Probabilistic Algorithms, Integration, and Empirical Evaluation for Disambiguating Multiple Selections in Frustum-Based Pointing

Greg Schmidt [1,2] Dennis G. Brown [1] Erik B. Tomlin [1] J. Edward Swan II [1,3] Yohan Baillot [1,2]
[1]3D Virtual & Mixed Environments Lab, U.S. Naval Research Laboratory
[2]ITT Industries, Advanced Engineering & Sciences
[3]Department of Computer Science & Engineering, Mississippi State University

Abstract— **There are a few fundamental pointing-based user interface techniques for performing selections in 3D environments. One of these techniques is ray pointing, which makes selections by determining intersections of objects along a single ray cast from the user. This technique is susceptible to inaccuracy of the user interface tracking technology and user noise (e.g., jittery hand, see actual viewing ray in Figure 1 (bottom) and how it misses the object when error is factored in). Another technique is the frustum-based (or equivalent cone-based) approach which casts a frustum from the user into the 3D scene. Selections are made by performing intersections of objects with the frustum. This technique resolves the imprecision associated with the ray pointing technique (see how the lightly shaded cone in Figure 1 (bottom) still intersects the object), but may produce multiple ambiguous selections as shown in Figure 2.**

This paper presents probabilistic selection algorithms and integration schemes that address some of the ambiguities associated with the frustum-based selection technique. The selection algorithms assign probabilities that the user has selected particular objects using a set of low-level 3D intersection-based selection techniques and the relationship of the objects in a hierarchical database. Final selections are generated by integating the outputs from each selection algorithm using one of several weighting schemes. We implemented the selection algoithms and weighting schemes within our distributed augmented reality (AR) and virtual reality (VR) architecture. Next, we performed several experiments to empirically evaluate the probabilistic selection techniques and integration schemes. Our results show that the combined selection and integration algorithms are effective at disambiguating multiple selections. This paper thoroughly describes the architecture, experiments, and results.

Index Terms— **interaction, selection, algorithms, augmented reality, virtual reality, hierarchical databases**

I. Introduction

Many virtual and augmented reality systems present the user with a rendering of a 3D world containing distinct objects that the user can query or manipulate. To perform these actions on objects, the user usually must first select the object. While there are many ways to select objects, pointing at the desired object is common and natural. Selection by pointing can happen using a range of devices, from a common 2D mouse controlling a cursor on a 2D projection of the 3D world, to a full six-degree-of-freedom (DOF) hand-held tracking device. Selection can also happen without using the hands at all, by using head orientation (assuming the head is tracked) or gaze direction using an eye tracker.

We assert that all user selection operations are susceptible to error. First, there is human error: the imprecision that comes from lack of experience, not enough motor control to do fine grained selection, or fatigue developed during a session; Wingrave et al. [1] studied a number of correlations between certain attributes of users and their ability to perform selections. Second, there is equipment error, which could be noise, drift, and lag in a 6DOF tracking system, or simply not enough resolution on a wheel-based device to perform a fine selection. Finally, there are ambiguities associated with the scene itself, such as when the user tries to select one object occluded by another object. In our main application area, mobile augmented reality, this is a common problem because users have "x-ray vision" and can see spatial information, such as the position of a collaborator, that may be occluded by real or virtual objects—in this example, the collaborator may be behind a building. These errors can lead to selections that are totally incorrect, such as when using a ray-based selection that chooses a single object, or to ambiguous results when using multiple selection techniques that can choose many candidate objects.

We designed a pointing-based probabilistic selection algorithm that alleviates some of the error in user selections. This technique takes into consideration the hierarchical structure of the scene objects (e.g., a door is a child of a wall, which is a child of a building, and so on). It assigns probabilities that the user has selected particular objects, within a frustum along the user's pointing direction, using a set of low-level 3D intersection-based selection techniques and the relationship of the objects in a hierarchical database, and makes the final selection using one of several weighting schemes. We implemented

This paper is based on "Toward Disambiguating Multiple Selections for Frustum-Based Pointing," by G. Schmidt, D.G. Brown, E.B. Tomlin, J.E. Swan II, and Y. Baillot, which appeared in the Proceedings of the IEEE Symposium on 3D User Interfaces 2006 (3DUI), Alexandria, VA, USA, March 2006.

{gregory.schmidt, dennis.g.brown, erik.tomlin}@nrl.navy.mil
baillot@ittid.com
swan@acm.org

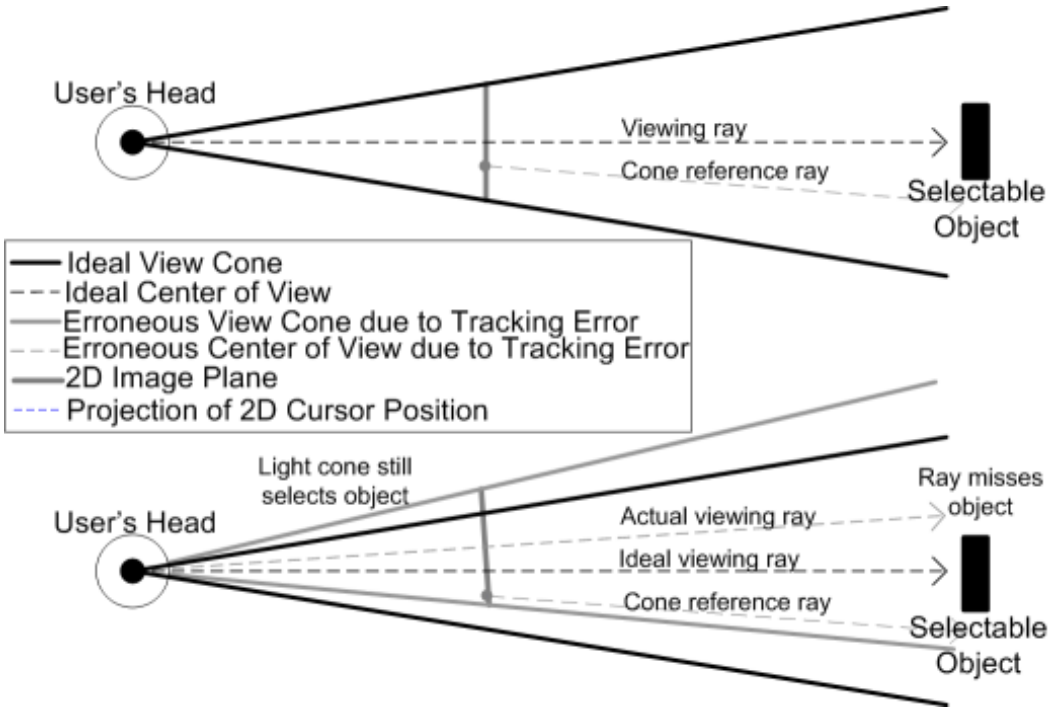

Figure 1. Motivation for the problem we address. The ray pointing technique fails to the select object when error is factored in. The cone works fine, but introduces ambiguity in the selection.

Figure 2. Shows the problem with the frustum-based (or cone-based) technique where multiple objects intersect with the viewing frustum. Left depicts the cone projecting from User 1's hand. Right shows an overview of the projected cone and interesecting objects.

the techniques in a testbed consisting of a command-and-control center (immersive VR and 2D map systems) and several field operatives (mobile AR, both man-wearable and vehicle-mounted) all using a shared database. We used the implementation to perform several experiments to evaluate the low-level selection techniques, to evaluate several weighting schemes for the integration algorithm, and to show that the algorithm can effectively disambiguate multiple selections.

After describing related work, we detail the low-level selection techniques and their implementation within our testbed. We then present the design and discuss the results of the experiments described above. This paper is an invited extension to work published at the IEEE Symposium on 3D User Interfaces 2006 [2]. This version adds a detailed discussion of the system architecture and the implementation of the selection algorithms.

II. Related Work

Selection in 3D environments has been an active research topic since the first virtual environments were implemented. Hinckley et al. [3] presented a survey of, and a common framework for, techniques for 3D interaction. Liang and Green [4] developed the spotlight method of selection, which alleviated some issues with using ray-based selection for small and far objects, but introduced the problem of multiple selections, for which they set up rules to choose one of the possible selections. Mine [5] described a few techniques for selection, along with other interactions to be supported in virtual environments. Forsberg et al. [6] developed two novel selection techniques, aperture (an extension of spotlight) and orientation, to deal with the imprecision of ray-based selection using a 6DOF input device. Pierce et al. [7] introduced a set of selection techniques using tracked hands and the 2D projection of a 3D scene. In his thesis, Bowman [8] gave a thorough survey of 3D selection techniques at the time, and introduced a novel technique for selection and manipulation.

In more recent work, researchers have dissected the task of selection even further, and have created novel techniques for specific application domains such

as Augmented Reality (AR) and multimodal systems. Wingrave et al. [9] discovered that users do not have an internal model of how they expect the environment to behave (for example, in performing selections), but instead they adapt to the existing environment using feedback received when performing tasks. Olwal et al. [10] developed some of the first algorithms for selecting in AR. Their technique attaches a virtual volumetric region of interest to parts of a user's body. When the user moves the body part, interacting with objects in the environment, a rich set of statistical data is generated and is used for processing selections. Kolsch et al. [11] developed a real-time hand gesture recognition system that can act as the sole input device for a mobile AR system. Kaiser et al. [12] developed mutual disambiguation techniques and evaluated their effectiveness for 3D multimodal interaction in AR and Virtual Reality (VR). They showed that mutual disambiguation accounts for over 45% of their system's successfully recognized multimodal commands.

In designing and implementing our testbed, we looked at the abundance of work towards developing natural user interaction techniques. Some of the key contributions have been: recognizing speech [13], combining pointing with speech [14]; recognizing sign language [15], hand movements [16], and body movements [17]; multimodal mutual disambiguation [18]; and reducing processing of gestures [19]. Recent work that applies some of these techniques can be found in [20]–[23]. We utilized some of the above user interface building blocks in the design of our architecture.

We acknowledge that multimodal systems, such as those mentioned above, are an effective way to alleviate some selection issues (for example, the user points to a group of objects that includes a window, and says the word "window," giving the system the means to disambiguate the selection), and we have implemented multimodal (speech and pointing) processing in our system for disambiguating different types of objects (for example, windows, walls, and buildings). However, if there is more than one object of the same type (for example, four windows) in the selection space, then the system described above will fail since the utterance "window" is ambiguous and does not help correct a wrong selection. In this example, either more sophisticated speech semantics or pointing-based selection processing techniques are needed.

In this paper, we have chosen to focus solely on improving pointing-based selection. Our selection algorithm introduces the concept of executing multiple selection techniques in parallel and choosing the final selection from the results of those techniques using a weighting scheme. The low-level techniques and the integration algorithm are described in the next section.

III. Probabilistic Selection and Integration Algorithms

Selection by pointing in 3D environments is inherently imprecise when the user is allowed to select occluded objects—the user may have the impression of pointing to a specific object, for example, but the system may not know for sure which object in the pointing direction is meant to be selected. Users often make pointing errors, especially when selecting small objects, objects at a distance, or when trying to make a selection quickly. Furthermore, pointing provides the object's direction, but not distance, so when several objects lie in the direction the user is pointing, it remains unclear which object the user intended to select.

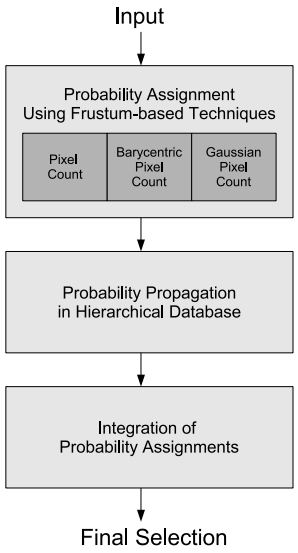

Figure 3. Flow of the pointing-based selection algorithm.

To deal with selection ambiguity, we designed a probabilistic selection algorithm that generates lists of candidate objects the user may have meant to select, and probability estimates of how likely it is the user meant to select each object. The algorithm combines several intersection algorithms and the hierarchical structure of the dataset, and then integrates the resulting candidate selections. The processing steps of the algorithm are shown in Figure 3, which we describe in each of the following sections.

A. Frustum Intersection Algorithms

We have designed a set of three algorithms which attempt to mitigate the ambiguity associated with ray intersection. Each algorithm is based on the concept of rendering the scene into a small *selection frustum* (see Figure 4); the rendered scene is viewed by a camera pointing coincident to the selection ray, and the frustum is rendered into an off-screen image buffer. The algorithms then count and classify the pixels in this buffer, and use these counts to create a list $(o_1, p_1), \ldots, (o_n, p_n)$ of potentially selected objects o_i and associated selection probabilities p_i. This list is in the format required for multimodal integration with other input streams [24], such as (for example) a list $(v_1, p_1), \ldots, (v_n, p_n)$ of probability-ranked p_i interpretations of a voice commands v_i.

As described in more detail below, each of these algorithms has differing utility depending on the user's preferences for making selections, on what type of object the user is trying to select, and on its relationship to

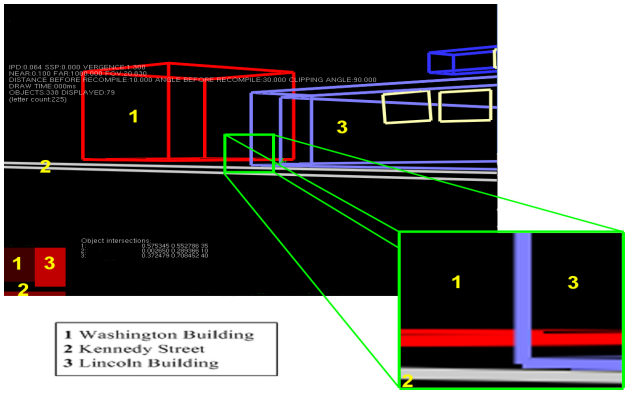

Figure 4. The operation of the PIXEL-COUNT algorithm. The scene is rendered into the small selection frustum shown in the center of the image.

other objects in the scene. We have designed the three intersection algorithms such that each has a different user preference for selection. These preferences are: (1) select the item nearest the central pointing ray; (2) select the largest item in the viewing frustum; and (3) select using a combination of the two other approaches. We wanted to find out if having several algorithms available based on different user preferences increases the chances for correctly selecting objects. These algorithms could either be used individually or executed in parallel and their results integrated together.

We describe each intersection algorithm in more detail, and then show how their output lists of candidate selections are integrated when the algorithms are run in parallel.

Pixel-Count The PIXEL-COUNT algorithm preferentially orders objects according to their projected size in the selection frustum. PIXEL-COUNT simply counts the number of pixels occupied by each object, and weighs the objects accordingly. This pixel-counting technique is a very fast way of implementing ordering of objects by projected size. A similar technique has been reported by Olwal [10].

PIXEL-COUNT
Input: 3D direction
Output: list $(o_1, p_1), \ldots, (o_n, p_n)$ of candidate objects o_i and associated probabilities p_i
1 calculate a small frustum about 3D direction
2 **for** each object o_i in the frustum
3 render o_i into the frustum
4 $pix_i \leftarrow$ number of pixels covered by o_i
5 weights $w_i \leftarrow pix_i / total\text{-}frustum\text{-}pixels$
6 assign probabilities p_i from weights w_i
7 sort (o_i, p_i) list by decreasing probabilities p_i

Figure 4 demonstrates PIXEL-COUNT. The green square in the center of the image demonstrates one size of the selection frustum; in the lower-right the frustum contents are enlarged. Note that the frustum size may be adjusted by the user. The square in the lower left corner shows a low-resolution re-rendering of the frustum contents.

The PIXEL-COUNT algorithm is robust to noise and pointing ambiguity. However, it inherently assumes the user is attempting to select larger objects, and it does not work well for selecting small objects near larger objects.

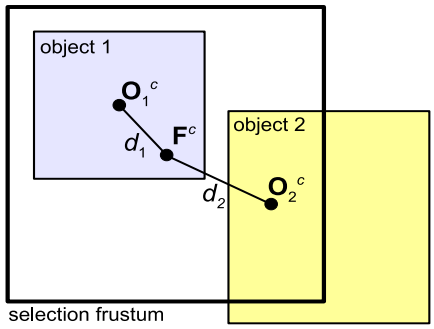

Figure 5. The operation of the BARYCENTRIC-PIXEL-COUNT algorithm. Because $d_1 < d_2$, each pixel of object 1 will be weighted more heavily than the pixels of object 2.

Barycentric-Pixel-Count This algorithm was motivated by our observation that users tend to point toward the center of the visible part of the object they wish to select. Figure 5 describes how BARYCENTRIC-PIXEL-COUNT operates. The algorithm calculates the center point of the visible portion of each object ($\mathbf{O}_1^c$ and $\mathbf{O}_2^c$), and then determines the distance to the center of the selection frustum (d_1 and d_2). It then weighs each object's pixels with the inverse of this distance; so in Figure 5 object 1's pixels are weighted by $1/d_1$ and object 2's pixels by $1/d_2$. Since it is assumed that the user is intending to look at one object, probabilities are estimated by normalizing the weights across all of the weighted pixels.

BARYCENTRIC-PIXEL-COUNT
Input: 3D direction
Output: list $(o_1, p_1), \ldots, (o_n, p_n)$ of candidate objects o_i and associated probabilities p_i
1 calculate a small frustum about 3D direction
2 let $\mathbf{F}^c$ be the center of the frustum
3 **for** each object o_i in the frustum
4 let $\mathbf{O}_i^c$ be the center of the visible portion of o_i
5 $bary\text{-}weight \leftarrow 1/\|\mathbf{F}^c - \mathbf{O}_i^c\|$
6 render o_i into the frustum
7 **for** each pixel a generated by o_i
8 $pix_i \leftarrow pix_i + a * bary\text{-}weight$
9 weights $w_i \leftarrow pix_i / total\text{-}frustum\text{-}pixels$
10 assign probabilities p_i from weights w_i
11 sort (o_i, p_i) list by decreasing probabilities p_i

BARYCENTRIC-PIXEL-COUNT works very well for selecting small objects near larger objects, but it does not work well if the user points away from the center of an object, or if the object has a shape such that the Barycentric center does not lie within the object itself.

Gaussian-Pixel-Count The GAUSSIAN-PIXEL-COUNT algorithm is also motivated by the general observation that users tend to center the objects they want to select. However, this algorithm tries to address the failing of the BARYCENTRIC-PIXEL-COUNT algorithm, which occurs when the Barycentric center does not lie within the object itself. GAUSSIAN-PIXEL-COUNT operates by applying a Gaussian mask, centered in the selection frustum, to each

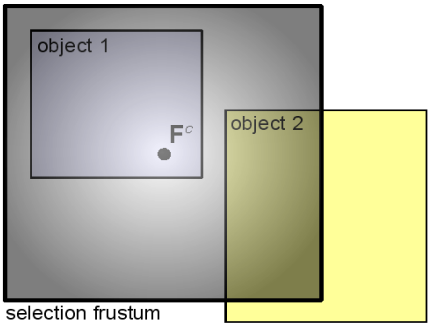

Figure 6. The operation of the GAUSSIAN-PIXEL-COUNT algorithm. Pixels in objects 1 and 2 are weighted by a circularly symmetric Gaussian function centered at $\mathbf{F}^c$.

object's pixels. The mask operates, in effect, by assigning weights to each pixel based on its distance from the center ray according to a Gaussian bell curve. Figure 6 describes how GAUSSIAN-PIXEL-COUNT operates. The filtered output for each individual object is combined in an accumulation buffer. Probabilities are assigned, again, assuming one object is intended to be selected, by normalizing across the weighted pixels.

GAUSSIAN-PIXEL-COUNT
Input: 3D direction
Output: list $(o_1, p_1), \ldots, (o_n, p_n)$ of candidate objects o_i and associated probabilities p_i
1 calculate a small frustum about 3D direction
2 calculate a Gaussian filter G centered in frustum
3 **for** each object o_i in the frustum
4 render o_i into the frustum
5 **for** each pixel a generated by o_i
6 $pix_i \leftarrow pix_i + a * G$
7 weight $w_i \leftarrow pix_i / total\text{-}weighted\text{-}frustum\text{-}pixels$
8 assign probabilities p_i from weights w_i
9 sort (o_i, p_i) list by decreasing probabilities

The algorithm is less susceptible to being biased by large visible objects and it favors selecting objects near the central viewing ray.

B. Probability Propagation in Hierachical Database

The probability estimates generated by the ray intersection algorithms assume that a single object occupies a given space in the viewing frustum. This assumption is not the case for a hierarchically-organized database, which contains objects composed of smaller objects, which in turn, may be composed of even smaller objects, and so forth. This type of database can have several objects occupying a given space, though there will always be a relationship between the occupying objects. An example of a hierarchically-organized database, which we use in our testing, and the inter-relationships between objects, is illustrated in Figure 7.

In order to assign probabilities properly for a hierarchically-organized database, we (1) set the ray-intersection algorithms to probabilities for the lowest level structure for each pixel, and (2) propagate the probabilities up the tree hierarchy from the leaf nodes. Since

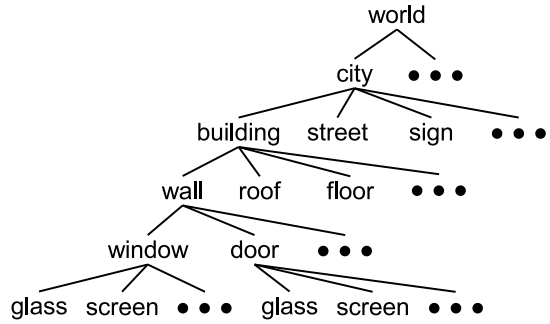

Figure 7. A portion of a hierarchically-organized database that contains urban data such as buildings, streets, signs, etc.

our ray intersection algorithm returns the lowest-level structures, we use the following algorithm to propagate probabilities up the database hierarchy:

PROBABILITY-PROPAGATION
Input: old list $L_O = (o_1, p_1), \ldots, (o_n, p_n)$ of objects o_i and associated probabilities p_i
Output: new list $L_N = (o_1, p_1), \ldots, (o_m, p_m)$
1 create empty list L_N
2 **for** each object o_i in L_O
3 **if** o_i not in L_N **then**
4 add o_i to L_N
5 $o_i.weight \leftarrow p_i$
6 **for** each recursive parent of o_i
7 **if** $o_i.parent$ not in L_N **then**
8 add $o_i.parent$ to L_N
9 $o_i.parent.weight \leftarrow o_i.parent.weight + p_i$
10 **for** each object o_i in L_N
11 normalize $o_i.weight$
12 assign probability p_i from $o_i.weight$
13 sort L_N by decreasing probabilities

One subtlety to note is in line 5, where we consider the probability for each pair as a "weight" for that pair, since we perform operations with these values that do not strictly consider the values as probabilities. The resulting probability assignments estimate likelihood that any of the occupying objects for a given space is the desired selection. For example, using the hierarchy in Figure 7, for a ray intersecting a window, its probability of selection is equal to the probability of selecting the wall, building, city, and world, at the intersecting pixel. Another property to understand using this probability propagation approach is the probabilities for the parents are always at least as much as the probabilities for any child. Keep in mind, another reasonable and equally valid manner of assigning probabilities is to consider the hierarchical nature of the database at a lower level, when the ray intersection algorithms estimate the probabilities. This approach may lead to different but still similar assignments of probabilities.

C. Integration of Probability Assignments

The three lists, L_P, L_B, L_G, of objects and associated probabilities generated by the ray intersection algorithms and probability propagation algorithm need to be combined into one list L_N. One caveat to this process is

that each list may contain a slightly different set of object-probability pairs due to differences in the how each algorithm operates. Thus, the elements of the lists will have to be matched and like items combined. Another important note is that, in the process described below, just as in the probability propagation algorithm, we consider the probability for each pair as a "weight" for that pair, since we perform operations with these values that do not strictly consider the values as probabilities. A naive integration approach would be, for each object, to simply average the weights assigned to that object from the different algorithms. This approach, however, does not take into consideration the strengths and weaknesses of each of the three algorithms. A more appropriate way to integrate them is to assign a weight to each algorithm, W_i, based on how well each performs in comparison to the others. The lists are then integrated by the following WEIGHTED-INTEGRATION algorithm.

WEIGHTED-INTEGRATION

Input: 3 lists L_G, L_B, $L_P = (o_1, p_1), \ldots, (o_n, p_n)$ of objects o_i and associated probabilities p_i

Output: new list $L_N = (o_1, p_1), \ldots, (o_m, p_m)$

```
create empty list L_N
for each list L_j in L_G, L_B, L_P
    for each object o_i in L_j
        if o_i not in L_N then
            add pair to L_N
        else
            find o_i in L_N
        L_N.o_i.weight ← p_i * W_j
for each object o_i in L_N
    normalize o_i.weight
    assign probability p_i from o_i.weight
sort L_N by decreasing probabilities
```

The integration weights, $W_i, i = G, B, P$, corresponding to the intersection algorithms GAUSSIAN-PIXEL-COUNT, BARYCENTRIC-PIXEL-COUNT, and PIXEL-COUNT, respectively, are initially arbitrarily assigned to $\frac{1}{3}$ each (giving the same effect as the naive method of averaging). However, we acknowledge that having proper weight assignments for data integration is important for optimizing performance and is a difficult task. We made several attempts to refine the weight assignments. We used the performance estimates of the intersection algorithms to influence the assignment of the weights in the integration. In one case, we normalized the algorithm performance estimates and used those as the weight values. For the other case, we used the normalized performance estimates as a guide to refine the weight assignments. More details about the assignment of weights is given in Section V.

IV. ALGORITHM IMPLEMENTATION

In order to test the selection algorithms and integration schemes, we developed a multimodal interaction system and user interface for our combined AR and VR system, termed BARS™(Battlefield Augmented Reality System) [25]. We have been developing this system at

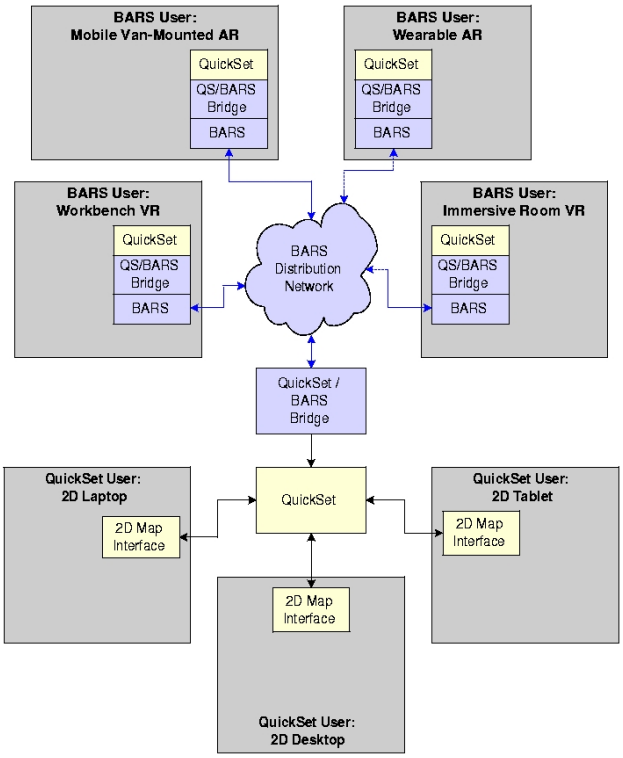

Figure 8. High level architectural view of system and interconnections with several instances of the architecture on different platforms.

our laboratory since 1998. This multimodal interaction system adds a range of support for a large number of input and display devices, interaction and display algorithms, as well as the interface techniques and associated selection algorithms and integration schemes described in this paper. The multimodal interaction system operates as a nexus, handling input communications, converting and processing heterogenous data, and handling output communications between computer and user. When multiple platforms are linked together, each has a copy of the interaction architecture running on it, and they share a common distributed database. Figure 8 shows the interlinking of several different platforms running the interaction system. Figure 2 (left) shows the system operating with an overview camera, on two mobile users, and in a van.

The multimodal interaction system consists of several interconnected pieces. Figure 9 shows all the major interaction components of the system, including selected features from BARS and Quickset. Quickset [24] is an agent-based architecture that supports probabilistic, asynchronous input events. These pieces include several modules contained within BARS: Speech UI, Ink UI, and Selection UI, as well as the QS/BARS bridge. The links between the systems include object database synchronization messages so that each subsystem has a consistent world database, object selection information so that the mulitmodal services in Quickset know what's happening in BARS, and control of the speech recognizer.

We implemented our probabilistic selection algorithms and integration schemes within the following four components of the testbed architecture:

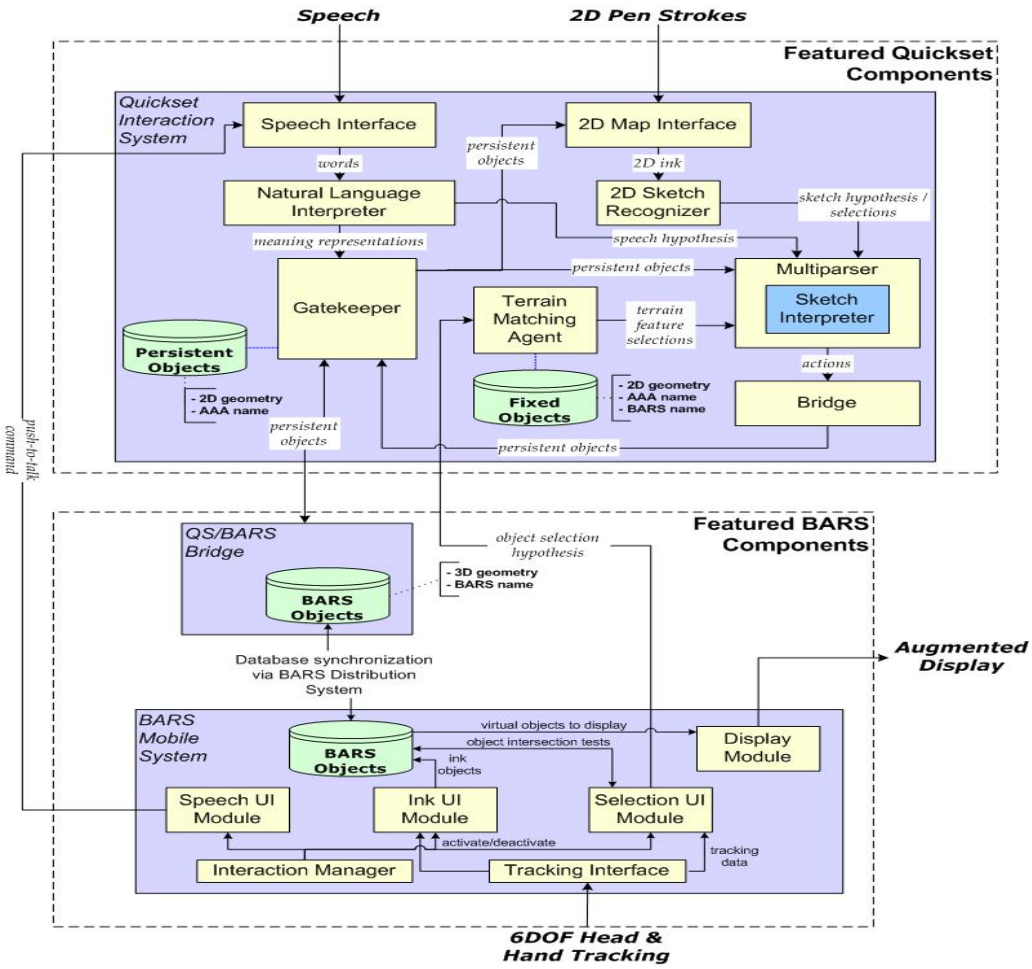

Figure 9. Key components and communication channels of the integrated system.

Selection System: The Selection System uses several sources of input:

- *User Viewpoint*: A vector projecting forward from the user's head.
- *Rotational Offset of User Viewpoint*: A vector projecting through the 2D cursor position on the screen.
- *Hand-Based 3D Shape*: A three-dimensional shape (cylinder, cone, sphere, etc.) attached to the position of the user's hand.
- *2D Digital Ink*: A 2D ink stroke.
- *3D Digital Ink*: A 3D ink stroke.
- *Speech*: Speech filtered by grammar and natural language interpreter.

The first three sources feed the selection algorithms that are the main focus of this paper.

Speech System: The Speech System gathers the results of a natural language agent, which is built on top of a commercial speech-to-text engine. For each context of the system (selecting objects, issuing commands, etc), a different grammar is enforced. By limiting the parsed speech to a specific grammar set, we slot the utterances into specific words or phrases we expect to hear, increasing the likelihood of getting useful speech from the user. The system determines the probability that a given utterance matches each word or phrase in the grammar. In the object selection context, the grammar consists of the names of types of selectable objects. The speech provider determines a list of object types that the user probably intended to select, along with the probability that each type was spoken.

Multiparser Integration System: This system consists of the Quickset Multiparser, Bridge, and Gatekeeper. The core features of these sub-systems are described in Cohen [24]. We extended them to link properly with the BARS system, our selection tasks, and our database, and we designed a communication protocol for passing speech and selection-based gestures. In order to utilize specific grammars, we designed database elements to process variations of speech commands and references to different objects in the database. For example, some commands are "select this", "reset the", and "danger on this". Some example objects are "building", "window", and "street". The grammar defines rules to process the commands, which consist of verbs, articles, and objects.

We augmented the Gatekeeper and Bridge with modules that perform the proper syntax conversion and protocol interpretation. These modules communicate with the speech and Quickset/BARS bridge. They also handle transfers of appropriate database elements and conversions to the persistent object types accepted by the Quickset architecture.

The Multiparser sub-system integrates feature informa-

tion from the ranked lists of candidate selection objects. This process rules out inconsistent information while fusing complementary or redundant information through a matching process [23]. The process primarily depends on a type hierarchy and a set of spatio-temporal constraints. Speech, object type, and the choice of articles and prepositions assist in the unification and disambiguation of some ambiguous selection possibilities.

Quickset/BARS Distribution System: A special type of BARS application is the *bridge* type. A bridge joins the BARS distribution system and an outside system, translating object creation and change events between BARS and the outside system. A bridge represents objects in BARS and a connected system by maintaining a map that links the BARS and outside system's objects. To connect BARS and Quickset, we created a new bridge implementation that acts as both a Quickset agent and a BARS instance.

V. Performance Evaluation

We conducted several experiments to gain a better understanding of the effectiveness of the algorithms for disambiguating multiple pointing selections. We approached this task by first comparing empirically the three intersection algorithms head-to-head to learn the strengths and weaknesses of each algorithm for specific dataset cases. Second, we evaluated the integration algorithm, exploring several weighting schemes, to determine how best to utilize combinations of the intersection algorithms in parallel. Lastly, we conducted a short experiment to demonstrate and empirically evaluate how well the algorithms work for disambiguating selections.

A. Comparison of Intersection Algorithms

We conducted three experiments to compare the three intersection algorithms head-to-head. The first experiment was designed to test the experimental protocol, flushing out any design and testing issues, and used a simple real-world urban dataset and a few test cases. Each successive experiment increased the detail of the datasets and complexity of the test cases. Comparing any algorithm thoroughly typically requires running an extensive set of experiments with multiple datasets and conditions. The goal of these experiments was to get a general feel for the accuracy of each algorithm for performing selection using a real-world urban dataset.

Since our overall goal is to apply these algorithms for disambiguating multiple selections, we acknowledge that precision plays a role in how we evaluate the algorithms. In particular, the selection cases we address only become interesting when high precision is required, otherwise the selections could be easily performed using well-known ray-based pointing techniques. We ran a set of preliminary tests to evaluate the degree of precision needed for the experiments and tested the selection techniques on a range of sizes for objects and varying amounts of space between each. Some of the test cases are shown in Figure 10. We

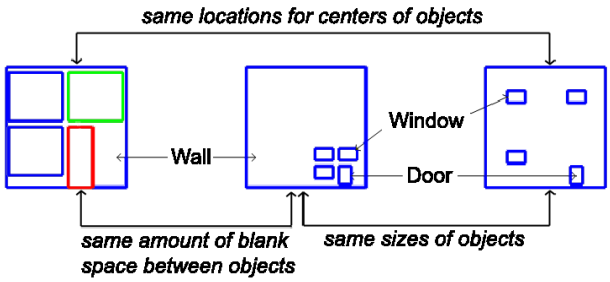

Figure 10. Three of the test cases used to determine the degree of precision needed by the experiments. We varied the size of several windows, and the spacing between each, and asked several users to perform selections of the windows for each test case. Statistics were collected to determine the precision required for our experiments.

used the precision estimates to guide our choice of the test cases for the experiments, making sure that the required degree of precision was high, but low enough to collect meaningful data to compare the three algorithms. Later, when we tested the algorithms for disambiguating selections of smaller, distant objects, we increased the degree of precision required. We next describe the experiment preparations, experimental protocol, and each experiment.

Experimental Protocol For each of the three experiments, we recruited two to four subjects from our development team. Each subject wore a tracked, see-through head-mounted display (HMD) that included a microphone for the speech recognizer. The orientation of the head was used as the pointing direction. The system showed a cross-hair cursor at the center of the view frustum for the pointing direction. The view frustum's borders were made invisible to the user and its size was kept fixed throughout the experiments. The experiments operated under the assumption that the user's selection strategy was to point directly through the center of the object intended to be selected. Other strategies, such as selecting the largest item in the view frustum, are evaluated later in Section V-C.

Each subject was given a training session to become familiar with the pointing and speech input interaction mechanism. The subjects were prompted by the experiment administrator to perform selections using head direction combined with a voice command. For example, the subject is asked to point at the upper right window of the left building, shown in Figure 10, while speaking "secure this window." The system responds by changing the color, based on the voice command, of what the system determines to be the correct selection. In Figure 10, the verb "clear" triggers a change in the color of the upper right window to green.

The trials of each experiment asked the subjects to perform a sequence of selections over a range of test cases. The selection test cases presented were sets of windows, doors, walls, and buildings. The speech actions included "danger on this object," "clear this object," and "reset this object" — these commands are meaningful in the urban situational awareness domain of the BARS system [25]. Data were collected for the three frustum-based algorithms. Cases where the speech recognizer

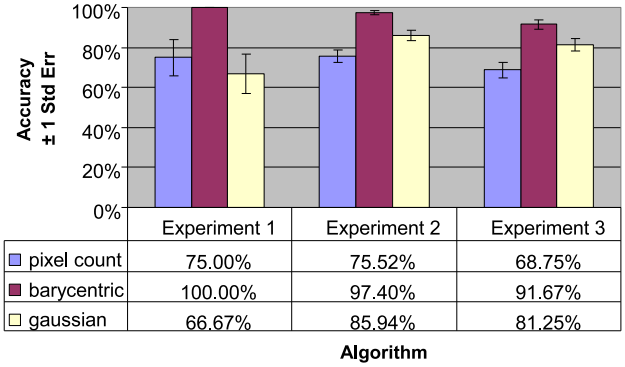

Figure 12. The accuracy (mean percentage of correct selections) given by the three intersection algorithms in Experiments 1–3.

failed were thrown out, since we are not evaluating the speech recognizer nor the multimodal features of the system. The test cases were presented in a counterbalanced manner in order to eliminate any learning-effect biases.

Experiment 1 The first experiment used a small subset of the dataset shown in Figure 11. The dataset contains semantic information about the buildings surrounding our laboratory—it is a real-world database used by our system for various development, demonstration, and evaluation purposes. We ran the experiment, following the experimental protocol above, using two subjects. Each subject was asked to perform selections of windows and doors in three different buildings. We collected statistics for 12 different cases per user, for a total of 24 cases. We recorded the *accuracy* of the intersection algorithms; we recorded a boolean value of '1' if an algorithm made a correct selection, and a '0' if it made an incorrect selection. Figure 12 shows the accuracy performance of each of the three algorithms for Experiment 1.

Experiment 2 The second experiment expanded on the first experiment, this time using a larger dataset and a broader set of test cases. As before, the dataset chosen is a subset of Figure 11's dataset. We only presented the ground-level view of the dataset to the subjects. The test cases were determined based on observed strengths and weaknesses of the three intersection algorithms. We designed the test protocol such that the subjects would make selections for situations where each of the algorithms is weak, and cases where each is strong; we tested an equal number of "good" and "bad" test cases for each algorithm. We used four subjects and collected the performance statistics for 192 selection cases (48 per subject). The users were asked to make selections of windows and buildings. We recorded the accuracy performance for each of the three algorithms; the results for Experiment 2 are shown in Figure 12.

Experiment 3 The third experiment used the largest portion of our test dataset (Figure 11), as well as the broadest set of test cases, which we believe better explored the strengths and weaknesses of the intersection algorithms. This experiment followed the same protocol as Experiment 2, with three subjects and 144 selection cases (48 per subject), and an equal amount of good and bad test cases for each algorithm. We again recorded the accuracy performance for each of the three algorithms; the results are shown in Figure 12.

Results and Discussion We analyzed the accuracy performance results with a one-way analysis of variance (ANOVA). In addition to the p-values, the measure of *effect significance*, we calculated and reported ω^2, a measure of *effect size*. ω^2 is an approximate measure of the percentage of the observed variance that can be explained by the effect.

As suggested by Figure 12, we found a strong effect of algorithm for each of the experiments (Experiment 1: $F(2,69) = 5.07,\ p = .009,\ \omega^2 = 10.3\%$; Experiment 2: $F(2,573) = 20.7,\ p < .000,\ \omega^2 = 6.4\%$; Experiment 3: $F(2,429) = 12.7,\ p < .000,\ \omega^2 = 5.2\%$). The error bars in Figure 12, which show ± 1 standard error, indicate that BARYCENTRIC-PIXEL-COUNT had greater accuracy than both GAUSSIAN-PIXEL-COUNT and PIXEL-COUNT for all three experiments. For Experiments 2 and 3, GAUSSIAN-PIXEL-COUNT had greater accuracy than PIXEL-COUNT.

This analysis empirically validates that the choice of intersection algorithm makes a difference in selection accuracy. Approaches similar to what we are calling Barycentric have been presented as Liang and Green's spotlight method [4] and the aperture method of Forsberg et al. [6], but here we present the first empirical evidence for the general effectiveness of this technique.

However, although the data show that the BARYCENTRIC-PIXEL-COUNT algorithm clearly outperforms the other algorithms (for this choice of dataset and degree of pointing precision), we can observe some interesting test cases if we allow the user to view the same dataset from above the buildings. For example, we observed that concave shape patterns show up more often looking from above than from the ground-level views—see Figure 13. One weakness of the BARYCENTRIC-PIXEL-COUNT algorithm is how it operates on concave objects. The algorithm relies on an estimation of the center of the objects trying to be selected. As seen in Figure 13, one estimation of the center of Building 1 is located at the position where the 'o' is shown. Due to the complex nature of the behavior of the algorithm's weighting function, we decided to empirically evaluate the test case. We ran a quick study collecting statistics for how well different regions of the buildings could be selected properly using the BARYCENTRIC-PIXEL-COUNT algorithm. Pointing in the regions shown in blue (lighter shade) properly select the correct building, while the red (darker shade) regions fail. Empirical results show regions 'B' and 'C' fail 85% of the time.

B. Evaluation of Integration

We conducted three experiments to evaluate different weighting schemes for the integration algorithm. Our objective was to determine how to best utilize different

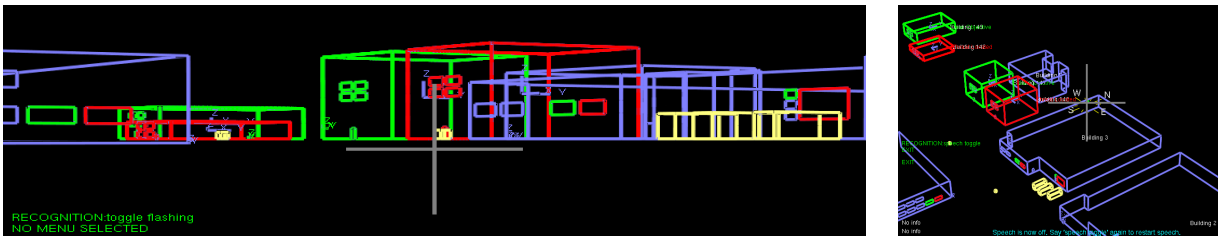

Figure 11. The dataset used in Experiments 1–3. Progressively smaller subsets of the dataset, with different test cases, were used in Experiments 1 and 2. (Left) Ground-level views; these are the views that subjects saw during the experiments. (Right) Elevated view; given to give the reader a feel for the dataset's layout.

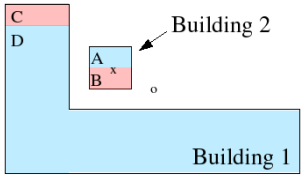

Figure 13. The effect of BARYCENTRIC-PIXEL-COUNT on selection involving a concave shape. The blue (lighter shade) regions work properly, while the red (darker shade) fail. Empirical results show regions B and C fail 85% of the time.

combinations of the intersection algorithms, including determining when to use an intersection algorithm by itself or when to combine the algorithms in parallel. We focused mainly on finding an optimal weight assignment for the typical use case and compared it against the best intersection algorithm (BARYCENTRIC-PIXEL-COUNT). In each experiment, we used the datasets and experimental protocol from the experiments in Section V-A. We applied several different weighting schemes (described below) to the integration algorithm, and for each scheme, ran an experiment and assessed the performance of the integration. We show the weight assignments for each experiment in Table I and the performance results of the integrations in Figure 14.

TABLE I.
WEIGHT ASSIGNMENTS FOR EXPERIMENTS.

Exp	Scheme	W_P	W_B	W_G
1	Equal Weighting	.3333	.3333	.3333
1	Performance-Proportional	.3103	.4138	.2759
1	Performance-Differential	.3031	.4238	.2731
2	Equal Weighting	.3333	.3333	.3333
3	Equal Weighting	.3333	.3333	.3333
3	Performance-Proportional	.28	.38	.34
3	2-Parm-Search (Scheme A)	.1894	.4688	.3418
3	2-Parm-Search (Scheme B)	.1138	.5400	.3462
3	Adhoc (Scheme C)	.1000	.5000	.4000
3	Majority Voting (Scheme D)	n/a	n/a	n/a

Weighting Schemes The weighting schemes listed in Table I and Figure 14 are:

- **Equal Weighting:** Simply assign the weights $W_P = W_B = W_G = \frac{1}{3}$.
- **Performance-Proportional Weighting:** Assign the weights $W_i = norm(A_i),\ for\ i = P,B,G$, where A_i are the accuracy estimates for i-th intersection algorithm.
- **Performance-Differential Weighting:** Assign the weights $W_i = sum(P_{i,j}^{correct} - P_{i,j}^{next\_highest})/n,\ for\ i = P,B,G,\ for\ j = 1..n$, where $P^{correct}$ and $P^{next\_highest}$ are the probabilities of the correct and next highest selection, n is the number of trials used to estimate the performance of algorithm i. Negative differences can be assigned zero.
- **Two-Parameter-Search Weighting:** Start with an estimate for one of the weights (assume weight W_1). Next, use performance estimates of the algorithms to compute the ratio $r = \frac{A_1 - A_2}{A_1 - A_3}$, where A_1, A_2, and A_3 are the respective algorithm accuracies. Compute the weights by solving the two equations: $W_1 + W_2 + W_3 = 1$ and $W_1 - W_2 = r(W_1 - W_3)$.
- **Adhoc Weighting:** Assign the weights by hand, based on observed trends in the performance of the other weighting schemes.
- **Majority Voting:** Select the candidate object that has the majority of the votes across the intersection algorithms. Empirical data, when using the equal weighting scheme, shows: (a) if all three vote for one object, the integration always votes for that object; and (b) if two of the three vote for the same object, the integration algorithm selects the same object a majority of the time.

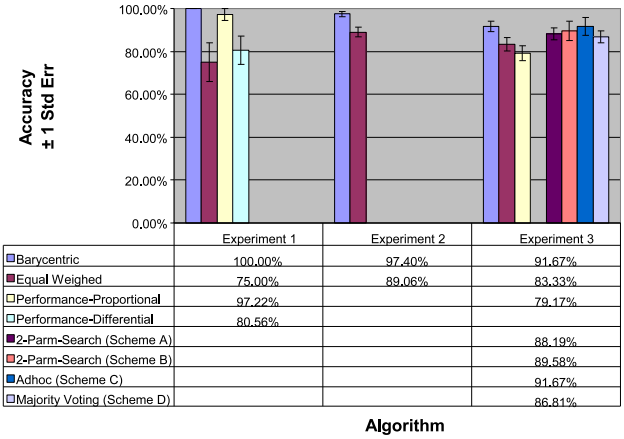

Figure 14. The accuracy of the integration schemes versus the BARYCENTRIC-PIXEL-COUNT intersection algorithm in Experiments 1–3.

Results and Discussion We again analyzed the accuracy performance with a one-way ANOVA. We found a strong effect of algorithm/integration scheme for Experiment 1 ($F(3,116) = 4.37,\ p = .006,\ \omega^2 = 7.8\%$) and Experiment 2 ($F(1,382) = 10.8,\ p = .001,\ \omega^2 = 2.5\%$). In addition, we found an effect for Experiment 3 ($F(6,809) = 2.17,\ p = .044,\ \omega^2 = .85\%$), and while this effect is significant, it is a substantially weaker effect than the others reported in this paper. The error bars indicate that in Experiment 1, performance breaks down into two groups: (1) the Barycentric intersection algorithm and the Performance-Proportional integration scheme, and

(2) the Equal Weighted and Performance-Differential schemes, with group (1) performing better than group (2). In Experiment 2, the Barycentric algorithm performed better than the Equal Weighted scheme. We only tested one integration scheme in this experiment because we decided it would be more interesting to look at test cases where the best intersection algorithm (Barycentric) had a lower global effectiveness, which motivated the next experiment. In Experiment 3, the Equal Weighted and Performance-Proportional schemes appear worse than the others, but otherwise the performance of all the schemes (and the Barycentric algorithm) is comparable. This relative equality is why the effect significance and size is substantially smaller for Experiment 3.

C. Evaluation of Disambiguation

The overall goal of our research was to develop methods for disambiguating multiple selections. We made strides towards this goal by developing several intersection algorithms and an integration algorithm that can combine them in many ways. The combination can be automatic, by using one of the schemes described previously, or it can be manual, by allowing the user to explicitly set the weights in the integration algorithm—for example, to only use the BARYCENTRIC-PIXEL-COUNT algorithm, the weights are set to $W_B = 1$ and $W_P = W_G = 0$. However, we have not yet shown that the integration algorithm is effective at disambiguating multiple selections, and we will address that now.

We conducted Experiment 4 to demonstrate, and evaluate empirically, the effectiveness of the integration algorithm in a simple disambiguation scenario. The experiment was performed using two subjects. We developed a dataset that requires a higher degree of precision for selecting by pointing than was necessary in the previous experiments. The dataset consists of a wall with nine windows; we placed a large window in the center and arranged eight smaller rim windows surrounding the center window. We asked subjects to select each of the nine windows separately, and the order of selection was random.

We collected statistics for 18 selections of the middle window and 31 selections of the outer windows. We analyzed the data considering how different selection strategies could be used for the scenario. Specifically, we considered how the windows could be selected using four strategies. The first strategy is to simply point at the window—"select object by pointing at it." The second strategy is to select the largest window, no matter where the user is pointing—"select largest object." The third and fourth strategies are combinations of the previous two, called "select largest object while pointing at it" and "select largest object while not pointing at it." We analyzed the data considering these four selection strategies and show the results in Figure 15.

The PIXEL-COUNT algorithm worked perfectly for selecting the largest object across all the selection cases. The most interesting result is the overlapping case, "select largest object while pointing at it," where the results show the success of the PIXEL-COUNT algorithm in comparison to the weaker performing BARYCENTRIC-PIXEL-COUNT algorithm. This result seems to indicate that as the precision required increases, the BARYCENTRIC-PIXEL-COUNT algorithm will not be as reliable as the PIXEL-COUNT for selecting the largest item.

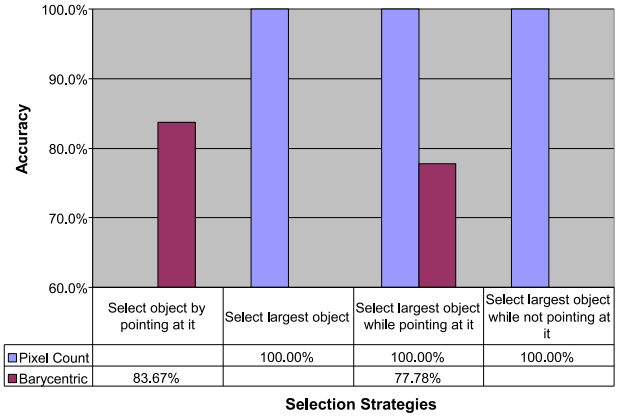

Figure 15. Comparison between the PIXEL-COUNT and BARYCENTRIC-PIXEL-COUNT algorithms over a range of user selection strategies. The dataset contains small, distant objects that are very difficult to select by pointing. The noise in pointing is high enough that BARYCENTRIC-PIXEL-COUNT fails often.

The difference in the performances between the "select largest object while pointing at it" and the "select object by pointing at it" strategies indicate that the surrounding objects may be selected correctly more often than the middle object. This phenomenon may occur because the rim objects have free space on some of their sides. If this is the case, the middle object has a slight disadvantage using the "select object by pointing at it" strategy.

VI. CONCLUSIONS AND FUTURE WORK

We presented three 3D pointing-based object selection techniques, PIXEL-COUNT, GAUSSIAN-PIXEL-COUNT, and BARYCENTRIC-PIXEL-COUNT, and applied them to the case of selection using a hierarchically-organized object database. We empirically demonstrated the general effectiveness of the BARYCENTRIC-PIXEL-COUNT technique. However, there are cases where that technique fails, so we developed an integration algorithm to try to leverage the strengths of each technique by combining their results using one of many weighting schemes. We evaluated these schemes and presented a careful analysis of the results. The bottom line is that different selection schemes work best in different scenarios, and the selection integration algorithm can disambiguate multiple selections.

There are several ways to improve this work. First, we need to take advantage of the semantic information in our database, for example, if it seems the user is selecting a window, it could be that the user is actually trying to select an object behind that window. Second, we can involve the user in the selection process beyond simple pointing. Perhaps the user could manipulate a dial or similar controller to scroll though the multiple selections until

the correct object is selected. Third, we could determine the best sets of weights for certain common types of databases, or even for different areas within a single database, and establish "weight profiles" to be employed for those databases or areas of databases. A modification of that idea is to classify weight assignments by the situations in which they work best, and then use that information for developing better performing automated selection techniques that adjust to the situation on-the-fly by swapping weight assignments. Another possibility is to have the algorithms learn from user indications as to whether the correct item was chosen or not. An implementation issue to address is how to properly handle ties in probabilities. For future studies, we would also like to add more subjects in the experiments.

ACKNOWLEDGEMENTS

We thank Matt Wesson, Dave McGee, Phil Cohen, Dennis Lin, Eric Burns, John Park, Elliott Cooper-Balis, Derek Overby, Aaron Bryden, Jason Jerald, Brian Hurley, Joshua Eliason, Jesus Arango, Eric Klein, Doug Maxwell, Simon Julier, Patrick Violante, and Mark Livingston for their contributions. This research was sponsored by the Office of Naval Research under grants #N00014-04-WR-2-0049, #N00014-02-1-0038 and #N00014-04-WX-2-0053 and by the Naval Research Laboratory Base Program grant #N00014-04-WX-3-0005.

REFERENCES

[1] C. Wingrave, R. Tintner, B. Walker, D. Bowman, and L. Hodges, "Exploring individual differences in raybased selection: Strategies and traits," in *Proc. of IEEE Virtual Reality Conference*, 2005, pp. 163–170.

[2] G. Schmidt, D. Brown, E. Tomlin, J. Swan, and Y. Baillot, "Toward disambiguating multiple selections for frustum-based pointing," in *Proceedings of the IEEE Symposium on 3D User Interfaces 2006 (3DUI)*.

[3] K. Hinckley, R. Pausch, J. Goble, and N. Kassell, "A survey of design issues in spatial input," in *Proceedings of User Interface Software and Technology 1994*, 1994, pp. 213–222.

[4] J. Liang and M. Green, "Jdcad: A highly interactive 3d modeling system," *Computers and Graphics*, vol. 18, no. 4, pp. 499–506, 1994.

[5] M. Mine, "Virtual environment interaction techniques, Tech. Rep. UNC Chanel Hill Computer Science Technical Report TR95-018, 1995.

[6] A. Foresberg, K. Herndon, and R. Zeleznik, "Aperture based selection for immersive virtual environments," in *Proceedings of User Interface Software and Technology 1995*, 1995, pp. 95–96.

[7] J. Pierce, A. Forsberg, M. Conway, S. Hong, R. Zeleznik, and M. Mine, "Image plane interaction techniques in 3d immersive environments," in *Proceedings of 1997 Symposium on Interactive 3D Graphics*, 1997, pp. 39–43.

[8] D. Bowman, "Interation techniques for common tasks in immersive virtual environments: Design, evaluation, and application," Ph.D. dissertation, Georgia Institute of Technology, Atlanta, Georgia, 1999.

[9] C. Wingrave, D. Bowman, and N. Ramakrishnan, "Towards preferences in virtual environment interfaces," in *Proc. of Eighth Eurographics Workshop on Virtual Environments*, 2002, pp. 63–72.

[10] A. Olwal, H. Benko, and S. Feiner, "Senseshapes: Using statistical geometry for object selection in a multimodal augmented reality system," in *Proc. IEEE Intl. Symposium on Mixed and Augmented Reality (ISMAR'03)*, Tokyo, Japan, Oct. 2003, pp. 300–301.

[11] M. Kolsch, M. Turk, and T. Hllerer, "Vision-based interfaces for mobility," in *Proc. of Intl. Conference on Mobile and Ubiquitous Systems*, 2004.

[12] E. Kaiser, A. Olwal, D. McGee, H. Benko, A. Corradini, X. Li, S. Feiner, and P. R. Cohen., "Mutual disambiguation of 3d multimodal interaction in augmented and virtual reality." in *Proc. Intl. Conference on Multimodal Interaction-Perceptual User Interfaces*, Vancouver, BC, Canada, Nov. 2003.

[13] J. Forgie and C. Forgie, "Results obtained from a vowel recognition computer program," *Journal of Acoustic Society of America*, vol. 31, no. 11, pp. 1480–1489, 1959.

[14] R. Bolt, ""put-that-there": Voice and gesture at the graphics interface," in *SIGGRAPH '80: Proc. of Computer graphics and interactive techniques*, New York, NY, 1980, pp. 262–270.

[15] L. Messing, R. Erenshteyn, R. Foulds, S. Galuska, and G. Stern, "American sign language computer recognition: Its present and its promise," in *Intl. Society for Augmentative and Alternative Communication*, 1994, pp. 289–291.

[16] J. Rehg and T. Kanade, "Digiteyes: Vision-based human hand tracking, Tech. Rep. CMU TR CMU-CS-93-220, Extended version of paper in ECCV May 1994 Stockholm, [ftp://reports.adm.cs.cmu.edu/usr/anon/1993/CMU-CS-93-220.ps.Z, 1993. [Online]. Available: citeseer.ist.psu.edu/rehg93digiteyes.html

[17] D. Gavrila and L. Davis, "Tracking of humans in action: A 3d model-based approach," in *Image Understanding Workshop*, 1996, pp. 737–746.

[18] S. Oviatt, "Mutual disambiguation of recognition errors in a multimodel architecture," in *Proc. of SIGCHI Conference on Human Factors in Computing Systems*, New York, NY, 1999, pp. 576–583.

[19] G. Schmidt and D. House, "Model-based motion filtering for improving arm gesture recognition performance," in *Gesture-based Communication in Human-Computer Interaction: Selected and Revised Papers from Intl. Gesture Workshop 2003, Lecture Notes in Computer Science*, vol. 2915. Springer-Verlag, 2004, pp. 210–230.

[20] M. Latoschik, "Designing transition networks for multimodal vr-interactions using a markup language," in *Proc. 4th IEEE International Conference on Multimodal Interfaces*, 2002, pp. 411–416.

[21] A. Wilson and S. Shafer, "Xwand: Ui for intelligent spaces," in *CHI '03: Proc. of SIGCHI conference on Human Factors in Computing Systems*, New York, NY, 2003, pp. 545–552.

[22] B. Thomas and W. Piekarski, "Glove based user interaction techniques for augmented reality in an outdoor environment," *Virtual Reality*, vol. 6, no. 3, pp. 167–180, Oct. 2002. [Online]. Available: http://www.springerlink.com/index/10.1007/s100550200017

[23] M. Johnston, P. Cohen, D. McGee, S. Oviatt, *et al.*, "Unification-based multimodal integration," in *Proc. Meeting of Assoc. for Computational Linguistics*, 1997, pp. 281–288.

[24] P. Cohen, M. Johnston, D. McGee, S. Oviatt, *et al.*, "Quickset: multimodal interaction for distributed applications," in *Proc. of Intl. Multimedia Conference*, New York, NY, 1997, pp. 31–40.

[25] S. Julier, Y. Baillot, M. Lanzagorta, D. Brown, and L. Rosenblum, "Bars: Battlefield augmented reality system," in *NATO Symp. on Information Processing Techniques for Military Systems*, Oct. 2000.