A Screen-Space Formulation for 2D and 3D Direct Manipulation

Jason L. Reisman *Philip L. Davidson* *Jefferson Y. Han*
Perceptive Pixel, Inc.
{ jason, philipd } @ perceptivepixel.com

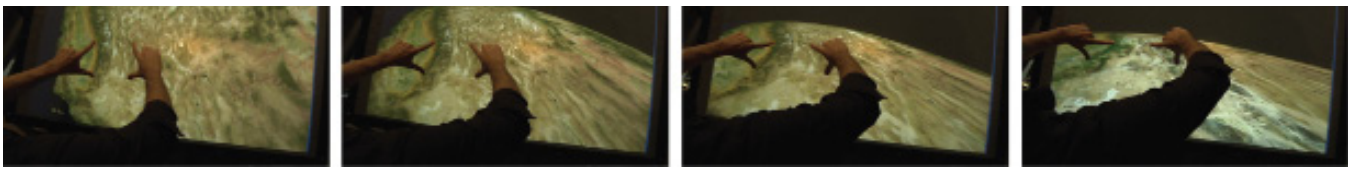

Figure 1: A 4-finger viewpoint manipulation

ABSTRACT
Rotate-Scale-Translate (RST) interactions have become the *de facto* standard when interacting with two-dimensional (2D) contexts in single-touch and multi-touch environments. Because the use of RST has thus far focused almost entirely on 2D, there are not yet standard techniques for extending these principles into three dimensions. In this paper we describe a screen-space method which fully captures the semantics of the traditional 2D RST multi-touch interaction, but also allows us to extend these same principles into three-dimensional (3D) interaction. Just like RST allows users to directly manipulate 2D contexts with two or more points, our method allows the user to directly manipulate 3D objects with three or more points. We show some novel interactions, which take perspective into account and are thus not available in orthographic environments. Furthermore, we identify key ambiguities and unexpected behaviors that arise when performing direct manipulation in 3D and offer solutions to mitigate the difficulties each presents. Finally, we show how to extend our method to meet application-specific control objectives, as well as show our method working in some example environments.

ACM Classification: H5.2 [Information interfaces and presentation]: User Interfaces. - Interaction Styles.

General terms: Design, Algorithms, Human Factors Experimentation

Keywords: Direct Manipulation, Multi-touch, Pressure, Optimization, Constraints

1 INTRODUCTION

The appeal of direct manipulation with multi-touch interfaces stems from the experience it offers. As the user slides their fingers along a touch surface, objects react by rotating, translating, and scaling themselves so that the same point on an object always remains underneath the same fingertip. Since objects move in a predictable and realistic fashion, users are given the impression of "gripping" real objects. Direct manipulation essentially provides an intuitive and controllable mapping between points in an object's local space and points in screen space, without the need for any explicit gesture processing.

As evidence of the appeal of direct manipulation, Rotate-Scale-Translate (RST) interaction has become the *de facto* standard when interacting with 2D contexts in a multi-touch environment. However due to the fact that RST-style interaction has thus far been almost entirely in 2D, it isn't entirely clear how to extend these principles into three dimensions. While many ways exist to manipulate 3D objects in a multi-touch environment, as far as we know none of them provide direct control. This is in contrast to what many users have come to expect when manipulating 2D objects such as photos, maps, documents, etc..

In this paper we describe a screen-space approach which attempts to resolve this limitation. Our method completely captures the 2D semantics of RST, while additionally allowing us to extend these same principles into three dimensions. We show that our method offers fine-grained control for any number of contact points and highlight some novel bimanual interactions that 3D direct manipulation enables. Furthermore, we discuss the issues which emerge when using a screen-space 3D direct manipulator, and offer solutions to help mitigate their effects. We believe these issues to be sufficiently general that they should occur in any screen-space direct manipulator which operates in 3D, regardless of whether one uses our particular formulation or another. Finally, we discuss how to extend our method by a combination of penalties and weights, and demonstrate that 3D direct manipulation is useful for a variety of tasks.

2 RELATED WORK

There exists a rich history of exploring 3D motion with 2D input devices. Chen et al [2] evaluate 3D rotation tasks, and find that continuous 2D rotation controllers such as 2DOF angular manipulators and 'Virtual Trackballs' improve rotation task performance over standard slider

UIST'09, October 4–7, 2009, Victoria, British Columbia, Canada.

implementations. Neilson et al. [9] demonstrate a range of tools for direct manipulation of three-dimensional translation, scaling, and rotation tasks using 2D input devices. These tools use graphical cues to dynamically select between a range of 1 and 2DOF control domains.

Most current touch manipulators in 3D restrict the user to indirect manipulation, rather than letting the user "touch" and affect the object directly. Perhaps most prominent among these is the Arcball rotational controller [11] which uses the product of quaternions to rotate an object about its center, thus allowing viewing from any direction. While Arcball does provide an intuitive rotational control, it does not allow the user to completely set the object's new position, as the center of rotation is fixed. Several bimanual techniques for indirect 3D manipulation were also explored on the responsive workbench in [3].

Common touch controls for higher order 3D manipulation factor the full 6 degrees-of-freedom (DOF) interaction into separate lower-order direct and indirect manipulation components - for example, composing the standard Arcball rotation control with well-known 4DOF planar interaction techniques [7]. Such control techniques are most intuitive when the motion of the contact point has a direct (0th or 1st order) mapping to the screen-space motion of a controlled point on the object (Arcball can be modified to achieve direct point manipulation). When this is not the case, it may be necessary to render graphical feedback in the form of control handles, rotation axes, or 'racks' [12], to indicate the expected effect of control point motion on the object's apparent motion. Graphical feedback is also effective when the full 3D motion of the object is artificially constrained.

In multi-point interaction, it is important to provide consistent (re)assignment of contact points to control DOF as contact points are added or removed from the interaction. This is generally achieved via priority, relative location, or through system-provided identification of contact points, as shown in [7]. The assignment paradigm is critical to composing multi-finger interaction gestures, especially when considering that the user should not experience any unexpected changes in the control mapping for each finger. Grossman demonstrated a bimanual assignment model (using a motion tracking system) for separable control of rotation and translation using full hand tracking on a 3D volumetric display [6] .

The drawback in separation and assignment of control in 6DOF manipulation is that our experience of real-world manipulation is a more holistic process - the regions grasped on an object remain in contact with the fingers and hand as the hand is moved to a position that satisfies those positional constraints. This continuous contact explains the ease with which users interact with 4DOF two-point interaction modes, as the manipulation satisfies direct point-to-point correspondence throughout the interaction.

Gleicher et al. in [5] noted the linear relationship between screen-space derivatives and transform derivatives, and used this to implement a screen-space direct manipulator. While Gleicher focuses on solving for camera motion, the system may be formulated with respect to object manipulation. As points move in screen-space, a linear system is solved to provide the corresponding movement in transform parameters. This enables a powerful direct manipulation mechanism, and is easily extended to handle a wide range of constraints. In our experience though, when used for real-time interactive applications, the forward integration of camera parameter derivatives may result in camera oscillations.

Some peculiarities of controlling camera motion via screen-space control points are addressed by Kyung et al. [8]. The authors note that in order to meet constraints the path taken by the camera's parameters may include rapid changes resulting in unpredictable camera motions. To handle this gracefully they smooth out the camera's motion by interpolating the camera parameters from immediately before and immediately after the regions of rapid change.

A variant of the numerical approach is to use a physics engine, and impart forces or constraints to objects in the scene so that they respond to control point manipulation in a direct fashion. In "physical user interfaces" interactive components are embodied as solid elements in the simulation environment. BumpTop [1] allows objects to be moved and collected via spring forces. Fröhlich et al [4] demonstrated 6DOF bimanual manipulation via spring forces on the Responsive Workbench. Recently, Wilson et al. in [13] tried two approaches, one which creates a solid element as a proxy in the scene, and another approach where the objects are manipulated by a stream of fluid particles.

While the underlying physical simulation can provide a number of convincing side effects during the interaction (inertia, collision), the complications of the approach lie in representing interactive forces in a manner that is consistent with both the capabilities of stable physical simulation and to user expectation, especially when the user operates the system in a strongly non-physical manner.

3 MACHINERY

Our method is very much in the spirit of Through The Lens Camera Control [5]. Recalling our definition of direct manipulation as a controllable mapping between points in the object's local-space and points in screen-space, each contact point defines a constraint which ensures the screen-space projection of the object-space point "touched" always remains underneath the user's fingertip. This amounts to continually updating the transformation which maps points in the object's local-space to points in screen-space such that the multiple constraints are best met.

3.1 Energy

We define the function $s(\mathbf{x},\mathbf{q})$ to be the function which maps object-space point $\mathbf{x}$ into screen-space point $\mathbf{p}$ as

$$\mathbf{p} = s(\mathbf{x},\mathbf{q}) = h(\mathbf{PM}(\mathbf{q})\mathbf{x})$$

in which $\mathbf{P}$ is the projection matrix, $\mathbf{M}$ is a matrix

parameterized by the vector **q** which maps **x** into world-space, and h is the viewport transformation. **M** is most likely the product of several matrices which are parameterized by the transform values (e.g., rotation, scaling, translation, etc.) and **P** is a constant matrix which describes the camera's field of view as well as how an object's image size changes with distance to the camera. Note that this means an object point's corresponding image point can only be altered by the local to world-space transformation, as we are not manipulating any of the camera's DOF.

We solve for the best-fit transform parameters by minimizing a quadratic energy function that measures the total squared error between the contact points' screen-space projection and their corresponding screen-space target positions. If $\mathbf{x}_i$ and $\mathbf{p}_i$ are the position of the i^{th} object-space contact point and screen-space target points, respectively, then our energy is defined as

$$E = \sum_i \left\| \mathrm{s}(\mathbf{x}_i, \mathbf{q}) - \mathbf{p}_i \right\|^2$$

and we look for solutions which minimize E w.r.t. **q**.

Our use of an unconstrained energy quadratic in screen-space error ensures us that the interface will continue to be responsive regardless of how many fingers a user places on the touch-screen. Note that minimizations which involve a large number of contact points may come at the expense of some slippage - i.e., deviation from screen-space target positions. As is usually the case when measuring error in L^2, the slippage will tend to be spread out as evenly as possible among the contact points. Alternative schemes for distributing the error will be discussed in Section 6.

3.2 World Space Transformation

So far we have made no mention regarding the type of transformation we wish to employ to map object-space point **x** into world-space. The choice of transformation, and particularly its DOF, is crucial to ensure the desired interaction. Since we are interested in extending familiar RST interactions from 2D into 3D, a transformation which inherently captures this style of interaction seems like a valid place to begin. That is, our transformation **M**(**q**) should contain the product of rotation (**R**), scaling (**S**), and translation (**T**) matrices. Since we are operating in 3D, we must also consider how the DOF which parameterize these matrices interact with the projection matrix (**P**).

Depending on our choice of **P**, from the user's point of view some of these DOF may be ambiguous. For example, if **P** is chosen to be a perspective transformation, scaling the object up will be similar in effect to translating the object so it becomes closer to the camera. However, if **P** is chosen to be an orthographic projection, then translating toward or away from the camera will have no effect, leaving scaling as the sole DOF capable of changing the object's screen-space extent. For these reasons all seven transform DOF (3 for rotation, 3 for translation, and 1 for scaling) may not be required or even useful at all times.

For the manipulators described in this paper, we assume that we are working with a perspective camera and thus an object's screen-space extent is affected by both translation and scaling. Since these DOF are redundant from the user's point of view, we make the assumption that our local-to-world-space transform's set of DOF will include translation along the camera's Z axis, but not scaling. Then, similar to the quaternion camera model described in [5], we can define our transform parameters **q** and local to world-space transform **M**(**q**) as

$$\mathbf{q} = \left[tx\ ty\ tz\ qx\ qy\ qz \right]$$
$$\mathbf{M}(\mathbf{q}) = \mathbf{T}(tx,ty,tz)\mathbf{Q}(qx,qy,qz)$$

in which **T** is the usual translation matrix and **Q** is a unit-quaternion matrix in which $qw = \sqrt{1 - qx^2 - qy^2 - qz^2}$. Note that this leaves us with a total of 6 DOF, a number which is especially important with regard to the solution method we chose to employ.

3.3 Minimization Method

A variety of methods may be used to minimize our nonlinear energy, including stepping through the transform parameter space in a manner similar to [5]. However, we achieved the best performance when treating the minimization as a nonlinear least-squares problem. By using a Levenberg-Marquardt algorithm [10], the minimization can be completed in a fraction of a millisecond on a 3 GHz Intel CPU. This allows for smooth, low-cost interactions on commodity hardware.

The only restriction Levenberg-Marquardt places on our problem is that we have at least as many terms in our energy as DOF. Each contact point adds two terms to our energy (error in screen x, screen y), so for one and two-point interactions this may mean locking down some DOF to the transform's current values. However, this can be made entirely consistent with standard RST-style interactions for one and two-point interactions. For minimizations involving three or more contact points, all of the transform's DOF can be employed.

4 INITIAL EXPERIENCES

We first verified our method by recreating a 4DOF RST manipulator. Adhering to the usual RST convention, we allowed only translation along camera X and Y when used with a single contact point, while the second contact added full 3D translation and Z axis rotation. When controlling 2D planar objects (embedded in 3D), we could not discern any difference between our manipulator and other 2D RST controllers.

Our initial 6DOF manipulator straightforwardly extended RST's mapping between the number of contact points and active DOF. The one and two-finger interactions remained the same as with the 4DOF manipulator, while the third contact enabled the remaining two DOF (rotation about the camera's X and Y axes). An immediate observation was that when we made motions with 3 or more fingers which were analogous to the familiar translation and scaling motions of 2-finger

4DOF control, the object moved in the expected manner (as if it were still under control of a 4DOF manipulator). However, what was *not* immediately clear were the motions expected of the user for controlling the two newly enabled DOF. Rotation into and out of the screen seemed to require more intricate motion of the fingers, and thus we proceeded with experimentation to learn how to effectively control all six DOF with this particular manipulator.

4.1 Three Finger Rotations

After a few minutes of use we learned to rotate objects freely in 3D by using three-finger, two-handed motions. (These motions were also possible with a single hand, but we felt they were much more comfortable when done bimanually.) A common approach was to pin the object down with two fingers of the non-dominant hand, while using a single finger from the dominant hand to swing the object either into or out of the screen. With this gesture, shown in Figure 2 (top row), we could define an axis by connecting the two contacts of the non-dominant hand, and then easily rotate the object about that axis.

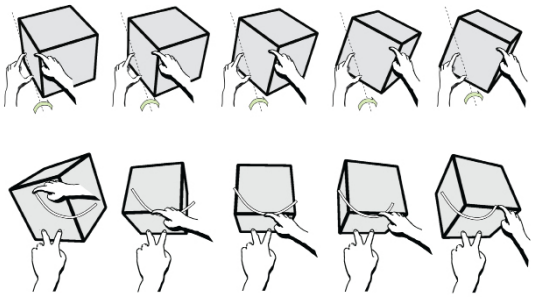

Figure 2: Two three-finger rotations

Rotating in this manner worked particularly well when the contact points were at nonuniform depths relative to the camera. Under these conditions, the orientation of the model combined with the resulting foreshortening provided us with enough spatial cues to easily control the object's motion. The path the dominant hand's finger needed to take in order to rotate the object about the axis was clear, and the object rotated either into or out of the screen as intended.

We also quickly observed that we were not limited to only rotating the object about the axis defined by the non-dominant hand. If we moved the dominant finger in a manner which did not coincide with rotating about this axis, the object turned to follow the direction of the dominant finger. While this turning did not permit the same range of motion as rotating about the axis, the interaction still felt fluid and natural. We called this overall type of operation a swivel interaction. An example is depicted in Figure 2 (bottom row).

When the contacts were all close to the same depth, however, the initial direction of rotation became more difficult to control. This was immediately apparent on planar objects whose initial orientations faced the camera. An example of this can be seen by looking ahead to Figure 5. As shown, the object sometimes rotated out of the screen when we intended it to rotate into the screen (or vice-versa). We could still achieve the desired rotation, but this required swivelling the object slightly out of the image plane in the manner described in the previous paragraph. Once this out-of-plane rotation occurred, we fell back on the usual spatial cues in order to guide the object into the desired orientation.

Rotating an object using contacts which were initially at different depths into positions in which all the contacts were at roughly equal depths also produced some surprising results. As shown in Figure 6, when the contacts were rotated to near equal depths, the motion rapidly changed to rotation about a different axis coupled with translation of the object towards the camera. After some thought this behavior made sense; since matching the contacts using the original rotation became impossible, the continued motion is a 3D variant of the scaling caused by moving points apart in 2D. Regardless, we still found this surprising and undesirable. We expected that a smooth, consistent motion of the fingertips would cause an equally smooth and consistent motion of the object. However, as noted in [8], this property is not guaranteed to be satisfied by Through The Lens techniques. We discuss methods to address both this rotational issue as well as the one described the previous paragraph in the next section.

Further experimentation also revealed that large rotations sometimes required multiple gestures to complete. For example, when using three fingers to rotate an object by 180 degrees or more, the contact point in motion necessarily became occluded as the object swung around. Our method still supported manipulating the occluded contact point (as it still mapped to a valid screen-space location), and despite not being visible the contact wasn't difficult to control. However, as it is impossible for one hand to pass through another, we had to remove and replace one or more of the contacts to complete the desired rotation. This limitation is not shared by [11,7], which support full 360 degree rotation with a single motion.

4.2 Emergent Interactions

Upon further use, we observed instances in which a small amount of finger motion could produce rotations of up to 180 degrees. These interactions were not immediately apparent, but rather became evident through use, and once discovered were easy to repeat. As far as we know, the following interactions cannot be supported by other manipulators.

For example, we learned to place two fingers from the dominant hand on the object and one from the other in a triangular configuration. We then rotated the dominant hand so that the three points became nearly collinear. As this happened the object rotated such that all three contacts were on a plane oriented 90 degrees away from the camera. (Whether the object rotated into or out of the screen

Figure 3: A three-finger shear rotate

depended on its initial orientation plus whether we rotated the manipulating hand clockwise or counterclockwise.) As shown in Figure 3, the rotation was accompanied by a twisting and scaling of the object, the image of which resembled a shearing operation. For this reason, we called this type of interaction a shear rotation.

We also found that new viewpoints could be achieved by exploiting foreshortening. As shown in Figure 4, this type of operation involved first placing four contact points on the object in a configuration that mimics the foreshortening at the initial viewpoint. The object could then easily be maneuvered into a new viewpoint by moving the fingers to positions which describe the foreshortening at the target view. This interaction could be used to flip an object almost 180 degrees by changing its viewpoint from one oblique perspective from to another. However, because of the arm and wrist motion required, this gesture felt most comfortable when used to rotate an object by 90 degrees or less. For an example of this working on a terrain navigation system, see Figure 1.

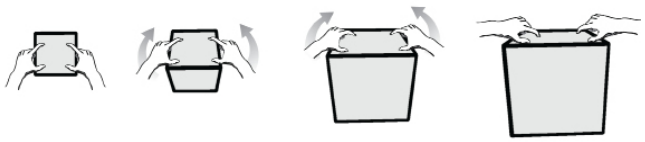

Figure 4: A four-finger perspective rotate

5 ROTATIONAL EXTREMA

As noted in the previous section, rotations including axes other than the camera's Z axis can produce some surprising results. Upon closer examination we found that these are caused by rotational extrema - i.e., points in the transform space at which screen-space distances are maximized with respect to rotational DOF. As our initial experiences informed us, these situations arise frequently during normal use. In this section we explore two major classes of rotational extrema problems and offer ways to identify and curb their influence.

5.1 Ambiguous Rotations

These situations occur when the axis of rotation is correct, but the object rotates in the opposite direction from what was expected. As described in section 4.1, this usually occurs when one contact point is brought closer to the axis defined by the other two stationary contacts. If all three contact points are at roughly equal depths, the axis of rotation will lie in the camera's X-Y plane, and the screen-space distance between the contacts is at a maximum with respect to the angle of rotation about the axis. Regardless of whether the object rotates clockwise or counterclockwise about this axis, the points will be brought closer together in screen-space. Figure 5 demonstrates such a situation, in which the contact points will be matched either by rotating the object away from (5.ii) or towards (5.iii) the user.

Without any intervention, it is entirely up to how the solver chooses descent directions to decide between the two possibilities. However, by influencing the solver into favoring one solution over the other, we can mitigate the effects of this type of ambiguity.

5.1.1 Biasing The Solver to Resolve Ambiguities

To mitigate rotational ambiguities with a gradient-based solver, one can simply bias the solver by choosing a starting point that favors one direction over the other. A small perturbation from a maxima is all that is required for many descent-based algorithms to start descending down a particular path in the solution space. In our case this means selecting the axis which we would like to rotate counter-clockwise about, and using a small displacement in this direction as our starting point for rotational update DOF (e.g., quaternion values qx, qy, and qz.)

For example, in the situation presented in Figure 5, we can bias the solver into selecting the solution which rotates the cube towards the user by initializing qy to a small negative value (such as -10^{-1}) instead of zero. The solver will then start by considering solutions which include a clockwise rotation about Y. As a more complicated example, and one which takes the user interaction into account, we may wish to bias the solver according to the following rules: moving left on the screen most likely indicates a negative rotation about the Y-axis; moving to the right is a positive rotation. Similarly, moving upwards and downwards indicates a negative and positive rotation about the X-axis, respectively. (This particular choice of biases corresponds roughly to the types of rotations that can be done with an Arcball controller.) We can then bias the update to X and/or Y rotational updates by perturbing qx and/or qy such that descending the gradient from that point will cause the solution to contain the rotation we desire.

Note that the perturbation shouldn't have any adverse behavior away from local maxima. In this case the perturbation will either push the DOF towards the local minima or away from it. If the initial position is perturbed towards the local minima then the algorithm will converge as normal. If the perturbation is away from the local minima, then because we are away from a local maxima, and assuming that the perturbation is sufficiently small, we will also converge to the same solution. In this case the solver should descend the gradient from the perturbed point past the original pre-perturbed point, and wind up at the same local minima.

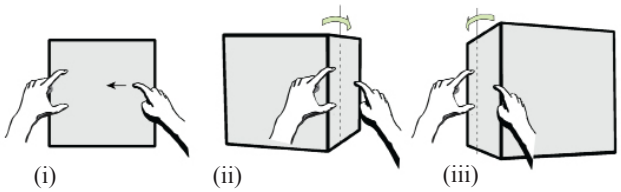

Figure 5: Illustration of an ambiguous rotation

5.1.2 Using Pressure to Resolve Ambiguities

We also experimented with using pressure to correct rotational ambiguities. Pressure provides users with an extra half-dimension (i.e., a non-negative value) which can be used in addition to the usual 2D motion along the surface of a touch-screen. We thus wished to give users a tool to

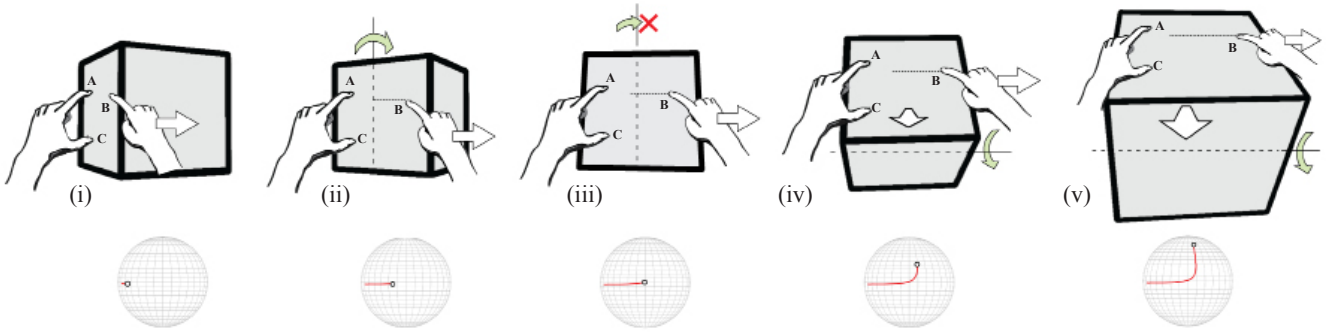

Figure 6: Illustration of an interaction exhibiting rotational exhaustion, and the path of the best-fit plane's normal

"push" a contact point further into the screen. Rotational ambiguities can then be resolved by pushing one side of the object into the screen when beginning a rotation.

One way pushing can be accomplished is by adding a penalty to our energy which tries to influence the solver into moving the point in the desired direction in world-space. The desired depth, z_{target}, can be determined from the current pressure value and the object's current world-space depth. We can then measure the contact point's deviation from the target depth via the quadratic penalty

$$\lambda(z_{target} - \langle \mathbf{z}, \mathbf{M}(\mathbf{q})\mathbf{x} \rangle)^2$$

in which $\mathbf{x}$ is the point to be pushed, $\mathbf{z}$ is the unit-length vector pointing into the screen, and λ is a weighting coefficient used to correct the difference in dimensionality between the world-space penalty and the screen-space energy. λ can be viewed as a trade-off factor of how much slippage among the contact points we are willing to accept for the point to pushed into the screen. For a more in depth discussion of penalties and a suitable choice of value for λ, please see section 7.

5.2 Rotational Exhaustion

Rotation about an axis can also increase the screen-space distance between a pair of contact points as they are moved apart by the user. However, this distance will eventually be maximized with respect to that axis. Referring to Figure 6, as point B is dragged to the right, a rotation about camera Y will increase the length of vector AB until its screen-space length is maximized (6.iii) with respect to that rotation. However, it is still possible for three contact points to be matched exactly, although the apparent DOF used to perform the matching will necessarily have to change.

Most commonly this means engaging a translation in Z (6.iv) which brings the object closer to the camera. However, as the object translates forward, the constraints on points A and C induce a new rotation roughly about AB, which is perceptually perpendicular to the previous axis of rotation. We wish to detect the point at which this rapid transition happens, and take some appropriate course of action to either control or correct the shift in interaction behavior.

5.2.1 Waxing and Waning Interactions

We can gain some insight into the problem by fitting a plane to the contact points and examining the path the plane's normal takes during a typical interaction exhibiting rotational exhaustion. Figure 6 (bottom) also depicts the path of the normal in the interaction shown. The normal initially starts rotating to become more and more parallel with the eye vector, and then takes a sharp turn and quickly becomes less and less parallel. The sharp bend in the normal's path characterizes a rotational exhaustion as the axis of rotation quickly changes from the initial axis to another. Note that in general the new axis is not necessarily perpendicular to the original, and the normal does not always become exactly parallel to the eye vector. Rotational exhaustion can occur from any starting orientation, rotating about any axis, not only those axes which lie in the X-Y plane. Regardless of object orientation though, all rotational exhaustions exhibit this type of bend in their normal path.

We refer to this phenomena as a waxing/waning interaction. When the plane's normal is becoming more and more parallel with the look vector, we say the plane is *waxing*. Similarly, when the plane's normal is becoming more and more orthogonal to the look vector, we say the plane is *waning*. It can be observed that the rapid shift in rotational axis occurs at the transition point from waxing to waning. That is, when the plane's normal is maximally parallel to the eye vector, and then starts turning away. Again, this does not necessarily mean that the two vectors are actually parallel, but rather that given the constraints set by the contact points, they won't get any more parallel then they are at this point.

To detect when this occurs, we define $w(\mathbf{x},\mathbf{q})$ to be the a function which measures the angle point $\mathbf{x}$'s normal makes with the eye vector at $\mathbf{x}$ given transform parameters $\mathbf{q}$, and look for minima in this function with respect to time. We evaluate this function at the centroid of the contact points using the normal of the best-fit plane. Assuming the world-space eye vector at the centroid can be computed from the transform values, and that we have the plane's normal in local-space, this gives us

$$w(\mathbf{x}_{centroid}, \mathbf{q}) = 1 - \langle \mathbf{eye}_{centroid}, \mathbf{M}(\mathbf{q})\mathbf{n}_{plane} \rangle$$

Minima in w are simply an indication that the plane's orientation has shifted from waxing to waning, but are not sufficient as an indicator of rotational exhaustion. In Figure 7, point B is moved in a small orbit, moving first to the right (7.ii), and induces a minima in w (7.iii, bottom) as point B starts to return toward points A and C. Such changes in

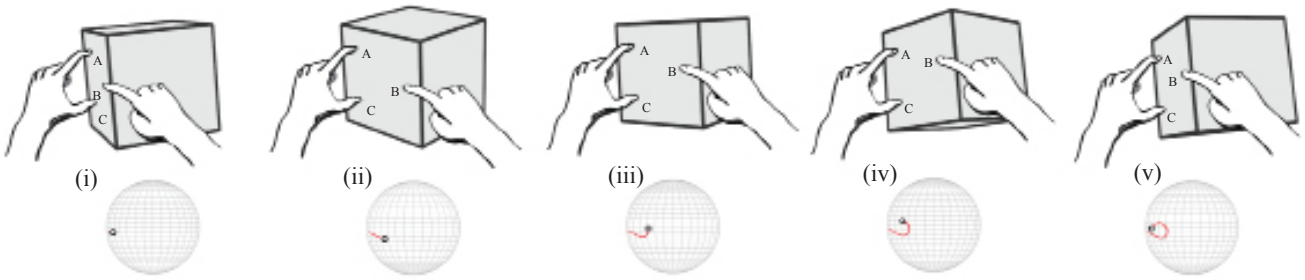

Figure 7: Illustration of a waxing/waning interaction which is not a rotational exhaustion

orientation can be expected when a control point reverses or otherwise modifies its current trajectory. The key difference in the case of rotational exhaustion (as shown in Fig. 6) is that the shift in orientation (and detected minima) occur as B is moved along a relatively consistent trajectory. Thus, we impose the additional criterion that each constraint point's trajectory should not deviate too much from a straight line.

Finally, it may be desirable to require that the change in w from a point's initial position to its current position be greater than some minimum threshold. This essentially requires that the surface point must have rotated by more than this minimum amount, and should help protect against false positives. Depending on the application, this may be important because if a minima in w is detected but the total change of w is small, then this likely indicates that the interaction started near a minima and thus may not qualify as a true rotational exhaustion.

5.2.2 Interaction Correction

Once detected, we experimented with "correcting" the interaction by limiting which DOF the minimization is allowed to operate upon. It would seem natural that since the interaction happens by exhausting rotational DOF that simply constraining all rotational DOF to their current values should be sufficient to correct the problem. Of course, when this is employed, the object stops rotating before unexpected behavior occurs. However, without any rotational DOF, no further changes to the object's orientation can be made. This is in stark contrast the types of manipulations which were possible immediately before the correction began.

For this reason we further experimented with restricting translational DOF instead of rotational ones. Since translation along camera Z is engaged once the original rotational component is exhausted, and this translation must work hand-in-hand with the new rotation in order for the contact points to be matched, restricting translation along camera Z seemed to be a viable alternative. We felt that this results in a more natural feel than restricting rotation, as the object can still move in subtle ways and many of the same manipulations are still possible regardless of whether translation along Z is enabled or not.

Finally, we must consider when to re-allow the solver to operate in all six DOF again. Ideally this should happen when the screen-space target points return close to their pre-slippage positions, as this is the most natural location for the user to "unwind" to. Fortunately we already have a means by which to measure the total slippage among contact points: our energy. If energy is increasing, it means that slippage is among the contact points is increasing as well. On the other hand, if energy is decreasing then the contact points must be moving closer towards the points on the object which were originally touched. We can therefore select a reasonable threshold that the energy must fall beneath at which point the missing DOF are activated again.

6 ERROR DISTRIBUTION

Whenever our camera is overconstrained, such as in the presence of greater than three contact points, when we add additional terms to our energy, or limit the available DOF as described in the previous section, we must necessarily incur some error. In fact, just removing a single DOF (e.g., translation along Z) will result in slippage among three or more contact points. Because our energy measures error in L^2, the error will tend to be distributed as evenly as possible among the contact points. As shown in Figure 8 (i), this results in none of the contacts matching their target positions exactly. While this may be sufficient for some applications, it is entirely possible that this is not desirable behavior for others.

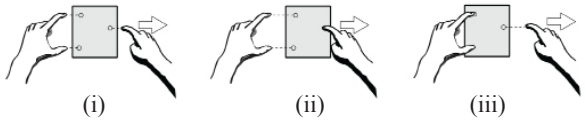

Figure 8: Three contact-point weighting schemes

By adding a normalized weighting coefficient to each contact point's terms in our energy, the distribution of error among the contact points can be controlled. This additional coefficient simply declares how "important" a particular point is relative to the other points, and thus the degree to which it will be matched during the optimization. If ω_i is the weight of the i^{th} contact, then with weighting coefficients included our energy becomes

$$E = \sum_i \omega_i \left\| s(\mathbf{x}_i, \mathbf{q}) - \mathbf{p}_i \right\|^2$$

$$\sum_i \omega_i = 1$$

The question of how to properly choose a weighting criteria still remains. The answer to this of course is application-

specific. However, in order to avoid assigning importance to the contact points in advance, we believe two general guidelines should be that the weights themselves can be derived from some measurable property of the interaction, as well as can easily be adjusted throughout. With these guidelines in mind, we experimented with four weighting criteria to control error distribution when the available DOF are limited by applying the rotational exhaustion "correction" described in section 5.2.2.

The first two schemes made use of the screen-space distance a contact has travelled from its initial position to define weights. Screen-space distances seemed well-suited for this purpose as a contact point's motion may be viewed as an indication of the user's intent. If a contact point moves quite a bit while the others remain relatively fixed, then the point in motion may be considered more important than the others, and thus should be matched more closely. If weighting is defined in this manner, then when translation along camera Z is disabled, the object will translate to follow the dominant hand's fingers across the screen while the non-dominant hand's contacts exert little influence on the object's motion. Thus, this weighting scheme effectively falls back on the single-finger translation one would expect from a 4DOF RST controller. This can be seen illustrated in Figure 8 (ii).

An alternative interpretation of using distance to define weights is to consider the stationary points more important than the ones in motion. In this case, the contacts of the non-dominant hand, which were used to create the rotational axis, pin the object down while the dominant hand's contacts are free to roam around the screen. An example of this is shown in Figure 8 (iii). As a result, the points in motion can exert only a small rotational influence on the pinned object, and thus this weighting scheme maintains the object's position and orientation at the time the correction is applied and translation along Z is disabled.

We also explored the use of pressure to control the spread of error. By defining a contact's weight in terms of how hard a user is pressing against the screen, we give the user a tool with which to pin points down. The harder a user presses at a contact relative to how they press at other contact points, the harder the solver will try to match that point. Because pressure values are continuous and easily adjusted, it is easy to dynamically re-weight contact points on the fly.

One difficulty with using pressure values to control error distribution arises from the fact that fingers are not styluses, and should not be expected to function in the same manner. When pressing a finger hard against a touch-screen and moving at the same time, friction may cause the interaction to be unpleasant and thus undesirable. This naturally makes pressure more well-suited for making stationary contact points sticky rather than trying to more heavily weight points in motion.

7 PENALTIES

Depending on the application, one may wish to constrain the object's motion to meet task-specific ends. Our method supports such constraints via quadratic penalties. Each penalty function simply measures deviation from some ideal condition. For example, in an application which allows a globe to be spun, one may wish to enforce the condition that the globe's center always remains constant in world-space. A penalty representing this would simply measure the distance from the globe's original center and sufficiently increase the energy if movement from this position should occur. With penalties included, the most general form of our energy can be written as

$$E = \sum_i \omega_i \left\| s(\mathbf{x}_i, \mathbf{q}) - \mathbf{p}_i \right\|^2 + \sum_j \lambda_j g_j(\mathbf{q})^2$$

in which $g_j(\mathbf{q})$ is the j^{th} penalty function with weight coefficient λ_j.

Penalties are not true, hard constraints in the strictest sense. However, through a careful choice of penalty terms one can often achieve the same result. Since each contact point adds exactly two terms to our energy, one and two-point manipulations will contain fewer terms than the six available DOF. Sophisticated one and two-point interactions can thus be created by using penalty terms to constrain the remaining DOF without sacrificing the exactness of the solution. Ideally the total number of terms in our energy should be close or equal to the number of DOF. While our method doesn't enforce a limit on the number of terms, the presence of large numbers of penalty terms will likely cause the interaction to noticeably degrade.

7.1 Weighting Penalties

When our energy is overconstrained (i.e., when the number terms in our energy exceed the number of available DOF), properly weighting the penalty functions may be required to ensure that each penalty continues to function sufficiently as a constraint. Choosing a value for λ to weight a screen-space penalty essentially informs the solver of the trade-off one is willing to accept between violating the penalty and the amount of the slippage one will allow the contact points to incur. The weight chosen indicates that a violation of one pixel is equivalent to λ pixels of slippage of an unweighted contact point, and if λ is large enough the solver should work harder to ensure that the penalty only receives an equivalently small amount of the error. Because our energy is quadratic in contact point slippage, in practice weighting screen-space penalties in the range of 10 to 10^4 seems to keep the penalty violations small and slippage to a minimum.

Choosing a value for λ for world-space penalties can be more challenging. Due to the difference in dimensionality, small violations of penalties in world-space are potentially much more disastrous than small violations of penalties in screen-space. A violation of one-pixel in screen-space may be barely noticeable to a user, but a violation of the same magnitude in world-space will likely be glaring. If is therefore crucial to weight the penalty so such gross violations cannot occur during the minimization. If an application can only tolerate a violation of 10^{-k} in world-space before violations become noticeable, one must set λ to be *at least* 10^{2k}. This is equivalent to the max tolerable violation in world-space

Figure 12: Single (top row) and two-point (bottom row) interactions on a terrain navigation system

being about one pixel of error in screen-space. Furthermore, in order for the penalty to carry even more significance than slippage among the contact points during the optimization, λ must be sufficiently increased beyond the lower bound of 10^{2k}. In practice a value of λ in the range of 10^{2k+1} to 10^{2k+4} seems to work well.

Finally, assuming the penalty is not violated at the start of the interaction, it is important not to set λ too large and overweight the penalty. If a penalty is overweighted, then any movement at all will result in penalty values which dwarf error associated with contact point slippage. As a result, the object will simply stop responding to input.

7.2 Penalties In Action

We experimented with a wide array of screen-space and world-space penalties. The following set seemed to be of the most value: vector fixed in world-space, point fixed in world-space, distance between two points fixed in world-space, point constrained to plane in world-space, direction between points fixed in screen-space, distance between points fixed in screen-space. Since our method presumes the presence of functions which project from local-space into both world-space and screen-space, each of these penalties can be easily implemented via dot products and distance computations in the appropriate space.

Once we had these penalties implemented, we were able to create rich interactions with just a few lines of code in our example implementation. For example, Figure 9 depicts a two-point interaction in which a penalty has been added to only allow rotation about the first contact point's normal. This single penalty enables two-point rotation about arbitrary 3D axes.

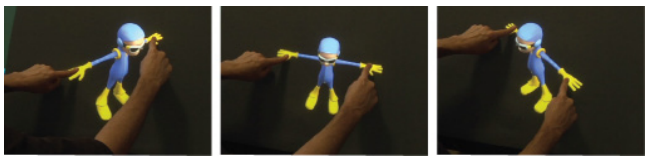

Figure 9: Rotation constrained about a particular axis

Multiple penalties can be used in tandem to produce more interesting interactions then a single penalty would allow alone. For instance, penalizing a point's deviation from a plane alone won't stop the rest of object from orienting itself in some arbitrary manner with respect to the plane. By additionally penalizing the rotation of the point's normal and tangent vectors, the object is forced to slide along the plane without any twisting. Figure 10 shows a single point interaction in which a cube is constrained to move in the plane perpendicular to the contact point's normal and twisting is disallowed.

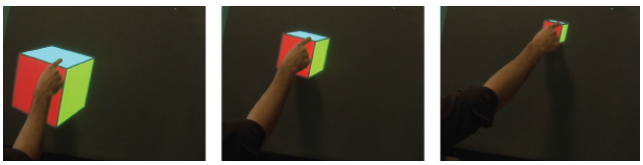

Figure 10: Translation along a plane without twisting

7.3 Navigation Tasks

If the object under control is much larger than the camera's field of view, then our method can also be applied to navigation tasks. Since the model responds to input in a very RST-like manner, users can leverage preexisting multi-touch experiences to perform new multi-touch tasks, such as traversal through virtual environments. Figure 11 shows a user performing RST-like interactions to navigate into a complicated model.

Figure 11: Traversing through a large model

For objects which are not free-floating, the addition of some simple penalties may be required to ensure the desired interaction. For example, by constraining the center of a globe to always remain at the same location in world-space, our method can readily be applied to globe navigation. For single-point interactions, we can make the globe rotate underneath the user's fingertip by penalizing changes in the distance between the camera and the center of the globe. Since the user can then spin the globe with a single finger without affecting the camera's distance to the globe, the presence of these two penalties effectively allow our method to mimic an Arcball controller. The interaction still behaves correctly at arbitrary camera orientations, such as the one shown in Figure 12 (top row).

For terrain navigation interactions involving two or more contact points, the distance penalty is dropped. The user

can zoom in and out with two fingers, and tilt the globe or change perspective with three or more fingers. Because our controller operates in screen-space, our method even permits novel terrain interactions, such as pinning one point on the surface down to a particular location on screen, and then spinning the earth around it. Figure 12 (bottom row) depicts this type of interaction, in which the user has pinned down a mountain and spins the world around it. Notice that the position of the mountain does not change on screen.

8 CONCLUSION AND FUTURE WORK

In this paper we have presented a screen-space method which allows direct control in 2D and 3D on a multi-touch surface. We suspect that the issues we have explored when working via direct manipulation in 3D are sufficiently general and apply not just to our particular formulation, but to any screen-space 3D direct manipulator. Despite these difficulties, which we believe are mitigable, there very likely exist many interesting applications for 3D direct manipulators.

There is still much left to do. We would like to continue expanding our understanding of the unexpected phenomena which popped up when directly manipulating in 3D. This includes exploring ways to make the interactions as predictable as possible, with as little intervention as possible. We would also like to expand the capabilities of our method. The addition of penalties specified by metrics such as the speed of the user's fingers may be useful in guiding the optimization into more faithfully capturing the user's intent. Additionally, methods to implement "undo" operations, by backtracking through solution space should also be looked into.

Finally, we would like to explore integration with optimization algorithms which allow hard constraints. This does not mean that we will discard our penalty framework. Rather, hard constraints and penalties can work in tandem to ensure desired interactions.

9 REFERENCES

1. Agarawala, A. and Balakrishnan, R. 2006. Keepin' it real: pushing the desktop metaphor with physics, piles and the pen. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (Montréal, Québec, Canada, April 22 - 27, 2006). R. Grinter, T. Rodden, P. Aoki, E. Cutrell, R. Jeffries, and G. Olson, Eds. CHI '06. ACM, New York, NY, 1283-1292.

2. Chen, M., Mountford, S. J., and Sellen, A. 1988. A study in interactive 3-D rotation using 2-D control devices. In *Proceedings of the 15th Annual Conference on Computer Graphics and interactive Techniques* R. J. Beach, Ed. SIGGRAPH '88. ACM, New York, NY, 121-129.

3 Cutler, L. D., Fröhlich, B., and Hanrahan, P. 1997. Two-handed direct manipulation on the responsive workbench. In *Proceedings of the 1997 Symposium on interactive 3D Graphics* (Providence, Rhode Island, United States, April 27 - 30, 1997). SI3D '97. ACM, New York, NY, 107-114. DOI= http://doi.acm.org/10.1145/253284.253315

4. Fröhlich, B., Tramberend, H., Beers, A., Agrawala, M., and Baraff, D. 2000. Physically-Based Manipulation on the Responsive Workbench. In Proceedings of the IEEE Virtual Reality 2000 Conference (March 18 - 22, 2000). VR. IEEE Computer Society, Washington, DC, 5.

5. Gleicher, M. and Witkin, A. 1992. Through-the-Lens Camera Control. In *Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques* J. J. Thomas, Ed. SIGGRAPH '92. ACM, New York, NY, 331-340.

6. Grossman, T., Wigdor, D., and Balakrishnan, R. 2004. Multi-finger gestural interaction with 3d volumetric displays. In *Proceedings of the 17th Annual ACM Symposium on User interface Software and Technology* (Santa Fe, NM, USA, October 24 - 27, 2004). UIST '04. ACM, New York, NY, 61-70. DOI= http://doi.acm.org/10.1145/1029632.1029644

7. Hancock, M., Carpendale, S., and Cockburn, A. 2007. Shallow-depth 3d interaction: design and evaluation of one-, two- and three-touch techniques. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (San Jose, California, USA, April 28 - May 03, 2007). CHI '07. ACM, New York, NY, 1147-1156.

8. Kyung, M., Kim, M., and Hong, S. J. 1996. A new approach to through-the-lens camera control. *Graph. Models Image Process*. 58, 3 (May. 1996), 262-285.

9. Nielson, G. M. and Olsen, D. R. 1987. Direct manipulation techniques for 3D objects using 2D locator devices. In *Proceedings of the 1986 Workshop on interactive 3D Graphics* (Chapel Hill, North Carolina, United States). F. Crow and S. M. Pizer, Eds. SI3D '86. ACM, New York, NY, 175-182

10. Nocedal, J. and Wright, M. 1999. Numerical Optimization. Springer, New York

11. Shoemake, K. 1992. ARCBALL: a user interface for specifying three-dimensional orientation using a mouse. In *Proceedings of the Conference on Graphics interface '92* (Vancouver, British Columbia, Canada). K. S. Booth and A. Fournier, Eds. Morgan Kaufmann Publishers, San Francisco, CA, 151-156.

12. Snibbe, S. S., Herndon, K. P., Robbins, D. C., Conner, D. B., and van Dam, A. 1992. Using deformations to explore 3D widget design. In Proceedings of the 19th Annual Conference on Computer Graphics and interactive Techniques J. J. Thomas, Ed. SIGGRAPH '92. ACM, New York, NY, 351-352.

13. Wilson, A. D., Izadi, S., Hilliges, O., Garcia-Mendoza, A., and Kirk, D. 2008. Bringing physics to the surface. In *Proceedings of the 21st Annual ACM Symposium on User interface Software and Technology* (Monterey, CA, USA, October 19 - 22, 2008). UIST '08. ACM, New York, NY, 67-76