2017

Characterization of multiphase flows integrating X-ray imaging and virtual reality

Timothy Burkgren Morgan
Iowa State University

Characterization of multiphase flows integrating X-ray imaging and virtual reality

by

Timothy Burkgren Morgan

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Co-majors: Human Computer Interaction; Mechanical Engineering

Program of Study Committee:
Theodore J. Heindel, Co-major Professor
Judy M. Vance, Co-major Professor
Hui Hu
Gene S. Takle
Eliot H. Winer

Iowa State University
Ames, Iowa
2017

DEDICATION

To my grandparents, Calvin and Marjorie Burkgren and Leo and Darlene Morgan, thank you for your endless love and support. You instilled in me the value of education, the value of hard work, and the value of having a little fun along the way!

TABLE OF CONTENTS

Page

LIST OF FIGURES

LIST OF TABLES

NOMENCLATURE

Abbreviations

ABS	Acrylonitrile Butadiene Styrene (plastic)
API	Application Programming Interface
ART	Algebraic Reconstruction Technique
ASME	American Society of Mechanical Engineers
CAVE	CAVE Automatic Virtual Environment
CAD	Computer Aided Design
CCD	Charge-Coupled Device
CMOS	Complimentary Metal-Oxide-Semiconductor
COR	Center of Rotation
CPU	Central Processing Unit
CT	Computed Tomography
DTW	Dynamic Time Warping
DVR	Direct Volume Rendering
EBT	Electron Beam Tomography
ECT	Electrical Capacitance Tomography
EIT	Electrical Impedance Tomography
EDT	Engineering Design Team
FAAST	Flexible Action and Articulated Skeleton Toolkit (computer program)
FBP	Filtered Backprojection
FDK	Feldkamp, Davis, Kress (algorithm for cone beam CT reconstruction)
FEDSM	Fluids Engineering Division Summer Meeting (ASME conference)

FPS	Frames Per Second
GLSL	OpenGL Shading Language
GPU	Graphics Processing Unit
HAL	Hardware Abstraction Layer
HMD	Head-Mounted Display
HU	Hounsfield Unit
IVR	Indirect Volume Rendering
KVR	Kinect with Virtual Reality (computer program)
LCD	Liquid Crystal Display
LPM	Liters per Minute
METaL	Multimodal Experience Testbed and Laboratory
MRI	Magnetic Resonance Imaging
NMR	Nuclear Magnetic Resonance
NURBS	Non-Uniform Rational Basis Spline
PIV	Particle Image Velocimetry
Pixel	Picture Element
PTV	Particle Tracking Velocimetry
RF	Radio Frequency
SART	Simultaneous Algebraic Reconstruction Technique
SDK	Software Development Kit
SIMD	Single Instruction, Multiple Data
SNR	Signal to Noise Ratio
ROI	Region of Interest

USB	Universal Serial Bus
Voxel	Volume Element
VRPN	Virtual Reality Peripheral Network
XFloViz	X-ray Flow Visualization Facility
X-Rip	X-ray Image Processor (computer program)
XPTV	X-ray Particle Tracking Velocimetry

Roman Symbols

a	Scaled distance to sampled position in the x-direction (mm)
	Total error estimate of a Kalman filter (-)
	Subscript indicating a specific Kinect skeleton joint, per Table 7.1 (-)
A	Peak amplitude (m)
$\boldsymbol{A}$	Weighting matrix for algebraic reconstruction (-)
A_x	x-direction polynomial coefficient matrix for image unwarping (-)
A_y	y-direction polynomial coefficient matrix for image unwarping (-)
b	Scaled sampling distance in the z-direction (mm)
B_0	External magnetic field strength (T)
B	Bulk CT slice (-)
C	Gas CT slice (-)
d	Source to center distance (mm)
$D(i,j,k)$	Voxel intensity at (i,j,k) (-)
e	Error vector in algebraic reconstruction (-)
E	Photon energy (eV)
E_{max}	Maximum photon energy (eV)

f	Far plane z-coordinate of viewing volume (-)
	Frequency (Hz)
$f(x,y)$	Reconstructed tomographic slice (-)
$f(r,\phi)$	Reconstructed tomographic slice in polar coordinates (-)
$f(x,y,z)$	Reconstructed tomographic volume (-)
F	Kalman filter state transition matrix (-)
$g(t)$	Filter kernel for FBP (-)
$G(i,j,k)$	Gradient at voxel (i,j,k) (-)
G	Algebraically reconstructed slice (-)
$\widehat{G}^{(q)}$	Estimated slice reconstruction at iteration q (-)
i	Voxel position in the x-direction (-)
I	X-ray intensity ($\frac{W}{m^2}$)
	Voxel intensity (-)
	Width of a tomographic slice (-)
I_0	Initial X-ray intensity ($\frac{W}{m^2}$)
I_{ave}	Average pixel intensity (-)
I_b	Voxel intensity, bulk CT (-)
I_{dark}	Dark field pixel intensity (-)
I_f	Voxel intensity, flow CT (-)
I_g	Voxel intensity, gas CT (-)
I_{im}	Uncorrected pixel intensity (-)
I_m	Measured voxel intensity (-)
I_{new}	Normalized pixel intensity (-)

I_{ref}	Baseline voxel intensity (-)
j	Voxel position in the y-direction (-)
J	Height of a tomographic slice (-)
k	Voxel position in the z-direction (-)
	Kalman filter time index (-)
l	X-ray path length (cm)
	Left plane x-coordinate of viewing volume (-)
m	Second statistical moment (-)
M	Number of projections (-)
Max	Maximum possible pixel value (-)
Min	Minimum possible pixel value (-)
n	Number of voxels (-)
	Near plane z-coordinate of viewing volume (-)
	Bit depth of an individual channel of an image (-)
$n_{columns}$	Number of columns in the unwarping grid (-)
n_{rows}	Number of rows in the unwarping grid (-)
N	Number of voxels in one direction (-)
	Number of pixels in one row of a projection (-)
	Number of triangles in a triangle strip (-)
p	Number of voxel chunks (-)
$p(t, \theta)$	X-ray projection at angle θ, detector position *t* (-)
p	X-ray projection vector for algebraic reconstruction (-)
	Three space position (m)

p_a	Position of Kinect skeleton joint *a* (m)
p_{camera}	Virtual camera position (-)
$p_{current}$	Position of the current Kinect skeleton joint in hierarchy (m)
$p_{previous}$	Position of the previous Kinect skeleton joint in hierarchy (m)
p_{target}	Virtual camera target position (-)
P	Kalman filter estimate covariance matrix (-)
r	Radius from the volume center (mm)
	Right plane x-coordinate of viewing volume (-)
	Viewing volume aspect ratio (-)
r_0	Radius from the center of the slice to the feature center (-)
r	Rotation axis vector (-)
R	Kinect skeleton joint orientation matrix (-)
s	Scaling vector (-)
t	Time (s)
	X-ray detector position (-)
	Top plane y-coordinate of viewing volume (-)
Δt	Time between steps *k* and k-1 (s)
T_1	Spin-lattice relaxation time constant (s)
T_2	Spin-spin relaxation time constant (s)
T_2^*	Observed spin-spin relaxation time constant (s)
T	Transformation matrix (-)
u	Original point or vector in homogeneous coordinates (-)
v	Three-space vector (-)

v_{up}	Up direction vector (-)
w	Transformed point or vector in homogeneous coordinates (-)
w_k	Kalman filter process noise at time step k (-)
x	x-position in a computed tomography reconstruction (-)
	x-position of a Kinect skeleton joint (m)
Δx	Distance between sampled points in the x-direction (-)
$\dot{x}$	x-direction velocity of a Kinect skeleton joint ($\frac{m}{s}$)
$\ddot{x}$	x-direction acceleration of a Kinect skeleton joint ($\frac{m}{s^2}$)
x_c	Corrected x-position (-)
x_{cen}	Center x-position (-)
x_d	Distorted x-position (-)
x_{max}	Maximum x-position (-)
x_{min}	Minimum x-position (-)
x_{scale}	Viewing volume x-axis scaling factor (-)
x_{space}	On center distance between clusters in the x-direction (-)
$\mathbf{x}$	x-axis vector (-)
$\mathbf{x}_a$	x-axis vector of Kinect skeleton joint *a* (-)
$\mathbf{x}'$	x-axis vector in the Kinect coordinate space (-)
$\mathbf{x}'_a$	x-axis vector in the Kinect coordinate space of skeleton joint *a* (-)
x_k	Kalman filter state estimate vector at time step k (-)
y	y-position in a computed tomography reconstruction (-)
	y-position of a Kinect skeleton joint (m)
Δy	Distance between sampled points in the y-direction (-)

$\dot{y}$	y-direction velocity of a Kinect skeleton joint ($\frac{m}{s}$)
$\ddot{y}$	y-direction acceleration of a Kinect skeleton joint ($\frac{m}{s^2}$)
y_c	Corrected y-position (-)
y_{cen}	Center y-position (-)
y_d	Distorted y-position (-)
y_{max}	Maximum y-position (-)
y_{min}	Minimum y-position (-)
y_{scale}	Viewing volume y-axis scaling factor (-)
y_{space}	On center distance between clusters in the y-direction (-)
y	y-axis vector (-)
y_a	y-axis vector of Kinect skeleton joint *a* (-)
z	z-position in a computed tomography reconstruction (-)
	z-position of a Kinect skeleton joint (m)
Δz	Distance between sampled points in the z-direction (-)
$\dot{z}$	z-direction velocity of a Kinect skeleton joint ($\frac{m}{s}$)
$\ddot{z}$	z-direction acceleration of a Kinect skeleton joint ($\frac{m}{s^2}$)
z	z-axis vector (-)
z_a	z-axis vector of Kinect skeleton joint *a* (-)
Z	Atomic number (-)

Greek Symbols

α	Virtual camera field of view (degrees)
α_i	Distance from the center of the i^{th} unknown projection to the feature center (-)

α_p	Distance from the center of the known projection *p* to the feature center (-)
$\Delta\alpha$	Shift from a known projection to a calculated projection in the x-direction (-)
γ	Gyromagnetic ratio ($\frac{\text{radians}}{\text{s*T}}$)
ε	Void fraction (-)
ε_b	Bulk void fraction (-)
θ	Projection angle (degrees)
	Angle of rotation transform (degrees)
θ_a	Angle of Kinect skeleton joint *a* (degrees)
θ_i	Angle of the i^{th} unknown projection (degrees)
θ_p	Angle of the known projection *p* (degrees)
$\lambda^{(q)}$	Relaxation factor at iteration *q* (-)
μ	Arithmetic mean (-)
μ'	Arithmetic mean of a portion of a data set (-)
$\frac{\mu}{\rho}$	X-ray mass attenuation coefficient ($\frac{\text{cm}^2}{\text{g}}$)
ρ	Density ($\frac{\text{g}}{\text{cm}^3}$)
ρ_b	Bulk density ($\frac{\text{g}}{\text{cm}^3}$)
ρ_p	True particle density ($\frac{\text{g}}{\text{cm}^3}$)
σ	Sample standard deviation (-)
σ_{ob}	Kalman filter observation standard deviation (m)
σ_{pr}	Kalman filter process standard deviation ($\frac{\text{m}}{\text{s}^3}$)
ϕ	Angle from the x-axis within the slice (degrees)
ω_0	Larmor angular frequency ($\frac{\text{radians}}{\text{s}}$)

ACKNOWLEDGEMENTS

The completion of this dissertation would not have been possible without a huge number of people who provided their assistance, support, and friendship along the way. While I would like to thank each one individually, space will not permit it. So, to all my family, friends, peers, and coworkers, to all the members of my committee, and to all the faculty and staff of the Mechanical Engineering and Human Computer Interaction Departments at Iowa State University, THANK YOU!

Specifically, I would like to thank my co-major professors, Dr. Theodore Heindel and Dr. Judy Vance. Without all your advice, wisdom, support, and patience, I would never have been able to make it through the Ph.D. program. I would also like to thank the members of my program of study committee, Dr. Hui Hu, Dr. Gene Takle, and Dr. Eliot Winer. Your feedback, support, and advice has been invaluable.

I would also like to thank my parents, Ron Morgan and Marta Burkgren, for their love and support at every turn. And my sister, Meridith Burkgren Morgan, for her love and support, and for making sure my ego never got too big.

Additionally, the following people provided assistance with portions of this dissertation, and deserve specific recognition.

- CHAPTER 4: I would like to thank Ben Halls for his collaboration on testing out the high-speed camera in the X-ray system. Also, I would like to thank Dr. Terrence Meyer, for letting us put his high-speed camera in a high radiation area, not knowing what effects it might have on the camera.

- CHAPTER 5: I would like to thank Laurel Barnet, Ben Engebrecht, Anne Gustafson, and Emily Whitemarsh for their assistance in collecting and processing the data used in this study.
- CHAPTER 6: I would like to thank Laurel Barnet, Thomas Burtnett, Xi Chen, and Anna Riesen for their assistance in design and building the X-ray phantoms used in this study, as well as their assistance with collecting the data used in this study.
- CHAPTER 7: I would like to thank Diana Jarrell, Patrick Carlson and Ryan Pavlik for their many fruitful discussions about the implementation of this project and their work recompiling VRJuggler to support the VRPN text device.
- CHAPTER 8: I would like to thank Leif Berg and Patrick Carlson for their many discussions about how to implement this system and the usability of different interaction methods. I would also like to thank Thomas Burtnett for his assistance implementing the system.

Finally, I would like to thank the many groups that provided support for this research. Portions of this work and my Ph.D. studies were supported by the Bergles Professorship in Thermal Science, the Joseph C. and Elizabeth A. Anderlik Professorship in Engineering, the Office of Naval Research (Dr. Knox Millsap, Program Manager), the Army Research Office (Dr. Ralph Anthenien, Program Manager), the State of Iowa Power Fund, and Iowa State University. The X-ray facility used in this research was funded by the National Science Foundation under award number CTS-0216367 and Iowa State University.

ABSTRACT

Multiphase flows are used in a wide variety of industries, from energy production to pharmaceutical manufacturing. However, because of the complexity of the flows and difficulty measuring them, it is challenging to characterize the phenomena inside a multiphase flow. To help overcome this challenge, researchers have used numerous types of noninvasive measurement techniques to record the phenomena that occur inside the flow. One technique that has shown much success is X-ray imaging. While capable of high spatial resolutions, X-ray imaging generally has poor temporal resolution.

This research improves the characterization of multiphase flows in three ways. First, an X-ray image intensifier is modified to use a high-speed camera to push the temporal limits of what is possible with current tube source X-ray imaging technology. Using this system, sample flows were imaged at 1000 frames per second without a reduction in spatial resolution. Next, the sensitivity of X-ray computed tomography (CT) measurements to changes in acquisition parameters is analyzed. While in theory CT measurements should be stable over a range of acquisition parameters, previous research has indicated otherwise. The analysis of this sensitivity shows that, while raw CT values are strongly affected by changes to acquisition parameters, if proper calibration techniques are used, acquisition parameters do not significantly influence the results for multiphase flow imaging. Finally, two algorithms are analyzed for their suitability to reconstruct an approximate tomographic slice from only two X-ray projections. These algorithms increase the spatial error in the measurement, as compared to traditional CT; however, they allow for very high temporal resolutions for 3D imaging. The only limit on the speed of this measurement technique is the image intensifier-camera setup, which was shown to be capable of imaging at a rate of at least 1000 FPS.

While advances in measurement techniques for multiphase flows are one part of improving multiphase flow characterization, the challenge extends beyond measurement techniques. For improved measurement techniques to be useful, the data must be accessible to scientists in a way that maximizes the comprehension of the phenomena. To this end, this work also presents a system for using the Microsoft Kinect sensor to provide natural, non-contact interaction with multiphase flow data. Furthermore, this system is constructed so that it is trivial to add natural, non-contact interaction to immersive visualization applications. Therefore, multiple visualization applications can be built that are optimized to specific types of data, but all leverage the same natural interaction. Finally, the research is concluded by proposing a system that integrates the improved X-ray measurements, with the Kinect interaction system, and a CAVE automatic virtual environment (CAVE) to present scientists with the multiphase flow measurements in an intuitive and inherently three-dimensional manner.

CHAPTER 1:

INTRODUCTION

Multiphase flows are used in a wide variety of industries; however, because of the complexity of the flows, difficulty measuring them, and limitations in the visualization of the measurements, it is challenging to characterize the phenomena inside a multiphase flow. To aid in overcoming this challenge, this research combines improvements in noninvasive X-ray measurements with virtual reality to provide a system that scientists can use to naturally and intuitively characterize multiphase flows.

1.1 Motivation

The understanding of multiphase flows is a necessity in a broad range of modern industries. From energy production to pharmaceutical manufacturing, a thorough understanding of the hydrodynamics of the system is required to improve the efficiency and effectiveness of various processes. However, increasing the understanding of fluid flows is a challenging multi-faceted problem involving not only raw data, but also how the data are presented to scientists for interpretation.

Unfortunately, many of the multiphase flows of industrial importance are extremely difficult to measure experimentally. One example of this is the flow that occurs in fluidized beds, which are used to burn biomass in some power plants. In such a system, crushed biomass is added to a bed of hot sand, and air is injected from the bottom causing the granular material to behave as a fluid. However, due to the opaque nature of the sand and biomass, it is impossible to observe the hydrodynamics occurring inside the reactor visually. Point measurements can be taken with probes, but the presence of the probe in the flow can

change the hydrodynamics of the system. In order to obtain better measurements of opaque systems, many noninvasive flow measurement methods have been developed, as summarized in Section 2.1. However, each system has specific limitations. Magnetic resonance imaging and computed tomography, for example, have excellent three-dimensional spatial resolution, but the time required to acquire a data set limits their usage to time-averaged measurements. Other measurement techniques, such as electrical impedance tomography, have excellent temporal resolution, but are severely limited in spatial resolution. A final group of measurement techniques, including visible light particle tracking velocimetry and X-ray particle tracking velocimetry, has good spatial and temporal resolution, but only for a small number of particles, specifically introduced into the flow to aid in measurement. A measurement system that enables the direct measurement of a multiphase flow with high spatial and temporal resolution does not exist yet.

However, obtaining better raw data about a multiphase flow is only part of the problem. A single computed tomography scan can easily generate over 100,000,000 individual data points. For a scientist to effectively characterize the multiphase flow phenomena, this data must be presented in a way that is intuitive and easy to interact with. Currently, many scientists are forced to view flow data in ways that make data rendering easy, instead of ways that make understanding easy. For example, X-ray computed tomography (CT) scans are often viewed as individual slices instead of a full three-dimensional dataset. When looking at individual tomographic slices, it is difficult, even for trained scientists, to understand where phenomena occur in the flow, and what the three-dimensional flow structures look like. With advances in computation and rendering, the technology now exists to render scientific data sets in three-dimensions. Using these advances to display data in a manner that is intuitive to

users will allow the user to focus on understanding the flow instead of the mechanics of visualization.

Finally, while three-dimensional rendering greatly assists in understanding, interaction with the data representation provides scientists with the ability to fully explore the data. In the physical world, it is common to manipulate objects to see how they react to varying conditions, and having the ability to interact with data in the virtual world is equally important. For example, the use of head tracking to update a computer rendering for a user's physical movement has been shown to be more important than stereo displays for user immersion. Rendering data in virtual reality, coupled with the best natural interaction methods available, will allow scientists to interact with the data as if it were a physical object. Because the data are provided to the users in a manner that closely mimics the real world, the users can leverage their previous experiences in the real world to form a mental model of the flow's structure more quickly and more accurately than they could by viewing a static, abstract representation.

1.2 Objectives

This dissertation uses a novel combination of improved noninvasive multiphase flow measurement techniques with natural user interaction in virtual reality to aid in the characterization of multiphase flows. In order to achieve this goal, the research has the following objectives:

1) Increase the frame rate of X-ray stereographic data collection to allow for the accurate three-dimensional, time-varying measurement of high velocity multiphase flow phenomena.

2) Determine the sensitivity of X-ray computed tomography measurements to changes in acquisition parameters and in turn, provide multiphase flow researchers guidance on how to select appropriate acquisition parameters.
3) Improve tomographic reconstruction to allow the generation of time-varying three-dimensional data sets from stereographic X-ray measurements of multiphase flows.
4) Advance user interaction through the development of a natural, intuitive method of interacting with virtual reality, while minimizing user encumbrances that could limit user adoption.
5) Propose a system to combine noninvasive multiphase flow imaging with virtual reality to aid in the characterization of multiphase flows.

1.3 Outline

First, a review of current state of the field and background for this research is presented in Chapter 2. Note this chapter is intended to cover topics that have broad applicability across this research. Topics that are more specific to a single chapter of research will be reviewed in the chapter of relevance. Next, Chapter 3 will cover methods used in this research. Again, this is intended to cover topics with broad applicability, specific methods will be covered in the pertinent chapter. Chapter 4 presents research to test the temporal limits of a tube based X-ray system of measurement and prove that high-speed cameras can be coupled effectively with X-ray image intensifiers. In Chapter 5, a detailed analysis of how X-ray computed tomography measurements respond to changes in acquisition parameters is presented. In Chapter 6, two approximate computed tomography reconstructions are presented that allow for the generation of three-dimensional data from

only two X-ray projections. Because two synchronized X-ray projections can be obtained at very high speeds, this algorithm allows approximate four-dimensional data to be generated.

Chapter 7 shifts focus from the measurement of multiphase flows to the visualization, specifically how to interact with the data. In this chapter, a system for using multiple Microsoft Kinect sensors as an input device for a CAVE automatic virtual environment (CAVE) is presented. Chapter 8 brings the X-rays and the virtual reality together to propose a system for visualizing three-dimensional multiphase flow data in virtual reality. Finally, Chapter 9 closes with overall conclusions and comments on the future opportunities in this research area.

CHAPTER 2:

LITERATURE REVIEW

Improving the characterization of multiphase flow experiments is an inherently multidisciplinary task, which requires a background in fluid mechanics, data processing, computer graphics, and human computer interaction. This section will summarize the current research available in these areas, with a focus on how that research advances multiphase flow understanding. First, the state of the art in noninvasive multiphase flow measurement is reviewed in Section 2.1. Next, Section 2.2 covers the reconstruction algorithms that have been developed for noninvasive imaging using computed tomography. An overview of the techniques available to render the volumetric datasets produced by computed tomography reconstruct are provided in Section 2.3. Finally, Section 2.4 discusses the current techniques for interacting with scientific data in virtual reality. A brief summary of the review is provided in Section 2.5.

2.1 Noninvasive Multiphase Flow Measurement

The accurate measurement of multiphase flows has long posed a great challenge for scientists. Most flows of scientific interest are dynamic, requiring measurement systems to have a high temporal resolution. They also contain features on a number of length scales, requiring measurement systems to image relatively large areas at high spatial resolutions. Perhaps most challenging is that many multiphase flows are opaque to visible light, rendering imaging methods developed for transparent systems useless. Furthermore, any instrumentation that sits inside the flow has the potential to change the flow characteristics. Therefore, the ideal tool to measure multiphase flows must have high temporal and spatial

resolution, work with visibly opaque systems, and be noninvasive to the flow. While there are no measurement techniques available that can meet all these criteria, there are a number of techniques available that meet some of the criteria. Some of the most common are optical techniques, electrical impedance tomography, magnetic resonance imaging, and X-ray imaging, which includes X-ray radiography and X-ray computed tomography. Each of these techniques will be described in the subsequent sections.

2.1.1 Optical Techniques

The most basic methods of measuring multiphase flows are optical techniques. All of the optical techniques use cameras to record the interaction of visible light with the multiphase flow. However, the different optical techniques vary how the light is generated (i.e., the flow may be externally illuminated or the flow may luminesce) and how the raw data are processed. Due to the use of visible light, all optical techniques operate best on optically transparent flows as the transparency of the flow permits the measurement of phenomena inside the flow. Optical techniques may also be used on optically opaque flows; however, they will be limited to measuring only the outer surface of the flow.

The simplest optical technique for multiphase flow measurement is direct imaging. In direct imaging, a flow is illuminated by an external light source and the light the flow reflects is recorded as an image using one or more cameras. Direct imaging is particularly useful in measuring flow structures when the different phases have different optical properties, for example the shape of bubbles in an air-water flow. Another example of direct imaging in multiphase flow research is in binary particle flows. By using four video cameras and controlled lighting, Kingston and Heindel (2014) measured the mixing of two materials at the surface of a binary granular flow with high spatial and temporal resolution. However, in

cases such as this, where the flow is opaque, direct imaging provides no information about what occurs beneath the surface of the flow.

A more advanced technique is particle tracking velocimetry (PTV). In PTV, a flow is seeded with several neutrally-buoyant tracer particles, which have a high visual contrast with the background. One or more cameras are placed around the system to image the flow. By measuring a particle's displacement between consecutive frames and knowing the time between frames, the pathline and velocity of the tracer particle can be determined. With enough particles and enough images, a velocity field for the flow can be generated (Jain et al., 2002; Nishino et al., 1989). However, particle tracking requires an optically transparent system and neutrally buoyant tracer particles. Furthermore, if there are too many tracer particles in the flow, the likelihood of particles occluding each other increases, reducing the ability of individual particles to be tracked from frame to frame.

If the number of tracer particles is increased to the point that it becomes infeasible or impossible to track them each individually, the flow can be measured using particle image velocimetry (PIV). In traditional PIV, a single plane within the system is illuminated with a sheet of laser light. The laser light reflects off the tracer particles into the camera, located perpendicular to the illuminated plane. The movement of the particles can then be measured by taking two consecutive image frames or by using two pulses of the laser to image both positions on one recorded frame. By calculating the correlation between the first and second images, the 2D velocity field for the imaged plane can be found (Adrian, 1991). This system can also be extended to 3D to measure the velocity field within a measured volume (Elsinga et al., 2006). However, like PTV, PIV can only measure a transparent flow.

2.1.2 Electrical Impedance Tomography (EIT)

Another method of multiphase flow imaging is electrical impedance tomography (EIT). Unlike optical techniques, EIT works on flows that are visually opaque. In a typical EIT setup, a series of probes are arranged around the edge of the containment vessel. These probes can be flush with the walls of the vessel, making the system noninvasive. To take a measurement, an electric potential is applied to one probe, and the other probes measure the field they receive. Then the probe is turned off, and the one next to it is excited, and so on until all probes have been energized. From these measurements, the electrical impedance of the flow, in the plane of the probes, can be reconstructed. Typically, the capacitance of the flow is measured (referred to as electrical capacitance tomography, ECT), as most flows of interest are electrical insulators; however, inductance or resistance can also be measured (Chaouki et al., 1997). Because the measurement system contains no moving parts, EIT is capable of measuring flows at high temporal resolutions, over 1000 Hz for each plane (van Ommen and Mudde, 2008). However, the reconstruction of the slices is a very difficult problem because EIT is a soft field measurement technique, meaning that a change in the impedance at one location effects the measurement at every other location. Because of these limitations, EIT has a poor spatial resolution, on the order of 5% of the containment vessel diameter (Dickin et al., 1993).

2.1.3 Magnetic Resonance Imaging (MRI)

Another technique that has been used to image multiphase flows is magnetic resonance imaging (MRI). MRI measures a multiphase flow by detecting the spatial and temporal variations in the quantum spin of atomic nuclei. These spatial variations can be correlated to the concentration of specific isotopes in a flow, for example the distribution of water in an

air-water flow. Advances in MRI systems have also made it possible to directly measure the velocity or acceleration field of a flow.

At an atomic level, an MRI measures the net spin of atomic nuclei. In any atom in which the nucleus contains an odd number of protons and/or neutrons, the atomic nucleus has a net spin (½ in the case of an odd number of protons or neutrons and 1 in the case of both an odd number of protons and an odd number of neutrons). This net spin causes the nucleus to have a very small magnetic field. Without the presence of an external magnetic field, all the nuclei will be aligned at random, and the nuclei's magnetic fields will, on average, cancel each other out. When a flow containing atomic nuclei of net spin is placed in an external magnetic field, the magnetic torque from the net spin will tend to align the atomic nuclei with the magnetic field (Gore et al., 1981). However, due to thermal effects, not all nuclei will align with the field. In the case of ^{1}H, at room temperature, roughly one in a million more nuclei will align with the external field than would be expected without the external magnetic field. The exact number of nuclei that align with the external magnetic field is dependent on the strength of the magnetic field and the temperature of the flow (Bottomley, 1983).

Since so few nuclei align with the external magnetic field, the net magnetic field introduced into the object is extremely small, and thus difficult to measure (Gore et al., 1981). However, as the nuclei align with the external magnetic field they oscillate around the magnetic field. This oscillation is the phenomenon of nuclear magnetic resonance, or NMR (Bottomley, 1983). Due to this oscillation, the nuclei also emit electromagnetic energy at very specific frequencies. The angular frequency of the oscillation, ω_0, is known as the Larmor frequency, and is given by:

$$\omega_0 = \gamma B_0 \tag{2.1}$$

where γ is the gyromagnetic ratio, which is dependent on the nuclei type, and B_0 is the strength of the magnetic field (Fukushima, 1999). As an example, in a 1 T magnet, ^{1}H has a Larmor frequency of 42.57 MHz, which is within the radio frequency (RF) band of the electromagnet spectrum (Bottomley, 1983).

The existence of the NMR phenomenon by itself is not enough to produce measurements. The individual nuclei have random phases, making it impossible to measure the RF signal. To enable measurements, an MRI machine uses a weaker secondary electromagnet to perturb the primary electric field. The secondary magnet generates different pulses to produce different measurements. The most important are the so-called 90° and the 180° pulses, which cause the bulk magnetization of the nuclei to turn 90° or 180° from the primary magnetic field, respectively. Using combinations of these pulses, two properties of the material can be measured: the spin-lattice relaxation time constant (also called the longitudinal or T_1 relaxation time constant) and the spin-spin relaxation time constant (also called the transverse or T_2 relaxation time constant) (Bottomley, 1983). In both cases, the value of the time constant is that of the inverse exponential constant in a first order exponential function (Gore et al., 1981). In the case of T_1, the length of the time constant is based on the time it takes for the nuclei to return to their equilibrium alignment with the primary magnetic field. This value can vary from a few milliseconds to months, depending on the state of matter (in general, fluids have shorter relaxation times due to the greater freedom of motion at the atomic level). The T_2 relaxation time is based on the coherence of the oscillation phase between the nuclei. After a 90° pulse, significantly more nuclei oscillate in phase with each other than at equilibrium. As time passes, the nuclei slowly fall out of phase with each other, canceling out their respective electromagnetic

emissions and reducing the net signal strength detected. The value of T_2 typically varies from a few microseconds to a few minutes (Bottomley, 1983).

Finally, there are two complicating factors MRI machines must overcome to image a material. First, there are always slight variations in the local magnetic field of the primary magnet. Due to these variations, a single excitation pulse will cause the material to emit a pulse with a significantly shorter RF signal (called T_2^*) than the true relaxation time. To cancel out these effects, special sequences of excitation pulses are used (Bottomley, 1983). Second, if only excitation pulses are used in combination with the primary magnet, there is no way to discern which part of the material is causing the signal, and thus an average of the entire volume is measured. To determine the local variation of the response, a magnetic gradient is applied, causing the Larmor frequency of the nuclei to change with respect to their position in the object. Using combinations of gradients in different directions, 2D and 3D datasets can be obtained (Bottomley, 1983)

MRI has several properties that make it useful in the measurement of multiphase flows. First, like electrical impedance tomography, MRI has the capability to image opaque flows. Second, MRI is capable of achieving excellent spatial resolution (sub-millimeter) (Chaouki et al., 1997). Finally, MRI is an extremely flexible imaging modality. By varying the excitation pulses and gradients, it is possible to tag portions of the flow magnetically to monitor its evolution, measure chemical reactions within the flow, or directly measure the velocity or acceleration of the flow (Ehrichs et al., 1995; Fukushima, 1999; Markl et al., 2012).

While MRI is one of the most flexible noninvasive imaging modalities available, it also suffers from several drawbacks. First, the signal to noise ratio in MRI data is approximately

proportional to $B_0^{\frac{7}{4}}$, thus extremely powerful primary magnets are required for MRI (Fukushima, 1999). Due to these powerful magnetic fields, any ferromagnetic material, such as steel valves, must be kept away from the MRI machine. Second, the time required to image in three dimensions is significant. A typical 3D acquisition can take 20 minutes or more (Bottomley, 1983; Markl et al., 2012). Despite this, time-resolved MRI has been achieved in a periodic flow with sub-second resolution, although this is not generalizable to a generic flow (Markl et al., 2012). Additionally, there is ongoing work in methods to accelerate MRI by using special excitation pulse sequences and through the use of multichannel receiver coils (Blaimer et al., 2004; Mansfield, 1977).

2.1.4 X-ray Imaging

Another important tool for making noninvasive measurements of multiphase flows is X-ray imaging. X-rays were originally discovered by Wilhelm Röntgen in 1895 while studying cathode ray tubes (Röntgen, 1896). For this discovery, Röntgen was awarded the first Nobel Prize in Physics in 1901. Since their discovery, X-rays have become an important tool in the imaging of multiphase flows (Heindel, 2011; Rowe and Partridge, 1965; van Ommen and Mudde, 2008). There are two primary forms of X-ray imaging that have been used in multiphase flow research: radiography (Section 2.1.4.3) and computed tomography (Section 2.1.4.5). Radiography is capable of higher temporal resolution than computed tomography; however, computed tomography is capable of making 3D measurements. Furthermore, both methods can be used with two different types of X-ray sources: tube sources (Section 2.1.4.1) and synchrotron sources (Section 2.1.4.2).

2.1.4.1 Tube X-ray Sources

The tube source is the simplest type of X-ray source. Inside an X-ray tube is a vacuum chamber containing an anode and a cathode. When a high voltage (on the order of kilovolts) is applied between the anode and the cathode, electrons are emitted from the cathode and impact the anode. When the electrons impact the anode, the anode emits X-ray photons with energies less than or equal to the electrical potential across the tube.

There are two physical phenomena involved in the production of X-rays in a tube source: bremsstrahlung radiation and characteristic radiation. Bremsstrahlung radiation occurs when the electron (in fact any charged particle, although electrons are by far the most commonly used) travels near an atomic nucleus. Because the electron and the nucleus have opposite electric charges, they will be attracted to each other, causing a deceleration of the electron and bending its path. This deceleration causes the electron to lose kinetic energy and emit a photon with energy proportional to the kinetic energy lost by the electron. Since the electron can lose any amount of energy up to its total kinetic energy, bremsstrahlung radiation produces a wide distribution of X-ray energies (Hsieh, 2009). The closer the electron passes to the nucleus, the stronger the electric field it encounters, and the higher the energy of the photon produced. In the extreme case, the electron directly impacts the nucleus and yields all its energy to the emitted photon.

The second type of radiation emitted from a tube source is characteristic radiation. Characteristic radiation occurs when the free electron collides with one of the inner shell electrons in the target material. When this collision occurs with enough energy, the target electron will be ejected from its orbit, and an electron from an outer shell will move inwards to fill the hole. When this electron moves inward, it emits a photon with the difference in

binding energy between its original shell and its new shell. This means there are a limited number of energies at which photons can be emitted for any given target material, and those energies are characteristic to the material used for the anode. When the X-ray spectrum emitted by a source is graphed, spikes in intensity occur at the energies of the characteristic X-rays (Hsieh, 2009).

A special type of tube source is the flash X-ray source. The primary difference between a standard tube source and a flash X-ray tube is the flash sources typically run at much high power, but for very short periods (on the order of nanoseconds) (Boyer et al., 2005). In any X-ray tube, a large portion of the energy of the electrons is converted to heat in the anode. If too much power is run through the tube, the anode can melt, destroying the source. In flash sources, the power is very high, but they are used only for a short pulse to keep the total energy the anode has to absorb low. However, this typically limits flash sources to a small number of flashes before the source requires a long cooling period.

2.1.4.2 Synchrotron Sources

A synchrotron source is fundamentally different from a tube X-ray source. Synchrotron sources use particle accelerators to generate X-rays from the magnetic bending of relativistic electrons. In a synchrotron source, narrow bunches of electrons are generated in a booster ring and then injected into a large storage ring. Both rings are constructed of large hollow tubes that are maintained at a very hard vacuum to reduce the probability of the electrons impacting matter. The electrons are contained inside the rings using powerful magnetic fields, and radiofrequency generators are used to accelerate the electrons to the desired speed (Smith, 1995). When the electrons' path is bent using the synchrotrons bending magnets, an acceleration is imparted on them, causing the electrons to lose energy in the form of

synchrotron radiation. This is the magnetic equivalent of bremsstrahlung radiation (Bilderback et al., 2005). Because the electrons used in synchrotron sources are moving at near the speed of light, the electrons follow very closely behind the photons they create. As the bending electrons generate more photons, those photons will also follow closely behind the earlier emitted photons. This creates a time-squeezing effect, which greatly amplifies the intensity of the X-rays emitted, but only when the observer is looking nearly straight at the incoming photons (Kim, 1989).

The radiation resulting from synchrotron sources has two important properties. First, it is extremely bright relative to tube sources. This allows for the imaging of fast moving phenomena, such as shockwaves (MacPhee et al., 2002). Second, the radiation from synchrotron source is coherent, allowing for the imaging of objects using the phase of the X-ray radiation instead of the magnitude (Hwu et al., 2002; Lee and Kim, 2005). Finally, it should be noted that, while synchrotron radiation produces a wide spectrum of X-ray energies, its spectrum is often narrowed to nearly a single energy by using a monochromator.

2.1.4.3 Radiography

Irrespective of which type of X-ray source is used, the simplest usage of X-rays for multiphase flow measurement is X-ray radiography. Radiography is the process of taking a traditional X-ray image, the type doctors perform to detect broken bones. This can be thought of as an image of the shadow cast by a semi-transparent object. The opacity of a material measured with X-rays is known as its X-ray attenuation, which is correlated to the materials density and the energy of the incoming X-ray photons. A more rigorous explanation is that a radiograph is an image where the X-ray intensity, I, at each pixel is the

line integral of the object's X-ray attenuation along the path of the X-ray. The attenuation follows the Beer-Lambert law:

$$I = I_0 e^{-\left(\frac{\mu}{\rho}\right)\rho l} \tag{2.2}$$

where I_0 is the initial X-ray intensity, $\frac{\mu}{\rho}$ is the mass attenuation coefficient of the material, ρ is the density of the material, and l is the X-ray path length through the object (Heindel, 2011). This equation assumes a monochromatic X-ray source and a single, homogeneous material. When a heterogeneous object is imaged with a monochromatic source, the final X-ray intensity at the X-ray detector becomes:

$$I = I_0 e^{-\int_L \left(\frac{\mu(z)}{\rho(z)}\right)\rho(z)dz} \tag{2.3}$$

where, $\int_L dz$ is the line integral along the X-ray path, with $\frac{\mu}{\rho}$ and ρ being the same as before, except now they are functions of the location in the X-ray path, instead of constants (Epstein, 2003). When a polychromatic source is used to image a heterogeneous object, the final X-ray intensity is:

$$I = \int_0^{E_{max}} I_0(E) e^{-\int_L \left(\frac{\mu(z,E)}{\rho(z)}\right)\rho(z)dz} \, dE \tag{2.4}$$

where E is the photon energy, and E_{max} is the maximum photon energy emitted by the source (Macovski, 1983). All other variables are the same as before, although the initial intensity and the mass attenuation coefficient are now both functions of photon energy.

The X-ray attenuation of a flow is produced by three physical phenomena: the photoelectric effect, Compton scattering, and pair production. Pair production only occurs at extremely high photon energies, which are beyond what most X-ray sources can produce, and thus the phenomenon will not be covered. At lower X-ray energies (less than 100 keV), the photoelectric effect is the predominant mode of attenuation (Ketcham and Carlson, 2001).

The photoelectric effect was first explained in 1905 by Albert Einstein, and for this work he was awarded the Nobel Prize in Physics in 1921 (Arons and Peppard, 1965; Einstein, 1905; Hsieh, 2009). When a photon is attenuated via the photoelectric effect, the incoming photon has more energy than the binding energy of an inner electron in an atom of the material and in the resulting interaction, the entire energy of the X-ray photon is transferred to the electron. This interaction destroys the X-ray photon and ejects the electron (now referred to as a free electron or photoelectron) from the atom. When an electron from an outer shell moves inward to fill the hole left by the ejected electron, the atom emits a new photon of lower energy than the original photon. However, these emitted photons are typically of such low energy that they are totally attenuated inside the material (Hsieh, 2009). Attenuation due to the photoelectric effect can be particularly useful in identifying different materials as the attenuation it produces is proportional to Z^3, where Z is the atomic number of the element (Hsieh, 2009).

The second mode of X-ray attenuation within a material is Compton scattering, which is the predominate mode of attenuation from roughly 100 keV to 5 MeV. Compton scattering was first explained by Arthur Compton in 1923, work for which he received the 1927 Novel Prize in Physics (Compton, 1923; Hsieh, 2009). Compton scattering occurs when the energy of the incoming photon is significantly higher than the binding energy of the impacted electron. Unlike the photoelectric effect, only some of the photon's energy is lost to the electron in Compton scattering (albeit enough to free the electron from the atom), and the photon is deflected away from its original trajectory. The energy of the photon after the collision is dependent on the angle at which the photon is scattered, with the highest energies at the smallest scattering angles.

2.1.4.4 Radiography Enhancements

While radiography has found common usage in noninvasive multiphase flow imaging, its usefulness is limited because it is ultimately a 2D projection of a 3D object. This loss of information has led to some enhancements to try to improve radiography's usefulness. The first enhancement is stereography. Stereography acquires two or more radiographs from different viewpoints. Just as with visible light, by using two viewpoints to measure the flow, much of the 3D information about the flow can be recovered (Kingston et al., 2014; Morgan and Heindel, 2010).

X-ray imaging can be further enhanced by borrowing velocimetry techniques from visible light imaging. For example, Lee and Kim (2005) have applied PIV techniques to synchrotron images to measure the 2D velocities of blood flows. Combining stereography and velocimetry techniques has shown great promise in the measurement of multiphase flows. Seeger et al. (2001) developed the use of X-ray particle tracking velocimetry (XPTV) with stereoscopic images and applied it to gas-liquid flows. Based on that work, Shimada et al. (2007) have also experimented with XPTV in slurry flows.

2.1.4.5 Computed Tomography (CT)

Another mode of X-ray imaging is X-ray computed tomography (CT). Computed tomography extends the concept of stereography to many viewpoints and adds a reconstruction step to generate a 3D volume in which each point (called a voxel, which stands for volume element) represents the X-ray attenuation of that point in space. The reconstruction step is important to the accuracy of CT scans, and is discussed in Section 2.2. In concept, CT is not limited solely to objects illuminated by X-rays; however, in practice the term computed tomography typically refers to X-ray CT unless otherwise noted.

The concept of CT was first conceived independently by Allen Cormack in the early 1960s, although the mathematical foundation (i.e., the Radon transform) on which CT is based was first described by Johann Radon in 1917 (Cormack, 1963, 1964, Radon, 1917, 1986). In 1967, Godfrey Hounsfield, working independently of Cormack, built the first CT scanner intended to scan humans for signs of disease (Hounsfield, 1976; Hsieh, 2009). After scanners became available for medical use, they soon moved into other scientific endeavors, such as flow imaging. In 1979, Cormack and Hounsfield shared the Nobel Prize in Physiology and Medicine for their work on CT scanners (Hsieh, 2009).

Traditionally, CT scanners are classified into four generations based on the mechanics of how the scanner acquires each projection. It is important to note that all four generations of scanners acquire one slice at a time. If multiple slices are needed to measure the flow of interest, either the flow or the scanner has to be translated and the slice scanning process repeated. In first generation scanners, a single narrow "pencil" beam of X-rays is projected onto a single point detector. The source and detector are then translated together to collect multiple measurements, which together make one projection. The source and detector are then rotated around a common origin and the process of acquiring a projection is repeated. While this provides a parallel X-ray beam, which is advantageous for reconstruction, the process to acquire one scan is very time consuming. Second generation scanners are very similar to first generation scanners, except a very narrow fan shaped X-ray beam is used instead of a "pencil" beam and multiple detectors are used simultaneously. This allows the CT scan to be completed with fewer translations between the rotations. Furthermore, because of the narrow angular spread of the fan beam, second generation scanners still

maintain an X-ray beam that is sufficiently parallel to use a parallel beam reconstruction algorithm (Ketcham and Carlson, 2001).

Third and fourth generation scanners eliminate the translations between rotations by using a wide fan beam. In a third generation scanner (which is the most common style in use today), a single wide X-ray beam is projected onto a wide array of point detectors, which are capable of imaging the entire projection in one shot. The source-detector pair is then rotated around a common center to take the next projection. In modern scanners this rotation can happen quickly, allowing a single slice to be imaged in 0.5 s or less (Hsieh, 2009). Fourth generation scanners have a continuous, 360° array of detectors, which remain stationary while the source rotates to project onto different parts of the array. While fourth generation scanners are capable of self-calibration and higher resolutions than third generation scanners, they are more costly. Both third and fourth generation scanners require a fan beam reconstruction to account for the wide spread angle of the X-ray beam, which is more computationally complex than the parallel beam method used in first and second generation scanners.

First through fourth generation scanners all scan a single slice at a time. Therefore, at 0.5 s per slice, a scanner would still take over four minutes to scan 512 slices, an average scan size. This speed is insufficient for most time varying flows, although time averaged values may be acquired. Several methods have been proposed to reduce the required scan time. One method is to scan many slices at once, known as volume CT scanning. To achieve volume CT scanning, a wide X-ray beam spread is required in two directions, resulting in a cone beam. This again results in a more complex reconstruction step, and causes some reduction in spatial resolution (Ketcham and Carlson, 2001).

Other methods to reduce scan time focus on eliminating the rotating mass of the system. One way this can be done by using multiple stationary source-detector pairs (Mudde, 2011; Wu et al., 2007). Having stationary sources and detectors eliminates all moving parts and creates a system that is capable of extremely high temporal resolutions (2500 Hz). However, the number of projections is limited to the number of source-detector pairs, which significantly reduces the spatial resolution. Furthermore, these systems also typically measure a small number of slices so they can use a fan beam reconstruction, thus a translation of either the source or the flow is required to measure a large number of slices. A second method to eliminate the rotation mass of the system is to use a custom X-ray source, known as an electron beam source. An electron beam source differs from a normal tube source in that it contains high voltage electric fields to deflect the electron beam within the source so it impacts at different locations on a large anode. Thus, a moving X-ray source can be achieved without any physical moving parts (Budoff and Gul, 2006; Fischer et al., 2008). These so called electron beam tomography (EBT) systems are capable of scan rates up to 10,000 Hz, while maintaining sub-millimeter resolution (Bieberle et al., 2010; Mudde, 2011). However, up to this point, they are limited to scanning a small number of slices at one time.

2.2 Computed Tomography (CT) Reconstruction

In computed tomography, the most difficult and most critical step is the reconstruction. The reconstruction transforms the set of projections the scanner measured into a 3D representation of the measured flow. This process is sensitive to noise in the projections and the number of views measured. However, by understanding the reconstruction process, a balance can be found between the quality of the reconstruction and the quantity and quality of views required, which in turn correlates to the time required to acquire a CT.

Mathematically, a CT is simply a Radon transform of the object being scanned. That is to say, it is a set of line integrals, which represent the function being measured. In order to determine the values of the function, all that is needed is to invert the Radon transform. However, calculating the inverse of a Radon transform is not a trivial problem, which is exacerbated by the discrete sampling and complex scanning geometries used in real CT scanners. The most computationally efficient method of solving this problem is via the Fourier projection-slice theorem. While computationally efficient, the Fourier projection-slice theorem cannot handle advanced scanning geometries. Therefore, two other classes of algorithms are typically used to calculate the inverse Radon transform: filtered backprojection (FBP) and algebraic reconstruction techniques (ART). It should also be noted, that while this section specifically discusses reconstruction techniques for X-ray CT, the same basic principles can be applied to any form of tomography.

2.2.1 Fourier Projection-Slice Theorem

In the 2D case, the concept of the Fourier projection-slice theorem (also called the central slice theorem) is as follows. First, the Fourier transform of the projection is determined. The result of this transform is translated so it is along the line parallel to the projection, but centered on the origin of the 2D Fourier space. This is repeated for all projections and therefore fills the 2D Fourier. Once all the projections have been added, the 2D inverse Fourier transform is computed, and that result is the spatial domain slice (Epstein, 2003; Hsieh, 2009). The Fourier projection-slice theorem also holds in three-dimensions, with the projections being planes instead of lines. While this calculation is computationally efficient, it has some drawbacks. First, since aligning the projections in the Fourier space requires a resampling from Cartesian coordinates to polar coordinates, an interpolation is

required. However, interpolations in the frequency domain create much greater errors than interpolations in the spatial domain (Hsieh, 2009). Furthermore, the Fourier projection-slice theorem is only valid for parallel beam projections.

2.2.2 Filtered Backprojection (FBP)

To overcome the limitations associated with reconstruction via the Fourier projection-slice theorem, filtered backprojection was developed. The concept behind the filtered backprojection algorithm is relatively simple. First, consider a simple backprojection. In this case, the ray from the X-ray source to each projected point at one angle is computed, and the projection value is added to each point the ray passes through. This is repeated for all the projected angles, averaging the values at each point. This returns a reconstruction of the slice, but it will be blurred (Shepp and Kruskal, 1978). To correct for this blur, a filtering step is added prior to the backprojection. In this filtering step, the projected intensities are convolved with a filtering kernel. The most basic kernel is a ramp filter; however, numerous other kernels have been proposed (Hsieh, 2009; Shepp and Kruskal, 1978; Shepp and Logan, 1974).

While the concept of the FBP algorithm is applicable to any scanning configuration, the details of the mathematics vary by configuration. The simplest configuration is the parallel beam case. In this case the FBP is:

$$f(x,y) = \int_0^{\pi} p(t,\theta) * g(t) d\theta \tag{2.5}$$

where $p(t,\theta)$ is the projection at angle θ and detector position t, $g(t)$ is the filter kernel, $f(x,y)$ is the reconstructed slice, and $*$ is the convolution operator (Hsieh, 2009). There are two interesting features of note with the parallel FBP algorithm. First, in the parallel case, only 180° of projections are required to reconstruct the object. Second, because X-rays are

attenuated in an exponential fashion (see the Beer- Lambert law in Eq. (2.3)), the natural logarithm of the raw data from the detector must be computed to generate $p(t,\theta)$. While the second observation holds true for all geometries and reconstruction methods, the first is only true for a parallel beam CT scanning geometry.

A more common and more complicated geometry is the fan beam geometry. A fan beam CT scanner can be designed in two different ways. One way is with a curved detector, such that the angle between each measured point on the detector (relative to the source) is the same. The second method is with a flat detector and evenly spaced measured points. The second method is the more common, and will be considered here. In a fan beam CT system, the divergence of the beam causes the portion of the object near the detector to be sampled at a higher rate than the portion of the object near the X-ray source. When computing the FBP, this is accounted for with scaling terms. This gives the equation:

$$f(r,\phi) = \frac{1}{4\pi^2}\int_0^{2\pi} \frac{d^2}{(d + r\cos(\phi-\theta))^2}\left(\frac{d}{\sqrt{d^2+a^2}}p(r,\theta)\right) * g(t)d\theta \qquad (2.6)$$

where d is the source to center distance, r is the radius in the slice from the center, ϕ is the angle from the x-axis within the slice, and a is the scaled distance to the sampled position in the source (Feldkamp et al., 1984; Hsieh, 2009).

While fan and parallel beam backprojections are sufficient for earlier forms of computed tomography, where only one slice is scanned at a time, they will produce significant artifacts in more recent volume CT systems, which use conical beams. Furthermore, because the beam diverges in the z-direction, the plane of the projection for a given point varies from projection to projection. The solution to this problem is an approximate filtered backprojection algorithm, which was developed by Feldkamp, et al. (1984), and has gained

widespread adoption since. This algorithm is often called the FDK algorithm, in honor of its authors (Yan et al., 2008). The FDK algorithm is given by:

$$f(x,y,z) = \frac{1}{4\pi^2}\int_0^{2\pi} \frac{d^2}{(d + x\cos\theta + y\sin\theta)^2}\left(\frac{d}{\sqrt{d^2 + a^2 + b^2}}p(r,\theta)\right) * g(t)d\theta \qquad (2.7)$$

where *a* and *b* are the scaled sampling distances in the *x*- and *z*-directions, with respect to the detector (Feldkamp et al., 1984; Yan et al., 2008).

From a close examination of the FBP algorithms, it can be seen that for all cases the computational complexity is $O(MN^3)$, where *M* is the number of projections, and *N* is the number of voxels in one direction of the volume, assuming the volume is a cube. While this is more complicated than the Fourier projection-slice theorem, it is less complicated than ART algorithms. Due to this complexity, it is advantageous to compute the algorithm on a highly parallel processor, such as a GPU (Wang et al., 2010; Yan et al., 2008). Thankfully, both the convolution operation and the backprojection operation map well to highly parallel processors.

2.2.3 Algebraic Reconstruction Techniques (ART)

The final class of algorithms used in CT reconstruction is the algebraic reconstruction techniques, also called iterative reconstruction. In ART, the CT is modeled as a large system of equations:

$$\boldsymbol{p} = \boldsymbol{A} \cdot \boldsymbol{G} + \boldsymbol{e} \qquad (2.8)$$

where $\boldsymbol{p}$ is the projections, $\boldsymbol{G}$ is the reconstructed object, $\boldsymbol{A}$ is a weighting matrix, and $\boldsymbol{e}$ is the error of the system. While this system is relatively trivial to solve for very small volumes, it is difficult to calculate for large volumes as the system is nearly always under or over constrained, depending on the number and size of the projections, and the size of the volume. Thus, iterative techniques are required to solve the equation and minimize the error, $\boldsymbol{e}$.

It is interesting to note that the original CT scanner built by Hounsfield computed the reconstruction using an algebraic technique. However, ART was soon abandoned in favor of the more computationally efficient, and at the time, more accurate FBP methods (Shepp and Kruskal, 1978). With recent increases in computing power, ART is starting to return to usage. This is due in large part to the ability of ART to model the physics of X-ray CT while doing the reconstruction. This allows the reconstruction to compensate for inaccuracies in the assumptions made by other reconstructions. For example, ART reconstruction can use polyenergetic X-rays and finite source size instead of assuming a monoenergetic, infinitesimal source. Additionally, ART has been shown to handle the reconstruction of CTs from a limited number of projections better than FBP (Hsieh, 2009).

2.3 Volume Visualization

Regardless of which algorithm is used to reconstruct a CT scan, the final output is a volumetric dataset, or volume. A volume generally consists of a 3D rectilinear grid of regularly spaced voxels, with each voxel having a scalar value. Advanced types of volumes exist, which use other forms of grids (e.g., tetrahedral grids), irregularly spaced voxels, or vector-valued voxels (Engel et al., 2006). However, only the algorithms for rendering rectilinear grids are discussed in this section. Furthermore, while this review focuses on the rendering of CT data, it should be noted that the same volume rendering techniques could be applied to any volume dataset, irrespective of how it was generated.

Due to the difference in input data between traditional computer graphics (which rely on a large number of triangles to represent an object's surface) and volume rendering (which aims to render a large, dense set of points in 3D space) a fundamentally different approach to rendering is required for volumetric data. There are two categories of volume rendering

defined in the literature: indirect volume rendering (IVR) and direct volume rendering (DVR) (Meissner et al., 2000). Indirect volume rendering is not a true rendering of the volume. Instead, an intermediate geometry is created from the volume to represent a given property of the data. The simplest approach to IVR is to view individual slices aligned with one of the volume's grid axes (Section 2.3.2.1). While simple, such a rendering technique requires a great deal of imagination on the part of the viewer to understand 3D structures in the data. A more advanced IVR method is isosurfacing (Section 2.3.2.2), which provides a rendering of one 3D surface within the volume. However, isosurfacing provides one surface, not a true rendering of the entirety of the volume.

When a single surface within the volume is insufficient to visualize the data of interest, one of the DVR methods may be employed. In order to obtain a DVR, the rendering engine needs to evaluate the volume rendering integral, which is a mathematical model of how light travels through the volume. The volume rendering integral is mathematically derived from the Radon transform. As it is extremely difficult to evaluate the volume rendering integral analytically, several different methods of approximating it have been developed. Five of these DVR methods will be reviewed here: texture-based volume rendering (Section 2.3.3.1), volume splatting (Section 2.3.3.2), shear-warp rendering (Section 2.3.3.3), volume ray casting (Section 2.3.3.4), and frequency domain rendering (Section 2.3.3.5). All of these methods were initially developed using a central processing unit (CPU) as the computation engine, and then modified to run on a graphical processing unit (GPU) as the capabilities of GPUs became more generalized. Due to this development, it is critical to understand the basics of traditional computer graphics and GPU computing. Thus, a brief overview has been provided in Section 2.3.1.

2.3.1 Introduction to Computer Graphics

Traditional computer graphics use 3D surfaces to model objects and then projects them onto the computer screen to render the scene. This is achieved by approximating the surfaces as a collection of triangles and then projecting those triangles onto the screen space using a virtual camera. Once the triangles are projected, they are converted to pixels by the GPU's rasterizer. While this style of rendering is incompatible with volumetric data, it is important to understand it, as many of the mathematical foundations are the same. Furthermore, there are many creative ways in which the traditional rendering pipeline has been utilized to achieve direct volume rendering on the GPU.

In graphics programming there are two basic paradigms of processing the data: fixed-function pipeline and programmable pipeline. Both methods use the pipeline analogy, in which input data are passed to the GPU, and then it is processed in a sequential series of stages, with each stage taking input data, transforming it in some manner, and passing it to the next stage. When the final stage in the series is reached, the fully transformed data is passed to the output, which in the case of computer graphics is typically an image rendered on the computer screen (Möller and Haines, 1999). It is important to note that while the stages of the pipeline run sequentially, the data may be (and in practice usually are) processed in parallel within an individual stage and multiple stages can run simultaneously on different input data.

The difference between the fixed-function pipeline and the programmable pipeline lies in the flexibility of the stages (Zink et al., 2011). In the fixed-function pipeline, the function of each stage is pre-determined by the GPU designers and application programming interface (API) writers. The application programmer may have the ability to change parameters

controlling how a stage operates, but is not free to implement the stage in an entirely different manner. The programmable pipeline allows the application programmer to change how a stage operates by using a shader, which is a small program that runs on a GPU and implements one stage of the pipeline. While this greatly increases the flexibility of graphics cards, there are still some stages within the programmable pipeline that remain fixed-function because the performance benefits of a fixed-function stage outweigh the value of flexibility for those stages. Furthermore, the order in which the stages operate is still fixed, although some stages may be omitted if the programmer chooses. More information on the programmable pipeline can be found in Section 2.3.1.1. Finally, it should be noted that the fixed-function pipeline is now deprecated and has been removed from the latest versions of most graphics libraries, as the programmable pipeline is significantly more flexible and retains the ability to implement functionality identical to the fixed-function pipeline.

There are two main APIs used to produce computer graphics: DirectX (specifically, the Direct3D portion of DirectX) and OpenGL. Both APIs provide cross-vendor hardware support via a hardware abstraction layer (HAL). However, OpenGL also supports cross-platform graphics programming, whereas DirectX is Microsoft Windows specific. While there are some differences in features between the two APIs, the major concepts are the same, as both use the same mathematical foundations and both currently use the programmable pipeline paradigm. At the time of writing, the latest version of DirectX is v11.1 and the latest version of OpenGL is v4.4. For the purpose of clarity, DirectX terminology will be used herein when there is a difference between DirectX and OpenGL; however, the terminology differences will be noted as concepts are introduced.

2.3.1.1 Programmable Pipeline

Support for the programmable pipeline was introduced in DirectX v8.0 and OpenGL v2.0 and has expanded ever since ("History of OpenGL," 2013; Microsoft, 2000). As the abilities of the fixed-function pipeline can also be achieved using the programmable pipeline, support for the fixed-function pipeline was removed in DirectX v10.0 and OpenGL v3.1 ("Fixed Function Pipeline," 2012; Microsoft, n.d.-d). Therefore, to understand modern graphics rendering, a detailed understanding of the programmable pipeline is important, whereas a detailed understanding of the fixed-function pipeline is not necessary. Thus, only the programmable pipeline will be detailed. It should be noted that the programmable pipeline described herein is that of the most recent version of the respective APIs (DirectX 11.1 and OpenGL 4.4), older versions of the APIs are still in use and may contain only a subset of the stages and features described. The complete programmable rendering pipeline for the latest versions of both DirectX and OpenGL are shown schematically in Figure 2.1.

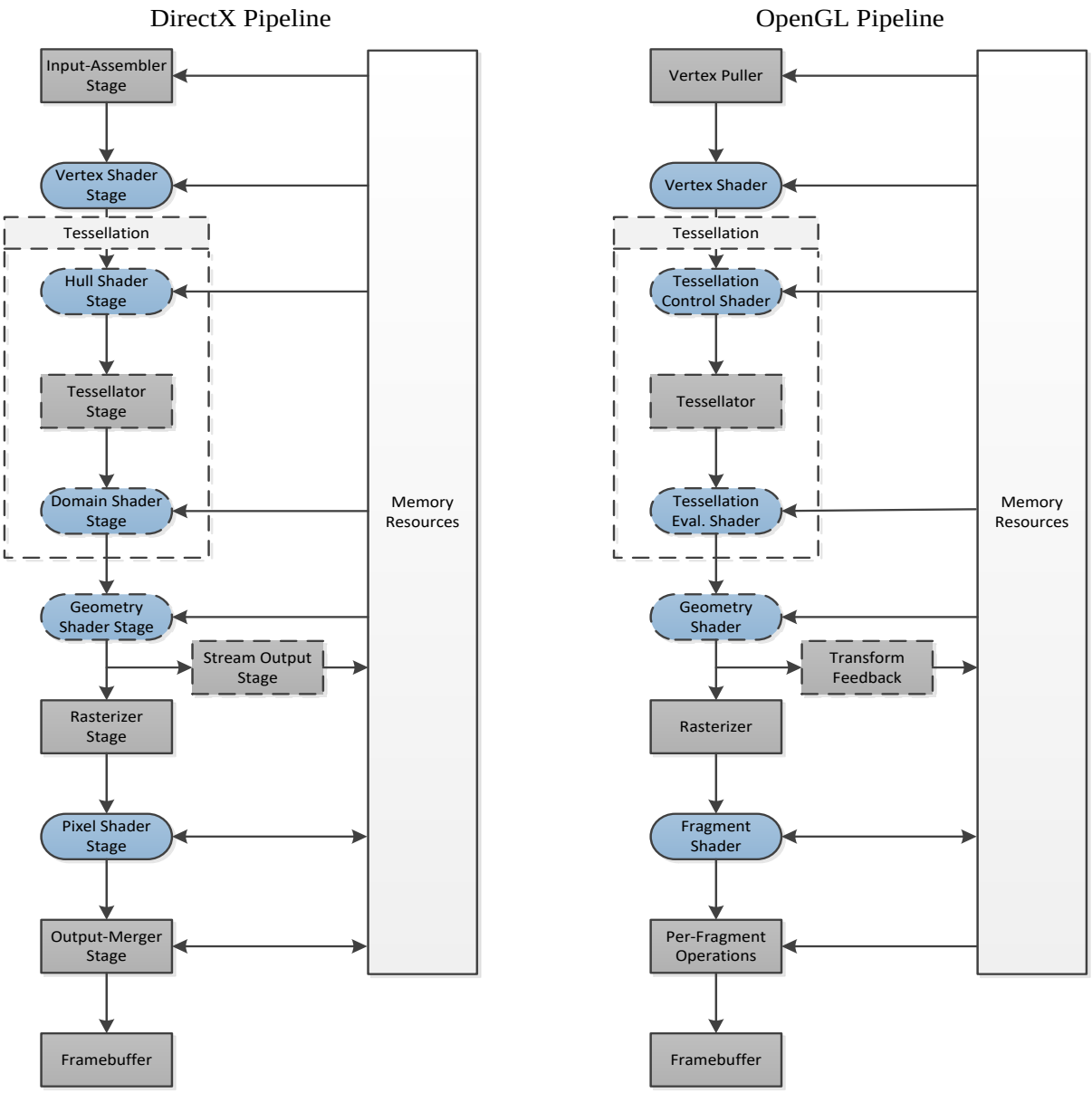

Figure 2.1: The DirectX (left) and OpenGL (right) programmable graphics pipelines. The stages in ellipses are programmable and the stages with dashed outlines are optional (adapted from Khronos Group, 2012; Microsoft, n.d.-f; "Rendering Pipeline Overview," 2012).

At its most basic level, the programmable pipeline is a set of small programs, where each program describes one of the steps required to transform the triangle descriptions into a rendered on-screen image. The first step in this process is a fixed function stage called the "Input-Assembler Stage" (or "Vertex Puller" in OpenGL parlance). The input-assembler stage is responsible for copying all the data about the input primitives from CPU memory to

GPU memory. While triangles (referred to as a mesh when all the triangles involved define a single object) are the most commonly used primitives, primitives could also be points, lines, or mathematical curve descriptions, such as for Bézier curves or Non-Uniform Rational Basis-Splines (NURBS). The common feature shared by all these primitives is that the geometry is defined by a finite number of points, and that their mathematical formulations are a function of the primitive, not of the geometry the primitive represents. Taking a triangular mesh as an example, the entire object is defined by an array, which contains the vertices of all the triangles necessary to create the shape. While this mesh is often an approximation of the true object shape, any finite object can be approximately represented by a finite number of vertices with the only geometry dependent variables being the position of the vertices and number of vertices. However, in almost all triangular meshes, a single vertex is used by two or more triangles. The simplest way to implement triangle rendering is to replicate the vertex for each triangle; however, this leads to inefficient processing due to redundancies in data storage and processing.

The input-assembler provides two methods of mitigating this inefficiency. The first method is indexing. With an indexed primitive, each unique vertex is stored in an array (the vertex array) and passed to the input-assembler. Additionally, a second array (the index array), containing the locations of vertices within the vertex array, is also passed to the input-assembler. Each set of three sequential indices within the index array represent three vertices in the vertex array, which together represent one triangle in the mesh. While the index array adds some overhead, it typically requires less overhead than what would be required if all the repeated vertices were stored and processed. The second method of improving efficiency is a triangle strip. In a basic triangular mesh (also called a triangle list), each set of three vertices

forms a triangle, requiring 3N vertices (where N is the number of triangles). In a triangle strip, it is assumed that the last two vertices of the last triangle are the first two vertices of the next triangle. Thus the vertex array (A, B, C, D) would have two triangles, one with vertices at points A, B, and C, and the second with vertices at points B, C, and D. This method only requires 2 + N vertices for N triangles (Luna, 2008). It should also be noted that indexing and triangle strips can be used together, if the programmer desires. These efficiency gains become more important as more information is used to describe each vertex. Such additional information often includes a surface normal and a texture coordinate, both of which can be used for shading. In addition to the obvious memory usage reduction, there is also a reduction in GPU processor usage, as the input-assembler can instruct the GPU to process each vertex only once, and sort out which vertex belongs to which triangle later.

The data processing starts in the next stage of the pipeline, the vertex shader. The vertex shader's purpose is to run a mapping process, which calculates a geometric transformation on each vertex to produce exactly one output vertex for each input vertex. Due to the mapping nature of the vertex shader, each vertex can be processed independently, without any knowledge of any other vertex. The vertex shader typically does a series of transformations, shown in Figure 2.2, on vertices, model (or local) space to world space, world space to view (or camera) space, and view space to homogeneous clip space. A final transformation from homogeneous clip space to normalized device coordinates is done in the rasterizer stage.

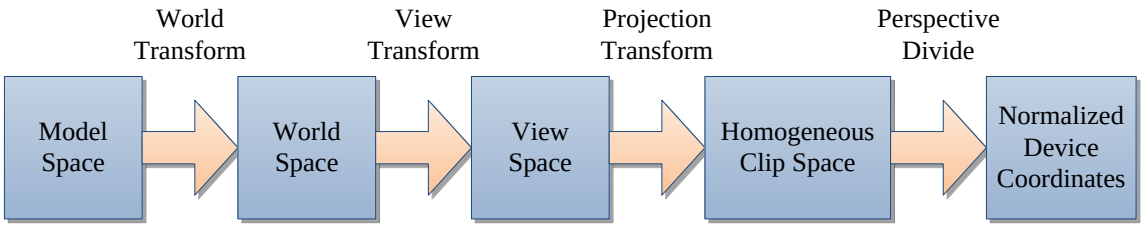

Figure 2.2: The sequence of geometric transformations used in traditional computer graphics.

The first geometric transformation done by the vertex shader is to transform the primitives from model space to world space. This transformation is necessary because mesh objects are created in their own coordinate space, about an origin that makes sense for the model. This has the advantages of making object modeling easier, and allowing the same model to be used multiple times in the same rendering scene. However, in order to place the model in the correct position in the scene, the model's vertices must be transformed into the world coordinate system. This is achieved through geometric transforms (most commonly translation and rotation transforms), which are implemented as matrix calculations. Before discussing the mathematics, it is important to note that DirectX traditionally uses a left-handed coordinate system, while OpenGL traditionally uses a right-handed coordinate system (Möller and Haines, 1999). However, with the advent of the programmable pipeline, it has become possible for both DirectX and OpenGL to use either right-handed or left-handed coordinate systems; therefore, all formulas herein will assume a right-handed coordinate system.

In computer graphics, a 3D point is represented by a four-tuple, where: $\boldsymbol{p} = (x, y, z, 1)$. Similarly, a 3D vector is represented by a four-tuple, where: $\boldsymbol{v} = (x, y, z, 0)$. The difference in the fourth term of the four-tuple allows the same 4×4 transformation matrices to be used for both points and vectors—a practice is known as "homogeneous coordinates" (Luna, 2008). There is an exception to this involving non-uniform scaling transforms of normal vectors (Möller and Haines, 1999); however, this case is rare in practice, and will not be covered. Using homogeneous coordinates and four-tuple row vectors to store points and vectors, any transform can be represented by the equation:

$$\boldsymbol{w} = \boldsymbol{u} * \boldsymbol{T} \tag{2.9}$$

where $\boldsymbol{u}$ is the original point or vector, $\boldsymbol{T}$ is the transformation matrix, and $\boldsymbol{w}$ is the transformed point or vector. This convention assumes that the four-tuple is stored as a row vector, which is a commonly used convention in DirectX (Zink et al., 2011); however, other computer graphics references (Möller and Haines, 1999) use a column-vector format to store the four-tuples, e.g., $\boldsymbol{p} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$. In this case the transformation becomes:

$$\boldsymbol{w} = \boldsymbol{T}^{\mathrm{T}} * \boldsymbol{u} \tag{2.10}$$

where $\boldsymbol{T}^{\mathrm{T}}$ is the transpose of the matrix $\boldsymbol{T}$.

The most basic type of transformation used in computer graphics is the translation transform, which displaces a point by a given vector $\boldsymbol{b}$. The translation transform is defined by the matrix:

$$\boldsymbol{T} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ b_x & b_y & b_z & 1 \end{bmatrix} \tag{2.11}$$

Note that when the translation matrix is used to transform a vector, it results in the original vector. The use of homogenous coordinates maintains the properties of the vector—it has a magnitude and a direction, but no position. The next common transformation is the rotation transformation. The most general form of this transformation is rotating a given angle, θ, around an axis, given by the normalized vector $\boldsymbol{r}$ (Möller and Haines, 1999). In this case, the rotation matrix is:

$$\boldsymbol{T} = \begin{bmatrix} \cos\theta + (1-\cos\theta)r_x^2 & (1-\cos\theta)r_x r_y + r_z \sin\theta & (1-\cos\theta)r_x r_z - r_y \sin\theta & 0 \\ (1-\cos\theta)r_x r_y - r_z \sin\theta & \cos\theta + (1-\cos\theta)r_y^2 & (1-\cos\theta)r_y r_z + r_x \sin\theta & 0 \\ (1-\cos\theta)r_x r_z + r_y \sin\theta & (1-\cos\theta)r_y r_z - r_x \sin\theta & \cos\theta + (1-\cos\theta)r_z^2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.12}$$

However, for the common cases of rotation around the x, y, or z-axis, this matrix can be simplified (Zink et al., 2011). Substituting in $\boldsymbol{r}$ = (1, 0, 0) for the rotation axis, equation (2.12) simplifies to the rotation matrix for a rotation about the x-axis:

$$\boldsymbol{T} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.13}$$

Similarly, the rotation matrix for a rotation about the y-axis is:

$$\boldsymbol{T} = \begin{bmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin 0 & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.14}$$

and the rotation matrix for the rotation about the z-axis is:

$$\boldsymbol{T} = \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.15}$$

The final common transformation is the scaling transformation. It is most commonly used when an object is modeled using one type of unit, but rendered in another. This conversion is particularly important in virtual reality, as objects must be rendered at their real size in order to achieve proper binocular disparity. The scaling transformation is:

$$\boldsymbol{T} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.16}$$

where $\boldsymbol{s}$ is a vector representing the percentage of scaling in each direction. As previously noted, if the scaling is non-uniform (i.e., $s_x \neq s_y \neq s_z$), special consideration must be taken when transforming surface normal vectors.

In most computer graphics, the use of a single transformation on its own is uncommon. In most cases, multiple transformations need to be combined to achieve the desired result.

This is achieved by simply multiplying the two (or more) transformation matrices together. However, it is important to note that matrix multiplications are not commutative, that is to say, the order of the multiplication matters. For example, when an object is rotated and translated, if the translation occurs first, the object will rotate about the new coordinate system origin. Conversely, if the rotation occurs first, the rotation will occur about the original origin and then the object will be translated to its new position. This is of particular importance when an arbitrarily positioned object needs to be rotated about the object's center. To achieve this, the object's coordinate system must be translated to move the coordinate system origin to the object's center, then the object is rotated, and finally the coordinate system is translated again to return the origin to the proper location. In a case such as this, where a sequence of transformations needs to be done on multiple vertices, the matrices may be multiplied once, and the resulting transformation matrix can be used to transform all the vertices. This pre-multiplication reduces the computational load on the GPU. Finally, while all of the listed transformation matrices can be created manually, there are functions available in both DirectX and OpenGL to simplify the creation of transformation matrices.

The second transformation that the vertex shader can perform is the conversion from world space to view space. View space is defined with respect to a virtual camera, and aligns the coordinate system so the virtual camera is positioned at the origin. This transformation can be created by calculating the translation and rotation matrices necessary to convert between the two coordinate spaces. However, in practice it is easier to create the view matrix based on where the camera is located in the world coordinates, where the camera is looking, and which direction is up for the camera. The creation of this transformation is a two-step

process. First, the x-, y-, and z-axes must be computed from the camera position ($\boldsymbol{p}_{camera}$), the position the camera is looking at ($\boldsymbol{p}_{target}$), and the normalized up direction vector ($\boldsymbol{v}_{up}$). This is done using the equations:

$$\boldsymbol{z} = \frac{\boldsymbol{p}_{camera} - \boldsymbol{p}_{target}}{\left\|\boldsymbol{p}_{camera} - \boldsymbol{p}_{target}\right\|} \tag{2.17}$$

$$\boldsymbol{x} = \frac{\boldsymbol{v}_{up} \times \boldsymbol{z}}{\left\|\boldsymbol{v}_{up} \times \boldsymbol{z}\right\|} \tag{2.18}$$

$$\boldsymbol{y} = \boldsymbol{z} \times \boldsymbol{x} \tag{2.19}$$

where $\boldsymbol{x}$, $\boldsymbol{y}$, and $\boldsymbol{z}$ are the x-, y-, and z-axes, respectively, $\|\boldsymbol{v}\|$ denotes the magnitude of the vector $\boldsymbol{v}$, and × denotes the vector cross product. Once the axes are calculated, the transformation matrix to move from world space to view space is created using:

$$\boldsymbol{T} = \begin{bmatrix} x_x & y_x & z_x & 0 \\ x_y & y_y & z_y & 0 \\ x_z & y_z & z_z & 0 \\ \boldsymbol{x} \cdot \boldsymbol{p}_{camera} & \boldsymbol{y} \cdot \boldsymbol{p}_{camera} & \boldsymbol{z} \cdot \boldsymbol{p}_{camera} & 1 \end{bmatrix} \tag{2.20}$$

where x_x denotes the x value of the vector $\boldsymbol{x}$, y_x denotes the x value of the vector $\boldsymbol{y}$, etc., and $\boldsymbol{i} \cdot \boldsymbol{j}$ denotes the dot product of vectors $\boldsymbol{i}$ and $\boldsymbol{j}$ (Luna, 2008; Microsoft, n.d.-a).

The final transformation the vertex shader can perform is the projection of vertices from the view space to homogeneous clip space. Homogeneous clip space is a normalized coordinate space used by the graphics card to calculate what geometry should be clipped out of the rendered image and what geometry occludes other geometry. This space has x and y values from -1 to 1, with z-values varying based on the handedness of the coordinate system and the API used. Left-handed coordinates, with z-values from 0 to 1 (near to far) are most common in DirectX. OpenGL also typically uses left-handed coordinates in homogeneous clip space, despite its use of right-handed coordinates elsewhere in the API; however, it typically scales the z-value from -1 to 1 (near to far) (Möller and Haines, 1999).

There are two types of projections used in computer graphics to achieve the view space to homogeneous clip space transformation: orthographic projection and perspective projection. In an orthographic projection, parallel lines in the view space will remain parallel after the projection (Möller and Haines, 1999). This is commonly used in computer aided design (CAD) applications. For a right-handed coordinate system with a viewing volume of (1, 1, 0) to (-1, -1, -1), the orthographic projection matrix is:

$$\begin{bmatrix} \frac{2}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2}{t-b} & 0 & 0 \\ 0 & 0 & \frac{1}{n-f} & 0 \\ -\frac{r+l}{r-l} & -\frac{t+b}{t-b} & -\frac{n}{n-f} & 1 \end{bmatrix} \tag{2.21}$$

where *r* and *l* are the x-coordinates of the right and left planes, respectively, *t* and *b* are the y-coordinates of the top and bottom planes, respectively, and *n* and *f* are the z-coordinates of the near and far plane respectively. In this coordinate system, the z-value will decrease as an object gets further away; however, it is often preferable to have the z-value increase as an object's distance from the camera increases. Due to this preference, it is common for computer graphics to use a left-handed coordinate system in the projection space, even if right-handed coordinates are used in other places. The orthographic projection matrices for other handedness and view volumes are available in other sources (Khronos Group, 2012; Luna, 2008).

In contrast, perspective projection causes lines that are parallel in the view space to converge toward a single point after the projection. This simulates the apparent size decrease of objects with increased distance that humans observe in everyday life. The additional distance cues of the perspective projection make it common in video games, and almost

mandatory in virtual reality (Sherman and Craig, 2003). While more common, the perspective projection is more complex and has to be computed in two parts. The first part scales the view space into homogeneous clip space and sets a scaling factor, based on the vertex's depth in the scene, to the w-value. This step is accomplished using the projection matrix (for a right-handed DirectX viewing volume):

$$\begin{bmatrix} x_{scale} & 0 & 0 & 0 \\ 0 & y_{scale} & 0 & 0 \\ 0 & 0 & \frac{f}{n-f} & -1 \\ 0 & 0 & \frac{n*f}{n-f} & 0 \end{bmatrix} \tag{2.22}$$

where *n* and *f* define the z-location of the near and far planes, respectively, and x_{scale} and y_{scale} are defined by:

$$x_{scale} = \frac{1}{r \tan\left(\frac{\alpha}{2}\right)} \tag{2.23}$$

$$y_{scale} = \frac{1}{\tan\left(\frac{\alpha}{2}\right)} \tag{2.24}$$

where *r* is the aspect ratio of the rendered image (width/height) and α is the field of view of the virtual camera (Luna, 2008; Microsoft, n.d.-b). However, this projection makes the assumption that the view frustum (the truncated, square pyramid that represents the volume the camera can see) is symmetric. In most video game applications this is true; however, in virtual reality the frustum is usually asymmetric because it is defined based on the user's location relative to the viewing screen (Cruz-Neira et al., 1993b). In this case, the frustum is defined by the left, right, top and bottom locations at the near plane, as well as the near plane and far plane positions, using the projection matrix:

$$\begin{bmatrix} \frac{2n}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2n}{t-b} & 0 & 0 \\ \frac{l+r}{r-l} & \frac{t+b}{t-b} & \frac{f}{n-f} & -1 \\ 0 & 0 & \frac{n*f}{n-f} & 0 \end{bmatrix} \tag{2.25}$$

where the locations are denoted as *l*, *r*, *t*, *b*, *n*, and *f*, respectively (Microsoft, n.d.-c; Möller and Haines, 1999). The second step, which is the same whether the perspective projection uses a symmetric or asymmetric projection, divides the x-, y-, and z-values by the w-value to scale the vertex position and create the illusion of distance. This step is known as the perspective divide, and is computed in the rasterizer stage. Finally, it should be noted that while the projection transformation, as well as the view space to homogeneous clip space transformation, are typically performed in the vertex shader, they do not have to be. The only requirement is that these transformations are applied prior to reaching the rasterizer stage of the pipeline.

After the vertex shader, the pipeline can, optionally, go through a tessellation process. The tessellation process is comprised of three stages, the hull shader stage (tessellation control shader in OpenGL), the tessellator stage, and the domain shader stage (tessellation evaluation shader in OpenGL). If the tessellation process is used, all three stages are required in DirectX. OpenGL only requires the tessellator stage and the tessellation evaluation shader, leaving the tessellation control shader fully optional. Unlike the rest of the stages in the rendering pipeline, the stages in the tessellation process are not intended to work with traditional triangular meshes. The tessellation process is designed to generate triangular meshes from mathematically represented surfaces, such as Bézier curves or NURBS (Piegl and Tiller, 1997; Zink et al., 2011).

The first tessellation stage is the hull shader. The hull shader is a setup step that controls the position of control points to use in the tessellation process and the operating parameters of the tessellator stage. In DirectX, this is performed with two separate shader functions, which together make up the hull shader stage. In OpenGL, this setup can be done either with a single shader function, or the stage may be omitted and fixed parameters can be sent to the tessellator from the CPU. When setting up the control points (or patch vertices in OpenGL parlance), the hull shader can create and destroy control points; however, a more common use is to apply geometric transformations that were not applied in the vertex shader. This is particularly efficient because most object representations used in tessellation are invariant to affine transformations, meaning the rendered object will be the same whether the geometric transformations are done on the control points or the generated vertices. Since there are typically significantly more generated vertices than control points, it is much more efficient to transform the control points than the vertices. Note however, that any transformations performed in the vertex shader stage are passed on to the hull shader, so it is common for the hull shader to pass control points through without any manipulation.

The second part of the hull shader, setting up the tessellator operation, simply provides instructions to the fixed-function tessellator stage on what type of domain (isoline, triangle, or quad) to use, and how finely to break up each side of the domain, as well as how finely to partition the interior of the domain. Note that a domain is different from a geometric object, such as a triangular mesh. A domain is a space over which a parametric equation—that mathematically represents the object—is calculated. The divisions in the domain are the points at which the parametric equation is calculated. Thus, the more finely partitioned the domain, the more accurate the rendered approximation of the mathematical object. By using

the hull shader to vary how finely objects are partitioned, based on parameters such as the depth of the object in a scene, the hull shader can achieve a high-quality level-of-detail effect using the same input (the control points) for both very finely and very coarsely detailed cases.

Once the hull shader calculates how finely to partition a domain, that information is sent to the tessellator stage. The tessellator stage is a fixed-function stage, whose sole purpose is to calculate the division of the domain based on the information passed to it from the hull shader. The calculated positions in the domain, along with the control points from the hull shader, are then passed to the domain shader.

The domain shader's purpose is to evaluate the parametric equation that represents the object, converting the domain position and control points into a vertex. The different parametric equations that could be used to represent the object are too numerous to be covered here; however, Shreiner et al. (2013) provide an example of a Bézier patch in OpenGL, and Piegl and Tiller (1997) provide an excellent overview of NURBS, albeit without information on GPU implementation. In addition to the vertex calculation, the domain shader can also compute geometric transformations, although, as previously noted, this is more efficient to do on the control points for any affine transformations. The domain shader can also compute other information about the vertex, such as the surface normal or texture coordinates.

The next stage in the rendering pipeline is the geometry shader stage. The geometry shader is an optional, programmable stage that processes whole primitives, which it receives from either the vertex shader or the domain shader, depending on whether tessellation is in use. The advantage of the geometry shader is that it has the capability to create and destroy

primitives. Due to this capability, a common usage of the geometry shader is to render particle systems. A particle system is a collection of small objects, such as a collection of triangles, that are used to represent dynamic, diffuse phenomena, e.g., dust and fire, in computer graphics. Each small object (or particle) moves independently to represent the dynamic nature of the phenomenon. The geometry shader is beneficial for rendering particle systems because each particle can be represented by a single point in space, and then used to create one or more larger objects (for example a triangle) in the geometry shader. This processing method allows more large particles to be rendered for the same computational cost than what could be achieved if each large particle was fully generated at the beginning of the pipeline. The geometry shader can also implement the same geometric transformations as the vertex shader; however, it is not recommended, as it computes the transformations less efficiently (Zink et al., 2011). Finally, the geometry shader is also capable of outputting transformed geometry to the stream output stage (transform feedback stage in OpenGL). This stream output is available to the CPU and can be used for a multitude of tasks, including physics calculations, CPU-based rendering effects, and saving geometry to the hard disk drive. However, the most common use of the stream output is debugging shader programs.

The next step after the geometry shader stage is the rasterizer stage. The rasterizer stage is a mandatory fixed-function pipeline stage. The purpose of the rasterizer is to convert the geometric data the pipeline has processed up to this point into fragments. In most cases, fragments will be further processed by the pixel shader, and then rendered on screen as pixels. To get the fragments to the pixel shader, the rasterizer goes through a series of steps (Luna, 2008; Zink et al., 2011). First, the rasterizer culls primitives that are not visible in the

scene. This includes culling primitives that are entirely outside the viewing volume of the camera, as well as culling surfaces that are facing away from the camera (known as back-face culling). Since most objects are intended to represent real, 3D objects, the side of the triangle facing the inside of the object can never be seen, and thus when the inside (or back-face) of a triangle is facing the camera, it is typically safe to assume it is occluded by the front-face of another triangle, and thus the back-face is culled. If this is not a safe assumption, back-face culling can be turned off. Once non-visible primitives are culled, geometry that crosses the boundary of visibility is clipped. When the rasterizer encounters an object that is partially inside the viewable area and partially outside, it calculates where the primitive crosses the boundary, inserts new vertices at these points, and eliminates the vertices outside the viewable area. Once this is complete, the rasterizer has a complete set of geometry, including only what will be visible in the final scene. At this point, it performs the perspective divide (as explained with the perspective transformation). Finally, the rasterizer samples the geometry to create one (or more in the case of multi-sample anti-aliasing) fragment for each pixel in the final viewport. Any additional information included with the vertices of the primitives, such as texture coordinates or color, will also be interpolated and the results of the interpolation are associated with the respective fragment.

Once all the fragments are generated, the result is passed to the pixel shader (fragment shader in OpenGL terminology). The pixel shader is the final programmable stage in the pipeline and is the second of the two mandatory programmable stages. The primary purpose of the pixel shader is to apply a color to the pixel fragment based on the simulated lighting conditions. This is typically done using an algorithm, such as the Blinn-Phong model, that breaks the light down into an ambient color, diffuse color, and specular color and combines

them together using the position of the simulated light source and the interpolated surface normal of the fragment (Blinn, 1977). In addition to setting the color of the fragment, the pixel shader can also set its depth or cull the fragment. These allow the programmer significant control over the rendering of the fragment.

The final stage in the pipeline is the output merger stage (per-fragment operations stage in OpenGL). The output merger is a fixed function stage that combines all the fragments together. The output merger does this via depth testing and blending. Depth testing is the most common processing for the output merger, and it simply tests if a fragment is occluded by another fragment, and if it is, it omits the occluded fragment from the rendering. This is necessary because the processing pipeline, up to this point, has operated on a per object basis. This allows for different types of objects, for example triangle meshes and NURBS surfaces, to be rendered in the same scene with different rendering algorithms, but it means most depth testing has to be done at the output merger. Finally, the output merger can also do blending, which is the combining of two fragments together to simulate a semi-transparent object. This is not commonly used in traditional computer graphics, but can be very important in volume rendering.

Once the output merger is done processing, the rendered scene is written to the framebuffer. The framebuffer is not a true stage in the pipeline, as it is simply a storage location and no data processing occurs; however, it is important to the final rendering of the data. The simplest version of the framebuffer uses a single buffer both to write the rendered scene to and to read the data from when it is time for display on screen. While simple and memory efficient, single buffering causes reduced rendering quality because there is no way to synchronize the rendering of the graphics with the display of the graphics. This causes a

tearing effect, where a visual discontinuity occurs on screen because the rendered image is partially the new frame, and partially the old frame. To solve this, double buffering is typically used. In double buffering a back buffer is used to write the latest update to, while a separate front buffer is read to display the scene on screen. When the back buffer is fully written, the GPU will wait until the monitor is done rendering the image on screen, and then swap the front and back buffers. This puts the latest image (that was on the back buffer) on the front buffer to be rendered, and puts the old image (that was on the front buffer) on the back buffer to be redrawn with the latest update. This solves the tearing issue, but can significantly slow down the imagery refresh rate if the GPU is only capable of rendering the images just slightly slower than the monitor is capable of rendering images (Möller and Haines, 1999).

While double buffering is by far the most common framebuffer technique, there are two other important techniques. The first is triple buffering. Triple buffering adds a third buffer called the pending buffer. In double buffering, once the draw on the back buffer is completed, no rendering can be done until the buffers are swapped, otherwise the system risks not having a full scene available when it is time to swap buffers. To avoid this, triple buffering uses the pending buffer to draw continually to, while the back buffer holds the latest full update. When it comes time to swap buffers, the back buffer moves to the front buffer. Any unfinished rendering on the pending buffer is finished and it moves to the back buffer, and the front buffer becomes the pending buffer. This allows the rendering engine to run as fast as possible, but also introduces up to one frame of latency. In theory, this system could be extended to any number of buffers, at the cost of more latency.

The final buffering method is quad buffering. Despite its name, it is not an extension of triple buffering. Quad buffering is double buffering for stereoscopic images. In systems with active stereo, the left and right eye imagery are displayed in alternating fashion on screen, and then filtered with shuttered glasses. Due to the difference between the left-eye image and the right-eye image, a separate swap chain must be maintained for each eye.

2.3.2 Indirect Volume Rendering (IVR)

Due to the fundamental difference in data structure between traditional surface data and volume data, direct volume rendering is a challenging task. Therefore, one of the ways researchers have tried to visualize volume data is to convert it into a derived surface that fits into the traditional rendering pipeline. These indirect volume rendering methods have traditionally taken two forms. One form is to select a surface within the volume, and apply a texture to it, representing the value of the voxel at that point on the surface. The simplest form of this is rendering a single slice of the volume. The second form of IVR is to extract a surface at which all points have the same voxel value, known as an isosurface. This is similar to the isobars on a weather map—every point on the isobar line represents a location with the same barometric pressure.

2.3.2.1 Slice Rendering

Slice rendering (often referred to as multiplanar reconstruction or multiplanar reformation in the medical field) is the simplest way to render volumetric information. In slice rendering, a single plane, or slice, is cut through the volume, the voxels are mapped to pixels on the slice, and the resulting textured slice is rendered as a 2D picture. When the slice plane is aligned with one of the planes in the volume, this mapping is trivial. In cases where the plane is askew to the volume (often called an oblique plane) an interpolation is

required to map the volume data to the slice pixels (Ney et al., 1989). Alternatively, the mapping can be done in the Fourier domain, which eliminates the spatial domain interpolation (Kramer et al., 1990).

A more advanced version of slice rendering is used to approximate a 3D view. In this case, multiple slices of information are generated and applied as textures to rendered 3D planes. While there are a huge number of possible arrangements of the planes, the most common represents the volume by showing its primary planes (referred to as the xy-plane, yz-plane, and xz-plane in scientific work, or transverse plane, sagittal plane, and coronal plane in medicine). Figure 2.3 shows two possible combinations of this. In Figure 2.3a, the exterior surface of the region of interest is represented by six textured planes (only three are visible as rendered). In Figure 2.3b the interior of the volume is represented by three orthogonal planes. Slice renderings such as these are often used to make selections in a volume, as selection in a directly rendered volume is difficult (Ney and Fishman, 1991).

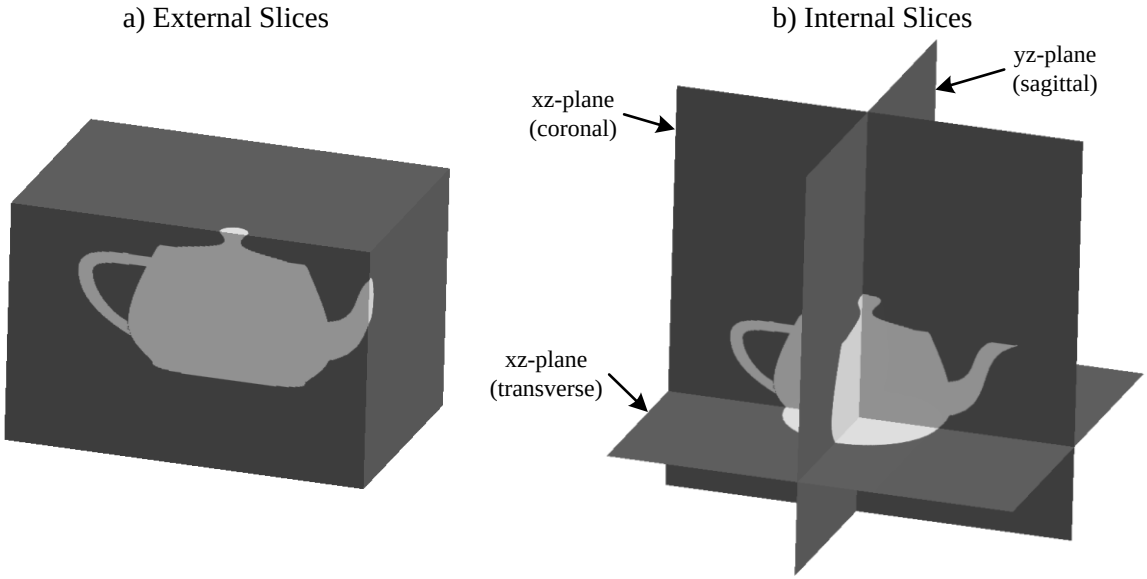

Figure 2.3: A volumetric version of the Utah teapot displayed as both external slices (a) and internal slices (b). Note, the two renderings use different regions of interest to show the teapot clearly.

While slice rendering is an extremely simple form of displaying a volume, it shows an extremely limited portion of the data, and provides very little information about the three-dimensional structure of the data. In spite of these limitations, it is still commonly used, particularly in the medical field (Maher et al., 2004). In limited cases, slice rendering has even been shown to be more effective than DVR (Liu et al., 2011). However, in most cases the additional information DVR provides makes it more effective (Addis et al., 2001; Zuiderveld et al., 1996).

2.3.2.2 Isosurface Rendering

In contrast to slice rendering, where a surface is selected and the voxel values are mapped onto the surface, isosurfacing uses a voxel value and a surface is generated to represent all voxels of the same value. The voxel value can be set by the user or determined by an automatic segmentation algorithm. By generating a surface from the volume data, the geometry can be represented with a polygon mesh, which is easy to render with the traditional graphics rendering pipeline. However, while the rendering of the final surface is relatively simple, the extraction of the surface from the volume data is computationally intensive.

The most common method for the generation of the surface is the marching cubes algorithm (Lorensen and Cline, 1987). In the marching cubes algorithm, the generation of the surface is done on a per cube basis, where each cube's vertices are represented by eight voxels of the volume (shown in Figure 2.4). To determine how the cube should be triangulated, each vertex of the cube is assigned a value of either one or zero, depending on if its value is above or below the predetermined threshold value. Because each cube has eight vertices, each with two possible states (inside or outside the surface), there are 2^8, or 256,

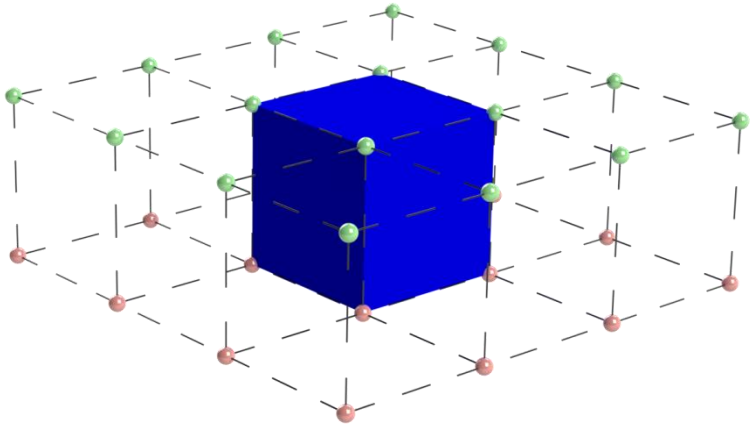

Figure 2.4: The cube (blue) used for the marching cubes algorithm, inside a field of voxels (red and green spheres) representing two slices of the volume (adapted from Lorensen and Cline, 1987).

possible geometries for any given cube. Conveniently, this is the exact size of a byte, allowing one bit in each byte to represent the state of one vertex, and the whole byte to act as an index to the appropriate form of triangulation.

Holding 256 different triangulation forms in memory is not space efficient. By analyzing the possibilities, Lorensen and Cline realized that if the values of the vertices are opposite, the triangulation is the same. For example, a cube with all eight vertices inside the threshold creates the same triangulation (no triangles) as a cube with all eight vertices outside the threshold. This reduces the unique possibilities to 128 possible triangulations. By accounting for rotational symmetry, they were able to reduce the triangulations to 15 unique possibilities, shown in Figure 2.5.

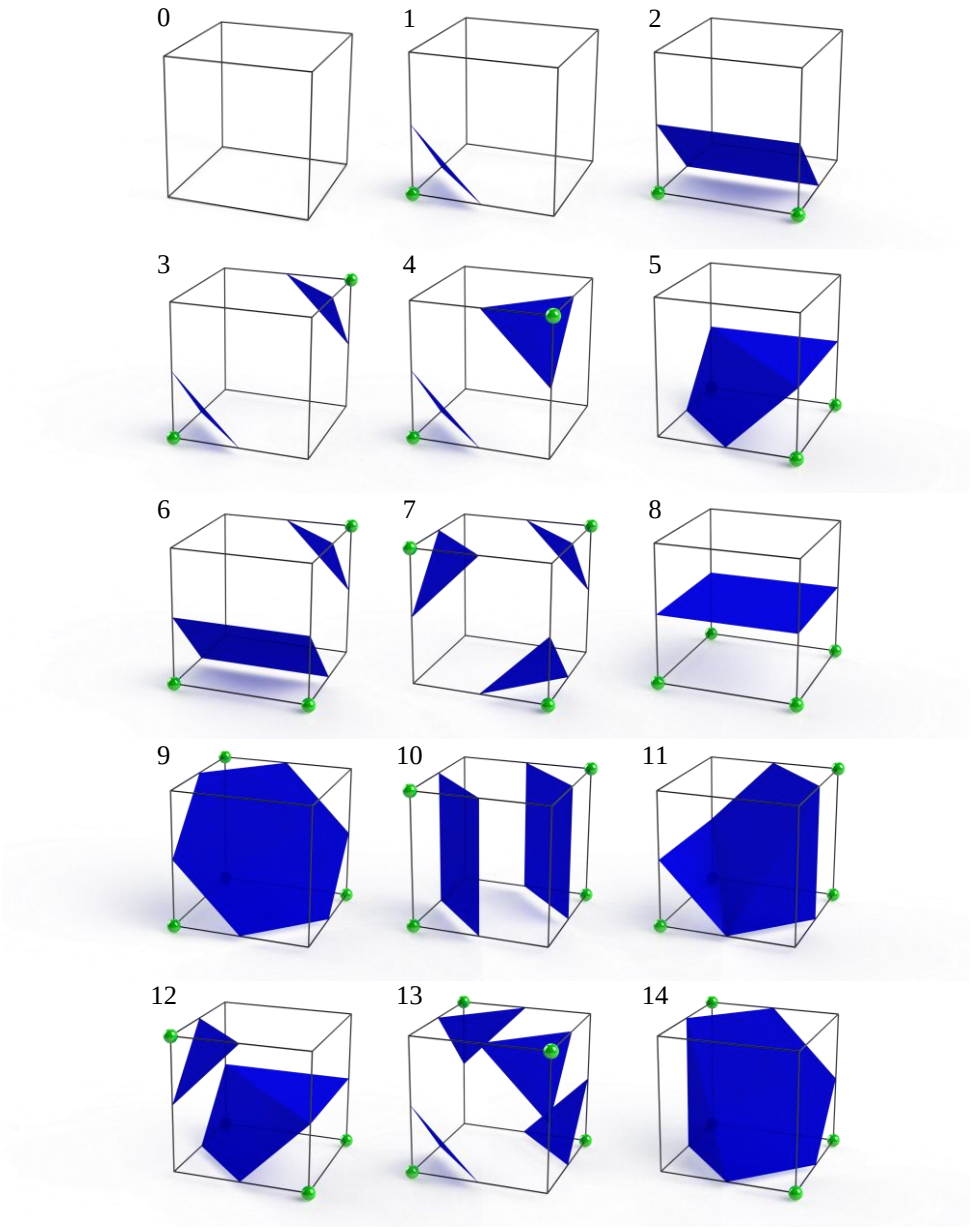

Figure 2.5: The 15 unique triangulation cases in the marching cubes algorithm. The green spheres denote a voxel intensity above the threshold, while the vertices without spheres denote a voxel below the threshold (adapted from Hansen and Johnson, 2005; Lorensen and Cline, 1987).

It is important to note that the mapping between the index and the 15 possible triangulations (typically implemented in code as a lookup table) only provides information on which edges of the triangle vertices occur. The location of the vertex on the edge is not known at this point. To calculate where the vertex lies on the edge, a linear interpolation is used. A different interpolation algorithm can be used, but Lorensen and Cline (1987) found no significant improvement in visual quality using higher order interpolations.

Finally, the marching cubes algorithm calculates the normal vector of each vertex, which is required for traditional shading algorithms. To find the vertex normal, the gradient of each voxel is calculated using the central difference method:

$$G_x(i,j,k) = \frac{D(i+1,j,k) - D(i-1,j,k)}{\Delta x} \tag{2.26}$$

$$G_y(i,j,k) = \frac{D(i,j+1,k) - D(i,j-1,k)}{\Delta y} \tag{2.27}$$

$$G_z(i,j,k) = \frac{D(i,j,k+1) - D(i,j,k-1)}{\Delta z} \tag{2.28}$$

where $G(i,j,k)$ is the gradient and $D(i,j,k)$ is the intensity of the voxel located at the position i, j, k and Δx, Δy, Δz are the x, y- and z-distances between the sampled points, respectively.

As can be seen from the algorithm, the computation of the isosurface can be extremely computationally expensive for large volumes. Additionally, the generated isosurface can contain a large number of polygons when extracted from large volumetric data sets. To address these problems, several methods have been developed to accelerate the calculation of the surface by using less computationally expensive algorithms and by splitting work across multiple processing units. Algorithms have also been developed to reduce the number of polygons in the surface mesh, thus improving rendering performance at the possible cost of

rendering quality. A review of these methods can be found in The Visualization Handbook (Hansen and Johnson, 2005).

While the use of optimized isosurfaces can increase the performance of the volume rendering, there are several problems with isosurfacing. First, only a limited number of objects can be rendering due to the need to define and precompute the surface of each object. Secondly, if the user desires to change the surface definition, a computationally expensive recalculation is necessary, which reduces the interactivity of the application. Finally, isosurfaces do a poor job of describing surfaces which vary smoothly (Meissner et al., 2000).

2.3.3 Direct Volume Rendering (DVR)

Unlike indirect volume rendering, direct volume rendering produces a view of the volume without using any intermediate geometry. This allows the entire volume to be rendered, with internal features made visible by applying a transfer function to make parts of the volume transparent. Even though DVR does not create intermediate geometry, it is still possible to render surfaces extracted from the volume. Furthermore, there is evidence that DVR can display higher quality surfaces than with an isosurface algorithm because DVR allows for a range of intensities to be included in the surface, instead of a single value (Hopper et al., 2000; Levoy, 1988). However, due to the amount of data DVR algorithms have to process, they come at a very high computational cost. To mitigate the computational cost, DVR algorithms have been increasingly designed to run on GPUs, which have more raw computational power than CPUs, albeit at the cost of a more restricted programming model.

2.3.3.1 Texture-Based Rendering

The first method to accelerate direct volume rendering on dedicated graphics hardware was texture-based volume rendering. Texture-based rendering works by rendering a plane for each slice in the volume and applying a texture to it, which is the extracted slice for that plane (Engel et al., 2006). Once the planes are extracted and textured, they can be rendered using the traditional computer graphics rendering pipeline, described in Section 2.3.1. The details of how the planes are set up can vary, but there are two basic methods. The first method is the object-aligned method. In this method, the planes are aligned with the axes of the volume, thus reducing the need to interpolate data. The problem with this approach is that when the volume is turned past 45 degrees along an axis, the user can begin to see between the planes, causing unwanted artifacts. To remedy this, the rendering engine must change the orientation of the planes to align with a different axis. To complete this reorientation step, the new planes either need to be precomputed and stored in memory (which results in the entire volume being stored in memory three times) or the new planes can be computed on the fly when the switch occurs, which has the potential to create a noticeable delay in the rendering.

The second method is the image-aligned method. In this method, multiple planes are stacked parallel to the image being rendered and the textures are interpolated from the volume data onto the planes. While this approach produces a higher quality rendering, the need to recompute the slice planes every time the view is changed can degrade performance. Furthermore, both image-aligned and axis-aligned texture rendering suffer from the problem that they must render every voxel, whether it is important or not. In most volumetric data

sets, a large number of voxels are removed by setting them to transparent, and thus do not need to be rendered.

2.3.3.2 Splatting

Splatting is one of the oldest methods of direct volume rendering (Westover, 1990). While most other volume rendering methods consider what happens to a ray coming from the screen through the volume, splatting takes the opposite approach. In splatting, every voxel in the volume is projected from the volume onto the screen. If the voxels are projected as an infinitesimal point, inevitably, most of the projected voxels will fall between pixels on the screen. Therefore, each voxel is considered to occupy a finite volume in space with the value of the voxel decreasing as the distance from the center of the voxel increases. This estimation of a voxel is known as a 3D basis function kernel, and can take the form of any statistical distribution, but the most commonly used is a 3D Gaussian kernel (Hansen and Johnson, 2005). Because splatting is projecting 3D information onto the 2D image plane, this kernel can be preintegrated in one direction, resulting in a footprint that mathematically describes how the voxels value will be distributed across the pixels of the image plane. Due to the radial symmetry of the 3D Gaussian kernel, this footprint is the same irrespective of volume orientation when an orthographic projection is used (Westover, 1990). To implement a perspective projection, the kernel has to be integrated at each distance from the camera. However, the change in sampling frequency from front to back in perspective projection can cause aliasing in the image. To prevent this, Zwicker et al. (2001) proposed using an elliptical weighted average basis function, which widens the basis function (along an axis perpendicular to the ray to the camera) to maintain a consistent sampling rate.

Splatting is capable of producing very high quality renderings. Furthermore, because it evaluates the volume on a voxel-by-voxel basis, it is easy to implement an algorithm that runs no calculations on voxels that the user has selected not to render (based on their value or spatial position). Such an algorithm is known as empty space leaping, and while it is not unique to splatting, it is easiest to implement in splatting algorithms. Due to this acceleration, splatting works best when there are a relatively few number of voxels of interest compared to the number of pixels in the rendered image (Meissner et al., 2000). The downside to this approach is that some modern acceleration approaches, such as early ray termination (which ends the computation of the volume rendering integral for a given pixel in the projection after it has exceeded a preset level of opacity), do not fit into the splatting framework. Thus, when highly opaque renderings are desired, splatting often requires significantly more calculations than other algorithms.

2.3.3.3 Shear-Warping

Shear-warping is related to texture based rendering, and is recognized as one of the fastest rendering methods available (Meissner et al., 2000). Shear-warping was originally proposed by Lacroute and Levoy (1994) and operates on the theory that the 3D view transform can be broken into a 1D shear operation and a 2D warping operation (Hansen and Johnson, 2005). To achieve this rendering, the volume is represented as a stack of slices, with the slices perpendicular to the closest axis to the viewing ray. The slices are then sheared by translating them, individually, in progressively greater amounts going back through the volume. Next, this stack of sheared-slices is composited onto an intermediate image plane at the front of the volume. Finally, the composite image is warped to create the final, view correct image. This process is shown in Figure 2.6. Note, this procedure is

specifically for an orthographic projection. If a perspective projection is desired, the slices need to be scaled so they get smaller as the slices get further away from the screen.

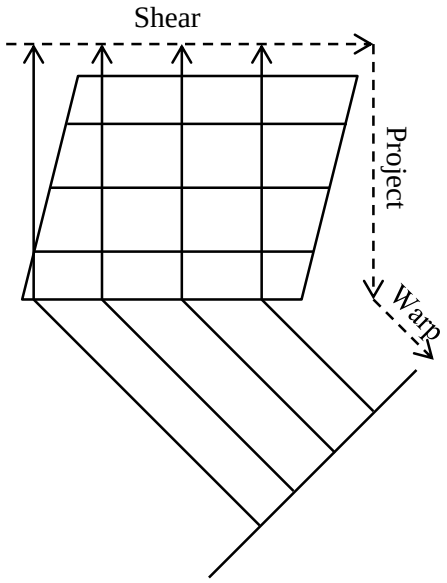

Figure 2.6: The process of shear-warping as viewed from above for the case of orthographic projection (adapted from Hansen and Johnson, 2005).

While shear-warp rendering is extremely fast, it suffers from the same slice realignment problem as texture based rendering. Additionally, while empty space skipping can be efficiently implemented in shear-warping by using run-length encoding (a compression algorithm that stores the number of times a value, in this case an empty voxel, repeats instead of each data point), early ray termination is not available. Finally, while the quality of shear-warp rendering is good, it is difficult to implement advanced rendering techniques, such as specular reflection, within the shear-warp framework.

2.3.3.4 Ray Casting

Generally considered the highest quality, as well as the slowest, volume rendering method, volume ray casting is the area of most current research (Gobbetti et al., 2008; Knoll et al., 2009; Lux and Fröhlich, 2009). Volume ray casting works by generating a cube,

which bounds the volume. A special texture (shown in Figure 2.7) is mapped onto the front and back surfaces of the box, from which it is simple to calculate the vector of the direction a ray travels through the volume (Krüger and Westermann, 2003). From this vector, a ray is produced which accumulates the color of the volume along that line by incrementally stepping through the volume. At each step in the volume, the voxel value is interpolated for that point (along with the gradient, if necessary for rendering reflections). The interpolated value is then colored per the transfer function and composited. Ray casting may use either a front to back or a back to front compositing method; however, front to back is the most common as it permits the implementation of early ray termination (Hansen and Johnson, 2005).

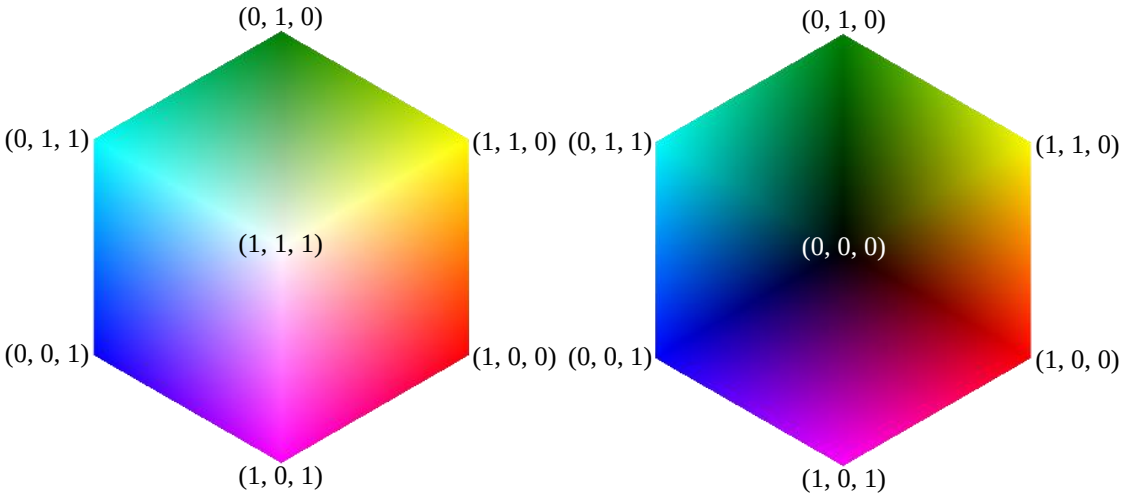

Figure 2.7: The cube texture used to generate rays in ray tracing. Each vertex has the same color as its position, allowing the interpolated color value to represent the start or end position of the ray (adapted from Krüger and Westermann, 2003).

One of the key problems with volume ray casting is that in order to get a quality projection, very small step sizes should be used (at least the inter-voxel spacing, per the Nyquist criteria). In practice, larger step sizes can produce satisfactory results, depending on how densely the volume was originally sampled. Furthermore, a small amount of noise may be introduced into the starting position of the rays (a process known as jittering) to reduce

artifacts from low sampling frequencies (Engel et al., 2006). Finally, ray casting has the advantage that empty spaces in the volume can be detected and skipped, as well as the ability to terminate rays early if the opacity of the ray is sufficiently high. However, the implementation of empty space leaping in ray casting is more difficult than it is in splatting.

2.3.3.5 Fourier Rendering

Fourier (or frequency domain) volume rendering is one of the least used but quickest forms of volume rendering. All other forms of volume rendering have a computational cost on the order of $O(M^3)$, whereas Fourier volume rendering has a cost on the order of $O(M^2 \log M)$ (Hansen and Johnson, 2005). This efficiency is achieved through the use of the Fourier projection-slice theorem. The Fourier projection-slice theorem states that, for an X-ray projection of an object, the inverse two-dimensional Fourier transform of a plane, parallel to the viewing plane, which passes through the origin of the three-dimensional Fourier transform of the object, is the same as the projected slice if it had been calculated in the spatial domain, such as by volume ray casting. Through the use of fast Fourier transforms, this inverse Fourier transform calculation becomes computationally efficient. However, the Fourier transform requires a complex input. Therefore, given that the result of a CT scan will always be a real valued data set, the algorithm may substitute the fast Hartley transform in place of the fast Fourier transform (Totsuka & Levoy, 1993).

Despite its computational efficiency, Fourier volume rendering has several problems, which has limited its adoption. First, it is difficult to apply transfer functions and clipping planes to the volume, generally requiring the initial three-dimensional domain transformation to be recalculated, a very expensive operation. Second, the Fourier projection-slice theorem only holds true for parallel rays, meaning it only produces orthographic projections, not

perspective projections, which can cause perceptual issues when rendered in a stereoscopic, virtual reality environment. Finally, Fourier volume rendering only works for an X-ray projection, meaning that all voxels will be summed into the final image, eliminating the ability to use opacity to selectively show and hide voxels.

2.4 User Interaction in Virtual Reality (VR)

While volume rendering is the most visible piece of visualizing experimental data in virtual reality, it cannot provide greater insight into the data without methods for efficient, intuitive user interaction. To achieve effective 3D interaction, three components need to work together: the display, the input device(s), and the interaction task design. These will be reviewed in this section, with special consideration given to their relevance to volumetric data visualization.

2.4.1 Display Devices

The most important display devices for data visualization are visual display devices. However, in VR it is common to find a wide range of display devices used to provide information to the user, such as haptic devices to display forces and tactile sensations or speaker arrays to provide a 3D aural display (Sherman and Craig, 2003).

The simplest type of visual display used in VR is called a fishtank VR system. In a fishtank VR system, a computer monitor or small projection screen is used to provide visuals to the user (Bowman et al., 2004). These monitors typically support rendering stereoscopic visuals to provide the user with the illusion of 3D. The key difference between a standard computer and a fishtank VR system is that a tracking system is used to determine the user's head position relative to the display. The head position is then used to recompute the visual display image to provide first person visual interaction. Fishtank displays are advantageous

because of their relative simplicity and low cost. However, their low field of regard (the amount of angular space around the user, regardless of where the user happens to be looking, that is filled with visuals) causes fishtank VR systems to be less immersive than other VR systems (Sherman and Craig, 2003). Despite this limitation, fishtank VR systems have been found to be useful for tasks where the user is outside the data looking in (Demiralp et al., 2006).

One display type that solves the limited field of regard problem is the head-mounted display (HMD). In a HMD, the visuals are displayed on small screens in front of each eye—like the lenses on a pair of glasses. Since the screens are so close to the eye, optics placed between the eye and the screen allow the eye to focus on the visuals properly. Because the displays are affixed to the head, they move with the head and provide visuals in whatever direction the user happens to be looking (a 360 degree field of regard). This requires that the HMD also include head tracking so the visual may be updated in accordance with the users head movement. For this reason, generally, only HMDs with head tracking are considered to provide virtual reality. Additionally, in most HMDs only computer generated imagery can be seen (see-through HMDs used for augmented reality are the exception to this), allowing a virtual environment to be displayed without any occlusion from real objects, such as the users body (Bowman et al., 2004). However, the inability to see real objects can result in a feeling of disembodiment. Furthermore, this complete occlusion of the real world also precludes collaborators from interacting with the HMD user in an immersive manner unless each collaborator had their own HMD and the movements of all the collaborators are tracked and rendered in the virtual environment. Additionally, HMDs typically have a limited field of view (the angular area the users can see without moving his head) which can lead to a

feeling of "tunnel vision" in users. HMDs are also more sensitive to lag in the tracking system, as the user's view is entirely dependent on what the HMD renders. Latency in the tracker can cause conflicts between the user's vestibular system and visual system and lead to motion sickness (Sherman and Craig, 2003).

The final common class of visual displays are surround screen projection VR systems, also known by the brand name CAVE, which is a recursive acronym for CAVE automatic virtual environment (Bowman et al., 2004). In a surround screen VR system, the graphics are displayed on projected walls (and optionally a floor and/or ceiling) and the user is free to walk around in the space enclosed by the walls (Cruz-Neira et al., 1993b). This design allows a wide field of regard (up to 360 degrees, depending on the specific design) while reducing the sensitivity of the user to lag. In an HMD, when the user's head rotates, the HMD needs to draw a new image for the new view. In surround screen VR systems, the imagery for all rotational views is displayed on the walls already, assuming the availability of a wall on which to display it, so the user simply has to rotate his head to look at it. However, this is only true for rotations; any translations in the user's position will still be susceptible to lag in the system. Furthermore, surround screen VR systems are good for collaboration because multiple users can see what the head-tracked user is looking at and any nonverbal communication (such as pointing) that the other users are making. However, because most surround screen VR systems only support rendering one viewpoint, only the user who is tracked will see the correct viewpoint. Non-tracked users are able to see imagery, but it will be increasingly distorted the further they get from the head-tracked user. However, the biggest drawback of CAVEs is their size. The interior of a typical CAVE is the size of a

small room and the space (including vertical space if a floor or ceiling is used) required around it for the projection system is significant.

2.4.2 Input Devices

Irrespective of how the visuals are displayed, one or more input devices are necessary to achieve useful, immersive virtual reality. Unlike desktop computing, where interaction is achieved primarily with a keyboard and mouse, the variety of input devices for VR is enormous and often task specific. Furthermore, it is common for a user in VR to interact from a standing position and to be mobile in the space. Due to this, it is important that the device be ergonomic to use and not present any encumbrances (limitations to the freedom of movement) to the user.

One input device commonly found in VR systems is a tracker. The tracker provides the computer with the current position and orientation of one or more tracking targets. Nearly all virtual reality systems track the position and orientation of the user's head and use this information to update the graphics accordingly. Trackers can also be used to track the position of another input device, such as a glove or a wand, so the system can also locate the input device in space. There are a multitude of different physical principles on which trackers can operate, all with their unique positives and negatives.

The simplest type of tracking system is mechanical trackers. Mechanical trackers physically attach the tracked object to a fixed position through a series of linkages. At the joints of these linkages are encoders that detect their motion and use that information to calculate the position of the tracked object. While mechanical tracking systems are high precision and low latency, they are cumbersome to use and limit the tracked object's range of

motion. Due to this, mechanical trackers have mostly fallen out of use, with the exception of haptics devices that require mechanical linkages to provide force feedback anyway.

Another common type of tracking is optical tracking. Optical tracking uses one or more cameras (typically infrared cameras to avoid interference from changes in the ambient light conditions) and tracking markers to determine an object's position. As each camera is only capable of measuring positions in two spatial dimensions, at least two cameras are required to achieve 3D position tracking and at least three cameras are required for 3D position and orientation tracking. Optical trackers also require tracking markers. Tracking markers can either be optically reflective to return light to the camera or they can be active markers that emit their own light using small light emitting diodes. There are also inside-out and outside-in versions of optical tracking. In the inside-out variants, cameras are attached to the point to be tracked, and the tracking markers are fixed in space. By determining which tracking markers the cameras can see, and the markers position in the image, the position of the tracked point can be determined (Welch et al., 1999). Outside-in systems use fixed cameras and attach the tracking markers to the tracked point. In general, optical trackers are high precision, but require line of sight between the tracking markers and the cameras.

A similar system to optical tracking is ultrasonic tracking. Ultrasonic tracking uses a series of ultrasonic emitters and microphones. By mounting emitters around the space and emitting an ultrasonic signal at intervals the microphones can triangulate where they are located in space. Similar to optical tracking, ultrasonic tracking requires at least three microphones and three receivers to achieve full 3D position and orientation tracking. Ultrasonic trackers are accurate and not affected by the line-of-sight issue that optical

trackers have. However, there is a minimum required distance between microphones, making the receiver units larger than other forms of tracking.

A fourth type of tracking is electromagnetic tracking. Electromagnetic tracking uses electromagnetic coils to generate magnetic fields. By aligning the coils in different directions, fields of different orientations can be generated, and in turn measured by a second set of electromagnetic coils in the receiver unit. Using three coils in the emitter and three in the receiver, electromagnetic trackers are capable of measuring 3D position and orientation. However, due to the rapid reduction in electromagnetic field strength with distance from the emitter, electromagnetic trackers have a limited useful range. Furthermore, any metal in the tracked volume can distort the electromagnetic fields and reduce the accuracy of the tracker.

The final common form of tracker in VR is the inertial tracker. Inertial trackers use accelerometers and gyroscopes to measure the relative movement of an object. Unlike other trackers, the entire tracking system can be affixed to the tracked object. This allows inertial trackers to track a much larger area than other trackers. However, because inertial trackers are only capable of measuring relative movements, any error in the measurements accumulates and can eventually result in significant errors in the absolute position of the tracked point. To combat this, inertial trackers are sometimes combined with an absolute position tracker (such as an ultrasonic tracker) to compensate for error accumulation in the inertial tracker.

Another common input device in virtual reality is the wand. Like many input devices in VR, a wand supports different input modes in one device. Most wands, like the Intersense IS-900 wand shown in Figure 2.8, combine buttons, a joystick, and a tracking target. This allows the user to complete multiple tasks using a single device. For example, a wand

equipped with tracking allows a user to intersect a virtual object and then select the intersected object for manipulation using a button. A different button could then be used to allow the user to draw a 3D line along the path the wand travels. However, remembering the mappings of the buttons can be challenging if too many functions are used and if there are no affordances indicating what button operates what function (Bowman et al., 2004).

Figure 2.8: One type of wand (the Intersense IS-900) used to interact with virtual reality.

Another tool commonly used in VR is the data glove. There are two types of information that can be provided by a data glove, and depending on the design of a specific glove, it may provide either or both. The first type of information a data glove can provide is an analog value indicating the degree of bend of the user's fingers. Other gloves are designed to provide discrete events indicating if a user's fingers are touching each other, and if so, which fingers are touching. When combined with a tracker, data gloves can provide a natural method of interacting with VR. If a user wants to pick up an object, the user moves the data glove (presumable affixed to the user's hand) so it intersects the virtual object and then closes his hand to indicate to the computer that the object is to be selected. While this interaction is not completely realistic, as most data gloves are not capable of providing any tactile feedback to the user about the selected object, it is still an intuitive method of interacting with the virtual world. Data gloves can also be used to achieve more abstract

interaction. For example, pressing two fingers together could be used to start an animation, and a different pair of fingers could be pressed together to pause the animation. The primary drawback to data gloves is the time to set up the glove for each user. In addition to the time it takes to put the glove on (which on its own may not be substantial, but if combined with multiple other devices could become tedious), the bend sensors in the glove typically need to be calibrated for each user's hand (Bowman et al., 2004).

From an encumbrance standpoint, one of the most promising input devices is the microphone, which supports voice recognition input. By using a wide area microphone, the computer can monitor the user's speech without the user wearing a local microphone (although a local microphone for the user has some advantages). Using speech recognition, the user is able to tell the computer what to do, instead of trying to remember a button mapping (Otaduy et al., 2009). However, due to the limitations in speech recognition software, the computer may or may not know how to interpret what the user is telling it. Furthermore, the processing required for speech recognition causes a delay between the command and the result, making speech recognition unsuitable for interactions where precise timing is required. Finally, speech recognition can generate false recognitions due to unrelated conversation within earshot of the microphone (Mrvaljevic and Sun, 2009). To reduce the likelihood of a false recognition, methods, such as using a push-to-talk button or speaking a specific initiation command, have been developed (Sherman and Craig, 2003).

Another input device that has been gaining popularity recently is the camera. Like voice recognition, cameras do not require the user to wear the input device. Furthermore, when cameras are combined with pattern recognition software, the system is able to identify gestural events occurring in the frame (Rigoll et al., 1997). In addition to gesture

recognition, recent advances in cameras have led to cameras than can sense an object's depth from the camera, such as the Microsoft Kinect. These systems can be further enhanced with software that can estimate the pose of a user's body from the depth image (Shotton et al., 2011). This allows the camera to act as a low-accuracy, markerless tracking system. Similar to voice recognition, cameras have a substantial latency due to the image processing necessary to turn the raw data into useable information, and thus are not suitable for tasks that require precise timing. Furthermore, they can be easily fooled if the user becomes occluded by another object in the physical space.

A final input device of increasing popularity is the mobile device, often a smartphone or tablet running custom software (Kim et al., 2009). Mobile devices collect multiple sensors (typically at least a microphone, touchscreen, and accelerometer) into one pre-engineered, handheld, ergonomic package. The touchscreen on a mobile device is also coupled with a display screen that can provide feedback specifically about the interaction task. Furthermore, the existence of a physical surface reduces the degrees of freedom of the interaction, which can make some tasks easier to accomplish. However, holding and interacting with a mobile device for a long period of time can be tiresome and there is evidence to show that secondary display screens may reduce immersion (Fu et al., 2010).

When choosing an input device for a specific VR environment it is important to consider a few factors. First, the type of data the device provides and how that maps to the desired task. For example, in selecting an object, a button could be used to cycle through all objects or a wand could be used to intersect the desired object. Depending on the number and size of objects in the virtual environment, and the degree of realism required, either solution could prove to be the best available. Second, the ergonomics of the system are important. For

example, if text input is required, a traditional keyboard could be used on a fishtank VR system where the user is sitting down. However, in a CAVE the user is traditionally standing up, making the use of a keyboard difficult without a freestanding support for the keyboard (which could limit the user's mobility). It is because of these challenges that specialized input devices are often designed and built for specific tasks in VR (Bowman et al., 2004).

2.4.3 Interaction Tasks

Interaction tasks can be grouped into four areas: 1) selection and manipulation, 2) travel and wayfinding, 3) system control, and 4) symbolic input (Bowman et al., 2004). Using current volumetric data visualization tools, the user is typically looking at a dataset from the outside, looking in. In order to understand the dataset, the user will view it from different angles, and change the parameters of how it is rendered to show different information. These tasks fall under manipulation and system control, respectively.

2.4.3.1 Selection and Manipulation

The ability to manipulate the viewpoint of a volumetric data set interactively may be the most important user interaction in data visualization. The depth cues provided to the viewer are extremely important for their understanding of the three-dimensional structures they are viewing (Zhang et al., 2001). Because volumetric data is traditionally viewed from the outside in, the simplest method of changing the viewpoint is for the user to physically move around the volume (He et al., 2007). This is a natural interaction and helps increase both immersion and understanding (Haubner et al., 1997). However, because of the fatigue involved in doing this repetitively, users often prefer to have a secondary method to manipulate the data. Previously proposed methods for manipulation include using a wand or

tracked glove to intersect, pick up, and manipulate the object, or using a joystick to change the object's positioning (Bowman et al., 2004)

2.4.3.2 Travel and Wayfinding

Travel and wayfinding are two interrelated tasks. Travel consists of the mechanics of moving from place to place, while wayfinding is the task of understanding where one is in the environment and how to get to a designed location. In general, the most natural method of travel is the physical motion of the user (Bowman et al., 2004). However, time, space, and fatigue constraints often prevent the user from performing all travel using physical motion, thus other methods must be considered. One common method is walking in place. By tracking the movement of the user's feet, the system can estimate how far and in what direction the user is moving in the virtual world. However, this is less realistic than physical movement, and is just as tiring to the user. Another method of travel is pointing. In this method the user indicates where the user would like to go by pointing a tracked object in the desired direction of motion. To move, the user then presses a button, and moves in a constant velocity in the pointed direction. This method is not fatiguing, however, if both large and small adjustments to the user's position are required, the use of a fixed velocity can be problematic. A final method of travel is the world in miniature technique (Stoakley et al., 1995). In the world in miniature technique, a small map or 3D representation of the virtual world is displayed to the user. The user can then move to a new position by selecting the position on the miniature map. The user is then transported to the new position. While the use of a miniature world helps the user understand the surrounds, the virtual transportation can be a somewhat jarring experience that reduces the immersion of the system.

2.4.3.3 System Control

While system control is a very broad category that can include everything from opening a file to changing the color of a pointer, there are two system control tasks of particular importance in volumetric data visualization: volume slicing and transfer function manipulation. Both of these tasks are ways of selectively removing extraneous data from the rendered volume so the user can focus on a selected point or points of interest.

One method that has been proposed for applying clipping planes is to track the user's hands. When the user wants to add a clipping plane, the user can simply draw the clipping planes needed to eliminate unwanted geometry (He et al., 2007). While this method is intuitive, it requires the location of the desired clipping plane to be within the physical reach of the user. Furthermore, having one's arms extended for a long period of time is tiresome (Bowman et al., 2004). One solution to this is to us a miniature representation of the data as a physical prop, and a second tracked prop to represent the clipping plane (Sherman and Craig, 2003). This is especially useful when there is a known geometric bounding to the data, such as the visualization of a MRI of the human brain, because the prop can show the bounding in miniature and help keep the user oriented in the data.

The second system control task of particular importance in volumetric data visualization is the manipulation of transfer functions. The transfer function maps the properties of a voxel (typically intensity, but intensity gradients can also be used) to a specific color and opacity. This allows the user to control the appearance of a volume and selectively hide irrelevant information. However, transfer functions are extremely challenging to implement in VR because they require precise values to be set and, despite decades of research into image processing, the most prevalent way of defining a transfer function remains trial and

error (Pfister et al., 2001). In previous work, transfer functions have been adjusted using onscreen menus (He et al., 2007). While this is effective when a limited number of predefined transfer functions are available, if a new transfer function is desired, it would be a tedious task to create one using just a 3D menu. Because the setting of a transfer function is not an inherently 3D task, it is a good candidate for reducing the degrees of freedom available to the user while setting it. While this could be achieved in many ways, the simplest is to provide a physical object for the user to interact on. This method has never, to the author's knowledge, been tried with transfer functions; however, small touch screen devices have been found to be successful at providing limited degree of freedom system control in VR (Bowman et al., 2004).

2.4.3.4 Symbolic Input

Symbolic input tasks are those that convert some input into a set of symbols to be stored in the computer. The most common symbolic input task is typing on a keyboard to generate text. However, keyboards work best when placed on a desk, an option typically not available in VR. In some cases, the intent of symbolic input can be achieved without actually generating the symbols. For example, if a user wants to annotate an interesting feature in a volume, the user could mark the position and then record an audio annotation or write the annotation in digital ink on a touchscreen. While there are algorithms available to convert both audio and digital ink to text, it is typically more efficient to leave the annotation in its raw form for a human to decipher later. When precise symbols need to be input, there are several modified forms of keyboards designed to be handheld, such as chord keyboards and soft thumb keyboards on mobile devices (Bowman et al., 2004). However, as the number of

symbols increase, so does the difficulty in remembering key functions and the challenges in creating an ergonomic device.

2.4.4 Data Visualization in Virtual Reality

The use of virtual reality to visualize scientific data not only has a long history, the visualization of scientific data is one of the driving forces in advancing the science of virtual reality (Brooks, 1999). In fact, the first surround screen projection VR system was designed specifically for scientific data visualization (Cruz-Neira et al., 1993b). Virtual reality has the advantage of being able to display data in a way that is natural to the users, while not being constrained by the laws of physics (Cruz-Neira et al., 1993a; van Dam et al., 2000).

In the field of fluid flows, this has often involved the rendering of simulated flows. For example, a famous early virtual reality application was the virtual wind tunnel. The virtual wind tunnel allowed users to look at simulated flow data over a model of the space shuttle using streamlines, pressure maps, and other classic flow visualization tools, except in an immersive 3D environment (Bryson, 1996).

While volumetric data is used in flow visualization, it has more often been coupled with virtual reality in the context of medical visualizations. For example, volumetric MRI data has been examined in virtual reality to assist in the understanding of brain function (Chen et al., 2011). Volume rendering in VR has also found use in surgical training and planning (Robb, 2008).

2.5 Summary

A review of the literature shows that noninvasive measurement is an important tool for the study of multiphase flow. However, there are no methods currently available that provide both high spatial and temporal resolution. Furthermore, even if such a method did exist, it

would be limited by the currently available volumetric rendering tools, which provide limited 3D information and user immersion. The aim of this research is to enhance the understanding of multiphase flows by improving the data acquisition, processing, and visualization. These improvements will ultimately allow researchers greater insight into a wide range of multiphase flows.

CHAPTER 3:

METHODS

The completion of this research is possible because of two key facilities at Iowa State University. The first is the X-ray Flow Visualization facility. Completed in 2003, this one-of-a-kind facility was designed specifically for characterizing fluid flows using X-rays. The details of this facility can be found in Section 3.1. The second crucial facility used in this work is the Multimodal Experience Testbed and Laboratory (METaL). METaL is a CAVE automatic virtual environment (CAVE), designed for experimental studies in virtual assembly, and is conveniently co-located in the same laboratory with the X-ray Flow Visualization facility. Details on the Multimodal Experience Testbed and Laboratory can be found in Section 3.2.

3.1 X-ray Flow Measurement

The X-ray measurement portion of this work will be completed using the X-ray Flow Visualization (XFloViz) facility at Iowa State University. The XFloViz facility is designed specifically for multiphase flow measurements using X-rays and is capable of three different types of X-ray measurements: radiography, stereography, and computed tomography (Heindel et al., 2008; Hubers, 2005; Striegel, 2005). To obtain all these measurements, the XFloViz facility has at its disposal two Lorad LPX 200 liquid-cooled, tube X-ray sources for X-ray generation and three X-ray detectors: one scintillator and two image intensifiers. The detectors are mounted on sliding rails, allowing for easy interchange of detectors. The entire source-detector setup is mounted on a slew ring to provide 360° rotation around the object of interest. The object of interest can also be moved vertically using a 910 kg (2000 lbs.)

vertical lift with 2.75 m (9 ft.) of travel. Finally, to protect the operators, the entire imaging chamber is encased with approximately 8164 kg (9 tons.) of lead shielding. A diagram of the XFloViz, with the lead shielding removed for clarity, is provided in Figure 3.1.

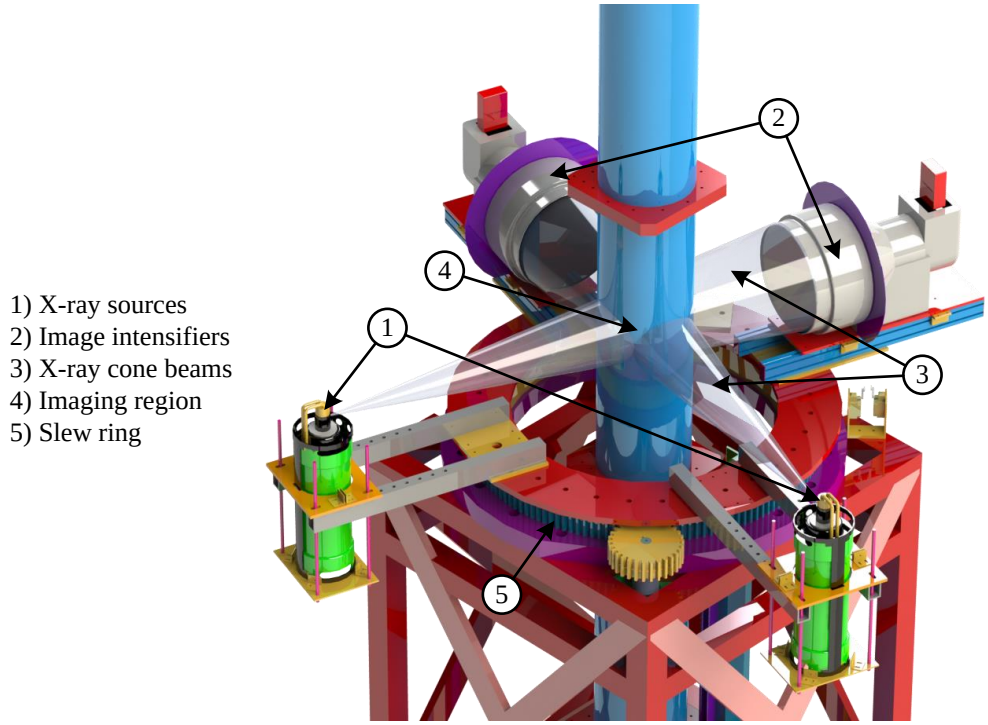

Figure 3.1: Schematic of the X-ray Flow Visualization facility's imaging equipment.

Each LPX 200 X-ray source has a 1.5 mm focal spot with a beryllium output window providing a 60° horizontal and 40° vertical conical X-ray beam. Each source is capable of producing tube potentials from 20 to 200 kV and currents from 0.1 to 10.0 mA. Due to the polychromatic nature of tube X-ray sources, external filters are used (typically aluminum or copper) to reduce the lower energy radiation. The two sources, and their respective X-ray detectors, are at 90 degrees from one another to allow the 3D positions to be easily determined from the X-ray images created by the two source-detector pairs.

The first type of X-ray detector is an X-ray image intensifier, also referred to as an intensifier. The XFloViz facility has a pair of intensifiers (one for each source), with each

intensifier containing a Precise Optics PS164X X-ray intensifier connected to a monochrome DVC-1412 charge-coupled device (CCD) camera. The detectors can be temporally synchronized to do X-ray stereography. The image intensifiers feature a 40.6 cm (16.0 in) diameter input phosphor and a 3.5 cm (1.4 in) diameter output phosphor. The input phosphor is backed by a vacuum chamber causing the X-ray photons to be re-emitted in the vacuum chamber as electrons. These electrons are accelerated and focused onto the output phosphor using high voltage electric fields. While the conversion to electrons allows the image to be intensified significantly, it also introduces a warping artifact due to external magnetic fields altering the path of the electrons. This artifact must be digitally corrected for, as explained in Section 3.1.2.1. The DVC-1412 cameras on the intensifiers are capable of a native frame size up to 1392 × 1040 active pixels, at a maximum frame rate of 10.2 frames per second (FPS), and 12-bits of resolution depth. These cameras are also capable of binning. When binning is applied, adjacent pixels on the detector are treated as one large pixel. This causes the camera to be more sensitive to light and enables higher frames rates, at the expense of spatial resolution. For example, when a 2 × 2 binning is applied, four pixels are treated as one, reducing the resolution to 696 × 520 active pixels, but increasing the frame rate to 20 FPS. The DVC-1412 cameras are capable of binning sizes of 1 × 1 (native), 1 × 2, 2 × 2, 4 × 4, and 8 × 8. When the cameras are temporally synchronized, there is a slight loss of frame rate due to the synchronization overhead. In the 2 × 2 case, this typically reduces the theoretical maximum frame rate of 20 FPS to an actual frame rate of 18 FPS. Each camera is linked to the acquisition computer by a CameraLink connection to an Engineering Design Team (EDT) PCI DV C-Link card, running in the CameraLink base mode.

The second detector available in the XFloViz facility is a single Hamamatsu Photonics cesium-iodide scintillator screen, paired with an Apogee Imaging Systems Alta U9 CCD camera via a mirror and a Nikon Nikkor 50 mm lens. This camera is capable of resolutions up to 3072 × 2048 at 16-bits of resolution depth; however, it requires several seconds between images to download data via its universal serial bus (USB) 2.0 connection with the acquisition computer. The Alta U9 is also capable of operating at numerous binning modes. The most commonly used binning modes are 1 × 1, 2 × 2, and 4 × 4. The Alta U9 camera is also equipped with a thermoelectric cooler, allowing the camera sensor to be cooled up to 50° C (90° F) below the ambient temperature. Cooling the camera sensor reduces the noise in the image. The scintillator detector is used primarily for high-resolution computed tomography where spatial resolution is more important than temporal resolution.

3.1.1 Imaging Parameters and Their Effects

To acquire a radiograph, either type of detector available at the XFloViz facility may be used, with each detector having its own distinct strengths and weaknesses. The intensifier detectors' primary advantage is the significant intensification of the X-ray image. Due to this intensification, the detector is capable of much higher frame rates than the scintillator, making it suitable for dynamic radiography. This intensification comes at the cost of noise and distortion. The distortion, in the form of a warped image, can be corrected digitally using the algorithm in Section 3.1.2.1. In contrast to the intensifier, the scintillator is an extremely low noise, distortion free detector. However, it does so at the cost of imaging time. Since the scintillator contains no method of intensifying the relatively weak X-ray image (it is converted directly from the X-ray spectrum to the visible spectrum so it can be measured with a standard camera), relatively long exposures are necessary to achieve quality

images. This renders the scintillator useful only for time-averaged measurements of dynamic flows, and for the measurement of extremely slow phenomena.

Regardless of which detector is used, several parameters need to be adjusted for optimal imaging: source voltage, source current, camera exposure time, camera binning, camera gain, detector location, and object location. The source voltage, source current, and camera shutter speed all impact the amount of light the camera receives, with various tradeoffs for each. Increasing the source voltage yields a strong increase in the amount of light received by the detector. Not only is the total energy of the X-ray beam increased with an increase in voltage, most materials have lower X-ray attenuation coefficients at higher photon energies, yielding a strong increase in light received by the detector. The total power of the beam can also be increased by increasing the current of the X-ray source. This causes more X-ray photons to be emitted, but without changing the energy spectrum of the emitted photons. This is particularly useful in cases where the imaged object is thin and weakly X-ray attenuating. By using a low source voltage, but high source current, the higher attenuation coefficients for low energy photons can be used to increase the contrast from the background, while the high current provides sufficient X-ray energy on the detector to obtain a radiograph.

The other methods of increasing the quantity of light the camera receives have more negative costs associated with them and require careful consideration of the measurement goals to use. The exposure time of the camera increases the light on the detector by increasing the duration of light collection. In time-averaged measurements, this can be beneficial, as it optically averages the flow over a period of time. However, when numerous radiographs need to be collected (such as in the case of CT scans) increasing the exposure of

a frame also reduces the throughput of the system. In the case of high-velocity flows, exposure time often needs to be minimized to reduce motion blur, requiring the user to find a balance between the brightness of the image and the blur introduced into the image. The blur can also be reduced digitally using a deconvolution; however, such processing is beyond the scope of this work and improving the quality of the original image is always preferable, when possible (Lucy, 1974).

The next two parameters, camera binning and camera gain, adjust the brightness of the image, without changing the amount of light the image sensor receives. First, adjusting the binning of the camera adjusts the brightness of the image by increasing the light incident on each effective pixel instead of increasing the light incident on the entire sensor. It achieves this by combining multiple adjacent pixels into a single, larger pixel. This increases the brightness of the image, at the cost of image resolution. Depending on the size of the features to be imaged, this may or may not be a worthwhile tradeoff. Camera gain on the other hand is simply an analog intensification of the electrical signal on the camera's imaging sensor. While this can improve the brightness of an image, it does so at the cost of noise. Therefore, it is rarely used, and only as a method of last resort.

Finally, the last two parameters, object location and detector location, affect the magnification of the image and the penumbral blur. Since the X-ray sources in the XFloViz facility are cone-beam sources, the placement of the object in the beam, relative to the location of the detector, will change the magnification of the image on the detector. That is to say, the closer the object is to the source, the more magnified its projection on the detector. Similarly, the closer the object is to the detector, the closer to actual size the object appears on the screen. This magnification effect can be useful in the imaging of small objects;

however, it also introduces a blur due to the penumbra of the object. The assumption in the magnification is that all X-rays are emitted from a single point in space, and thus there is no penumbra. However, the actual source has a focal spot of 1.5 mm (0.06 in), meaning that the actual emission location of a single photon may have been anywhere in a 1.5 mm (0.06 in) diameter circle. This causes a blur at the edge of the image, called the penumbra. The magnitude of this effect though is also correlated to the locations of the object and the detector. The closer the object is to the detector, the lower the effect of the penumbra and vice versa.

3.1.2 X-ray Image Processing

Independent of whether the XFloViz facility is used to acquire radiography, stereography, or computed tomography images, there is always post-processing of the images to improve their utility. While the exact details of which processes are used and how they are configured varies based on the requirements of an experiment, they generally follow a consistent pattern based on the measurement type. Radiographs are typically converted to 16-bit, normalized, and then unwarped. Stereographs typically follow the same steps and, additionally, generally combine each frame from the two cameras into one image for easier visualization. In the case of computed tomography, the processing steps are typically normalization, center of rotation (COR) determination, and volume reconstruction. The bit conversion and unwarping steps are generally unnecessary because CTs are usually obtained using the Alta U9 camera, which already has a native bit-depth of 16-bits and does not suffer from the unwarping artifacts introduced by the image intensifiers on the other cameras.

All of the image processing methods for the XFloViz facility, except for the COR determination and CT reconstruction, are implemented in a custom software package known

as the X-ray Image Processor, or X-Rip. The functions of the unwarping and normalization algorithms are detailed in the following sections due to their complexity and importance to this research. Bit conversion and frame combination are not covered in this dissertation, as they are trivial operations (a multiplication by 16 and an array concatenation, respectively). Similarly, CT reconstruction is not covered due to the numerous algorithms and implementations available (covered in Section 2.2); although, it is worth noting that the fan-beam filtered backprojection algorithm is used most commonly in the XFloViz facility. Finally, COR determination is not covered in detail, as it is simply a trial and error variation of a CT reconstruction parameter, from which the user selects the best result.

3.1.2.1 Image Unwarping

Due to the nature of the intensifiers, the image is susceptible to distortion by both internal and external electromagnetic fields. To correct for this, an unwarping algorithm is applied. The unwarping algorithm was originally developed by NASA for correcting distorted images taken with vidicon tube cameras on the Mariner 6, 7, and 9 missions (O'Handley and Green, 1972). It was later updated by Haaker, et al. (1988) for use with X-ray image intensifiers and has been extended since (Doering, 1992; Striegel, 2005). The conceptual function of the algorithm is that an object with a known structure is imaged, and that distorted image is mapped to the known structure of the object using a polynomial equation. Once this equation is known, it can be applied to any image that is acquired with the same settings. For the unwarping algorithm, the equation is broken into two parts, one for the x-direction and one for the y-direction. The mapping equations are:

$$x_c = \sum_{i=0}^{3}\sum_{j=0}^{3} A_x(i,j)x_d^j y_d^i \tag{3.1}$$

$$y_c = \sum_{i=0}^{3}\sum_{j=0}^{3} A_y(i,j)x_d^j y_d^i \tag{3.2}$$

where x_c and y_c are the corrected coordinates of a given pixel, $\boldsymbol{A_x}$ and $\boldsymbol{A_y}$ are the 4 × 4 polynomial coefficient matrices, and x_d and y_d are the original, distorted position of the pixel. Note that because the warping is (in part) dependent on external electromagnetic fields, the polynomial equation for unwarping is specific to the intensifier's physical position in space and needs to be recomputed if the intensifier is moved. It should also be recomputed if a long period of time passes between when the calibration is calculated and when radiographies are taken, as the external electromagnetic fields can be transient. With this method of unwarping calibration, accuracy of ±0.5 pixels is obtainable (Doering, 1992).

For the XFloViz facility, the known structure used to generate unwarping parameters is a 1.59 mm (0.06 inch) thick stainless steel plate with 2 mm (0.08 inch) holes located in a 12.7 mm (0.50 inch) on center rectilinear grid. To obtain the unwarping parameters, the plate is attached to the front of the image intensifier. A single radiograph is then acquired using the same settings that will be used for acquiring the data later. It is recommended that the flow system be removed from the imaging before acquiring the unwarping image. However, if the flow system contains magnetic components, it must be left in place for the acquisition of the calibration parameters, as the presence of the object will influence the unwarping parameters.

Once the unwarping calibration image is acquired, the user must choose a threshold value. All pixels with an intensity above this value will be set to white and considered part of a hole in the calibration plate. All pixels with an intensity below the threshold will be set to black and be ignored. Once the threshold is applied, the white pixels are grouped into clusters using a Von Neumann neighborhood (four-connected neighborhood). After the pixels are clustered, the centroid of each cluster is computed. With the centroids computed,

every cluster is iterated through and the nearest eight clusters to it are found. Any of those eight nearest clusters that are within ±10 degrees of horizontal from the original cluster are considered to be in the same row as the original cluster. Similarly, any clusters that are within ±10 degrees of vertical from the original cluster are considered to be in the same column as the original cluster. Once all clusters have been analyzed, the algorithm has a unique row index and column index for each point.

The calculation of the row and column indices in X-Rip is a deviation from the algorithm used by Striegel (2005). In Striegel's algorithm, the user was required to select a region of interest containing a full grid of clusters, where every row had the same number of clusters as every other row and every column had the same number of clusters as every other column. This meant Streigel did not need to know which row or column a cluster was in. However, with some severely warped images, it can be difficult to achieve a region of interest with a full grid. Furthermore, because the image from the intensifier is a circular imaging region inscribed in the rectangular image array of the camera, clusters near the edge of the image had to be omitted from the region of interest to ensure that all the rows and columns were the same length and height when using Striegel's approach. Clipping the edges off resulted in poor quality unwarping outside the region of interested used to generate the parameters. In contrast, because the method used in this research knows the row index and column index of each point, no region of interest selection is required for calibration, resulting in a simpler calibration process for the user and a more accurate result.

Once the clusters are found and indexed, a second, theoretical cluster grid is created to represent where the clusters would be if there were no distortion. To do this the center of the grid is found using:

$$x_{cen} = \frac{(x_{max} + x_{min})}{2} \tag{3.3}$$

$$y_{cen} = \frac{(y_{max} + y_{min})}{2} \tag{3.4}$$

where x_{cen} and y_{cen} are the center coordinates of the grid, x_{min} and y_{min} are the minimum coordinates as measured on the distorted grid, and x_{max} and y_{max} are the maximum coordinates as measured on the distorted grid. The spacing between clusters on the theoretical grid clusters is calculated by:

$$x_{space} = y_{space} = \frac{\frac{x_{max} - x_{min}}{n_{rows}} + \frac{y_{max} - y_{min}}{n_{columns}}}{2} \tag{3.5}$$

where x_{space} and y_{space} are the on center distance between the clusters in the x and y directions, respectively, and n_{rows} and $n_{columns}$ are the total number of rows and columns, respectively. Note that the spacing in the x-direction is the same as the spacing in the y-direction because the physical grid has spacing that is the same in the x and y directions. All the coordinate positions are measured in pixels. Because the pixel indices were previously found for the distorted grid, the algorithm is able to omit clusters for which there is no data (i.e., the clusters are outside the viewable area of the camera), instead of assuming a full rectangular grid.

In order to determine the A_x and A_y of Eqs. (3.1) and (3.2), respectively, the calibration routine sets x_d and y_d to the theoretical grid points and x_c and y_c to the measured grid centroids, and solves for $\boldsymbol{A_x}$ and $\boldsymbol{A_y}$ using a curve fitting algorithm. Note that, as Eqs. (3.1) and (3.2) are written, one would expect the theoretical grid points to map to x_c and y_c instead of x_d and y_d; however, this reversed mapping makes the interpolation step simpler when an image is unwarped. When an image is unwarped, Eqs. (3.1) and (3.2) are run for every pixel in the image, using the previously calculated $\boldsymbol{A_x}$ and $\boldsymbol{A_y}$ matrices and the x and y-positions of

each pixel used for x_d and y_d. This, combined with how the calibration step was set up, causes the calculation to return the position in the distorted image (x_c and y_c) where the intensity for the corrected position (x_d and y_d) will be found. The result of this unwarping process is shown in Figure 3.2. Note that the variable assignment used is counterintuitive to how Eqs. (3.1) and (3.2) are written (indeed, the calculation could be run in the intuitive order), but because the corrected value often lies in between the integer pixel values the computer can represent, an interpolation step is required. If the unwarping algorithm is run in the intuitive order, the intensities will have to be splatted onto the representable pixels. By running the algorithm in reverse, a simple bilinear interpolation can be computed on the original image to return the appropriate value for the corrected pixel location. Finally, note that while a bilinear interpolation is commonly used, any interpolation algorithm could be used.

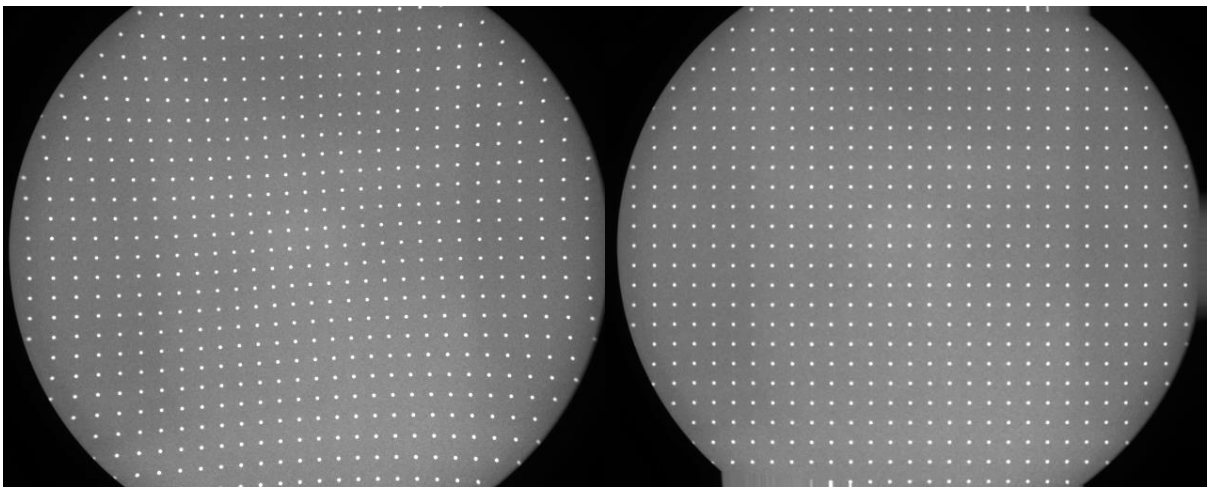

Figure 3.2: The effect of the unwarping calibration on an image. The original unwarped image of the calibration grid is shown on the left. On the right is the same image with the unwarping calibration applied.

3.1.2.2 Image Normalization

The purpose of image normalization is to compensate for any non-uniformities in the pixel response of the detector. To achieve this, two calibration images are taken, with no

object in the imaging region. These images are known as the flat and dark images. The flat image (Figure 3.3a) is taken with the X-ray source on and all X-ray parameters set as they will be during the actual test. The dark image (Figure 3.3b) is taken with the X-ray source turned off, but all other parameters the same. This provides a minimum and maximum value for each pixel, which can, in turn, be used to compensate for non-uniformities between the pixels (Figure 3.3c). Note that the flat and dark images are often an average of multiple frames to reduce random noise.

Figure 3.3: The flat (a), and dark (b) images are the inputs to the normalization algorithm. The result of a linear normalization (c) is the removal of any location dependent pixel intensity variation. Note, a flat frame has been normalized to show the result without any interference from an imaged object and the normalized image (c) has been contrast enhanced to better show the remaining noise.

X-Rip supports three algorithms for image normalization. The basic form of image normalization is a version of linear interpolation. The intensity of each pixel the image, I_{im}, is converted to a normalized intensity, I_{new}, by calculating where the original intensity lies in the original range, and then rescaling it to the new minimum and maximum values. This is achieved by the equation:

$$I_{new} = Min + (I_{im} - I_{dark}) \frac{Max - Min}{I_{flat} - I_{dark}} \tag{3.6}$$

where I_{flat} and I_{dark} are the intensities of the flat and dark image, respectively, and *Max* and *Min* are the new maximum and minimum values for the image. This equation, as with all the normalization algorithms described herein, is applied to each individual pixel, taking the intensity in the image, flat frame, and dark frame from the same location in each respective

image. In practice, *Min* is almost always 0 and *Max* is typically the maximum number that can be represented given the bit depth of the image ($2^n - 1$, where n represents the bit depth of an individual channel of the image). With this in mind, Eq (3.6) simplifies to:

$$I_{new} = (I_{im} - I_{dark}) \frac{2^n - 1}{I_{flat} - I_{dark}} \tag{3.7}$$

The one drawback to this normalization is that because it scales to the maximum representable value of the image format, random noise in the flat and dark image sometimes causes the new intensity value to exceed the maximum representable value. This causes a small loss of data when the new intensity value is clipped to maintain its value within the representable range. Therefore, in practice it is often preferable to reduce *Max* to a value below the maximum representable value and thereby prevent clipping.

The other two forms of normalization that X-Rip supports are modified forms of a linear normalization with slight variations in the assumptions made. The first of these methods is from Striegel's (2005), and was implemented in his FX Visual software. This normalization algorithm is:

$$I_{new} = (I_{im} - I_{dark}) \left(\frac{I_{ave}}{I_{flat}} \right) \tag{3.8}$$

where I_{ave} is the average intensity value in the flat image, excluding those intensities where the difference between the flat frame and dark frame intensity is less than 650 (assuming a 12-bit grayscale image). These pixels are excluded because the algorithm was designed specifically for use with X-ray image intensifiers, which leave a region in each corner of the image without X-ray illumination, and thus without any usable data. The inclusion of these inactive areas would artificially reduce the image average, and render the normalized image too dark to be useful. The specific pixel intensity used for exclusion was determined by

Striegel to be optimal for the XFloViz facility; however, in facilities using different detectors, a different value may be appropriate. This version of the normalization algorithm also assumes that the dark intensity is always zero. While this is typically a reasonable approximation, there are gain effects at some acquisition settings which do not match this assumption. Therefore, the user should be careful to ensure this assumption is valid when using this algorithm.

The final version of normalization available in X-Rip comes from the software package PS CT. This software package originated at the Center for Nondestructive Evaluation (CNDE) at Iowa State University, and is designed to acquire and process CT scans. It uses the normalization equation:

$$I_{new} = (I_{im} - I_{dark})\left(\frac{I_{ave}}{I_{flat} - I_{dark}}\right) \tag{3.9}$$

Like the FX Visual normalizations, I_{ave} is the average intensity value in the flat image; however, unlike the FX Visual normalizations, the PS CT average does not exclude any intensity values. By comparing Eqs. (3.7) and (3.9), it is easy to see that the only difference is that Eq. (3.7) uses the maximum representable intensity for the maximum value, while Eq. (3.9) uses the average intensity in the flat image. This means the PS CT normalization is significantly less likely to lose data due to clipping, but it also means that it uses less of the image's resolution depth, and thus the image has less contrast.

A comparison of all three normalization methods can be found in Figure 3.4, which shows the line intensity at row 255 of a flat field image that has been normalized with each of the three methods.. Both the linear and PS CT normalization show a good normalization of pixel nonuniformity, but it is clear that the PS CT normalization only uses about 70% of the available resolution depth, while the linear normalization occasionally surpasses the

maximum representable pixel intensity. Striegel's FX Visual normalization shows areas where the assumption of a dark value of zero is not valid around pixel position 0.5%. However, for the majority of the image width, all three methods provide acceptable results.

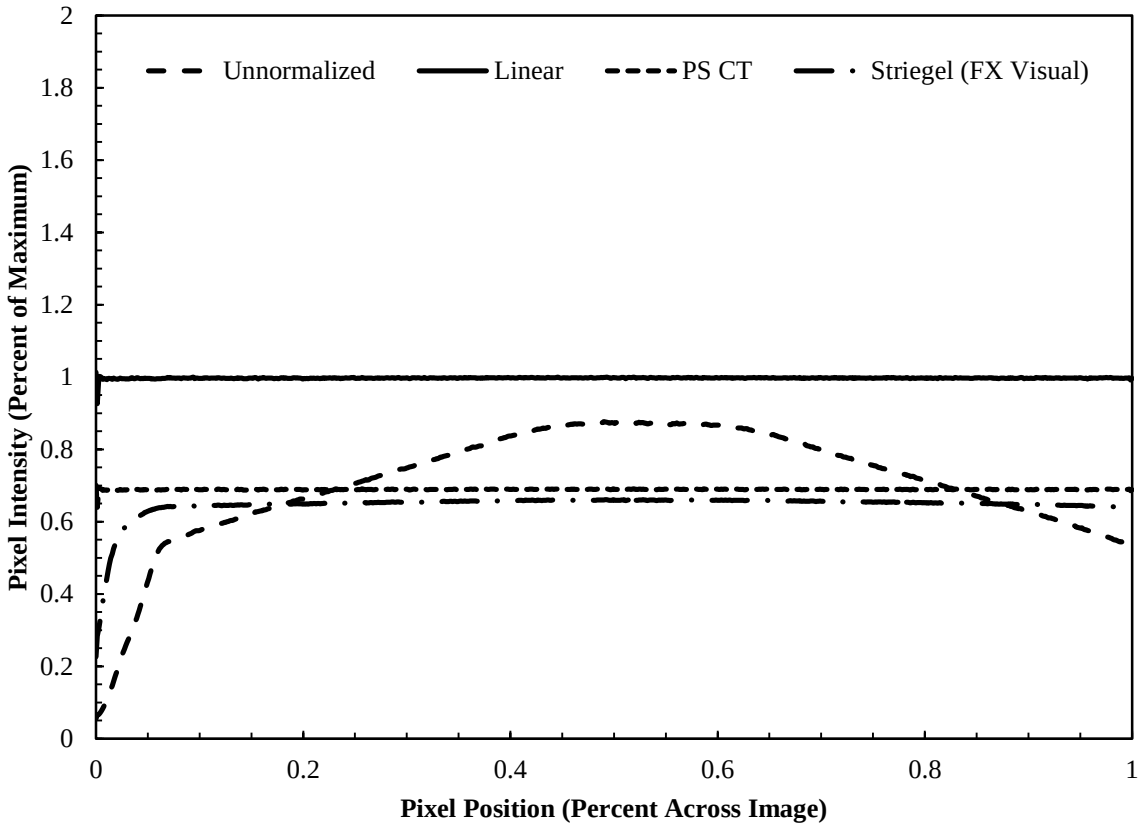

Figure 3.4: A comparison of the four available normalization methods. The data is from row 255 of a flat image, normalized using a different flat image and a dark image for the same settings. The horizontal axis is scaled to the percent of the distance a given pixel is located across the image, and the vertical axis is scaled to the percentage the intensity value is of the maximum representable intensity.

3.2 Immersive Visualization

The second portion of this work is to visualize multiphase flow measurements in an immersive manner. Immersive visualization is advantageous because it allows a user to explore spatial relationships in a manner that closely matches their experiences in the real world. Furthermore, the multiphase flow measurements in this work capture all three spatial

dimensions, thus being able to visualize those measurements in a 3D, immersive system is beneficial.

The visualization portion of this work will be completed at the Multimodal Experience Testbed and Laboratory (METaL). METaL is a CAVE automatic virtual environment facility built by Mechdyne Corporation. In CAVE and CAVE-style virtual reality systems, stereoscopy is achieved by projecting field sequential active stereoscopic images onto projection screens, which act as the physical boundaries of the interaction space. To further enhance the illusion of 3D, and in turn the user's immersion, the location of the user in the physical space is tracked with sensors and the computer-generated imagery is updated in accordance with the user's movements.

In the METaL implementation of a CAVE (shown in Figure 3.5), there are two walls and a floor, all of which display projected imagery. The left wall is a 2.7 m × 3.7 m (9 ft. × 12 ft.) rear-projected screen. The right wall is also a rear-projected screen, but it measures

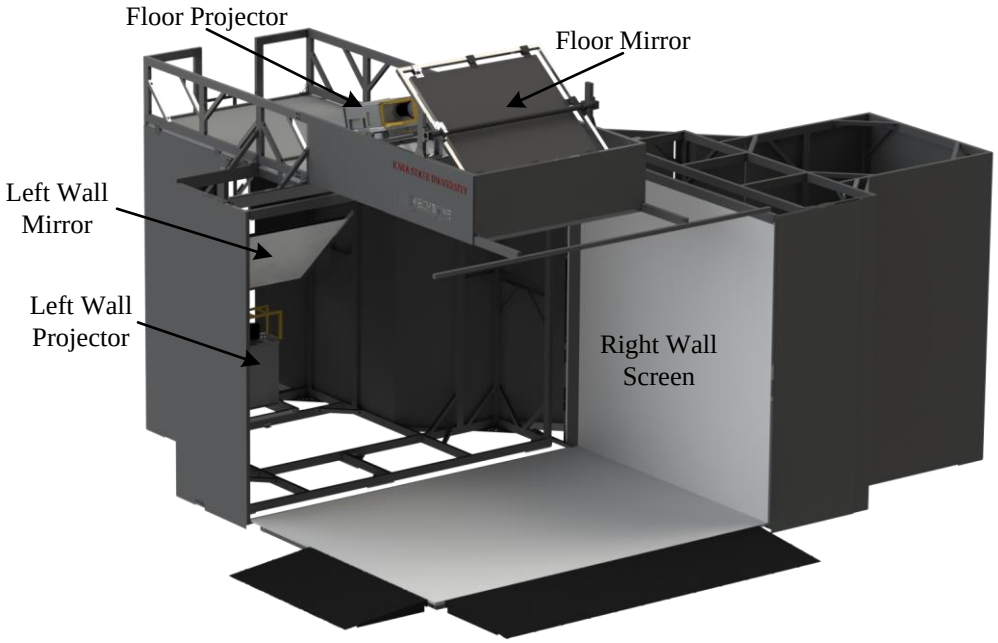

Figure 3.5: Schematic of METaL. Note the screen of the left wall is removed for clarity.

2.7 m × 2.7 m (9 ft. × 9 ft.). In contrast, the floor is a 3.7 m × 2.7 m (12 ft. × 9 ft.) front projected screen, made from fiberboard with a plastic laminate overlay. All the display surfaces are projected with Digital Projection International TITAN WUXGA-3D projectors (one per surface). These projectors are capable of rendering field sequential stereo images at 120 Hz (60 Hz per eye), with a maximum resolution of 1920 × 1200 pixels. In order to match the aspect ratio of the screens, the left wall and floor projectors display 1600 × 1200 pixels, with the right wall projector displaying at 1200 × 1200. To display the images to the user properly, XPAND X101 active shutter glasses are used (Figure 3.6). These glasses contain an LCD (liquid crystal display) shutter in each lens to ensure that the left eye only views the left frame and the right eye only views the right frame. The X101 glasses use an infrared signal (broadcast from emitters located behind the screens) to synchronize the LCD shutters on the glasses with the left eye and right eye images displayed by the projectors.

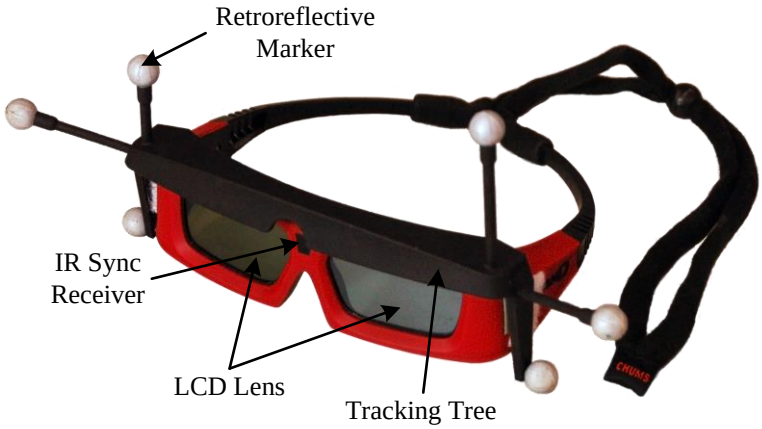

Figure 3.6: The XPAND X101 tracked glasses used in the METaL virtual environment.

All of the graphics for the projectors are rendered on a dedicated graphics computer which consists of dual Intel Xeon X5677 quad-core processors, 24 GB of random access memory (RAM), and dual NVIDIA Quadro Plex 2200-D2 visual computing systems. Each projector has its own, dedicated NVIDIA Quadro FX 5800 within the Quadro Plex 2200-D2, with one extra GPU available for other calculation tasks. There is also a head node, which is

identical to the render node except it contains only a single NVIDIA Quadro FX 5800 instead of dual Quadro Plex units. The head node is used primarily for controlling the tracking system.

The position tracking system in METaL is an ART TrackPack 4. This system uses four infrared cameras to measure the position of retroreflective tracking markers. The markers are illuminated by an infrared flash attached to the camera, and return a bright signal on the corresponding camera. By knowing the location of the cameras, and the location of the markers in the camera image, the TrackPack controller calculates the position of the marker and transmits it via Ethernet to the other computers in the system. With a single marker, the TrackPack system is only capable of calculating the position of an object, but not the orientation. However, by using a "tree" that contains four or more markers rigidly located at predetermined positions relative to each other, the TrackPack is capable of calculating the position and orientation of the tree. For most work in METaL, there are two tree-style markers used, one is attached to a pair of 3D glasses to determine the position and orientation of the users head (Figure 3.6) and the other is attached to a Nintendo Wii controller (or WiiMote), which is used as a wand for user input into the virtual environment.

The WiiMote wand, shown in Figure 3.7, is the primary input device for most VR applications in METaL. As previously noted, a tracking tree is attached to the WiiMote to provide position and orientation information. The buttons on the WiiMote are made usable in VR through the use of the virtual reality peripheral network (VRPN). The WiiMote is setup to use VRPN through a dedicated Linux server, which connects to the WiiMote via Bluetooth and then transmits information about the button states across METaL's local network. While the WiiMote also has an onboard camera and accelerometer, these sensors

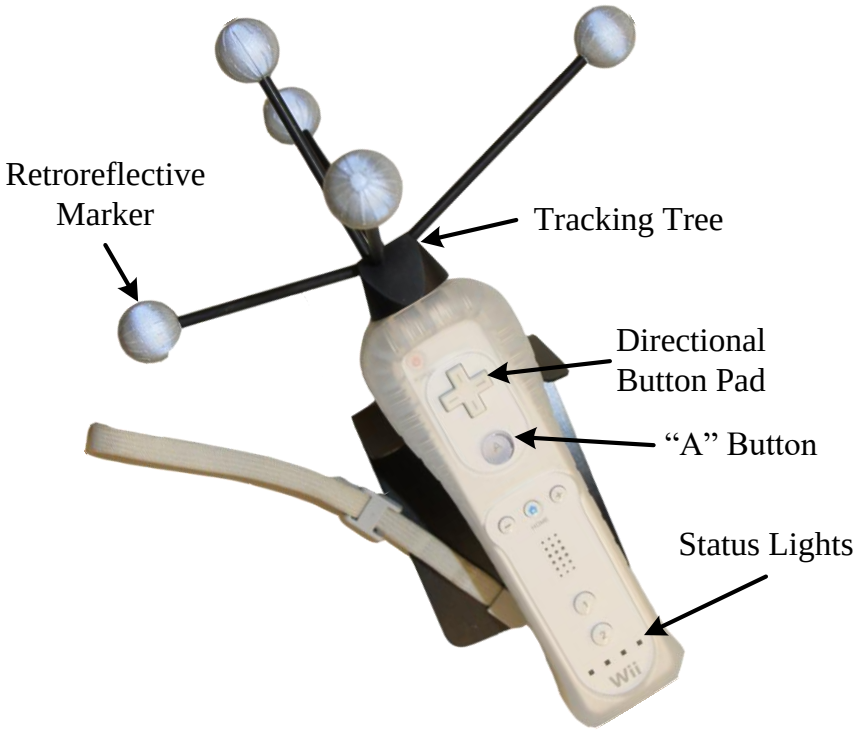

Figure 3.7: The tracked WiiMote for the METaL virtual environment.

are not configured for use in this application. The camera is blocked by the mount for the tracking tree, rending it useless, and the accelerometer data is largely redundant with the data provided by the tracker. However, support for the optional "nunchuck" add-on controller has been retained.

Finally, METaL also includes support for rendering spatial sound. The sound is generated using a Creative Sound Blaster Xi-Fi Titanium sound card on both the head and render nodes (only one source may be used at a time). These cards are linked via a TOSLINK optical audio cable to a Yamaha RX-V367 receiver. The receiver is setup with five full range speakers and a subwoofer. The full range speakers are located above the user on the cantilever structure that supports the floor projector mirror, while the subwoofer is situated on the ground behind the left wall screen.

3.2.1 VR JuggLua

While there are multiple toolkits available to create software for virtual reality (such as VR Juggler, CAVELib, and MiddleVR), the one used most commonly in METaL is VR JuggLua (Pavlik and Vance, 2012). VR JuggLua is a VR framework built on top of the VR Juggler platform and allows applications to be built interactively using the Lua scripting language and the Navigation Testbed interactive scripting console (shown in Figure 3.8). In contrast, pure VR Juggler applications are written in C++ and must be compiled prior to use.

The use of VR JuggLua in METaL provides several advantages. First, the Lua scripting language is simpler than C++ making it easier for a novice programmer to learn. Second, because VR JuggLua implements a read-evaluate-print loop, a programmer can add code and immediately see its impact on the application. This is extremely useful in VR because it allows for the fine-tuning of object positions and lighting though a trial-and-error process without the long compile time associated with a compiled language. Third, because VR JuggLua is built on top of VR Juggler, any code written in VR JuggLua can run on any

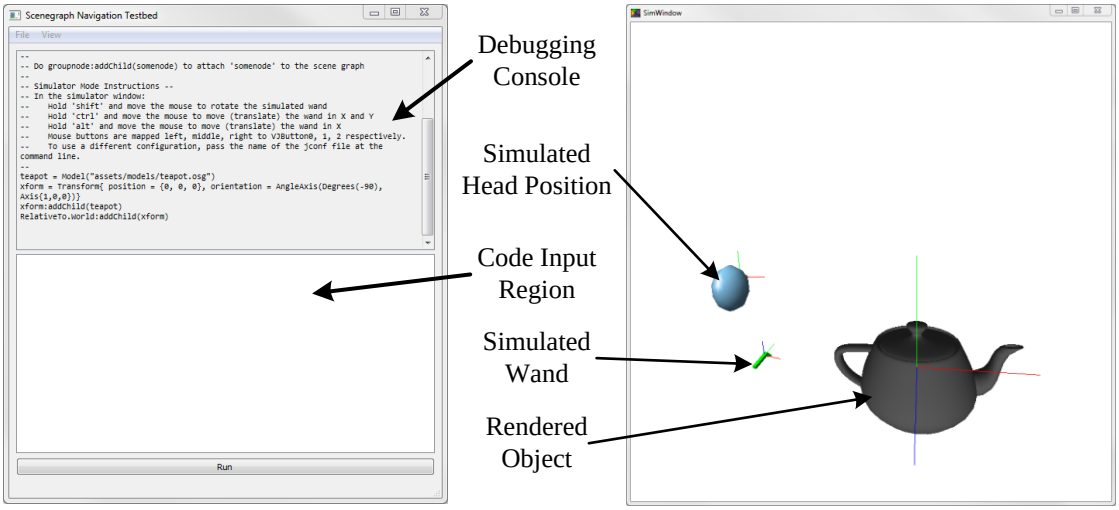

Figure 3.8: The graphical user interface of VR JuggLua, shown in simulation mode. The navigation testbed (left) allows for the input of Lua code while the program is running. The simulation window (right) provides a preview of what the user will see in virtual reality.

of the wide variety of systems that support VR Juggler, using the same configuration files. For example, a visualization program can be built in VR JuggLua on the desktop using simulator mode, and then by changing only the configuration file, it can be run on METaL for testing an immersive environment. When the code is ready for active use, the configuration file can be changed once more, and the program can be experienced in a six-sided VR system, such as the C6. This allows programming full applications using desktop hardware and seamlessly moving to higher end systems to gain an immersive experience.

As an example of how VR JuggLua code operates, a shader implementation of Phong shading is provided in Figure 3.9 (Phong, 1975). The first thing to note about this example is that it contains two separate programming languages. Lua provides the interaction with OSG, but OpenGL Shading Language (GLSL) is used for the shader program that runs on the GPU. Technically, Lua interprets the GLSL code as a generic string, but when the string is passed to the appropriate OSG function, it will parse and compile it for execution on the GPU.

The code in Figure 3.9 operates as follows. First, a model is loaded from the hard drive and saved to a variable. In the example, a relative path is used to load the model; however, Lua supports both relative and absolute paths. Next, the model is added to the rendered scene by attaching it to the scene graph node that positions objects relative to the world space. Because there are no transformations applied to the teapot model, its origin is placed at the world origin, but a transformation to position it elsewhere could easily be applied.

```
--Adapted from:
--https://github.com/vancegroup/vr-jugglua/blob/master/examples/advanced/phong-shading.lua

--Load a model of a teapot using a relative path
teapot = Model([[assets/models/teapot.osg]])

--Create a transform to move the teapot to x=1, y=0, z=-1
--And rotate it -90° about the X-axis
teapotTransform = Transform{position = {1, 0, -1}, orientation = AngleAxis(Degrees(-90), Axis{1, 0, 0})}

--Add the teapot to the transform
teapotTransform:addChild(teapot)

--Add the transform to the scene
RelativeTo.World:addChild(teapotTransform)

--Function to apply a shader to a model
applyShaderToStateSet = function(stateset)
    --Define the vertex shader
    local vertexShader = osg.Shader(osg.Shader.Type.VERTEX,
    [[
        //GLSL code for the vertex shader
        varying vec3 normalVector;
        varying vec3 viewVector;

        void main(void)
        {
            viewVector = vec3(gl_ModelViewMatrix * gl_Vertex);
            normalVector = normalize(gl_NormalMatrix * gl_Normal);
            gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
        }
    ]])

    --Define the fragment (pixel) shader
    local fragmentShader = osg.Shader(osg.Shader.Type.FRAGMENT,
    [[
        //GLSL code for the fragment shader
        varying vec3 normalVector;
        varying vec3 viewVector;

        void main(void)
        {
            vec3 lightVector = normalize(gl_LightSource[0].position.xyz - viewVector);
            vec3 eyeVector = normalize(-viewVector);
            vec3 reflectionVector = normalize(-reflect(lightVector, normalVector));

            //Calculate the ambient light term
            vec4 ambient = gl_FrontLightProduct[0].ambient;

            //Calculate the diffuse light term
            vec4 diffuse = gl_FrontLightProduct[0].diffuse
                           * max(dot(normalVector, lightVector), 0.0);
            diffuse = clamp(diffuse, 0.0, 1.0);

            //Calculate the specular light term
            vec4 specular = gl_FrontLightProduct[0].specular
                            * pow(max(dot(reflectionVector, eyeVector), 0.0), 0.3
                            * gl_FrontMaterial.shininess);
            specular = clamp(specular, 0.0, 1.0);

            //Write the final color to the fragment
            gl_FragColor = gl_FrontLightModelProduct.sceneColor + ambient + diffuse + specular;
        }
    ]])

    --Set the shaders to run on the GPU
    local program = osg.Program()
    program:addShader(vertexShader)
    program:addShader(fragmentShader)
    stateset:setAttributeAndModes(program, osg.StateAttribute.Values.ON)
end

--Apply the shader to the teapot model
applyShaderToStateSet(teapot:getOrCreateStateSet())
```

Figure 3.9: VR JuggLua code to load a teapot model and render it with GPU-based Phong shading. Note, this example contains two separate programming languages: Lua (black) and GLSL (blue).

Next, a Lua function is defined to apply the shader (in this case a Phong shader) to the rendering state of an object. In this example, this function is only used on the teapot, but it could be used on multiple models if more were loaded. Inside this Lua function are two GLSL functions, the vertex shader and the fragment shader (called a pixel shader in DirectX). The syntax of the GLSL language is very similar to C; however, there are a few unique features. First, vectors are native variable types in GLSL. For example, "vec3" is used multiple times and indicates that the variable is a three-component floating point vector. Second, GLSL implicitly assumes that the code is run for each vertex or fragment (depending on the shader type).

Once the GLSL functions are defined, the final step in this Lua function is to package them together as a complete shader program, and load the complete shader program to the GPU. The final step in the example is to apply the Lua function to the teapot. The results of this code produce the image shown in Figure 3.10.

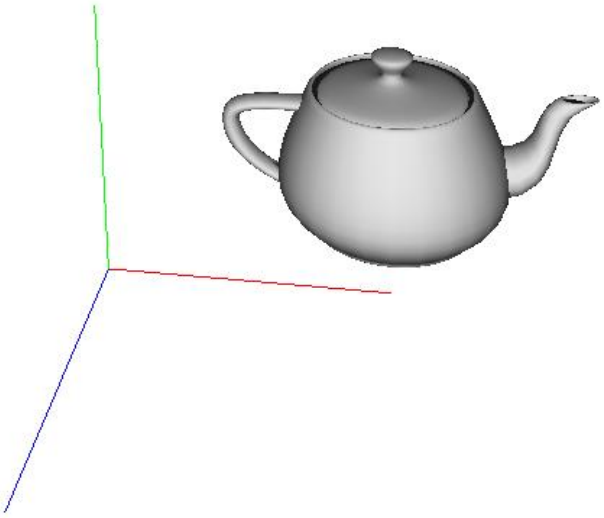

Figure 3.10: The Utah teapot rendering using GPU-based Phong shading via VR JuggLua using the code in Figure 3.9.

3.2.2 Kinect Sensor

To augment the optical tracking system in METaL, it is equipped with multiple Kinect for Windows sensors. There are two versions of the sensor available. The original Kinect for Windows sensors, which will be referred to as the Kinect v1, is a version of the Kinect for the Xbox 360 gaming system with an enhanced firmware. The second version of the Kinect for Windows is the Kinect sensor for the Xbox One system, paired with a converter box that permits it to be used on a Windows computer. This is referred to as the Kinect v2.

The Kinect v1 sensor is a structured light sensor consisting of an infrared projector and an infrared camera to determine the depth of objects in its view. It does this by projecting a pattern of infrared dots and using the distortions in this pattern to calculate the depth. Together, the infrared project and infrared camera are often referred to as the depth camera. The Kinect v1also contains a color camera to provide a 2D visible light image, a microphone array for doing voice recognition, and an accelerometer (Zhang, 2012). The arrangement of the cameras on the Kinect is shown in Figure 3.11; the microphone array and accelerometer are internal components, and are not visible. Both the color and depth cameras are capable of frame rates up to 30 FPS with a resolution of 640 × 480 and a field of view of 57°

Figure 3.11: The Microsoft Kinect sensor, version 1.

horizontally and 43° vertically (Microsoft, 2014b). The color camera is also capable of operating at a higher resolution (1280 × 960 pixels) by lowering the frame rate to 12 FPS. The microphone array captures monaural audio at 16 kHz with 24-bit depth (Microsoft, 2014b). Additionally, it is able to estimate the angle of the audio source relative to the Kinect in 10° increments from −50° to 50° (Microsoft, 2012b).

The Kinect v2 sensor also uses infrared light to determine the depth of objects in its view. However, its depth sensing is based off an infrared time-of-flight sensor instead of structured light (Lun and Zhao, 2015). The depth sensor is capable of a resolution of 512 × 424 at 30 FPS and a field of view of 70° horizontally and 60° vertically. It also contains a high definition color camera with a resolution of 1920 × 1080 at 30 FPS (Microsoft, n.d.-f). Like the Kinect v1, the Kinect v2 also contains a microphone array for voice recognition and audio source angle estimation. However, the Kinect v2 does not contain an accelerometer or tilt motor.

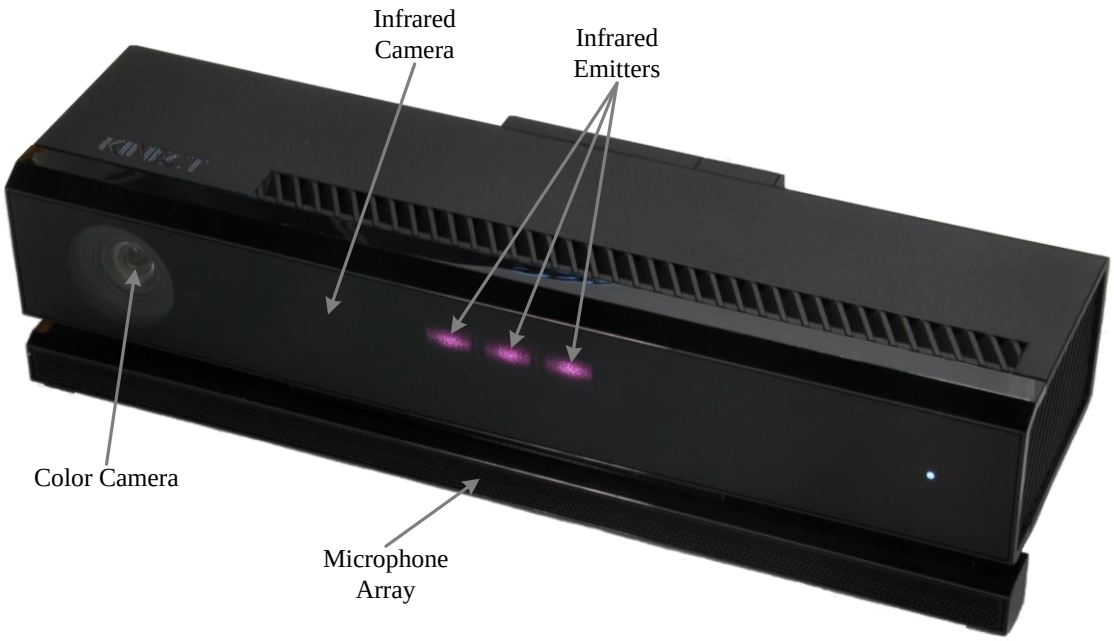

Figure 3.12: The Microsoft Kinect sensor, version 2.

One of the key advantages of the Kinect is that it supports markerless tracking. Most tracking systems, like the ART TrackPack 4 used in METaL, require a tracking target to be affixed to each point that is to be tracked. The Kinect uses its depth camera and image recognition software to identify humans in the image and estimate how the bodies are positioned relative to the Kinect (Shotton et al., 2011). To leverage this ability in METaL, multiple Kinects are positioned around the interaction area. The data from the Kinects supports encumbrance free gesture recognition, voice recognition, speaker position determination, and skeletal tracking. While the quality of the tracking data from the Kinect sensors is much lower than what METaL's optical tracking system provides, it is still a useful complementary tracking system. For example, the optical tracking system is ideal for providing head tracking within METaL, as it provides a low-latency, high-precision position and orientation. While the user is somewhat encumbered due to the tracking marker, the user must wear glasses anyway for the stereoscopy to function, thus the extra encumbrance from the tracking marker on the glasses is minimal. Conversely, the Kinect sensor's data are high-latency and prone to noise, which would likely make the user ill if used for head tracking. However, because the optical tracking requires a marker for each tracked position, the markerless tracking of the Kinect provides significantly lower encumbrances. This is particularly useful for tasks such as 3D object manipulation, where high latency in movement is unlikely to significantly reduce a user's immersion, but encumbering objects may affect immersion.

CHAPTER 4:

A HIGH-SPEED X-RAY DETECTOR SYSTEM FOR NONINVASIVE FLUID FLOW MEASUREMENTS

One of the main challenges with the X-ray imaging of fluid flows, as discussed in Chapter 1, is that the acquisition speed of most systems is too slow to image the high-speed dynamics of many common multiphase flows. Working towards objective one from Section 1.2, this chapter examines the feasibility of increasing the acquisition speed of X-ray radiography by coupling a high-speed camera with an X-ray image intensifier. This chapter is based off a paper presented at the 2013 American Society of Mechanical Engineers (ASME) Fluids Engineering Division Summer Meeting (FEDSM).[1]

4.1 Abstract

The opaque nature of many multiphase flows has long posed a significant challenge to the visualization and measurement of flow phenomena. To overcome this difficulty, X-ray imaging, both in the form of radiography and computed tomography, has been used successfully to quantify various multiphase flow phenomena. However, the relatively low temporal resolution of typical X-ray systems limit their use to moderately slow flows and time-average values. This paper discusses the development of an X-ray detection system capable of high-speed radiographic imaging that can be used to visualize multiphase flows. Details of the hardware will be given and then applied to sample multiphase flows in which

[1] Based on Morgan, T. B., Halls, B. R., Meyer, T. R., and Heindel, T. J. (2013). A High-Speed X-ray Detector System for Noninvasive Fluid Flow Measurements. In *ASME 2013 Fluids Engineering Division Summer Meeting (FEDSM2013)* (p. FEDSM2013-16427). Incline Village, NV, USA: ASME. doi:10.1115/FEDSM2013-16427

X-ray radiographic images of up to 1,000 frames per second were realized. The sample flows address two different multiphase flow arrangements. The first is a gas-liquid system representative of a small bubble column. The second is a gas-solid system typically found in a fluidized bed operation. Sample images are presented and potential challenges and solutions are discussed.

4.2 Introduction

The use of dynamic X-ray imaging started with the development of fluoroscopic X-ray systems, which used a phosphor screen to convert X-rays into visible light that an observer would view directly (Cartz, 1995). While these systems allowed scientists to view flows in real time, they could not record data for later analysis or slow down events that were too fast to be observed by the human eye. The use of X-ray sensitive film allowed for the direct recording of data, but due to the relative insensitivity of X-ray film, this process required long exposures or high X-ray intensity, and time consuming development processes (Boyer et al., 2005; Chotas et al., 1999). Therefore, it was not until the development of digital X-ray detection systems that time-sequenced radiography became the powerful tool for fluid flow research it is today (Heindel, 2011).

However, the current state of X-ray imaging still generally limits time sequences to standard video frame rates. Most direct X-ray detectors are only capable of 30 frames per second (FPS), and indirect detectors are limited by the decay rate of the phosphor screen (Gruner et al., 2002; Seibert, 2006). Flash X-ray systems use high-power, short duration X-ray pulses to take images at higher speeds, but are generally limited to generating a small number of frames because of energy storage bank recharge times and anode deterioration (Boyer et al., 2005; Heindel, 2011). For example, Romero and Smith (1965) used flash X-

ray radiography to examine fluidized beds, but were limited to two radiographs, at different spatial locations, per experiment. Heindel and Monefeldt (1997, 1998) later used flash X-ray to examine pulp suspensions in bubble columns, and although they achieved 30 nanosecond exposure time, they were limited to single X-ray frames. Finally, Grady and Kipp (1994) and Boyer et al. (2005) used flash radiography to image projectiles, with Boyer et al. achieving up to 50 consecutive frames before significant anode deterioration.

Synchrotron X-ray sources have also been used to image fluids at high speed (MacPhee et al., 2002; Royer et al., 2005; Wang et al., 2008). For example, Royer et al. (2005) observed impact-induced granular jets at frame rates up to 5000 FPS using the Advanced Photon Source at Argonne National Laboratory. MacPhee et al. (2002) was able to image shock wave generation in high-pressure sprays at over 100,000 FPS, also using the Advanced Photon Source. However, synchrotron sources are cost prohibitive for most fluid flow research.

Finally, there have been a few studies using continuous X-ray sources to examine systems at high speed. One early, moderately high speed fluid study was completed by Rowe and Partridge (1965), who used an X-ray intensifier and cinematographic film camera to achieve frame rates of 50 FPS. A more recent study by Zolfaghari et al. (2002) used a digital CCD camera and X-ray intensifier to observe current interruption in a circuit breaker at 4000 FPS. However, the high material density and well-defined material boundaries inside a circuit breaker require a less sensitive system than one typically needed for fluid flow visualization.

This paper will summarize current efforts to produce high-speed radiographic images of highly dynamic, opaque multiphase flows using an X-ray image intensifier and high-speed camera.

4.3 Experimental Setup

This study used the X-ray Flow Visualization Facility at Iowa State University (Heindel et al., 2008). However, the image acquisition system was modified from its standard arrangement to significantly increase the imaging speed. The standard LORAD LPX 200 X-ray source was used to provide the radiation. This source provides a conical polychromatic X-ray beam with a maximum tube potential of 200 kV and maximum tube current of 10 mA with a maximum power output of 900 W. This source was paired with a Precise Optics PS164X image intensifier to convert the X-ray photons into viable light. This particular intensifier is designed to use a remotely controlled C-mount lens paired with a CCD camera, such as the DVC-1412 used in previous studies (Heindel et al., 2008). To increase the speed of the system, the CCD camera was removed and replaced with a complementary metal-oxide-semiconductor (CMOS) based Photron FASTCAM SA5 high-speed camera. The SA5 is well suited for this application due to both its resolution (1024 × 1024 pixels) and speed (7000 FPS at full resolution, up to 1,000,000 FPS at reduced resolution). It is also extremely sensitive (ISO 10,000 equivalent), enabling it to image at high frame rates despite the low light intensity inherent in X-ray detectors. The use of this camera also required replacing the stock intensifier lens with a Nikon Nikkor 50 mm F-mount lens. Furthermore, a custom lens mount was required on the camera to shorten the flange focal length by 3.13 mm (0.13 in), and allow the lens to achieve the true infinite focus distance, as required by the intensifier optics. Finally, the camera was shielded all the way

around by a 6.35 mm (0.25 in) thick lead shield to prevent damage to the camera from the high intensity radiation. The X-ray setup is schematically represented in Figure 4.1.

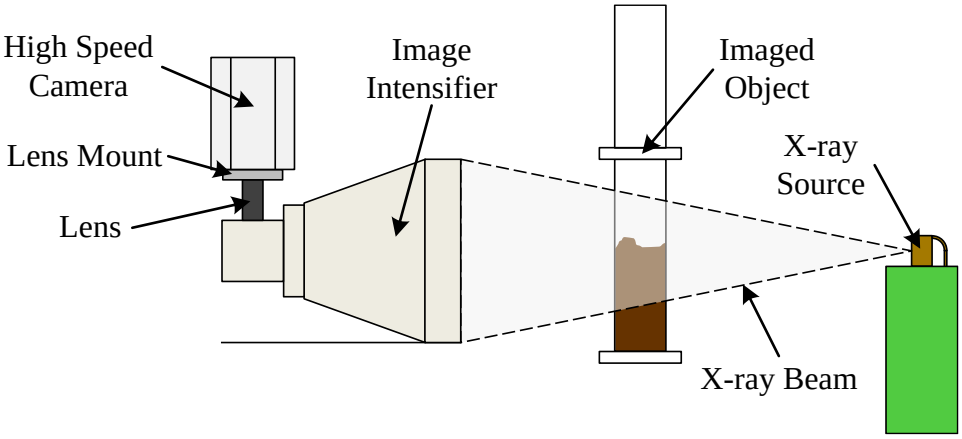

Figure 4.1: The imaging setup for the high speed camera. Note that the image intensifier has an internal mirror to allow the camera to be mounted out of the primary X-ray beam. Lead shielding is omitted from the schematic for clarity.

All images were acquired using the standard acquisition software provided with the SA5 camera system. This produced a sequence of 12-bit tiff images. These images were then digitally processed to normalize the images and remove the pincushion artifact caused by the image intensifier. The result of this processing can be seen in Figure 4.2, which shows the raw and corrected image of a calibration grid. This calibration grid is a sheet of 1.9 mm (0.074 in) thick stainless steel with an array of 2 mm (0.078 in) holes drilled at 12.7 mm (0.5 in) on center intervals, both vertically and horizontally. More details on the correction algorithm can be found in Section 3.1.2. The corrections cause some artifacts at the edges of the image; however, this is outside the area of interest, so their effect is negligible.

To test the effectiveness of the camera system, two flow systems were used in this study. The first is an 8.0 cm (3.15 in) diameter bubble column. It was filled to a height of two bed diameters with water, and air was injected from the bottom through a central 1.0 cm (0.39 in) diameter by 1.5 cm high (0.59 in) porous injector. For the imaging of this system, the flow rate was held constant at 50 LPM (13.2 GPM) by a computerized flow controller, producing

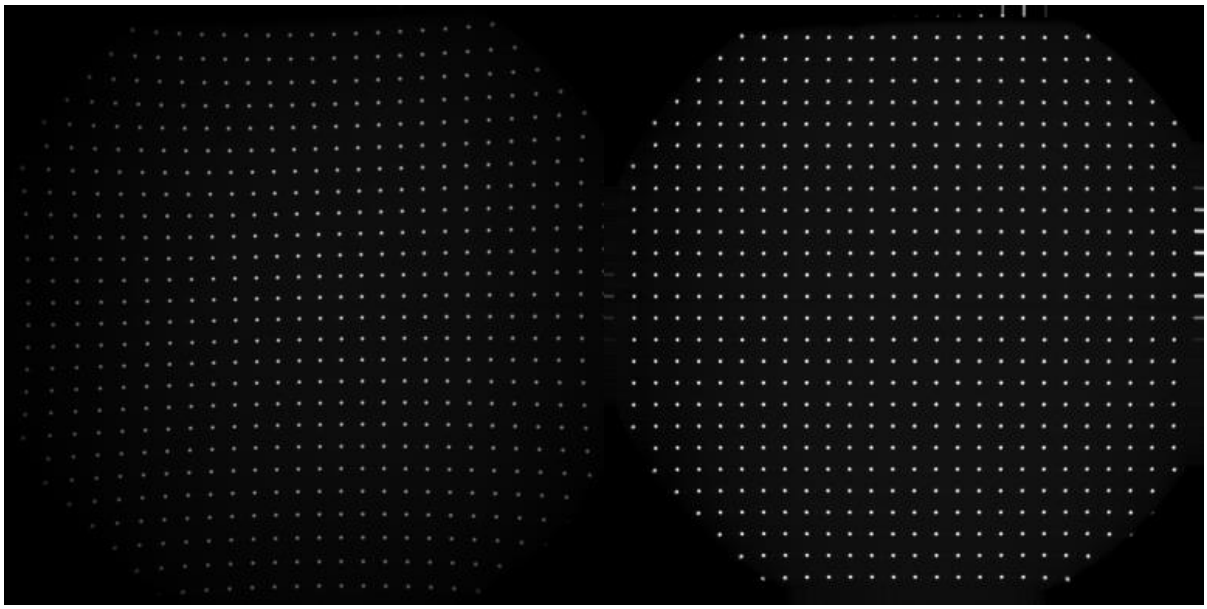

Figure 4.2: A comparison of a radiograph of the X-ray intensifier calibration grid before and after image processing. The unmodified frame, left, shows a pincushion distortion. The corrected frame, right, has the rectilinear structure of the calibration grid restored.

a superficial gas velocity of 17 cm/s (6.7 in/s), in which the flow regime was clearly churn turbulent. Once the flow was operating at a steady flow rate, the camera was triggered to take a 1000 frame sequence. Each image in this sequence was acquired at full resolution (1024 × 1024 pixels), with an exposure of 16.3 μs and each image was taken 1 ms apart (for a frame rate of 1000 FPS). The short exposure reduced the effects of motion blur, while the 1000 FPS frame rate was selected to maximize the length of time the flow was imaged, while still keeping the inter-frame flow movement small. To achieve such a short exposure time the X-ray power was set at 100 kV and 9.0 mA.

The second flow system consisted of a 15.24 cm (6 in) internal diameter fluidized bed that was filled to a height of one bed diameter with crushed walnut shell, sieved to a particle size range of 500–600 μm (0.020–0.024 in). Air was injected from the bottom through a distributor plate (Drake and Heindel, 2011). The air flow through this system was maintained at 280 LPM (74.0 GPM)—approximately two times minimum fluidization—by

the computerized flow controller resulting in a superficial gas velocity of 26 cm/s (10.2 in/s). For this system, 10,000 frames were acquired at 1000 FPS using full resolution. However, for this test a longer exposure was required to provide enough intensity to image the system due to its larger diameter and dense material. In this case, an exposure of 50.2 μs was used and the X-ray power was set at 80 kV and 7 mA.

The fluidized bed flow was also seeded with a tracer particle to allow the analysis of the particle movement from the image sequence. This particle was a 2.03 mm (0.08 in) diameter lead sphere, inside an 8 mm (0.32 in) diameter foam sphere (Drake et al., 2011). In order to track this particle, a normalized cross-correlation method was used. This method computes the similarity between a template image (in this case a radiograph of just the particle) and each point in the image. The particle tracking then finds the point of highest correlation, and marks that as the particle location (Drake et al., 2009; Morgan and Heindel, 2010).

4.4 Results and Discussion

An analysis of the bubble column sequence shows that the air tends to rise in the center of the column, with recirculation currents along the edges of the column. Once the air reaches approximately 1.5 column diameters above the bottom of the column, a foam-like region of high gas fraction begins, which matches closely with visual observations of the column's operation. By tracking the leading edge of bubbles as they rise, the velocity of the bubble can be ascertained. For the bubble in Figure 4.3, this measurement yields a bubble rise velocity of 55.4 cm/s ± 0.1 cm/s (21.8 in/s ± 0.04 in/s).

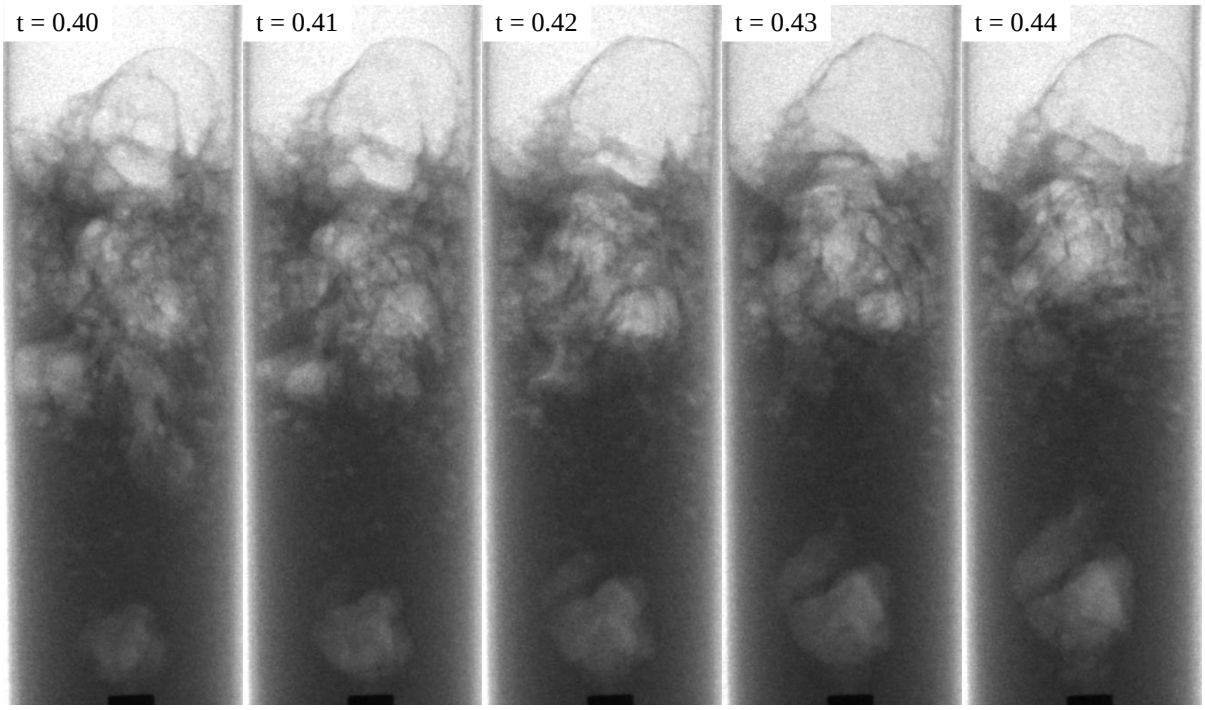

Figure 4.3: A gas-liquid system with gas bubbles (lighter gray regions) rising from a central injector. Images shown from time t=0.40 s to t=0.44 s. Every tenth frame is shown to illustrate the bubble movement more clearly.

An examination of the fluidized bed in Figure 4.4 shows that the distributor plate maintains a relatively even distribution of bubbles across the bed. This is consistent with the findings of Drake and Heindel (Drake and Heindel, 2011) obtained through X-ray computed tomography scans. The addition of a tracer particle to the flow, shown Figure 4.4, allowed for the evaluation of granular movement within the flow, shown in Figure 4.5.

The tracking of the particle revealed downward flow zones at both sides of the bed, as projected onto the X-ray detector. However, these zones do not appear to be large enough to trap the particle fully, as it never reaches the bottom of the bed throughout the entire test. While this is just a small example of particle motion inside a fluidized bed, it shows the clarity with which particle tracking data may be obtained using high-speed radiography. Previous research using the same normalized cross-correlation algorithm was only able to identify the particle correctly 70–95% of the time, depending on the particle shape and flow

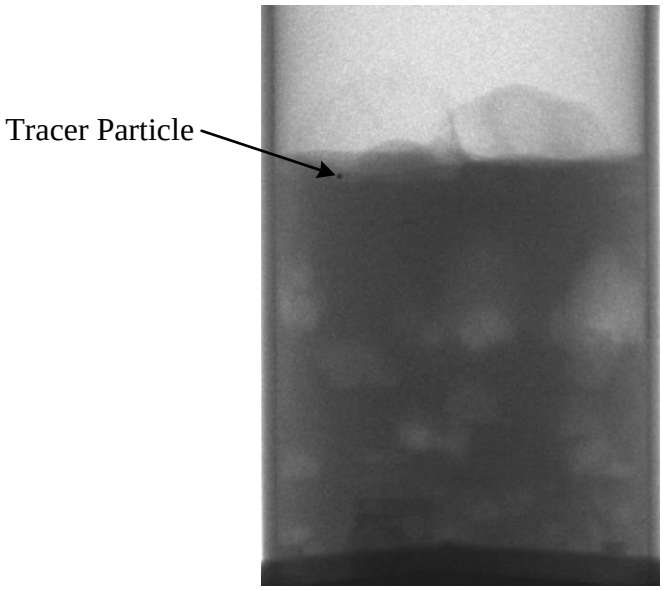

Figure 4.4: A gas-solid system with gas bubbles (lighter gray regions) rising from a uniform distributor on the bottom. This image was acquired at t = 1.050 s.

conditions (Drake et al., 2009; Morgan and Heindel, 2010). Using the high-speed radiographs, the particle was correctly identified 99.98% of the time. This increase can be attributed to the extremely short exposure time, as compared to earlier studies.

Both flows show the ability of the X-ray system to image at high speeds. The primary limitations of high-speed X-ray imaging with a tube source—output image intensity and phosphor decay—were non-issues in this case. The full output power of the source was sufficient to provide a bright enough output image from the intensifier to support exposures down to 16.3 μs while still using more than 75% of the camera's intensity range. If some loss of intensity range is acceptable, the exposure times could be further reduced. As for the phosphor decay, no effects from the time response were found at 1000 FPS. This provides enough speed to examine many flows of industrial interest in depth. Furthermore, the exposure times are short enough that the frame rates could be increased significantly if the phosphor decay rate is fast enough.

4.5 Summary

This work demonstrates that the pairing of a high-speed camera with an X-ray image intensifier is capable of imaging fluid flows at high speed. The system has been proven to image at 1000 FPS, with exposures as low as 16.3 μs. The system is capable of revealing the dynamic details of a fluid flow that cannot be observed with other methods, such as computed tomography. Furthermore, the high quality particle tracking results will provide a powerful quantitative tool to determine experimental flow velocities inside opaque systems.

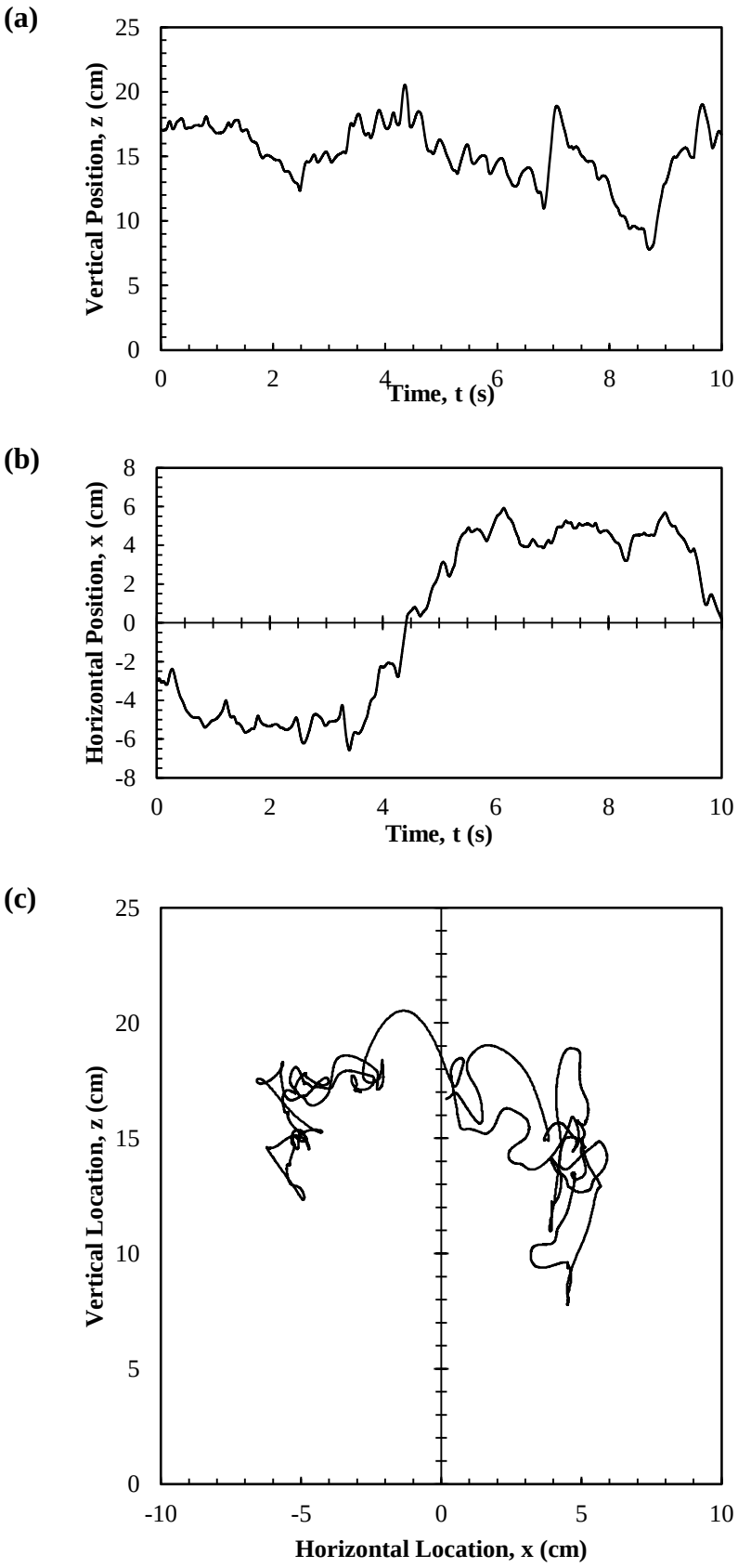

Figure 4.5: The path of the tracer particle in a fluidized bed, as tracked by the normalized cross-correlation method for a 10 s period. From one source-detector pair the x-position vs. time (a), z-position vs. time (b), and x-position vs. z-position (c) can be determined. Another source-detector pair would be required to determine the y-position.

CHAPTER 5:

SENSITIVITY OF X-RAY COMPUTED TOMOGRAPHY MEASUREMENTS OF A GAS-SOLID FLOW TO VARIATIONS IN ACQUISITION PARAMETERS

Continuing towards the goal of improving X-ray imaging as a tool for noninvasive characterization of fluid flows, this chapter contributes to object two from Section 1.2 by demonstrating that the results of X-ray computed tomography flow measurements are not dependent on the choices the researcher makes in imaging parameters. Specifically, it presents an examination of the effects of changing X-ray acquisition parameters on the resulting fluid flow measurements. This chapter is based on a paper that was published in *Flow Measurement and Instrumentation* in June 2017.[2]

5.1 Abstract

Due to its high spatial resolution and non-invasive nature, X-ray computed tomography has become a popular method for determining the flow characteristics of multiphase flows. However, because many of the X-ray computed tomography systems used for non-destructive imaging of multiphase flows provide the operator wide leeway in the selection of imaging parameters, the potential exists for errors to be introduced into the measurements if the algorithms are sensitive to these changes. In this paper, a representative multiphase flow (specifically, a fluidized bed) is imaged with a wide range of X-ray tube electrical potentials, currents, and detector exposure times and reconstructed with a wide range of centers of

[2] Based on Morgan, T. B., and Heindel, T. J. (2017). Sensitivity of X-ray Computed Tomography Measurements of a Gas-Solid Flow to Variations in Acquisition Parameters. Flow Measurement and Instrumentation, 55(June), 82–90. doi:10.1016/j.flowmeasinst.2016.10.011

rotation. The results of these tests show that while the raw computed tomography (CT) intensities are sensitive to these parameter variations, once the measurements are calibrated to reference images (in this case through a void fraction calculation), the final results are insensitive to most changes. In the extreme cases where there is some sensitivity to the parameter changes, the causes and practical implications are discussed.

5.2 Introduction

One of the primary challenges in the measurement of multiphase flows has been determining the flow characteristics inside the flow because many of the flows of interest are opaque or contained within an opaque vessel. This opaque nature limits any optical measurements to the surface of the flow (van Ommen and Mudde, 2008). Furthermore, many common flow sensors, such as pitot tubes and hot wire anemometers, intrude into the flow creating the potential for the sensor to change the flow characteristics (Boyer et al., 2002; Whitemarsh et al., 2016). The way around these limitations is to use noninvasive measurement methods. While many methods for noninvasive imaging have been proposed and tested, one of the best solutions for achieving high spatial resolution in three dimensions is X-ray CT. However, X-ray CT requires the acquisition of numerous projections from different angles around the flow of interest. This results in long scan times (on the order of 15 minutes for the scanner in this study) and limits the use of X-ray CT measurement of time-averaged values for most flows. Due to this limitation, one of the most common applications of X-ray CT in multiphase flows is to determine the local time-average void fraction of a flow (Heindel, 2011; Ikeda et al., 1983), which is also called the local gas fraction or local gas holdup.

When acquiring an X-ray CT scan, there are a large number of parameters the operator needs to set, such as tube voltage, tube current, exposure time, and number of projections. In medical CT imaging, X-ray dosage is strictly prescribed to minimize a patient's exposure to radiation which limits the range of available settings radiologists can use (Fazel et al., 2009). However, in CT imaging of multiphase flows, the radiation dose the flow receives is typically not a concern, giving the operator wide leeway in the selection of acquisition parameters. While nonlinearities in the X-ray mass attenuation coefficient can lead to certain X-ray energies yielding better contrast between materials, it is not always considered when selecting X-ray parameters (Ketcham and Carlson, 2001). Furthermore, even when nonlinearities in the mass attenuation coefficient are considered, there remains a range of parameters that can be selected. Thus, the choice of parameters is typically as much art as science, with the operator selecting parameters based on what looks "best." This research will analyze how variations in the operator's selection of tube voltage, tube current, exposure time, and center of rotation impact the results of multiphase flow scans.

To understand how a change in image acquisition parameters can impact the results of a CT scan, this study looks at both uncalibrated CT reconstructions and local phase fraction results. A change in tube voltage, and in turn the average X-ray photon energy, will increase or decrease the brightness of a projection. Additionally, when the tube voltage is changed, nonlinearities in the X-ray mass attenuation coefficient can cause the ratio of intensities between the flow phases to change, leading to an over or underestimation of the phase fraction. Like tube voltage changes, tube current and exposure changes will also result in a change in the brightness of the projections. However, these changes should not impact the local X-ray mass attenuation because they are only dependent on the material and incident

X-ray photon energy. Even without the effects of mass attenuation coefficient nonlinearities, changes in projection brightness will impact how much of the detector's dynamic range is used, and could lead to changes in signal-to-noise ratio. Additionally, under all parameter variations there are physical phenomena that are not modeled in the reconstruction algorithm, such as beam hardening and partial volume effects, that can lead to artifacts in the reconstruction (Baxter and Sorenson, 1981; Goodsitt et al., 2006). Finally, medical radiology research has shown that, even in medical settings where the parameters and calibration are strictly prescribed, variations in acquisition parameters and variations between CT scanners can lead to differences in the raw CT number (Groell et al., 2000; Levi et al., 1982). Thus, before accepting X-ray CT as a quantitative method for measuring multiphase flows, variations in the image acquisition parameters should be tested to determine if the desired results are sensitive to these variations.

5.3 Experimental Setup and Methods

Determining the sensitivity of multiphase flow CT scans to the imaging parameters requires three key components: (i) a test system that includes both a representative CT scanner and a representative multiphase flow to scan, (ii) a method of determining baseline imaging parameters to use as a reference, which should reflect the typical process a researcher would use to select imaging parameters, and (iii) a method of analysis to determine the influence of the imaging parameters on the final results. Section 5.3.1 will cover the CT scanner and multiphase flow used in this study to test the sensitivity. The process used to select the baseline scanning and reconstruction parameters, as well as how those parameters were varied for testing, is discussed in Section 5.3.2. Finally, Section 5.3.3 discusses the methods used to analyze the impact of the parameters to the scan results.

5.3.1 Test System

To represent the conditions of a multiphase flow experiment, the comparison of X-ray CT parameters was done using a real multiphase flow as a test object instead of an artificial phantom. The selected flow was a 10.2 cm diameter fluidized bed contained inside an acrylic column. This flow is representative of previous laboratory scale systems investigated with X-ray CT (Drake and Heindel, 2012b; Escudero and Heindel, 2011; Franka and Heindel, 2009). The bed was filled with 500-600 μm glass beads to a static bed height of 10.2 cm. The bed was fluidized to two times the minimum fluidization velocity, which was determined to be a volumetric flow rate of 144 lpm of air. The air was humidified before injecting it into the fluidized bed by bubbling it through a tank of water to prevent static electricity from building up in the fluidized bed. The volumetric flow rate of the air was measured using a 0-1000 lpm Aalborg GFM771 flow meter with a maximum error of 2% of full scale. The flow rate was maintained by a computer controlled Aalborg SMV40-SVF2-A proportional valve.

The X-ray Flow Visualization (XFloViz) facility at Iowa State University was used to acquire the CT images for this study. This system has been described in detail by Heindel et al. (2008); therefore, only a brief overview will be provided. The XFloViz facility, shown in Figure 5.1, has two Lorad LPX 200 X-ray sources that are able to maintain a tube voltage within ±1 kV and tube current within ±0.1 mA of the user selected setting. The X-ray sources are mounted at 90° to each other around a slew ring that provides 360° rotation around the object of interest, allowing X-ray projections to be acquired from numerous angles around the object. Across from each source is an X-ray detector. There are two types of detectors available in the XFloViz facility. One type is a Hamamatsu Photonics CsI scintillator screen

1) X-ray sources
2) Scintillator detector
3) Intensified detector
4) Fluidized bed
5) Slew ring

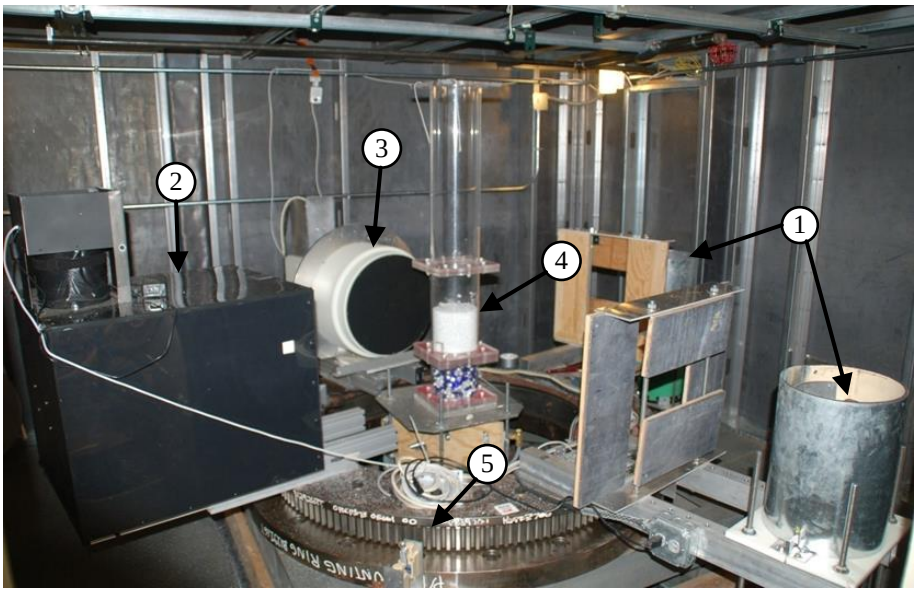

Figure 5.1: An image of the X-ray Flow Visualization facility used in this study. Note that, although two X-ray source-detector pairs are available, only one pair was used to acquire the CT scans in this study.

paired with an Apogee Imaging Systems Alta U9 CCD camera. The second type of detector is a Precise Optics PS164X X-ray image intensifier with a Digital Video Camera Company DVC-1412 CCD camera. For this study, only one X-ray source was used and it was paired with the CsI scintillator detector. This provides a higher spatial resolution at the cost of a longer exposure time.

In order to acquire a CT scan, numerous X-ray projections are required. To accomplish this, the system takes a radiographic image at one position and transfers it to a computer for storage and processing. Next, the system rotates the source-detector pair a preset amount (in this case 1°) around the imaging region using the slew ring. An image is then acquired at the new position and the entire process repeats until projections have been acquired from all 360° around the bed. After all the radiographic projections have been acquired, the scan is complete. However, in order to produce a useful 3D data set, the radiographic projections must be reconstructed. In the XFloViz facility, this is done using an in-house implementation of the filtered backprojection (FBP) algorithm (Zhang, 2003). The resulting

reconstruction creates a 3D volume of information, where each voxel (short for volume element, which is the 3D equivalent of a pixel) within the volume represents the X-ray attenuation of the material at that point in space. Because it takes time to acquire the 360 projections, the generated 3D data are necessarily time-averaged. This process has been shown to be highly repeatable in multiphase flows (Drake and Heindel, 2011).

While most of the inputs to the reconstruction algorithm are well defined, the implementation of the FBP algorithm requires the determination of the location of the center of rotation (COR) of the scanner. The COR is dependent on the location of the camera relative to the source, which is adjustable in the XFloViz facility, and the physical geometry of the scanner. The current method for determining the COR (described in Section 5.3.2) is dependent on the user's judgement of image quality, and thus is subject to error. It is important to note that, for this study, no digital image processing was done on the projections prior to the CT reconstruction. This creates a "worst case scenario" for potential intensity variations from condition to condition, providing a more rigorous test of the sensitivity of the CT results to acquisition parameter changes. Similarly, a beam hardening correction is also available as part of the reconstruction process. However, unlike previous studies using this system (Drake and Heindel, 2012b; Franka and Heindel, 2009), the beam hardening correction was not used in this study so that any errors introduced by beam hardening would be visible.

5.3.2 Determination of Baseline Parameters

The fluidized bed was first imaged using qualitatively determined "best" parameters to provide a baseline for comparison that is representative of the typical parameter selection process. These X-ray parameters were determined by first increasing the X-ray tube voltage

until it provided sufficient penetrating power that the X-rays were not absorbed completely by the fluidized bed. The camera exposure time was then selected to be long enough that most of the camera's intensity range was used, while still minimizing the amount of time required for a complete scan. Finally, the projection intensity was fine-tuned with the X-ray tube current to provide a background intensity of approximately 90% of the full intensity range. The qualitatively "best" settings were determined to be: 150 kV tube potential, 3 mA tube current, and 1 second exposure.

To select the baseline parameter for the COR used in the reconstruction, the operator reconstructed a single slice of a CT with an arbitrarily selected COR. In this case the center of the projection, 384 pixels, was used (the COR value is specified as the number of pixels from the left edge of the projection). From there, several more versions of the same slice were reconstructed with different COR values until the qualitatively sharpest slice was found. The process of reconstructing slices and selecting the sharpest was repeated in an iterative manner to refine the COR until the changes to the COR became so small they no longer produced any visually distinguishable changes to the slice. The COR that yielded the sharpest slice is typically used as the COR for a full volume reconstruction, and thus it was selected to be the baseline COR. This baseline value was determined to be 384.56 pixels from the left edge of the projection. Note that, because the COR represents a mapping of where the projected centerline of the volume is on the projection, fractional pixel values are acceptable. Furthermore, the mapping from projection to volume in the reconstruction (the backprojection step), usually requires interpolation between pixels anyway, so a fractional value for the COR does not introduce any additional interpolation. Finally, it is important to note that since the COR is dependent on the physical setup of the CT scanner, the COR only

needs to be determined once and that value can be used for all scans taken with the same scanner geometry.

To analyze the sensitivity of the reconstruction to the selection of the COR parameter, the COR was varied to either side of the baseline value until the reconstruction started displaying the fluidized bed containment vessel as two concentric columns (a common artifact of a severely incorrect COR). To select the parameter ranges for the tube voltage, tube current, and exposure, each parameter was varied individually from the baseline value. All three variables were increased individually until either the maximum value allowed by the system was reached, or until the projection intensity exceeded the maximum measureable intensity on the detector. The parameters were decreased individually from baseline until the projection contrast was so low it became difficult to discern features in the flow. The X-ray parameters and CORs used for testing are shown in Table 5.1.

Finally, the CT scanner in this study also has optional filters to reduce beam hardening effects and variable camera binning to change the resolution of the scan. These parameters were held constant in all scans, using one 0.6 mm thick copper filter and one 1.6 mm thick aluminum filter, placed directly in front of the X-ray source, to remove low energy X-rays (and in turn, reduce beam hardening effects). The camera binning was set to 4 × 4 binning mode, yielding a projection resolution of 768 × 512 pixels. Also held constant was the camera sensor temperature (0 °C), the distance between the X-ray source and the detector (1880 mm), and the distance between the X-ray source and the center of the imaging region (1295 mm). The resulting baseline flow CT volume and the derived void fraction volume, as calculated by the method presented in Section 5.3.3, are shown in Figure 5.2.

Table 5.1: X-ray computed tomography acquisition and reconstruction parameters varied to test scan sensitivity.

Scan Set	Tube Potential (kV)	Tube Current (mA)	Exposure per Projection (s)	Center of Rotation (pixels)
Baseline	150	3.0	1.00	384.56
Tube Potential Variation	100	3.0	1.00	384.56
	120	3.0	1.00	384.56
	140	3.0	1.00	384.56
	160	3.0	1.00	384.56
	180	3.0	1.00	384.56
	200	3.0	1.00	384.56
Tube Current Variation	150	2.0	1.00	384.56
	150	2.5	1.00	384.56
	150	3.5	1.00	384.56
	150	4.0	1.00	384.56
Exposure Variation	150	3.0	0.50	384.56
	150	3.0	0.75	384.56
	150	3.0	1.25	384.56
Center of Rotation Variation	150	3.0	1.00	374.56
	150	3.0	1.00	379.56
	150	3.0	1.00	381.56
	150	3.0	1.00	382.56
	150	3.0	1.00	383.56
	150	3.0	1.00	384.06
	150	3.0	1.00	384.26
	150	3.0	1.00	384.36
	150	3.0	1.00	384.46
	150	3.0	1.00	384.66
	150	3.0	1.00	384.76
	150	3.0	1.00	384.86
	150	3.0	1.00	385.06
	150	3.0	1.00	385.56
	150	3.0	1.00	386.56
	150	3.0	1.00	387.56
	150	3.0	1.00	389.56
	150	3.0	1.00	394.56

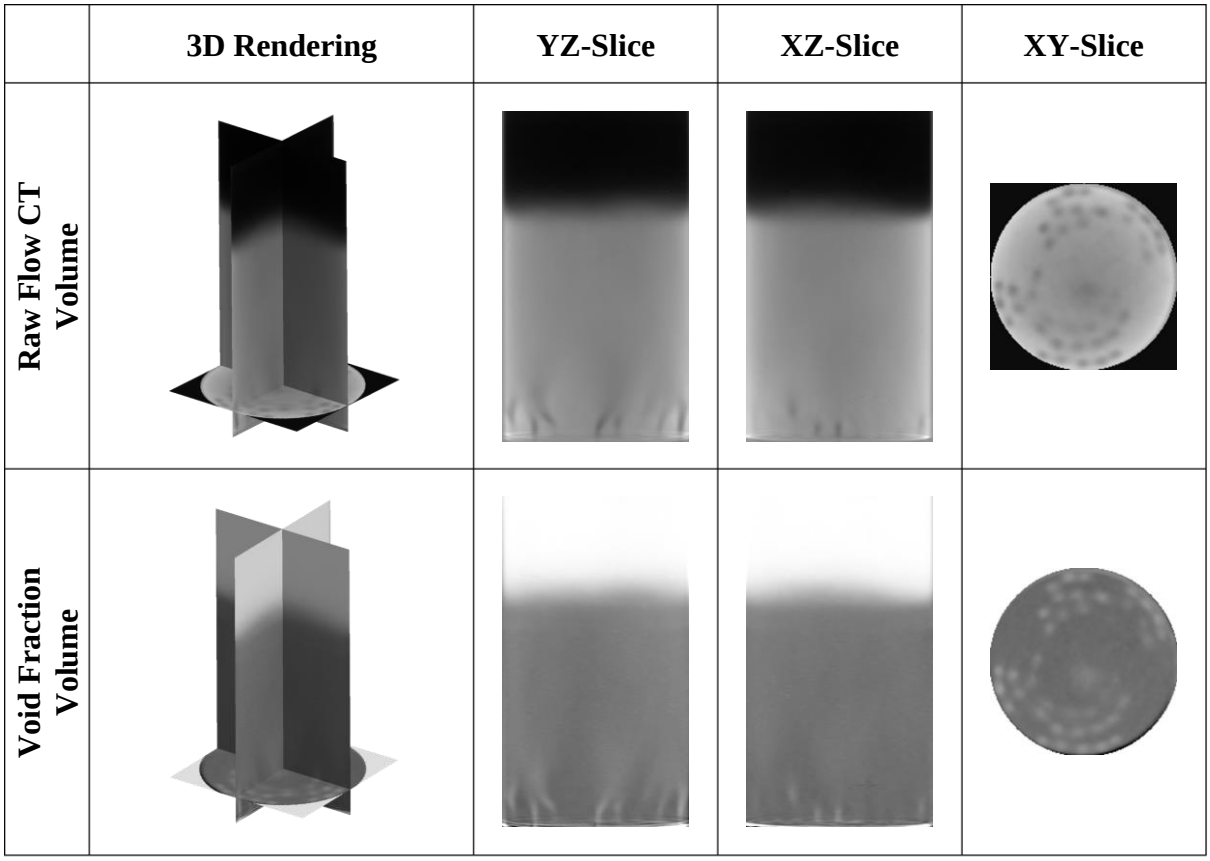

Figure 5.2: Four views of the baseline CT volume and void fraction volume. The planes in the 3D view are rendered at the same position the 2D slices are taken from. Note that numerous slices have been omitted for clarity.

5.3.3 Analysis Methods

To analyze the data from the varied CT parameter volumes, multiple methods were used. However, there are two processes that are common to all the methods. The first is to determine which part of the reconstructed volume contains the fluidized bed, called the region of interest (ROI). This is done by first inscribing a circle inside the bed on the top slice of the volume. This determines the diameter, in voxels, of the region of interest. Next, the location of the bottom of the bed must be determined. This is typically not a sharply defined location (due to artifacts introduced by the use of a fan beam reconstruction on a system that technically has cone beam geometry), so the center of the gradient is used. Finally, the region is extended vertically to include as much of the freeboard as possible. It is

important to note that when comparing ROIs between volumes, the ROIs must be exactly the same size, but they do not need to be in the same position within their respective volumes. The use of an ROI provides two benefits. First, from scan to scan the fluidized bed may not be placed in exactly the same position within the imaging region and the use of an ROI allows for the bed regions to be compared despite this misalignment. Second, the use of an ROI greatly reduces the amount of data to be processed without the loss of valuable information, since only the data from within the circulating bed is of scientific interest. For example, in this study, each ROI contained over 10,000,000 voxels, but the reconstructed region is generally much larger. Finally, note that if a small portion of the containing vessel is included within the ROI, the potential exists for errors to be introduced into the measurements. This is a relatively common artifact that causes measurements near the wall of the containment vessel to be unreliable.

The second process that is necessary for all scans is the calculation of the void fraction. While time averaged hydrodynamic structures may be visible in a raw CT scan of a multiphase flow, the real value of X-ray CT is its ability to uniquely determine the material density at each voxel. To do this, some form of calibration to a known reference is required, which is provided in the void fraction calculation in this study. To calculate the void fraction of a fluidized bed, three CT scans must be acquired using the same parameters, one with the vessel empty (the gas scan), one with the vessel full of the bed material but not operating (the bulk scan), and one with the multiphase flow operating at the desired condition (the flow scan). From these three scans, the time-average void fraction at each voxel is determined by:

$$\varepsilon = \frac{I_f - I_b + (I_g - I_f)\varepsilon_b}{I_g - I_b} \tag{5.1}$$

where I_b, I_g, I_f are the voxel intensities from the bulk, gas, and flow scans respectively (Heindel, 2011). For a static fluidized bed, the bulk void fraction, ε_b, is calculated by:

$$\varepsilon_b = 1 - \frac{\rho_b}{\rho_p} \tag{5.2}$$

where ρ_b is the bulk density of the granular material and ρ_p is the true particle density of the granular material, which is measured with a pycnometer. To show the impact of this void fraction calculation, the effects of parameter variation will be analyzed on both the raw flow CT and on the calculated void fraction volume.

Due to the number of voxels involved in each scan (on the order of 10,000,000 voxels within the ROI) and the inherent three-dimensional nature of the data, it is challenging to directly compare one scan to another without reducing the data in some fashion. This reduction typically relies on traditional descriptive statistics, primarily the arithmetic mean and standard deviation. However, the traditional algorithms for calculating these statistics do not handle large data sets well, thus it is imperative to use a version of the arithmetic mean and standard deviation formula that is both numerically stable and can process data in a parallel fashion. To meet these requirements, the algorithms presented by Bennett et al. (2009) were used. For convenience, the formulas are also presented here. The computation of the average and standard deviation is a two-step process. The first step is to split the data into smaller chunks. These chunks are processed in parallel, but the local data within each chunk are processed sequentially using a local update formula. These local formulas are:

$$\mu'_{l,i} = \sum_{i=1}^{n'_l} \left(\frac{I_i - \mu'_{l,i-1}}{i} \right) \tag{5.3}$$

$$m_{l,i} = \sum_{i=1}^{n'_l} \left((I_i - \mu'_{l,i-1})(I_i - \mu'_{l,i}) \right) \tag{5.4}$$

where $\mu'_{l,i}$ is the arithmetic mean of the first i voxel intensity values I_i within the local data chunk l of size n'_l, and $m_{l,i}$ is the second statistical moment of the same intensity values. The local means always initialize to zero, that is to say $\mu'_{l,0} = 0$. Once the mean and second moment for all the data chunks have been calculated the mean and second moment for the entire data set can be calculated by:

$$n_j = \sum_{l=1}^{j} n'_l \tag{5.5}$$

$$\mu_j = \sum_{j,l=1}^{p} n'_l \left(\frac{\mu'_l - \mu_{j-1}}{n_{j-1} + n'_l} \right) \tag{5.6}$$

$$m_j = \sum_{j,l=1}^{p} \left(m_l + n_{j-1} n'_l \frac{\left(\mu'_l - \mu_{j-1}\right)^2}{n_{j-1} + n'_l} \right) \tag{5.7}$$

where n_j is the number of voxels in the first j data chunks, p is the total number of data chunks, μ_j is the mean of the first j data chunks, and m_j is the second moment of the first j data chunks. When $j = p$, μ_j is the mean of the entire data set, or simply μ, m_j is the second moment of the entire data set, or simply m, and n_j is the total number of voxels in the entire data set, or simply n. Once again, the mean is initialized to zero (i.e., $\mu_0 = 0$). Finally, the sample standard deviation, σ, is determined from the second statistical moment by:

$$\sigma = \sqrt{\frac{m}{n-1}} \tag{5.8}$$

While the effects of changing the parameters can be analyzed by simply taking the average intensity value of the entire CT, such an analysis masks any spatial dependencies that might indicate the cause of the change. An example of such a change would be increased beam hardening effects due to a lower X-ray tube potential, which would cause a greater reduction in intensity in the center of the flow than at the edges. The volumes were

analyzed for spatially dependent variations in two ways. First, the annular averages were calculated and plotted using the method developed by Drake and Heindel (2012a). The second method calculates the average intensity of each slice moving vertically through the bed. Both of these methods reduce the dimensionality of the data and use the previously described method for calculating the mean and standard deviation of the data.

Finally, to better understand spatial differences introduced by changing the X-ray parameters, a method to calculate the per voxel percent difference from a baseline measurement is required. Because the normal range of values for a raw CT ranges from -1000 to +3000 Hounsfield Units (HU) and includes 0 (Heindel, 2011), the percent difference could be undefined for a voxel; hence, the percent difference for raw CT values will not be considered. The potential for undefined values also exists in the void fraction results; however, in void fraction there is a logical way to correct for this. Although the possible range of values for void fraction is 0 to 1, fluidized beds contain a granular material instead of a liquid and the lowest theoretical void fraction possible in the bed is the bulk void fraction, as given by Eq. (5.2). For the glass beads used in this study, the bulk void fraction is 0.40, thus any value less than this must be erroneous. Such erroneous values are typically introduced by including a small piece of the containment vessel wall in the ROI. Based on this, it is safe to assume that any extremely small calculated void fraction values may be excluded from the percent difference calculation, leading to:

$$D = \begin{cases} \text{Undefined}, & I_{ref} \leq 0.01 \\ \dfrac{I_m - I_{ref}}{I_{ref}}, & I_{ref} > 0.01 \end{cases} \tag{5.9}$$

where I_{ref} is the baseline voxel intensity and I_m is the measured voxel intensity in the varied parameter CT scan. The percent difference calculation is performed on a voxel-by-voxel

basis and results in a new volumetric data set that can be analyzed using the previously presented methods. By setting erroneously low void fraction values to undefined when calculating the percent difference, those voxels are omitted from the calculation of the statistics. This results in less than 0.03% of all voxels being excluded from the calculations.

5.4 Results and Discussion

As noted in Section 5.3.2, four different parameters were varied from their baseline value: X-ray tube potential, X-ray tube current, detector exposure time, and the COR used in the reconstruction. In the following sections, the effects of each of these variations will be examined using the methods from Section 5.3.3. First, the error that exists under consistent conditions must be determined to provide a baseline for comparison. This error was examined by Drake and Heindel (2011) on a per-plane and per-annulus basis. Based on their analysis, the error was found to be approximately ±4% of full scale for the calculated average void fraction values. To verify this error, a series of 20 CT scans were taken for each of the required scans (gas, bulk, and flow) at the baseline acquisition parameters. From these data, it was found that the average CT intensity of the flow CT within the ROI was 538.5 HU and the average void fraction within the ROI was 0.627. The average per voxel percent difference for the void fraction was 1.8%. While this error analysis is more detailed than that of Drake and Heindel, it does not account for variations that could occur in the flow from day to day when scanning, which Drake and Heindel did include. Thus, taking into consideration Drake and Heindel's results, a baseline percent error of ±3% of full scale will be used as a reference in this work. Note that this error is only used for comparison, and does not affect the results of any of the calculations.

The impact of changing the X-ray parameters from the baseline values on the raw CT and void fraction values, relative to the baseline error, will be presented next. First, a cursory review of overall results will be given by looking at whole volume averages in Section 5.4.1. Next, Section 5.4.2 will examine any spatial variations within the cases where the X-ray tube or detector parameters are changed and discuss possible causes for the spatial discrepancies. Finally, Section 5.4.3 will treat the effects of changing the COR individually, since changes to this parameter only effect the final reconstruction, not the raw data collected by the CT scanner.

5.4.1 Effects on Whole ROI Averages

To get a high level understanding of any variations that may occur in a CT due to the acquisition parameters, the average CT intensity value within the entire ROI is examined. The results of this are shown by the closed symbols in Figure 5.3. Similarly, a cursory view of any variations that exist in the void fraction information can be obtained by averaging all the void fraction values within the ROI of the void fraction volume. These values are shown by the open symbols in Figure 5.3. Note that, since the baseline condition is the same for all tests, the symbols for zero percent deviation from baseline overlay one another.

From this information, it appears that there is not a strong dependence on the average CT value due to changes in either current or exposure. However, the average CT intensity strongly decreases as the X-ray tube voltage increases. Similarly, observing the average void fraction values, changes in X-ray tube current and camera exposure do not affect the void fraction significantly. Additionally, the average void fraction does not change significantly with changes in X-ray tube voltage. This provides an initial indication that the void fraction

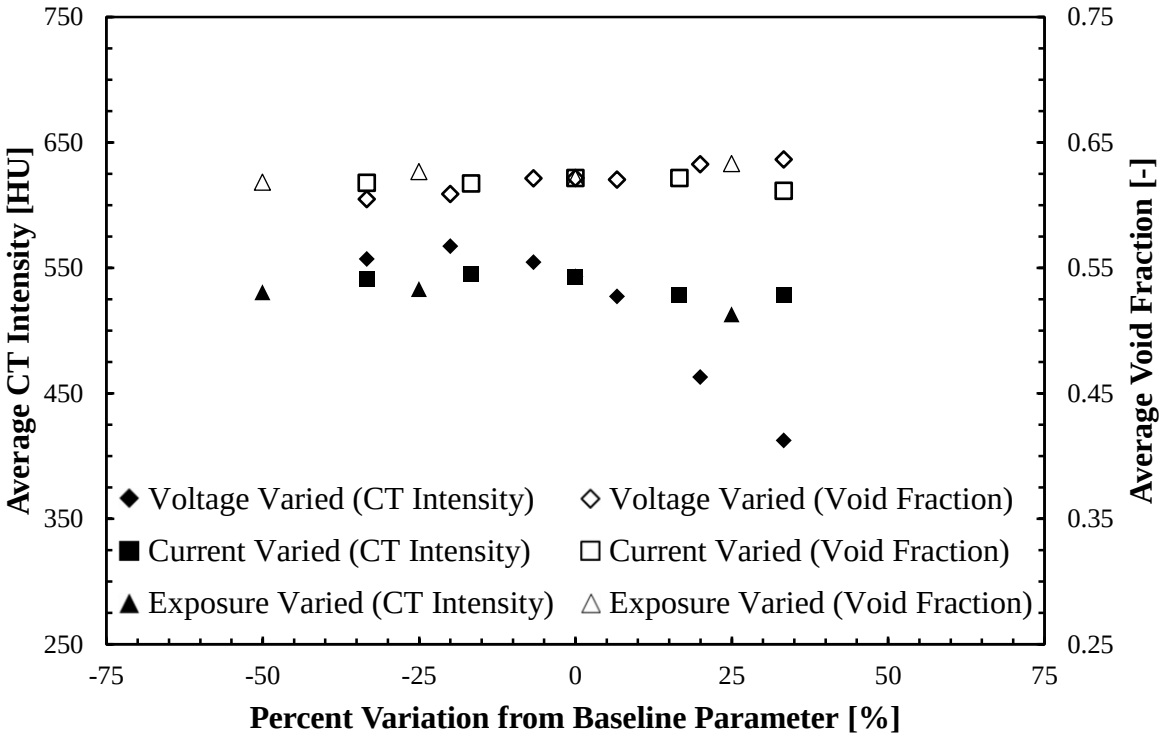

Figure 5.3: The average CT and average void fraction for the entire ROI.

calculation is correcting for systemic changes introduced by changing X-ray acquisition parameters. The sources of the observed changes will be further examined in Section 5.4.2.

Next, consider the effect of changing the COR on the average flow CT intensity and void fraction. This information, shown in Figure 5.4, again uses closed symbols for average flow CT intensity and open symbols for average void fraction. In both cases, there appears to be very little change from the baseline. The worst case difference (baseline COR minus 10 pixels) is less than 3% difference from baseline average CT intensity and less than 0.5% difference from the baseline average void fraction. However, this result is somewhat misleading, as will be discussed in Section 5.4.3.

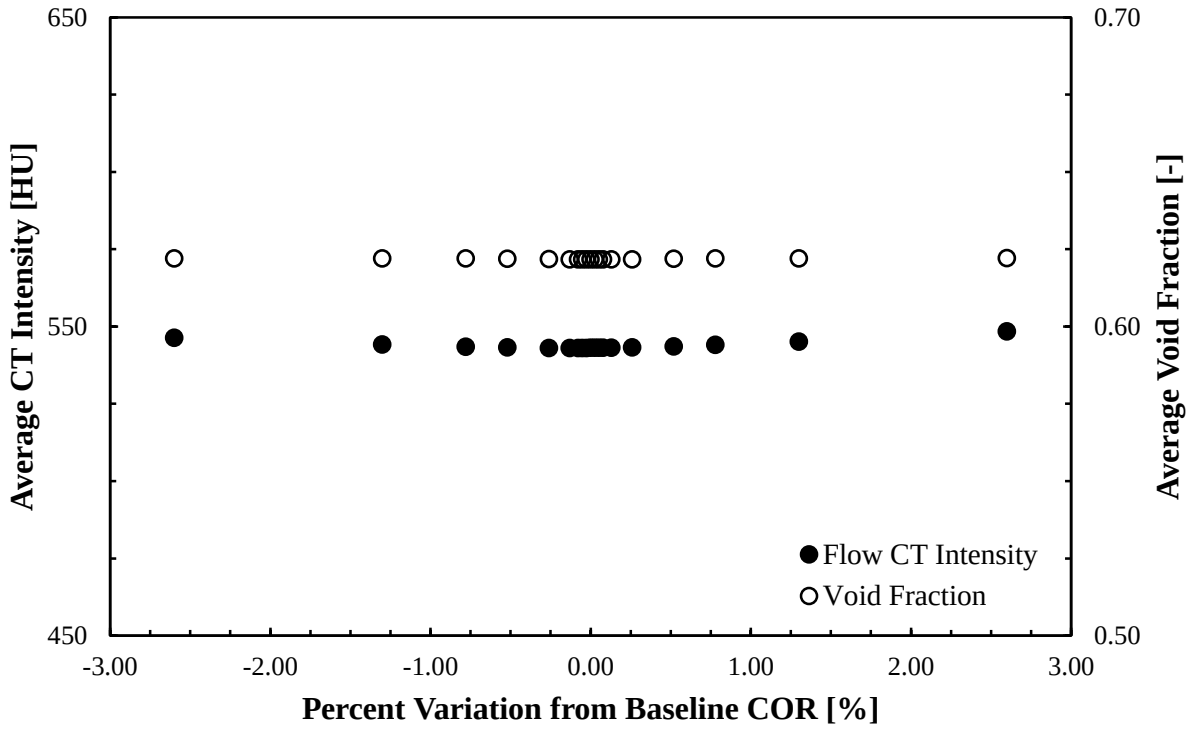

Figure 5.4: The average CT intensity for the flow condition and average void fraction value for the entire ROI with varied CORs.

5.4.2 Effects of Tube Current, Voltage, and Detector Exposure

The average analysis of tube current in Section 5.4.1 indicates that changes in the X-ray tube current do not significantly influence either the raw average flow CT intensity values or average void fraction values. To further this analysis, consider Figure 5.5 which shows the average annular flow CT intensity and Figure 5.6 which shows the average annular void fraction for the various X-ray tube current settings. For the flow CT annuli, currents from 2.0 to 3.0 mA appear to all provide nearly the same results, with the 3.5 and 4.0 mA cases providing slightly lower average intensity values. Furthermore, any changes that exist in the average CT values appear to be different by a constant amount across all annuli. This is consistent with a uniformly brighter image being recorded by the X-ray detector. There are some slight variations near the edges of the bed; however, this is consistent with occasional vessel wall inclusions in the ROI as previously discussed.

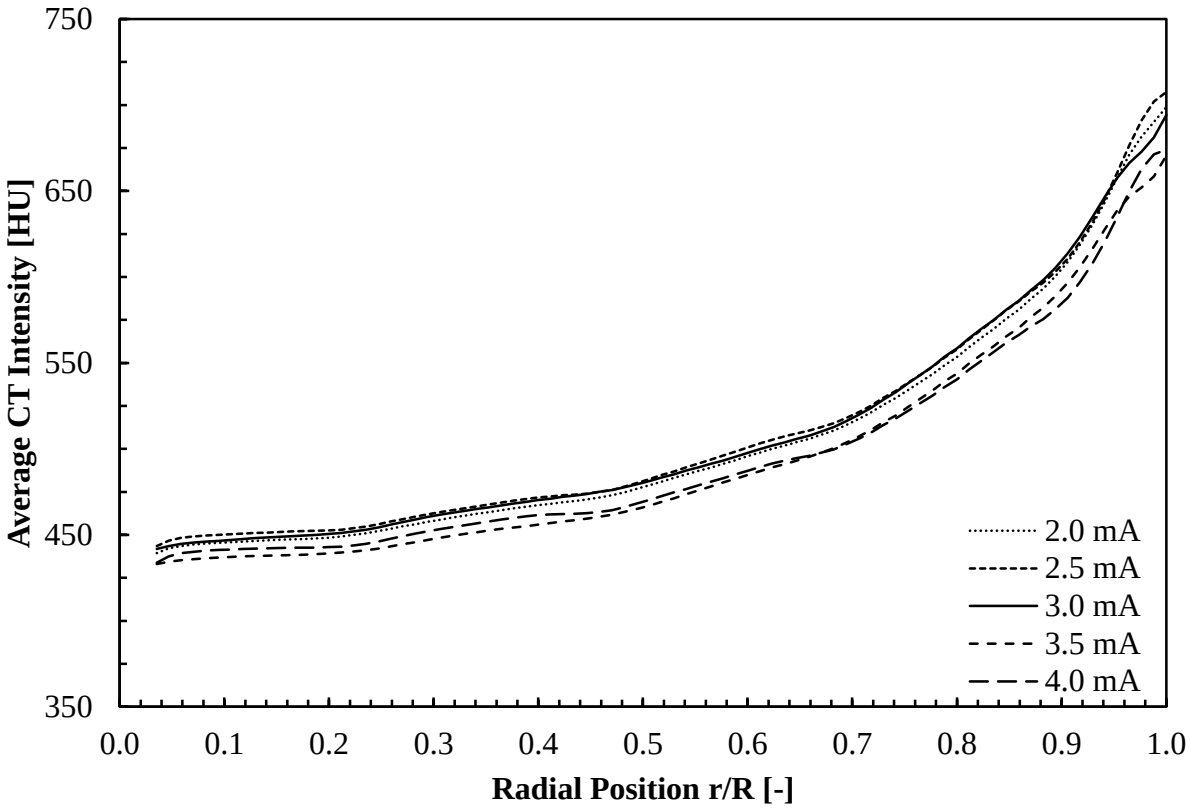

Figure 5.5: The average annular CT intensity of the flow CT for varied X-ray tube currents.

The average annular void fraction for the current varied condition reinforces this result. As shown in Figure 5.6, there are some variations in void fraction between currents; however, they are well within the 3.0% expected baseline variation. Furthermore, there does not appear to be a consistent pattern to the differences with respect to the current variations. The 4.0 mA and 2.5 mA cases both have a slightly lower average void fraction, while the 3.5 mA case is nearly identical to the baseline. These results indicate that variations in X-ray tube current do not impact the results of X-ray CT measured void fraction, despite the variations it introduces to the raw CT intensities.

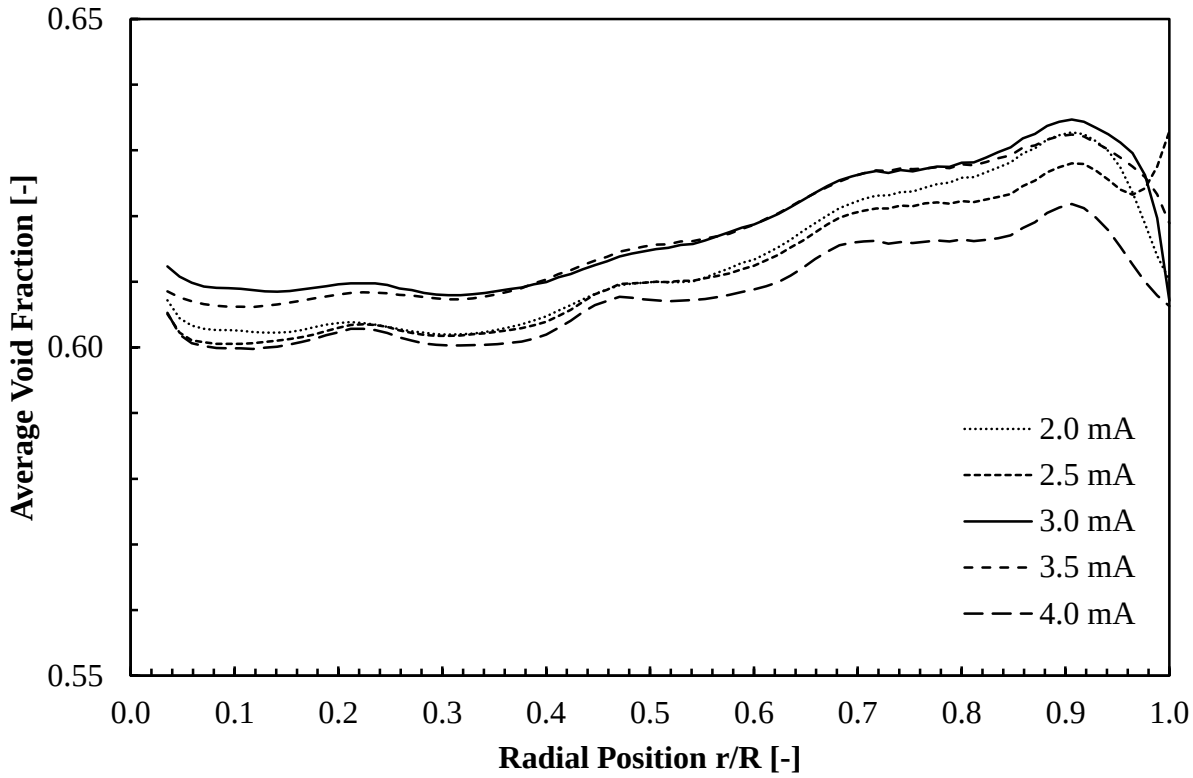

Figure 5.6: The annular average void fraction for varied X-ray tube currents. Note, only 10% of the full range (0 to 1) of the average void fraction is shown in order to show differences more clearly.

Changes to the X-ray detector exposure would be expected to provide similar results, as changing the X-ray tube current and detector exposure both change the number of X-ray photons incident on the detector. Though the method is different (an increase in tube current generates more photons per second and an increase in exposure allows more total photons to be collected by holding the shutter open longer), both are directly related to the total brightness of the recorded image. To verify this, first consider Figure 5.7, which shows the average slice intensity in the flow CT, where 466 slices compose the measurement domain. The average slice intensity generally decreases as the exposure increases, the same effect observed with increasing current. This change is particularly clear in the freeboard region of the fluidizied bed (heights of h/D > 1.4). It should also be noted that the height where the flow begins to transition from the bed region to the freeboard region is lower in the 0.75 s

and 1.25 s cases than in the 1.00 s and 0.50 s cases. This is suspected to be the result of slight variations in the fill level of the bed. There is also a distinct increase in CT intensity between h/D = 0.05 and h/D = 0.10. This is the result of jetting immediately above the aeration plate (Escudero and Heindel, 2015).

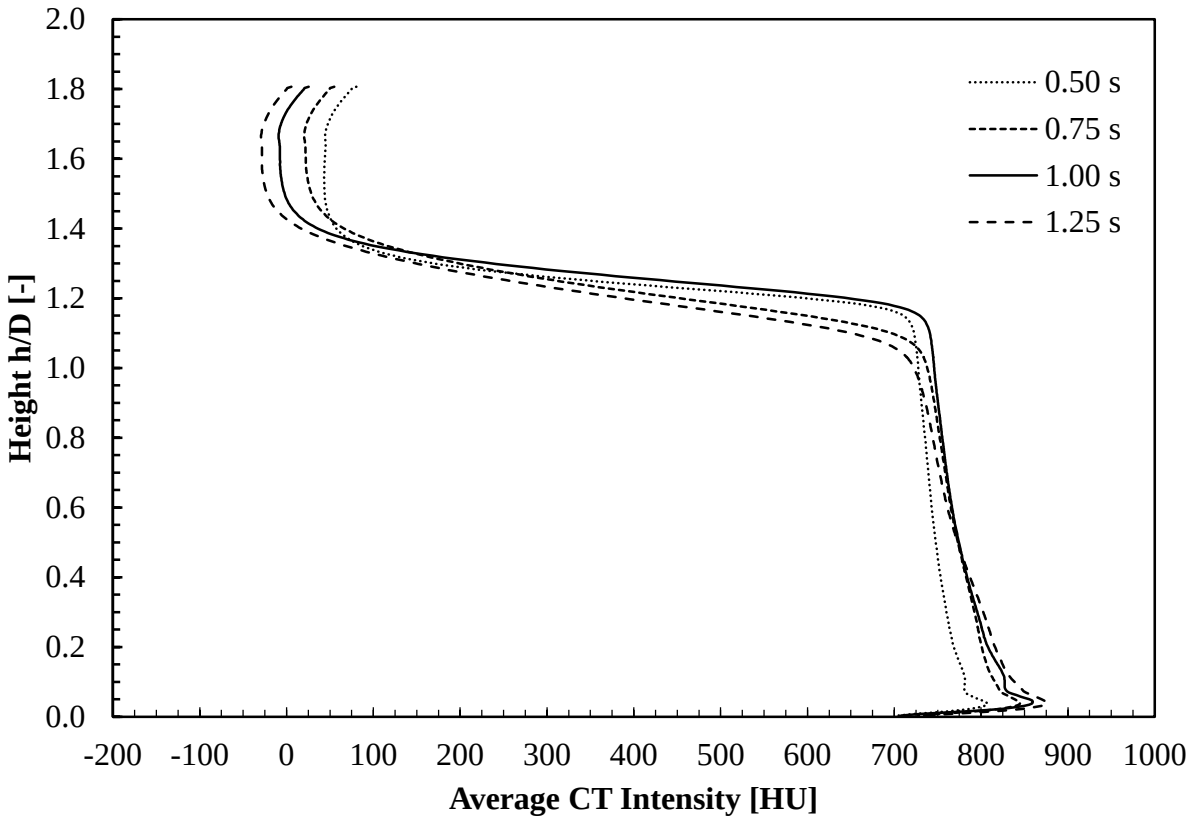

Figure 5.7: The slice average CT intensity of the flow CT for varied X-ray detector exposure times.

Similar to the current varied case, the changes in the flow CT caused by changing the exposure should be canceled out by calculating the void fraction. This is verified in Figure 5.8, which shows the slice average void fraction for the varied X-ray detector exposure times. The anomaly in the transition from the bed region to the freeboard region is still present in the void fraction, further indicating that it may be an artifact of the flow, not the measurement. In the freeboard region however, where large changes in CT intensity were seen, there is almost no difference in the slice average void fraction. Some variations in void fraction measurement do exist in the bed region, however, they are within the ±3% difference of expected variation.

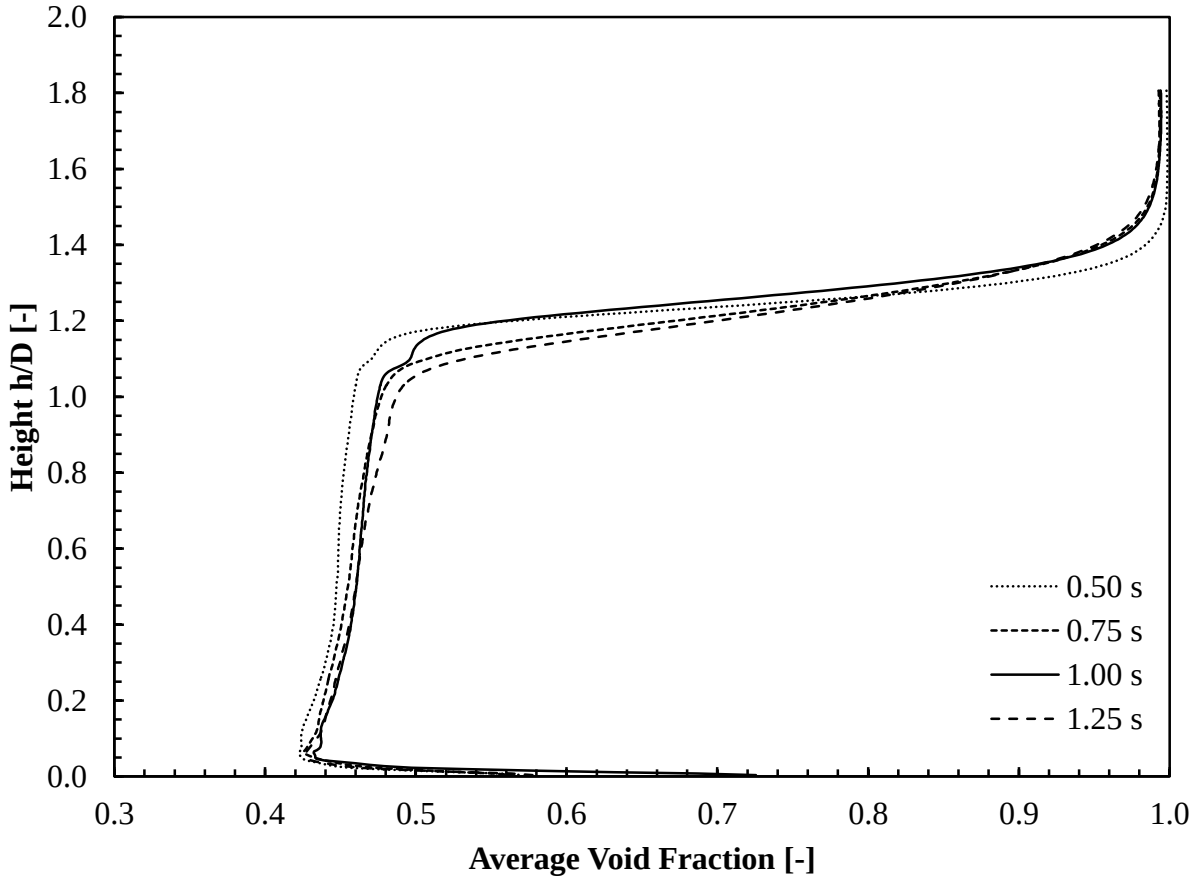

Figure 5.8: The slice average void fraction for varied X-ray detector exposure times.

Unlike the effects of current and exposure variation, the average CT intensity decreases significantly at higher voltages (e.g., Figure 5.3). This effect is shown in Figure 5.9, where the annular average CT intensity is plotted as a function of radial position for a range of X-ray tube voltages. The decrease in CT intensity with increasing X-ray tube voltages is to be expected since the raw CT values are not calibrated to the materials used in the fluidized bed and a higher X-ray tube voltage will result in a brighter image on the X-ray detector. However, there is also a sharp increase in the CT intensity within the outer 25% of the vessel that is more prominent in the lower voltage CTs and flatter in the higher voltage CTs. This provides a strong indication that the CT values near the center of the scans are artificially lowered due to beam hardening effects, which was not corrected for in this study, particularly at lower X-ray tube voltages (Ketcham and Carlson, 2001). This is further evidenced by the void fraction percent difference annular averages (seen in Figure 5.10) that show greater differences toward the center of the fluidized bed than at the edges (the large variations at the extreme edges are likely due to parts of the containment vessel getting erroneously included in the ROI). This strongly indicates that there are unaccounted effects of beam hardening that influence the data. However, even so, the void fraction values are within the expected range of error for the CT scanner. Based on these observations, it is strongly recommended that X-ray users correct for beam hardening effects whenever possible, as was done in previous studies by Drake and Heindel (2012b), Franka and Heindel (2009), and others. However, when such a correction is not possible, it is recommended that the system be scanned with as high an X-ray tube voltage as possible to minimize the effects of beam hardening.

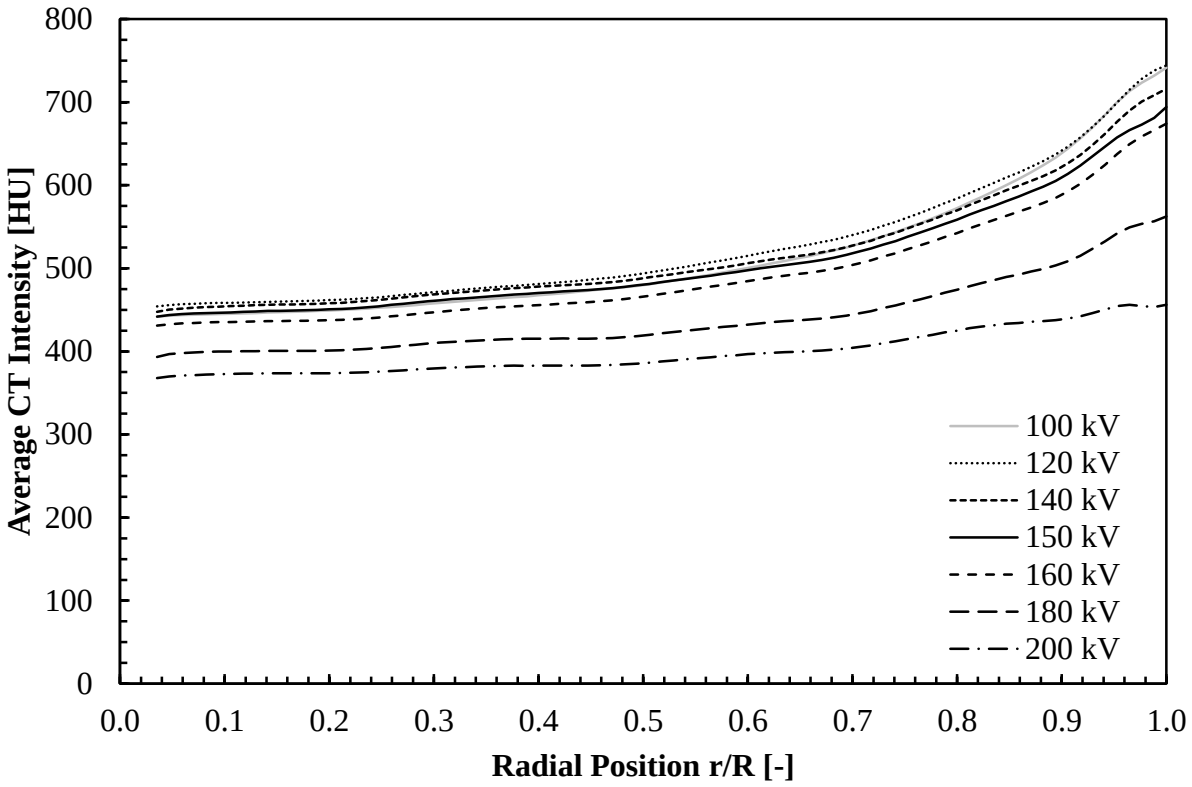

Figure 5.9: The annular average CT intensity for flow CTs with varied X-ray tube voltages.

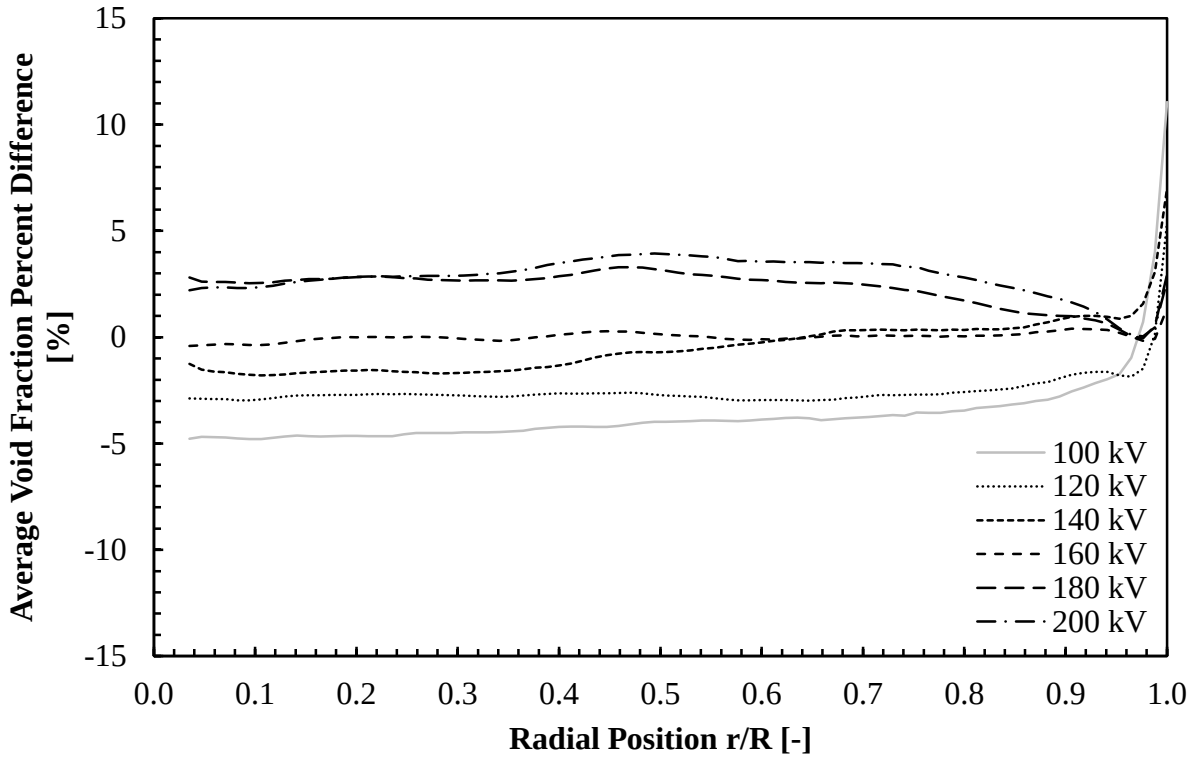

Figure 5.10: The annular average percent difference of the void fraction values for varied X-ray tube voltages. Recall that 150 kV is the reference condition.

When analyzing the plane averages for the voltage variation, the beam hardening effects are hidden, as they are averaged out within the plane; however, one anomaly does appear. As seen in Figure 5.11, the average per plane intensity value increases with decreasing tube voltage, except in the case of 100 kV, which has a much lower average plane intensity within the bed region than expected if the trend held. The suspected cause of this anomaly is the extremely low contrast (roughly 6% of full range on the X-ray detector) within the bed is masking flow structures. Further evidence for this can be seen by observing contour maps of the bed slices (Figure 5.12), which show some non-uniform flow structures within the 150 kV and 200 kV CT slices. However, the 100 kV CT slice shows no flow structures within the bed region. While the anomaly does not appear in the void fraction plane averages (Figure 5.13), it is proposed that, because the measured void fraction is so close to the void fraction of the packed bed to begin with, the missing flow structures do not have enough size to significantly change the average plane ROI in the 100 kV voltage case.

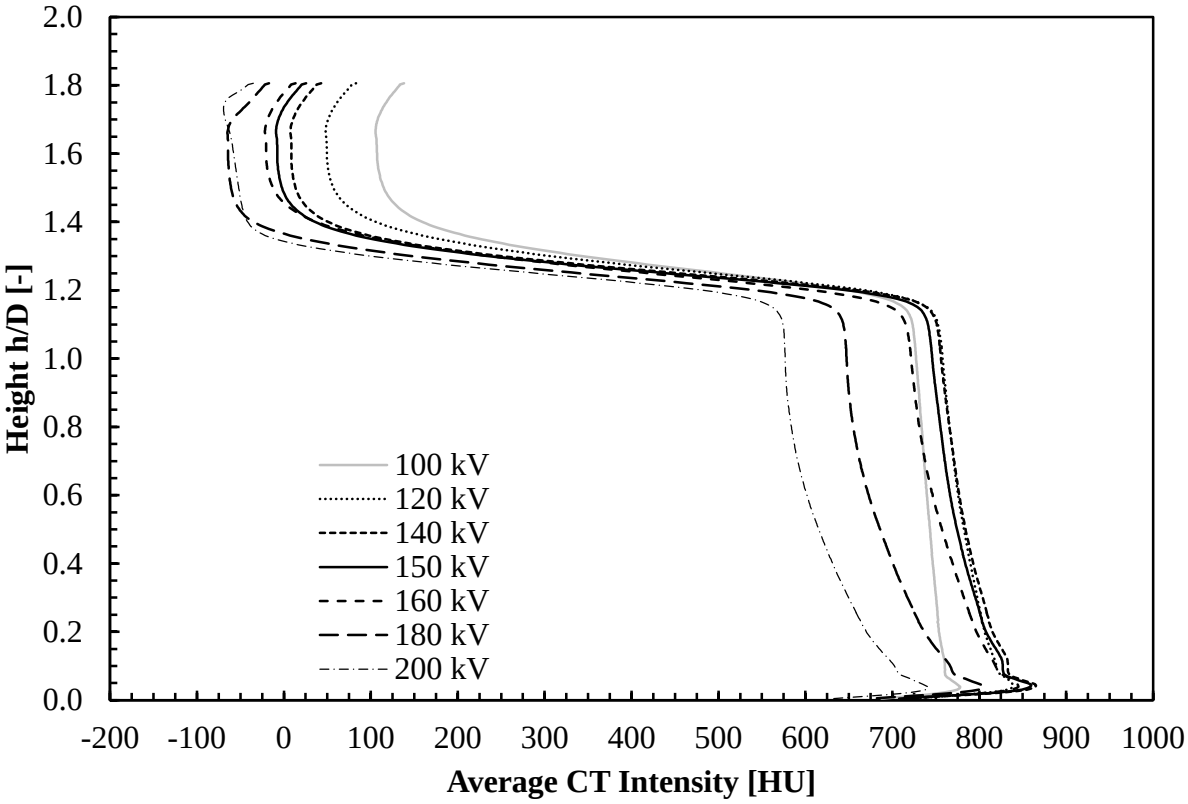

Figure 5.11: The average CT intensity of the flow CT by slice for varied X-ray tube voltages.

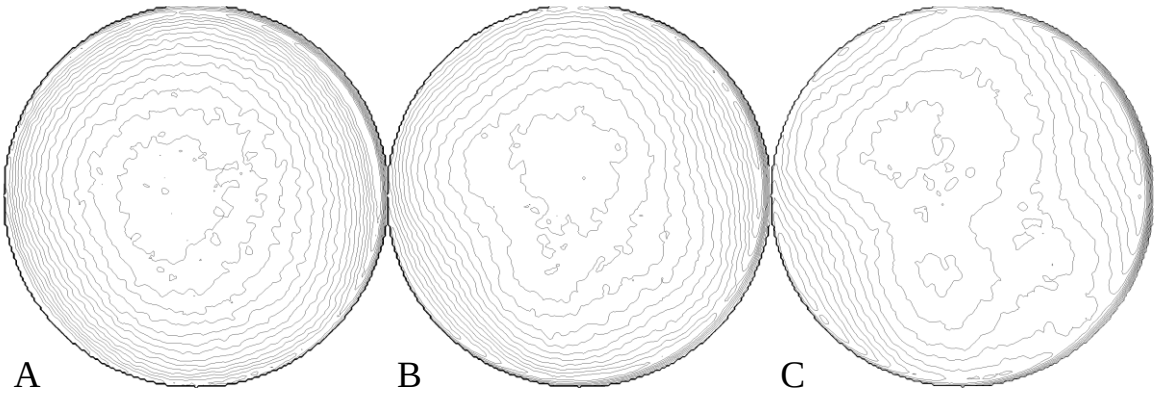

Figure 5.11: The flow CT slice contour maps at h/D=0.64 for X-ray tube voltages A) 100 kV, B) 150 kV, and C) 200 kV, where h is the height above the aeration plate and D is the fluidized bed diameter. The contours are at intervals of 25 CT values from I = 400 to I = 1000.

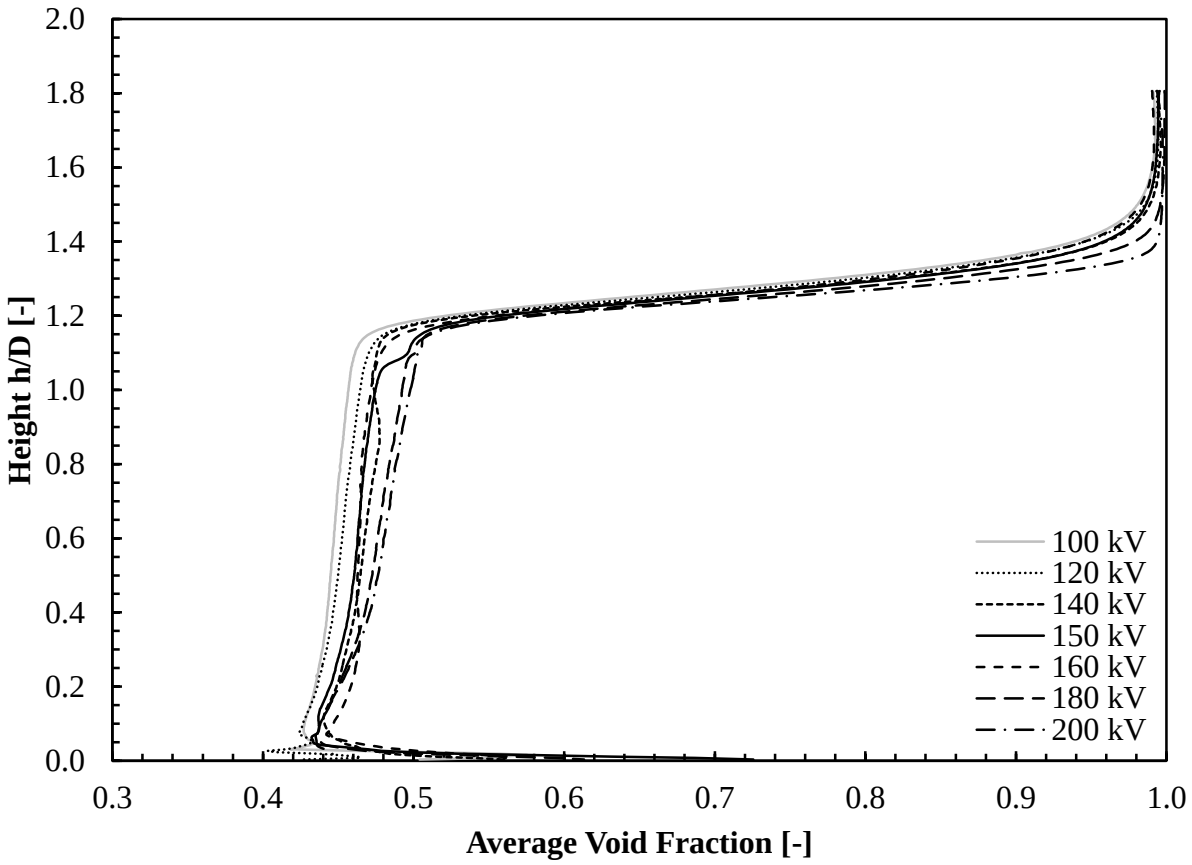

Figure 5.13: The average void fraction by slice for varied X-ray tube voltages.

5.4.3 Effects of Center of Rotation Variation

As noted in Section 5.4.1, there is almost no change in the average CT intensities introduced by changing the COR used in the reconstruction. However, from the standpoint of visual error, the errors introduced by changes in the COR are more dramatic than those introduced from changes to other parameters. As shown in Figure 5.14, a change in the COR changes the resulting CT from accurately representing the geometry of the fluidized bed to showing it as two concentric columns with a change of only 10 pixels, a mere 2.6% change in COR. Clearly, from a geometric standpoint, the effect of changing the ROI is significant. However, from the viewpoint of average void fraction, the effects are less distinct. This difference can be attributed to the nature of the FBP algorithm. The backprojection step of

the FBP algorithm combines every pixel from every projection that contributes to a given point, and thus does a significant amount of averaging. By changing the COR of the reconstruction, the exact location within the ROI where data from a projection contribute to the volume may change, but it is still contained within the ROI in most cases. This results in almost no change in the average ROI intensity. Thus, looking solely at the average ROI intensity provides an incomplete indication of what is occurring within the volume in the case of changing the COR.

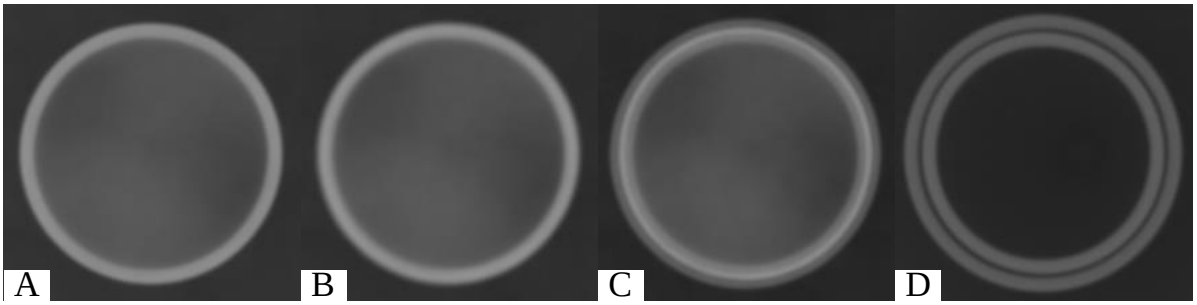

Figure 5.14: The baseline flow CT sliced at height h/D=1.30, reconstructed at the baseline COR A) +0.0 pixels B) +2.0 pixels C) +5.0 pixels and D) +10.0 pixels. Note how the fluidized bed walls start to appear as two concentric columns as the COR increases from the baseline. Similar artifacts are seen as the COR is decreased from the baseline (not shown).

A better way to analyze the errors introduced by changing the COR is to look at the annular data, as the COR introduces changes within a slice instead of across slices. Figure 5.15 shows the average per voxel void fraction percent difference from the baseline COR for several variations on the COR. It is clear that, while the average percent difference introduced into the volume is small, it is strongly dependent on the location within the ROI. It should also be observed that the change is roughly symmetric, i.e., a change of +10 pixels to the COR will introduce roughly the same error as a change of -10 pixels. However, the most important result is that even a change of ±5 pixels to the COR, produces a distinguishable change from the baseline COR (Figure 5.14), but it introduces on average less than 0.5% difference in the final results (Figure 5.15).

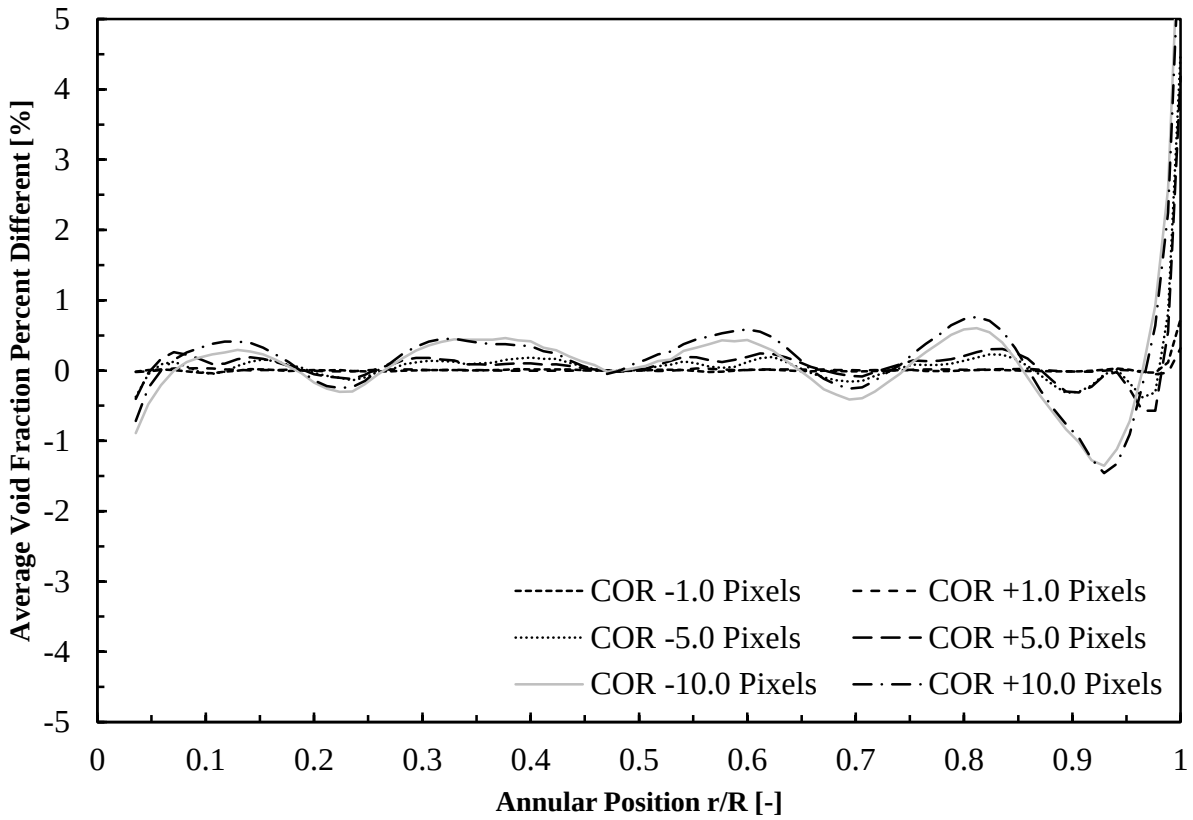

Figure 5.15: The annular average void fraction percent difference from baseline with changes in COR. Note that several CORs have been excluded from the figure for clarity.

It is also of note that the average percent error has a roughly sinusoidal profile in the radial direction when the COR is varied. This radial variation in error is due to the phenomenon of jetting, a phenomenon which has been previously studied in this system by Escudero and Heindel (2015). The holes in the aeration plate are arranged with a single hole in the middle, surrounded by four concentric rings of holes. The location of these rings corresponds with the locations of the valleys in the errors. This radially dependent error occurs because the jets low in the fluidized bed create regions of high void fraction, which are blurred with the surrounding regions of lower void fraction when the COR is varied from the true COR of the CT system. This results in a lower average void fraction at the annuli where the jets occur, and a higher average void fraction in the annuli immediately adjacent the jets when the COR is varied.

5.5 Conclusions

A sensitivity analysis of X-ray computed tomography data to changes in the acquisition and reconstruction parameters has shown some important features for consideration. First, when changing X-ray tube voltage, operators need to be aware of the potential for beam hardening effects and use corrective algorithms to compensate for the effects whenever possible. When correcting for beam hardening effects is not possible, or not practical, this research has shown that X-ray CT can still provide acceptable quantitative information, but the user needs to be aware that there will be an increase in error. Furthermore, when beam hardening is not corrected for, a slightly lower average void fraction in the central region of the fluidized bed may be the result of beam hardening and not actual flow structures. Changes to the X-ray tube current and X-ray detector, however, will change the raw intensity values of a CT, but do not introduce any significant errors in calibrated measurements, such as void fraction calculations. Second, when selecting a center of rotation for reconstruction, this research has shown that while the visual geometry changes are large, the impact on the average void fraction is relatively small. Furthermore, provided the operator selects a center of rotation that provides a reconstruction free of major geometric distortions, even if the center of rotation is suboptimal, it will have a negligible effect on the final results. Finally, while there are a few potential pitfalls when making large variations to the X-ray tube voltage, overall, any potential changes to the results of a CT scan, when properly accounted for through calibration or reference images (e.g., by using the scans to calculate void fraction), are within the expected error of the system. Thus, while different users are likely to select different operating conditions that appear “best” to them, this human variability will not significantly impact the results of the CT scan or resulting void fraction calculations.

CHAPTER 6:

APPROXIMATE 3D RECONSTRUCTION TECHNIQUES FOR CHARACTERIZING MULTIPHASE FLOWS FROM X-RAY STEREOGRAPHIC IMAGING

In the previous two chapters, it has been shown that it is possible to acquire radiographs at high speed using a tube source, and that properly calibrated computed tomography scans are insensitive to acquisition parameters. This chapter builds upon that work by merging the high-speed imaging capability of radiography and the ability of computed tomography to determine X-ray attenuation coefficients at a 3D point reliably. This is accomplished using two approximated computed tomography reconstructions to generate tomographic slices from only two X-ray projections using X-ray stereographic imaging, which can be acquired at high speed. In so doing, this work contributes to both objectives one and three as outlined in Section 1.2. This chapter is based on a draft paper to be submitted to *Flow Measurement and Instrumentation*.[3]

6.1 Abstract

In the three-dimensional imaging of multiphase flows, a tradeoff exists between temporal resolution and spatial resolution. Techniques such as magnetic resonance imaging and X-ray computed tomography provide excellent three-dimensional spatial resolution, but take a long time to acquire (on the order of minutes or hours, depending on the specifics of the system). Other techniques, such as electrical impedance tomography and X-ray particle

[3] Based on draft Morgan, T. B., Vance, J. M., and Heindel, T. J. (2017) Approximate 3D Reconstruction Techniques for Characterizing Multiphase Flows from X-ray Stereographic Imaging. To be submitted to *Flow Measurement and Instrumentation*

tracking velocimetry, are capable of achieving very high temporal resolutions, but have very low spatial resolution, or are limited to the measurement of a relatively small number of tagged particles. This research examines the possibility of combining the techniques of X-ray computed tomography and X-ray stereography to achieve both a high spatial resolution and a high temporal resolution. This is done by testing the capabilities and limitations of two methods for approximating a computed tomography reconstruction from only two X-ray stereographic projections, combined with a priori information from computed tomography scans of the system in a static state.

6.2 Introduction

One of the limitations of noninvasive multiphase flow imaging is there is always a significant tradeoff between achieving a high three-dimensional resolution, and the speed of the imaging. For example, X-ray computed tomography (CT) and magnetic resonance imaging (MRI) can both achieve high spatial resolutions in three-dimensions, in excess of 100,000,000 voxels. However, because of the long imaging time of these systems (on the order of minutes or hours, depending on the system and acquisition parameters), they are only suitable for acquiring time averaged or static data (Chaouki et al., 1997; Fukushima, 1999; Heindel, 2011). Other systems, such as electrical impedance tomography systems, are capable of achieving high temporal resolutions (in excess of 1000 frames per second), but have very limited spatial resolution (Chaouki et al., 1997; van Ommen and Mudde, 2008). Similarly, X-ray particle tracking velocimetry is capable of measuring a particle position with good accuracy and temporal resolution, but is limited to measuring the position of a small number of particles within a flow (Kingston et al., 2014; Morgan and Heindel, 2010; Seeger et al., 2001; Shimada et al., 2007).

To overcome the inherent tradeoff between spatial and temporal resolution in noninvasive multiphase flow imaging, several methods of improving the temporal resolution of X-ray CT have been proposed. Bieberle et al. (2010) developed an X-ray system that uses a scanned electron beam to generate X-rays from different points along a linear tungsten target. This system is capable of acquiring two tomographic slices at temporal speeds of at least 2500 frames per second (FPS) and spatial resolutions of 1 mm (0.04 in). While this system was only designed to acquire two tomographic slices, the concept could be extended to multiple slices.

Another concept that has been proposed for increasing the speed of CT measurement, for both X-ray CT and γ-ray CT, is the use of multiple radiation sources and detectors. The limitation of such systems is the traditional filtered backprojection algorithm for CT reconstruction requires numerous projections around an object to generate an accurate tomographic reconstruction (Mudde et al., 2008). Therefore, systems based on a small number of source-detector pairs require more advanced reconstruction techniques. Mudde et al. (2005, 2008) examined the use of both three and five source-detector pairs using the simultaneous algebraic reconstruction technique (SART) for reconstruction (Andersen and Kak, 1984). Mudde et al. found that five source-detector pairs were required to achieve adequate spatial resolution using this system; however, they were able to achieve a spatial resolution of 5 mm (0.20 in) and a temporal resolution of 200 FPS. A similar approach was used by Hu et al. (Hu et al., 2005), who used two X-ray source detector pairs to do tomographic imaging of three-phase flows. To solve the reconstruction problem from only two sources, Hu et al. formulated the reconstruction problem as an underdefined system of equations, and then added smoothing equations to force intensity continuity from voxel to

voxel. This results in an overdefined system of equations, which can then be solved using a matrix pseudoinverse. Using this system, they were able to image three-phase flows at a speed of five FPS with a spatial resolution of 4 mm (0.16 in). However, this system was not able to achieve clear reconstructions of coaxial multiphase flows (Hu et al., 2005).

This work will examine how a two X-ray source-detector pair system performs for high-speed tomography using approximated CT reconstructions. The experimental setup used to test these algorithms is presented in Section 6.3. Two methods to approximate the CT reconstruction from only two X-ray projections are presented in Section 6.4, with the resulting reconstructions of experimental data presented in Section 6.5.

6.3 Experimental Setup

The data for validating the approximate 3D reconstruction algorithms was generated by taking computed tomography scans of an X-ray phantom (a test object of known size, shape, and material properties) in the X-ray Flow Visualization (XFloViz) facility at Iowa State University. This system, seen in Figure 6.1, contains two Lorad LPX 200 X-ray sources, which are capable of tube potentials up to 200 kV, and a maximum power output of 900 W. For algorithm development, it was desired to have a full computed tomography scan of the object for reference. Therefore, one of the sources was paired with a Hamamatsu Photonics CsI scintillator screen and imaged by an Apogee Instruments Alta U9 CCD (charge-coupled device) camera. This setup provides a low noise, distortion free image, at the cost of acquisition speed. Each 768 × 512 frame requires a one second exposure time, and 360 frames, one at each of the 360° around the object, are required to provide a complete CT scan. However, the X-ray phantoms used in this study were static, therefore, the acquisition time was not an issue. From this CT scan, a volume was reconstructed to provide a basis for

1) X-ray sources
2) Scintillator detector
3) Intensified detector
4) Fluidized bed
5) Slew ring

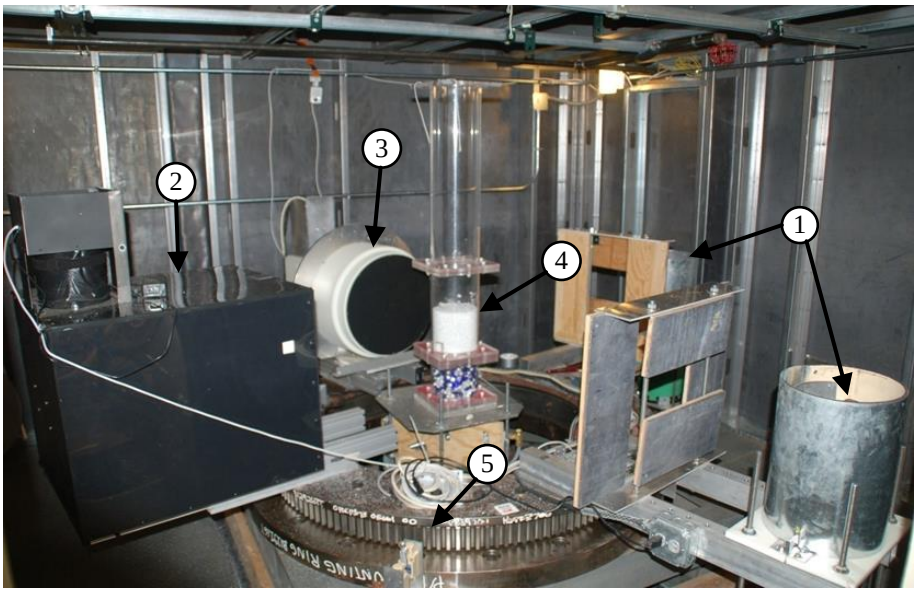

Figure 6.1: An image of the X-ray Flow Visualization facility used in this study. Note that only one source and the scintillator was used to acquire the CT scans in this study. The stereography scans were acquired with both X-ray sources and two intensified detectors.

comparison using an in house implementation of the filtered backprojection algorithm (Zhang, 2003). From the raw CT scan data, radiographs can also be extracted at 90° intervals to represent the data that would be acquired using the X-ray intensifiers.

For the evaluation on a real multiphase flow, the system was used to do X-ray computed tomography, as described above, and used for X-ray stereography. To acquire X-ray stereography, each source is paired with a Precise Optics PS164X X-ray image intensifier to do time-resolved imaging. The X-ray image intensifiers are time-synchronized to provide two radiographs from different angles (90° apart).

The phantom used in this work was made from acrylonitrile butadiene styrene (ABS) plastic and were constructed using additive manufacturing (i.e., 3D printing). The object used for testing in this paper is a solid sphere of diameter 25.4 mm (1 in). However, numerous other test objects are available. The test object has a threaded hole in the bottom center to accept a nylon threaded rod. This allows it to be mounted in various locations on an acrylic test platform.

To evaluate the algorithms with a real multiphase flow, a fluidized bed was used. This fluidized bed consisted of a 15.2 cm (6 in) internal diameter acrylic column filled with ground walnut shell particles in the size range 500 μm to 600 μm (0.020 in to 0.024 in). This bed was injected with air at 1.25 times the minimum fluidization velocity. More information on this fluidized bed system can be found in Drake (2011).

6.4 Reconstruction Methods

Two reconstruction methods were tested to examine their ability to reconstruct tomographic slices from only two X-ray projections. The first method, discussed in Section 6.4.1, is a locally axisymmetric filtered backprojection. This method estimates the unknown projections between the known projections, allowing a standard backprojection algorithm to be used to reconstruct the slice. The second method, presented in Section 6.4.2, is based on the SART reconstruction method. To improve the results as compared to a standard SART algorithm, the reconstruction results are bounded by known information about the flow.

6.4.1 Locally Axisymmetric Filtered Backprojection

The locally axisymmetric filtered backprojection (FBP) method is based on the assumption that individual features in the flow are approximately round about a local axis that is perpendicular to the tomographic slice. However, because these individual features may not occur at the center of the tomographic slice, the axis of rotation needs to be shifted so it intersects the center point of the feature. To do this, the centroid of the feature must be identified in each projection and projected into the volume to identify the feature's center position within the tomographic slice. This procedure is identical to the procedure for X-ray particle tracking and can be done manually or using a computer vision algorithm, such as the

normalized cross-correlation algorithm (Drake et al., 2009; Kingston et al., 2014; Morgan and Heindel, 2010). For this study, all the centroids were found manually.

Once the feature's center position is found, the missing projections can be found by shifting the original projections such that they match where the object would have projected to, had a projection been acquired at that location. The geometry of this shift is shown in Figure 6.2. For the parallel beam case the amount of the shift required in the image is:

$$\Delta\alpha = \alpha_p - \alpha_i \tag{6.1}$$

where $\Delta\alpha$ is the shift in the projection (in pixels), α_p is the distance from the center of the projection to the center of the feature in the known projection, and α_i is the distance from the center of the projection to the center of the feature in the i^{th} unknown projection. Since the position of the feature center in the volume is known, α_p and α_i can be calculated by:

$$\alpha_p = r_0 \sin(\phi - \theta_p) \tag{6.2}$$

$$\alpha_i = r_0 \sin(\phi - \theta_i) \tag{6.3}$$

where r_0 is the radius to the feature center in the slice, ϕ is the angle to the feature center in the slice, θ_p is the angle of the known projection, and θ_i is the angle of the i^{th} projection being generated. Combining Eqs. (6.1) - (6.3), the full equation for the shift becomes:

$$\Delta\alpha = r_0\big(\sin(\phi - \theta_p) - \sin(\phi - \theta_i)\big) \tag{6.4}$$

However, the center position of the feature is known in Cartesian coordinates, and Eq. (6.4) is in polar coordinates. Therefore, converting to Cartesian coordinates, the final equation becomes:

$$\Delta\alpha = \sqrt{x^2 + y^2}\big(\sin(\mathrm{atan2}(y,x) - \theta_p) - \sin(\mathrm{atan2}(y,x) - \theta_i)\big) \tag{6.5}$$

where x is the x-position of the feature center, y is the y-position of the feature center, and atan2(y, x) is the modified arctangent function to prevent divide-by-zero errors and return an angle in the range [0, 360) degrees:

$$\operatorname{atan2}(y,x) = \begin{cases} \arctan\left(\frac{y}{x}\right), & \text{if } x > 0 \text{ and } y \geq 0 \\ \arctan\left(\frac{y}{x}\right) + 360, & \text{if } x > 0 \text{ and } y < 0 \\ \arctan\left(\frac{y}{x}\right) + 180, & \text{if } x < 0 \text{ and } y \geq 0 \\ \arctan\left(\frac{y}{x}\right) + 270, & \text{if } x > 0 \text{ and } y \geq 0 \\ 90, & \text{if } x = 0 \text{ and } y > 0 \\ 270, & \text{if } x = 0 \text{ and } y < 0 \\ \text{undefined}, & \text{if } x = 0 \text{ and } y = 0 \end{cases} \tag{6.6}$$

Once all of the missing projections have been created by shifting the known projections using Eq. (6.5), the reconstruction can be completed normally, using the filtered backprojection algorithm (Eq. (2.5)). Finally, it should be noted that Eq. (6.5) assumes parallel beam scanning geometry. If fan beam scanning geometry is assumed, a scaling factor in the horizontal direction of the projection is required, in addition to the shift. If cone beam scanning geometry is assumed, scaling factors in both the horizontal and vertical direction of the projection are required, as well as a shift in the vertical direction, in addition to the shift in the horizontal direction.

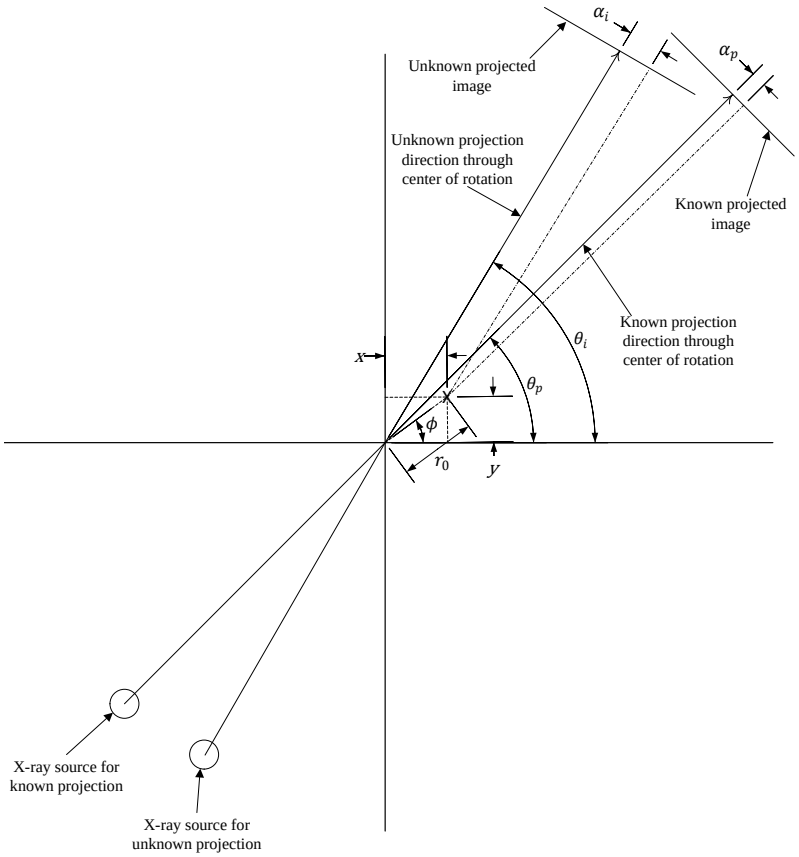

Figure 6.2: The geometry of shifting a projection to create a missing projection from a known projection, assuming a parallel X-ray beam.

6.4.2 Simultaneous Algebraic Reconstruction Technique with A Priori Information

The simultaneous algebraic reconstruction technique (SART) was originally developed by Andersen and Kak (1984) to improve upon the older algebraic reconstruction technique (ART), which was subject to salt and pepper noise (Andersen, 1989). While more computationally intensive that the FBP algorithm, the SART algorithm has the advantage of being able to incorporate a priori knowledge into the reconstruction (Hsieh, 2009).

SART formulates the reconstruction problem as a system of equations:

$$\boldsymbol{p} = \boldsymbol{A} \cdot \boldsymbol{G} + \boldsymbol{e} \tag{6.7}$$

where $\boldsymbol{p}$ is the projection, $\boldsymbol{A}$ is a weighting matrix that defines the contribution of each voxel in the reconstruction to the projection, $\boldsymbol{G}$ is the reconstructed slice (or volume if all the slices

are reconstructed simultaneously), and $\boldsymbol{e}$ is the error of the system. To solve for the reconstructed slice, the SART algorithm calculates the error between the measured projection and the projection as it would be based on the current slice estimation (which is typically initialized to a matrix of all zeros) and then modifies the slice estimation based on the average error over the entire projection. This iterative update formula (Andersen and Kak, 1984) is:

$$\widehat{\boldsymbol{G}}_{ij}^{(q+1)} = \widehat{\boldsymbol{G}}_{ij}^{(q)} + \lambda^{(q)} \frac{\sum_{n=1}^{N} \boldsymbol{A}_{ijmn} \dfrac{\boldsymbol{p}_{mn} - \sum_{i=1}^{I}\sum_{j=1}^{J} \boldsymbol{A}_{ijmn} \widehat{\boldsymbol{G}}_{ij}^{(q)}}{\sum_{i=1}^{I}\sum_{j=1}^{J} \boldsymbol{A}_{ijmn}}}{\sum_{n=1}^{N} \boldsymbol{A}_{ijmn}} \tag{6.8}$$

where $\widehat{\boldsymbol{G}}^{(q)}$ is the estimation of the true slice ($\boldsymbol{G}$) at iteration q, $\lambda^{(q)}$ is the relaxation factor at iteration q, i is the x-position within the slice of width I, j is the y-position within the slice of height J, m is the projection index of the total M projections, and n is the ray index within the slice of width N. This equation is calculated for each projection angle m, and can be iterated to reduce the error. In practice, a reasonable reconstruction can be achieved in a single iteration (Andersen and Kak, 1984).

While in general the weighting matrix $\boldsymbol{A}$ is dependent on the system geometry and the type of interpolation used, in certain cases it can be simplified. For this paper, it will be assumed that a parallel beam geometry is used, with projections at 0 degrees and 90 degrees, and the size of the volume is equal to the width of the projection (e.g., $I = J = N$). In this cased the weighting matrix for the 0 degree projection is:

$$\boldsymbol{A}_{ijn,0} = \begin{cases} \frac{1}{I}, & n = j \\ 0, & n \neq j \end{cases} \tag{6.9}$$

and the weighting matrix for the 90 degree is:

$$\boldsymbol{A}_{ijn,90} = \begin{cases} \frac{1}{J}, & n = i \\ 0, & n \neq i \end{cases} \tag{6.10}$$

Under normal circumstances, Eq. (6.8) would be sufficient to provide an accurate reconstruction of the tomographic slice. However, with only two projections available from stereography, the reconstruction has a tendency to blur the projection across the entire slice, resulting in a single, large, cross-shaped object in the reconstruction (see Section 6.5 for an example). However, in a multiphase flow additional information is available. When determining the time average gas fraction of a multiphase flow from CT data, it is common practice to acquire a static CT with the containment vessel full of the denser phase (typically referred to as the bulk CT), as well as a static CT with the containment vessel empty (typically referred to as the gas CT) (Heindel, 2011). These same CTs can be used to provide a lower and upper limit on the possible intensity values of the reconstruction. This is done by clamping the estimated voxel values in the slice after every iteration of Eq. (6.8). This clamping equation is:

$$\widehat{\boldsymbol{G}}_{ij}^{(q)} = \begin{cases} \boldsymbol{B}_{ij}, & \text{if } \widehat{\boldsymbol{G}}_{ij}^{(q)} < \boldsymbol{B}_{ij} \\ \boldsymbol{C}_{ij}, & \text{if } \widehat{\boldsymbol{G}}_{ij}^{(q)} > \boldsymbol{C}_{ij} \\ \widehat{\boldsymbol{G}}_{ij}^{(q)}, & \text{if } \boldsymbol{B}_{ij} \leq \widehat{\boldsymbol{G}}_{ij}^{(q)} \leq \boldsymbol{C}_{ij} \end{cases} \tag{6.11}$$

where $\boldsymbol{B}$ is the bulk CT slice (denser phase), and $\boldsymbol{C}$ is the gas CT slice (less dense phase).

6.5 Experimental Results

To examine the usefulness of the approximated CT reconstructions, two different tests were conducted. First, the sphere phantom was imaged and reconstructed with a full set of CT projections, using the unmodified reconstruction method with only two projections, and finally using the modified reconstruction algorithms. These results are shown in Section 6.5.1. Second, a real multiphase flow was reconstructed using the modified algorithms and the results were compared to the individual projections. These results are shown in Section 6.5.2. Finally, it should be noted that, while all the results shown in this paper are for a single tomographic slice, there are no limitations preventing these methods from being used on full volumes or with time sequences of projections.

6.5.1 Phantom Imaging

To test the locally axisymmetric CT reconstruction, the test sphere was placed off center in the imaging region, and a CT was acquired. The reconstructed CT using all 360 projections of this phantom is shown in Figure 6.3a as a baseline reference. From the 360 projections, the projections at 0 degrees and 90 degrees were extracted. Figure 6.3b shows the result of using only these two projections to reconstruct the slice using the filtered backprojection algorithm. It is clear to see that, while the general position of the sphere is identifiable, the shape is not reconstructed and there are significant streak artifacts. Finally, the missing 358 projections were generated using the shifting algorithm presented in Section 6.4.1, the results of which are in Figure 6.3c. The general shape of the sphere is reconstructed correctly. In fact, the reconstruction using the shifted projections is nearly identical to the reconstruction using the full CT dataset. Note, however, that the accurate shape reconstruction only occurs because the phantom is spherical in shape. If the phantom

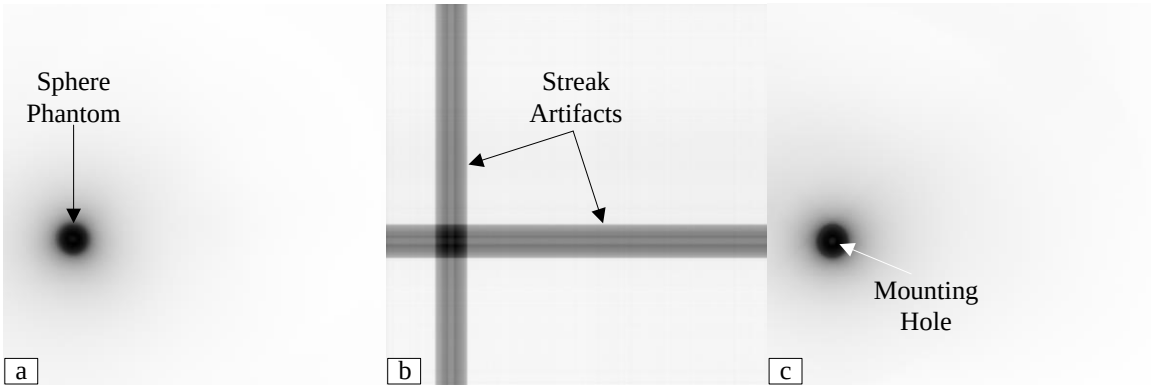

Figure 6.3: The ABS sphere phantom positioned off-center and reconstructed with a) the full 360 projections, b) only the 0 degree and 90 degree projections, and c) 360 projections generated by shifting the 0 degree and 90 degree projections.

were not an axisymmetric shape, the shape would not be accurately reconstructed. There is also some blurring at the outside edge of the phantom. However, as this is present in both the reference slice and the locally axisymmetric FBP reconstruction, this is believed to be due to the use of a parallel beam reconstruction with a scanner geometry that is truly cone beam.

The SART reconstruction with a priori information does not use the position of the object as an input, and therefore it is not important where the object is in the imaging region when testing the reconstruction. Therefore, to minimize the error due to the parallel beam assumption, seen in the locally axisymmetric reconstruction, the test sphere was moved to the center of the platform. There are two parameters in the SART reconstruction that are not examined in this paper, the relaxation factor (λ) and the number of iterations (Q). Neither factor was observed to have a significant impact on the reconstructed slice during testing. Therefore, for all SART tests in this paper, the relaxation factor has been set to $\lambda = 1.0$ and the number of iterations has been set to $Q = 5$. For reference, the slice reconstruction using the filtered backprojection algorithm and all 360 projections is shown in Figure 6.4a. The reconstruction using the SART algorithm and only the projections at 0 degrees and 90 degrees, with no a priori information, is shown in Figure 6.4b. Note how the unmodified

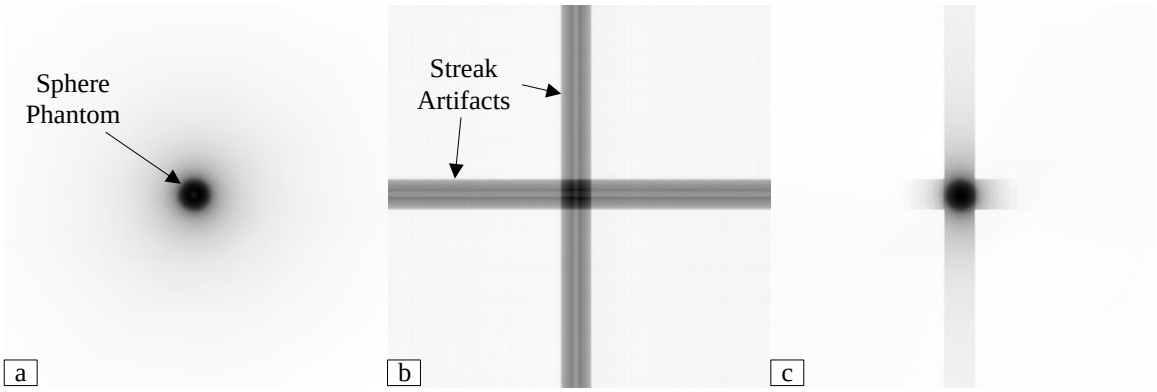

Figure 6.4: The ABS sphere phantom positioned in the center of the imaging region and reconstructed with a) the full 360 projections using the FBP algorithm, b) only the 0 degree and 90 degree projections using the SART algorithm, and c) only the 0 degree and 90 degree projections, with the intensity limited by a priori CT slices of the bulk and gas conditions.

SART reconstruction causes significant streak artifacts, similar to the unmodified FBP algorithm. Finally, the two projection SART reconstruction with a priori information is shown in Figure 6.4c. Unlike the locally axisymmetric FBP algorithm, there are still significant streak artifacts present in this reconstruction. However, they are significantly reduced as compared to the SART slice with no a priori information. Additionally, despite the streak artifacts, the general shape of the sphere phantom is accurately reconstructed. However, the a priori SART slice is missing the central void where the hole to mount the sphere is located. This is an indication that the a priori bulk CT slice (which does not contain the mounting hole) is forcing the reconstruction towards the correct shape.

6.5.2 Multiphase Flow Imaging

To evaluate the reconstructions on a real flow, a fluidized bed of ground walnut shell was used. The raw projections at 0 degrees and 90 degrees are shown in Figure 6.5. Note that, because these images were taken with an X-ray intensifier, the noise level is appreciably higher than the projections used to reconstruct the CTs in the previous section.

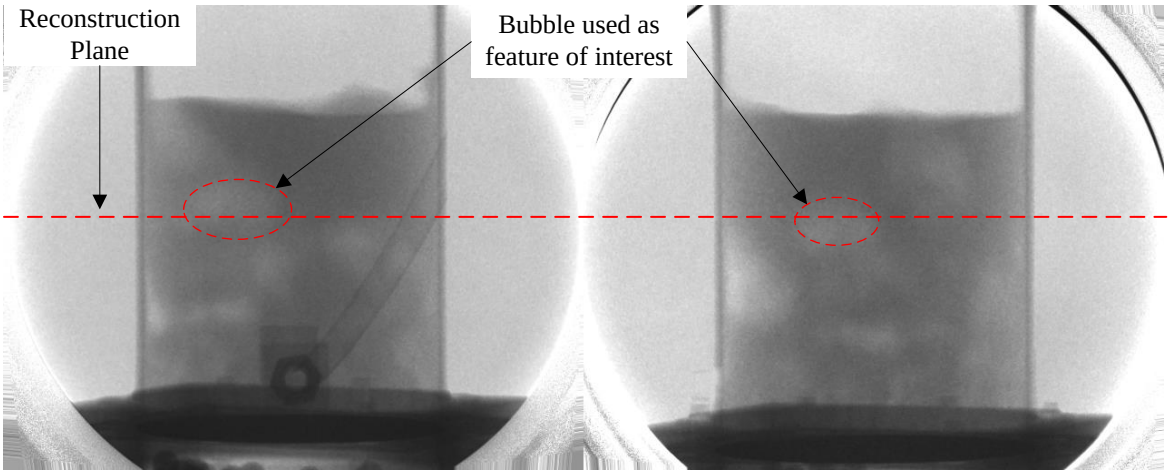

Figure 6.5: The original projections of the fluidized bed used to test the reconstruction algorithms on limited data of a real multiphase flow. The dashed red line indicates the height at which the slices were reconstructed.

The first evaluation with the real multiphase flow data is using the locally axisymmetric FBP algorithm to reconstruct the flow within the bed by assuming the axisymmetric feature is the bed itself. The center of the bed was determined manually and used as the axis of symmetry in the reconstruction. The resulting slice is shown in Figure 6.6a. While most of the slice appears as a circular blur, there are two features that can be seen clearly. The first is the wall of the containment vessel. This is expected, as the vessel was used to define reconstruction axis, and thus must be axisymmetric about the reconstruction axis. However, the second feature, the darker area to the right of the center of the reconstruction (which indicates an area of lower gas holdup), was not expected due to its local nature. However, by examining the projections in Figure 6.5, it can be seen that there are no bubbles to the right of the central axis at that slice location, and it is likely a spot of low gas holdup. Unfortunately, since no full CT can be acquired for comparison in a dynamic system such as the bubbling fluidized bed, it cannot be conclusively confirmed that this is a feature and not an artifact.

The second evaluation is to use the locally axisymmetric FBP algorithm to reconstruct an individual feature within the flow (Figure 6.6b). The bubble on the left of the projections

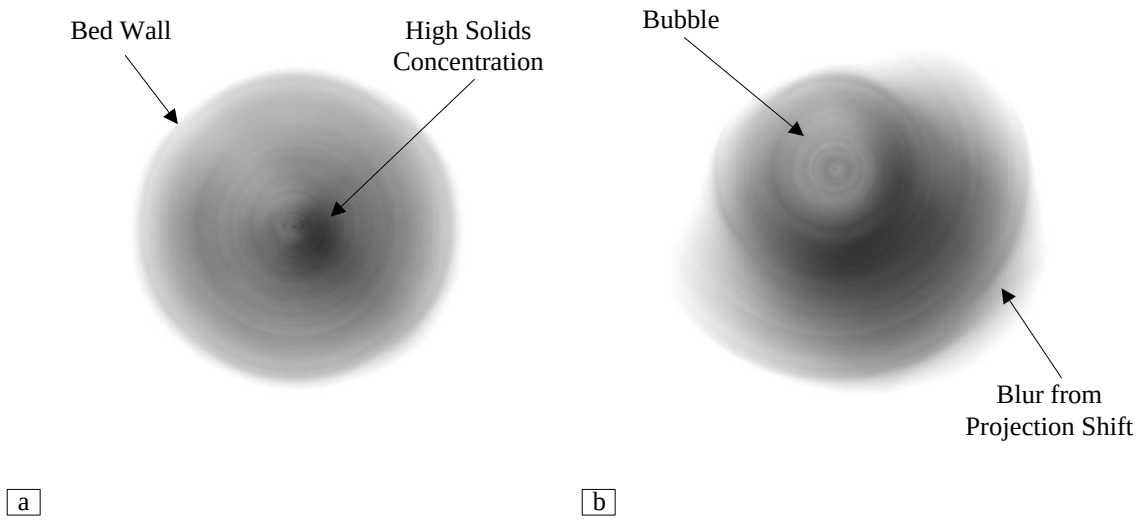

Figure 6.6: The locally axisymmetric FBP reconstructions of the fluidized bed assuming a) the bed is the feature of interest and b) the bubble crossing the slice is the object of interest. Note that the brightness and contrast of these slices have been adjusted to enhance the visibility of the features in the reconstruction.

in Figure 6.5, intersecting the slice level, was selected as the feature of interest. Again, the central axis of the feature was found manually, and the locally axisymmetric FBP algorithm was used to calculate the slice around the feature's axis. The bubble can clearly be seen as the higher intensity region in the center of the slice, with the rest of the flow blurred out around it. While such a reconstruction may be useful for estimating bubble size, the rest of the slice is useless because of the blur introduced by the axis shift.

The final evaluation of real flow data is to reconstruct the same fluidized bed projections using the SART algorithm with the a priori information of the bulk bed CT and the gas bed CT. The result of this reconstruction is shown in Figure 6.7. This reconstruction also shows the dark region to the right of the center of the bed, further indicating that this is a true flow feature. However, the bed region has a "plaid" appearance due to a combination of the noise inherent in the X-ray image intensifiers used to acquire the projections and the SART algorithm's tendency to streak the projections across the slice. It should also be noted that the wall of the containment vessel is clearly visible, but the reconstructed flow does not

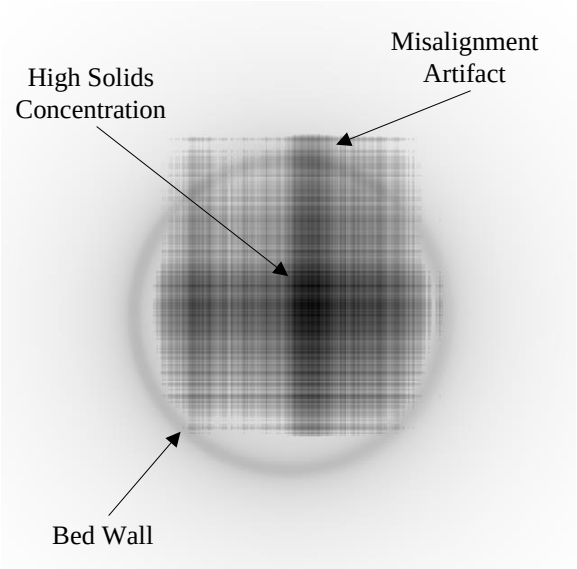

Figure 6.7: The fluidized bed reconstructed with the SART algorithm using a priori information to limit the intensity range.

appear to fit entirely within it. The containment vessel wall is clearly visible because the a priori CTs have nearly identical intensities in both the bulk and gas CT in the wall region, thus it strongly forces the SART slice towards that narrow intensity range, resulting in the clearly defined wall. However, because the a priori CTs and the flow projections were not imaged using the same camera system, there is some misalignment between the two. This misalignment results in the appearance of the flow to exceed the boundaries of the containment vessel.

6.6 Conclusions

It is clear from this research that the algorithms to reconstruct an approximate CT reconstruction from only two stereographic X-ray projections are not yet ready to be used for multiphase flow measurement in the same way X-ray CT or X-ray particle tracking velocimetry are used. However, the algorithms also show areas of promise. In particular, the locally axisymmetric filtered backprojection reconstruction of a fluidized bed was able to almost perfectly reconstruct the sphere phantom from only two projections and identify a

region of low gas holdup that was not globally axisymmetric. While these results alone are not sufficient to prove the effectiveness of stereographic X-ray projections for approximating CT reconstructions, it is reason to continue improving the algorithms and evaluating other approaches to the problem. The first step in this future work will be to extend all the algorithms to handle cone beam geometry so the true geometry of the system is replicated in the reconstructions. From there, methods to reduce the noise in the projections acquired with the X-ray image intensifier need to be evaluated to improve the continuity of the reconstructions. Finally, the FBP and SART algorithms are only two of a plethora of reconstruction algorithms available. While most algorithms are intended for far more projections than X-ray stereography provides, it is worth evaluating their comparative merit for such a case.

CHAPTER 7:

DEVELOPMENT OF A NONCONTACT USER INTERACTION SYSTEM FOR SURROUND-SCREEN VIRTUAL ENVIRONMENTS

Shifting focus from X-ray imaging, this chapter presents a system designed to integrate natural, noncontact user interaction with CAVE-style virtual environments (objective four from Section 1.2). While this may initially seem unrelated to the characterization of multiphase flows, one of the major challenges in working with large X-ray computer tomography datasets is how to visualize the data efficiently and effectively. A key component of that, particularly in immersive environments, is how to interact with the data. This chapter is based on a draft being prepared for submission to *Presence: Teleoperators and Virtual Environments*.[4]

7.1 Abstract

Since the introduction of the Microsoft Kinect sensor in 2010, there has been a significant amount of research into its use in a wide variety of fields, including virtual reality. However, the use of Kinect sensors in CAVE-style virtual environments has been slowed by the large tracked area that requires multiple Kinect sensors, and by the complication in implementing many of the desired user interactions, such as voice recognition and gesture recognition, in pre-existing CAVE applications. This paper describes the challenges of using the Kinect sensor in a CAVE-style virtual environment, and discusses the implementation of a software system designed to simplify the implementation of Kinect interaction with CAVE-

[4] Based on draft Morgan, T. B., Heindel, T. J., and Vance, J. M. (2017). Development of a Noncontact User Interaction System for Surround-Screen Virtual Environments. To be submitted to *Presence: Teleoperators and Virtual Environments*.

style environments by abstracting the Kinect data as a Virtual Reality Peripheral Network server. In particular, this paper discusses the details of implementing skeleton merging, skeleton filtering, and gesture recognition. In addition, a method of generating joint orientations from joint positions that is logically consistent with the Microsoft method is presented. Finally, the techniques implemented in the system are validated with both simulated data and publically available human motion datasets.

7.2 Introduction

The use of low cost body based tracking has increased dramatically in recent years thanks in part to the introduction of the Microsoft Kinect sensor (Zhang, 2012). This commoditization of body tracking has enabled many more developers to implement non-contact user interactions in their code; however, the details of implementation still remain one of the biggest barriers to its usage (Takala et al., 2012). Furthermore, the Kinect was designed specifically for video game interactions and thus assumes that the user will be nearly directly facing the sensor and interacting with visuals on a single screen. In many areas in which the Kinect sensor is being adopted for user interaction, these assumptions are not valid. In particular, several challenges arise when using a Kinect sensor in a CAVE-style virtual environment:

1) The large tracked area makes it difficult for one Kinect to cover the entire CAVE. Optical occlusion from the CAVE walls exacerbates this problem and forces the Kinect sensor(s) to be placed in sub-optimal positions. Typical placements can include above the walls looking down, in the corners of the CAVE, or behind the user.

2) The Kinect sensor has inherently high noise and high latency (Livingston et al., 2012).
3) The high computational cost of using Kinect sensors takes valuable computing time from the rendering of visuals.
4) Even for programmers with experience in virtual reality, implementing the algorithms required for using the Kinect for 3D user interaction can be difficult (Takala et al., 2012).

This paper presents the Kinect with Virtual Reality (KVR) system, which is designed to address the challenges of using the Microsoft Kinect sensor as an input device for virtual reality, with a specific focus on CAVE-style virtual environments. The KVR system is available as free, open-source software from https://github.com/vancegroup/KVR.

7.3 Background

The original Microsoft Kinect sensor for the Microsoft Xbox 360 game console was released on November 4, 2010 and quickly became the fastest selling consumer electronics device to date (Zhang, 2012). While much of this success was due to consumers using the device as intended, a large number of researchers began to use the Kinect as a human-computer interface across a wide variety of fields, including virtual reality, robotics, medical image visualization, and rehabilitation (Gallo et al., 2011; Lun and Zhao, 2015; Morato et al., 2014; Williamson et al., 2012). Subsequent to the release of the original Kinect sensor, a slightly updated version of the sensor was released with official support for the Microsoft windows platform and improved control over the camera parameters. However, from a hardware standpoint, these two sensors are virtual identical and will both be referred to as the Kinect v1. The Kinect v1 sensor uses infrared structured light to determine the depth of

objects in its viewing area, with a resolution of 640 × 480 at 30 frames per second (FPS). The Kinect v1 sensor contains a color camera with a resolution of 640 × 480 at 30 FPS, with support for other resolutions and frame rates. Finally, the Kinect v1 sensor contains a motor to adjust the sensor tilt, a three-axis accelerometer, and a microphone array (Microsoft, 2014b; Zhang, 2012).

A second version of the Kinect sensor was released alongside the Xbox One game console and official support for Microsoft Windows was released in 2014. This updated version of the sensor is referred to herein as the Kinect v2 and improves upon the original in nearly every way. The Kinect v2 senses depth with a resolution of 512 × 424 at 30 FPS using a time-of-flight sensor (Lun and Zhao, 2015). The Kinect v2 also contains a 1920 × 1080, 30 FPS color camera, along with a microphone array (Microsoft, n.d.-f). The only features from the Kinect v1 that are absent on the Kinect v2 are the accelerometer and tilt motor.

The first problem with using the Kinect sensor in virtual reality is the tracking area is often larger than what a single Kinect can cover. While this problem could be mitigated by placing the Kinects roughly at eye level in front of the users, this would typically place them behind the projection screens of the CAVE, rendering them useless. One type of CAVE, the blue-c, uses liquid crystal projection screens that allow the screens to be selectively transparent so cameras placed behind the screen can image the users (Gross et al., 2003). However, most CAVE systems are not equipped with this feature, and to the best of the author's knowledge, such a system has never been tried with a Kinect sensor. Thus, CAVE users are left with the challenge of merging data from multiple Kinects into a single usable data stream. Multiple researchers have observed this problem and addressed it with different

solutions. One solution to this problem is sensor scheduling (Faion et al., 2012). This strategy will track the object of interest (typically, but not always, the system user) and only uses the data from the Kinect sensor that has the "best" view of the object of interest. This method has the benefit of being able to use external shutters on the Kinect sensors to reduce interference between multiple Kinects (Berger et al., 2011). However, all the information from the sensors without the "best" view is lost. A second solution to this problem is to use data fusion to combine all the Kinect skeletons into a single stream. This was done by Williamson et al. (2012) using a weighted averaging method to track dismounted soldiers in training simulations. Multiple research groups have also used Kalman filtering to solve the data fusion problem with multiple Kinects (Li et al., 2014; Masse et al., 2013; Moon et al., 2016). However, none of these papers address handling of cases where multiple users may be tracked by the Kinect. Furthermore, Williamson et al. (2012) note that when a user's back is facing the Kinect sensor, the Kinect often assumes that the user is facing the sensor, leading to a left-right reversal. While Williamson et al. propose a method to handle this based on assumed poses, a more robust method is needed.

The second problem with using the Kinect sensor is its high noise and latency, which is not unique to the Kinect, but nevertheless important to efficient and effective user interaction (Casiez et al., 2012). Previous research has shown that noise in the Kinect v1 skeleton data is depth dependent and has an average noise of 6.9 mm when the user is 3.5 m from the sensor. The same research also showed an average latency in the skeleton data of 106 ms (Livingston et al., 2012). Due to this noise, Microsoft has recommended the use of filtering on skeleton data; however, the filtering has the potential to add additional latency to the system (Azimi, 2012). Therefore, a filtering method that is capable of both noise reduction and prediction to

reduce the effects of latency is preferable. The methods of filtering recommended by Microsoft are all variants of the auto regressive moving average (ARMA) filter (Azimi, 2012). Another popular filter in virtual reality applications is the 1€ filter (Casiez et al., 2012). The 1€ filter is a first-order low pass filter with a cutoff frequency that is adjusted based on the speed of the input signal. This leads to a filter that does more jitter reduction when the user is relatively still, but is more responsive when the user is moving quickly. This filter has been shown to perform well when compared to other filters (Casiez et al., 2012); however, it lacks a method to predict ahead to account for inherent system latency. A final class of filters that has been used extensively in virtual reality is the Kalman filter (Welch, 2009). As seen previously, the Kalman filter has also seen extensive use to fuse data from multiple Kinects. It is also effective at filtering the data streams to reduce noise and has a built-in method for predicting ahead to compensate for latency (Hargrave, 1989). However, the selection of an appropriate state model can be a significant challenge and cause the filter to underperform (Brown and Hwang, 1997; Casiez et al., 2012; Welch, 2009).

The third problem with using Kinect sensors in a CAVE-style virtual environment is the relatively high computational cost of the calculations. While the Kinect algorithm for deriving skeletons from the depth images was designed for speed (it runs in about 5 ms on the Xbox 360 hardware), the cost of running multiple Kinects and filtering adds up quickly for a real-time application such as virtual reality (Shotton et al., 2011). Multiple researchers have separated the rendering from the Kinect calculations by using one or more computers to do the Kinect calculations, and then transmit the results to the rendering machine(s) over a standard network (Moon et al., 2016; Williamson et al., 2012). This is a common strategy in virtual reality, and one for which the Virtual Reality Peripheral Network (VRPN) was

specifically designed to handle (Taylor et al., 2001). VRPN is a device-independent software that permits input devices to transmit their data to a client computer which handles the image rendering. This frees the rendering computer from the overhead of the input device calculations and frees the rendering from needing a device specific code for the input device. While many input devices support VRPN, there is currently one software package, the Flexible Action and Articulated Skeleton Toolkit (FAAST) that supports transmitting Kinect sensor data over VRPN.

The final problem with using Kinect sensors in CAVE-style virtual environments is the difficulty in implementing useful user interactions on top of the raw Kinect data (Takala et al., 2012). FAAST is one attempt to solve this issue. It abstracts the details of processing the Kinect data and allows for simplified implementation of rule-based gestures. However, it lacks several desirable features for use with CAVE-style virtual environments, notably support for simultaneous use of multiple Kinect sensors, skeleton filtering and prediction, and voice recognition. Support for the Kinect sensor has also been integrated into multiple virtual reality toolkits, including MiddleVR and the Reality-Based User Interface System (RUIS) (Kuntz, 2015; Takala, 2014). While this support abstracts some of the details, it leaves more complicated tasks, such as merging and gesture recognition, to the application programmer. Furthermore, if an application was not built on the platform, designed for a Kinect sensor from the beginning, it can take significant changes to add it in later.

7.4 Implementation

With the challenges of using a Kinect sensor as an input device for CAVE-style virtual environments in mind, and considering the limitations of previous systems, the basic design parameters of the KVR system are that the system should:

- support all of the Kinect's sensing modalities,
- support both the Kinect v1 and Kinect v2,
- allow for the merging of skeletons from several sensors,
- abstract the Kinect data sufficiently that Kinect interaction can be added to pre-existing virtual reality applications with minimal programming effort.

7.4.1 System Architecture

The best way to abstract the Kinect data to allow it to be used in pre-existing virtual reality applications is VRPN. VRPN has wide support across many VR toolkits and it is device and operating system independent. This allows a VR application that was written for one input device using VRPN to be replaced by a Kinect feature of the same input class by simply redirecting the application to a different VRPN server. For example, a system that obtains hand position over VRPN from a marker-based optical tracking system can receive hand position from a Kinect by simply having the application connect to the KVR VRPN server instead of the marker-based tracker's VRPN server. Additionally, VRPN is available as an open source project, including bindings for the Microsoft .NET Framework through the VRPN.Net project (Taylor et al., 2001; VanderKnyff, 2008).

Once it was decided to abstract the Kinect information using VRPN server, it was decided to build the KVR system on top of the Microsoft .NET Framework version 4.5 and official Microsoft Kinect for Windows SDK (software development kit). The official SDK was selected over an open source SDK, such as OpenNI, primarily due to its better support for the microphone array on the Kinect sensor and its better support for the Kinect v2. This choice does introduce a limitation that the KVR server must run on a Windows operating

system. However, this restriction is mitigated by the use of VRPN to communicate over a standard computer network between the client and the server.

In order to handle both the Kinect v1 and Kinect v2 within the same software, it was necessary to split the handling of each type of Kinect into its own assembly. Additionally, the Kinect v2 SDK only supports a single Kinect v2 per computer, and it is likely that in a CAVE environment, multiple Kinect v2 sensors will be needed for full tracking. Therefore, a third Kinect type was introduced, the networked Kinect. This is simply a wrapper around a VRPN client that allows the VRPN output of one KVR skeleton on one computer, to be input into KVR on another computer as if it were a Kinect sensor. Additionally, since it is implemented as a VRPN tracker client, any tracking device that outputs VRPN can be input as if it were a Kinect. For example, if an application requires high-precision tracking of a user's hands, but doesn't require as much precision for the rest of the body, the user's hands can be tracked with a marker-based optical tracking system and the rest of the body with Kinect sensors. KVR can then integrate all the measurements as a single skeleton stream for the application to use.

The splitting of the handling of each type of Kinect into its own assembly resulted in the KVR system being built as five separate assemblies, represented by the dark blue boxes in Figure 7.1. The Kinect with VR server assembly combines all the merging of the Kinect data, handles transmitting the data over VRPN, and provides a graphic user interface for the user to control all the settings. The assemblies to handle each type of Kinect feed data to the server assembly; however, each of these assemblies is loaded at run time, so that if an assembly is missing (for example, if the system is being run on Windows 7, which does not support the Kinect v2), the user can still operate the server, albeit with reduced functionality.

The final component of the KVR system is the Kinect Base assembly. This provides a common set of classes and interfaces that allow all three types of Kinect to communicate to the server using the same data types.

The rest of the assemblies presented in Figure 7.1 are libraries that the KVR system is dependent on, with unmodified assemblies in light blue, and customized assemblies in medium blue. VRPN.Net was modified to add support for the VRPN Imager device, and thereby allow the server to transmit the raw Kinect color and depth streams, if the user so desires. Eigen.Net is a new library that was created specifically for KVR, but could have wide applicability beyond KVR. It is a .Net wrapper around the Eigen linear algebra library (TuxFamily, 2017). This library was required because the Kalman filtering used to merge and filter the skeleton data uses a large number of linear algebra calculations, and .NET v4.5 does not natively support the single instruction, multiple data (SIMD) processor instructions required to make these calculations run fast enough to maintain the Kinect's frame rate. These customized libraries are both available at https://github.com/vancegroup.

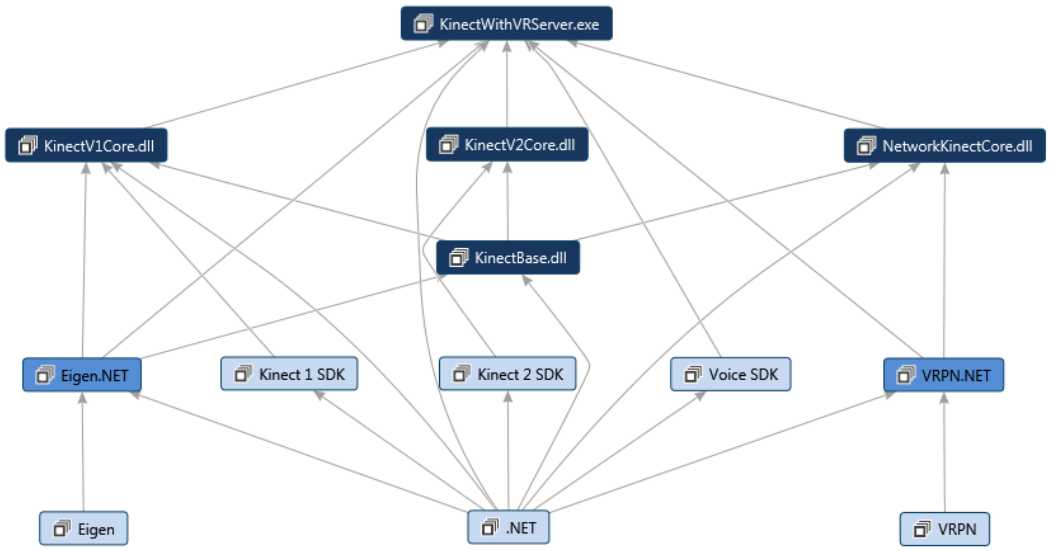

Figure 7.1: The architecture of the KVR system. Each box represents a single assembly. The dark blue boxes, collectively, make up the KVR system, while the medium and light blue are libraries KVR is dependent on. The medium blue are open source libraries that had to be written or modified, the light blue libraries were used unmodified.

7.4.2 Features

The KVR system has numerous features to assist users in interfacing Kinect sensors with virtual reality. While many of these features are simply wrappers around pre-existing Kinect functionality or fairly straightforward to implement, a few desire special attention. Those features are skeleton merging and filtering (covered in Section 7.4.3), the calculation of joint orientations (covered in Section 7.4.4), and gesture recognition (covered in Section 7.4.5).

Of the remaining features, the most significant one for advancing user interaction in virtual reality is voice recognition. The Kinect sensor (both v1 and v2) contains a microphone array that allow the Kinect to optimize its audio stream for a specific location (specified by an angle from the center of the Kinect's view). This audio stream can then be sent to the Microsoft Speech Recognition SDK to do voice recognition. In order to interface voice recognition with VRPN, the recognition events are turned into either VRPN button presses or VRPN text messages by KVR.

In most CAVE applications, a single user will be in control of the interaction with the system. In these applications, it is advantageous to have the Kinect optimize its audio stream for the position where that user is located. This can be done in KVR by instructing the system which user's skeleton position to monitor for voice recognition (this does not guarantee that another user won't be heard, only that the monitored user has the best chance of being heard). However, this can also be done by using a feedback sensor. KVR supports one VRPN tracker sensor for feedback, which allows an external tracker to be used to optimize functions in KVR relative to that position. In most use cases, this feedback sensor would be the head tracked position of the user in the CAVE. Because most CAVEs only support a single head tracked user, this user will likely be the most important user to monitor.

By using the feedback sensor in KVR, the voice recognition position can be set so that the Kinect is always optimizing its voice recognition for the head tracked user. Additionally, this feedback sensor can also be used to sort the skeletons, so that the ordering of the merged skeletons will be based off the distance from the head tracked user.

Additionally, the KVR system supports some features that are available for each Kinect sensor individually. First, all the available Kinect settings on both the Kinect v1 and Kinect v2 are exposed so the user can adjust them as necessary for an application. Second, the raw skeleton streams from both the Kinect v1 and the Kinect v2 can, optionally, be transmitted so that KVR's merging and filtering algorithms can be bypassed by users, if desired. This functionality is not available for the networked Kinects, since their raw skeleton streams are already available over VRPN from another server. Third, the acceleration measured by the Kinect v1 sensor's accelerometer and the measured angle to a sound source, as measured by the Kinect v1 or Kinect v2 sensor's microphone array, can be made available over VRPN as a VRPN analog device. Finally, KVR supports the ability to transmit the raw images from the Kinect v1 and Kinect v2 color and depth streams over a VRPN Imager server. This allows applications to access the video from the Kinect for further processing, even if the application is running on a system that does not support the Kinect sensor. However, it is cautioned that the transmission of color and depth streams should be used judiciously. The VRPN Imager device does not support any image compression, and the uncompressed image data can quickly saturate the available network bandwidth.

7.4.3 Skeleton Merging and Filtering

The merging and filtering of skeleton data in KVR are both accomplished using a Kalman filter. To merge and filter the skeletons, each time a Kinect sensor processes a frame with one or more skeletons in it, the Kinect forwards those skeletons to the skeleton merger. This merger converts all the skeletons to a common coordinate system, determines which skeletons belong to the same user and should be merged together, and then integrates all the measurements into a set of Kalman filters representing each user's skeleton. While the Kalman filtering is the last step in this process, it will be considered first here, as it aids in the understanding of the other steps.

A Kalman filter is a set of linear equations that allow the recursive estimation of the true value of a measured quantity. Kalman filters are advantageous for use in virtual reality because they provide a method to reduce the noise from sensors, the ability to integrate measurements from multiple sensors, and a method to estimate what the state of the sensor will be at a future point in time (Welch, 2009). This ability to do predictive tracking is particularly useful in sensors with high latency, such as the Kinect.

To use a Kalman filter, an appropriate state model must first be determined. This model is given by:

$$\boldsymbol{x}_k = \boldsymbol{F}\boldsymbol{x}_{k-1} + \boldsymbol{w}_k \tag{7.1}$$

where $\boldsymbol{x}_k$ is the state estimation at time k, $\boldsymbol{x}_{k-1}$ is the state estimation at the previous time step ($k-1$), $\boldsymbol{F}$ is the state transition model, and $\boldsymbol{w}_k$ is the process noise (Brown and Hwang, 1997). The state transition model mathematically defines how the state changes from one time step to the next, and determining the correct model is critical to achieving good filter performance (Welch, 2009). Since human motion must follow the laws of physics, a

position-velocity-acceleration model was selected to be applied on a per joint basis (Brown and Hwang, 1997). This model is:

$$\boldsymbol{F} = \begin{bmatrix} 1 & \Delta t & \frac{\Delta t}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & \Delta t & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & \Delta t & \frac{\Delta t}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & \Delta t & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & \Delta t & \frac{\Delta t}{2} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{7.2}$$

where Δt is the time, in seconds, between steps k and $k-1$. The state (position) of the joint in this model is given by the column vector:

$$\boldsymbol{x} = \begin{bmatrix} x \\ \dot{x} \\ \ddot{x} \\ y \\ \dot{y} \\ \ddot{y} \\ z \\ \dot{z} \\ \ddot{z} \end{bmatrix} \tag{7.3}$$

where x, y, and z are the three-space position of the joint, $\dot{x}$, $\dot{y}$, and $\dot{z}$ are the three-space velocity of the joint, and $\ddot{x}$, $\ddot{y}$, and $\ddot{z}$ are the three-space acceleration of the joint. While this Kalman filter integrates all three components of the joint position in a single filter, the filter could be rewritten so that each component is filtered separately and yields identical results.

While the position-velocity-acceleration model accurately reflects the physics of joint movement, there is no way to know the input force provided by the user's muscles. Therefore, a relatively high process noise must be used in order for the system to respond without introducing too much lag. The effects of this will be examined in Section 7.5.1. Finally, it should be noted that this is only one possible state model for the Kalman filter, and

it is not claimed to be the optimal one. However, because all the Kalman filtering is implemented in KVR using matrix math (via the Eigen matrix library), users with knowledge of Kalman filtering can easily implement other state models as desired.

To accomplish both merging and filtering in a single Kalman filter, the skeleton data from each Kinect must first be transformed into a common coordinate space, which will be referred to as the global coordinate system. Since CAVE systems already have a coordinate space used for head tracking users and the rendering of visuals, it is typically preferable to have the coordinate system of the Kinects match that of the CAVE. However, this coordinate system will vary from system to system. Therefore, the determination of coordinate systems for the Kinect is done by having the user set the position and orientation of each Kinect inside the user's desired coordinate system. From there, the required transformation matrix to convert from Kinect coordinates to global coordinates is simple.

However, precisely measuring the position and orientation of each Kinect is challenging. One solution to this would be some form of an calibration to a known reference (Berger et al., 2011; Li et al., 2014). However, this can introduce its own challenges, particularly for networked Kinect sensors where the color and depth sensor information may not be available. Due to this, and given that it is unlikely that the Kinects will be moved frequently after the initial setup, it was decided to use a manually set position and orientation as the only way to position the Kinect sensor in the global space. While the position and orientation of each Kinect has to be fully manually defined for the Kinect v2 and networked Kinects, the accelerometer on the Kinect v1 can be used to help simplify the orientation determination (Pavlik and Vance, 2010). This is accomplished by assuming that the acceleration due to gravity should be aligned with the negative y-axis of the Kinect's coordinate system. An

angle-axis rotation between the negative y-axis vector and the direction of gravity as measured by the accelerometer can then be calculated. Once this is done, the user only needs to manually define the position and yaw of the sensor. However, the measurement noise of the accelerometer introduces a significant amount of noise into the global coordinate system position of the skeleton. To resolve this, the acceleration from the Kinect is filtered with a simple Kalman filter that estimates the acceleration to be constant, but uses a time varying process noise covariance to allow new measurements to be integrated slowly, should the orientation change.

Once all the skeletons are in the global coordinate system, the next step is to determine which skeletons should be merged together. In a CAVE system, it is likely that there will be multiple users in the interaction space simultaneously, and ideally, the merging system should return a single skeleton for each user. To accomplish this, the KVR system keeps a collection of Kalman filters, which model the state of each joint in each skeleton, that the system has recently tracked (a skeleton is removed if none of its joints have been updated within the past five seconds). Each time a frame is received from a Kinect, the skeleton merging system will get an updated position estimate from each of the skeletons it has been tracking. Every predicted skeleton is then compared against the received skeleton to determine the average distance between each joint in the predicted and the received skeleton. If the smallest average distance found is less than 0.3 m (11.8 in), the skeletons are assumed to be the same. If no skeleton is found that has an average distance of less than 0.3 m (11.8 in), KVR will assume it is a previously unseen user and create a new set of Kalman filters for it. The 0.3 m (11.8 in) threshold was determined by trial and error to balance the

possibility of two unique users' skeletons from being, incorrectly, merged together, against the possibility of two views of the same skeleton being treated as independent users.

However, as Williamson et al. (2012) note, the Kinect sensor has trouble determining reliably if a user is facing the Kinect or facing away from the Kinect. In order to handle this issue, the received skeleton is compared both normally and with the left and right joints reversed (e.g., the left hand is assumed to really be the right hand). If a shorter average distance is found using the reversed joints, it will assume the users back is facing the Kinect and integrate the joint measurements into the Kalman filters accordingly. The only limitation of this method is that it assumes the user is facing the Kinect the first time the system detects the user.

Finally, there are two details to be handled in the Kalman filter that were not previously discussed. First, the system needs a method to determine the tracking state of the joints after filtering. There are several possibilities, such as using the tracking state of the last incorporated joint or statistical models like adding an additional Kalman filter for monitoring the tracking state or using Hidden Markov Models. However, because the Kalman filter of the joint already includes a statistical estimation of the quality of the joint, in the form of the estimate covariance, it was decided to use this error estimate instead. This is done by calculating the natural log of the matrix Frobenius norm:

$$a = \ln\|\boldsymbol{P}\| = \ln\sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|p_{ij}|} \tag{7.4}$$

where a is the total error estimate, $\boldsymbol{P}$ is the estimate covariance matrix, m is the number of rows of the matrix, and n is the number of columns of the matrix. If $a < 2.0$, and the joint has been updated in the last second, the joint is considered tracked. If $a \geq 2.0$ and $a < 4.0$,

the joint is considered inferred. If $a \geq 4.0$ the joint is considered not tracked. Note, these thresholds were determined experimentally, and can be changed in the code if the user desires looser or tighter bounding.

The second issue that must be handled is filtering the orientation of the joints. Both the Kinect and VRPN represent the orientations as quaternions, which could be filtered. However, because rotation quaternions are non-linear, an extended Kalman filter would be required, which is more complicated and computationally expensive (Marins et al., 2001; Yun et al., 2003). Furthermore, these results may be inconsistent with what is provided by the Kinect SDK. However, there is a different approach to obtaining joint orientations that is made possible by first realizing that the Kinect does not measure orientations, but instead calculates the orientations based on the measured joint positions (Microsoft, 2012a). Unfortunately, this algorithm has not been made publically available; therefore, an algorithm that is logically consistent with the Kinect SDK orientations will be presented in Section 7.4.4. The KVR system uses this algorithm to calculate new joint orientations from the filtered joints.

7.4.4 Joint Orientation Algorithm

Defining a joint orientation algorithm that is logically consistent with the joint orientations from the Kinect is complicated not only by the lack of documentation from Microsoft, but also by the fact that the Kinect v1 and Kinect v2 do not calculate the joint orientations in exactly the same way. What is known from available documentation is that the calculations are done in a hierarchical fashion, with the hip center joint being the root joint, and that the y-axis of each joint's coordinate system should be along the line from the previous joint to the current joint (Microsoft, 2012a). Based on testing, it appears that the

Kinect v2 does not provide any orientation information about the terminal joints (e.g., the head), but instead uses an identity matrix for those orientations. The Kinect v1 does provide an orientation for the terminal joints. Therefore, to provide as much information as possible to the user, it was decided to maintain consistency with Kinect v1 orientations instead of the Kinect v2. This has the additional advantage that the Kinect v1 SDK allows joint positions to be moved in code, and will recalculate the orientations, thus allowing for comparisons based on artificially generated skeletons. Based on this, algorithms were tested by trial and error until an algorithm that was logically consistent with the Kinect v1 SDK's method of calculating joint orientations was found. It should be noted, that this method is not claimed to be identical to the Kinect v1 SDK's algorithm, only that it is logically consistent. However, as will be shown in Section 7.5.2, it is very close.

Before a method can be created, a joint hierarchy must be defined. This hierarchy must include all the joints from the Kinect v1 and the Kinect v2, which don't use identical joints. First, some joint remapping is required, as the Kinect v2 changed the names of some of the joints, but maintained essentially the same anatomical positions (Microsoft, 2012a, 2014a). These joint mappings are: the Kinect v2 spine shoulder is mapped to the Kinect v1 shoulder center, the Kinect v2 spine mid is mapped to the Kinect v1 spine, and the Kinect v2 spine base is mapped to the Kinect v1 hip center. The Kinect v2 neck, hand tip, and thumb joints have no corresponding joint in the Kinect v1 and will be treated as unique joints. A complete list of the joint mappings, as well as what sensor number they map to in VRPN can be found in Table 7.1. With this joint mapping defined, a hierarchy can be defined. This hierarchy is shown in Figure 7.2.

Table 7.1: Mapping of joints from the Kinect v1 and Kinect v2 to the KVR system and the corresponding VRPN sensor number. Note that the VRPN sensor numbers were selected to maintain compatibility with the FAAST system (Suma et al., 2013), resulting in sensor numbers four and ten not being used.

Kinect v1 Joint	Kinect v2 Joint	KVR Joint	Abbreviation	VRPN Sensor
Head	Head	Head	Hd	0
Shoulder Center	Spine Shoulder	Shoulder Center	SC	1
Spine	Spine Mid	Spine	Sp	2
Hip Center	Spine Base	Hip Center	HC	3
Shoulder Left	Shoulder Left	Shoulder Left	SL	5
Elbow Left	Elbow Left	Elbow Left	EL	6
Wrist Left	Wrist Left	Wrist Left	WL	7
Hand Left	Hand Left	Hand Left	HnL	8
	Hand Tip Left	Hand Tip Left	HTL	9
Shoulder Right	Shoulder Right	Shoulder Right	SR	11
Elbow Right	Elbow Right	Elbow Right	ER	12
Wrist Right	Wrist Right	Wrist Right	WR	13
Hand Right	Hand Right	Hand Right	HnR	14
	Hand Tip Right	Hand Tip Right	HTR	15
Hip Left	Hip Left	Hip Left	HL	16
Knee Left	Knee Left	Knee Left	KL	17
Ankle Left	Ankle Left	Ankle Left	AL	18
Foot Left	Foot Left	Foot Left	FL	19
Hip Right	Hip Right	Hip Right	HR	20
Knee Right	Knee Right	Knee Right	KR	21
Ankle Right	Ankle Right	Ankle Right	AR	22
Foot Right	Foot Right	Foot Right	FR	23
	Neck	Neck	Nk	24
	Thumb Left	Thumb Left	TL	25
	Thumb Right	Thumb Right	TR	26

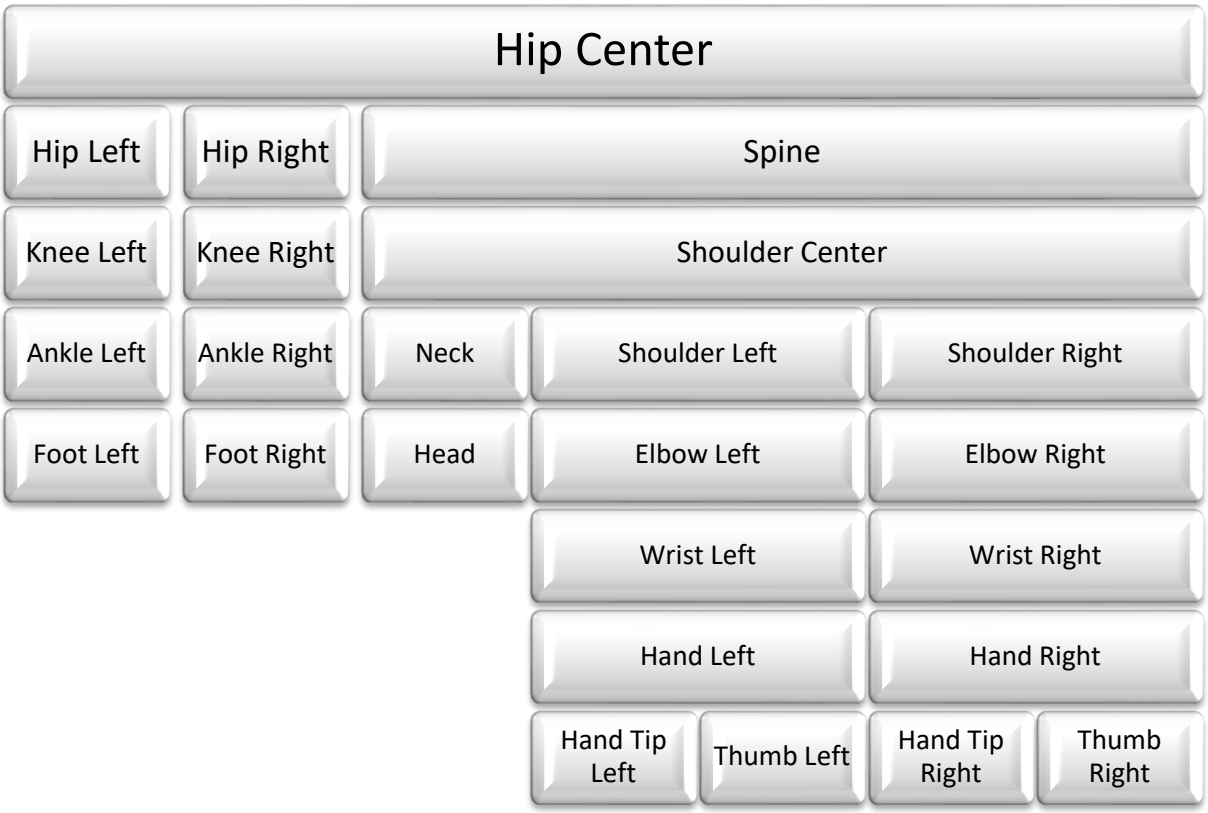

Figure 7.2: The joint hierarchy used for calculating joint orientations in the KVR system. Adapted from the Kinect v1 joint orientation hierarchy (Microsoft, 2012a).

For all the joint orientations, they can first be constructed in a method that is very similar to a look-at matrix from computer graphics, as defined in Section 2.3.1.1. First, since it is known that the y-axis of the orientation will be the vector from the previous joint in the hierarchy to the current joint, it can be generated by the equation:

$$\boldsymbol{y} = \frac{\boldsymbol{p}_{current} - \boldsymbol{p}_{previous}}{\left\|\boldsymbol{p}_{current} - \boldsymbol{p}_{previous}\right\|} \tag{7.5}$$

where $\boldsymbol{p}_{current}$ is the three-space position of the current joint in the hierarchy, $\boldsymbol{p}_{previous}$ is the three-space position of the previous joint in the hierarchy, $\boldsymbol{y}$ is the y-axis vector of the joint orientation space, and $\|\boldsymbol{v}\|$ denotes the magnitude of the vector $\boldsymbol{v}$. Next, either the z-axis vector or the x-axis vector of the orientation space can be defined; however, in practice the z-axis vector is typically defined first by:

$$\boldsymbol{z} = \frac{\boldsymbol{x}' \times \boldsymbol{y}}{\|\boldsymbol{x}' \times \boldsymbol{y}\|} \tag{7.6}$$

where $\boldsymbol{z}$ is the z-axis vector of the orientation space, $\boldsymbol{x}'$ is the x-axis vector in the Kinect space, and $\times$ denotes the vector cross product. The remaining orientation space axis (in this case the x-axis vector) is defined by:

$$\boldsymbol{x} = \boldsymbol{y} \times \boldsymbol{z} \tag{7.7}$$

where $\boldsymbol{x}$ is the x-axis vector in the orientation space. Finally, the three orientation space axes can be combined into an orientation rotation matrix ($\boldsymbol{R}$) by:

$$\boldsymbol{R} = \begin{bmatrix} x_x & x_y & x_z & 0 \\ y_x & y_y & y_z & 0 \\ z_x & z_y & z_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7.8}$$

where x_x denotes the x value of the vector $\boldsymbol{x}$, y_x denotes the x value of the vector $\boldsymbol{y}$, and so on.

Since the hip center (referred to as the "spine base" by the Kinect v2) is the root joint, its calculation will be considered first. Because the hip center has no parent joint from which to define the y-axis vector, it is calculated differently from the rest of the joints. Collectively, the hip center, hip left, and hip right joints define a plane, which can be used to generate the orientation of the hip center. The z-axis vector of the orientation is the normal of this plane, as calculated by:

$$\boldsymbol{z} = \frac{(\boldsymbol{p}_{HC} - \boldsymbol{p}_{HR}) \times (\boldsymbol{p}_{HC} - \boldsymbol{p}_{HL})}{\|(\boldsymbol{p}_{HC} - \boldsymbol{p}_{HR}) \times (\boldsymbol{p}_{HC} - \boldsymbol{p}_{HL})\|} \tag{7.9}$$

where $\boldsymbol{p}_a$ is the three-space position of joint a, and a denotes the joint as abbreviated in Table 7.1. Next the y-axis vector is defined as the upward direction in the plane by:

$$\boldsymbol{y} = \frac{\boldsymbol{z} \times (\boldsymbol{p}_{HR} - \boldsymbol{p}_{HL})}{\|\boldsymbol{z} \times (\boldsymbol{p}_{HR} - \boldsymbol{p}_{HL})\|} \tag{7.10}$$

Finally, the orientation can be completed by calculating the x-axis vector and the orientation matrix using Eq. (7.7) and Eq. (7.8), respectively.

Next, the algorithm moves up the skeleton to the spine. The spine orientation is calculated using Eq. (7.5) - (7.8), with $\boldsymbol{x}'_{Sp}$ defined as:

$$\boldsymbol{x}'_{Sp} = \boldsymbol{p}_{SL} - \boldsymbol{p}_{SR} \tag{7.11}$$

The shoulder center is defined similarly with $\boldsymbol{x}'_{SC}$ defined as:

$$\boldsymbol{x}'_{SC} = \boldsymbol{x}'_{Sp} = \boldsymbol{p}_{SL} - \boldsymbol{p}_{SR} \tag{7.12}$$

The head and neck both defined using the same $\boldsymbol{x}'$ as the shoulder center and spine. That is to say:

$$\boldsymbol{x}'_{Hd} = \boldsymbol{x}'_{Nk} = \boldsymbol{x}'_{SC} = \boldsymbol{x}'_{Sp} = \boldsymbol{p}_{SL} - \boldsymbol{p}_{SR} \tag{7.13}$$

However, the neck joint will not be tracked in all cases. If the KVR server only has Kinect v1 sensors as inputs, the neck will not be tracked, but it will be tracked if a Kinect v2 sensor is present. Therefore, the y-axis vector of the head orientation is defined as:

$$\boldsymbol{y} = \begin{cases} \dfrac{\boldsymbol{p}_{Hd} - \boldsymbol{p}_{SC}}{\|\boldsymbol{p}_{Hd} - \boldsymbol{p}_{SC}\|}, & \text{if neck is not tracked} \\ \dfrac{\boldsymbol{p}_{Hd} - \boldsymbol{p}_{Nk}}{\|\boldsymbol{p}_{Hd} - \boldsymbol{p}_{Nk}\|}, & \text{otherwise} \end{cases} \tag{7.14}$$

The x-axis vector and orientation matrix of the head are always calculated using Eq. (7.7) and Eq. (7.8), respectively.

Moving down the left arm from the shoulder center the first joint is the left shoulder. The y-axis vector is as defined in Eq. (7.5); however, the z-axis vector and x-axis vector are calculated in the reverse order of the previous joints. Additionally, the z-axis vector of the shoulder center is used to calculate the x-axis vector of the left shoulder by:

$$\boldsymbol{x} = \frac{\boldsymbol{y} \times \boldsymbol{z}_{sc}}{\|\boldsymbol{y} \times \boldsymbol{z}_{sc}\|} \tag{7.15}$$

which leads to the z-axis vector being calculated by:

$$\boldsymbol{z} = \boldsymbol{x} \times \boldsymbol{y} \tag{7.16}$$

The next joint, the left elbow also presents an interesting calculation because the way it is calculated changes depending on the angle of the elbow joint. The y-axis vector is always defined as given by Eq. (7.5). The z-axis vector of the orientation, however, is elbow angle dependent. This angle is defined by:

$$\cos\theta_{EL} = (\boldsymbol{p}_{EL} - \boldsymbol{p}_{SC}) \cdot (\boldsymbol{p}_{WL} - \boldsymbol{p}_{EL}) \tag{7.17}$$

where $\boldsymbol{i} \cdot \boldsymbol{j}$ denotes the dot product of vectors $\boldsymbol{i}$ and $\boldsymbol{j}$. Additionally, the calculation of the z-axis vector also uses the left shoulder angle, defined by:

$$\cos\theta_{SL} = (\boldsymbol{p}_{SC} - \boldsymbol{p}_{Sp}) \cdot (\boldsymbol{p}_{EL} - \boldsymbol{p}_{SL}) \tag{7.18}$$

Based on this, the z-axis vector is defined as:

$$\boldsymbol{z} = \begin{cases} \dfrac{(\boldsymbol{y} \times (\boldsymbol{p}_{WL} - \boldsymbol{p}_{EL})) \times \boldsymbol{y}}{\|(\boldsymbol{y} \times (\boldsymbol{p}_{WL} - \boldsymbol{p}_{EL})) \times \boldsymbol{y}\|}, & \text{if } |\cos\theta_{EL}| < 0.94 \\ \dfrac{\boldsymbol{y} \times \boldsymbol{x}_{SC}}{\|\boldsymbol{y} \times \boldsymbol{x}_{SC}\|} & \text{if } |\cos\theta_{EL}| \geq 0.94 \text{ and } \cos\theta_{SL} \leq 0 \\ \dfrac{\boldsymbol{y} \times (\boldsymbol{p}_{SC} - \boldsymbol{p}_{Sp})}{\|\boldsymbol{y} \times (\boldsymbol{p}_{SC} - \boldsymbol{p}_{Sp})\|} & \text{if } |\cos\theta_{EL}| \geq 0.94 \text{ and } \cos\theta_{SL} > 0 \end{cases} \tag{7.19}$$

The threshold of 0.94 was determined by trial and error to match the Kinect v1 SDK joint algorithm as closely as possible. From there, the x-axis vector is calculated as defined in Eq. (7.7) and the orientation matrix can be generated by Eq. (7.8). Moving on down the arm, the left wrist can be calculated by Eqs. (7.5) - (7.8), with $\boldsymbol{x}'_{WL}$ defined as:

$$\boldsymbol{x}'_{WL} = \boldsymbol{x}_{EL} \tag{7.20}$$

The hand is similar, except that it has the calculations of the x-axis vector and z-axis vector reversed, such that:

$$\boldsymbol{x} = \frac{\boldsymbol{y} \times \boldsymbol{z}_{WL}}{\|\boldsymbol{y} \times \boldsymbol{z}_{WL}\|} \tag{7.21}$$

$$\boldsymbol{z} = \boldsymbol{x} \times \boldsymbol{y} \tag{7.22}$$

Again, the y-axis vector is calculated by Eq. (7.5) and the final orientation matrix by Eq. (7.8). The left hand tip and left thumb do not exist as joints in the Kinect v1, and their rotation matrix is always the identity matrix in the Kinect v2; therefore, the choice of orientation can be made free of any constraints for these two joints. To maintain consistency with the left hand joint as defined by the Kinect v1, it was decided to define the orientation of the left hand tip and left thumb in the same way, except with the x-axis vector defined as:

$$\boldsymbol{x} = \frac{\boldsymbol{y} \times \boldsymbol{z}_{HnL}}{\|\boldsymbol{y} \times \boldsymbol{z}_{HnL}\|} \tag{7.23}$$

The y-axis vector is again defined by Eq. (7.5), the z-axis vector by Eq. (7.22), and the final orientation matrix defined by Eq. (7.8). The right arm joint orientations are defined identically to the left arm, except with the corresponding right joint being used in place of the left joint.

Moving back down the skeleton to the hips, the left hip is defined by Eqs. (7.5) - (7.8), with $\boldsymbol{x}'_{HL}$ defined as:

$$\boldsymbol{x}'_{HL} = \boldsymbol{x}_{HC} \tag{7.24}$$

Next, the left knee and left ankle are calculated. The y-axis vector calculations are done, as normal, by Eq. (7.5). The z-axis vector calculations are both dependent on the knee angle, as defined by:

$$\cos\theta_{KL} = (\boldsymbol{p}_{KL} - \boldsymbol{p}_{HL}) \cdot (\boldsymbol{p}_{AL} - \boldsymbol{p}_{KL}) \tag{7.25}$$

and in practice the z-axes vectors of the knee and ankle must be done together, as in one of the three cases, the orientation of the knee is dependent on the orientation of the ankle. The z-axis vector of the left knee is calculated by:

$$\boldsymbol{z_{KL}} = \begin{cases} \dfrac{\boldsymbol{x}_{AL} \times \boldsymbol{y}_{KL}}{\|\boldsymbol{x}_{AL} \times \boldsymbol{y}_{KL}\|} & \text{if left knee is tracked and } \cos\theta_{EL} < 0.972 \\ \dfrac{\boldsymbol{y}_{KL} \times \boldsymbol{x}_{HC}}{\|\boldsymbol{y}_{KL} \times \boldsymbol{x}_{HC}\|} & \text{if left knee is tracked and } \cos\theta_{EL} \geq 0.972 \\ \dfrac{\boldsymbol{y}_{KL} \times \boldsymbol{x}_{HC}}{\|\boldsymbol{y}_{KL} \times \boldsymbol{x}_{HC}\|} & \text{if left knee is inferred} \end{cases} \tag{7.26}$$

and the z-axis vector of the left ankle is calculated by:

$$\boldsymbol{z_{AL}} = \begin{cases} \dfrac{\boldsymbol{y}_{AL} \times \boldsymbol{x}_{HC}}{\|\boldsymbol{y}_{AL} \times \boldsymbol{x}_{HC}\|} & \text{if left knee is tracked and } \cos\theta_{EL} < 0.972 \\ \dfrac{\boldsymbol{x}_{KL} \times \boldsymbol{y}_{AL}}{\|\boldsymbol{x}_{KL} \times \boldsymbol{y}_{AL}\|} & \text{if left knee is tracked and } \cos\theta_{EL} \geq 0.972 \\ \dfrac{\boldsymbol{x}_{KL} \times \boldsymbol{y}_{AL}}{\|\boldsymbol{x}_{KL} \times \boldsymbol{y}_{AL}\|} & \text{if left knee is inferred} \end{cases} \tag{7.27}$$

The threshold of 0.972 in Eq. (7.26) and Eq. (7.27) was determined by trial and error to match the Kinect v1 SDK algorithm as closely as possible. The z-axes vectors of the left knee and left ankle can be calculated by Eq. (7.7) and the final orientation matrices by Eq. (7.8). The left foot is then calculated by Eqs. (7.5) - (7.8), with $\boldsymbol{x}'_{FL}$ defined by:

$$\boldsymbol{x}'_{FL} = \boldsymbol{x}_{AL} \tag{7.28}$$

Like the arm joints, the right leg joint orientations are calculated in the same manner as the left leg joint orientations, except with the appropriate right joint used in place of the left joint. Finally, since VRPN uses quaternions to represent orientations instead of matrices, the orientation matrices are converted to quaternions using standard matrix to quaternion conversions (Möller and Haines, 1999).

7.4.5 Gesture Recognition

There are numerous gesture recognition algorithms available for virtual reality, including the $3 Recognizer, hierarchical gesture recognition, Dynamic Time Warping (DTW), and Hidden Markov Models (HMM) (Celebi et al., 2013; Kratz and Rohs, 2010; Kristensson et

al., 2012; Lee and Kim, 1999; Suma et al., 2013). In selecting an algorithm for gesture recognition in the KVR system, it was considered necessary for the recognition algorithm to work online, without a predefined starting point or stopping point, and for the algorithm to learn by example gestures, instead of requiring manual coding and tuning of gesture parameters. Based on that criteria, DTW and HMM were the two best options, and a discrete HMM method of gesture recognition was selected.

To implement a HM-based gesture recognizer in the KVR system, a discrete, left-to-right HMM is run for each gesture that is trained, as shown in Figure 7.3. As new skeleton data becomes available (from the skeleton merging), the latest joint position is added to the HMM. However, this joint must be processed first. The first step in this processing is to convert the position into a coordinate system that is relative to the user's body so that the gesture can be recognized independent of the user's orientation relative to the global coordinate system, and independent of the user's body size. The coordinate system selected for this is the coordinate system of the shoulder center joint, with all the joint lengths normalized to the distance between the users left shoulder and right shoulder. This coordinate space will be referred to as normalized shoulder coordinates. This coordinate

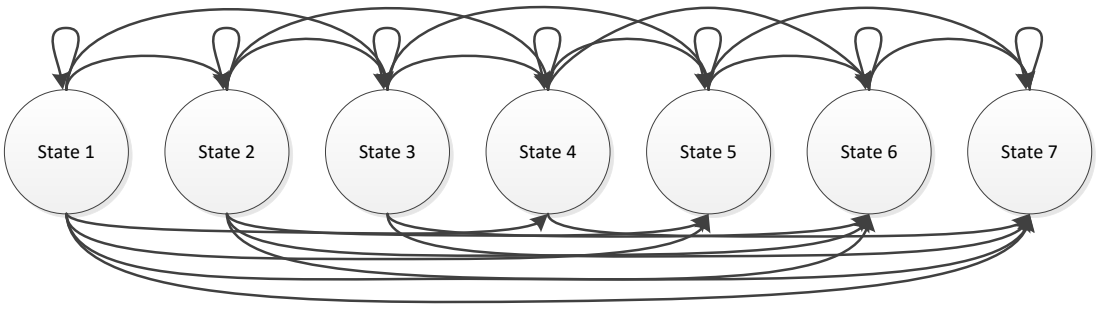

Figure 7.3: A seven state, left-to-right hidden Markov model. In this model, a state can transition to itself, or to any of the states ahead of it, but it can never transition to a previous state.

system was selected because there is a high probability that the user's shoulders will be in view if the user is tracked, whereas other central joints, such as the hip center, could be occluded if the user is close to a wall of the CAVE and the Kinect is located above. The selection of the shoulder width as the normalizing coordinate system was made because the shoulder tracking is relatively stable. While user height or user arm length would likely make a more accurate measurement of the body size variability between users, the height and arm length are both quite noisy due to the faster movements of the terminal joints (i.e., the hands and feet).

Even given the relative stability of the shoulder measurements, the conversion to a normalized, body-centric coordinate system contributes additional measurement error to the already noisy Kinect measurements. To help reduce this effect, two techniques were used. First, a Kalman filter was used to produce a filtered estimate of the shoulder width. Since the calculation of the shoulder length is a non-linear operation, it would require an extended Kalman filter if the raw shoulder positions were used as the measurements. Therefore, the shoulder width was calculated from the shoulder positions outside the Kalman filter, and the calculated width was used as the measurement input to the Kalman filter, allowing the shoulder width to be filtered using a constant scalar model, with a time varying process covariance. Finally, a relatively small process noise was used in the filter to allow the filter to generate a fairly stable shoulder estimate, that is insensitive to changes in measured value.

The second technique that was used to cope with the high noise in the positions within the normalized shoulder coordinates was to use relatively few states in the HMM. By having fewer states, there is more statistical variation inherent in each state, thus allowing for more

tolerance in measurement noise. However, this also increases the chances of the recognizer creating a false positive recognition.

In order to convert the joint position into a discrete symbol for use in the HMM, the k-means clustering algorithm was used. When the gesture is trained, the centroids of the clusters are determined based on all the observed joint positions in the training data. When the gesture is run, the algorithm can quickly compute which cluster the position is nearest to, and thus which symbol it should represent in the HMM. The discrete HMM model in KVR uses a left-to-right model that is defined and trained as described in the Rabiner tutorial (Rabiner, 1989). The HMM in KVR uses a seven state, fourteen symbol model, although this can be changed in code by users to generate a more or less strict gesture model.

The final piece that needs to be determined during the training is the threshold for a gesture to be considered as identified. To do this, all the training data are run through the HMM to find the natural log of the probability of each sequence. Twice the average log probability of all the training sequence is used as the threshold (which is slightly over half the probability). This threshold can be adjusted by the user to make the detection more or less sensitive by adjusting a scalar on the user interface.

Finally, it should be noted that due to the computation expense of running the HMM, the KVR system can only support running a single gesture at a time, on a single joint, of a single skeleton at this point in time. Other ways of running the HMM, such as using a graphics processing unit, are being explored with the intent to extend this to more gestures and joints in the future.

7.5 Validation

To validate the performance of the KVR system, the key components of the system were tested against known information, both in the form of simulated data and by feeding skeleton data from publically available datasets into the system via the networked Kinect interface. The end-to-end system latency and voice recognition performance were not tested here, as their performance is primarily dependent on the performance of external libraries, not the KVR system itself.

7.5.1 Skeleton Filtering

To validate the skeleton filtering, the Kalman filter for the skeleton was first tested using simulated joint data of a single joint moving in both a sinusoidal wave and a square wave in the x-direction, and static in the y- and z-directions. The wave forms were sampled at 33 ms intervals, with noise artificially added to the true value of the wave using a Gaussian pseudo-random number generator to simulate the measurement noise inherent in the Kinect. Both the sine wave and the square wave were set to a peak amplitude $A = 1$ m and a frequency of $f = 0.5$ Hz. The filter was tested using both the filtered data immediately after measurement integration (referred to as the filtered data) and with the measurement predicted ahead 106 ms (the predicted data), which is the average latency of the Kinect v1 sensor as determined by Livingston et al. (2012). To test the filter performance, the peak amplitude and the phase shift of the sine wave were measured by curve fitting both the filtered and the predicted data to a sinewave using Matlab R2016a. The performance of the square wave was tested by measuring the average overshoot and settling time on the rising side of the square wave, for both the filtered and the predicted data.

As the physics model used in the Kalman filter does not account for the muscular force inputs into the user's motions, the performance of the filter is fundamentally related to the magnitudes of the process noise and the observation noise used in the filtering calculations. The observation noise is well quantified from tests of the Kinect's tracking performance, and for the simulated tests is fixed at $\sigma_{ob} = 0.005$ m, which is representative of the average noise inherent in the Kinect v1 sensor (Livingston et al., 2012). Note that in the KVR system, the exact observation noise is dependent upon both on the sensor type and the measured distance from the sensor. With this value set, the process noise was tested at values of $\sigma_{pr} = 1\ \frac{m}{s^3}$, $\sigma_{pr} = 2\ \frac{m}{s^3}$, $\sigma_{pr} = 3\ \frac{m}{s^3}$, and $\sigma_{pr} = 4\ \frac{m}{s^3}$. A noise level of 50 dB SNR (signal to noise ratio) was used for these tests. The results are summarized in Table 7.2 and Table 7.3.

Table 7.2: The filter performance data for the 0.5 Hz, 1 m peak amplitude sine wave with a 50 dB SNR.

	Filtered Data		Predicted Data	
Process Noise ($\frac{m}{s^3}$)	Amplitude (m)	Phase Shift (ms)	Amplitude (m)	Phase Shift (ms)
1.0	1.012	4.9	1.076	22.5
2.0	1.004	2.5	1.043	16.5
3.0	1.002	1.6	1.031	13.6
4.0	1.002	1.2	1.025	11.9

Table 7.3: The filter performance data for the 0.5 Hz, 1 m peak amplitude square wave with a 50 dB SNR.

	Filtered Data		Predicted Data	
Process Noise ($\frac{m}{s^3}$)	Overshoot (%)	Settling Time (s)	Overshoot (%)	Settling Time (s)
1.0	43.9	0.602	221	0.639
2.0	39.7	0.480	275	0.515
3.0	38.2	0.422	322	0.466
4.0	35.3	0.384	363	0.393

Based on the information from these tests, it was decided to use a process noise of $\sigma_{pr} = 3\ \frac{m}{s^3}$ to provide the best balance of filtering and minimal lag introduced by the filter. However, different cases may require different filter parameters. Therefore, an option is

provided in the user interface to adjust this value and thereby provide more or less data filtering, with the corresponding tradeoff in lag. Finally, it should be noted that the predicted data has significant overshoot. This is unsurprising given that it is trying to predict over three frames ahead of the Kinect sensor. It should be clear from this, that while predictive tracking may be able to provide some reduction in the apparent latency from the Kinect sensor, it is not capable of reliably eliminating the entire 106 ms average latency.

To test how the filter handles different noise levels, the 1 m peak amplitude, 0.5 Hz sine wave was retested at noise levels of 50 dB, 36 db, and 10 dB SNR. The results of these simulations, for the filtered data only, are shown in Figure 7.4. It is clear that with 50 dB of

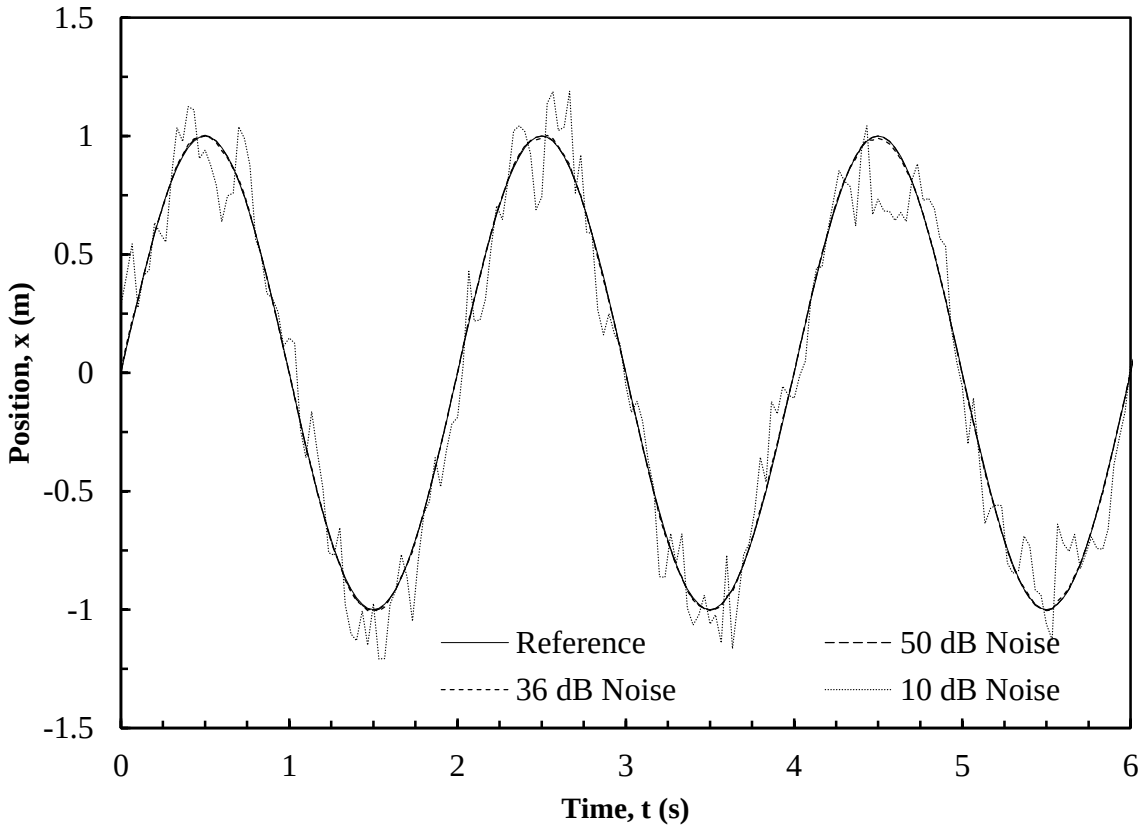

Figure 7.4: The effect of varied signal to noise ratios on the Kalman filter. Note that the 50 dB curve and the 36 dB curve lie underneath the reference curve in most places.

noise and 36 dB of noise, the filtered data almost perfectly follows the reference 0.5 Hz sine wave. However, at 10 dB of noise, the noise level is sufficient that it is difficult for the filter to reconstruct the original waveform. This can be seen particularly well around the peaks and valleys of the sine wave.

Finally, the filter was tested against real human motion data, using the Cornell Activity Dataset 60 (CAD 60) (Sung et al., 2011). Testing against this dataset generates far too many data sequences to show here, therefore two selected curves are shown to illustrate what works well, and what doesn't. The first data sequence is the z-axis of the right hand of the random movement sequence of person 1 from the CAD 60 dataset, shown in Figure 7.5. This sequence has relatively large movement (about a 1 m range), as well as large sections

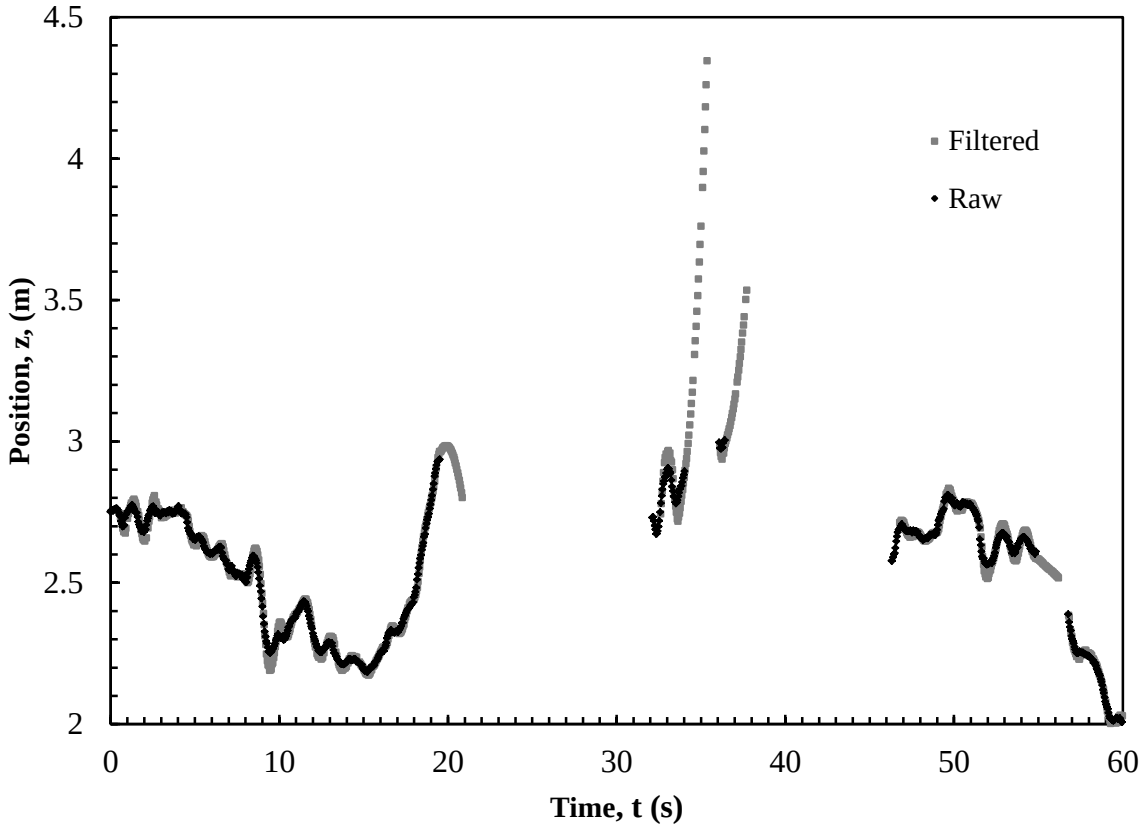

Figure 7.5: The raw z-position (black) and filtered z-position (gray) of the right hand of person 1, random motion sequence, from the CAD 60 dataset.

where tracking is lost and the filter must estimate the joint position. It can be seen that, in general, the filtered data follows the raw data well; however, there is some overshoot on direction changes, as can be seen at $t = 9$ s. In addition, the filter performance after tracking is lost is mixed. When tracking is lost at $t = 34$ s, the filter quickly starts producing invalid data. However, when the tracking is lost at $t = 55$ s, the filter maintains a reasonable estimate for almost a full second.

The second selected data sequence is the y-position of person 1, random motion, from the CAD 60 dataset. This data is relatively static, with few missing data points. It can be seen from Figure 7.6 that the filter very closely follows the movement of the raw data. In fact, for a position this static, it would be preferable for the filter to remove more motion noise from the raw data; but this filtering power was lost in reducing the filter lag.

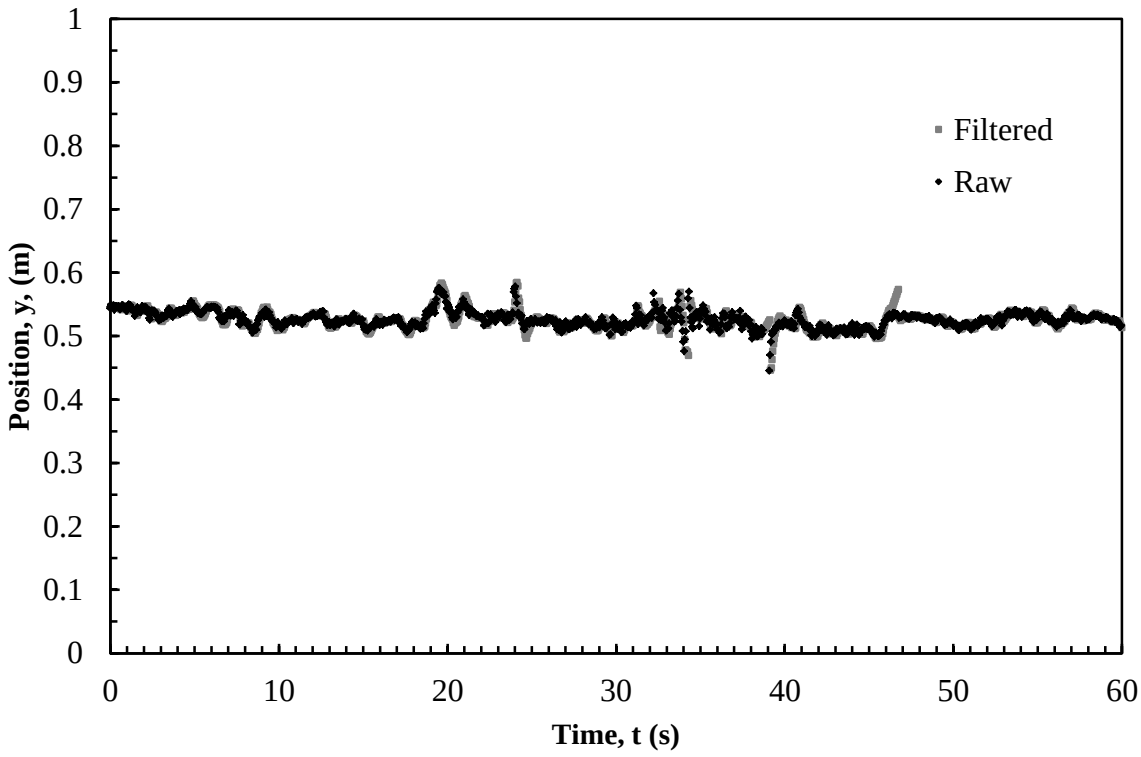

Figure 7.6: The raw y-position (black) and filtered y-position (gray) of the head of person 1, random motion sequence, from the CAD 60 dataset.

7.5.2 Joint Orientation Algorithm

To test the consistency of the joint orientation algorithm with the algorithm in the Kinect v1 SDK, joint orientations were generated with both algorithms using known skeletons and compared. This comparison was done by calculating the angle between the $\boldsymbol{x}$, $\boldsymbol{y}$, and $\boldsymbol{z}$ vectors of each orientation matrix. The first set of skeletons that was used for the comparison was a set of 10,000 skeletons generated using a uniform pseudo-random number generator, where each component of each joint position was pseudo-randomly generated within a range of -3 to 3 m. From this data set, the average angular difference for each orientation vector was computed. As most of the angular differences are quite small, the number of "incorrect" orientations was also counted, with an "incorrect" orientation being any orientation that had an error greater than 0.5°. Based on this random data, the average $\boldsymbol{y}$ vector error for all joints was less than 0.000005°, with no "incorrect" orientations in the y-direction. This indicates that the joint orientation algorithm is nearly identically replicating the one known constraint of the joint orientations – that the y-axis of the orientation points from the previous joint to the current. With it known that there is no error in the y-axis, the x-axis and z-axis must be constrained to a single plane of possibilities, and thus the angular difference of the $\boldsymbol{x}$ vector and the angular difference of the $\boldsymbol{z}$ vector must be the same for a given joint. Therefore, going forward, the vector of difference will not be specified, as it could equally refer to the $\boldsymbol{x}$ or $\boldsymbol{z}$ vector.

The average difference and number of 'incorrect" orientations are listed in Table 7.4. Note that only the joints that are available in the Kinect v1 were simulated, as those are the only ones for which a joint for comparison can be calculated. It can be seen that only the joints on the arm (elbow, wrist, and hand, both left and right) show any "incorrect" joints and

that the average difference on the remaining joints is small enough to be insignificant. Even with the occasional error in the arm joints, they still are close enough to be considered the same orientation in greater than 99.8% of cases. Furthermore, it should be noted that the "incorrect" joints in the wrist and hand only occur due to an incorrect calculation in the wrist being propagated down the joint hierarchy.

Table 7.4: Comparison of the Kinect v1 SDK's joint orientation algorithm with the KVR system joint orientation algorithm using pseudo-random skeletons.

Joint	Average Difference (degrees)	Number of "incorrect" out of 10,000 trials (-)
Hip Center	0.000003	0
Spine	0.000002	0
Shoulder Center	0.000002	0
Head	0.000003	0
Shoulder Left	0.000002	0
Elbow Left	0.073947	13
Wrist Left	0.073574	13
Hand Left	0.075711	13
Shoulder Right	0.000003	0
Elbow Right	0.043858	6
Wrist Right	0.043691	6
Hand Right	0.043219	6
Hip Left	0.000003	0
Knee Left	0.000003	0
Ankle Left	0.000003	0
Foot Left	0.000003	0
Hip Right	0.000003	0
Knee Right	0.000004	0
Ankle Right	0.000003	0
Foot Right	0.000003	0

However, the vast majority of the skeletons created by the pseudo-random skeleton generating process are not possible for a human to create. Therefore, the same tests were repeated using the entire CAD 60 dataset for the input skeletons, which resulted in 84,299 total tested skeletons (Sung et al., 2011). This requires a slight remapping of joints, however, as the CAD 60 dataset was acquired using OpenNI instead of the official Kinect v1 SDK.

First, a hip center joint was artificially created, using the center point between the left hip and right hip. Second, the hand joints in the CAD 60 data were used as the wrist joints, and the foot joints were used as the ankle joints. This meant that the hands and feet orientations were not calculated. The results of this test is summarized in Table 7.5. Like the pseudo-random data, no significant differences were found between the Kinect v1 SDK algorithm and the KVR system's algorithm for the z-axis, thus only the difference on the x-axis is presented (which is identical to the difference on the z-axis). It can been seen from the data, that the occasional "incorrect" orientation in the elbow joints again occurs. However, there is also one "incorrect" orientation that occurs in the left knee which was not seen in the pseudo-random data. This "incorrect" orientation propagates to the ankle orientation, and would

Table 7.5: Comparison of the Kinect v1 SDK's joint orientation algorithm with the KVR system joint orientation algorithm using the CAD 60 dataset.

Joint	Average Difference (degrees)	Number of "incorrect" out of 84,299 trials (-)
Hip Center	0.000006	0
Spine	0.000001	0
Shoulder Center	0.000001	0
Head	0.000001	0
Shoulder Left	0.000001	0
Elbow Left	0.000447	1
Wrist Left	0.000456	1
Hand Left	N/A	N/A
Shoulder Right	0.000001	0
Elbow Right	0.027905	44
Wrist Right	0.027177	44
Hand Right	N/A	N/A
Hip Left	0.000006	0
Knee Left	0.000043	1
Ankle Left	0.000059	1
Foot Left	N/A	N/A
Hip Right	0.000006	0
Knee Right	0.000001	0
Ankle Right	0.000002	0
Foot Right	N/A	N/A

propagate to the foot, had it been tracked. However, even given that, on the real data, greater than 99.94% of all skeletons produce correct orientations for all joints. One interesting item of note, all the "incorrect" orientations occurred on data from person 2 in the CAD 60 dataset. At this point, it is unclear if this was just coincidence, of if there is something about the movements of person 2 or the way person 2 was tracked that contributed to the occurrence of the "incorrect" joint orientations.

7.5.3 Gesture Recognition

To validate the gesture recognition in the KVR system, the gestures were tested using the Microsoft Research Cambridge-12 (MSRC-12) gesture dataset (Fothergill et al., 2012). The skeleton sequence from this dataset was transmitted to the KVR system via a VRPN server to the networked Kinect interface that KVR provides. Since the KVR system currently only supports recognition of a single gesture, monitoring a single joint, and many of the gestures in the MSRC-12 dataset involve multiple joints, it was decided to test the gesture recognition on the "Change Weapon" gesture from the MSRC-12 dataset.

To train the gesture, the first two repetitions of the gesture from the first five people instructed with images and text were used (a total of ten training sets). Only ten training sets were used to mimic a realistic training size that a KVR user would train when making a custom CAVE application, and the images and text instructed dataset was selected from Fothergill et al. (2012), who indicate that this method of training provides the best coverage. The gesture recognition was then tested against all ten repetitions of each of the five remaining people for the "Change Weapon" gesture. In this test, the gesture was correctly recognized 86% of the time. However, there was also an average of 3.4 spurious or duplicate recognitions per sequence. Most often, the system recognized a single gesture twice due to

the probability going above the threshold twice in rapid succession. This indicates that fine-tuning of the threshold may be able to reduce the spurious detections.

Additionally, the gesture recognizer was tested against all 10 repetitions of the "Start System" gesture from the MSRC-12 dataset to determine the likelihood of a false detection during non-gesture movements. From this testing, an average of 3.6 false detections were detected, per sequence. It is interesting to note, however, that over 66% of all the false detections occurred during a single sequence, indicating that variability in the way both gestures, and non-gestures are performed between people may have a significant impact on the quality of detection. Therefore, it is recommended that when using gesture recognition in virtual reality, that the gesture training be done, at least in part, on the intended system user whenever possible.

Finally, it should be noted that, during the testing, it was observed that the quality of the recognition is heavily dependent on how the training is done. In particular, defining when a gesture starts and stops on the training data had a significant impact on the quality of the recognition. Therefore, it is recommended that the user records the training data with an interactive playback system (such as the Kinect Studio software provided with the official Kinect SDK) and experiment with training the gesture with multiple different start and stop positions to determine which ones work best (Microsoft, n.d.-g).

7.6 Conclusions

Using the Kinect sensor in a CAVE-style virtual environment has its challenges – there are few good places to put the Kinect sensors, multiple Kinects are often required for tracking, and integrating the Kinect's user interaction systems with VR applications can be difficult. However, the potential to provide unencumbered full body tracking has made the

Kinect sensor popular in VR anyway. This paper has presented the Kinect with Virtual Reality system, a VRPN interface to abstract the challenges of using Kinect's in CAVEs from the details of implementing a VR application. As has been shown, this system successfully integrates skeleton filtering, the merging of skeletons from multiple Kinect sensors, voice recognition, and gesture recognition. Additionally, for what is believed to be the first time publically, an algorithm to calculate joint orientations from Kinect joint positions, that is logically consistent with the Kinect v1 method, has been demonstrated.

While this system has room for improvement, the open source nature of the system means that users can modify it to fit their own needs. Additionally, developers of virtual reality applications can use the system as it stands, and as improvements to the KVR system are made, they will be able to see those improvements in their VR applications without having to change a single line of code. Finally, this software system has been made open source with the hope that other researchers will build upon the foundations herein. Many of the pieces this system works to integrate into virtual reality, such as sensor filtering, predictive tracking, and gesture recognition, are research topics unto themselves. It is hoped that researchers with specific expertise in those areas will contribute to the continued improvement of this software, allowing them to provide their achievements to a wider audience, and also allowing those with expertise in virtual reality to focus on improving user interaction within VR, instead of building the prerequisite mechanics behind the interaction.

CHAPTER 8:

A PROPOSED SYSTEM FOR INTERACTIVE VISUALIZATION OF VOLUMETRIC MULTIPHASE FLOW DATA IN VIRTUAL REALITY

In this chapter, objective five of this research is addressed by proposing a system to visualize and interact with multiphase flow data in virtual reality. While the implementation and evaluation of this system is beyond the scope of this dissertation, this chapter is important because it provides the overarching vision of what can be achieved using the tools and techniques developed throughout this research. Note that, while this system has not fully been implemented, pieces of it have been realized as test cases, and everything that is proposed here is fully achievable with current technology.

8.1 Abstract

As the amount of three-dimensional multiphase flow data that can be collected grows, the ability to visualize it effectively becomes increasingly critical (Hansen and Johnson, 2005). While a plethora of tools exist to visualize multiphase flow measurements on standard computer screens, there are relatively few that leverage the third dimension provided by virtual reality. Additionally, many previous visualization methods have high barriers to use in the real world because of challenging user interfaces and the encumbrances upon the user required to achieve tracking. This paper presents one vision of how to leverage the benefits of virtual reality, and overcome its challenges, in order to provide researchers with a better tool for visualizing multiphase flow data.

8.2 Introduction

Since the early days of virtual reality (VR), one of the active areas of research has been the interactive visualization of fluid flow data in virtual environments (Bryson and Levit, 1992). Virtual reality is particularly useful in flow visualization because it allows the three-dimensional structures of the flow to be visualized without reducing the data to a two-dimensional rendering. It also has advantages over the visual observation of real flows because it allows users to see things that are not visible to the naked eye (such as the inside of opaque flows), and the users presence does not disturb the flow in question. However, much of the visualization of flows in virtual reality has been done using computationally generated data from flow simulations (Duncan and Vance, 2007; Hansen and Johnson, 2005). There is still a need for systems designed for the visualization of experimentally obtained flow data.

One of the main challenges in designing such a system is the wide variety in measurements that can be obtained of a flow. Magnetic resonance imaging (MRI), X-ray computed tomography (CT), ultrasonic tomography, electrical impedance tomography (EIT), particle image velocimetry, and particle tracking velocimetry, to name a few, all produce slightly different measurements of a fluid flow and thus all have slightly different requirements for visualization (Chaouki et al., 1997; van Ommen and Mudde, 2008). However, the tomographic techniques (MRI, CT, EIT, and ultrasonic tomography) all produce a three-dimensional scalar volume of an individual property of the flow. For example, CT produces a volumetric dataset where each volume element (voxel) corresponds to the time-averaged density of the flow at that location (Heindel, 2011). Leveraging this data similarity, this paper proposes a virtual reality application dedicated to the visualization

of volumetric multiphase fluid flow data. Specifically, it will focus on X-ray computed tomography data from the X-ray Flow Visualization (XFloViz) facility at Iowa State University (Heindel et al., 2008); however, the data commonality should provide for applicability beyond just X-ray CT measurements.

Experimental volumetric data is not unique to the area of flow measurement. Volumetric measurements are commonly found in medical imaging in the form of MRI and CT scans. There have been many attempts to visualize these data in virtual reality, which provides a good foundation for the work herein (Haubner et al., 1997; He et al., 2007; Noon, 2012). However, it should be noted that there is one key difference between medical volumes and fluid flow volumes. In medical imaging, there are distinct organs with relatively sharp transitions. In flow data, however, the data are typically time averaged, resulting in relatively diffuse transitions and few sharp features. The result of this is that segmentation (the defining of object boundaries), which is critical in medical visualization, is not of much use in the visualization of fluid flows. Conversely, the use of a region of interest (ROI) to selectively remove data from the visualization takes on greater importance in flow visualization.

8.3 Proposed System

He et al. (He et al., 2007) identified four basic interaction tasks required for volume visualization in virtual reality:

- Volume object transformation (rotation, translation, etc.)
- Volume exploration (virtual tools such as clipping planes and segmentation)
- Transfer function specification
- System control (opening data files, closing the software, etc.)

Based on these tasks, five critical tasks for flow visualization in virtual reality have been defined:

- Viewpoint manipulation
- Viewpoint sharing
- Region of interest selection
- Transfer function specification
- System control

Each of these five tasks will be considering in the following sections.

However, prior to the discussion of per-task implementations, the hardware to be used must first be specified, as the capabilities of the hardware will inform the trade-offs required in implementing the tasks. First, the virtual environment selected to be used is the Multimodal Environment Testbed and Laboratory (METaL) at Iowa State University. This system was selected because it is a CAVE-style system which provides better collaboration between users than do head mounted display (HMD) systems. Additionally, while this system has a higher up-front cost than do HMD systems, the operating costs are low enough (about $1 per hour) that users can be free to explore data without worrying about minimizing their time in the environment to limit costs. To provide user interaction, two systems are available. Head tracking and a tracked wand (a modified Nintendo Wii Remote) are provided using an Advanced Realtime Tracking (ART) optical marker-based tracking system. Four Kinect sensors (two Kinect v1 sensor and two Kinect v2 sensors) are also available to provide full-body markerless tracking of the users via the Kinect with VR server (Chapter 7). In this study, the ART head tracking will be used to provide high-precision head tracking to reduce the likelihood of cybersickness. However, the Kinect sensors will be used

instead of the wand to provide unencumbered fully body user interaction, at the cost of reduced precision.

The rendering of the volumetric data in this system will be done using an outside-in method. That is, the user stands in the virtual environment and looks into the rendered data from the outside, much as the user would if the user was looking at a real flow inside a transparent containment vessel. The data will be rendered using standard direct volume rendering methods. In particular, the volume will be rendered using GPU (graphics processing unit) volume ray casting. This limits the system to rendering datasets which are smaller than the amount of memory available on the GPU; however, given the amount of memory available on modern graphics cards, this is only expected to be a serious limitation in the largest of the available datasets.

8.3.1 Viewpoint Manipulation

Effective viewpoint manipulation is one of the most important techniques for the visualization of multiphase flow data, as it assists the user in understanding the complex spatial relationships within the data (Bowman and McMahan, 2007). The most basic, and most important, way to achieve this is through the head tracking of the user, and corresponding changes in viewpoint. Research has shown this method to be efficient, natural, and leads to a higher spatial knowledge in the user (Bowman et al., 2004). However, it is not possible for the user to view the data from all angles using head tracking alone. Therefore, a second method of viewpoint manipulation is necessary. This is provided in the ability to translate, rotate, and scale the visualization. If all three manipulations are afforded for, it would require nine degrees of freedom (DOF) in manipulation; however, in the visualization of multiphase flow data, cases where the user would want to scale the data

anisotropically are rare. Therefore, only isotropic scaling is provided to the user, reducing the manipulation to seven degrees of freedom. Complicating these considerations further is the fact that, while the Kinect sensor's joint positions are relatively accurate (albeit noisy), the joint orientation information is unreliable. Therefore, each of the user's hands should be treated as an independent 3-DOF tracked point.

One method that has shown to be efficient for translation and rotation using two 3-DOF tracked hands is the handle bar technique. In this technique, the virtual object is manipulated as if it is rigidly attached to a bicycle handlebar, and each end of the bar is grabbed by the user's hands (Bossavit et al., 2014). This technique particularly excels at complex rotations. However, it has been shown to provide poorer results in translation tasks due to the requirement to use both hands simultaneously. Therefore, for manipulating the multiphase flow visualization object, a modified handle bar technique was selected. In this technique, rotation and scale are combined in the handle bar rotation, but there is no translation provided. To select the object, the user moves both hands inside a virtual bounding box around the visualization, and then closes both hands. As the user's hands move, the rotation of the object is adjusted based off the angle from where the handle bar was when the user's hands first closed. The scale is then computed based off the ratio between the distance between the users hands at a given time and the distance between the hands when they were first closed.

Since this modified handle bar method does not provide a method for the user to translate the visualization, a separate method must be provided. The ability to translate the visualization is provided by a single handed interaction. When only one of the user's hands is closed inside the bounding box of the visualization, the visualization becomes virtually

attached to the user's hand until the hand is reopened or the other hand is closed to engage the handle bar rotation-scale interaction.

Finally, there are a few pit falls in this technique. First, for logical consistency, the bounding box of the volume must scale with the scaling of the visualization. However, if the user scales the visualization extremely small, the bounding box may be hard to re-enter with both hands to scale the visualization larger again. Therefore, there is a minimum size the bounding box will scale to, even if the visualization is scaled small. The second possible pitfall occurs if a user manipulates an object very near to one of the walls of the CAVE, and due to tracker error, it gets stuck behind the wall where the user cannot interact with it. This could be accounted for by allowing the user to move in virtual space, instead of just moving in real space. While various flying metaphors exist to do this, users tend to use physical motion less when virtual motion techniques are available (Bowman et al., 2004; Mine et al., 1997). This in turn would reduce the spatial understanding advantages provided by head tracking the user. Considering that this system is intended to primarily work as an outside-in visualization device, it was decided not to include any method for virtual motion. If a visualization gets into an unmanipulable position, it can be reset to its original position using a reset command in the menu (Section 8.3.5).

8.3.2 Region of Interest Selection

To allow the user to selectively remove information from the visualization, a method of changing the ROI of the volume must be provided. To do this, a widget is provided on screen for each of the six required clipping planes to define a rectangular prism ROI. This widget consists of a semi-transparent plane with a large sphere at the end of the normal vector of the plane. When a user's hand intersects the sphere, it changes from gray to green

to provide a visual indication that it has been intersected. The plane is then selected by the user's hand closing while intersecting it. From there, the movement of the hand translates the clipping plane in or out along its axis until the user's hand re-opens. The ROI widgets can be shown or hidden using either voice commands, or by selecting a menu item.

8.3.3 Transfer Function Specification

The specification of transfer functions is one of the most challenging tasks to accomplish in virtual reality, as it requires a great deal of precision. Studies on the related technique of windowing have found it to be less efficient to do using non-contact interfaces as with a traditional keyboard and mouse interface (Juhnke et al., 2013). To provide the required interaction, two possible methods are made available.

The first method of adjusting the transfer function is using a touchscreen (Duncan and Vance, 2007; Kim et al., 2009; Krum et al., 2014). The use of a touchscreen helps improve accuracy by reducing the degrees of freedom of the movement (Bowman et al., 2004). However, as viewpoint manipulation requires both of the user's hands, it does not leave the user with a hand to hold the touchscreen when it is not in use. Therefore, a stand is provided to the user for the touchscreen. This is not an optimal solution, as it can occlude certain viewpoints and adds a physical obstacle the user must avoid. However, this solution has the potential to be improved through the use of a mobile robot to optimally position the touchscreen wherever the user needs it (Pavlik et al., 2013).

The second method of manipulating the transfer function increases accuracy by scaling the manipulation to a large size, which is more compatible with the precision of movement of a user's arms in VR. To do this, a grayscale bar is presented on the floor of the CAVE, to correspond to the gray levels in the volume. To manipulate the color and transparency that

maps to that point, a large spike (resembling a lawn dart) is used for each color mapping position. On top of this spike is a colored sphere, representing the color that point should be mapped to. When the user's hand intersects this sphere, four large sliders appear (for red, green, blue, and opacity), which the user can then use to adjust the color and transparency. To put a new point in the transfer function, a spike is provided off the grayscale bar, which the user can pick up and place where they desire on the bar. To remove a point from the transfer function, the spike is simply moved off the bar. This transfer function selection tool can be selectively shown or hidden from the system menu.

8.3.4 Viewpoint Sharing

Ultimately, the goal of visualizing multiphase flow data in virtual reality is to find unique and interesting features in the flow, which the user will likely want to share with other researchers. If there is another researcher with the user while they are interacting with the data in VR, this can be accommodated in the CAVE by simply having the second researcher stand near the head tracked user. However, this still doesn't give the user the exact perspective, nor does it help communicate with users who are not physically present during the visualization. To accommodate this, two options are provided, both in the menu system (Section 8.3.5) and via voice command. The first option is to freeze the head tracking. By freezing the head tracking temporarily, non-head tracked users can swap places with the head tracked user and see exactly what they were seeing. While this doesn't provide the secondary users with the spatial understanding advantage of head tracking, it does allow them to see exactly what the head tracked user was seeing.

The second option provided is to save a screenshot of what the user is seeing. This allows the user to save a visual record of what they were seeing, that can then be used to

share with colleagues. To simplify interaction, these screenshots are saved, with sequential numbering, in a location defined prior to starting the visualization. After completing the visualization, the user can sort and rename screenshots using the traditional desktop interface.

8.3.5 System Control

The final interaction method for the system is the system control. To provide system control in virtual reality, flat menus are often provided that can be interacted with either using direct manipulation or a ray cast from a wand (Bowman et al., 2004; Mine et al., 1997). However, these interfaces often appear to be a forcing of the traditional windows, icons, menus, and pointer (WIMP) system on virtual reality. Based on Fitts' law, the ideal menu system would place all the menu items in a sphere around the user's dominant arm, thereby making the distance to any menu item the same (Fitts, 1954). However, in practice, there is usually no way for the system to know which of the user's arms is dominant, and a fully spherical menu would occlude the rest of the environment.

Based on the drawbacks of previous VR menu systems and Fitts' law, a menu system is proposed that both minimizes user effort, and utilizes the three-dimensional capabilities of virtual reality. This system has been named the "halo menu." In this system, a ring of three-dimensional icons floats centered around the use'rs head. This keeps the distance to each menu item roughly the same distance from the user, and keeps the menu from occluding the virtual environment. Additionally, the ring can be moved up or down by the user, allowing it to be more or less in view as desired. Interaction with the menu is achieved by intersecting the user's hand with a virtual bounding box around each icon. When this is done, the icon changes from grayscale to color and, if there is a menu associated with the item, it is automatically lowered. To select an item from the menu, the user intersects it with a hand

and then grabs the item. In addition to the ability for the user to control the menus using hand grabs, each icon has text underneath it. The system implements voice recognition so the user can select the item by simply speaking the name of the item to be selected.

8.4 Conclusions

In this paper, a system to interact with experimentally obtained volumetric data of multiphase flows was proposed. This system is based on principles from medical volume visualization systems, as well as research into direct interaction with virtual reality. However, it has some unique features to increase adoption in the multiphase flow research community. Microsoft Kinect sensors were selected as the primary mode of interaction, as they provide no encumbrances that may reduce a user's desire to use the system. The user's viewpoint can naturally be changed either using the CAVE's head tracking, or via a modified handle bar manipulation method. The ability to control the ROI of the volume has been made available via clipping plane widgets, and transfer function control is available via both a touchscreen interface, and a system of large-scale spikes in the virtual environment. To assist the user in sharing unique flow features, the ability to freeze the head tracking to show locally present users features of interest is provided. The ability to take screenshots from the head tracked user's point of view is also provided so remote users can be provided visualization easily. Finally, a unique system of menus has been proposed to leverage the three-dimensionality of VR, minimize occlusion of the virtual environment by the menu, and allow natural user interaction with the system control functions.

Going forward, user studies of this system will be required to identify what components work well for multiphase flow researchers, and what components still need work. Additionally, studies need to be conducted to examine the barriers to use that exist for virtual

reality in multiphase flow research. Bowman and McMahan (Bowman and McMahan, 2007) once stated "if all that these technologies provide for the user are oohs and ahs and a unique user experience, it would be difficult to justify the expense and development complexity that immersive VR requires." It is worth adding to that sentiment: it doesn't matter how much better a VR visualization is, if no one is willing to use it.

CHAPTER 9:

CONCLUSIONS AND FUTURE WORK

Throughout this dissertation, work has been presented intended to advance the current state of multiphase flow characterization by integrating the fields of noninvasive X-ray imaging and virtual reality. Towards this end, five objectives were laid out in Section 1.2 – increasing the frame rate of X-ray stereography, determining the sensitivity of X-ray computed tomography to changes in acquisition parameters, improving tomographic reconstruction from limited data, advancing natural user interaction with virtual reality, and proposing a system that visualizes the X-ray data in virtual reality in a natural way. The work done to achieve these goals is summarized in Section 9.1. Finally, every piece of research inevitably results in as many questions as answers. A number of possible future paths that have arisen from this research are presented in Section 9.2.

9.1 Conclusions

The first objective of this research was to show that the frame rate of X-ray imaging could be increased to allow for high-speed X-ray imaging. This was addressed in Chapter 4 of this dissertation, which demonstrated a proof of concept system for high-speed radiograph acquisition. It was shown to produce 1024 × 1024 radiographs at 1000 FPS, with the potential to run at even higher speeds. In addition to the increased speed, the faster shutter speed was able to achieve a high-quality stop motion effect, eliminating the motion blur found with high velocity flows measured using an older camera system. This imaging clarity was shown to be extremely beneficial for doing X-ray particle tracking, as the particle

recognition rates increase to 99.98%, well beyond the 70–90% of previous studies. Furthermore, while camera synchronization was not tested due to the lack of a second camera, the camera synchronization is nearly ubiquitous on high-speed cameras now, and it would be trivial to implement should a second camera become available.

Towards the second objective of this research, determining the sensitivity of X-ray computed tomography measurements to changes in acquisition parameters, Chapter 5 presents a study of the sensitivity of acquisition parameters on CTs of a gas-solid flow. This study found that, in general, raw CT values are an unreliable measurement, and are changed significantly when detector exposure time, X-ray tube voltage, or X-ray tube current is changed. However, by calibrating the system using the gas holdup calculation (which is standard practice in multiphase flow measurement) the results are insensitive to acquisition parameters – provided sufficient X-ray energy is used so that the image is neither grossly under or over exposed. Additionally, it has been shown that while large changes to the center of rotation used in reconstruction can cause significant geometric distortions in the image, those distortions do not appreciably change the average results.

The third objective of this research was to improve tomography reconstruction to allow for the generation of time-varying three-dimensional datasets. While this would be a trivial task if an acquisition system was available that was capable of imaging at 360 different angles around the object in a fraction of a second, such a system is not available. Therefore, Chapter 6 of this research focused on using two time-synchronized radiographs to reconstruct an approximated tomographic slice. Two algorithms were presented to do the approximate CT reconstruction, a locally axisymmetric filtered backprojection algorithm and a simultaneous algebraic reconstruction technique using a priori information. The locally

axisymmetric filtered backprojection algorithm was able to nearly perfectly match a full CT reconstruction for scans with a small number of distinct features, that were axisymmetric, such as the sphere phantom. Additionally, the algorithm had some success at identifying features within a dynamic fluidized bed. However, the requirement to identify specific features prior to reconstruction limits this algorithm's suitability in cases where distinct features do not exist, or are too numerous to manually identify and reconstruct. Conversely, the simultaneous algebraic reconstruction technique with a priori information did not require the identification of individual features; however, it was not as successful at reconstructing the geometry of the objects. Additionally, it was found that noise in the projections created significant artifacts in the slice reconstructions. Finally, while neither of these algorithms was tested on time sequences of data, the extension to them is trivial.

Next, the fourth objective, advancing natural user interaction in virtual reality, was addressed in Chapter 7. In order to improve user interaction, a flexible Kinect server was built that provides encumbrance free user interaction with multiphase flow data and accelerates the development of natural user interaction in other applications. This system specifically targeted CAVE-style virtual environments, which have more challenges using Kinect sensors. To handle these challenges, the system integrated voice recognition, a skeleton filtering and merging algorithm, a method for calculating joint orientations from joint positions, and basic gesture recognition. All of the information collected from the multiple Kinects it supports is merged, and then abstracted using VRPN. As a result, virtual environment designers can achieve the same performance as if the Kinect was hard-coded into the application, without any of the work required to hard code Kinect support.

Finally, objective five, proposing a system to use virtual reality to aid in the characterization of multiphase flows, was handled in Chapter 8. This chapter proposes using an outside-in viewing strategy to visualize CT scans of multiphase flows. The most important component of the interaction, changing viewpoints, can then be handled by the physical motion of the user around the object, a method that has been shown to improve user immersion. The ability to manipulate the size, position, and orientation of the dataset is also provided through one-handed translation and a two-handed combined scale-rotate method, both using the Kinect sensor and the KVR system presented in Chapter 7 for unencumbered interaction. Finally, a novel menu system, accessible by both voice and gestures is presented to maximize the availability of system functions, while minimizing its intrusiveness into the visualization.

9.2 Future Work

A researcher's work is never completely finished, and this work is no different. For all of the work presented herein, there are more questions that remain to be answered. Some of these are discussed below.

With regards to high-speed X-ray radiography, the future work has already begun. Since the original testing of high-speed imaging with X-rays, a new Photron AX50 high-speed camera has been purchased that will be dedicated to high-speed X-ray imaging. However, as this camera is being broken in, there are still numerous questions that remain. One of the key questions is what is the decay time of the phosphor in the X-ray image intensifier, can this effect be compensated for, and how? Additionally, initial testing seems to indicate that the intensifier noise is a more significant issue with the high-speed camera than it was at lower

speeds. This noise needs to be quantified, and methods for handling it need to be investigated.

When it comes to the sensitivity of X-ray computed tomography scans to changes in X-ray imaging parameters, most of the questions for the X-ray Flow Visualization facility have been answered. However, this is only a single system. To really understand the sensitivity and reliability of X-ray CT, the same tests must be repeated across multiple systems. Additionally, there is a potential error source that was not examined in this study – dynamic bias error. Dynamic bias error is a misestimation of the gas-holdup of a multiphase system due to the movement of the flow during the scan. This error has been shown to be non-negligible in γ-ray computed tomography (which typically has longer acquisition times than X-ray CT), but little investigation has been done on X-ray computed tomography (Andersson et al., 2012).

With regards to the approximate CT reconstruction algorithm, while this initial work has shown the viability of approximating a CT reconstruction from two time-synchronized radiographs, more work is still needed to improve the system. First, there are still a variety of possible reconstruction algorithms to analyze for their suitability. Second, while the system works on systems with relatively few features, the system still needs more work to handle systems of numerous features, such as highly turbulent air-water bubbling flows. Finally, the foundation that has been laid for the parallel-beam geometry needs to be extended to fan-beam and cone-beam systems to obtain more accurate size measurements.

Shifting focus to virtual reality, the work on the Kinect with VR system will continue. In particular, the skeleton filtering and gesture recognition need continued attention. With regards to filtering, while there are other possible state transition models to try, one of the

more promising ideas is to extend the concept of velocity dependent filtering from the 1€ Filter to Kalman filtering by increasing the assumed process noise at high velocities, and decreasing it during times of low process noise (Casiez et al., 2012). Additionally, the possibility exists to use the assumption that the user's bone lengths are constant (which should be a reasonable assumption) to improve the filtering. However, doing so will likely involve some fairly computationally intensive non-linear Bayesian statistics. Finally, with regards to gesture recognition, the current recognizer is serviceable for some gestures, but fails on others. The use of a Hidden Markov Model based recognizer is still believed to be a good choice. However, it is suspected that a better cluster algorithm than k-means should be used to discretize the data. Additionally, there exists the potential to improve the HMM by using a different system model. In voice recognition, time-decaying states have shown promise, and such a technique could also be applicable to gesture recognition (Rabiner, 1989). Another possibility is to leverage the fact that most gestures in VR use body motion instead of a set body-pose, and do the recognition on the velocity of the joints instead of the position.

With regards to the proposed system for visualizing CT data in virtual reality, the most important next step is to test the system with actual multiphase flow researchers and examine the usability of the system and the likelihood researchers would actually use it. Additionally, as the approximate CT reconstruction algorithms slowly push the technology towards time-sequences of volumetric data, the ability to render 4D information will become more important. Furthermore, the ability to visualize computational fluid dynamics simulation results with the experimental results in VR could also be implemented to provide a powerful tool for comparing simulations to real data. Finally, with the rapid advances being made in

head mounted displays, both for virtual reality and augmented reality, bringing this sort of visualization to large numbers of researchers may soon be possible. Therefore, finding ways to adapt the interactions from the CAVE, to a seated HMD system will be important.

Finally, this research has inspired the desire to investigate a few broad areas of research more thoroughly. First, while X-rays (both radiography and CT) are an excellent noninvasive method of characterizing multiphase flows, the technique is limited in some areas, particularly, when two of the phases of the flow have similar densities. One of the emerging tools for noninvasive flow imaging that has the potential to remedy this limitation is magnetic resonance imaging. The biggest challenge with MRI currently, is that nearly all the systems available are designed for medical use, and thus have a horizontal bore for imaging. The majority of multiphase flows of interest would require a machine with a vertical bore. However, should the funding be available to build a custom, vertical bore MRI machine for flow imaging, it has the potential to make enormous advances in multiphase flow characterization.

The second area of future work is in collaboration. As was found throughout this research, there are numerous research groups with great tools and expertise in flow measurement, but they tend to have poor tools for data visualization. There are also numerous research groups who have fantastic tools and expertise in visualization, but are often working with just a few sample datasets. The most important thing that can be done moving forward is to bring these groups together. Multiphase flow characterization will only be able to reach its maximum potential if both the instrumentation and the visualization of the flows is maximized. To that end, it is hoped that this dissertation is one step in the right direction.

REFERENCES

Addis, K. A., Hopper, K. D., Iyriboz, T. A., Liu, Y., Wise, S. W., Kasales, C. J., Blebea, J. S., et al. (2001). CT Angiography: In Vitro Comparison of Five Reconstruction Methods. *American Journal of Roentgenology*, *177*(5), 1171–1176. doi:10.2214/ajr.177.5.1771171

Adrian, R. J. (1991). Particle-Imaging Techniques for Experimental Fluid Mechanics. *Annual Review of Fluid Mechanics*, *23*(1), 261–304. doi:10.1146/annurev.fl.23.010191.001401

Andersen, A. H. (1989). Algebraic Reconstruction in CT from Limited Views. *IEEE Transactions on Medical Imaging*, *8*(1), 50–55. doi:10.1109/42.20361

Andersen, A. H., and Kak, A. C. (1984). Simultaneous Algebraic Reconstruction Technique (SART): A Superior Implementation of the ART Algorithm. *Ultrasonic Imaging*, *6*(1), 81–94. doi:10.1016/0161-7346(84)90008-7

Andersson, P., Sundén, E. A., Jacobsson Svärd, S., and Sjöstrand, H. (2012). Correction for Dynamic Bias Error in Transmission Measurements of Void Fraction. *Review of Scientific Instruments*, *83*(12), 125110. doi:10.1063/1.4772704

Arons, A. B., and Peppard, M. B. (1965). Einstein's Proposal of the Photon Concept—A Translation of the Annalen der Physik Paper of 1905. *American Journal of Physics*, *33*(5), 367–374. doi:10.1119/1.1971542

Azimi, M. (2012). Skeletal Joint Smoothing White Paper. *Microsoft Developer Network*. Retrieved June 9, 2017, from https://msdn.microsoft.com/en-us/library/jj131429.aspx

Baxter, B. S., and Sorenson, J. A. (1981). Factors Affecting the Measurement of Size and CT Number in Computed Tomography. *Investigative Radiology*, *16*(4), 337–341.

Bennett, J., Grout, R., Pébay, P., Roe, D., and Thompson, D. (2009). Numerically Stable, Single-Pass, Parallel Statistics Algorithms. In *2009 IEEE International Conference on Cluster Computing and Workshops* (pp. 1–8). New Orleans, LA, USA: IEEE. doi:10.1109/CLUSTR.2009.5289161

Berger, K., Ruhl, K., Schroeder, Y., Bruemmer, C., Scholz, A., and Magnor, M. (2011). Markerless Motion Capture Using Multiple Color-Depth Sensors. In P. Eisert, J. Hornegger, and K. Polthier (Eds.), *Vision, Modeling, and Visualization (VMV 2011)* (pp. 317–324). Berlin, Germany: Eurographics. doi:10.2312/PE/VMV/VMV11/317-324

Bieberle, M., Schleicher, E., Fischer, F., Koch, D., Menz, H.-J., Mayer, H.-G., and Hampel, U. (2010). Dual-Plane Ultrafast Limited-Angle Electron Beam X-ray Tomography. *Flow Measurement and Instrumentation*, *21*(3), 233–239. doi:10.1016/j.flowmeasinst.2009.12.001

Bilderback, D. H., Elleaume, P., and Weckert, E. (2005). Review of Third and Next Generation Synchrotron Light Sources. *Journal of Physics B: Atomic, Molecular and Optical Physics*, *38*(9), S773–S797. doi:10.1088/0953-4075/38/9/022

Blaimer, M., Breuer, F., Mueller, M., Heidemann, R. M., Griswold, M. A., and Jakob, P. M. (2004). SMASH, SENSE, PILS, GRAPPA: How to Choose the Optimal Method. *Topics in Magnetic Resonance Imaging*, *15*(4), 223–236. doi:10.1097/01.rmr.0000136558.09801.dd

Blinn, J. F. (1977). Models of Light Reflection for Computer Synthesized Pictures. *ACM SIGGRAPH Computer Graphics*, *11*(2), 192–198. doi:10.1145/965141.563893

Bossavit, B., Marzo, A., Ardaiz, O., De Cerio, L. D., and Pina, A. (2014). Design Choices and Their Implications for 3D Mid-Air Manipulation Techniques. *Presence: Teleoperators and Virtual Environments*, *23*(4), 377–392. doi:10.1162/PRES_a_00207

Bottomley, P. A. (1983). Nuclear Magnetic Resonance: Beyond Physical Imaging. *IEEE Spectrum*, *20*(2), 32–38. doi:10.1109/MSPEC.1983.6369002

Bowman, D. A., Kruijff, E., LaViola, J. J., and Poupyrev, I. (2004). *3D User Interfaces: Theory and Practice*. Boston, MA, USA: Addison-Wesley Professional.

Bowman, D. A., and McMahan, R. P. (2007). Virtual Reality: How Much Immersion Is Enough? *Computer*, *40*(7), 36–43. doi:10.1109/MC.2007.257

Boyer, C., Duquenne, A.-M., and Wild, G. (2002). Measuring Techniques in Gas–Liquid and Gas–Liquid–Solid Reactors. *Chemical Engineering Science*, *57*(16), 3185–3215. doi:10.1016/S0009-2509(02)00193-8

Boyer, C. N., Holland, G. E., and Seely, J. F. (2005). Intense Nanosecond Duration Source of 10–250 keV X rays Suitable for Imaging Projectile-Induced Cavitation in Human Cadaver Tissue. *Review of Scientific Instruments*, *76*(3), 35109. doi:10.1063/1.1864792

Brooks, F. P. (1999). What's Real About Virtual Reality? *IEEE Computer Graphics and Applications*, *19*(6), 16–27. doi:10.1109/38.799723

Brown, R. G., and Hwang, P. Y. C. (1997). *Introduction to Random Signals and Applied Kalman Filtering* (3rd ed.). New York, NY, USA: John Wiley & Sons, Inc.

Bryson, S. (1996). Virtual Reality in Scientific Visualization. *Communications of the ACM*, *39*(5), 62–71. doi:10.1145/229459.229467

Bryson, S., and Levit, C. (1992). The Virtual Wind Tunnel. *IEEE Computer Graphics and Applications*, *12*(4), 25–34. doi:10.1109/38.144824

Budoff, M. J., and Gul, K. (2006). Computed Tomographic Cardiovascular Imaging. *Seminars in Ultrasound, CT, and MRI*, *27*(1), 32–41. doi:10.1053/j.sult.2005.11.004

Cartz, L. (1995). *Nondestructive Testing*. Materials Park, OH: ASM International.

Casiez, G., Roussel, N., and Vogel, D. (2012). 1€ Filter: A Simple Speed-Based Low-Pass Filter for Noisy Input in Interactive Systems. In *Proceedings of the 2012 ACM Annual Conference on Human Factors in Computing Systems - CHI '12* (pp. 2527–2530). Austin, TX, USA. doi:10.1145/2207676.2208639

Celebi, S., Aydin, A. S., Temiz, T. T., and Arici, T. (2013). Gesture Recognition Using Skeleton Data with Weighted Dynamic Time Warping. In *8th International Conference on Computer Vision Theory and Applications (VISAPP 2013)* (pp. 620–625). Barcelona, Spain.

Chaouki, J., Larachi, F., and Duduković, M. P. (1997). Noninvasive Tomographic and Velocimetric Monitoring of Multiphase Flows. *Industrial & Engineering Chemistry Research*, *36*(11), 4476–4503. doi:10.1021/ie970210t

Chen, B., Moreland, J., and Zhang, J. (2011). Human Brain Functional MRI and DTI Visualization with Virtual Reality. In *Proceedings of the ASME 2011 World Conference on Innovative Virtual Reality* (pp. 343–349). Milan, Italy: ASME. doi:10.1115/WINVR2011-5565

Chotas, H. G., Dobbins, J. T., and Ravin, C. E. (1999). Principles of Digital Radiography with Large-Area, Electronically Readable Detectors: A Review of the Basics. *Radiology*, *210*(3), 595–599.

Compton, A. H. (1923). A Quantum Theory of the Scattering of X-rays by Light Elements. *Physical Review*, *21*(5), 483–502. doi:10.1103/PhysRev.21.483

Cormack, A. M. (1963). Representation of a Function by Its Line Integrals, with Some Radiological Applications. *Journal of Applied Physics*, *34*(9), 2722–2727. doi:10.1063/1.1729798

Cormack, A. M. (1964). Representation of a Function by Its Line Integrals, with Some Radiological Applications. II. *Journal of Applied Physics*, *35*(10), 2908–2913. doi:10.1063/1.1713127

Cruz-Neira, C., Leigh, J., Papka, M., Barnes, C., Cohen, S. M., Das, S., Engelmann, R., et al. (1993a). Scientists in Wonderland: A Report on Visualization Applications in the CAVE Virtual Reality Environment. In *Proceedings of IEEE 1993 Symposium on Research Frontiers in Virtual Reality* (pp. 59–66). San Jose, CA, USA: IEEE. doi:10.1109/VRAIS.1993.378262

Cruz-Neira, C., Sandin, D. J., and DeFanti, T. A. (1993b). Surround-Screen Projection-Based Virtual Reality: The Design and Implementation of the CAVE. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '93)* (pp. 135–142). ACM. doi:10.1145/166117.166134

Demiralp, C., Jackson, C. D., Karelitz, D. B., Zhang, S., and Laidlaw, D. H. (2006). CAVE and Fishtank Virtual-Reality Displays: A Qualitative and Quantitative Comparison. *IEEE Transactions on Visualization and Computer Graphics*, *12*(3), 323–330. doi:10.1109/TVCG.2006.42

Dickin, F. J., Williams, R. A., and Beck, M. S. (1993). Determination of Composition and Motion of Multicomponent Mixtures in Process Vessels Using Electrical Impedance Tomography-I. Principles and Process Engineering Applications. *Chemical Engineering Science*, *48*(10), 1883–1897. doi:10.1016/0009-2509(93)80358-W

Doering, E. R. (1992). *Three-Dimensional Flaw Reconstruction Using a Real-Time X-ray Imaging System*. Iowa State University.

Drake, J. B. (2011). *Hydrodynamic Characterization of 3D Fluidized Beds using Noninvasive Techniques*. PhD Dissertation, Mechanical Engineering, Iowa State University, Ames, IA.

Drake, J. B., and Heindel, T. J. (2011). The Repeatability and Uniformity of 3D Fluidized Beds. *Powder Technology*, *213*(1–3), 148–154. doi:10.1016/j.powtec.2011.07.027

Drake, J. B., and Heindel, T. J. (2012a). Comparisons of Annular Hydrodynamic Structures in 3D Fluidized Beds Using X-ray Computed Tomography Imaging. *ASME Journal of Fluids Engineering*, *134*(8), 81305. doi:10.1115/1.4007119

Drake, J. B., and Heindel, T. J. (2012b). Local Time-Average Gas Holdup Comparisons in Cold Flow Fluidized Beds with Side-Air Injection. *Chemical Engineering Science*, *68*(1), 157–165. doi:10.1016/j.ces.2011.09.023

Drake, J. B., Kenney, A. L., Morgan, T. B., and Heindel, T. J. (2011). Developing Tracer Particles for X-ray Particle Tracking Velocimetry. In *Proceedings of the ASME-JSME-KSME Joint Fluids Engineering Conference* (pp. 2685–2692). Hamamatsu, Shizuoka, Japan. doi:10.1115/AJK2011-11009

Drake, J. B., Tang, L., and Heindel, T. J. (2009). X-ray Particle Tracking Velocimetry in Fluidized Beds. In *ASME 2009 Fluids Engineering Division Summer Meeting (FEDSM2009)* (p. Paper FEDSM2009-78150). Vail, CO, USA: ASME Press. doi:10.1115/FEDSM2009-78150

Duncan, T. J., and Vance, J. M. (2007). Development of a Virtual Environment for Interactive Interrogation of Computational Mixing Data. *Journal of Mechanical Design*, *129*(3), 361–367. doi:10.1115/1.2409314

Ehrichs, E. E., Jaeger, H. M., Karczmar, G. S., Knight, J. B., Kuperman, V. Y., and Nagel, S. R. (1995). Granular Convection Observed by Magnetic Resonance Imaging. *Science*, *267*(5204), 1632–1634. doi:10.1126/science.267.5204.1632

Einstein, A. (1905). Über einen die Erzeugung und Verwandlung des Lichtes betreffenden heuristischen Gesichtspunkt. *Annalen Der Physik*, *322*(6), 132–148. doi:10.1002/andp.19053220607

Elsinga, G. E., Scarano, F., Wieneke, B., and Oudheusden, B. W. (2006). Tomographic Particle Image Velocimetry. *Experiments in Fluids*, *41*(6), 933–947. doi:10.1007/s00348-006-0212-z

Engel, K., Hadwger, M., Kniss, J., Rezk-Salama, C., and Weiskopf, D. (2006). *Real-Time Volume Graphics*. Wellesley, MA, USA: AK Peters Ltd.

Epstein, C. L. (2003). *Introduction to the Mathematics of Medical Imaging*. Upper Saddle River, NJ, USA: Pearson Education.

Escudero, D. R., and Heindel, T. J. (2011). Bed Height and Material Density Effects on Fluidized Bed Hydrodynamics. *Chemical Engineering Science*, *66*(16), 3648–3655. doi:10.1016/j.ces.2011.04.036

Escudero, D. R., and Heindel, T. J. (2015). Characterizing Jetting in an Acoustic Fluidized Bed Using X-ray Computed Tomography. *Journal of Fluids Engineering*, *138*(4), 41309. doi:10.1115/1.4031681

Faion, F., Friedberger, S., Zea, A., and Hanebeck, U. D. (2012). Intelligent Sensor-Scheduling for Multi-Kinect-Tracking. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 3993–3999). Vilamoura, Algarve, Portugal: IEEE. doi:10.1109/IROS.2012.6386007

Fazel, R., Krumholz, H. M., Wang, Y., Ross, J. S., Chen, J., Ting, H. H., Shah, N. D., et al. (2009). Exposure to Low-Dose Ionizing Radiation from Medical Imaging Procedures. *New England Journal of Medicine*, *361*(9), 849–857. doi:10.1056/NEJMoa0901249

Feldkamp, L. A., Davis, L. C., and Kress, J. W. (1984). Practical Cone-Beam Algorithm. *Journal of the Optical Society of America A*, *1*(6), 612. doi:10.1364/JOSAA.1.000612

Fischer, F., Hoppe, D., Schleicher, E., Mattausch, G., Flaske, H., Bartel, R., and Hampel, U. (2008). An Ultra Fast Electron Beam X-ray Tomography Scanner. *Measurement Science and Technology*, *19*(9), 94002. doi:10.1088/0957-0233/19/9/094002

Fitts, P. M. (1954). The Information Capacity of the Human Motor System in Controlling the Amplitude of Movement. *Journal of Experimental Psychology*, *47*(6), 381–391. doi:10.1037/h0055392

Fixed Function Pipeline. (2012). *OpenGL Wiki*. Retrieved January 1, 2014, from http://www.opengl.org/wiki/Fixed_Function_Pipeline

Fothergill, S., Mentis, H. M., Kohli, P., and Nowozin, S. (2012). Instructing People for Training Gestural Interactive Systems. In *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems - CHI '12* (pp. 1737–1746). Austin, TX, USA: ACM Press. doi:10.1145/2207676.2208303

Franka, N. P., and Heindel, T. J. (2009). Local Time-Averaged Gas Holdup in a Fluidized Bed with Side Air Injection using X-ray Computed Tomography. *Powder Technology*, *193*(1), 69–78. doi:10.1016/j.powtec.2009.02.008

Fu, C.-W., Goh, W.-B., and Ng, J. A. (2010). Multi-Touch Techniques for Exploring Large-Scale 3D Astrophysical Simulations. In *Proceedings of the 2010 SIGCHI Conference on Human Factors in Computing Systems (CHI '10)* (pp. 2213–2222). Atlanta, GA, USA. doi:10.1145/1753326.1753661

Fukushima, E. (1999). Nuclear Magnetic Resonance as a Tool to Study Flow. *Annual Review of Fluid Mechanics*, *31*(1), 95–123. doi:10.1146/annurev.fluid.31.1.95

Gallo, L., Placitelli, A. P., and Ciampi, M. (2011). Controller-Free Exploration of Medical Image Data: Experiencing the Kinect. In M. Olive and T. Solomonides (Eds.), *2011 24th International Symposium on Computer-Based Medical Systems (CBMS)* (pp. 1–6). Bristol, UK: IEEE. doi:10.1109/CBMS.2011.5999138

Gobbetti, E., Marton, F., and Guitián, J. A. I. (2008). A Single-Pass GPU Ray Casting Framework for Interactive Out-of-Core Rendering of Massive Volumetric Datasets. *The Visual Computer*, *24*(7–9), 797–806. doi:10.1007/s00371-008-0261-9

Goodsitt, M. M., Chan, H.-P., Way, T. W., Larson, S. C., Christodoulou, E. G., and Kim, J. (2006). Accuracy of the CT Numbers of Simulated Lung Nodules Imaged with Multi-Detector CT Scanners. *Medical Physics*, *33*(8), 3006–3017. doi:10.1118/1.2219332

Gore, J. C., Emery, E. W., Orr, J. S., and Doyle, F. H. (1981). Medical Nuclear Magnetic Resonance Imaging: I. Physical Principles. *Investigative Radiology*, *16*(4), 269–274.

Grady, D. E., and Kipp, M. E. (1994). Experimental and Computational Simulation of the High Velocity Impact of Copper Spheres on Steel Plates. *International Journal of Impact Engineering*, *15*(5), 645–660. doi:10.1016/0734-743X(94)90144-A

Groell, R., Rienmueller, R., Schaffler, G. J., Portugaller, H. R., Graif, E., and Willfurth, P. (2000). CT Number Variations Due to Different Image Acquisition and Reconstruction Parameters: A Thorax Phantom Study. *Computerized Medical Imaging and Graphics*, *24*(2), 53–58. doi:10.1016/S0895-6111(99)00043-9

Gross, M., Würmlin, S., Naef, M., Lamboray, E., Spagno, C., Kunz, A., Koller-Meier, E., et al. (2003). blue-c: A Spatially Immersive Display and 3D Video Portal for Telepresence. *ACM Transactions on Graphics (TOG) - Proceedings of ACM SIGGRAPH 2003*, *22*(3), 819–827. doi:10.1145/882262.882350

Gruner, S. M., Tate, M. W., and Eikenberry, E. F. (2002). Charge-Coupled Device Area X-ray Detectors. *Review of Scientific Instruments*, *73*(8), 2815. doi:10.1063/1.1488674

Haaker, P., Klotz, E., Koppe, R., and Linde, R. (1988). Real-Time Distortion Correction Of Digital X-ray II/TV Systems: An Application Example. In *Image Processing II* (Vol. 1027, pp. 261–266). Hamburg, Germany: SPIE. doi:10.1117/12.950292

Hansen, C. D., and Johnson, C. R. (Eds.). (2005). *The Visualization Handbook*. Burlington, MA, USA: Elsevier Academic Press.

Hargrave, P. J. (1989). A Tutorial Introduction to Kalman Filtering. In *IEE Colloquium on "Kalman Filters: Introduction, Applications and Future Developments"* (pp. 1–6). London, UK: IET.

Haubner, M., Krapichler, C., Losch, A., Englmeier, K.-H., and Van Eimeren, W. (1997). Virtual Reality in Medicine-Computer Graphics and Interaction Techniques. *IEEE Transactions on Information Technology in Biomedicine*, *1*(1), 61–72. doi:10.1109/4233.594047

He, C., Lewis, A., and Jo, J. (2007). A Novel Human Computer Interaction Paradigm for Volume Visualization in Projection-Based Virtual Environments. In *Smart Graphics* (pp. 49–60). Kyoto, Japan: Springer. doi:10.1007/978-3-540-73214-3_5

Heindel, T. J. (2011). A Review of X-ray Flow Visualization with Applications to Multiphase Flows. *ASME Journal of Fluids Engineering*, *133*(7), 74001. doi:10.1115/1.4004367

Heindel, T. J., Gray, J. N., and Jensen, T. C. (2008). An X-ray System for Visualizing Fluid Flows. *Flow Measurement and Instrumentation*, *19*(2), 67–78. doi:10.1016/j.flowmeasinst.2007.09.003

Heindel, T. J., and Monefeldt, J. L. (1997). Flash X-ray Radiography for Visualizing Gas Flows in Opaque Liquid/Fiber Suspensions. In *6th International Symposium on Gas-Liquid Two-Phase Flows*. Vancouver, BC, Canada.

Heindel, T. J., and Monefeldt, J. L. (1998). Observations of the Bubble Dynamics in a Pulp Suspension Using Flash X-ray Radiography. *TAPPI Journal*, *81*(11), 149–158.

History of OpenGL. (2013). *OpenGL Wiki*. Retrieved June 30, 2014, from http://www.opengl.org/wiki/History_of_OpenGL

Hopper, K. D., Iyriboz, A. T., Wise, S. W., Neuman, J. D., Mauger, D. T., and Kasales, C. J. (2000). Mucosal Detail at CT Virtual Reality: Surface versus Volume Rendering. *Radiology*, *214*(2), 517–522. doi:10.1148/radiology.214.2.r00fe34517

Hounsfield, G. N. (1976). Apparatus for Examining a Body by Radiation such as X or Gamma Radiation. *Patent No. 3,944,833*. United States of America.

Hsieh, J. (2009). *Computed Tomography: Principles, Design, Artifacts, and Recent Advances* (2nd ed.). Bellingham, WA, USA: SPIE. doi:10.1117/3.817303

Hu, B., Stewart, C., Hale, C. P., Lawrence, C. J., Hall, A. R. W., Zwiens, H., and Hewitt, G. F. (2005). Development of an X-ray Computed Tomography (CT) System with Sparse Sources: Application to Three-Phase Pipe Flow Visualization. *Experiments in Fluids*, *39*(4), 667–678. doi:10.1007/s00348-005-1008-2

Hubers, J. L. (2005). *An X-ray Visualization Facility for Large-Scale Multiphase Flows*. Iowa State University.

Hwu, Y., Tsai, W.-L., Groso, A., Margaritondo, G., and Je, J. H. (2002). Coherence-Enhanced Synchrotron Radiology: Simple Theory and Practical Applications. *Journal of Physics D: Applied Physics*, *35*(13), R105–R120. doi:10.1088/0022-3727/35/13/201

Ikeda, T., Kotani, K., Maeda, Y., and Kohno, H. (1983). Preliminary Study on Application of X-ray CT Scanner to Measurement of Void Fractions in Steady State Two-Phase Flows. *Journal of Nuclear Science and Technology*, *20*(1), 1–12. doi:10.1080/18811248.1983.9733354

Jain, N., Ottino, J. M., and Lueptow, R. M. (2002). An Experimental Study of the Flowing Granular Layer in a Rotating Tumbler. *Physics of Fluids*, *14*(2), 572–582. doi:10.1063/1.1431244

Juhnke, B., Berron, M., Philip, A., Williams, J., Holub, J., and Winer, E. (2013). Comparing the Microsoft Kinect to a Traditional Mouse for Adjusting the Viewed Tissue Densities of Three-Dimensional Anatomical Structures. In C. K. Abbey and C. R. Mello-Thomas (Eds.), *Medical Imaging 2013: Image Perception, Observer Performance, and Technology Assessment* (Vol. 8673, p. 86731M). Lake Buena Vista, FL, USA: SPIE. doi:10.1117/12.2006994

Ketcham, R. A., and Carlson, W. D. (2001). Acquisition, Optimization and Interpretation of X-ray Computed Tomographic Imagery: Applications to the Geosciences. *Computers & Geosciences*, *27*(4), 381–400. doi:10.1016/S0098-3004(00)00116-3

Khronos Group. (2012). OpenGL. In *SIGGRAPH 2012 - Birds of a Feather*. Los Angeles, CA, USA.

Kim, J.-S., Gračanin, D., Matković, K., and Quek, F. (2009). iPhone/iPod Touch as Input Devices for Navigation in Immersive Virtual Environments. In *IEEE Virtual Reality 2009* (pp. 261–262). Lafayette, LA, USA. doi:10.1109/VR.2009.4811045

Kim, K.-J. (1989). Characteristics of Synchrotron Radiation. In *AIP Conference Proceedings* (Vol. 184, pp. 565–632). Batavia, IL, USA: AIP. doi:10.1063/1.38046

Kingston, T. A., and Heindel, T. J. (2014). Optical Visualization and Composition Analysis to Quantify Continuous Granular Mixing Processes. *Powder Technology*, *262*, 257–264. doi:10.1016/j.powtec.2014.04.071

Kingston, T. A., Morgan, T. B., Geick, T. A., Robinson, T. R., and Heindel, T. J. (2014). A Cone-Beam Compensated Back-Projection Algorithm for X-ray Particle Tracking Velocimetry. *Flow Measurement and Instrumentation*, *39*, 64–75. doi:10.1016/j.flowmeasinst.2014.06.002

Knoll, A. M., Hijazi, Y., Westerteiger, R., Schott, M., Hansen, C. D., and Hagen, H. (2009). Volume Ray Casting with Peak Finding and Differential Sampling. *IEEE Transactions on Visualization and Computer Graphics*, *15*(6), 1571–8. doi:10.1109/TVCG.2009.204

Kramer, D. M., Kaufman, L., Guzman, R. J., and Hawryszko, C. (1990). A General Algorithm for Oblique Image Reconstruction. *IEEE Computer Graphics and Applications*, *10*(2), 62–65. doi:10.1109/38.50674

Kratz, S., and Rohs, M. (2010). A $3 Gesture Recognizer - Simple Gesture Recognition for Devices Equipped with 3D Acceleration Sensors. In *Proceedings of the 15th international conference on Intelligent user interfaces - IUI '10* (p. 341). Hong Kong, China: ACM Press. doi:10.1145/1719970.1720026

Kristensson, P. O., Nicholson, T. E. W., and Quigley, A. (2012). Continuous Recognition of One-Handed and Two-Handed Gestures Using 3D Full-Body Motion Tracking Sensors. In *Proceedings of the 2012 ACM international conference on Intelligent User Interfaces - IUI '12* (p. 89). Lisbon, Portugal: ACM Press. doi:10.1145/2166966.2166983

Krüger, J., and Westermann, R. (2003). Acceleration Techniques for GPU-Based Volume Rendering. In *Proceedings of the 14th IEEE Visualization 2003 (VIS'03)* (pp. 287–292). Seattle, Washington, USA: IEEE. doi:10.1109/VISUAL.2003.1250384

Krum, D. M., Phan, T., Dukes, L. C., Wang, P., and Bolas, M. (2014). A Demonstration of Tablet-Based Interaction Panels for Immersive Environments. In *IEEE Virtual Reality* (pp. 175–176). Minneapolis, MN, USA: IEEE. doi:10.1109/VR.2014.6802108

Kuntz, S. (2015). MiddleVR: A Generic VR Toolbox. In *2015 IEEE Virtual Reality Conference (VR 2015)* (pp. 391–392). Arles, France. doi:10.1109/VR.2015.7223460

Lacroute, P., and Levoy, M. (1994). Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques - SIGGRAPH '94* (pp. 451–458). Orlando, FL, USA: ACM Press. doi:10.1145/192161.192283

Lee, H.-K., and Kim, J. H. (1999). An HMM-Based Threshold Model Approach for Gesture Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *21*(10), 961–973. doi:10.1109/34.799904

Lee, S.-J., and Kim, G. B. (2005). Synchrotron Microimaging Technique for Measuring the Velocity Fields of Real Blood Flows. *Journal of Applied Physics*, *97*(6), 64701. doi:10.1063/1.1851596

Levi, C., Gray, J. E., McCullough, E. C., and Hattery, R. R. (1982). The Unreliability of CT Numbers as Absolute Values. *American Journal of Roentgenology*, *139*(3), 443–447. doi:10.2214/ajr.139.3.443

Levoy, M. (1988). Display of Surfaces from Volume Data. *IEEE Computer Graphics and Applications*, *8*(3), 29–37. doi:10.1109/38.511

Li, S., Pathirana, P. N., and Caelli, T. (2014). Multi-Kinect Skeleton Fusion for Physical Rehabilitation Monitoring. In *36th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)* (Vol. 2014, pp. 5060–5063). Chicago, IL, USA. doi:10.1109/EMBC.2014.6944762

Liu, J. H., Sun, Y., Kang, Z. C., Tian, X. H., Li, X., Yuan, Q. H., and Zhang, Y. (2011). Post-Processing Techniques for Aortic Computed Tomography Angiography. In *Proceedings of 2011 International Conference on Computer Science and Network Technology* (pp. 1186–1189). Harbin, China: IEEE. doi:10.1109/ICCSNT.2011.6182171

Livingston, M. A., Sebastian, J., Ai, Z., and Decker, J. W. (2012). Performance Measurements for the Microsoft Kinect Skeleton. In *2012 IEEE Virtual Reality (VR)* (pp. 119–120). Costa Mesa, CA: IEEE. doi:10.1109/VR.2012.6180911

Lorensen, W. E., and Cline, H. E. (1987). Marching Cubes: A High Resolution 3D Surface Construction Algorithm. *ACM SIGGRAPH Computer Graphics*, *21*(4), 163–169. doi:10.1145/37402.37422

Lucy, L. B. (1974). An Iterative Technique for the Rectification of Observed Distributions. *The Astronomical Journal*, *79*, 745. doi:10.1086/111605

Lun, R., and Zhao, W. (2015). A Survey of Applications and Human Motion Recognition with Microsoft Kinect. *International Journal of Pattern Recognition and Artificial Intelligence*, *29*(January), 49. doi:10.1142/S0218001415550083

Luna, F. D. (2008). *Introduction to 3D Game Programming with DirectX 10*. Sudbury, MA, USA: Wordware Publishing, Inc.

Lux, C., and Fröhlich, B. (2009). GPU-Based Ray Casting of Multiple Multi-Resolution Volume Datasets. In *2009 International Symposium on Visual Computing (IVSC '09)* (pp. 104–116). Las Vegas, NV, USA: Springer. doi:10.1007/978-3-642-10520-3_10

Macovski, A. (1983). *Medical Imaging Systems*. Englewood Cliffs, NJ, USA: Prentice-Hall.

MacPhee, A. G., Tate, M. W., Powell, C. F., Yue, Y., Renzi, M. J., Ercan, A., Narayanan, S., et al. (2002). X-ray Imaging of Shock Waves Generated by High-Pressure Fuel Sprays. *Science*, *295*(5558), 1261–1263. doi:10.1126/science.1068149

Maher, M. M., Kalra, M. K., Sahani, D. V., Perumpillichira, J. J., Rizzo, S., Saini, S., and Mueller, P. R. (2004). Techniques, Clinical Applications and Limitations of 3D Reconstruction in CT of the Abdomen. *Korean Journal of Radiology*, *5*(1), 55–67. doi:10.3348/kjr.2004.5.1.55

Mansfield, P. (1977). Multi-Planar Image Formation Using NMR Spin Echoes. *Journal of Physics C: Solid State Physics*, *10*(3), L55–L58. doi:10.1088/0022-3719/10/3/004

Marins, J. L., Yun, X., Bachmann, E. R., McGhee, R. B., and Zyda, M. J. (2001). An Extended Kalman Filter for Quaternion-Based Orientation Estimation Using MARG Sensors. In *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No.01CH37180)* (Vol. 4, pp. 2003–2011). Maui, HI, USA. doi:10.1109/IROS.2001.976367

Markl, M., Frydrychowicz, A., Kozerke, S., Hope, M., and Wieben, O. (2012). 4D Flow MRI. *Journal of Magnetic Resonance Imaging*, *36*(5), 1015–1036. doi:10.1002/jmri.23632

Masse, J.-T., Lerasle, F., Devy, M., Monin, A., Lefebvre, O., and Mas, S. (2013). Human Motion Capture Using Data Fusion of Multiple Skeleton Data. In *International Conference on Advanced Concepts for Intelligent Vision Systems (ACIVS 2013)* (pp. 126–137). Springer. doi:10.1007/978-3-319-02895-8_12

Meissner, M., Huang, J., Bartz, D., Mueller, K., and Crawfis, R. (2000). A Practical Evaluation of Popular Volume Rendering Algorithms. In *2000 IEEE Symposium on Volume Visualization (VV 2000)* (pp. 81–90). Salt Lake City, UT, USA: IEEE. doi:10.1109/VV.2000.10009

Microsoft. (n.d.-a). D3DXMatrixLookAtRH Function. *Windows Dev Center - Desktop*. Retrieved September 9, 2014, from http://msdn.microsoft.com/en-us/library/windows/desktop/bb205343(v=vs.85).aspx

Microsoft. (n.d.-b). D3DXMatrixPerspectiveFovRH Function. *Windows Dev Center - Desktop*. Retrieved September 9, 2014, from http://msdn.microsoft.com/en-us/library/windows/desktop/bb205351(v=vs.85).aspx

Microsoft. (n.d.-c). D3DXMatrixPerspectiveOffCenterRH Function. *Windows Dev Center - Desktop*. Retrieved September 9, 2014, from http://msdn.microsoft.com/en-us/library/windows/desktop/bb205354(v=vs.85).aspx

Microsoft. (n.d.-d). Deprecated Features (Direct3D 10). *Windows Dev Center - Desktop*. Retrieved January 1, 2014, from http://msdn.microsoft.com/en-us/library/windows/desktop/cc308047.aspx

Microsoft. (n.d.-e). Graphics Pipeline. *Windows Dev Center - Desktop*. Retrieved January 1, 2014, from http://msdn.microsoft.com/en-us/library/windows/desktop/ff476882.aspx

Microsoft. (n.d.-f). Kinect Hardware. *Windows Dev Center*. Retrieved June 10, 2017, from https://developer.microsoft.com/en-us/windows/kinect/hardware

Microsoft. (n.d.-g). Kinect Studio. *Microsoft Developer Network*. Retrieved June 20, 2017, from https://msdn.microsoft.com/en-us/library/dn785306.aspx

Microsoft. (2000). Microsoft Announces Release of DirectX 8.0. *Microsoft News Center*. Retrieved January 1, 2014, from http://www.microsoft.com/en-us/news/press/2000/nov00/directxlaunchpr.aspx

Microsoft. (2012a). Joint Orientation. *Microsoft Developer Network*. Retrieved August 17, 2016, from https://msdn.microsoft.com/en-us/library/hh973073.aspx

Microsoft. (2012b). Kinect for Windows SDK. *Kinect for Windows*. Retrieved March 7, 2012, from http://www.microsoft.com/en-us/kinectforwindows/develop/overview.aspx

Microsoft. (2014a). JointType Enumeration. *Microsoft Developer Network*. Retrieved June 16, 2017, from https://msdn.microsoft.com/en-us/library/windowspreview.kinect.jointtype.aspx

Microsoft. (2014b). Kinect for Windows Sensor Components and Specifications. *Microsoft Developer Network*. Retrieved October 7, 2014, from http://msdn.microsoft.com/en-us/library/jj131033.aspx

Mine, M. R., Brooks, F. P., and Sequin, C. H. (1997). Moving Objects in Space: Exploiting Proprioception in Virtual-Environment Interaction. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques - SIGGRAPH '97* (pp. 19–26). New York, NY, USA: ACM Press. doi:10.1145/258734.258747

Möller, T., and Haines, E. (1999). *Real-Time Rendering*. Natick, MA, USA: AK Peters Ltd.

Moon, S., Park, Y., Ko, D. W., and Suh, I. H. (2016). Multiple Kinect Sensor Fusion for Human Skeleton Tracking Using Kalman Filtering. *International Journal of Advanced Robotic Systems*, *13*(2), 1–10. doi:10.5772/62415

Morato, C., Kaipa, K. N., Zhao, B., and Gupta, S. K. (2014). Toward Safe Human Robot Collaboration by Using Multiple Kinects Based Real-time Human Tracking. *Journal of Computing and Information Science in Engineering*, *14*(1), 11006. doi:10.1115/1.4025810

Morgan, T. B., and Heindel, T. J. (2010). X-ray Particle Tracking of Dense Particle Motion in a Vibration-Excited Granular Bed. In *Proceedings of the ASME 2010 International Mechanical Engineering Congress and Exposition (IMECE2010)* (pp. 1709–1717). Vancouver, BC, Canada: ASME. doi:10.1115/IMECE2010-39106

Mrvaljevic, N., and Sun, Y. (2009). Comparison Between Speaker Dependent Mode and Speaker Independent Mode for Voice Recognition. In *35th Northeast Bioengineering Conference, 2009* (pp. 1–2). Boston, MA, USA. doi:10.1109/NEBC.2009.4967804

Mudde, R. F. (2011). Bubbles in a Fluidized Bed: A Fast X-ray Scanner. *AIChE Journal*, *57*(10), 2684–2690. doi:10.1002/aic.12469

Mudde, R. F., Alles, J., and van der Hagen, T. H. J. J. (2008). Feasibility Study of a Time-Resolving X-ray Tomographic System. *Measurement Science and Technology*, *19*(8), 85501. doi:10.1088/0957-0233/19/8/085501

Mudde, R. F., Bruneau, P. R. P., and van der Hagen, T. H. J. J. (2005). Time-Resolved γ-Densitometry Imaging within Fluidized Beds. *Industrial & Engineering Chemistry Research*, *44*, 6181–6187.

Ney, D. R., and Fishman, E. K. (1991). Editing Tools for 3D Medical Imaging. *IEEE Computer Graphics and Applications*, *11*(6), 63–71. doi:10.1109/38.103395

Ney, D. R., Fishman, E. K., Magid, D., and Kuhlman, J. E. (1989). Interactive Real-Time Multiplanar CT Imaging. *Radiology*, *170*(1), 275–276. doi:10.1148/radiology.170.1.2909110

Nishino, K., Kasagi, N., and Hirata, M. (1989). Three-Dimensional Particle Tracking Velocimetry Based on Automated Digital Image Processing. *Journal of Fluids Engineering*, *111*(4), 384. doi:10.1115/1.3243657

Noon, C. J. (2012). *A Volume Rendering Engine for Desktops, Laptops, Mobile Devices and Immersive Virtual Reality Systems using GPU-Based Volume Raycasting*. PhD Dissertation, Human Computer Interaction, Iowa State University, Ames, IA.

O'Handley, D. A., and Green, W. B. (1972). Recent Developments in Digital Image Processing at the Image Processing Laboratory at the Jet Propulsion Laboratory. *Proceedings of the IEEE*, *60*(7), 821–828. doi:10.1109/PROC.1972.8781

Otaduy, M. A., Igarashi, T., and LaViola, J. J. (2009). Interaction: Interfaces, Algorithms, and Applications. In *ACM SIGGRAPH 2009 Courses*. New Orleans, LA, USA: ACM. doi:10.1145/1667239.1667253

Pavlik, R. A., and Vance, J. M. (2010). A Modular Implementation of Wii Remote Head Tracking for Virtual Reality. In *ASME 2010 World Conference on Innovative Virtual Reality* (pp. 351–359). Ames, Iowa, USA: ASME. doi:10.1115/WINVR2010-3771

Pavlik, R. A., and Vance, J. M. (2012). VR JuggLua: A Framework for VR Applications Combining Lua, OpenSceneGraph, and VR Juggler. In *2012 5th Workshop on Software Engineering and Architectures for Realtime Interactive Systems (SEARIS)* (pp. 29–35). Singapore: IEEE. doi:10.1109/SEARIS.2012.6231166

Pavlik, R. A., Vance, J. M., and Luecke, G. R. (2013). Interacting With a Large Virtual Environment by Combining a Ground-Based Haptic Device and a Mobile Robot Base. In *Volume 2B: 33rd Computers and Information in Engineering Conference* (p. V02BT02A029). Portland, OR, USA: ASME. doi:10.1115/DETC2013-13441

Pfister, H., Lorensen, W. E., Bajaj, C., Kindlmann, G., Schroeder, W., Avila, L. S., Martin, K., et al. (2001). The Transfer Function Bake-Off. *IEEE Computer Graphics and Applications*, *21*(1), 16–22. doi:10.1109/38.920623

Phong, B. T. (1975). Illumination for Computer Generated Pictures. *Communications of the ACM*, *18*(6), 311–317. doi:10.1145/360825.360839

Piegl, L., and Tiller, W. (1997). *The NURBS Book* (2nd ed.). Berlin, Germany: Springer-Verlag.

Rabiner, L. R. (1989). A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proceedings of the IEEE*, *77*(2), 257–286. doi:10.1109/5.18626

Radon, J. (1917). Über die Bestimmung von Funktionen durch ihre Integralwerte längs gewisser Mannigfaltigkeiten. *Berichte Der Sächsischen Akadmie Der Wissenschaft*, *69*, 262–277.

Radon, J. (1986). On the Determination of Functions from Their Integral Values Along Certain Manifolds. *IEEE Transactions on Medical Imaging*, *5*(4), 170–176. doi:10.1109/TMI.1986.4307775

Rendering Pipeline Overview. (2012). *OpenGL Wiki*. Retrieved June 16, 2014, from http://www.opengl.org/wiki/Rendering_Pipeline_Overview

Rigoll, G., Kosmala, A., and Eickeler, S. (1997). High Performance Real-Time Gesture Recognition Using Hidden Markov Models. In *Proceedings of the International Gesture Workshop* (pp. 69–80). Bielefeld, Germany: Springer. doi:10.1007/BFb0052990

Robb, R. A. (2008). Medical Imaging and Virtual Reality: A Personal Perspective. *Virtual Reality*, *12*(4), 235–257. doi:10.1007/s10055-008-0104-z

Romero, J. B., and Smith, D. W. (1965). Flash X-ray Analysis of Fluidized Beds. *AIChE Journal*, *11*(4), 595–600.

Röntgen, W. C. (1896). On a New Kind of Rays. *Nature*, *53*(1369), 274–276. doi:10.1038/053274b0

Rowe, P. N., and Partridge, B. A. (1965). An X-ray Study of Bubbles in Fluidised Beds. *Transactions of the Institute of Chemical Engineers*, *43*, T157. doi:10.1016/S0263-8762(97)80009-3

Royer, J. R., Corwin, E. I., Flior, A., Cordero, M.-L., Rivers, M. L., Eng, P. J., and Jaeger, H. M. (2005). Formation of Granular Jets Observed by High-Speed X-ray Radiography. *Nature Physics*, *1*(3), 164–167. doi:10.1038/nphys175

Seeger, A., Affeld, K., Goubergrits, L., Wellnhofer, E., and Kertzscher, U. (2001). X-ray-Based Assessment of the Three-Dimensional Velocity of the Liquid Phase in a Bubble Column. *Experiments in Fluids*, *31*(2), 193–201. doi:10.1007/s003480100273

Seibert, J. A. (2006). Flat-Panel Detectors: How Much Better Are They? *Pediatric Radiology*, *36*(Supplement 2), 173–81. doi:10.1007/s00247-006-0208-0

Shepp, L. A., and Kruskal, J. B. (1978). Computerized Tomography: The New Medical X-ray Technology. *The American Mathematical Monthly*, *85*(6), 420–439.

Shepp, L. A., and Logan, B. F. (1974). Reconstructing Interior Head Tissue from X-ray Transmissions. *IEEE Transactions on Nuclear Science*, *21*(1), 228–236. doi:10.1109/TNS.1974.4327466

Sherman, W. R., and Craig, A. B. (2003). *Understanding Virtual Reality*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Shimada, T., Habu, H., Seike, Y., Ooya, S., Miyachi, H., and Ishikawa, M. (2007). X-ray Visualization Measurement of Slurry Flow in Solid Propellant Casting. *Flow Measurement and Instrumentation*, *18*(5–6), 235–240. doi:10.1016/j.flowmeasinst.2007.07.006

Shotton, J., Fitzgibbon, A., Cook, M., Sharp, T., Finocchio, M., Moore, R., Kipman, A., et al. (2011). Real-Time Human Pose Recognition in Parts from Single Depth Images. In *2011 IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2011)* (pp. 1297–1304). Providence, RI, USA: IEEE. doi:10.1109/CVPR.2011.5995316

Shreiner, D., Sellers, G., Kessenich, J., and Licea-Kane, B. (2013). *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 4.3* (8th ed.). Upper Saddle River, NJ, USA: Pearson Education.

Smith, J. V. (1995). Synchrotron X-ray Sources: Instrumental Characteristics. New Applications in Microanalysis, Tomography, Absorption Spectroscopy and Diffraction. *The Analyst*, *120*(5), 1231–1245. doi:10.1039/an9952001231

Stoakley, R., Conway, M. J., and Pausch, R. (1995). Virtual Reality on a WIM: Interactive Worlds in Miniature. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems - CHI '95* (pp. 265–272). Vancouver, BC, Canada. doi:10.1145/223904.223938

Striegel, A. C. (2005). *Development of Data Acquistion Software for X-ray Radiography, Computed Tomography, and Stereography of Multiphase Flow*. MS Thesis, Mechanical Engineering, Iowa State University, Ames, IA.

Suma, E. A., Krum, D. M., Lange, B., Koenig, S., Rizzo, A., and Bolas, M. (2013). Adapting User Interfaces for Gestural Interaction with the Flexible Action and Articulated Skeleton Toolkit. *Computers and Graphics*, *37*(3), 193–201. doi:10.1016/j.cag.2012.11.004

Sung, J., Ponce, C., Selman, B., and Saxena, A. (2011). Human Activity Detection from RGBD Images. In *AAAI Workshop on Plan, Activity and Intent Recognition (PAIR)* (pp. 47–55). San Francisco, CA, USA.

Takala, T. M. (2014). RUIS - A Toolkit for Developing Virtual Reality Applications with Spatial Interaction. In *Proceedings of the 2nd ACM symposium on Spatial user interaction (SUI '14)* (pp. 94–103). Honolulu, HI, USA. doi:10.1145/2659766.2659774

Takala, T. M., Rauhamaa, P., and Takala, T. (2012). Survey of 3DUI Applications and Development Challenges. In *IEEE Symposium on 3D User Interfaces 2012, 3DUI 2012 - Proceedings* (pp. 89–96). Orange County, California, USA: IEEE. doi:10.1109/3DUI.2012.6184190

Taylor, R. M. I., Hudson, T. C., Seeger, A., Weber, H., Juliano, J., and Helser, A. T. (2001). VRPN: A Device-Independent, Network-Transparent VR Peripheral System. In *VRST '01: Proceedings of the ACM Symposium on Virtual Reality Software and Technology* (pp. 55–61). Alberta, Canada. doi:10.1145/505008.505019

TuxFamily. (2017). Eigen. Retrieved June 18, 2017, from http://eigen.tuxfamily.org/index.php?title=Main_Page

van Dam, A., Forsberg, A. S., Laidlaw, D. H., LaViola, J. J., and Simpson, R. M. (2000). Immersive VR for Scientific Visualization: A Progress Report. *IEEE Computer Graphics and Applications*, *20*(6), 26–52. doi:10.1109/38.888006

van Ommen, J. R., and Mudde, R. F. (2008). Measuring the Gas-Solids Distribution in Fluidized Beds - A Review. *International Journal of Chemical Reactor Engineering*, *6*(R3), 1796. doi:10.2202/1542-6580.1796

VanderKnyff, C. (2008). VrpnNet 1.1.1. Retrieved April 20, 2012, from http://wwwx.cs.unc.edu/~chrisv/vrpnnet

Wang, B., Zhu, L., Jia, K., and Zheng, J. (2010). Accelerated Cone Beam CT Reconstruction Based on OpenCL. In *2010 International Conference on Image Analysis and Signal Processing* (pp. 291–295). IEEE. doi:10.1109/IASP.2010.5476110

Wang, Y., Liu, X., Im, K.-S., Lee, W.-K., Wang, J., Fezzaa, K., Hung, D. L. S., et al. (2008). Ultrafast X-ray Study of Dense-Liquid-Jet Flow Dynamics Using Structure-Tracking Velocimetry. *Nature Physics*, *4*(4), 305–309. doi:10.1038/nphys840

Welch, G., Bishop, G., Vicci, L., Brumback, S., Keller, K., and Colucci, D. (1999). The HiBall Tracker: High-Performance Wide-Area Tracking for Virtual and Augmented Environments. In *Proceedings of the ACM symposium on Virtual Reality Software and Technology* (pp. 1–10). London, UK. doi:10.1.1.46.5550

Welch, G. F. (2009). History: The Use of the Kalman Filter for Human Motion Tracking in Virtual Reality. *Presence: Teleoperators and Virtual Environments*, *18*(1), 72–91. doi:10.1162/pres.18.1.72

Westover, L. (1990). Footprint Evaluation for Volume Rendering. *ACM SIGGRAPH Computer Graphics*, *24*(4), 367–376. doi:10.1145/97880.97919

Whitemarsh, E. A., Escudero, D. R., and Heindel, T. J. (2016). Probe Effects on the Local Gas Holdup Conditions in a Fluidized Bed. *Powder Technology*, *294*, 191–201. doi:10.1016/j.powtec.2016.02.035

Williamson, B. M., LaViola, J. J. J., Roberts, T., and Garrity, P. (2012). Multi-Kinect Tracking for Dismounted Soldier Training. In *Proceedings of the Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC) 2012* (pp. 1727–1735). Orlando, FL, USA.

Wu, C., Cheng, Y., Ding, Y., Wei, F., and Jin, Y. (2007). A Novel X-ray Computed Tomography Method for Fast Measurement of Multiphase Flow. *Chemical Engineering Science*, *62*(16), 4325–4335. doi:10.1016/j.ces.2007.04.026

Yan, G., Tian, J., Zhu, S., Dai, Y., and Qin, C. (2008). Fast Cone-Beam CT Image Reconstruction Using GPU Hardware. *Journal of X-Ray Science and Technology*, *16*, 225–234.

Yun, X., Lizarraga, M., Bachmann, E. R., and McGhee, R. B. (2003). An Improved Quaternion-Based Kalman Filter for Real-Time Tracking of Rigid Body Orientation. In *2003 IEEE/RSJ International Conference on Intelligent Robots and Systems* (Vol. 2, pp. 1074–1079). Las Vegas, NV, USA. doi:10.1109/IROS.2003.1248787

Zhang, J. (2003). *Development of a High Resolution 3D Computed Tomography System: Data Acquisition, Reconstruction, and Visualization*. MS Thesis, Electrical and Computer Engineering, Iowa State University, Ames, IA.

Zhang, S., Demiralp, C., Keefe, D. F., DaSilva, M., Laidlaw, D. H., Greenberg, B. D., Basser, P. J., et al. (2001). An Immersive Virtual Environment for DT-MRI Volume Visualization Applications: A Case Study. In *Proceedings of Visualization 2001 (VIS '01)* (pp. 437–584). San Diego, CA, USA: IEEE. doi:10.1109/VISUAL.2001.964545

Zhang, Z. (2012). Microsoft Kinect Sensor and Its Effect. *IEEE Multimedia*, *19*(2), 4–10. doi:10.1109/MMUL.2012.24

Zink, J., Pattineo, M., and Hoxley, J. (2011). *Practical Rendering and Computation with Direct3D 11*. Boca Raton, FL: CRC Press.

Zolfaghari, A. M., Kellogg, E., Wendt, S. E., and Gray, J. N. (2002). High Speed X-ray Radiography Diagnostic of Current Interruption in Circuit Breakers. *Review of Scientific Instruments*, *73*(4), 1945. doi:10.1063/1.1461878

Zuiderveld, K. J., van Ooijen, P. M. A., Chin-A-Woeng, J. W. C., Buijs, P. C., Olree, M., and Post, F. H. (1996). Clinical Evaluation of Interactive Volume Visualization. In *Proceedings of Seventh Annual IEEE Visualization* (pp. 367–370). San Francisco, CA, USA: IEEE. doi:10.1109/VISUAL.1996.568134

Zwicker, M., Pfister, H., van Baar, J., and Gross, M. (2001). EWA Volume Splatting. In *Proceedings Visualization, 2001. VIS '01.* (pp. 29–538). San Diego, CA, USA: IEEE. doi:10.1109/VISUAL.2001.964490