been rendered. Any scan located on the backside of a 3D texel model should not be processed. Once complete, 2D textures (containing color and reliability information) for every scan in the model will be available for use in the second stage of the rendering pipeline. This process is known as *rendering to texture*. As a result, a complicated 3D sorting problem has been resolved with a simple 2D texture solution. Additionally, transparency sorting and object intersection algorithms will no longer be needed for proper model rendering.

Algorithm 4 Computing texture reliability for a single texel image.

1. Compute the *normal vector accuracy* for each triangle in the mesh. This is done *once* for each texel image in the scan set.
2. Render a single texel image. For each primitive in the image:
   (a) Compute the *point of view confidence*.
   (b) Interpolate *range confidence*.
   (c) Compute *accumulated texture reliability*.
   (d) Interpolate the pixel color.
   (e) Store corresponding pixel color and reliability data for future pipeline processing.

4.2.1 Normal Vector Accuracy

Computing accumulated texture reliability begins with *normal vector accuracy*. Using the normal vector orientation of a triangle face within a texel image mesh is a suitable method for determining accuracy. Triangle accuracies are constant values, and are computed using the original acquisition perspective of a texel image. Normal vectors oriented toward the acquisition center of projection usually correspond to triangles with smaller surface areas, which result in less texture interpolation (i.e. pixelation). Triangles whose normal vectors are rotated away from the frustum viewpoint are usually less accurate and often

correspond to stretched and noisy areas of the mesh. Using the cosine similarity, given by

$$\cos(\theta) = \frac{\mathbf{a} \cdot \mathbf{N}}{\|\mathbf{a}\|\|\mathbf{N}\|}, \tag{4.1}$$

between a triangle normal vector $\mathbf{N}$ and the acquisition COP vector $\mathbf{a}$ provides a suitable measure of accuracy. Figure 4.4 helps illustrate this principle. As the angular difference between these two vectors approaches zero degrees, the computed accuracy draws closer to one. As the angular difference increases, the accuracy level decreases, and eventually reaches zero (90 degree difference). The acquisition COP vector is not constant, which often results in accuracy degradation for triangles located further away from the center of the texel image. Even a perfectly flat surface can only have ideal accuracies when the triangle normal vectors and the acquisition COP vector are identically oriented (the image center). Examples of normal vector accuracy using an assortment of real and synthetic texel images is provided in Figure 4.5. To aid in visualizing the accuracy of the texel image meshes, a heat map has been applied. Cooler colors represent poorly rated triangles and warmer colors indicate triangles with higher accuracy ratings.

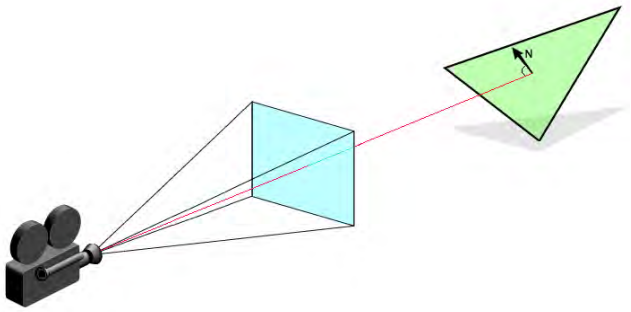
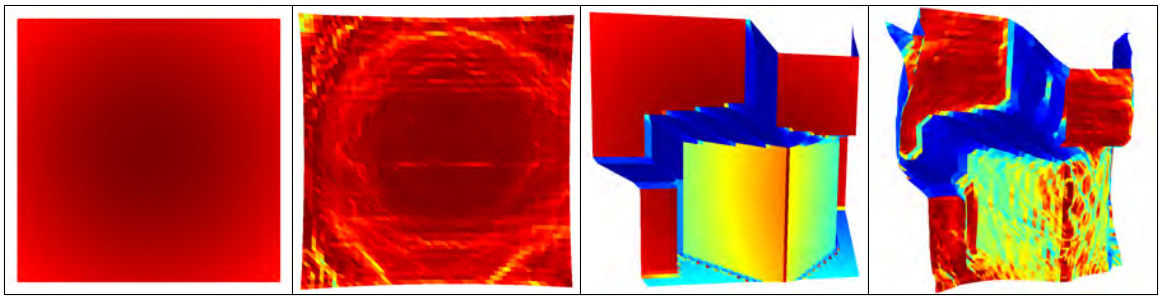


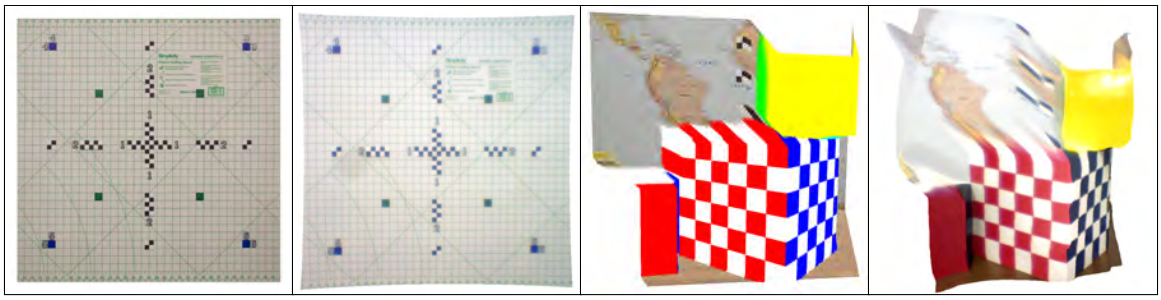
Fig. 4.4: Triangle normal vector accuracy.

4.2.2 POV Confidence

Although normal vector orientation assigns an accuracy value for every triangle in a texel image mesh, this value is constant and only represents the maximum possible accuracy for any given triangle. To account for deviations in virtual perspective, numerical



(a) Synthetic Plane (b) Actual Plane (c) Synthetic Cube (d) Actual Cube



(e) (f) (g) (h)

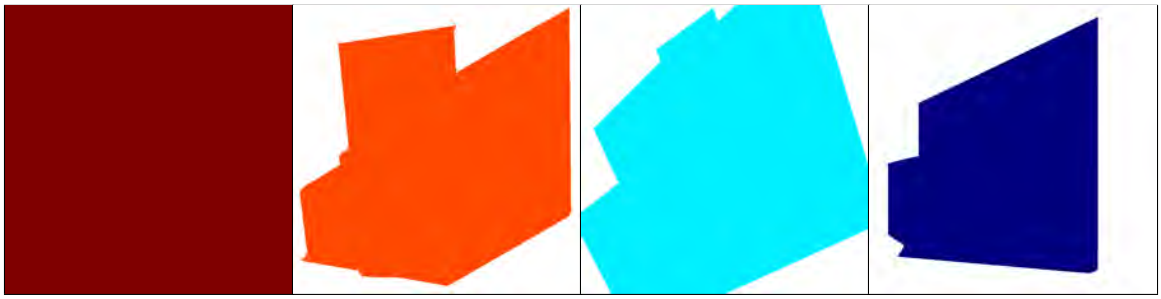
Fig. 4.5: An example of normal vector accuracy is given against a collection of real and synthetic texel images. For reference, the companion texture for (a)–(d) can be seen below each in (e)–(h), respectively.

values known as *confidences* must also be assigned to every texel image triangle. When used together, confidence and accuracy values define triangle reliability when the virtual camera deviates from the acquisition perspective of a texel image. Not surprisingly, as the perspective view of a triangle changes, the confidence of its texture will begin to degrade. Consequently, the most reliable view of any triangle is the view from which it was originally acquired. Any deviation from this acquisition perspective begins to decrease a triangle's confidence, thus changing its overall reliability. Although normal vector accuracy assigns a foundational level of certainty, it can only represent the maximum possible rating for a given triangle within a mesh and is therefore subject to change.

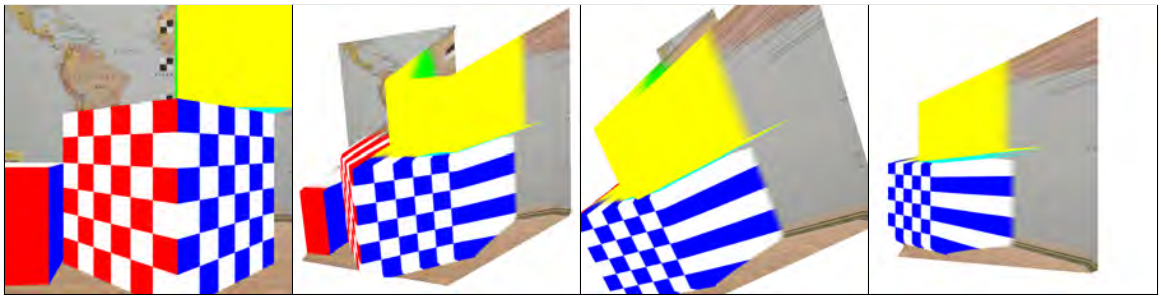
Point of view confidence is the correlation between the current view (virtual camera) and the acquisition view (real camera). When these two perspectives align, the POV confidence of any triangle in the mesh is said to be *ideal*. Not surprisingly, cosine similarity provides a reliable and inexpensive way to calculate these confidence values in real-time. A comparison is done between the acquisition COP vector and the orientation of the triangle mesh (as a whole) from the current point of view (virtual camera). The measure of confidence is computed using the normalized dot product (4.1) between these two vectors. Angle differences greater than 90 degrees are ignored and will not be rendered. An example of POV confidence is demonstrated in Figure 4.6. To aid in visualizing the mesh confidence, a heat map has been applied. Cooler colors represent poor confidence levels while warmer colors indicate meshes with higher ratings.

4.2.3 Range Confidence

Range confidence describes the correlation between the virtual and acquisition distances of a given triangle. The original acquisition distance (meters) is compared with the range, or distance, between rendered mesh triangles and the virtual camera COP. Vertex coordinates are commonly captured in metric units, and therefore, the rendered scene is required to use the same coordinate system for clipping plane placement and camera movement. These confidence values change as the virtual camera zooms closer to a rendered texel image, ensuring that the most reliable textures (less pixelated) are given priority dur-



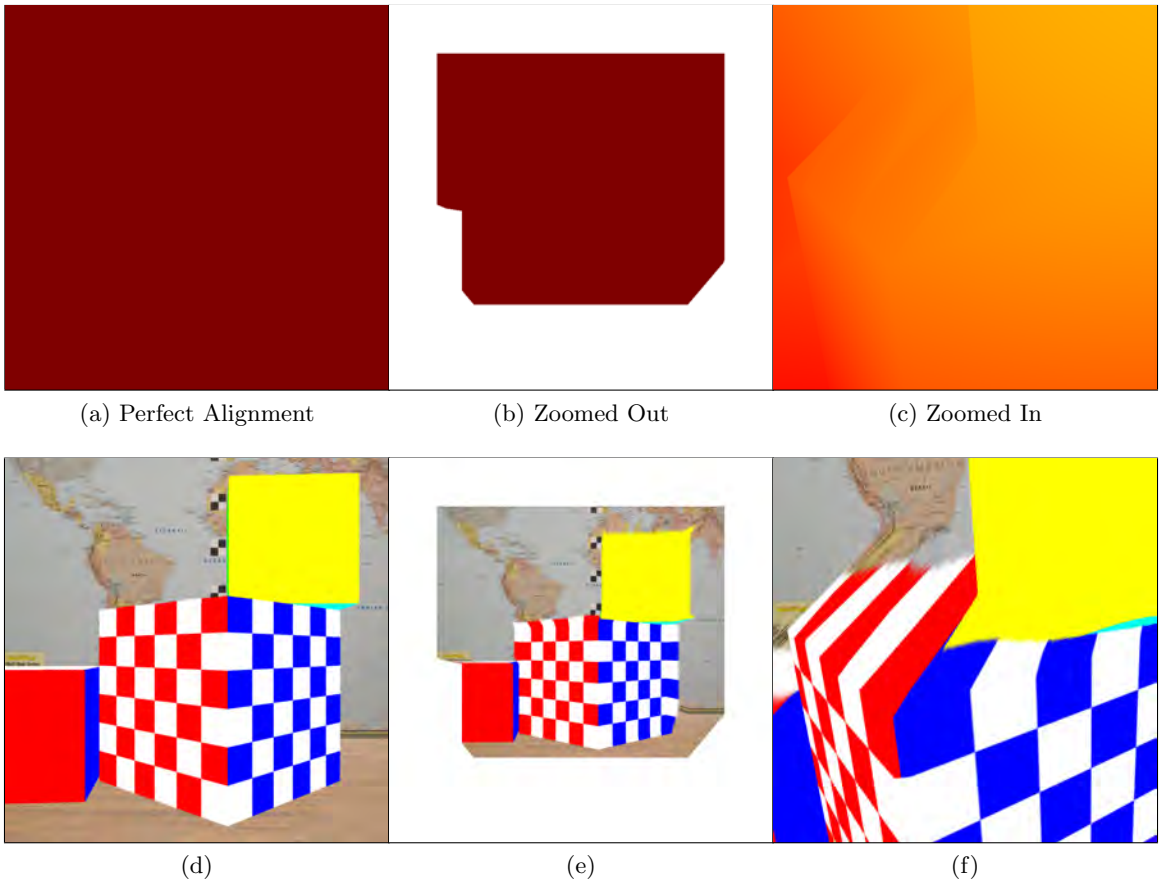
(a) Highest Confidence (b) Moderate Confidence (c) Low Confidence (d) Degraded Confidence



(e) (f) (g) (h)

Fig. 4.6: An example of POV confidence is given against a single synthetic texel image from a variety of perspectives. (a) Virtual and acquisition camera frustums perfectly aligned (highest possible confidence). (b)–(d) Diminishing confidence levels as the virtual perspective continues to deviate from the original acquisition viewpoint. For reference, the companion texture for (a)–(d) can be seen below each in (e)–(h), respectively.

ing render time. Higher confidence levels are given as the virtual and acquisition camera distances begin to align. Lower confidence values are assigned to triangles located beyond the reliable measurement range of the acquisition camera. An example of range confidence is demonstrated in Figure 4.7. To aid in visualizing the mesh confidence, a heat map has been applied. Cooler colors represent poor confidence levels while warmer colors indicate fragments with higher ratings.



(a) Perfect Alignment (b) Zoomed Out (c) Zoomed In

(d) (e) (f)

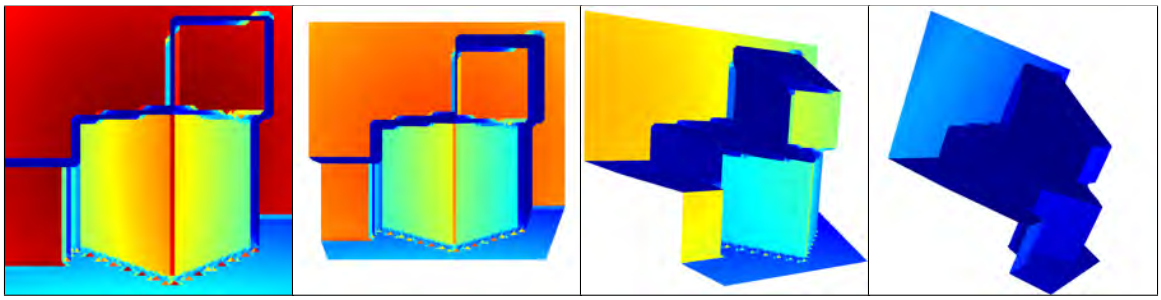
Fig. 4.7: An example of range confidence is given against a single synthetic texel image from a variety of perspectives. (a) Virtual and acquisition camera distances perfectly aligned (highest possible confidence). (b) No change in confidence for images viewed further away. (c) Diminishing confidence levels as the virtual perspective moves closer to the image (pixelation). For reference, the companion texture for (a)–(c) can be seen below each in (d)–(f), respectively.

4.2.4 Triangle Reliability

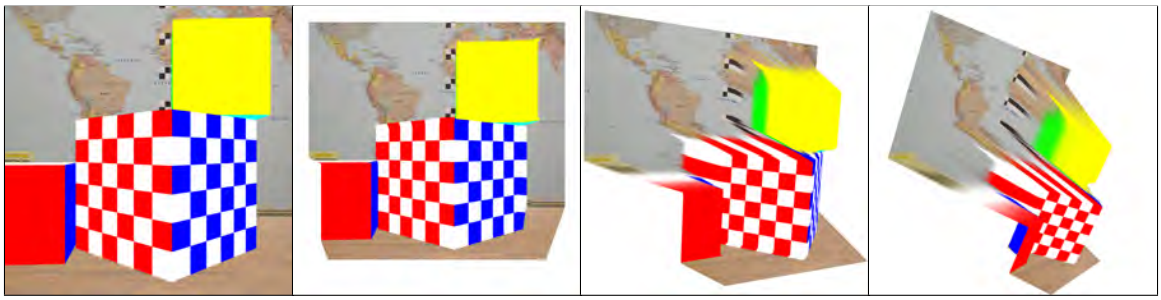
The accumulation of triangle normal vector accuracy, POV confidence, and range confidence constitutes the *accumulated texture reliability* measure of a mesh. The three measures are combined by taking the product of the measures. Accumulated reliability values represent the overall validity of any triangle for a given distance and orientation in virtual 3D space. These values are then interpolated on a per-fragment (pixel) level at render time using GLSL shaders. The accumulated reliability for a fragment is highest when it's virtual and real-world acquisition frustums perfectly coincide. Overlapping fragment areas of a texel model can now be accurately ranked and weighed using their individual reliability ratings. To ease alpha blending and computation, accumulated reliability values are computed as percentages, clamped between $[0.0, 1.0]$. Figure 4.8 illustrates accumulated reliability by showing a synthetically generated texel image from various viewpoints in virtual space. To aid in visualizing the reliability of the texel image mesh, a heat map has been applied. Cooler colors represent poorly rated triangles and warmer colors indicate triangles with higher reliability.

4.3 Rendering Texel Models

Once the individual 2.5D scans of a texel model have been rendered and evaluated for reliability, the resulting 2D texture outputs can be sorted and blended together on a per-pixel basis. For a texel model composed of n scans, there may be up to n overlapping fragments for any given area. An overview of the model rendering pipeline can be seen in Figure 4.9. Fragments which match the rendering background color, or those which have a reliability of zero, will be discarded. Once a list of valid fragments is obtained, they must be sorted according to their accumulated texture reliability ratings. The top ranked fragments are then extracted for processing. These fragments can be blended if they fall within a certain threshold, or simply the best fragment can be rendered. Displaying the best fragment works well with synthetic data, however real-world data benefits from moderate blending, as a result of higher noise levels and misregistration errors.



(a) Acquisition Alignment (b) Translated Perspective (c) Rotated Perspective (d) Degraded Perspective



(e) (f) (g) (h)

Fig. 4.8: An example of accumulated texture reliability is given against a single synthetic texel image shown from a variety of virtual perspectives. (a) Acquisition alignment, which equals the maximum possible reliability for a triangle mesh. (b) Degraded accumulated reliability levels as the virtual perspective is translated back and to the left. (c) The effect on reliability due to rotation. (d) Severely degraded perspective. The perspective is so degraded that triangles with reliability ratings of zero will be ignored by the rendering algorithm. For reference, the companion texture for (a)–(d) can be seen below each in (e)–(h), respectively.

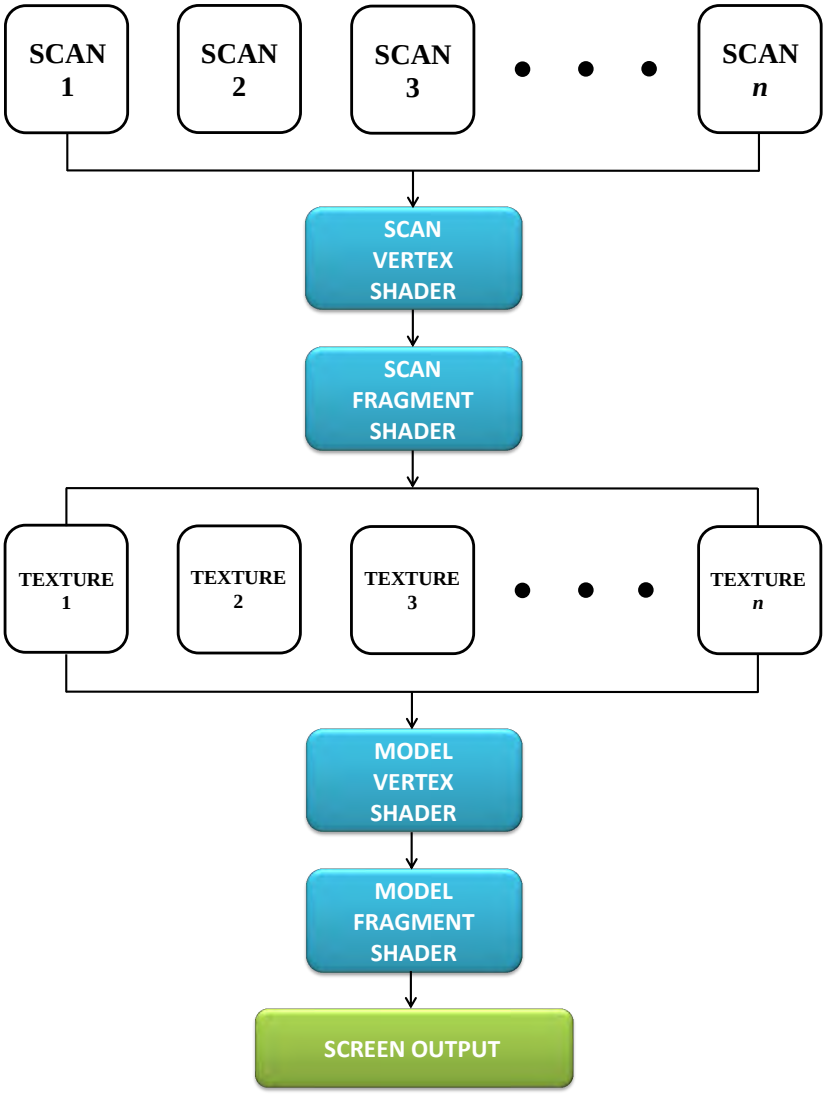
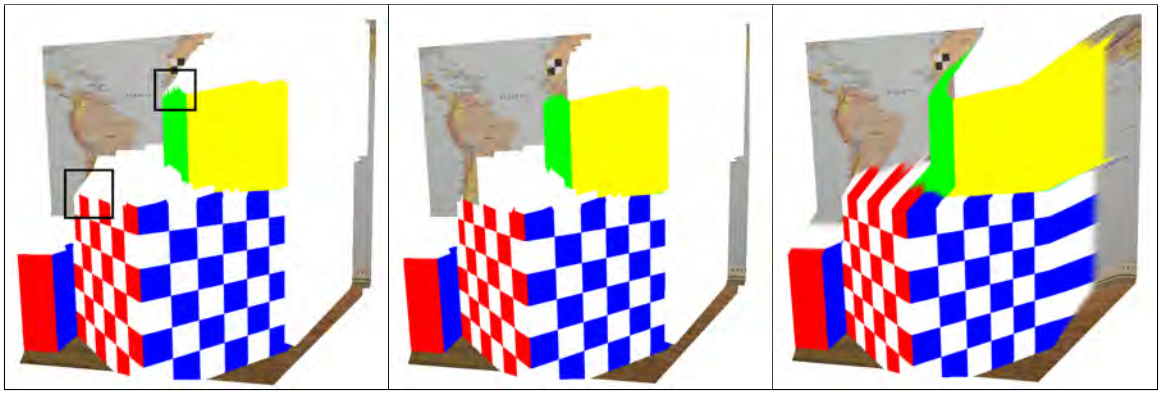


Fig. 4.9: Texel model rendering pipeline.

4.3.1 Scan Shaders

The first stage in the shader pipeline involves rendering every 2.5D texel image individually. The scans are then stored as 2D textures, which will be used form a complete 3D texel model in the second stage of the rendering pipeline. To begin, stretched and unreliable areas of the texel image are identified and removed using 3D geometric criteria. Normal vector accuracy, face surface area, and acute triangle angles usually produce similar results. Once discardable triangles have been identified, they must be properly removed in the fragment shader, as demonstrated in Figure 4.10. The remaining valid triangles are then given a reliability rating using the procedure outlined in Algorithm 4. The per-triangle data needed to compute reliability, including normal vectors and range measurements, are accessed in the fragment shader using the OpenGL *gl_PrimitiveID* keyword. Once finished, the rated triangles are then interpolated on a per-pixel level.



(a) Invalid Discard (b) Valid Discard (c) Original Scan

Fig. 4.10: Proper removal of unusable fragments. (a) Stretched triangles being discarded improperly, causing removed areas to block out valid textures in the background. (b) Valid discard method. (c) Original image given for reference.

The final step in scan processing involves rendering the color, reliability, and any additional fragment information (i.e. virtual range, etc.) of a scan to a 2D texture. These scan textures will then be used in the second stage of the rendering pipeline to sort and blend individual pixels. User defined framebuffer objects can be bound to the output of a GLSL fragment shader, resulting in images being drawn to texture, rather then directly onto the computer screen. For convenience, the alpha component of the image texture is

used to store per-pixel reliability values. For additional reference, a high level overview of framebuffer object *rendering to texture* is given in Figure 4.11. This process is repeated until all *visible* scans in the texel model have been rendered. Note that any scans located on the backside of the model should not be processed.

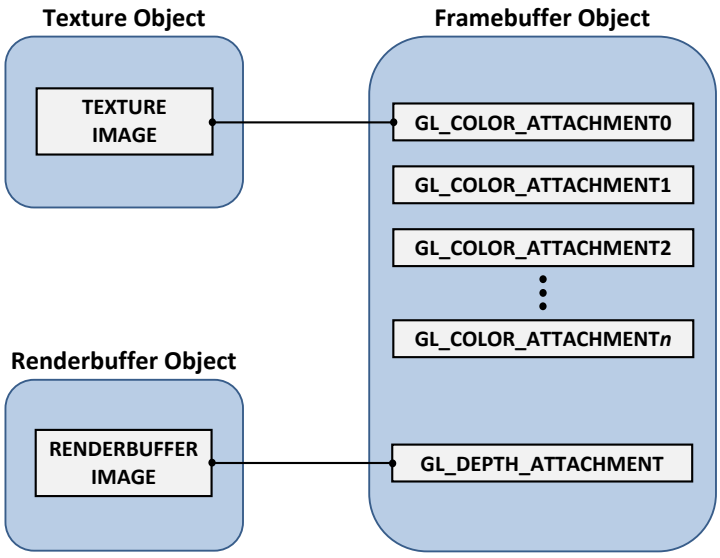


Fig. 4.11: Framebuffer object architecture.

4.3.2 Model Shaders

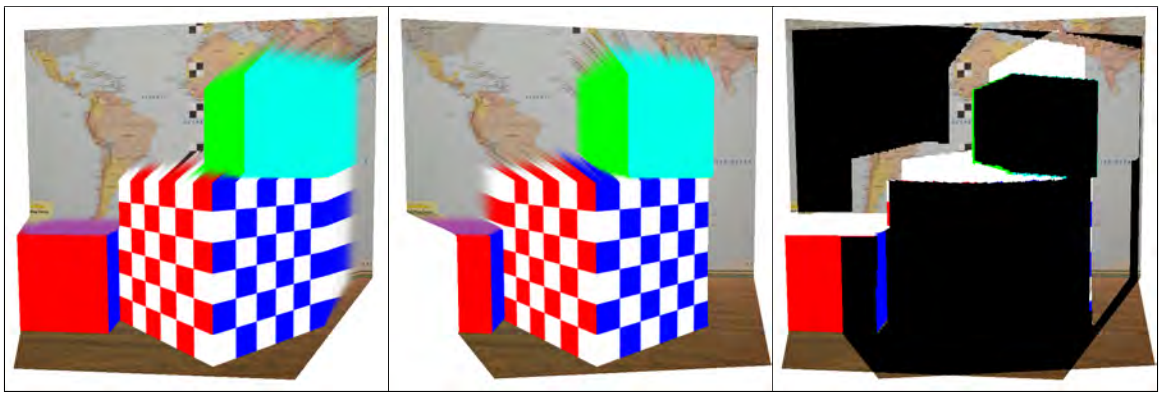
Once the individual scans of a texel model have been rendered, the resulting 2D texture outputs can be sorted and blended on a per-pixel basis, as summarized in Algorithm 5. Figure 4.12(a)–(b) shows two scans which have been separately rendered into textures during the first shader pass of the pipeline. The second stage (or second pass) fragment shader can now access these textures and render them together onto a 2D canvas. As a result, a complicated 3D sorting problem has been resolved with a simple 2D texture stack solution. To illustrate this concept, addition of the textures in Figure 4.12(a)–(b) can be seen in Figure 4.12(c), highlighting the overlapped texture areas targeted for processing.

To begin, invalid fragments are first removed from consideration. Fragments which match the rendering background color or those which have a reliability rating of zero will also be discarded. Once a list of valid fragments is obtained, it is sorted according to

Algorithm 5 Processing overlapped textures for a 3D texel model.

1. Extract valid pixels within a vertical column.
2. Eliminate non-range compliant pixels.
3. Sort remaining pixels according to their accumulated reliability ratings.
4. Process the final pixel group: show best pixel or perform blending.
5. Repeat steps 1-4 for all pixels across the 2D canvas.

reliability. The top ranked fragments are then extracted for processing once they pass a *range compliance* test (4.3.5). The remaining fragments can be blended if they fall within a certain threshold (adjustable percentage) or simply the best fragment can be rendered. This process is then repeated for every column within the 2D texture stack (bit planes) until the entire scene has been rendered.



(a) First Texture (b) Second Texture (c) Combined Textures

Fig. 4.12: Texture stack addition. (a)–(b) Unique texel images which have been separately rendered into 2D textures. (c) Texture addition of (a)–(b) where targeted areas of overlap are now clearly visible (stretched triangles have been removed from consideration).

4.3.3 Best Texture

For texel models with reduced noise, simply displaying the best texture for any given overlapping pixel usually provides an adequate rendering appearance. A visual illustration of real-time, dynamic texture assignment is shown using a synthetically generated data set

in Figure 4.13(a). The model is composed of three individual 2.5D texel images taken from different perspectives, which have been separately rendered into 2D textures. As a result, the final color assignment has been simplified by using a 2D texture stack. The pixels within every column of the texture stack are sorted according to reliability, enabling the best fragments to be brought forward at render time.

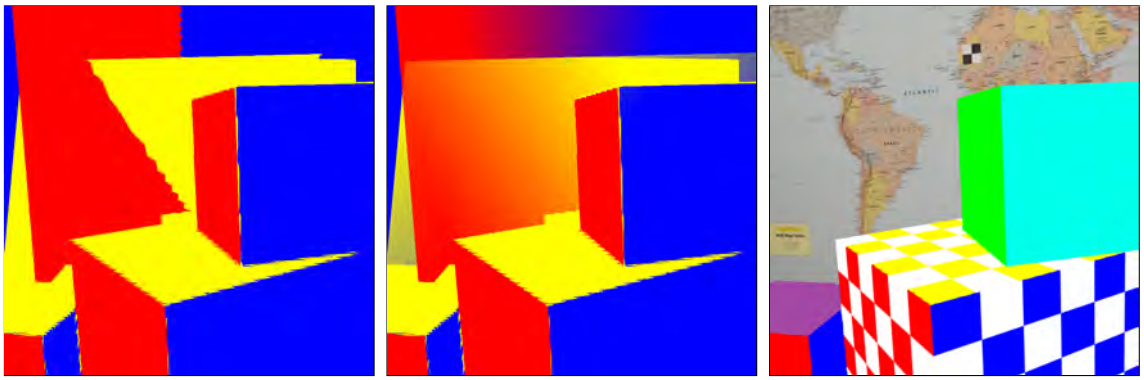
4.3.4 Blended Texture

Texture blending provides an alternative rendering method to the best texture approach. Blending utilizes the alpha component of a texture to designate the strength available for mixing. A weighted arithmetic mean is used to ensure greater influence is given to fragments with higher reliability. As the virtual perspective changes, the weighted texture blend is adjusted accordingly. Texture mixtures, computed from weighted reliability expressions, reduce noise artifacts and help smooth overall model appearance. A visual illustration of blended texture assignment is shown using a synthetically generated data set in Figure 4.13(b). The model is composed of three individual 2.5D texel images, taken from different perspectives, which have been separately rendered into 2D textures. The effect of blending is evident in Figure 4.13(b), which illustrates how the abrupt sharpness in reliability between the three texel images can be blended to reduce artifacts at the boundary.

4.3.5 Range Compliance

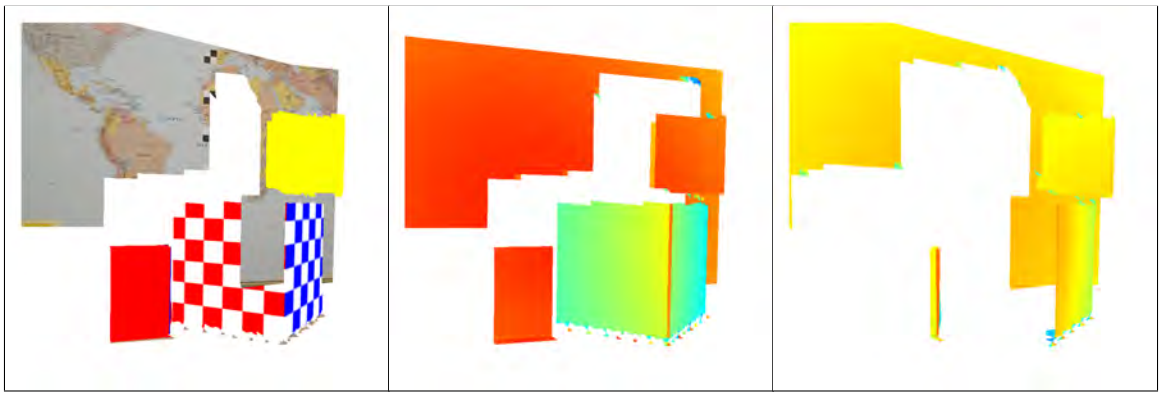
Although using a rendered stack of 2D images simplifies model texturing, doing so eliminates any inherited visibility properties. Hidden surface removal (HSR) is used to determine which surfaces should not be visible from a particular viewpoint. This process is sometimes called hiding and is often a necessary step to render images correctly. Texel model rendering is no exception, as evident in Figure 4.14. In this example, portions of a texel image with higher reliability can be erroneously seen through portions of a texture closer to the virtual camera. Specifically, a small section of the world map in the background can be seen through the checkered cube.

To prevent these windowing artifacts, range compliance must be used when comparing



(a) Best Texture (b) Blended Texture (c) Final Texture

Fig. 4.13: *Best* and *blended* texture assignment using a synthetically generated 3D model created from three texel images. (a) Sections of the image rendered from the best reliability measures in the first image (red), second image (blue), and third image (yellow). (b) Blended reliability. (c) Final rendered image using colors from pixels selected according to the reliability map given in (a).



(a) Texel Model (b) First Image (c) Second Image

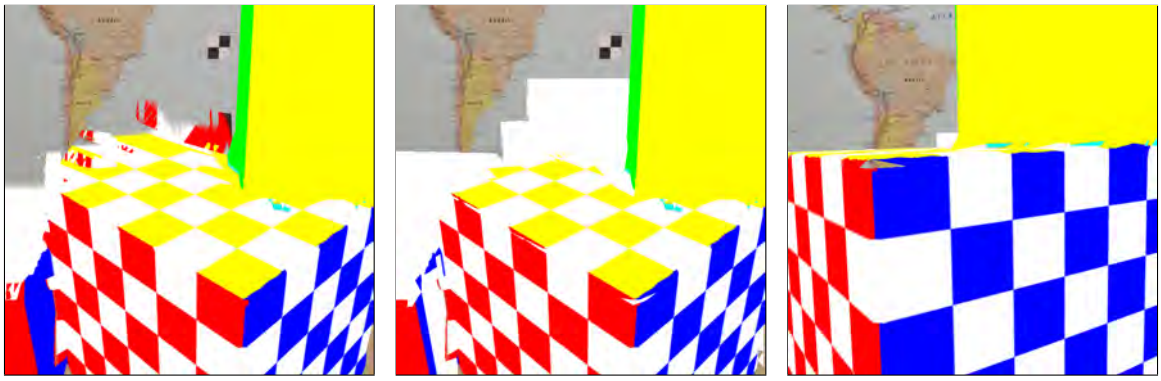
Fig. 4.14: Range compliance violation. (a) Two registered texel images being rendered together to form a 3D model, where portions of the second texel image can erroneously be seen through portions of the first. (b)–(c) Texture reliability heat maps for the two images in (a). Notice that image (c) has a slightly better reliability rating behind the checkered cube and begins to incorrectly overtake portions of (b).

overlapping textures. Verifying range compliance ensures that groups of fragments will only be considered for processing if they are within a reasonable distance from each other (i.e. they are range compliant). Assessing range compliance is done by comparing the virtual distances between all overlapping fragments and the virtual camera COP. The distance threshold is taken from acquisition camera noise characteristics. Any fragments located beyond two times the standard noise deviation should be ignored. Checking for range compliance ensures that only valid overlapping textures can be ranked, sorted, blended, and displayed.

4.3.6 Discontinuities

Stretched triangles continue to present numerous challenges when rendering 3D texel models. These areas are only valid when the virtual and acquisition camera perspectives perfectly conform, but rapidly degrade with any minor viewpoint deviation. As such, these textures have been deemed essentially *unusable* with the currently implemented rendering algorithm, and are removed before overlapped texture comparisons are performed. Stretched triangles are problematic because they often erroneously pass *range compliance* checks and are therefore falsely considered for ranking and blending. Although removing these areas improves overall algorithm performance, it occasionally introduces unexpected discontinuities within the model. This is especially noticeable around sharp and abrupt edges, as shown in Figure 4.15. These discontinuities are a direct result of inadequate sampling around abrupt edges. Unfortunately, these problems cannot be reasonably fixed with the current rendering algorithm being used. Techniques beyond triangles, such as *splats*, may hold the key to breaking up these stretched triangles, and allow certain parts of them to be hidden and others used.

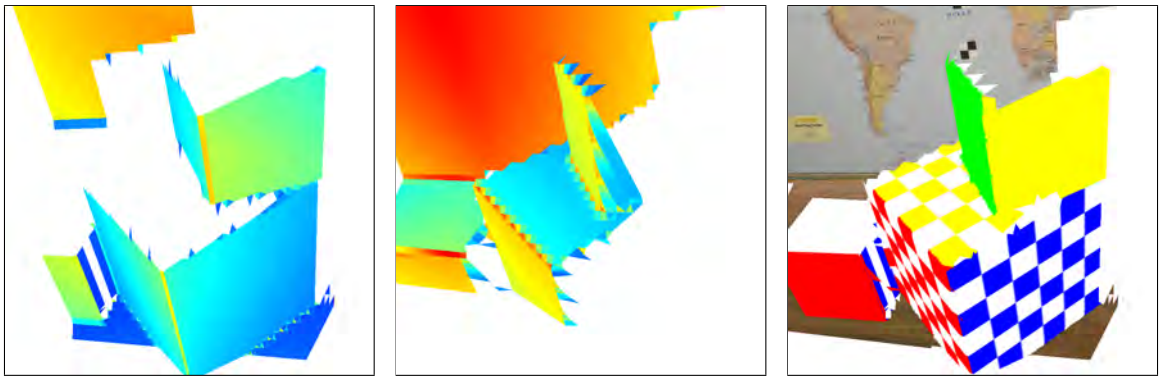
Stretched triangles pose additional problems when working with low resolution ladar sensors, as demonstrated in Figure 4.16. In Figure 4.16(a)–(b), two unique texel images undergo stretched triangle removal using a real-time geometric criteria. Notice however that small portions of these triangles remain along the abrupt side edges of the cube when rendered together in Figure 4.16(c). These small triangles, called *islands*, barely pass the



(a) Default (b) Removed (c) Discontinuities

Fig. 4.15: Stretched triangle discontinuities, where the impact is especially noticeable around sharp and abrupt edges of a model. (a) Rendered model where the stretched triangles have not been removed. (b) Rendered model in (a), where the stretched triangles have been removed. While the overall image appearance has drastically improved, small holes have appeared along the edges of the checkered cube. (c) Alternate view of (b) with discontinuities becoming easily noticed as small portions of the background begin to show through the corner of the cube.

stretched triangle elimination threshold, and therefore go unaltered. As virtual perspective changes, the erroneous reliabilities of these triangles often interferes with the ability to properly render texel models. Removing islands via a nearest neighbor algorithm such as constrained Delaunay triangulation improves algorithm performance. Integrating such a method however should be done during mesh construction (tessellation), before rendering even occurs. Future work regarding mesh analysis and stretched triangle removal warrants further investigation.



(a) First Image (b) Second Image (c) Merged Model

Fig. 4.16: Island discontinuities, where small triangles barely pass the stretched triangle elimination threshold, and therefore go unaltered. (a)–(b) Reliability maps for two unique texel images. (c) Combined rendering of (a)–(b) where merging islands begin to interfere with the ability to properly display the model.

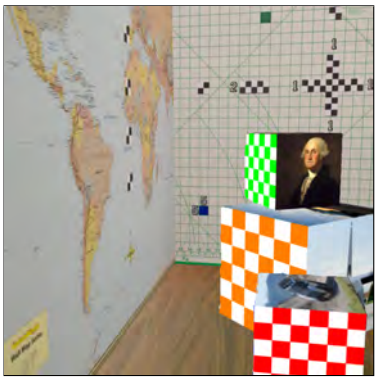
Chapter 5

Results

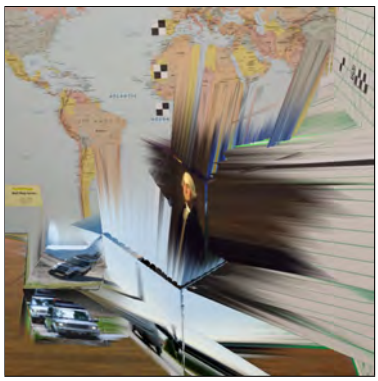
Final algorithm performance is demonstrated using a synthetic data set, shown in Figure 5.1. Real-time, dynamic texturing can be seen as the model is explored from a variety of perspectives. The model shown in Figure 5.1 is composed of 10 different 2.5D texel images, which have been captured from various viewpoints around the scene using a synthetic texel image generator. The advantages of using real-time, dynamic texture assignment can be seen in Figures 5.1(a) and 5.1(b). Although the perspectives of these two images are dramatically different, the model textures have adapted to accommodate virtual camera orientation and weighted texture reliability. For comparison, the original default rendering of these two model perspectives is given in Figures 5.1(c) and (d), respectively. The underlying triangle meshes are also given in Figures 5.1(e) and 5.1(f). As shown from these Figures, the problems with stretched triangles and overlapped textures have been visibly reduced using the method proposed in this thesis.

To further illustrate the concept of weighted reliability, additional examples showcasing *best* and *blended* texture assignments are given in Figure 5.2. As before, the model shown in Figure 5.2 is composed of 10 different 2.5D texel images, which has been captured from three different viewpoints around the scene. For clarity, Figures 5.2(b)–(c), (e)–(f), and (h)–(i) have a unique color assigned to each image, rather than a texture. As the model view changes, the underlying textures can be seen dynamically interacting according to their adjusted reliability ratings.

(a) First Perspective Texture

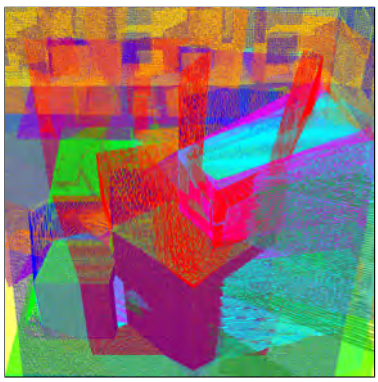


(b) Second Perspective Texture

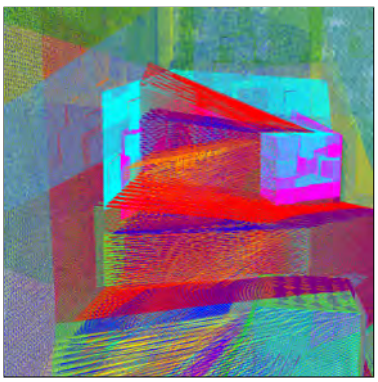


(c) First Perspective Default

(d) Second Perspective Default

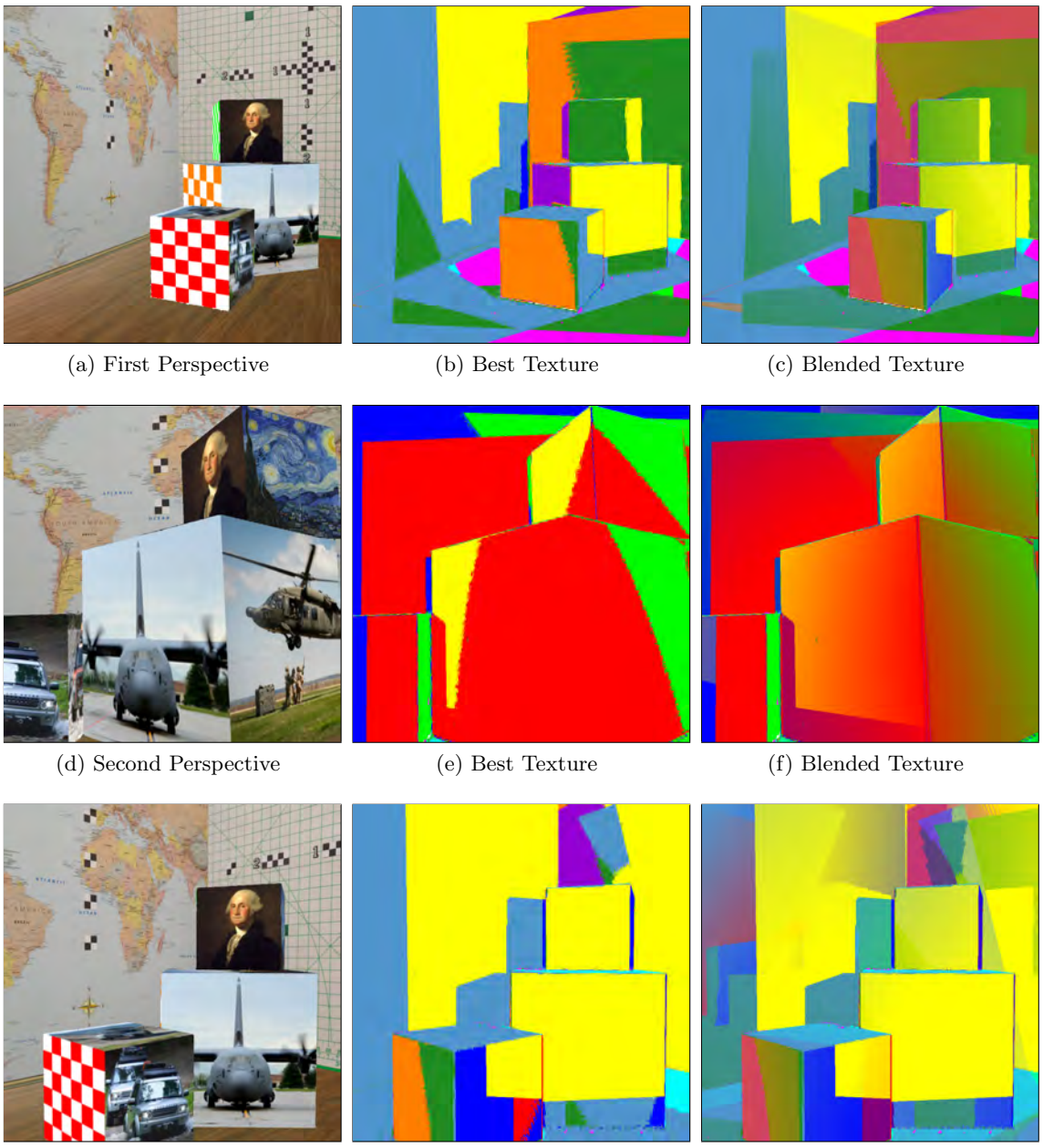


(e) First Perspective Mesh



(f) Second Perspective Mesh

Fig. 5.1: Dynamic texture assignment is shown using a synthetically generated 3D texel model. Figures (a) and (b) are the same scene viewed from two different perspectives. For comparison, the original default rendering of these two model perspectives is given in Figures (c) and (d), respectively. The underlying triangle meshes for these two model perspectives are shown in Figures (e) and (f).



(a) First Perspective (b) Best Texture (c) Blended Texture

(d) Second Perspective (e) Best Texture (f) Blended Texture

(g) Third Perspective (h) Best Texture (i) Blended Texture

Fig. 5.2: Weighted texture assignment is shown using a synthetically generated 3D texel model. Figures (a), (d), and (g) are the same scene viewed from three different perspectives using colors from pixels selected according to the reliability maps given in (c), (f), and (i) respectively. The underlying *best* and *blended* texture assignments (with a unique color assigned to each image, rather than a texture) for (a), (d), and (g) are given in Figures (b)–(c), (e)–(f), and (h)–(i), respectively.

Chapter 6

Conclusion

Standard rendering techniques are not suitable for texturing 3D texel models. Due to the independent nature of the underlying data sets, corrections to model textures must be done at render time. The most reliable texture for a given perspective is better defined using 3D geometric criteria in conjunction with a real-time, view-dependent ranking algorithm. The method described in this paper has shown promise in correctly rendering 3D texel models. Improved results were observed when using higher resolution texel models.

Unfortunately, misregistration errors can lead to poor rendering performance, as demonstrated in Figure 6.1. Although blending techniques help mitigate these problems, blurred and clouded texture areas can appear as a result. In addition, stretched and noisy triangles continue to present numerous challenges when rendering texel models. This is especially obvious when working with sharp and abrupt edges within a model. Future work may investigate solutions beyond triangle-based rendering. Additionally, better blending and alternative image-based processing techniques warrant further investigation.

(a) First Image (b) Second Image (c) Texel Model

Fig. 6.1: Misregistered 3D texel model rendering. (a)–(b) Unique texel images independently captured in the CAIL laboratory. (c) Rendered model of (a)–(b), where registration limitations become quickly apparent.

References

[1] R. Pack, P. Israelsen, and K. Sealy, "A co-boresighted synchronized ladar/eo imager for creating 3D images of dynamic scenes," in *Laser Radar Technology and Applications*, G. Kamerman, Ed., vol. 5791, May 2005, pp. 42–50.

[2] B. Boldt, S. Budge, R. Pack, and P. Israelsen, "A handheld texel camera for acquiring near-instantaneous 3D images," in *Signals, Systems and Computers, 2007. ACSSC 2007. Conference Record of the Forty-First Asilomar Conference on*, Nov 2007, pp. 953–957.

[3] S. Budge, "A calibration-and-error correction method for improved texel (fused ladar/digital camera) images," in *Laser Radar Technology and Applications*, M. Turner and G. Kamerman, Eds., vol. 8379, 2012, pp. 83 790S–83 790S–8.

[4] S. Budge and N. Badamikar, "Automatic registration of multiple texel images (fused lidar/digital imagery) for 3D image creation," in *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, ser. Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, vol. 8731, May 2013.

[5] V. Lempitsky and D. Ivanov, "Seamless mosaicing of image-based texture maps," in *Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on*, June 2007, pp. 1–6.

[6] P. E. Debevec, C. J. Taylor, and J. Malik, "Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach," in *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '96. New York, NY, USA: ACM, 1996, pp. 11–20.

[7] P. Debevec, Y. Yu, and G. Boshokov, "Efficient view-dependent image-based rendering with projective texture-mapping," EECS Department, University of California, Berkeley, Tech. Rep., 1998.

[8] P. Debevec, C. Taylor, and J. Malik, "Modeling and rendering architecture from photographs: A hybrid geometry-and-image-based approach," in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, 1996, pp. 11–20.

[9] C. Buehler, M. Bosse, L. McMillan, S. Gortler, and M. Cohen, "Unstructured lumigraph rendering," in *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, 2001, pp. 425–432.

[10] Y. Alj, G. Boisson, P. Bordes, M. Pressigout, and L. Morin, "Space carving mvd sequences for modeling natural 3D scenes," in *Three-Dimensional Image Processing (3DIP) and Applications*, vol. 8290, 2012, pp. 829 005–829 005–15.

[11] ——, "Multi-texturing 3D models: How to choose the best texture?" in *3D Imaging (IC3D), 2012 International Conference on*, Dec 2012, pp. 1–8.

[12] M. Iiyama, K. Kakusho, and M. Minoh, "Super-resolution texture mapping from multiple view images," in *Pattern Recognition (ICPR), 2010 20th International Conference on*, Aug 2010, pp. 1820–1823.

[13] L. Wang, S. B. Kang, R. Szeliski, and H.-Y. Shum, "Optimal texture map reconstruction from multiple views," in *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, vol. 1, 2001, pp. I–347–I–354 vol.1.

[14] Y. Yu, "Efficient visibility processing for projective texture-mapping," *Computer Science Division, University of California at Berkeley*, 1999.

[15] E. Coppock, D. Nicks, R. Nelson, and S. Schultz, "Real-time creation and dissemination of digital elevation mapping products using total sight flash lidar," in *American Society for Photogrammetry and Remote Sensing*, 2011, pp. 70–78.

[16] N. Badamikar, "Automatic registration of multiple texel images to form a 3-dimensional texel image," Master's thesis, Utah State University, 2014.

[17] R. Pack and F. Pack, "3D multispectral lidar," U.S. Patent Patent 6,664,529, Dec. 16, 2003.

[18] B. Boldt, "Point cloud matching with a handheld texel camera," Master's thesis, Utah State University, 2007.

[19] B. Nelson, "Image-based correction of ladar pointing estimates to improve merging of multiple ladar point clouds," Master's thesis, Utah State University, 2006.

[20] A. Jelalian, *Laser Radar Systems*, ser. Artech House radar library. Artech House, 1992.

[21] Cavemen, "Painter's algorithm," 10,000 B.C.

[22] C. C. Killpack and S. E. Budge, "Visualization of 3D images from multiple texel images created from fused ladar/digital imagery," in *Proc. SPIE*, vol. 9465, 2015.

[23] J. Huang and M. Carter, "Interactive transparency rendering for large cad models," *Visualization and Computer Graphics, IEEE Transactions on*, 2005.