02561 - Computer Graphics

DTU - Technical University of Denmark

Date of submission: December 19, 2021

Virtualizing physical object movements

Daniel F. Hauge (s201186)

Abstract

This project aims to virtualize and display a physical object with it's physical movements on screen. The project consists of an implementation which succesfully captures 3 degrees of freedom and uses the movements to display on screen.

The implementation and lab journal is hosted on: grafik.feveile-hauge.dk.

1 Introduction

With ever growing computing power at disposal, more opportunities for cool technology emerges. One such technology that has just recently picked up speed, is virtual reality and augmented reality. I postulate that virtual reality is by no means a matured complete technology, as only two human senses has convincingly been implemented. The cutting edge VR technology still mostly require all movements to happen in actual reality, ie. movements happen in reality and are captured, virtualized and rendered in VR. This project will try to tackle the problem of capturing movements to display physical movements in a virtual environment.

1.1 Problem statement

How can physical movement be captured and displayed in a virtual environment like on a HTML canvas.

1.2 Motivation & Usages

As previusly mentioned, VR technology still require bodily movements to happen in reality. So capturing physical movements is currently still relevant within VR technology. Just like VR, augmented reality also benefit from being able to capture physical movements to be used for displaying graphics.

Capturing physical movements is also relevant in film making. Movies like Lord of the Rings and Avatar, have used motion capture technology to great success.

Apart from being used for rendering graphics, capture of movements can be used in new computer interfaces, like some sort of gesture controller.

2 Method

2.1 Capture physical movements

One of the challenges is to capture the physical movements. There are different ways of capturing movements, and an often used method is using light. Light is often used as it is fast, accurate and reliable. Two of the virtual environment head mounted display firms, Valve and HTC with their corresponding VRHMD Index and Vive are using light for positioning. Their systems require atleast two running base stations, which are essentially casting light for the headset and controllers to be tracked.

Using base stations casting light is extensive and has it's disadvantages, so another way, is to use kinetics. Most modern smartphones have multiple sensors which can be used for capturing physical movements.

To capture orientation, a gyroscopic sensor can capture angular velocity, an accelerometer sensor can provide a reference point for which way is up and down by the earths gravity and a magnetometer sensor can tell which way is north et cetera. All together, they can provide an absolute orientation with respect to the earth. Rotating with euler angles have problems such as gimbal lock and weird rotation transitions, quaternions are therefor much prefered over euler angles. If the previously mentioned sensors are available, the Madgwick Filter algorithm can be used to determine the orientation with respect to the earth in the form quaternions. This will capture the 3 rotational motions: Roll, Pitch and Yaw.

To capture displacement, an accelerometer sensor in conjuction with an orientation can provide an absolute displacement. If sensors are started with a initial velocity v_0 of 0, then velocity can be updated with the acceleration a and sensor frequency t in the following equation:

$$v_1 = v_0 + a \cdot t \tag{1}$$

With initial velocity and new velocity, the displacement d can be calculated with the following equation:

$$d = \frac{v_0 + v_1}{2} \cdot t \tag{2}$$

The displacement can then be used to accumulate a total displacement vector which is then used to translate the object in virtual space. The acceleration sensor in a phone has locked acceleration coordinates, such that:

- x running along the width of the device, where right is the positive direction.
- y correspond to the height of the device, where up is the positive direction.
- z correspond to the depth of the device, where screen face is the positive direction.

Because the coordinates of the acceleration is locked, the displacements calculated from these accelerations will need to be adjusted. Before accumulating the displacement, they need to be adjusted according to the orientation. The adjustment can be done by applying/multiplying the quaternion/rotation matrix to the displacement. Given the rotation matrix R and the displacement d, d can be adjusted and a updated total displacement vector D can be calculated.

$$D_{new} = D_{old} + R \cdot d \tag{3}$$

With a initial total displacement of $D_x = D_y = D_z = 0$, the accelerometer can provide a way to capture the displacement movement. This will capture the 3 transitional motions: Surge, Sway and Heave.

2.2 Data transfer

Because the phone is the sensor, it will be rotated in all different ways and is therefor not used as interface for displaying the virtual environment. The phone has the captured sensor data, this data will need to be sent to another device for displaying the movements. There are multiple wireless options for transfering the sensor data. Internet connection through wifi is the choosen method for connecting the device to a more suitable display device.

2.3 Display

Displaying rotation can be done at different stages. The rotation could be applied in eye space, such that the object would appear rotated in the correct manner, but everything else would also be rotated the same. A preferable rotation application would be to world space, ie. the vertecies of the object. With the following equation all vertecies v_i can be rotated by an arbitrary amount of quaternions q_i:

$$q_i \cdot (q_{i-1} \cdot (\ldots (q_1 \cdot v_i \cdot q_1^{-1}) \ldots) \cdot q_{i-1}^{-1}) \cdot q_i^{-1} \tag{4}$$

A convenient way to work with quaternion rotation would be to calculate a rotation matrix. The rotation matrix can then be used in conjuctions with other transformation matrices. The rotational matrix R can be calculated from a quaternion q by the following equation:

$$R = \begin{bmatrix} 1 - 2 \cdot q_y^2 - 2 \cdot q_z^2 & 2 \cdot q_x \cdot q_y - 2 \cdot q_z \cdot q_w & 2 \cdot q_x \cdot q_z + 2 \cdot q_y \cdot q_w & 0 \\ 2 \cdot q_x \cdot q_y + 2 \cdot q_z \cdot q_w & 1 - 2 \cdot q_x 2 - 2 \cdot q_z^2 & 2 \cdot q_y \cdot q_z - 2 \cdot q_x \cdot q_w & 0 \\ 2 \cdot q_x \cdot q_z - 2 \cdot q_y \cdot q_w & 2 \cdot q_y \cdot q_z + 2 \cdot q_x \cdot q_w & 1 - 2 \cdot q_x^2 - 2 \cdot q_y^2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5}$$

Displaying a displacement is achieved by simple translation. With a total displacement vector d, a translation matrix T can be calculated:

$$T = \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6}$$

Transformation with the rotation and translation matrix is the choosen method for moving the object in virtual space.

3 Implementation

3.1 Capturing physical movements

3.1.1 Orientation

To make an implementation that would work for most devices, a widely available API should be used. There is support for **AbsoluteOrientationSensor** API in most modern browsers. This sensor provides an API for capturing orientation of a mobile device. The API outputs quaternion, which is very usefull for the currently choosen method. To use the phone with the web api for capturing movements, a new page on the website is created for the phone to visit. When visiting the page from a phone, the sensor data is captured and sent to another device which is visiting the display page. The sensor is initialized with frequency and reference frame, and a event listener can be added before starting.

The orientation sensor is setup and started with the following code snippet:

```
orientationSensor = new AbsoluteOrientationSensor({ frequency:  60,
referenceFrame:  'device' });
orientationSensor.addEventListener('reading', ev => {
      const x = orientationSensor.quaternion[0];
      const y = orientationSensor.quaternion[1];
      const z = orientationSensor.quaternion[2];
      const w = orientationSensor.quaternion[3];
      console.log(x, y, z, w);
});
orientationSensor.start();
```

On each read, the sensor object has an updated property called **quaternion**, which is the last captured physical rotational movements.

3.1.2 Displacement

Just like orientation, displacement is captured with a web api which is supported in most modern browsers. The page that captures phone data is extended to capture acceleration aswell. The **LinearAccelerationSensor** is used for capturing acceleration. Just like the orientation sensor, it is initialized with a frequency and an event listener can be added. Each readings contain an acceleration along x, y and z axis.

```
accelerationSensor = new LinearAccelerationSensor({frequency:  60});
accelerationSensor.addEventListener('reading', () => {
      const x = accelerationSensor.x;
      const y = accelerationSensor.y;
      const z = accelerationSensor.z;
      console.log(x, y, z, w);
});
accelerationSensor.start();
```

On each read, the sensor object has updated x, y and z properties, which correspond to the accelerations in x, y and z axis.

3.1.3 Support

An important implementation detail for using sensor web api's, is that it is required to access the website through a secure connection. In other words, HTTPS is required and regular HTTP is not adequate. Another thing is, that if the device does not have adequate sensors, the API's will not be defined in the javascript environment. Therefor error handling has also been implemented to let a user know if sensor is unavailable and more.

3.2 Data transfer

The data transfer has to happen between 2 clients of the website. One client is the phone that is visiting the page which setup capture sensors. Another client is a device that can display the virtual environment. The implementation is done with websockets. I found a golang and a javascript library from socket.io that implements websockets.

Using the library, dedicated connection logic is implemented, such that a phone can connect to another client. There is quite a bit more to how the data transfer implementation is done, but to keep it simple, only the actual transfering of data is covered. See **websocket.js** and **main.go** for full websocket code.

When a connection is established, the phone can then emit it's readings to the client via websockets. The following statement is emiting events with orientation sensor readings to the server.

```
ProjectSockets.Socket.emit("event", { id:  connectionId, type:
"orientation", value:  JSON.stringify(orientationSensor.quaternion)});
```

The website server is extended with a websocket server handler: Listeners for specific channels can then be added. The following listener is adding functionality for the server to receive the phone's events and broadcast them to the connected display client.

```
server.On("event", func(c *gosocketio.Channel, msg Event) string {
	if msg.EventType == "connection" {
		if observer[msg.SessionId] || !listener[msg.SessionId] {
			return "BAD"
		} else {
			observer[msg.SessionId] = true
			connections[c.Id()] = msg.SessionId
		}
	}
	c.BroadcastTo(msg.SessionId, msg.EventType, msg.Value)
	return "OK"
})
```

The following code will setup a listener on the display client that listens for events from the phone through the server. The event contain the readings from the phone.

```
ProjectSockets.Socket.on("orientation", orientationArray => {
	let quaternion = JSON.parse(orientationArray);
	if (ProjectSockets.OnOrientationRead != null) {
		ProjectSockets.OnOrientationRead(quaternion);
	}
});
```

This listener calls a function **OnOrientationRead** on every received readings from the phone. Similar listeners are made with acceleration, connection, disconnection and alignment. The function can be assigned to let the display device react to new orientations et cetera.

3.3 Display

To display the environment, first a virtual environment with a static object is implemented. Using the same methods and procedures as in worksheet 5 and 10, an object is loaded with a .OBJ file, and a quad is loaded with a texture for looks.

To rotate the object based on the phones readings, a variable holding a quaternion is declared and updated by each readings from the phone. The following code assigns the **OnOrientationRead** function to set a reference orientation and update the quaternion variable with respect to the reference.

```
ProjectSockets.OnOrientationRead = ori => {
	const newOri = [ori[0], ori[1], ori[2], ori[3]]
	if (Project.refSet == undefined) {
		Project.refSet = true;
		Project.q_rot_ref.set(newOri);
	}
	Project.q_rot.set(newOri);
	Project.q_rot.multiply(Project.q_rot_ref);
}
```

The quaternion.js contain a function for calculating the matrix, this is used when drawing the phone:

```
const rot = new Matrix4();
const mat4 = Project.q_rot.get_mat4();
const flattened = flatten(mat4);
rot.set(elements:  flattened);
Project.g_modelMatrix = Project.g_modelMatrix.multiply(rot);
```

The phone readings are now able to rotate a model matrix which is then used for drawing an object in a virtual environment. There are more implementation details which involves perspective alignment with the screen and more, these will not be covered to keep it succinct. See **project.js** for full code on display/drawing.

To draw the displacement, an attempt with similar approach to orientation is done. The following code is assigning the **OnAccelerationRead** to calculate a new total displacement vector.

```
const Displacement = {x:0, y:0, z:0};
const Velocity = {x:0, y:0, z:0};
const DeltaT = 1/15;
ProjectSockets.OnAccelerationRead = Acceleration => {
      const vX = Velocity.x + Acceleration.x * DeltaT
      const vY = Velocity.y + Acceleration.y * DeltaT
      const vZ = Velocity.z + Acceleration.z * DeltaT
      Displacement.x = 0.5 * (vX + Velocity.x) * DeltaT
      Displacement.y = 0.5 * (vY + Velocity.y) * DeltaT
      Displacement.z = 0.5 * (vZ + Velocity.z) * DeltaT
      Velocity.x = vX;
      Velocity.y = vY;
      Velocity.z = vZ;
      Project.PD.x = Displacement.x;
      Project.PD.y = Displacement.y;
      Project.PD.z = Displacement.z;
}
```

A translation matrix is then multiplied to the model matrix:

```
Project.g_modelMatrix.translate(
      Project.Phone_X + Project.PD.x,
      Project.Phone_Y + Project.PD.y,
      Project.Phone_Z + Project.PD.z);
```

Unfortunately, the implementation for displaying displacement is not working as intended. It seems as through acceleration is not accurate, as moving the phone from one stationary position to another, the velocity is not zero when the phone is done moving. The acceleration or deacceleration reads must have been larger to produce a non zero velocity. A different method could perhaps be better at implementing the solution.

4 Results

The solution is a website that when accessed with a phone and another display device, can capture rotational motion of the phone and display it's movements on a html canvas. Unfortunately, translational motion is not succesfully captured and displayed.

The following link is a youtube video of a demo of the solution in action: https://youtu.be/8K5ufYZokSE

In order to try the solution, a phone with a accelerometor sensor, gyroscopic sensor and magnetometer sensor is required. Also another device for displaying and phone with adequate browser that support these sensors. To try the solution, follow these steps:

- With display device (ex. computer) navigate with a browser to the running instance of the solution or https://grafik.feveile-hauge.dk (https is important)
- With phone, navigate to **URL/c?id=XXX** where XXX is substituted with code on the additional device.
- On phone, click the "Start sensor" button.
- Put phone on stable surface pointing towards the display device and click "Realign" button.

Following these steps will hopefully demonstrate how the solution is able to capture physical movements to be displayed virtually.

To run an instance of the solution locally. Have golang installed and run "go run main.go" in a terminal at root directory of the extracted zip, with a system environment variable "development" set to true.

5 Conclusion & Discussion

Physical movements can be captured with a modern smartphone's sensors. An accelerometor, magnetometer and gyroscopic sensor can be used to measure kinetics of the rigid body. Measurements from the sensors can be transfered to a different device via a internet connection with the help from websockets. The data is used to calculate transformations matrices, which are then used during rendering to display the movement in virtual space.

The solution succesfully captures 3 degrees of freedom with the rotational motions, and displays the movement on a html canvas. Unfortunately the remaining 3 degrees of freedom is not succesfully captured and displayed. The method choosen for translational motions is sensitive to accuracy, and perhaps a different method would have been better.

Regardless of missing degrees of freedom, I still think the project is cool and interesting to play with.