

Copyright

by

Andrew David Slininger

2011

**The Report Committee for Andrew David Slininger
Certifies that this is the approved version of the following report:**

**Application of Single and Multi-touch Gestures in a WebGL Molecule
Viewer**

**APPROVED BY
SUPERVISING COMMITTEE:**

Supervisor:

Chandrajit Bajaj

Kelly Gaither

**Application of Single and Multi-touch Gestures in a WebGL Molecule
Viewer**

by

Andrew David Slininger, B.S.Biomed.E.

Report

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Engineering

The University of Texas at Austin

August 2011

Acknowledgements

I would like to thank Dr. Bajaj for continually meeting with me and keeping me on topic. My report would have been extremely unfocused and rambling without his help. I would also like to thank Brandt Westing. He lent me his expertise and use of the Texas Advanced Computing Center's formidable equipment including a large touch sensitive perimeter device.

Abstract

Application of Single and Multi-touch Gestures in a WebGL Molecule Viewer

by

Andrew David Slininger, MSE

The University of Texas at Austin, 2011

SUPERVISOR: Chandrajit Bajaj

The number of devices with touch input such as smart phones, computers, and tablets has grown extensively in recent years. Native applications on these devices have access to this touch and gesture information and can provide a rich, interactive experience. Web applications, however, lack a consistent and uniform way to retrieve touch and gesture input. With the quality and robustness of web applications continually growing and replacing native applications in many areas, a way to access and harness touch input is critical. This paper proposes two JavaScript libraries that provide a reliable and easy way for web applications to use touch efficiently and effectively. First, `getTjs` abstracts the gathering of touch events for most mobile and desktop touch devices. `GenGesjs`, the second library, receives this information and identifies gestures based on the touch input. Web applications can have this gesture information pushed to them as it is received or instead request the most recent gestures when desired. An example of interfacing with both libraries is provided in the form of `WebMol`. `WebMol` is a web

application that allows for three dimensional viewing of molecules using WebGL. Gestures from GenGesjs are translated to interactions with the molecules, providing an intuitive interface for users. Using both of these libraries, web applications can easily tap into touch input resulting in an improved user experience regardless of the device.

Table of Contents

List of Tables	viii
List of Figures	ix
Chapter 1: Introduction	1
Background	1
Mobile touch devices	1
Desktop touch devices	3
Chapter 2: Touch Implementation	5
Chapter 3: Gestures	6
Chapter 4: Gesture Implementation	12
Chapter 5: Application	16
Tap	17
Double Tap	17
Drag	18
Flick	18
Pinch	18
Spread	18
Press	19
Press and Tap	19
Press and Drag	19
Rotate	19
Chapter 6: Future Work	20
Chapter 7: Conclusion	22
Appendix – getTjs, GenGesjs pseudo code and implementation	23
References	24

List of Tables

Table 1: W3C Touch Events Draft.	2
Table 2: W3C Touch Event Attributes Draft.	2
Table 3: Devices and the methods getTjs uses to extract touch movement.....	5
Table 4: Tap Gesture.....	6
Table 5: Double Tap Gesture.....	7
Table 6: Drag Gesture.	7
Table 7: Flick Gesture.....	8
Table 8: Pinch Gesture.....	8
Table 9: Spread Gesture.....	9
Table 10: Press Gesture.	9
Table 11: Press and Tap Gesture.	10
Table 12: Press and Drag Gesture.....	10
Table 13: Rotate Gesture.	11

List of Figures

Figure 1: TUIO output on port 3333 showing two touch points.....	4
Figure 2: getTjs and GenGesjs implementation overview for a mobile device.....	12
Figure 3: getTjs and GenGesjs implementation for a desktop touch device.	13
Figure 4: Mouse event equivalent of gestures	14
Figure 5: A molecule rendered in WebMol.	16
Figure 6: Axes comparison before and after a perspective change. Note the viewport axes do not change.	17

Chapter 1: Introduction

BACKGROUND

Devices with touch user interfaces have become very prevalent, largely fueled by the spread of smartphones and tablets. Native applications and operating systems running on these devices have harnessed the power of these input devices to provide a richer and more interactive user experience. A second trend is the growth of powerful web applications that have replaced many desktop applications. These robust web applications have several benefits over their desktop counterparts, one in particular being that they are accessible across a multitude of devices assuming a nearly ubiquitous internet connection is available. Combining the two has obvious benefits; pair the interactivity of multi-touch with rich, accessible web applications. Implementing this to work with all touch devices, however, requires a two-step approach.

MOBILE TOUCH DEVICES

Mobile touch devices are built from the ground up to support touch as often this is the only user input device available. This native support translates to JavaScript navigation events that are fired much like mouse events supported by all modern browsers. The iOS browser even supports gesture events that can return common gestures such as rotate or pinch to zoom¹. However, there is no standardization of these events between mobile operating systems and the W3C Web Events Working Group is just now publishing a first draft². Fortunately this draft follows, to a degree, the already implemented events by the most common mobile devices (Android, iOS).

¹ Apple Inc., “Gesture Event Class Reference”, Apple Inc., http://developer.apple.com/library/safari/#documentation/UserExperience/Reference/GestureEventClassReference/GestureEvent/GestureEvent.html#apple_ref/javascript/cl/GestureEvent (accessed July 22, 2011).

² W3C, “Touch Events Specification”, W3C, <https://dvcs.w3.org/hg/webevents/raw-file/tip/touchevents.html> (accessed July 24, 2011).

Event	Description
touchstart	Event when a new touch point is placed on the touch surface
touchend	Event when a touch point is removed from the touch surface
touchmove	Event when a touch point is moved along the touch surface
touchenter	Event when a new touch point enters a special area defined by the web page
touchleave	Event when a new touch point leaves a special area defined by the web page
touchcancel	Event when a touch point is lost, either intentionally or unintentionally

Table 1: W3C Touch Events Draft.

Attribute	Description
clientX	X-coordinate of point relative to the viewport, excluding any scroll offset
clientY	Y-coordinate of point relative to the viewport, excluding any scroll offset
force	The force of the touch event
identifier	A value that is unique for each touch point. Allows tracking touch points over time.
pageX	X-coordinate of point relative to the viewport, including any scroll offset
pageY	Y-coordinate of point relative to the viewport, including any scroll offset
radiusX	The radius of the ellipse which circumscribes the touch area along the x-axis, in pixels of the same scale as screenX
radiusY	The radius of the ellipse which circumscribes the touch area along the y-axis, in pixels of the same scale as screenY
rotationAngle	The angle (in degrees) that the ellipse described by radiusX and radiusY is rotated clockwise about its center
screenX	X-coordinate of point relative to the screen
screenY	Y-coordinate of point relative to the screen

Table 2: W3C Touch Event Attributes Draft.

The events of most interest are touchmove, touchend, and touchstart. These are already implemented in the Android and iOS browsers. They fire every time a touch input is detected and returns an array of events, one for each touch point. Using the event's identifier and x and y coordinates over time, gestures can be determined.

DESKTOP TOUCH DEVICES

While not nearly as widespread as mobile touch devices, desktop touch devices are becoming much more common. This is a growing field with a number of new devices hitting the market.

Touch events on desktop devices are much more difficult to capture in the web browser. Unlike mobile devices, they are not passed along from the OS. This means the JavaScript events such as touchmove are not supported and never fire. Instead, these must be retrieved from the touch input device's drivers. The difficulty is that for security reasons the browser restricts access to the local file system. This prevents code from a malicious website disrupting anything on the user's computer; it is contained in a sandbox. A browser specific plugin or a flash helper is the most common way to gain access to the local system to read or parse in touch events. These must be installed by the user and aren't restricted to the browser's sandbox.

The Microsoft Kinect is one such device that uses a camera in combination with an infrared projector/camera to track user input. Depthjs is a project that uses a JavaScript library along with a browser plugin to push Kinect movements to the browser³. Touch sensitive perimeter devices use an infrared frame around the perimeter of the screen to detect input. Many of these devices use a framework known as TIUO to provide an API for capturing touch events. Events are broadcast as an array on the local

³ Aaron Zinman and others, "Depthjs", MIT Media Lab, <http://depthjs.media.mit.edu/> (accessed July 19, 2011).

host's port 3333. Each touch point is represented as an element in the array and contains its ID, X and Y coordinates relative to the screen, speed, and acceleration. Accessing these events in a web browser requires a browser specific plugin, npTuioClient, along with a JavaScript library, TUIOjs ⁴.

```
set cur 0 <22> 0.7583333 0.66 14.03567 14.03567
set cur 1 <21> 0.6683334 0.42 14.03567 14.03567
set cur 0 <22> 0.7166666 0.65 25.31798 25.31798
set cur 1 <21> 0.71 0.43 25.31798 25.31798
set cur 0 <22> 0.6966667 0.63 14.4222 14.4222
```

Figure 1: TUIO output on port 3333 showing two touch points.

⁴ TUIO, "Software Implementing TUIO", TUIO, <http://www.tuio.org/?software> (accessed July 15, 2011).

Chapter 2: Touch Implementation

Due to the differences in touch input devices and how touch events are gathered, the best route to implementing a universal touch input web application is to abstract the gathering of these touch events to a JavaScript library, getTjs. This library will hide the constantly changing browser touch support from the creation of gestures. It will gather events from the three groups of devices described earlier: mouse, mobile touch, and desktop touch.

Device	Method
Mouse	JavaScript mouse events
Mobile Touch	JavaScript touch events
Desktop Touch	Browser Plugin and JavaScript library to return TUIO server running on the host

Table 3: Devices and the methods getTjs uses to extract touch movement.

Every time a touch movement is detected getTjs will extract the touch movements into an array. Each element of the array will be an object with an X and a Y value representing the location of the touch point on the screen. In the case of only having a mouse available, this array will only have one element. getTjs will pass this array to another JavaScript library or to the application by calling a function that must be implemented, buildgestures. If this function is not implemented, getTjs will continue to parse the events harmlessly without passing them along. Even though more detailed touch information is supported by several of the touch devices, this is enough information to build gestures and basic enough that all touch devices can provide it.

Chapter 3: Gestures

Gestures are a way of translating touch events controlled by the user into meaningful commands for the application. Most mobile touch devices implement a number of gestures for OS navigation and native applications. Very few provide any support in the browser so this must be handled by the web application. There are many different gesture definitions but the most thorough is the “Touch Gesture Reference Guide” by Luke Wroblewski⁵. The core set of gestures described can be implemented in a very useful way for browser interaction (Tables 4-13).


Name	Tap
Description	A single touch that lasts a short period of time. This is analogous to a left mouse click.
Uses	A select command
Image	

Table 4: Tap Gesture⁵.

⁵ Luke Wroblewski, “Touch Gesture Reference Guide”, LukeW Ideation + Design, <http://www.lukew.com/ff/entry.asp?1071> (accessed July 15, 2011).


Name	Double Tap
Description	A short touch and release followed by another short touch and release in the same location. If too much time passes between touches it will be interpreted as two separate tap gestures. This is analogous to a double left click with a mouse.
Uses	An emphasized select command. It can be used to bring an element into focus or to open it.
Image	 A line drawing of a hand with the index finger touching a circular target icon consisting of three concentric circles.

Table 5: Double Tap Gesture⁵.


Name	Drag
Description	A single touch and then slow movement without loss of contact.
Uses	If started at the position of an element on the screen, it can often represent movement of this element to another location. If the element is three dimensional and anchored to an axis, it could represent rotation around that axis. It can also adjust a variable attached to a slider scale such as increasing volume.
Image	 A line drawing of two hands. The left hand is positioned on the left, and the right hand is on the right. A thick black arrow points from the left hand to the right hand, indicating the direction of the drag movement.

Table 6: Drag Gesture⁵.


Name	Flick
Description	A single touch followed quick continuous movement, usually in a single direction. The movement speed must be above a certain threshold to distinguish it from a drag.
Uses	Moves an element out of focus or to some other partition in the application.
Image	

Table 7: Flick Gesture⁵.


Name	Pinch
Description	Two touch points made at the same time and both moved toward each other at a similar speed.
Uses	Zooming out the view of the application. If an element is in focus or if the pinch is performed directly on top of an element it will be scale it down to a smaller size.
Image	

Table 8: Pinch Gesture⁵.


Name	Spread
Description	Two touch points made in close vicinity and then both moved in opposite directions at a similar speed.
Uses	Zooming in the view of the application. If an element is in focus or if the pinch is performed directly on top of an element it will be scale it up to a larger size.
Image	

Table 9: Spread Gesture⁵.


Name	Press
Description	A single touch point in one location that is held. Must last longer than a certain threshold or be interpreted as a tap.
Uses	An emphasized select command, analogous to a right click from a mouse. It is often used to bring up secondary menus or options for the element selected.
Image	

Table 10: Press Gesture⁵


Name	Press and Tap
Description	A press followed by a tap while the press is still being held.
Uses	An emphasized select. It can also move an element from the press location to the tap location.
Image	

Table 11: Press and Tap Gesture⁵.

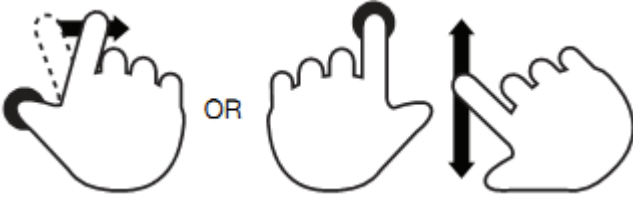
Name	Press and Drag
Description	A single touch followed by a drag made while the touch is still being held.
Uses	A drag command that is focused on a specific element. If a drag will perform an action on a group elements or the entire application, the press and drag will only perform the action on the element that is selected by the press
Image	

Table 12: Press and Drag Gesture⁵.

Name	Rotate
Description	A circular gesture performed by two touch points. Rotate can be performed in several ways. One is by making two touch points separated by a small distance and moving them both in a clockwise or counter-clockwise motion. Two presses where one is rotated in a clockwise or counter-clockwise motion while the other stays stationary is a second way. Lastly, two presses made in close vicinity and moved in the same clockwise or counterclockwise motion can represent a rotate. The two presses distinguish it from an arcing drag.
Uses	Rotate either the view of the application or an element. The rotate technique involving a held press with one touch point and rotate with the other in particular will rotate the element selected by the press.
Image	<p>The image shows three distinct hand gestures for rotating. The first shows two hands with fingers spread, moving together in a circular path indicated by two curved arrows. The second shows one hand with a stationary index finger and another hand moving in a circular path around it, also indicated by two curved arrows. The third shows a single hand with two fingers moving together in a circular path, indicated by two curved arrows. The words 'OR' are placed between the first and second diagrams, and between the second and third diagrams.</p>

Table 13: Rotate Gesture⁵.

Chapter 4: Gesture Implementation

Constructing gestures from touch coordinates can also be abstracted to a separate JavaScript library, GenGesjs. GenGesjs will receive touch events by interfacing with getTjs. It does this by implementing the buildgestures function that getTjs calls every time it has a new touch event. One of the most difficult aspects of creating gestures is that they are built from touch points over time. In order to detect a gesture, GenGesjs must keep historic touch data to compare to the incoming data. This allows it to detect touch changes overtime and translate them to gestures.

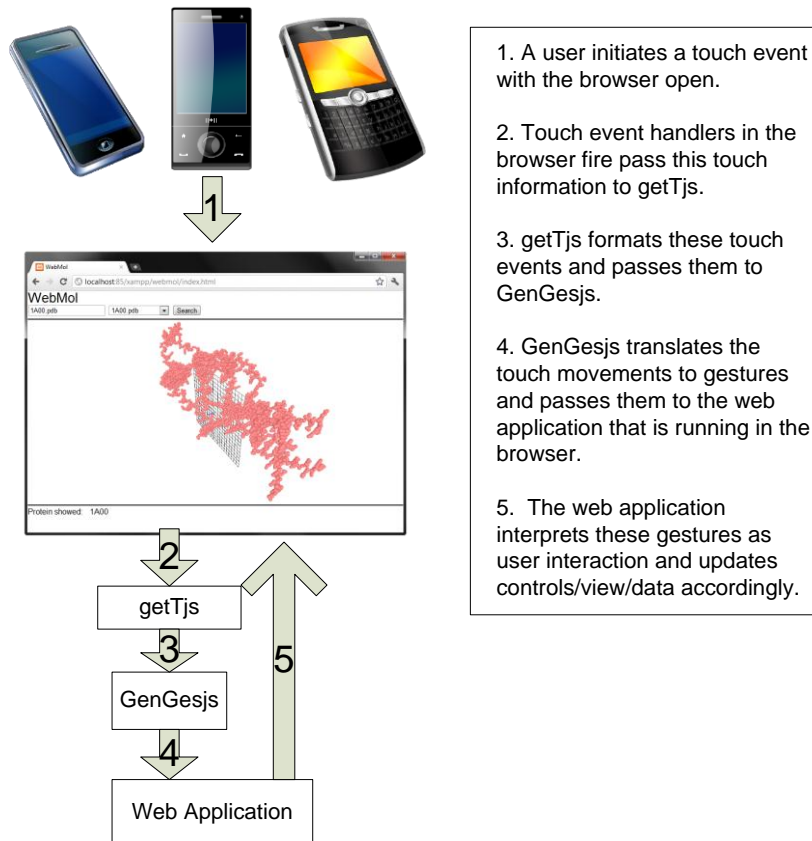
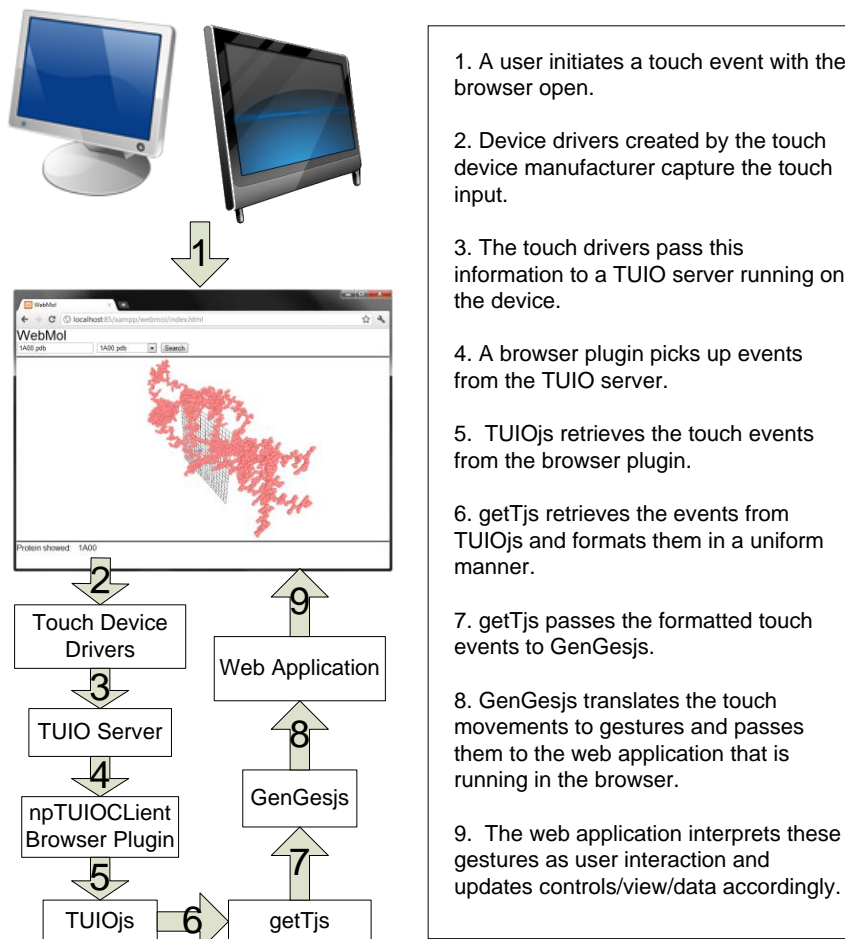


Figure 2: getTjs and GenGesjs implementation overview for a mobile device.



1. A user initiates a touch event with the browser open.
2. Device drivers created by the touch device manufacturer capture the touch input.
3. The touch drivers pass this information to a TUIO server running on the device.
4. A browser plugin picks up events from the TUIO server.
5. TUIOJs retrieves the touch events from the browser plugin.
6. getTjs retrieves the events from TUIOJs and formats them in a uniform manner.
7. getTjs passes the formatted touch events to GenGesjs.
8. GenGesjs translates the touch movements to gestures and passes them to the web application that is running in the browser.
9. The web application interprets these gestures as user interaction and updates controls/view/data accordingly.

Figure 3: getTjs and GenGesjs implementation for a desktop touch device.

GenGesjs will create a gesture object for gestures it detects. Each gesture object has several required attributes: the gesture type and the X and Y coordinates of the focus of the gesture. It also has several optional attributes that may contain data depending on the gesture: distance, secondary X and Y coordinates, and rotation. Distance is a number that represents how far a gesture has moved since it was last reported to the application. For example if a user starts a pinch gesture, GenGesjs will detect the gesture and based on how far the touch points have traveled, compute the distance and pass it to the web application. Gestures that will have a value for distance are drag, flick, pinch, spread,

and press and drag. Rotation is a value that represents the amount of rotation in degrees since the value was last passed to the web application. The rotation gesture is the only gesture that will contain a rotation value.

GenGesjs will not only have to build gestures from touch events, it will also have to build gestures from mouse events. This will allow applications using GenGesjs to simply react to the list of gestures and not worry about the user's input environment.

Gesture	Mouse Implementation
Tap	Left click.
Double Tap	Left double click.
Drag	A left click, moving slowly while left click is held, and then releasing the left click.
Flick	A left click, moving very quickly the while left click is held, and then releasing the left click
Pinch	Scroll wheel up.
Spread	Scroll wheel down.
Press	Right click.
Press and Tap	A right click with the left control keyboard button pressed. A single left click somewhere else on the viewport with the left control still being held.
Press and Drag	A right click with the left control keyboard button pressed. The mouse must then be moved with the right click and left control still held. When either the right click or the left control is released the gesture is completed.
Rotate	Hold the left alt key and scroll the mouse wheel. Scrolling up is clockwise rotation and scrolling down is a counter-clockwise rotation.

Figure 4: Mouse event equivalent of gestures

Once GenGesjs has identified a gesture, it can pass this to an application in two ways. The first is an event based approach. As soon as GenGesjs identifies a gesture it calls a function, `eventicked`, that must be implemented by the application and passes the gesture object to it as a parameter. In the `eventicked` function the application will interpret the effects of the gesture and perform necessary updates. In this model, updates to the application are completely touch event driven. When a user provides touch interaction the application responds. The second approach is application driven. GenGesjs provides a function, `returngestures`, that returns an array of gesture objects since the last time it was called. Applications that are continually refreshing their state on an interval can call this on every refresh to get an array of gestures that have occurred since the last refresh. The application will then perform necessary updates based on these gestures. This application driven model is very common with WebGL applications, especially ones that show animation.

Chapter 5: Application

WebMol is a WebGL application that can parse and display Protein Data Bank (PDB) files in a web browser. One or more molecules can be displayed in 3-D in a viewport in the center of the webpage.

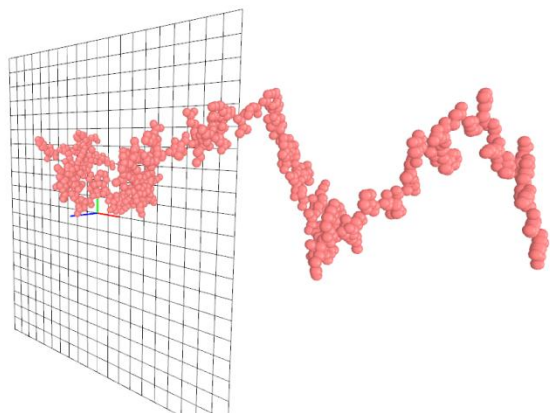


Figure 5: A molecule rendered in WebMol.

Each of the gestures described before can be used to interact with the viewport or with the rendered molecule such as moving it in space or changing one of its attributes such as size or opacity. WebMol and most other WebGL applications operate in three dimensions with three axes: X, Y and Z. Objects have an X, Y and Z value that represent their location in this space. What is displayed in the WebGL viewport is one perspective of this three-dimensional space. When building a powerful interface for WebGL we need not only a way to move molecules or other objects around, we also need a way to change our perspective. For example if a WebMol scene exists with two molecules, we first need to be able to move the molecules by performing actions such as rotating them and changing their X, Y, and Z values in relation to each other. We also need to change our perspective of these molecules such as viewing them from different angles or from

different distances. It is critical that we are able to do both using either gestures or a mouse. It is important to note that even though the WebGL three-dimensional space has fixed X, Y and Z axes, the viewport perspective does not. Horizontal is always the X axis, vertical is always the Y axis, and towards and away from the viewport is the always Z axis even if the scene is rotated and manipulated (Figure 6).

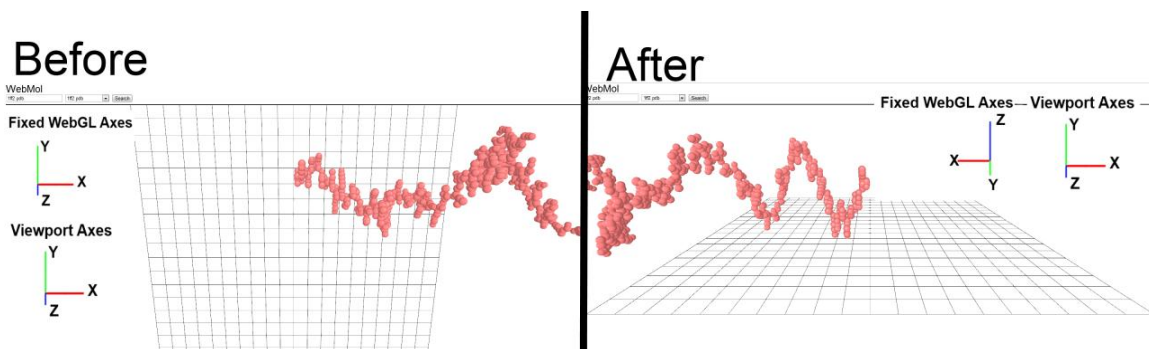


Figure 6: Axes comparison before and after a perspective change. Note the viewport axes do not change.

TAP

No action will happen when used in the viewport; the tap gesture will be used for manipulating the menu such as loading new molecules.

DOUBLE TAP

When performed on a molecule in the viewport, the double tap will select the molecule and open up more options that can be performed both with gestures and with the menu. If there are several molecules at the location of the double tap, the molecule closest to the viewport perspective will be selected. When selected, a molecule will have its opacity decreased to visually indicate that it has been selected and the menu will display the name of the selected molecule. The viewport will also “snap” to molecule,

moving so the molecule is in the center and anchoring it as the new origin of the viewport perspective.

DRAG

If no molecule is selected the drag gesture will rotate the entire scene about the origin. A horizontal drag will rotate around the viewport perspective's Y axis, a vertical drag will rotate around the viewport perspective's X axis, and a combination will rotate around both. If a molecule is selected, a drag will rotate around the molecule instead of the origin.

FLICK

If no molecule is selected a flick gesture behaves the same as a drag, rotating the scene. If a molecule is selected, the flick gesture will remove it from the scene.

PINCH

If no molecule is selected, the pinch gesture will zoom out the viewport from the origin by increasing the z value between the viewport perspective and the origin. If a molecule is selected, it will zoom away from the molecule. This will make the scene appear more distant. A maximum value exists that prevents zooming out too far.

SPREAD

If no molecule is selected, the spread gesture will zoom in the viewport towards the origin by decreasing the z value between the viewport perspective and the origin. If a molecule is selected, it will zoom in towards the molecule. This will make the scene appear closer. A minimum value exists that is the maximum the scene can be zoomed in.

PRESS

If performed on a molecule, the press gesture will open a menu that will allow actions to be performed on the molecule such as removing it, changing its color, etc. If performed where no molecule exists, a menu will appear that will allow properties or options to be set that affect the entire scene.

PRESS AND TAP

Press and tap when performed on a molecule will move the molecule to the location of the tap. This will move the molecule in two directions, the distance from the viewport perspective will stay the same.

PRESS AND DRAG

A press and drag with the press performed on a molecule will rotate the molecule. A horizontal drag will rotate the molecule around the viewport perspective's Y axis, a vertical drag will rotate around the viewport perspective's X, and a combination will rotate around both.

ROTATE

A rotate gesture performed without a molecule selected will rotate about the viewport perspective's Z axis with the origin. With a molecule selected, it will rotate about the viewport perspective's Z axis with the molecule.

Chapter 6: Future Work

This paper describes in detail how getTjs and GenGesjs work and how to implement them in an application but only provides pseudo code. Full working code for both libraries as well as integration with WebMol is the next step. A detailed description of how GenGesjs will build each gesture is necessary. Some additional gestures also need to be added such as a three finger swipe that can provide more functionality. One particular case this is needed is with rotating elements around all three axes. In the current form, the viewport perspective can be rotated in all three dimensions and zoomed in and out with gestures. Each molecule can be moved in all three directions but can only be rotated in two.

One thing to consider is how browsers will handle touch events in the future as touch based devices are used more and more frequently to access and use the internet. getTjs may become obsolete as all browsers will eventually implement the W3C Touch Events Specification. This would allow the gathering of touch events to be simplified and integrated into GenGesjs. Some of the functionality of GenGesjs may also be replaced if a W3C specification for gestures is published and implemented by browsers. This would probably be similar to what is already implemented in mobile Safari, allowing the browser to handle all detection of gestures. Event handlers could be bound to these events in web applications using JavaScript and then utilized. This would change the role of GenGesjs to be used as a library for custom gestures to be built from the browser touch events as well as a library for accessing the gestures in a polling format instead of an event based one. Many WebGL applications use a timer function to constantly rebuild the scene they are displaying. GenGesjs would catch every gesture event and make them available via the returngestures function for the application to call

at every refresh. GenGesjs could also be more tightly integrated with navigating three-dimensional space in WebGL applications. The implementation of how the gestures are used in WebMol could be implemented in GenGesjs instead, allowing it to act directly on the viewport for any type of similar WebGL application. For example, a beating heart represented in WebGL could interpret gestures in the same way as WebMol for navigating three-dimensional space and manipulating elements.

Chapter 7: Conclusion

WebMol and many other WebGL applications are extremely visual and lend themselves to viewing on touch based devices. Interactions with objects in three-dimensional space are much more fluid using multi-touch gestures. 3-D objects can be rotated, moved, and resized using touch input. Currently devices lack a way to access these touches events in the browser and use them with WebGL applications. This puts WebGL applications at a disadvantage to native applications that support multi-touch and weakens the WebGL platform. getTjs and GenGesjs provide a way for web applications to add touch support with minimal difficulty. No matter what hardware or browser is used to view the application, the same sets of gestures are detected. getTjs gathers touch information from the large number of unstandardized touch devices and GenGesjs translates the touch input into gestures. Implementing these gestures into a WebGL application yields a highly interactive and rich experience as demonstrated with WebMol.

Appendix – getTjs, GenGesjs pseudo code and implementation

```
//getTjs
$(window).bind('mousemove touchmove', function (e) {
  //parse out the coordinates etc goes here

  buildgestures(mycordarray);
});

//GenGesjs

var currentgesture = new Object();
var lastgesture = new Object();

function buildgestures(cordarray)
{
  //build gestures from cordarray

  //compare to last gesture

  currentgesture.type="pinchzoom";
  currentgesture.strength=".7";
  currentgesture.focusx=45;
  currentgesture.focusy=55;

  //if event driven this will execute
  //if not, gestures passed up via returngestures()
  if (jQuery.isFunction(eventticked) {
    eventticked(currentgesture);
  }
}

function returngestures()
{
  //return some sort of gesture array
  return currentgesture;
}

//application

//continuous redraw implementation
function tick() {
  requestAnimationFrame(tick);
  var nowgesture=returngestures();
  drawScene(nowgesture);
  animate();
}

//event based redraw
function eventticked(pushedgeature) {
  drawScene(pushedgeature);
}
```


References

Apple Inc., “Gesture Event Class Reference”, Apple Inc.,
http://developer.apple.com/library/safari/#documentation/UserExperience/Reference/GestureEventClassReference/GestureEvent/GestureEvent.html#//apple_ref/javascript/cl/GestureEvent.

TUIO, “Software Implementing TUIO”, TUIO,
<http://www.tuio.org/?software>.

W3C, “Touch Events Specification”, W3C,
<https://dvcs.w3.org/hg/webevents/raw-file/tip/touchevents.html>.

Wroblewski, Luke. “Touch Gesture Reference Guide”, LukeW Ideation + Design,
<http://www.lukew.com/ff/entry.asp?1071>.

Zinman, Aaron and others. “Depthjs”, MIT Media Lab,
<http://depthjs.media.mit.edu/>.