



A High Quality 3D Controller for Mobile and Desktop Web Applications

D. Fornari¹, L. Malomo² , Paolo Cignoni² 

¹University of Pisa, Italy

²ISTI - CNR, Pisa, Italy

Abstract

The interaction between a 2D input device (like a mouse or a touchscreen) and a 3D object on the screen with the purpose of examining it in detail is a well-studied interaction problem. The inherent difference in degrees of freedom between input devices and possible 3D transformations makes it difficult to intuitively map inputs to operations to be performed on 3D objects. Although, over the years, studies led to a wide variety of solutions to overcome this problem, most of them are not actually available in real-world applications. In particular, for 3D web applications, only basic solutions are often implemented, and even the most used web framework for 3D still lacks state of the art implementations. We will face the problem of 3D interaction through touch and mouse input, and we propose our implementation of a 3D view manipulator for web applications, which offers a natural control, advanced functionalities, and provides an easy-to-use interface for both desktop and mobile environments.

1. Introduction

Efficient and intuitive tools for 3D content manipulation are often essential components for all use cases where the user needs to inspect/explore a virtual object on the screen. These tools must succeed in giving the user the ability to control the spatial transformations of a 3D object and thus look interactively at different details of the displayed object in a simple, intuitive way.

In practice, we want to perform *rotate*, *scale* and *translate* (RST) operations, involving a total of 7 *degrees of freedom* (DOFs). However, this rich freedom conflicts with the just two degrees of freedom of screen based input devices, like mouse and touchscreen, and this gap does not enable a direct mapping between an input and a 3D transformation. While literature has provided many sophisticated solutions, briefly reported in Section 2, and while many of them have been implemented in commercial desktop applications, for web based applications the situation is still different. Most existing web frameworks for 3D offer very limited 3D interaction techniques and they often lack in providing a consistent mouse/touch support. For this reason we developed a novel 3D view controller for three.js [thr], which is currently the most used framework to build 3D web applications. The newly developed controller uses techniques which are considered to be the actual state of the art and aim to overcome the limitations of the controllers already provided by this library, providing advanced navigation functionalities and full touch support through an intuitive and natural interface.

2. State of the art

In theory, RST operations could be performed in a direct way, by manually modifying an object's transformation matrix, but obviously such an approach is not reasonable. Over the years a large amount of research has been dedicated to make 3D manipulation more intuitive by exploiting the available input devices. In the following we will mention only a few techniques that have been proposed in recent years; for a longer survey of existing approaches we refer to the literature in [RDH09, MCG12, MLF11].

The virtual trackball [Bel88] represents a milestone for rotation techniques. This manipulation method simulates the behavior of a real trackball device to control the orientation of an object, granting control over three rotational degrees of freedom using an input device with just two degrees of freedom. This is done by unprojecting the 2D screen coordinates of the cursor onto a virtual spherical surface in the 3D world (the trackball surface). Given two different cursor positions in screen space, p_a to p_b , their corresponding 3D points on this virtual surface, P_a and P_b , are computed. Then, from these two points, rotation axis and angle are computed as the cross product between $V_a = OP_a$ and $V_b = OP_b$, where O is the center of the virtual sphere, and the angle between V_a and V_b (see Figure 1).

Sticky Tools is a multi-touch manipulator which combines of three other manipulation paradigms: *Sticky Fingers*, *Opposable Thumbs* and *Virtual Tools* [HiCC09]. Once a finger gets in touch with an object, it remains "glued" to it so when the finger moves, the object follows it by maintaining the contact point. The *translate* operation (Figure 2a) is performed by dragging one finger, the *scale* operation (Figure 2c) can be done with a two-finger pinch

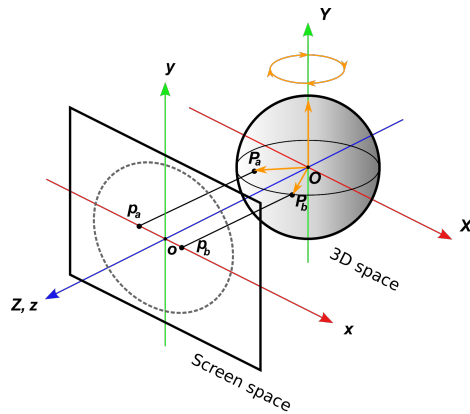


Figure 1: Axis and angle obtained by the unprojection of the cursor on the virtual trackball surface.

gesture and the *rotate* operation can be performed either by rotating two fingers on the screen (Figure 2b), which controls rotation around the Z axis or by placing two fingers on the screen and using a third to “flip” the object around the axis determined by the first two fingers (Figure 2d).

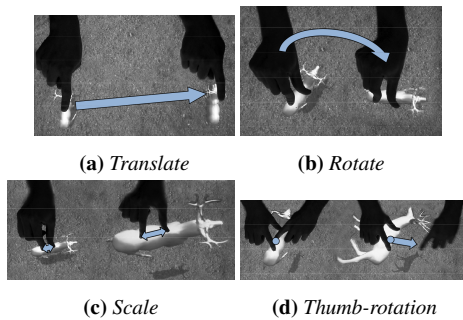


Figure 2: Performing operations with Sticky Tool. (a), (b) and (c) come from Sticky Fingers technique while (d) comes from Opposable Thumb method.

TouchSketch [WCOM15] is instead a widget-based manipulator for multi-touch devices, which exploits fingers’ movement on the screen to determine the manipulation mode and to control the transformation, along with a widget to specify an axis or plane constraint (Figure 3).

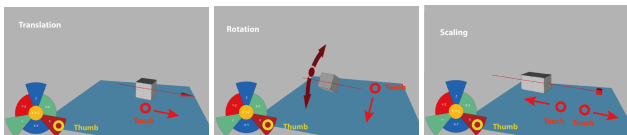


Figure 3: Performing RST operations with TouchSketch. Image from [WCOM15]

As suggested in the Introduction, we want to improve over the camera manipulation tools available in the three.js framework. This

framework currently offers two main controllers to perform interactive camera manipulation. The first controller is *OrbitControls*, which is a plain turntable camera manipulator that converts mouse x/y movement into rotation around X/Y axes. The rotation around Y is free, while the “up-down” rotation around X axis is constrained to avoid that the camera goes over the “north pole” or “south pole”. The other camera manipulator is *TrackballControls*, which behaves similarly to *OrbitControls*. The main difference is that rotation around the X axis is not constrained, but it suffers of bad cursor tracking; this results in a noticeably bad synchronization between cursor movement and the corresponding operation which is particularly annoying while performing a *rotate* operation. With respect to our proposed implementation, both manipulators lack rotation control around the Z axis and they are not “conservative” with respect to the *rotate* operation (i.e. returning to the starting point will not make the camera to return to its starting orientation). Moreover, they do not provide any additional features other than common RST operations.

3. Design and implementation

To provide a good general purpose 3D manipulator (controller) many design decisions should be taken to guarantee consistency and intuitiveness. The first choice to make when dealing with 3D manipulations is to pick between two dual metaphors to model the transformations: object vs. camera manipulation. In the first case the controller actually rotates the 3D object, while in the second case the camera moves around the object to actually implement the transformations. While from the user point of view the difference can be irrelevant, from the point of view of a developer designing a 3D interactive application the difference is significant. We opted for the second case for consistency with other three.js manipulators. As a consequence, we designed our controller API adopting a mimicking strategy, to eventually allow developers to replace, in the simplest manner as possible, the three.js manipulator in use with ours. In the following we describe some details of the basic transformations actually implemented in our manipulator.

In our controller the *rotation* is performed by clicking and dragging, and is achieved by using the virtual trackball technique. We chose the Bell’s version [Bel88] which, instead of a sphere, considers a surface which combines a hemisphere with a hyperbolic sheet, to extend the cursor mapping to the entire display (Figure 4). The unprojection operation consists in firing a ray from the camera towards the cursor position projected on the camera near plane. As result, we obtain a straight line passing through two points, which intersects the trackball surface to find the unprojected point (Figure 5).

The *translate* operation is performed with the right mouse button clicking and dragging. The cursor position is unprojected on a plane passing from the center of the trackball which is orthogonal to the view direction (the trackball plane). In a similar way to the virtual trackball, once the interaction starts, the initial cursor position p_a is unprojected to the trackball plane, obtaining the 3D point P_a . Then, every time the cursor is moved to a point p_b , its unprojection on the plane P_b is computed. To avoid drifting, the translation vector is obtained as the difference $\vec{T} = P_b - P_a$, which is applied every time to the initial transformation (Figure 6).

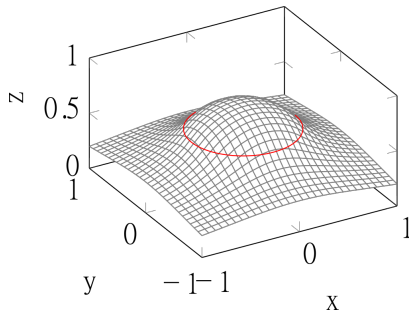


Figure 4: 3D representation of the trackball surface obtained by combining a sphere and a hyperbolic sheet.

The *zoom* operation is triggered with mouse wheel scroll and takes into account which type of camera is being used. A perspective camera simulates the human sight, where things closest to our eyes appear bigger. For this camera, zoom can be obtained by moving the camera along its view direction. On the other hand, with an orthographic camera, the distance from the camera does not impact the object appearance: in this case, zoom is obtained by modifying the camera’s projection matrix. In both cases we take care to avoid the common error of mapping linear movement into linear scaling, an approach that, while common, does not reflect the inherently geometric/exponential nature of the scaling operation.

3.1. Additional features

Besides the standard RST operations, two additional functionalities have been added to provide easier and faster 3D object inspection. The *focus* operation is activated with a double click and can be used to automatically set the focus in a user specified region on the object’s surface. It combines *translation* and *zoom* operation to set the target point on the object exactly in the middle of the trackball, which is a convenient position if the user wants to inspect that region. The other operation is the *field of view* (FOV) manipulation and it’s activated using the shift key + mouse wheel scrolling. The

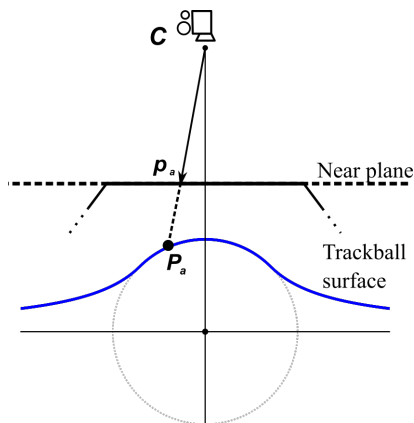


Figure 5: Cursor point unprojection on the virtual trackball surface.

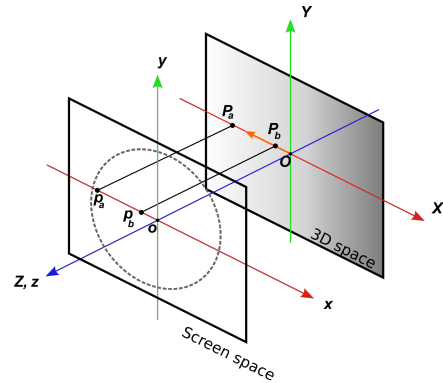


Figure 6: Translation vector (orange) obtained from the unprojection of the cursor on the trackball plane.

operation can be performed only when using a perspective camera and it’s implemented with a “vertigo-style” method: the camera is moved away from the object as the FOV angle is reduced, while it is brought closer when the FOV is enlarged.

3.2. Multi-touch interaction

While previous sections describe the behavior of mouse/keyboard interaction, in the following we describe how we naturally extended the our controller interaction scheme for multi-touch inputs. With tactile input devices, the user feels a sensation of real manipulation of the object, considering that the interaction is not mediated by some external device, but the user is actually “touching” the object. For this reason, providing a natural and realistic sense of manipulation is a critical aspect for this kind of interaction. In our controller implementation the operations are recognized according to the number of fingers on the screen and the gesture being performed.

One-finger drag is used to perform the *rotate* operation. The implementation is the same as for mouse/keyboard interaction (Figure 7a).

With a double tap gesture, the user can specify a point for the *focus* operation (Figure 7b).

Two-finger drag is used to perform the *translate* operation. To guarantee consistence between finger and object movement, the midpoint between the two fingers is taken into account and is unprojected on the trackball plane as for the mouse/keyboard *translate* operation.

The *zoom* is performed with a pinch gesture (shrink or pull apart). On the first interaction, the distance between fingers Δ_{fs} is stored, then, as the fingers distance changes, the amount of scaling is computed as $s = \frac{\Delta_{fs}}{\Delta_{fc}}$, with Δ_{fc} representing the current distance. In this type of interaction, the screen is seen as an elastic sheet, so when the user pulls apart fingers, the region between them is expanded (or reduced in case of shrink). This effect is obtained by unprojecting the midpoint between fingers on the trackball plane as for *translate* operation and using it as a pivot point for the scaling.

An additional operation, which can be performed only with multi-touch interaction, is the *Z-rotation*. Since the point of contact be-

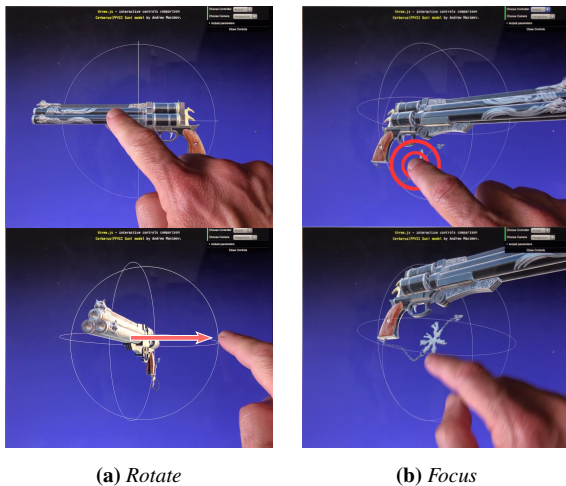


Figure 7: One-finger interaction.

tween fingers and the object should always be maintained, a two-finger rotation causes the object to rotate around the Z axis, with the pivot point being the middle point between two fingers.

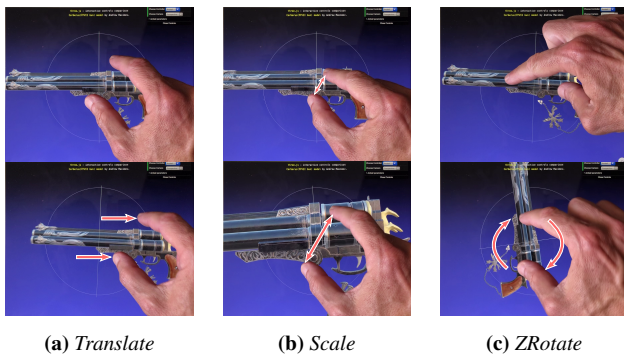


Figure 8: Two-fingers interaction for achieving combined Translation, Scaling and Rotation along the view direction (z axis).

Lastly, a vertical drag with three or more fingers is used to achieve the *field of view* manipulation, that allows to change the current camera parameters back and forth between wide-angle and telephoto lenses.

3.3. User experience improvements

To provide the user with an enjoyable experience is another key aspect that has been taken into consideration while developing this tool. To give the user a good feedback of the trackball rotation, we provided visual feedback for the 3D trackball sphere which is represented by three circular gizmos, each one orthogonal to one axis (X , Y and Z) and matching exactly the trackball sphere radius and position (Figure 10). We also made sure to provide a smooth transition between operations: in any moment, the user can swap from an operation to another without the need of interrupting it.



Figure 9: Field of view manipulation using three fingers.

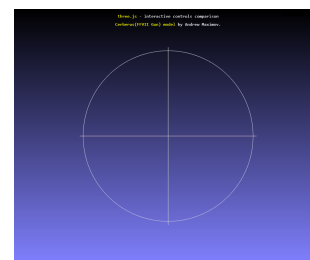


Figure 10: The gizmo of the virtual sphere represents the surface where drag actions are mapped into rotations and allows a better understanding of the interaction effects.

Additionally, mouse/keyboard inputs are fully customizable: developers can set or remove an association between an input combination and an operation to be performed anytime, providing thus a comfortable input system for 3D applications with different needs. Providing an informative feedback is also important for a pleasant user experience as users enjoy to see virtual elements behave like they do in the real world. For this reason, we introduced a dynamic damping effect for the *rotate* operation. Animation has also been added for the *focus* operation. To instantly translate an object from a position to another can result confusing. To avoid this undesirable effect, a cubic ease-out interpolation is used to perform a smooth translation when performing this operation. Animations are enabled by default and can be disabled using the provided API. Another interesting feature available was motivated by the common need of saving the current view state. We exposed this functionality in a very intuitive way, exploiting the clipboard. The user can save and re-apply the view state with the classical `ctrl+c` `ctrl+v` key combinations. The implementation is performed by serializing the views state in a human-readable json text which, interestingly enough, enables many practical possibilities such as saving the view state in a file for later use or even passing view state between different instances/web pages.

4. Conclusions

Our main goal was to provide the three.js developer community a state of the art manipulator that could enhance the user expe-

rience with respect to 3D object inspection. Although *OrbitControls* performs well in simple cases, and *TrackballControls* can be used as an alternative where full control over rotation around X axis is required, these controllers have severe limitations, negatively affecting the user experience. Thanks to an intuitive interaction scheme, which allow the user to perform *rotation* over all three axes in a precise way, and the advanced navigation functionalities, we strongly believe that our manipulator will result more versatile and pleasant to use in all situations. Thanks to an intuitive interaction scheme built for classic mouse/keyboard and touch interaction and advanced navigation functionalities, our manipulator is a powerful tool which outperforms other controllers and overcomes their limitations. The manipulator implementation has been submitted to the three.js developer community and it has been received enthusiastically. It has been already included the official three.js codebase. A simple demo showing the capabilities of the controller can be found at the link: https://threejs.org/examples/#misc_controls_arcball.

References

- [Bel88] BELL G.: Bell's trackball. Written as part of the "flip" demo to demonstrate the Silicon Graphics hardware, 1988. 1, 2
- [HiCC09] HANCOCK M., TEN CATE T., CARPENDALE S.: Sticky tools: Full 6dof force-based interaction for multi-touch tables. In *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces* (New York, NY, USA, 2009), ITS '09, Association for Computing Machinery, p. 133–140. doi:10.1145/1731903.1731930. 1
- [MCG12] MARTINET A., CASIEZ G., GRISONI L.: Integrality and separability of multitouch interaction techniques in 3d manipulation tasks. *IEEE Transactions on Visualization and Computer Graphics* 18, 3 (2012), 369–380. doi:10.1109/TVCG.2011.129. 1
- [MLF11] MENDES D., LOPES P., FERREIRA A.: Hands-on interactive tabletop lego application. In *Proceedings of the 8th International Conference on Advances in Computer Entertainment Technology* (New York, NY, USA, 2011), ACE '11, Association for Computing Machinery. doi:10.1145/2071423.2071447. 1
- [RDH09] REISMAN J. L., DAVIDSON P. L., HAN J. Y.: A screen-space formulation for 2d and 3d direct manipulation. In *Proceedings of the 22nd Annual ACM Symposium on User Interface Software and Technology* (New York, NY, USA, 2009), UIST '09, Association for Computing Machinery, p. 69–78. doi:10.1145/1622176.1622190. 1
- [thr] three.js javascript 3d library. <https://threejs.org/>. Online; accessed 20-October-2021. 1
- [WCOM15] WU S., CHELLALI A., OTMANE S., MOREAU G.: Touchsketch: A touch-based interface for 3d object manipulation and editing. In *Proceedings of the 21st ACM Symposium on Virtual Reality Software and Technology* (New York, NY, USA, 2015), VRST '15, Association for Computing Machinery, p. 59–68. doi:10.1145/2821592.2821606. 2