A Report on VR Spacecraft Assembly and its features

Daniel Tauritis, Waldemar Zeitler, Hannah Kathmann,
Marcel Merten, Patrick Plate, Dustin Augsten,
Andy Augsten, Arian Mehrfard, Thomas Tannous,
Florian Rohde, Dominik Sauer, Daniel Albensoeder
CGVR
Bremen University

CONTENTS

A Report on VR Spacecraft Assembly and its features

***Abstract*—This document provides inside into the technical details of the VR Spacecraft Assembly, giving an overview of the implementation and configuration. The input devices were selected to grant an immersive experience in virtual reality. Our Software allows users to configure a spacecraft by attaching various sensors.**

I. Introduction

This technical report presents the reader with the technical implementation and background of the Bachelor's project "VR Spacecraft Assembly". Points such as user experience are not explained in detail, as this is done in our study on this topic. We assume that the reader is familiar with the technology used, for example the Unreal Engine and the HTC Vive. Our project is entirely virtual reality based and therefore requires the use of a head mounted display and the associated motion controllers. Furthermore we make use of additional input methods such as the Leap Motion and Cybergloves. At the start of the application, the desired input device is selected by means of a visual gesture and the actual program is started afterwards.

II. Problem definition and features

A. PathSpline

Users can start and stop a spacecraft animation, during which the spacecraft takes off, orbits an asteroid and then lands lands after the users has stopped the animation.

This animation utilizes three PathSpline actors managed by the level blueprint.

A PathSpline consists of

- A spline component,
- A parameter defining the object to be moved along the spline,
- A curve describing the movement along that spline.

The spline path and the parameters must be set in each individual PathSpline instance. The function starting the animation will set the object's position along the spline over a period of time using the supplied curve.

The PathSplines used by the level blueprint must be tagged 'PathSplineStart', 'PathSplineLoop' and 'PathSplineEnd'. The level blueprint can then play these PathSplines in the proper sequence.

To toggle the animation users can press a button in the menu. This calls a custom event in the level blueprint, which initiates the animation, but also aligns the spacecraft with the asteroid. All rotation axes of the spacecraft are rounded to the nearest 90 degree value during takeoff, ensuring that the far end side of the spacecraft faces the asteroid. It will keep facing the asteroid by rotating around its z-axis once per orbit.

PathSplines can easily be reused elsewhere, as they are very flexible and only require their animation to be triggered externally. However, we deem it unlikely that they will be reused much in this project unless future developers wish to implement more animations.

B. RotatingAsteroid

We use various asteroid models to embellish the space environment. RotatingAsteroid actors placed in the level randomly select one of those models and randomize their scale, rotation and rotation speed. The maximum and minimum scale can also be overridden in each individual RotatingAsteroid instance to prevent specific asteroids from clipping into each other.

C. HeatmapMaterial

The heat generated by certain spacecraft parts can be visualized using the heatmap material. It can be applied to any mesh colors it according to proximity to heat sources.

Heat source locations are managed using the parameter collection 'HeatmapParameters'. Up to eight heat sources may be enabled here, and their radius and locations are set through various parameters.

HeatmapMaterial uses HeatmapParameters to apply a sphere mask around every heat source location. The mask's hardness is applied to every source's sphere mask and has been set inside the HeatmapMaterial to make every heat source emit heat in a similar way. The values of the sphere masks are checked for every point of the material's surface and added. This results in a value from zero (if no heat source is close) to eight (when eight heat sources are very close). Currently we use the following formulas to determine the HeatmapMaterial's color according to this value:

$$c(x) = b(x) \cdot Blue + g(x) \cdot Green + y(x) \cdot Yellow + r(x) \cdot Red$$

$$b(x) = \max\Big\{\min\big\{x \cdot -3 + 1, 1\big\}, 0\Big\}$$

$$g(x) = \max\Big\{-|(x \cdot 3 - 1) + 1|, 0\Big\}$$

$$y(x) = \max\left\{-|\left(\left(x - \frac{1}{3}\right) \cdot 3 - 1\right) + 1|, 0\right\}$$

$$r(x) = \max\left\{\min\left\{\left(x - \frac{2}{3}\right) \cdot 3, 1\right\}, 0\right\}$$

Values beyond x=1 will be colored red. However the input value x by eight to stretch the color range. Due to the software currently using two heat sources the color range was not changed.

The method for calculating the HeatmapMaterial color is an estimate for actual thermodynamics. The coloring could be improved by using an actor instead of a material to model heat being emitted by heat sources, however our approach has a much lower performance impact.

D. LeapPawn

The LeapPawn inherits from GeneralControllerPawn and calls its grab function when the leap detects the corresponding gesture. It also updates the locations of the HandR and HandL components to the user's palm positions. (to be continued)

E. CyberGloves and Polhemus

For the project we had the technologies of the CyberGlove version one and three and also the Polhemus Fastrak system. With these tools it was possible to create a semi realistic simulation to track hands in a virtual environment and also to grab objects with hand gestures.

Polhemus Fastrak: The Polhemus Fastrak system allows the capturing of the positions of special transmitters in real time. These transmitters are tracked by a magnetic field. The magnetic field was also a big problem in the project, because it is easily influenced by nearby metal or electronic devices, also the distance that is tracked is very limited. But through a k-means algorithm and some offsets it was possible to create a smooth tracking, if the distance was not to far away from the center of the magnetic field.

The output of the Polhemus was also a problem, because the serial implementation that was used had an error and because the output was set to "Euler Angles". After some debugging the error was found, which was a wrong position in the buffer, were the angles were saved. With the information of the data sheet from the Fastrak system it was also possible to switch the output from "Euler Angles" to "Quaternions".

CyberGloves: ... In section IV. the CyberGloves and functionality of them is described further.

F. Collision Detection

Ghosting: The ghosting is a simple holographic copy of the sensor that is being hold. The holographic copy of the Sensor shows the place where the sensor would be placed, if it would be dropped right now. Because the ghost is just a Copy of the original object that is hold, several Problems arose. It was necessary to avoid the grabbing of the holographic copies and holographic copies should not be able to create holographic copies of their own. To fix these issues a simple Boolean was set, which indicates if the sensor is a hologram or not. To avoid holograms at places they should not appear, they get destroyed whenever they don't overlap any plane or where they can't snap.

Snapping: The snapping is realized through a snapping volume, a so called plane. These planes are attached on any side of the orbiter and one is inside of the orbiter. The planes cover the whole side they stick to, so the player has the freedom to snap the sensors to any place on the orbiter. To snap the sensors that are hold, it is needed to overlap one of these planes. Because of various sensor sizes, the pivot point of the sensor is the point that matters for overlapping, to avoid unintentional behaviour. Because the ghosting needs to be updated every tick and the computation of the ghost is pretty much the same for the snapping, we are snapping every tick. The place where the ghost appears and where the sensor is about to snap to, is computed by a line trace which is orthogonal to the orbiters side. The rotation of the sensor is given by the rotation of the plane, so the sensor always looks away from the orbiter, and the rotation the sensor had before it was dropped. To avoid that a sensor is dropped to deep inside of the orbiter, there is a stick plane inside of the orbiter. This plane is a little different than the others. The plane inside first has to compute the side of the orbiter which is closest to the sensor that should be dropped. Therefore a dot product is used; afterwards it is possible to make a line trace that faces against this side to compute the intersection point. To avoid that the rotation of the orbiter leads to unintentional snapping, a note is taken that no more snapping shall be done after the first attempt to snap failed. To avoid that the sensors cover other sensors or unevenness's on the orbiter the ghosting system is used. Therefore it is checked if the holographic copy of the sensor, which shows the place it would snap, overlap with other sensors or a MyBump volume which lay's under the surface of the orbiter side. If so, the sensor is about to snap to an invalid place. In this case, if the sensor is dropped, it will stay right where it is without snapping. This way it is also checked if it is overlapping sensors which look over the neighbour planes into the area of the plane it is supposed to snap to. If additionally the sensor overlaps something and it is dropped, the sensor will switch right back to the last valid Place.

G. CyberGlove

The CyberGloves and the Polhemus are not created in the same file, so the CyberGloves can be used separately, without being connected to the Polhemus receiver. This is useful to debug the gloves on their own and to calibrate the offsets for the finger movements.

CyberGloves

To ensure a comfortable feeling in virtual reality it is necessary to have 90 fps during an application. For that

reason the serial access of the CyberGloves is threaded.

The movement of the hand and its fingers is done by Forward Kinematic. This means the calculation is started at the root of the hand (wrist) and is calculated towards the leafs (fingertips).

The gloves meed top be calibrated, to estimate the minimum and maximum values of opening, closing and moving the hand. This done in a special map (unreal engine map/level), where the user has to move the fingers and hand to get all movement patterns. During this process the minimum and maximum values are constantly updated and after all movements are done the level can be closed. The gained values are then saved in a configuration file, so that the calibration has to be done only when the configuration files is deleted or the project is newly generated.

For the grab process a simple gesture recognition algorithm is used. The algorithm checks every frame if the fingers of hand are in a grab position, so basically if the hand is closed. This is done by asking for the finger curvature, comparing that to the average value for the grab position and adding an offset to the average value. The offsets ensure that the grabbing is not done to early or to late.

Polhemus

The Polhemus is also threaded, because of the same reason as the CyberGloves. This is here especially necessary, because without the threading the application only gets 30 fps. To ensure a smooth transition of the values a double buffering algorithm is also used.

Because noise effects the positions received from the Polhemus a k-means algorithm is used, to ensure smooth movements in the level. This is also necessary because the further away the receiver is away from the transmitter the worse the tracking becomes.

For the Polhemus it is necessary to calibrate the position in the world. This done similar to the CyberGloves. First the controllers of the HTC Vive have to placed so that the Polhemus transmitter is between them and afterwards the calibration map has to be started, where the average of the controller positions is taken. This average is than the position of the Polhemus in the application level. The average will be saved in a configuration file for further use in different levels and it is only reseted if the configuration file is deleted, changed or the calibration map is started again.

H. Constraints

The Constraint system allows every sensor to have multiple restrictions on placing it. These can be a limitation regarding the number of sensors on one orbiter Side or on the whole orbiter, sensors that create a prohibited constellation with other sensors or a sensor being restricted to the sun or shadow side. Each plane on the orbiter has a reference to the other planes respectively to this sensor, in example the left, right or opposite one, aswell as an Array that contains the attached sensors. We discussed about using an internal model of the Planes, saving all the planes in an Array and iterating through that Array later. We came to the conclusion, that it will not be more effective than using the references, although it would have been more in line with the observer pattern. Afterwards however it became clear, that using an internal model would have made the system more extensible and therefore better.

I. Menu

The menu is built as a widget, which is displayed by tipping the left hand / controller upwards. The menu items ”Reset”, ”Controls”, ”Load / Save”, ”Visualization” and ”Quit” are displayed on the first menu level. ”Reset” reloads the level, discarding all changes, ”Controls” loads the controller selection Level. Activating ”Load / Save” switches to an submenu where you can save and load spacecraft configurations. Under ”Visualization” there are, ”toggle Heatmap” which toggles the heatmap material for all objects, ”Launch” starts a simulation of the orbiter rotating around an asteroid by using a path spline controlled by the level blueprint and ”Camera POV” activates the visualisation of all camara pov’s. ”Quit” is used to exit the application.

1) HTV Vive Controller: To select a menu point with the HTC Vive controller, point the laser beam of the right controller at it, until the progress bar is completely filled. This is done by the menu checking if the laser beam is overlapping the menu point, then running a timer for the selected event, which will execute it if the laser is not removed during the timer running.

2) CyberGloves / Leap Motion: To select a menu point with the Cybergloves or the Leap Motion, click on it with the right hand. This is done by constantly checking the distance between the menu and the right index finger. If it exceeds a given threshold, a widget interaction is triggerd at the point closest to the right index finger.

All procedures apply to potential submenus, e.g. The ”Load / Save” menu.

J. Savegame

The loading and storage system represents the objects in the world in an abstract form, which we call ”PartStruct”. Such a PartStruct contains all attributes of the objects and their values in a storable format.
We collect all objects of the Supertype ”Part” in the form of their PartStruct in an array, which we want to store.
This array is stored in a local file using the Unreal SaveGame module. For loading the contents of the file, we use the Unreal SaveGame module again, and then spawn an instance of the respective object in the world, depending on the ”type” attribute, and, if necessary, set type-specific attributes.

K. InputSelectionPawn

The InputSelectionMap lets users choose an input device. The code to perform this selection is contained in the InputSelectionPawn. Every tick a line trace from the camera along its forward vector is performed. When a user looks at invisible colliders in the InputSelectionMap, the selection process progresses and it visualized with a circular indicator. The selection time can be configured in the SelectionTime variable. After that time is reached, the selection progress is reset and the InputSelectionPawn checks the collider's tag. If the collider is tagged with an input device's name, that name is saved, the selection is visualized in the level, and the 'Start' collider is spawned. Selecting the start collider transitions to the Spacecraft level and loads it with the game mode associated with the input device name.

Additionally, after selecting the CyberGlove game mode, a tagged actor for transitioning to the calibration map is spawned. And users can look at the 'Exit' collider to exit the application at any time.

III. Conclusion and future work

A. CyberGloves and Polhemus

Duo to time issues and problems with the implementation of these tools some features could not be achieved and some problems could not be solved. So for further work on this project they can be made to issues which need to be solved.

Polhemus: The biggest problem of the Polhemus is the short range and the affection of noise. By a k-means algorithm the problem of the noise was reduced but the range is still a matter. This problem could be solved by switching from the short range to the long range Polhemus transmitter. For this some additional calibration would be need to fit it to the output of the long range system.

One more problem was the calibration to fit the virtual reality world, because every time the tracking stations of the HTC Vive were changed it was also necessary to change the offsets of the Polhemus positions for the gloves, to set them correctly in the world. For this a algorithm would necessary, which gets the orientation of the HTC Vive world and from this orientation sets the position of the gloves correctly.

CyberGloves: A future issue for the gloves would be the calibration of the fingers, especially the thumb. To create a more realistic movement in the virtual reality the fingers would need some more tweaking, to be as realistic as possible and to mimic the movements of the real world counterparts. Because of time issues and problems of the implementation the thumb movement cloud not be set correctly and realistic. At the moment the thumb is barley movable. To solve this it is necessary to try out some values on the thumb movement and see which rotations affect the thumb correctly to the CyberGlove thumb rotations.