A More Flexible Approach to Utilizing Depth Cameras for Hand & Touch Interaction

Thomas Butkiewicz

University of New Hampshire

Abstract—**Many researchers have utilized depth cameras for tracking user's hands to implement various interaction methods, such as touch-sensitive displays and gestural input. With the recent introduction of Microsoft's low-cost Kinect sensor, there is increased interest in this strategy. However, a review of the existing literature on these systems suggests that the majority suffer from similar limitations due to the image processing methods used to extract, segment, and relate the user's body to the environment/display. This paper presents a simple, efficient method for extracting interactions from depth images that is more flexible in terms of sensor placement, display orientation, and dependency on surface reflectivity.**

Index Terms— **Depth-camera, tracking, multi-touch, Kinect.**

I. INTRODUCTION

Multi-touch sensitive display surfaces and hand gesture-based interaction are increasingly popular interface methodologies. However, multi-touch overlays and multiple-camera motion tracking setups are still relatively expensive, limiting their adoption beyond well-funded commercial or research applications.

Microsoft recently released the low-cost Kinect device, intended to be used for tracking body movements for interaction with video games. Researchers have since repurposed the Kinect as a generic depth camera to perform many different types of hand tracking and touch-enabling of display surfaces.

The Kinect (and the PrimeSensor design it is based on) consists of a standard RGB camera, a laser-based projector that emits infrared light in a known, static dot-pattern, and a monochrome camera with an infrared pass filter that captures the reflection of the dot-pattern from the scene. The device calculates a depth image from the distortions observed in the reflected pattern.

Reviewing the methods used by other researchers to extract useful tracking information from these depth images reveals that most suffer from a few shared limitations due to assumptions or constraints in their image processing algorithms. This paper explores these limitations and presents a more flexible approach which overcomes them.

Manuscript Received on Feb1， 2012

II. RELATED WORK & LIMITATIONS

A number of researchers have designed systems utilizing depth cameras, most recently the Kinect, for hand-based interactions including emulating multi-touch displays, pose and gesture tracking. Based on the image-processing algorithms utilized, the vast majority of these systems can be classified into two categories: along-axis depth thresholding, and depth background subtraction.

In along-axis thresholding systems, the depth camera, display/ interaction plane, and user are all aligned roughly along a common axis. A good example of this setup is DepthTouch [2]. In the DepthTouch setup, the user stands in front of a vertical, transparent surface illuminated by a projector. A depth camera is placed behind the surface and pointed directly at it. The pixels in the depth image describe the distance to the camera, and due to the co-location of the display and camera along the same axis, the relative depth to the display. To segment users' interactions from the depth image, these systems use various, but similar, thresholding strategies. For example, DepthTouch discards any data too close or too far from the screen, then calculates the average depth value of the remaining pixels (claimed to be an approximation of the depth of the user's torso). This distance is used as a threshold to extract groups of pixels corresponding to the user's arms/hands. If an extracted group of pixels contains a depth value close enough to the screen, it is considered to be a touch and is processed as such. Since the camera's view vector and the screen's normal vector are roughly equal, the coordinates of touches in the depth image can be easily mapped through 2D translation and scaling to the screen coordinates. Ahn et al. [1] also extract the user's hands using depth thresholding, but it is not clear how they determine their threshold. (It appears to be fixed, thus limiting the user's position relative to the camera.)

The first and most obvious limitation of these along-axis thresholding systems is strict relative placement and orientation of the camera, display, and user. Depth cameras often have minimum working distances, for example the Kinect has a minimum distance of around 1 meter. (This has been improved significantly to around 41cm in the new "Kinect for Windows" device) These minimum working distances, and camera angle limitations, require choosing between one of two setups: the user can be close to the display, but the camera must be set back a distance behind the display, and the display must be transparent to the camera; or the camera can be in front of the display, but the user must be a significant distance from the display. In the former case, the display methodology is essentially restricted to projections on non-opaque surfaces, and space is wasted behind the display. In the latter case, the user is now physically distant and disconnected from the display surface, thus removing any real sense of physically manipulating virtual objects, as well as making accurate selection of onscreen objects difficult. In practice, this technique leads to users blindly poking the air in front of them, as it is hard to know where to touch in midair to manipulate objects displayed several meters away.

The approach presented here does not rely on this type of depth thresholding or assumptions that the camera/ interaction/display planes are on a common axis. Thus it is far more flexible in terms of the orientation and position of the camera, display, and user.

The second category of systems, those which rely on background subtraction within the depth image, are more flexible in terms of position and orientation of the camera, surface/display, and user. However, they suffer from their own unique limitations.

In these systems, the depth camera captures images of the environment and display surfaces without the user present. It records the average depth value at each pixel in the static scene and uses this to build a background image. During use, pixels where the user is present will show depth values closer to the camera than the background image. One can simply subtract the background image from the live depth image and threshold it (to remove noise) to get all pixels representing the user's body. Detection of touches or interactions with a surface is done by comparing depth values in the live depth image with those in the stored background image for that surface. This method is used in systems such as Wilson's recent design [5] and LightSpace [6], in which surfaces, such as tabletops and walls, have imagery projected onto them and are touch-enabled. Wilson [5] for example, computes a background image of per-pixel min and max depth ranges, and watches for touches by monitoring pixels to see if their depth value falls within a certain distance range of the background depth. Yoo et al. [7] use a depth camera to segment the user's body from the background, and then incorporate multiple RGB cameras to segment the hands from the body.

While background subtraction is a well-studied approach, and is reliable for RGB images (in which each pixel always receives an intensity value), in practice the depth images generated by depth cameras can exhibit idiosyncratic behaviors that limit background subtraction's effectiveness. Certain surfaces (e.g. glass and computer monitors) reflect the depth sensor's IR light poorly (or not at all, or away from the camera), and thus pixels in these areas have no depth values (or intermittent/unreliable depth values). Thus, building depth backgrounds for these surfaces is difficult or impossible. This applies to depth cameras measuring time-of-flight and those measuring distortion of structured light (e.g. Kinect); and is likely why most of these systems utilize projected displays, (as most monitor screens, by design, do not provide ideal reflectance characteristics needed for depth cameras.)

The approach presented here does not at any point need to measure the display surface (or any part of the environment). The only physical surface that the depth camera needs to be able to measure correctly is the skin of the user's hands. (Which presumably all depth cameras will do well.) Because it does not need to measure the display surfaces, any computer monitor, television, projection surface, etc. can be used, without complications from glass, anti-reflective coatings, etc.

Furthermore, background subtraction techniques perform best for surfaces that are orthogonal to the camera, and their performance degrades as the angle of the surface approaches parallel to the camera's viewing vector (the closer to parallel, the less pixels capture the surface). This presents a problem for vertical surfaces, because the camera must be placed at an angle other than orthogonal, as the user's body in front of the screen would obscure its vision of their hands. In contrast, the approach presented here will work on surfaces of any orientation relative to the camera, including vertical/parallel-to-camera surfaces, since the surface itself does not need to be measured by the camera.

III. IMPLEMENTATION DETAILS

3.1 Conversion to 3D Space

One major difference that differentiates this approach from other implementations is the conversion to, and processing of, depth camera data as a 3D point cloud. This allows for more robust segmentation and tracking, as well as greater flexibility in the placement and orientation of the sensor and display surface.

The first step involves conversion of the depth images received from the camera to 3D point clouds. We used a Microsoft Kinect sensor, which through OpenNI [3] provides a depth image with each pixel holding a depth value expressed in mm from the sensor. These values can be re-projected into 3D coordinates by knowing the horizontal and vertical viewing angles of the sensor. Each pixel in the depth image with a valid value (within sensor's working range) is transformed into a (x,y,z) point relative to the sensor. Each point's original (x,y) location in the depth image is stored to facilitate swapping to image-processing methods on sub-sets of the point cloud when advantageous to do so (e.g. connected components analysis). For depth cameras other than a Kinect, re-projection to 3D points may require other methods (e.g. camera calibration), but is generally a fairly simple operation. Figure 1 shows a raw depth image and re-projected point cloud.

3.2 Setup and Calibration

Exact placement or orientation of the depth camera relative to the display or user is not necessary. The sensor does not need to actually view the display surface itself; it only needs to see a finger-tip touching each corner of the display surface. The optimal setup for tracking hands interacting with a display is to place the sensor directly above the workspace. The sensor should be hung as low as possible (to increase area of interest resolution) while still being able to see all areas in which tracking is needed.

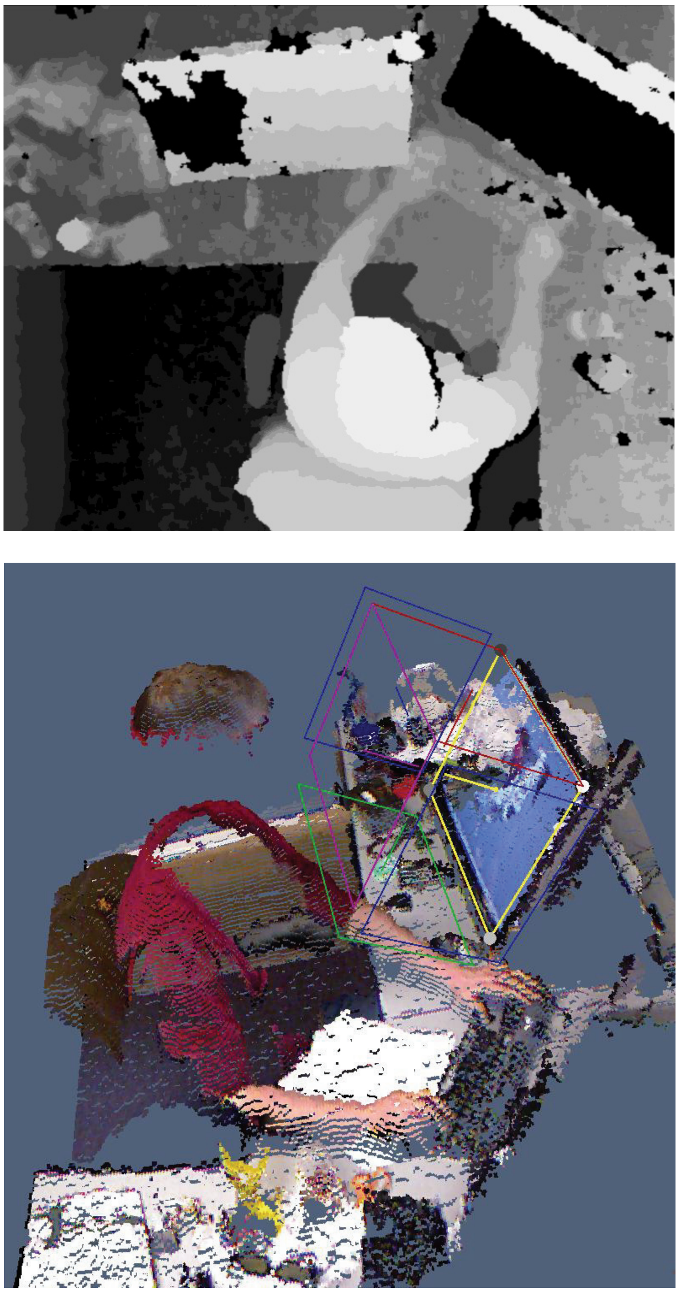

Fig. 1. (top) A raw depth image from the Kinect sensor. (bottom) A point cloud derived from the depth image, colored with data from the aligned RGB camera, and showing an interaction volume in front of a touchscreen.

Once the camera has been placed, the user must calibrate the display (or projection surface) they wish to interact with. We implemented this by showing a live RGB image from the sensor, holding a finger up to the upper right corner of the display screen, then clicking on the tip of the finger in the live image. This process is repeated for the other four corners, in a specific clockwise order.

As each corner is entered, the algorithm records the (x, y) coordinate and the depth value associated with the selected pixels. These are transformed to 3D locations in the sensor-relative coordinate system. The four corner points are averaged to find the center of the display, and then a plane fitting algorithm is used to determine the best fit plane about that central point to represent the display. The four corner points are projected onto the plane, to ensure they are coplanar. An average up vector (i.e. [[UL-LL] + [UR-LR]]/2) and average right vector are calculated, then the points are realigned relative to the center of the screen, such that they are all half of each directional vector away. This calibration results in: four coplanar, rectified corner locations, a screen plane normal, and orthogonal vectors for up and right along screen plane.

Using these values, and the display resolution, it is possible to project any points in sensor coordinates to points on the screen plane, and if within the screen boundaries, calculate a screen coordinate. This representation of the screen in sensor coordinates is also used to construct interaction volumes.

3.3 Building and Using Interaction Volumes

After calibration, an interaction volume (visible in the lower part of Figure 1) is defined in front of the display, within which hands/touches are detected and segmented for processing as touches or gestures.

The simplest method to construct an interaction volume is to extrude the display screen into a cuboid. This can be done simply by translating each of the corner points a set distance along the screen’s normal vector. The choice of distance to translate/extrude controls the depth of the interaction volume, and is directly determined by the desired interaction one wishes to capture. To emulate a multi-touch overlay on a display surface by watching for fingers entering the area directly in front of the display, only a small (~3cm) extrusion is needed. To capture the entire hand, (e.g. to get hand-pose information related to touch points,) then the screen should be extruded far more (~30cm.)

In these simple extrusion cases, the four original corners and the four extruded corners are used to calculate six planes which define the boundaries of the cuboid interaction volume.

As frames are received from the camera, they are converted to a 3D point cloud (as described in Section 3.1), and points within the interaction volume are extracted from this point cloud using these planar boundaries as efficient filters. This is done quickly by checking which side of the six planes each point falls on; if a point falls on the interior of all planes, it is added to a candidate point list. This set of points is then examined to extract the individual touches/hands present, as detailed in the next section.

This method of extracting points within a display-oriented volume-of-interest from the depth image is superior to both depth thresholding (because in 3D, the orientations of the camera, user, and display are irrelevant) and background subtraction (because the 3D interaction volume is not tied to a physical surface, and is not affected by changes in the environment, such as a coffee mug being placed on the desk below a monitor).

3.4 Finding Hands and Touches

The final step is taking the collection of candidate points and extracting the touches/hands of interest. This process is highly dependent on the exact interactions one wishes to extract. This section provides some practical examples.

3.4.1 Emulating a Multi-touch Display

This technique can be used to emulate a multi-touch display or extend touch detection behavior to arbitrary flat surfaces (such as a desktop), or even a plane in midair.

A thin interaction volume is defined by extruding the surface a few cm. When a finger is placed on the surface, the tip will enter the interaction volume and the candidate points list will contain points corresponding to that finger tip. These candidate points are then converted to one or more touch points in screen coordinates.

This can be done by putting the candidate points back into image space (recall that original depth image coordinates were stored), forming a binary image. Noise is filtered out with morphological opening (erosion then dilation). Connected components analysis is then used to identify the different “blobs” of points. (There will be one for each fingertip entering the interaction volume.)

For each 2D blob, the 3D points comprising the blob are checked to see which is closest to the screen plane. This point will correspond to the tip of the finger. This point’s 3D location is projected onto the screen plane, giving its screen coordinate in the sensor’s coordinate system. Then, knowing the resolution of the screen, it is trivial to convert into a (x, y) screen coordinate. This coordinate can then be directly used by an application or sent through a touch handling system such as TUIO [3].

3.4.2 Enhancing Multi-touch Surfaces

Emulating multi-touch with vision-based methods, while useful, is not nearly as accurate and reliable as using a well-made multi-touch display/overlay. While multi-touch displays/overlays can provide accurate finger touch locations with no occlusion issues, they do not generally provide information beyond the location of touch-points, such as which touch points belong to the same hand, or which hand (left vs. right) they originated from. This technique can extended to derive this highly-useful information about the touches being detected by a multi-touch screen.

First, an interaction volume in front of the screen is constructed as described previously, however now the planes defining the box are as follows: The front is the screen’s plane, the back is the screen plane translated along its normal by a distance depending on what exactly one wishes to determine (e.g. which touch points share hands, or right vs. left). This can be varied to from just deep enough to get the whole hand (~15cm) or extended to capture portions of the arms as well. (25cm seems to work well for left/right differentiation.) The other four sides are determined by extruding the edges of the screen, and then translating them outwards from the center (e.g. left edge moves 10cm along screen-right vector). This allows for the detection of parts of the hands that would naturally fall outside the extruded screen when touching near the edges.

Whenever a finger-down event is received from the multi-touch screen, a frame is pulled from the Kinect sensor and converted to a 3D point cloud. As in the previous section, any points that are within the interaction volume are extracted. These candidate points represent any objects (hands/arms) within the interaction volume when the touch was made.

To search for the hand associated with the touch event, the touch point’s original (x, y) location in screen coordinates is transformed to 3D sensor-coordinates. The candidate point closest to that touch-point must then be located.

A binary image is created, containing candidate points in image space. Noise is filtered with morphological opening. Connected components analysis is used to segment all objects in the image (there can be multiple hands or other objects in the interaction volume.) The object which either contains, or is closest to the (x, y) location of, the candidate point closest to the touch point (fingertip) is then extracted, representing the touching hand.

The set of candidate points which have original (x,y) image coordinates matching the binary mask of the selected object make up the hand/forearm associated with the touch point. An example of this result shown in Figure 2.

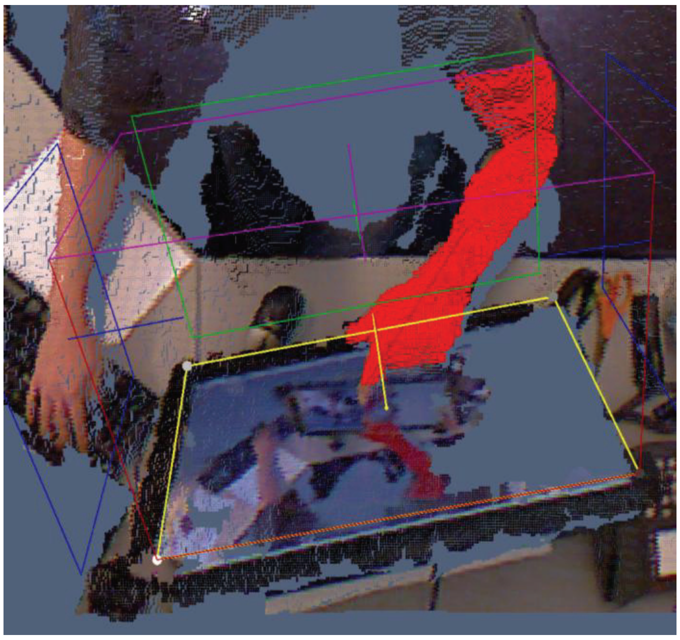
Fig. 2. A subset (red) of the point cloud representing the user’s forearm extracted using the interaction volume.

From these points, there are various ways to determine which hand (left vs. right) it is. One simple method that seems to work sufficiently is to find the points’ average 3D location (a rough estimation of the hand/forearm’s center-of-mass) and then compute the vector from the touch point to this center point. If the dot product of this vector and the screen-right vector is positive, the hand/forearm is approaching the touch point from the right and is thus likely the right hand; if negative, the left.

3.4.3 Other uses

Similar to the multi-touch emulation example, hover behaviors can be enabled by watching for fingers approaching the screen, and using the detection of such as a cue for an action such as popping up a menu at that location.

As shown in Figure 3, head tracking for improved stereoscopic rendering can be accomplished by isolating a second volume away from, but relative to a display, finding the top of the head, and extrapolating estimated eye positions from the gaze direction.

The user's hands can be reconstructed in 3D, to serve as highly flexible virtual interaction devices. This can be accomplished by segmenting out the user's arms/hands, building 3D meshes (using Delaunay triangulation) from the corresponding subsets of the point cloud, and finally texturing these meshes with the imagery from the color camera. The segmentation step is the most challenging, as one must not only segment foreground from background, but also prevent overlapping body parts from being connected together during triangulation. As visible in Figure 4, this can be done by checking the depth changes between adjacent foreground pixels (white) in the depth image, and adding triangulation barriers (grey) at the depth-jumps between neighboring regions.

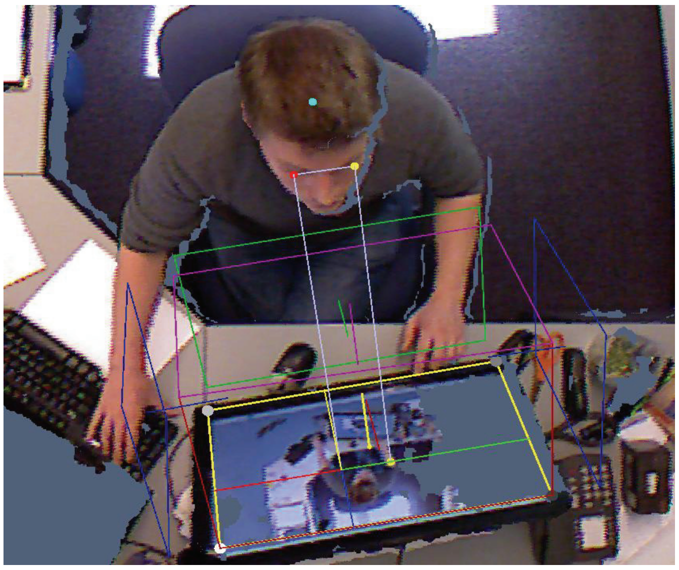
Fig. 3. Head tracking for stereoscopy using the Kinect.

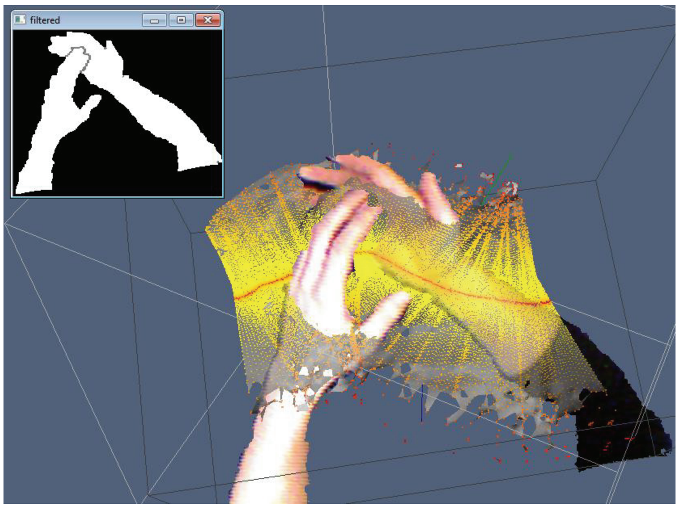

Fig. 4. A sonar data cleaning application that employs the user's hands as virtual tools. (upper right) The underlying segmentation between overlapping hands, which are individually triangulated and projected into the 3D workspace.

IV. CONCLUSION

It is obvious that depth cameras, such as the Kinect, can be extremely effective at capturing user's hand-based interactions. However, when utilizing depth cameras for these purposes, one should not rely solely on extending traditional 2D image processing methods such as thresholding and background subtraction, but instead design new algorithms, such as those proposed here, which take full advantage of the 3D nature of the depth measurements.

REFERENCES

[1] Y.-K. Ahn, Y.-C. Park, K.-S. Choi, W.-C. Park, H.-M. Seo, and K.-M. Jung, "3D spatial touch system based on time-of-flight camera", *WSEAS Trans. Info. Sci. and App*. vol. 6, issue 9, p. 1433-1442, 2009.

[2] H. Benko and A. D. Wilson, "DepthTouch: Using Depth-Sensing Camera to Enable Freehand Interactions On and Above the Interactive Surface", *Technical Report MSR-TR-2009-23*, Microsoft Research, 2009.

[3] M. Kaltenbrunner, T. Bovermann, R. Bencina, and E. Costanza, "TUIO - A Protocol for Table Based Tangible User Interfaces", 6th International *Workshop on Gesture in HCI and Simulation*, Vannes, France, 2005.

[4] OpenNI, "http://www.openni.org/", April 2011.

[5] A. D. Wilson, "Using a depth camera as a touch sensor", in *ACM Internat. Conf. on Interactive Tabletops and Surfaces (ITS '10)*, ACM, New York, NY, USA, p. 69-72, 2010.

[6] A. D. Wilson and H. Benko, "Combining multiple depth cameras and projectors for interactions on, above and between surfaces", *UIST '10*, ACM, New York, NY, USA, p. 273-282, 2010.

[7] B. Yoo, J.-J. Han, C. Choi, K. Yi, S. Suh, D. Park, and C. Kim, "3D user interface combining gaze and hand gestures for large-scale display", *CHI EA '10*, ACM, New York, NY, USA, p. 3709-3714, 2010.

Thomas Butkiewicz received a B.S. in Computer Science from Ithaca College in 2005, where he focused on computer graphics and virtual reality research. He then developed new interactive geospatial visualization techniques throughout graduate school at The University of North Carolina at Charlotte, receiving a Masters in Computer Science in 2007 and a Ph.D. in Computer Science in 2010. In 2011 he joined the The Center for Coastal and Ocean Mapping at The University of New Hampshire as a post doc, and is continuing his research there in a faculty research position.

Dr. Thomas Butkiewicz specializes in creating highly interactive visualizations, which allow users to perform complex visual analysis on geospatial datasets through unique, intuitive exploratory techniques. His research interests also include multi-touch and natural interfaces, virtual reality, stereoscopic displays, and image processing/computer vision. His current research centers around interactive, exploratory visual analysis for large 4D ocean flow simulations.