TECHNISCHE UNIVERSITÄT DRESDEN

FACULTY OF COMPUTER SCIENCE
INSTITUTE OF SOFTWARE AND MULTIMEDIA TECHNOLOGY
CHAIR OF COMPUTER GRAPHICS AND VISUALIZATION
PROF. DR. STEFAN GUMHOLD

Bachelor's Thesis

for obtaining the academic degree
Bachelor of Science

Development of a Natural VR User Interface Using Haptic Gloves

Lucas Waclawczyk
(Born 26. April 1998 in Bad Langensalza, Mat.-No.: 4686687)

Tutor: Prof. Stefan Gumhold

Dresden, August 20, 2020

Task Description

Write down your task...

Declaration of authorship

I hereby declare that I wrote this thesis on the subject

Development of a Natural VR User Interface Using Haptic Gloves

independently. I did not use any other aids, sources, figures or resources than those stated in the references. I clearly marked all passages that were taken from other sources and cited them correctly.

Furthermore I declare that – to my best knowledge – this work or parts of it have never before been submitted by me or somebody else at this or any other university.

Dresden, August 20, 2020

Lucas Waclawczyk

Kurzfassung

abstract text german

Abstract

abstract text english

Contents

1 Introduction

2 Development

2.1 Devices and Software

For the purpose of thesis, I used a pair of haptic gloves called Avatar VR which is a registered trademark of NeuroDigital Technologies, S. L., referred to as ND hereafter. The company's details can be found at `neurodigital.es`.

To use the Avatar VR on a computer, one needs to download, install and run the ND Suite. This program provides a background service and GUI for managing and accessing the devices. A connection can be established via Bluetooth after which all sensor data is graphically displayed in the GUI.

ND also supplies a developer's API in the form of three files (`NDAPI.h, NDAPI_x64.lib and NDAPI_x64.dll` or `x86` respectively) which need to be included into the respective project in a suitable manner. An instance of the `NDAPI.h` can be used to access the data of devices connected to ND Suite.

Additionally, a Vive tracker was used to determine the user's hand position and orientation. The Avatar VR can be mechanically connected to the tracker using a coupling, that needs to be acquired separately. The devices named so far and the way of coupling them can be seen in figure 2.1.

The foundation for the graphics software I developed is implemented in the `cgv` framework provided by the chair of computer graphics and visualization at TU Dresden.

A user can view the plugin on a computer screen with the `cgv_viewer` or on a Vive head mounted device (HMD) simply by connecting one to the computer.

2.2 Project Idea and Structure

When developing a VR user interface, the first question is: what is it supposed to do? To give this some context rather than just moving boxes, I decided to design it as a conn panel on the USS Voyager (Star Trek). The term *conn* is short for *control and navigation*, meaning the panel is used to stear the ship. The classes used to implement this and their interaction are shown in figure 2.2.

The main class and entry point carries the same name as the plugin itself. It is derived from the `cgv` classes

(a) Uncoupled devices, colors mark coupling points

(b) Coupled devices on user hand

Figure 2.1: Devices used to track the user's hand: Vive tracker, coupling, Avatar VR

`base`, `drawable`, `event_handler` and `provider`, and used for management of all objects. Its capability to handle `vr_pose_events` is used to forward tracker poses (position and orientation) to the `hands` and HMD poses to the `head_up_display`. It distributes `draw()` commands among the relevant members and manages the calibration routine described in section 2.5. This is the only class registered via object registration.

Written information can be displayed to the user with the `head_up_display` if an HMD is connected and on the console alternatively. The `head_up_display` is permanently adjusted to be displayed in the upper left corner of the user's visual field.

The `mesh` class renders the Voyager bridge. It uses the basic functionality of `mesh_render_info` provided in `cgv` to load and draw.

Around the bridge, space is simulated by a spherical shell filled with stars. The user never actually changes position in model space. Instead, stars are moved in the opposite direction of the simulated motion, the speed of which can be set via static `set_speed_*()` methods. The details of this are described in section 2.4.

All interactive elements are implemented as `panel_node` or its derived classes. A `conn_panel` constructs and manages a tree of such nodes. Elements are represented as boxes and combinations of boxes placed relative to their parent element. The `slider` and `lever` classes require a callback function and pointer to a `stars_sphere` as arguments in their constructor that are used to set some speed in the `stars_sphere`. The `conn_panel` is positioned right on top of the mesh's conn.

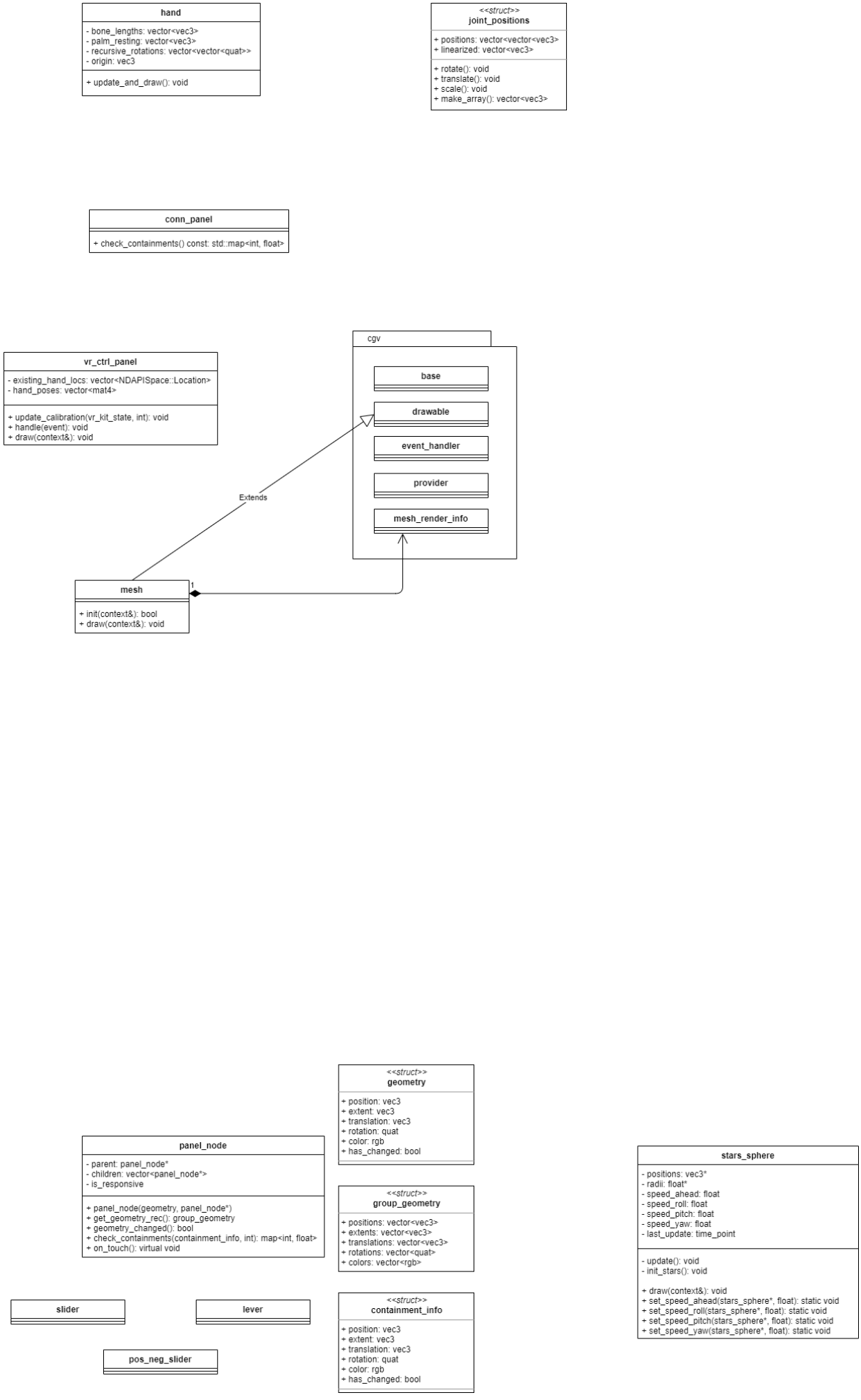
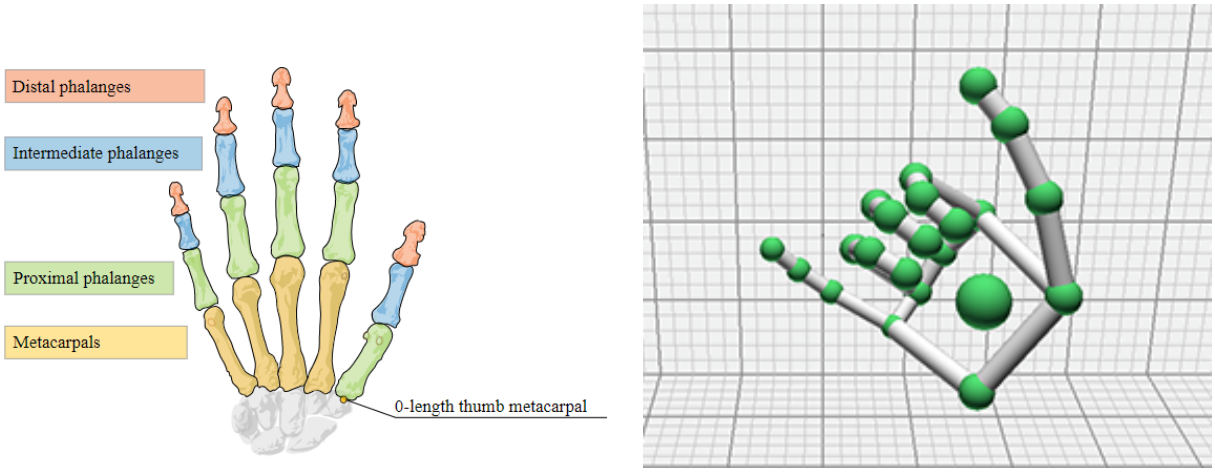


Figure 2.2: Class diagram of `cgv_oculus_nodal`, only most relevant elements shown



(a) General bone structure of the human hand

(b) Screenshot of an example hand representation in the Leap Motion API version 2.0, cropped

Figure 2.3: A model of the human hand; from [lea]

Interaction with the Avatar VR is managed by the singleton `nd_handler` which can be used to pass function calls through to an instance of `NDAPI`. This is frequently done in `nd_device`, a class for querying sensor data from the gloves and converting it for better compatibility with `cgv` and `vr_ctrl_panel`.

Finally, a representation of the user's hands is implemented in `hand`. It keeps track of several points and joints the details of which are described in section 2.3. During a `draw()` call, `vr_ctrl_panel` passes a pointer to its `conn_panel` to each `hand` which then passes on information about the current hand geometry wrapped as `containment_info` to the `conn_panel` for containment check. During this check, each `panel_node` containing some hand part can react with its `on_touch()` method. A `map<int, float>` describing the contained positions is returned to the hand, enabling it to react as well.

2.3 Skeletal Hand Model for the Avatar VR Haptic Glove

2.3.1 Abstraction of the Human Hand Skeleton

The human hand can be roughly divided into the parts shown in figure 2.3a, i.e. the *metacarpals*, *proximal phalanges*, *intermedial phalanges*, and *distal phalanges*. A joint connecting a metacarpal to a proximal phalanx is called *metacarpophalangeal joint*, and followed by a *proximal interphalangeal joint* and a *distal interphalangeal joint*.

Even though the carpals (grey in figure 2.3a) are physiologically quite relevant for hand movement, they stay relatively fixed compared to the other bone sets, and can thus be omitted in our simplified model. In anataomical nomenclature, the thumb is composed of proximal phalanx and distal phalanx only

and follows a metacarpal. However, this model assumes a missing metacarpal and existing intermedial phalanx instead which will be modelled by a 0-length metacarpal for uniformity reasons.

The Leap Motion API version 2.0[lea] uses the abstraction described above and adds

- an "end" joint per finger (at the end of the distal phalanx)
- a joint diagonally across the palm from the metacarpophalangeal joint of the index
- a joint in the middle of the palm

The most common representation includes cylinders for the bones and spheres for the joints. It is shown in figure 2.3b.

2.3.2 Technology and Function of the Avatar VR

The Avatar VR is equipped with three kinds of sensors:

- An *intertial measurement unit (IMU)* is located on the intermedial phalanx of each finger to measure its 3D rotation. The thumb is even equipped with two IMUs (one per phalanx) and the palm with an IMU that can measure both 3D orientation and 3D rotation. The "Motion" tab of ND Suite shown in figure 2.4a depicts the resulting orientations as orthonormal bases.
- At the locations marked in blue in figure 2.4b, the fabric is made of an electrically conductive material. These parts are used as contact sensors (short: contacts) to determine which of the marked locations are joined.
- The thumb also has a flex sensor, that will not be relevant for this thesis.

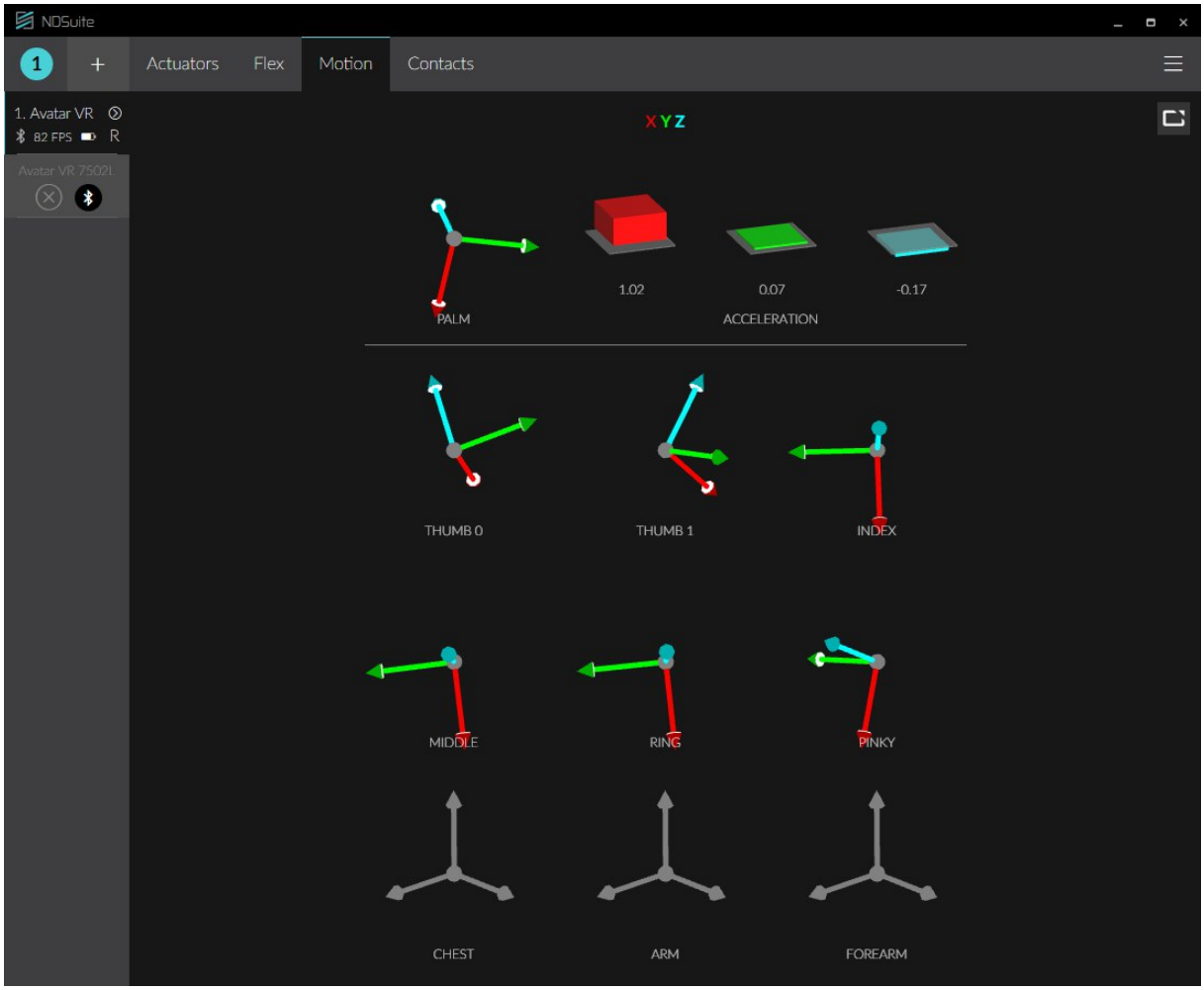
Furthermore, there are actuators at the locations marked in blue in figure 2.4c. These can be used to generate vibrations of variable intensity as haptic feedback in a way that "[. . .] the brain perceives it as 'Real Touch' input", according to the producer.

2.3.3 Using Data From the Avatar VR to Visualize the User's Hand

The same model used by the Leap Motion API version 2.0 will also be used here. Leap captures absolute joint positions and optimizes them for visualization. In contrast, my approach for the Avatar VR is to use a fixed resting state geometry of the user's hand (bone lengths and resting state joint positions). At the time of a `draw()` command, the geometry is adjusted according to the data given by the glove and the Vive tracker.

The joint position data is managed by the following struct (only most important parts included):

```
struct joint_positions {
        // positions of hand joints in model space
```



(a) "Motion" tab of ND Suite with orientations in an (optimistic) example hand pose

(b) Positions of contact sensors

(c) Positions of actuators

Figure 2.4: Sensors and actuators of the Avatar VR;
screenshots of ND Suite (b and c cropped)

```
        // structure: hand_part<phalanx<position>>
        vector<vector<vec3>> positions;
        // positions linearized in hand_part major format
        // set by make_array()
        vector<vec3> linearized;
        // correspondence of indices in linearized
        // to double indices in positions
        map<int, pair<int, int>> lin_to_anat;

        // rotate complete part
        void rotate(int part, quat rotation) {...}

        // translate complete part along neg. z-axis
        void translate_neg_z(int part, float z) {...}

        // translate all positions
        // used to move hand from construction origin
        // to model view location
        void translate(vec3 translation) {...}

        // scale all positions
        // used to realize hand size at construction origin
        void scale(float scale) {...}

        // generate linearized from positions
        vector<vec3> make_array() {...}
};
```

First, the palm joints' resting positions are rotated according to the orientation given by the Vive tracker. The construction of each finger begins at the end joint of the distal phalanx which is translated along the negative z-axis by the hard-coded length of the distal phalanx and rotated by the respective quaternion (see below). The other phalanges are constructed in the same manner, before the whole finger is translated to the respective metacarpophalangeal joint. Finally, the whole hand is scaled to model view space and translated to the model view position of the Vive tracker.

Because the Avatar VR only returns one quaternion per finger (except for the thumb), I had to think of a way to distribute the orientation along all three phalanges: First, the Euler angles are calculated from the quaternion[?]

$$\begin{aligned} \alpha &= \arctan\frac{2(q_0q_1 + q_2q_3)}{1 - 2(q_1^2 + q_2^2)} \\ \beta &= \arcsin(2(q_0q_2 - q_3q_1)) \\ \gamma &= \arctan\frac{2(q_0q_3 + q_1q_2)}{1 - 2(q_2^2 + q_3^2)} \end{aligned}$$

where α is the roll angle, β the pitch angle and γ the yaw angle, and the quaternion can be written as

(q_0, q_1, q_2, q_3). For implementation, one has to use the `atan2` function, because `arctan` only return values between $-\frac{\pi}{2}$ and $\frac{\pi}{2}$.

The `NDAPI` quaternions live in a space where the resting hand lies on the xz-plane and the fingers point along the positive z-axis. However, `vr_ctrl_panel` is oriented with the negative z-axis as "front" direction, so q_1 and q_2 need to be negated before use. Then roll corresponds to bending the finger, pitch to a left-right-motion and yaw to a motion humans can hardly perform with their fingers.

Since the interphalangeal jionts act as revolute joints, only a roll component can be applied to them. As shown in the code below, the roll angle is distributed according to three floats (`rot_split`). After some experimenting, $(.5, .5, .25)$ is an option for these numbers that gives a good approximation of most actual hand poses. It corresponds to phalanx rotation during grabbing. Yaw and pitch are completely executed in the metacarpophalangeal joint.

```
vec3 x(1, 0, 0), y(0, 1, 0), z(0, 0, 1);
recursive_rotations[finger][PROXIMAL] = palm_rotation
        * quat(z, yaw) * quat(y, pitch) * quat(x, rot_split.x() * roll);
recursive_rotations[finger][INTERMED] = quat(x, rot_split.y() * roll);
recursive_rotations[finger][DISTAL] = quat(x, min(1.4f, rot_split.z() * roll));
```

2.4 Simulation of Space

A schematic of the spherical shell used to simulate space is shown in figure 2.5. The outer radius determines the overall size of the sphere. A spherical region around the user that does not contain any stars is defined by the inner radius. This should include the complete bridge mesh to assure stars are only visible to the user on the view screen. Congruent with other parts of the plugin, the flight and view direction points along the negative z-axis.

Stars are rendered as small spheres whose radii follow a normal distribution. The shell is initialized with a constant number of stars with uniformally distributed positions. It saves the forward and angular speeds as float values. During a `draw()` call, the method `update()` calculates new positions for all stars. The following code is explained below.

```
void update()
{
        // to ensure realistic movement independent of frame rate
        chrono::steady_clock::time_point now = chrono::steady_clock::now();
        float ms_elapsed = chrono::duration_cast<chrono::milliseconds>(now - last_update).count();

        float distance_elapsed = speed_ahead * ms_elapsed;
        vec3 angles = ms_elapsed * vec3(speed_pitch, speed_yaw, speed_roll);
        mat3 rotation = cgv::math::rotate3(angles),
```

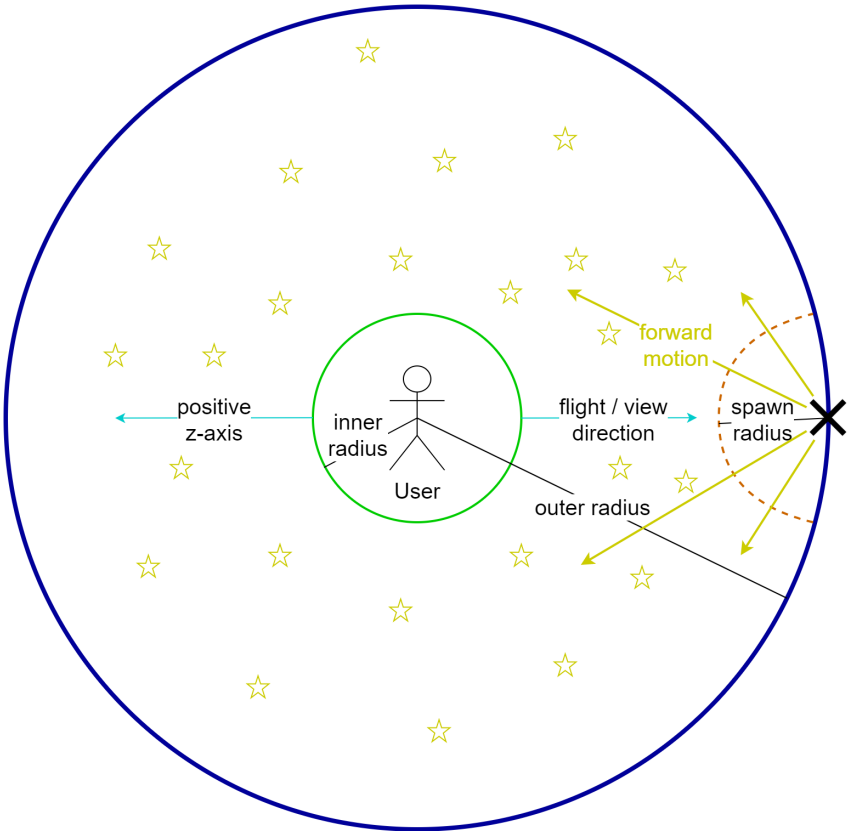


Figure 2.5: Schematic of the elements used in `stars_sphere` to simulate space

```
    inv_rotation = cgv::math::rotate3(-r_out / 2 * angles);

vec3 p;
for (size_t i = 0; i < num_stars; i++)
{
        // the following lines model dead ahead movement
        p = positions[i];
        float l = p.length();
        p.normalize();
        // now, p.z() is the cosine of the angle between
        // p and the pos. z-axis
        p *= l + p.z() * distance_elapsed;
        // translate p to model space
        p += origin;

        // check if p is outside the shell
        if (p.length() > r_out)
        {
                // get random angles from uniform distribution
                float alpha = dis_angles(gen), beta = dis_angles(gen);
                // init new position on spawn sphere
                p = r_out / 10 * vec3(
                        sin(alpha) * cos(beta),
```

```
                                sin(beta),
                                cos(alpha) * cos(beta)
                        ) + origin;
                        positions[i] = inv_rotation * p - origin;
                }
                else
                {
                        positions[i] = rotation * p - origin;
                }
        }

        last_update = now;
}
```

To somewhat simplify the math behind this calculation, I used the point on the outer hull right in front of the user as origin for the shell space, instead of the center of the shell. The origin is marked with a cross in figure 2.5. A model view matrix then translates the shell to model space.

When moving dead ahead (angular speeds are zero), stars are moved along the yellow "forward motion" arrows in figure 2.5. The bigger the angle between their position vector and the positive z-axis, the slower they move. More precisely, the length of a star's position vector is increased by the product of the cosine of the angle between the position vector and the positive z-axis, and the general distance it covered, calculated from `speed_ahead` and the time since the last update.

If a star moves outside the outer hull of the shell, its position is re-initialized randomly somewhere on a sphere of the spawn radius around the origin. It cannot be simply set to the origin because it needs a direction and the spawn radius should not be too small in order to also avoid the impression of stars spawning very densly.

An additional rotation is realized by simply rotating the stars around the user. After some experimenting, I realized the spawn sphere needs to be rotated inversely around the user as well to achieve a realistic experience and because otherwise, the newly spawned stars converge to a line at slow forward motion and fast rotations.

2.5 calibration

To make it possible to use `vr_ctrl_panel` in different rooms and setups, a calibration routine is required. This can be activated by joining index and thumb of one hand for three seconds. The objects in the scene are then no longer rendered.

The user position is determined from the HMD if connected. Otherwise, the user is asked to tap index

and thumb above their head and the position of the respective tracker at that time is used.

After that, a countdown of $3s$ is displayed and the user is instructed to hold their hands straight with the fingers pointing forward. At the end of the countdown, the hands are rendered again and the current pose is passed on to the `hand` instances for calibration. It becomes a new reference pose from which all following orientations are inferred.

This is necessary because the Avatar VR does not give sufficiently accurate measurements for a period longer than a few minutes. Since the device apparently accumulates quaternions from instantaneous acceleration rates, the resulting rotations tend to quickly diverge from the actual hand pose. This effect is stronger if the device was moved even slightly during its intrinsic calibration at startup time.

Finally, the conn panel is moved close to the hands and the new z-axis is set to run from the average of the hand positions and the user position. One can continue to adjust the panel position and z-axis by moving their hands until satisfied. A final acknowledgement by tapping index and thumb ends the calibration.

Between two stages, a short feedback pulse is generated by the actuators simultaneously to notify the user of the change. When the calibration is finished, a final pulse utilizes all actuators successively. The calibration can also be aborted resulting in three short pulses. One can then choose to reset to the previous calibration or keep the partially altered one.

Glossary

distal interphalangeal joint joint connecting an intermedial phalanx to a distal phalanx. 2

distal phalanx finger bone farthest from the palm of the human hand, shown in figure 2.3a. 2, 4, 6

intermedial phalanx finger bone of the human hand between proximal phalanx and distal phalanx, shown in figure 2.3a. 2, 4

metacarpal set of bones in the palm of the human hand, shown in figure 2.3a. 2, 4

metacarpophalangeal joint joint connecting a metacarpal to a proximal phalanx. 2

proximal interphalangeal joint joint connecting a proximal phalanx to a intermedial phalanx. 2

proximal phalanx finger bone closest to the palm of the human hand, shown in figure 2.3a. 2, 4, 6

Bibliography

[lea] Introducing the skeletal tracking model. accessed 18 Aug 2020.

[PGP+10] Philipp Pohlenz, Alexander GrÃ¶be, Andreas Petersik, Norman [von Sternberg], Bernhard Pflesser, Andreas Pommert, Karl-Heinz HÃ¶hne, Ulf Tiede, Ingo Springer, and Max Heiland. Virtual dental surgery as a new educational tool in dental school. *Journal of Cranio-Maxillofacial Surgery*, 38(8):560 – 564, 2010.

Acknowledgments

I’d like to thank...

Copyright Information

Voyager Bridge Mesh

Acquired from `https://www.trekmeshes.ch/` on 20 June 2020.

This mesh remains the property of Chainsaw_NL and Star trek Meshes. It however may be used to create images and scenes for non-profit use only.

Any images produced with this mesh do NOT have to be credited to the mesh author. But it would be nice.

Star Trek and Enterprise are copyright Paramount Pictures.