Virtual Reality for Space Science Outreach and Research

Markku Alho

October 23, 2017

Abstract

Virtual and augmented reality devices provide a novel, easily approachable and fast evolving medium for public outreach, and possibly a valuable tool for actual science. This report presents a short review of available equipment and software and presents the virtual reality demonstrator built for *Avaruusrekka 2017*, a space science and technology outreach event/expo truck in fall of 2017. Basic principles of the design of the software are given, with critical consideration on user experience and usability, including Visually Induced Motion Sickness. Improvements on the state of the art are discussed, and further consideration is given to scientific applications of a more advanced software. Some libraries and possible designs for science software are discussed, and some guidelines on how to generate visuals are noted at the end of the report.

Contents

1 Introduction **3**

2 VR and AR solutions **3**
2.1 State-of-the-Art consumer devices . . . 3
2.1.1 HTC Vive . . . 3
2.1.2 Oculus Rift . . . 3
2.1.3 Microsoft Mixed Reality . . . 3
2.1.4 PlayStation VR . . . 4
2.1.5 Cellphone-based . . . 4
2.2 Near future developments . . . 4
2.2.1 Microsoft Hololens . . . 4
2.2.2 Varjo . . . 4
2.3 Software . . . 5
2.3.1 Unity3D . . . 5
2.3.2 Unreal Engine . . . 5
2.3.3 Blender . . . 5
2.3.4 ParaView . . . 5

3 Floating in Space: VR for Space Outreach **5**
3.1 Design choices . . . 5
3.1.1 Visuals . . . 6
3.1.2 Intermission: Visually Induced Motion Sickness and disembodiment . . . 9
3.1.3 Movement and controls . . . 10
3.2 Description of scenarios . . . 12
3.2.1 The Earth system . . . 12
3.2.2 67P/Churyumov-Gerasimenko . . . 17
3.2.3 More scenarios . . . 17
3.3 Additional development . . . 18
3.3.1 Multi-platform compatibility and distribution . . . 19
3.4 Expo usage and observations . . . 20
3.5 Outreach: conclusions . . . 20

4 Diving to Data: VR for Space Research **22**

5 Software tips & tricks **23**
5.1 ParaView to Unity . . . 23
5.2 Unity . . . 24

1 Introduction

A short review of available virtual reality (VR) and augmented reality (AR) solutions is presented at first, with some considerations on software implementation. The reviews for hardware, software and software libraries are hardly exhaustive, but rather try and serve to produce an overall picture of the possibilities. For most part, the report focuses on applications as seen from the vantage point of a modeler – specifically, one of space plasma physics, and therefore, references to data usually concern 3-D simulations.

2 VR and AR solutions

2.1 State-of-the-Art consumer devices

The current state of the art is represented by two dedicated head mounted display (HMD) sets, the Vive and the Rift, and smartphone-based solutions like the Samsung Gear VR. The dedicated HMDs, with peripherals, achieve sub-millimeter location and corresponding orientation tracking in up to room-scale environments, while the smartphone-based solutions, relying on internal smartphone positional trackers, tend to have less positional accuracy, but potentially at a greatly reduced price.

2.1.1 HTC Vive

Developed by the Taiwanese smartphone manufacturer HTC, the Vive setup consists of a head-mounted display (HMD), two base stations, two controllers and a link box. The HMD and the controllers contain a set of photodetectors, while the Lighthouse-dubbed base stations fill the tracking volume with structured light[1]. The HMD contains a 2160x1200 pixel display, split to 1080x1200 pixels per eye.

The system is supported on Linux since February 2017, and on Mac OS X since summer 2017. Valve's SteamVR software is used to run the system, which can be interfaced through Valve's OpenVR library. OpenVR is designed to be independent of the actual equipment used, and Valve is as well working in collaboration with the Open Source VR (OSVR) project.

2.1.2 Oculus Rift

The Oculus Rift system is quite comparable to the HTC Vive, especially after publishing room-scale VR and dedicated controller components.

2.1.3 Microsoft Mixed Reality

Microsoft recently announced[2] its "Mixed Reality" platform for a VR-based Windows operating system. Several hardware producers have introduced head-mounted displays with controllers, at a lower price point than HTC or Oculus, so

far. Hardware specifications are similar to Vive and Rift, but with more emphasis on portability: the position tracking system is contained entirely within the headsets, requiring no external setup. Reportedly, this produces slightly worse tracking than with the competition. Additionally, these headsets have use dual cameras for tracking that might be used to display the actual surroundings as the backdrop of overlaid VR objects, but no such applications have been presented by Microsoft as of yet. The term "Mixed Reality" itself does not really describe anything else than VR so far.

2.1.4 PlayStation VR

An additional VR peripheral to the console gaming system, at slightly a lower resolution than the Vive and Rift.

2.1.5 Cellphone-based

Google Cardboard and Samsung Gear VR[1] are examples of VR systems utilizing a cheap (in the case of Google Cardboard, literally of made of cardboard) holder for a high-resolution smartphone, the screen of which provides the VR system its display and processing power. These systems generally fall behind dedicated systems in processing power and graphics quality, motion tracking, field of view and screen refresh rate, but, depending on the smartphone, may actually surpass the number of pixels available in the dedicated systems. Some systems include additional control peripherals.

Augmented reality software, in which the smartphone functions only as a window into the virtual view (perhaps with the phone's camera providing the backdrop) are quite commonplace (see e.g. Pokemon Go), and with no additional peripherals required, could be a more accessible medium for wide deployment and distribution of outreach software.

2.2 Near future developments

2.2.1 Microsoft Hololens

An augmented reality peripheral by Microsoft. A transparent lens system and built-in awareness of surroundings allow the system to render augmented reality elements on top of the real world. Reportedly[3], the prototypes, as of yet, have quite restricted field of view, seen as a serious detriment in the immersion. However, the system has received praise for the concept besides the FoV issue.

2.2.2 Varjo

Varjo is a Finnish startup developing extremely high pixel density VR displays. Their goal is to provide a display, in which high pixel-per-inch regions are rendered on-demand as requested by eye tracking, with less resolution used at peripheral areas. This would both mimic the operation of the human eye (that

[1] Developed in collaboration with Oculus

records high-resolution data from a relatively narrow field of view, with peripheral areas processed at a much lower fidelity) and enable contemporary graphics processing units operate at the required huge (effective) resolution.

2.3 Software

This section lists different software solutions for developing VR and AR software.

2.3.1 Unity3D

Unity[4] is the game engine and game development software used in producing the prototype and the demonstrator. The software is an industry standard, with notable users (and products) including the Finnish studio Colossal Order (e.g. Cities: Skylines) and Squad (Kerbal Space Program; including an education version as well).

2.3.2 Unreal Engine

Unreal Engine[5] is another game engine and game development software. Unreal Engine is another industry standard solution, famous for high-quality graphics, and is somewhat comparable to Unity.

2.3.3 Blender

The open-source Blender[6] software, mostly used for 3-D modeling and art, includes its own game engine and, nowadays, VR support.

2.3.4 ParaView

ParaView[7], an open source 3-D -visualization software, includes a rudimentary VR branch, with little features besides displaying plots in VR.

3 Floating in Space: VR for Space Outreach

This section first introduces scenarios that have been considered and implemented for the outreach demonstrator, and continues by describing the design solutions, both tried and implemented. Henceforth, references to VR (development) software will refer to the Unity software used to develop the demonstrator, and references to VR gear will be of the HTC Vive system.

3.1 Design choices

This section explores the design of both the informative (and hopefully appealing) visuals and the user interface of the software.

3.1.1 Visuals

Choosing how to generate a certain set of visuals depends, naturally, on the nature of the dataset being visualized. Plasma physics simulations, for example, generate scalar and vector fields (and in general, tensor fields as well) over the simulation domain. Meaningful visualization of these fields is necessary for correctly interpreting the generated datasets, and dedicated software exists for this task, such as VTK[8] and the VTK-based ParaView[7] and VisIt[9].

For scalar fields, a volumetrically rendered cloud would, theoretically, be the definitive solution for displaying scalar data in the whole of the 3D domain at once. This is, however, expensive both in terms of rendering and data storage, and might require a lot of fine-tuning [2]. So far, I have resorted to using colored isocontour surfaces to present scalar fields, as these are readily usable. That said, there are some open raycasting volumetric rendering libraries for Unity that could be used to achieve proper real-time volumetric renders. See e.g. Unity-RayTracing[10] and unity-ray-marching[11].

Displaying a set of surfaces in 3D can be done in a few different manners using different shaders. One option would be to render the surfaces as wholly opaque, creating a solid object. Although such a surface is very tangible and concrete, it would limit the presented data to a single surface (especially in the case of nested surfaces). However, having nested surfaces, especially in the case of isocontour surfaces is more than probable, so translucent surfaces are a potential solution. Even when properly shaded (e.g. with a Lambertian lighting model), these surfaces can blend together into a somewhat unintelligible, bland mess. Adding rim shading allows the designer to highlight the tangential portions of the surfaces and to fade out the surface portions whose normals are facing the viewer. The normal-aligned portions of the surface contribute little to the actual perception of the surface, so those can be made nearly fully transparent. This allows interpreting the 3D shape of the surface, while allowing for an unobstructed view of the innards of the system. Figure 1 on the following page demonstrates the differences between different shader choices.

It is noteworthy that rendering complex translucent surfaces can produce errors due to depth sorting issues[12]. These are practically unavoidable, and may manifest themselves as flickering surfaces, especially when the surfaces are lying very nearly on top of each others. Additionally, as the mesh surfaces are oriented, one needs to account for inside and outside meshes, either through somewhat involved shader scripting or using a duplicate mesh with its surface normals inverted. The latter was employed in the demonstrator, as it is very easy to implement, and perhaps more resilient against sorting errors.

Vector field visualizations are more simply and intuitively produced with streamlines, directly. So far, there is a technical issue with directly using stream-

[2] In my personal experience, ParaView usually does nice volumetric plots with default settings, while VisIt requires a lot of adjustment. Doing this properly inside e.g. Unity is another can of worms that I have not yet delved into, at all

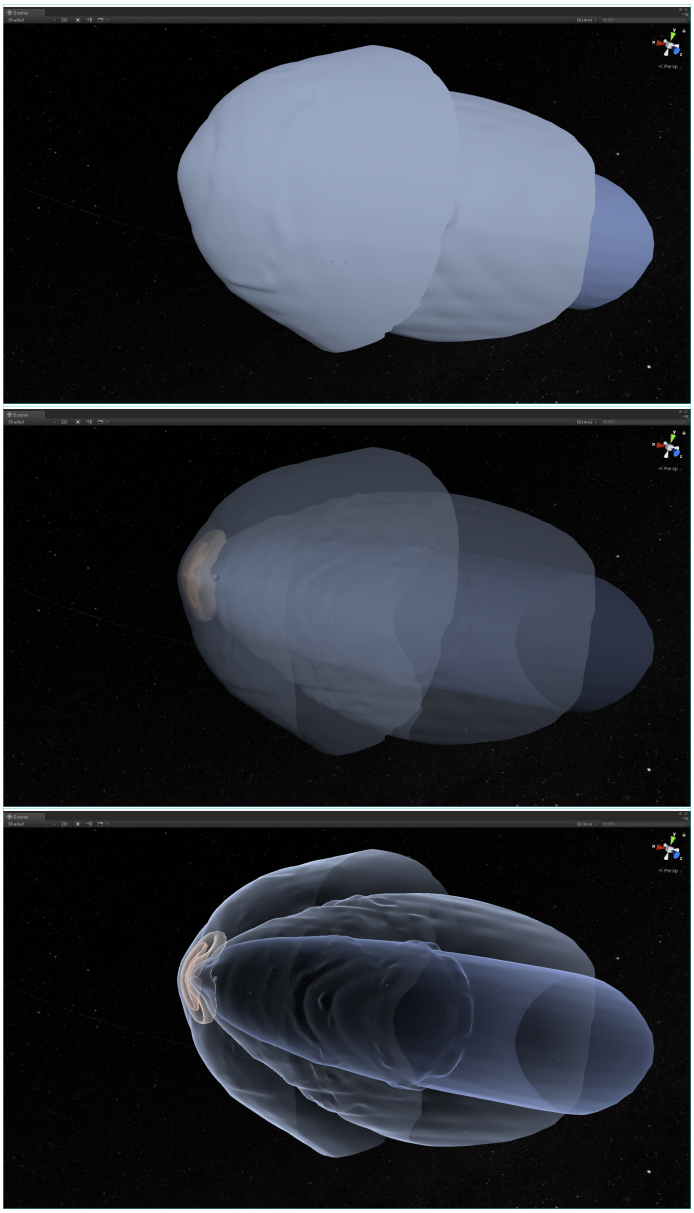


Figure 1: Opaque (top) vs. translucent (middle) vs. rim-shaded (bottom) surfaces, displayed using isocontours of plasma density in the Earth's magnetosphere.

lines from ParaView or VisIt: neither has yet displayed capabilities of exporting line meshes as-is, so e.g. with ParaView I have employed the tube filter to generate a triangulated mesh for the streamlines, imported these to Unity, and subsequently applied a normal extrusion shader to retain streamline visibility regardless of perspective effects. Displacing the vertices of the tube along the normal direction, by a distance proportionate to the distance between the vertex and the camera position works well enough for most occasions.

An alternative method of displaying streamlines would be to use a vector graphics library to draw the lines, such as Vectrosity[13]. Exporting streamlines in CSV from ParaView should do the trick[3], but this has not yet been implemented (see section 3.1.3 on page 10 for additional points).

Another alternative for vector field visualization is by using the vector field to ascribe manifolds and surfaces thereof that purport some meaning. NASA visuals (employing a rim shading effect) are visible here. In the clip, rim shaded surfaces are used to visualize a vector field, namely that of the magnetic field of the Earth. The actual definition of the surfaces in the NASA artist vision is not clear, but surfaces like those could be defined, for example, by taking bundles of field lines at constant latitudes on some small sphere outside the magnetic moment source region of the Earth. The field lines, when propagated from such a constant-latitude surface, would span these sorts of "magnetic bubbles", possibly useful for showing actual X-line configurations and propagation of magnetic flux in a Dungey cycle. L-shell surfaces can also be produced by seeding the "manifold field lines" at the magnetic equator, from points of constant equatorial radius. See section 5 on page 23 for an example of L-shell plotting. One should note that explaining these concepts to an audience member might not be straightforward.

Animated surfaces can be implemented in several flavors. Simple pulsing surfaces can be constructed within Unity, by using the animation controls and e.g. scaling a primitive object. Another way would be to use separate meshes for each animation frame, but this can produce choppy animations (unless almost each render frame has its own mesh, which, at 90 Hz, can lead to unwieldy datasets). Animated surfaces from ray tracing data by Mathias Fontell were included through simple mesh deformations (basically, an implementation of mesh keyframing[14]). The animated raytrace surfaces were presented by a set of vertices for each ray at given time steps (keyframes), extracted from the raytraces. The initial configuration of the rays (initial elevation and azimuth of the ray) was converted to a triangulated topology via *Delaunay triangulation*, forming an initial mesh.

Thereafter, the mesh vertex positions were linearly interpolated towards the corresponding positions at the next raytrace timestep on each frame update, with some suitably chosen velocity. The topology of the surface is constant in this sort of a deformation, which is both easy to implement and delivers rea-

[3]This is achieved through the Save data menu option, instead of export scene option in ParaView–this took a surprising while to notice!

sonable physical intuition. Improvements for the implemented mesh keyframing could consist of e.g. pre-calculating normals for keyframes, instead of recalculating them for each frame (which is quite suboptimal) and optimizing timesteps (constant times for ray paths are not strictly required: instead, ray-specific keyframes with correct time stamps could be used to reduce memory usage). In terms of physics, implementing some branch cuts between topological neighbours in the case of large separation would be advisable (e.g. when one ray refracts through the ionosphere and the next one reflects back to the ground), perhaps through removing the offending triangles.

Deforming surfaces between non-isomorphic meshes, or how to generate sets of isomorphic meshes to be used as keyframes from arbitrary data, has not been looked into in this scope. For visualizations of time-dependent advanced simulations, this might very well be necessary.

3.1.2 Intermission: Visually Induced Motion Sickness and disembodiment

Visually Induced Motion Sickness (VIMS), an area of active research in itself, is a feeling of nausea experienced when using immersive displays, due to mismatched sensory inputs from the eyes and the vestibular system of the user. The effect seems to be most pronounced with virtual rotations and head movements without a corresponding visual cue. The former van be produced by rotating the user camera in the virtual space, and the latter may arise from insufficient hardware (display lag) or software issues (interrupted rendering and/or head tracking). Linear vection, or apparent self-motion may contribute to VIMS as well, but relationship between VIMS and linear vection is not as clear as with head motions, virtual or uncompensated, that produce circular vection: the human sensory system is not as concerned with linear acceleration as with angular accelerations (see e.g. the vestibular system that is dedicated to sensing angular acceleration). Further examination of e.g. [15] and others should be performed. In particular, the results in [16] should be looked into, as they could be readily implemented to predict VIMS onset and to trigger VIMS-diminishing features (or limit VIMS-inducing features)–provided the proposed method actually works.

Some guidelines recommend using discontinuous velocities for the user's camera, as this removes visual cues of acceleration. Some recommendations are to not translate the user camera at all, but to "teleport" the user to target locations (as done, for example, in the SteamVR Home environments).

Another VR peculiarity is the feeling of disembodiment experienced by the users, when they cannot see their own limbs moving in the virtual world. Having the accurately mapped controllers as visible proxies for the user's hands could help a bit, but a proper solution would be to have a proper inverse kinematics avatar, that tries to calculate limb poses from controller position and orientation, using some realistically constrained set of joints and bones. FinalIK is

a potential solution[4], along others. An inverse kinematics avatar is yet to be implemented due to time constraints.

3.1.3 Movement and controls

The movement and control scheme in the demonstrator was designed, for the most part, to try and avoid the VIMS problem. In the end, however, this requirement had to be relaxed to provide for an easily approachable tour of the environment.

The basic VIMS-avoiding control scheme was built around a specific intuition: If you hold an object in your hand, and translate and/or rotate the said object manually, your brain readily understands that rotation to apply to the held object, and not to a translation and rotation of your head. This would seem an obvious fact of human perception, but would the same mapping work for arbitrarily large objects, or, indeed, the entirety of your environment? VR gear allows one to test this hypothesis, and although not rigorously tested here, it would seem to be the case: mapping the rotation of the user's controller to the entirety of the environment (or, equivalently in terms of user perception: inversely mapping the transformation to the user avatar) would seem to hijack this neural pathway for the benefit of the software designer.

The basic scheme, built around this hypothesis, is as follows, and is symmetrical with respect to controllers:

Translation Initiated with a slight pull of the trigger of a single controller. Maps the movement of the single-point location of the controller to an inverse displacement of the user avatar. Controller velocity is retained by the avatar to allow movement by floating through space.

Rotation Initiated with a full pull of the trigger of a single controller. Map the movement of the single controller as above, and additionally maps the angular velocity of the controller to an inverse angular displacement of the user avatar around the control point[5].

Scaling Initiated by pulling the triggers of both controllers. Allows the user to stretch and contract the world (or the inversely, the player character). Allows translation and rotation as above, although by using the mean location of the controllers as a control point for translation, and the mean angular velocity of the controllers around this control point for rotation. The (inverse) scale change of the environment (avatar) is calculated to correspond to the change between the distance between the controllers, and the position of the control point is held fixed between the scale changes, and along with rotation and translation, the controller locations will be

[4]Handily included in the VR Essentials package included in the purchase of Unity license.

[5]Rotating the scene around the controller would work as well, but this would require an additional skybox camera with separate functionality.

fixed during the scale change as well. This allows for e.g. intuitively grabbing the Earth at separate sides and stretching it to a desired size, with a constant mapping of the controllers to the fixed points that the user took hold of in the beginning.

The scheme seems very robust with respect to translation[6] and rotation, but unlimited scaling can lead to floating point errors, and new users tend to be confused by the scaling function. Stretching the universe is not as intuitive as holding a cup of coffee. Users tend to use the scaling function as a type of pinch-to-zoom function, which leads to unsatisfactory navigation as users continually stretch the universe to get closer to some object of interest. Instead, the target object recedes into distance, as the universe in between expands relative to the size of the avatar. Also, the scaling function is somewhat awkward when transitioning between large scale differences, requiring several full stretches or compressions to reach the desired scale.

That said, the author has been quite content with the scaling function[7], and plans to retain it for advanced users, as intuitively and coherently mapped stretching and contraction are quite pleasant tools to work with. An additional scaling function was implemented during the tour:

Single point scaling Initiated with press of trackpad, using the trackpad axis to determine scaling direction (stretch/contract) and possibly scaling speed. Controller position is used as the scaling pivot, so the user can use the controller as a 3D zoom function. The velocity of the user is set to zero during scaling.

The scaling speed in the one-handed scheme is not strictly tied to actual movements, and therefore can greatly exceed the one of two-handed scaling, which is very useful. The addition of a single-controller scaling function also completes a single-controller control scheme, which is quite advantageous during prolonged expo use, as one controller can always be left to recharge, and the single controller scheme is also easier to instruct in the use of.

Additionally, a point-and-click navigation method was tried: A "laser pointer" from the controller[8] was used to point and select a target object and position (including chosen translucent surfaces), and the player would be smoothly translated to such a location that the pointed object would be some 0.5 meters from the user controller. As pure translation manifests only as linear vection, with no rotation, it sidesteps the main VIMS problem of unmatched rotations.

Outreach user experience needed to be easily approachable, however, as outreach users, in general, have to be manually instructed in the use of both the VR system and the controls. The control scheme was retained, but navigation within the demonstrator was mostly organized through a menu and a set of

[6] Floating point errors related to translation can be dealt with by re-centering the scene so that the user avatar is always close to the origin - most often visible when combined with extremely small scales.

[7] Potentially delaying the design and implementation of new scaling functions...

[8] Technically, a raycast with a pointer graphic

preset tour points. The user controls were simplified from prototype versions to the above controls, i.e. everything is done with a single button (the trigger). The tour points also present a handy container for short audio narrations of the environment, reducing the workload of instructors. A brief tutorial video of the controls was included in an auxiliary dashboard, as well as acknowledgments and credits.

Tour points, or shortcuts, were defined by a location, view direction and a scale with respect to initial scale, and whenever the user chooses to travel to a tour point via menu, the user's location, rotation and scale are linearly interpolated to the new location and orientation. In a space environment, however, the scales and distances can vary wildly, and to account for single-precision floating point errors, especially when the target tour point is far away, the user is initially scaled up (or rather, the environment is scaled down), producing a sensation of growth for the user and subsequently allowing the linear translation to function without errors. The scale-translate system works currently when going from small scale to large scale (and to correspondingly large distances), extension to work automatically in all cases should still be implemented.

Potential improvements to the tour functionality would be to use an (optional) VIMS-setting, that would render some static overlay grid for the duration of rotations, to give the user a set of fixed, non-rotating points to focus on. The dashboard menus of the demonstrator do function in this way, provided the user actually looks at them during a transition (which is improbable, as the visuals during transitions can be pretty nice and informative). Another option would be to use only translation for the transitions, which would restrict the predefined tour views to a given angle, which is not very practical.

The dashboard, operated with a laser pointer function, consists of a virtual satellite camera view of the *Suomi 100* satellite, satellite and user position and velocity data, a running spectrogram of user audio (to demonstrate sonified space radio emissions), and controls for VR overlays (such as enable/disable specific visualizations, or change the Blue Marble month) and the aforementioned shortcuts. See figure 2 on the next page for the dashboard layout.

3.2 Description of scenarios

Quite a few space physics and remote sensing scenarios were considered for the outreach demonstrator, ranging from cometary nuclei to the entire solar system. However, as dictated by resources and logistics, only the case of Earth and its magnetosphere were implemented for the demonstrator. Especially, as the Earth system is already quite complex, the smooth operation of the demonstrator was seen to require a compact set of items to be presented.

3.2.1 The Earth system

The Earth system, as finally presented in the demonstrator, consists of the Earth's magnetosphere (Antti Lakka, GUMICS-4 MHD simulation), radio wave

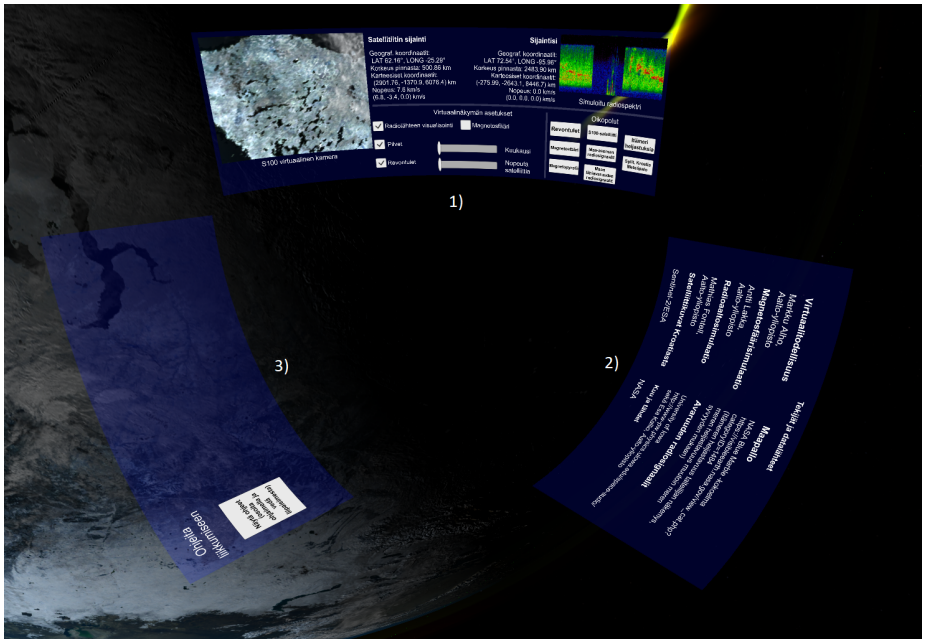
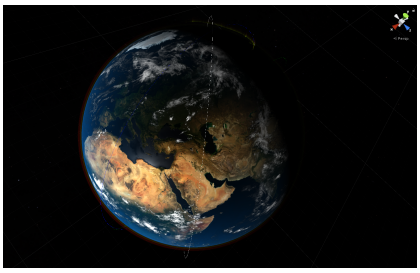
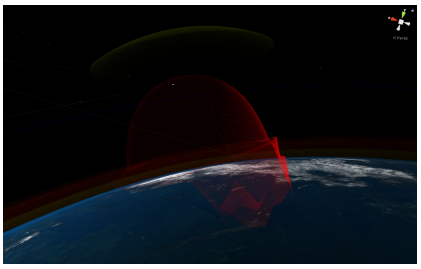


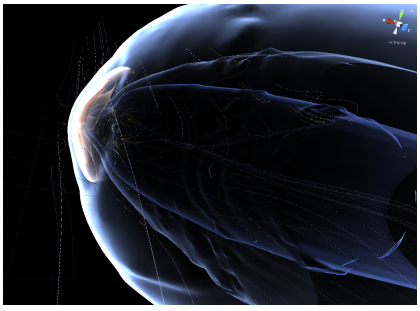
Figure 2: The demonstrator dashboards: 1) The main controls dashboard, with controls and data displays. 2) Credits panel. 3) Instruction video. The user is situated in the middle, with the dashboard tops being roughly at waist level.



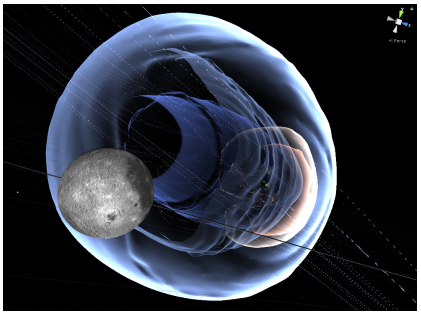
(a) The Earth, with the NASA Blue Marble dataset and the orbits of *Suomi 100* (blue), *Aalto-1* (white) and the *International Space Station* (green). Nightside aurora borealis (adjusted pre-made effect) is seen above the Arctic. Ionospheric airglow is also visible.



(b) Ray trace surfaces of Mathias Fontell implemented in Unity, emanating from Otaniemi ground station.



(c) Global magnetospheric MHD simulation from GUMICS-4 by Antti Lakka, visualized by the author. Magnetic field lines, including two reconnection sites and solar wind density contour surfaces are displayed from this view.



(d) The dark side of the Moon (mostly lit), visible against the backdrop of GUMICS-4 simulations

Figure 3: The Earth near-space environment, implement in Unity, chosen features.

propagation in the ionosphere (Mathias Fontell, Matray raytracing software), ionospheric airglow and aurorae (artist's vision), remote sensing data from the NASA Blue Marble Next Generation collection[17] as the actual Earth, and ESA Copernicus/Sentinel-2 imagery[18] for a specific Earth Observation demo. The Moon was additionally included, but not very much advertised, as the *Lunar Reconnaissance Orbiter* dataset height maps[19] suffered from import problems (albedo map used was from LRO WAC[20] as well, without problems). Chosen space physics portion of the demonstrator is displayed in figure 3, and Earth observation features in figure 4 on page 16.

The magnetosphere is represented by magnetic field lines of the global dipole field, compressed and stretched by the solar wind, and by the isocontour surfaces of solar wind density. The bow shock, magnetopause and tail lobes, as well as both dayside and tail reconnection is visible in the dataset. Tour points for the demonstrator were set to an overlook position in front of the magnetosphere, to the tail X-line and to the nightside aurorae. Visualizations of L-shells, radiation belts and current systems are natural further inclusions for a more developed demonstrator software.

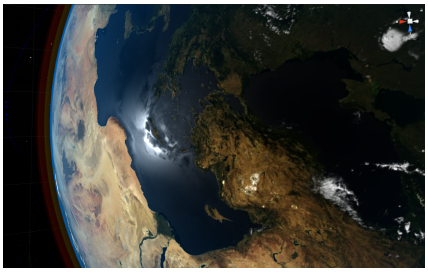
Additionally, space audio recordings (that is, sonified radio signals) from the near space of Earth, produced by the University of Iowa[21], were included as spatial sources of audio in the demonstrator. The avatar's audio stream was run through Fourier analysis and displayed as a running spectrogram in the user dashboard.

Extension to remote sensing is a natural inclusion in the outreach demonstrator, as the *Suomi 100* satellite does contain a wide-angle VIS camera. A corresponding camera was specified to be contained in the avatar of the *Suomi 100* satellite, rendering to a texture visible on the user's dashboard. The virtual camera was equipped with post-processing effects that adjust the effective exposure values of the camera. The prototype version enabled the user to interact with the satellite avatar, rotating (and even throwing it) in space, to test different camera angles. This feature was not included in the truck demonstrator.

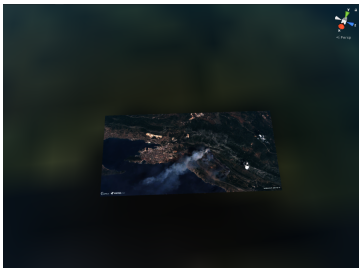
Remote sensing elements displayed in the demonstrator include: sea glitter (variable reflectivity/smoothness of oceans, Blue Marble bathymetry data used instead of actual wave/wind data), global monthly datasets and cloud cover snapshot (NASA Blue Marble, next generation), and ESA Copernicus/Sentinel-2 imagery of forest fires in Split, Croatia in the summer of 2017. Additionally, upper atmospheric lightning (sprites, elves and blue jets) were planned for implementation, but were deferred for a later date. Virtual *Suomi 100* imagery can be "splatted" onto the Earth (see section 5.2 on page 24) already in the prototype project, providing an additional publication medium for satellite results.

Other remote sensing instruments besides VIS band imaging can be envisioned. In a demo setting, hyperspectral imaging data cubes from *Aalto-1* could be included, with spectral channels displayed e.g. as stacked images, and an interface to control layer visibility and blending. Virtual cameras in Unity could be configured to image a different set of objects and textures, presenting e.g. Earth surface reflectivity at IR bands. Similar method could be used to provide a virtual ENA[9] instrument to reveal an otherwise hidden ENA emissivity distribution (implemented either through surfaces, or volumetric rendering once implemented). See Vectrosity "X-ray vision" demo for an illustration of potential functionality.

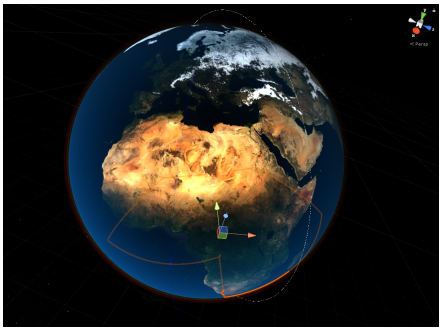
[9] Energetic Neutral Atom



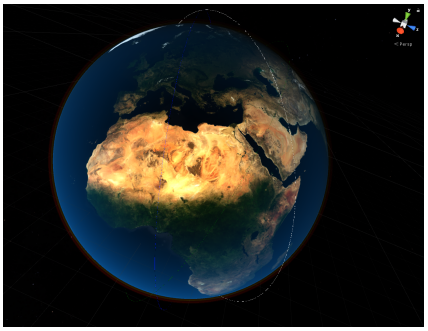
(a) Variable reflections of sunlight from the Mediterranean, emulating reflectivity modulation due to sea surface roughness. Some clouds visible.



(b) Split, Croatia: Forest fires of summer 2017, as seen by ESA Copernicus program *Sentinel-2*, overlaid on global data. Images before, during, and after the fires included. Note the low-resolution background due to technical limitations.



(c) The Old World hemisphere, no clouds, as produced from NASA Blue Marble. January 2004.



(d) The Old World hemisphere, no clouds, as produced from NASA Blue Marble. September 2004.

Figure 4: The Earth Observation system in Unity, chosen features. Monthly datasets from Blue Marble can be browsed in the demonstrator (see Figs. 4c and 4d)

3.2.2 67P/Churyumov-Gerasimenko

The original prototype scenario consisted of the plasma environment, simulated with the Aalto hybrid plasma code, of the comet 67P/Churyumov-Gerasimenko at approximately 2.4 AU from the Sun. One of the most striking features of the system are the large differences in scales, even for this relatively weak comet relatively far from the Sun, and the VR environment works very well in displaying the differences in scale: The Rosetta probe, included in the prototype (courtesy of NASA[22]), has its solar panels spanning some 30 m, while the cometary nucleus (Matthias Malmer's shape model and textures[23]) is some 4 km across, and the whole simulation domain is some ten thousand kilometers across.

Data included in the prototype, besides the aforementioned shape models, include cometary ion density isocontour surfaces (chosen at suitable values to spread the surfaces evenly across scales), solar wind proton bulk flow (in the sense of average motion, as the movement of the particles is kinetic in nature), and the draping of solar wind magnetic field lines by the cometary plasma environment. Colormaps and controls to affect data visibility are included in the hand-held menu. The Earth was included for scale (and can be seen to disappear in the distance at points, as the floating-point range of a single camera[10] clip planes is not sufficient with large scale differences).

For reference, see a screen capture video[24] of the 67P prototype in action, with the two-handed control scheme seen in action, and with a setup of a hand-held menu and info screen (replaced with the curved dashboard in the outreach demo). Some advanced control and measurement features, likewise cut from the Avaruusrekka demonstrator, are also seen in the clip.

The 67P prototype was not included in the public demonstrator (yet!) to keep the expo demo concise.

3.2.3 More scenarios

The Aalto hybrid model provides possibilities for demonstrating plasma environments across the solar system, with the author having simulated Mercury, Venus, Mars, comets, and parts of the lunar plasma environment. A long-term plan would be to include basically the whole of the solar system plasma environments in a single demonstrator across the scales, in collaboration with other groups capable of delivering some missing pieces of the system, such as University of Helsinki solar corona simulations. CMEs in the scale of the heliosphere, along with some backdrop of regular solar wind, could be presented as well. Including the huge Jovian and Kronian magnetospheres would be interesting as well, for example through Ilja Honkonen's MHD simulations, with inner parts of these systems, like Titan, presented with high-fidelity models. Having the reasonably well functioning prototype and the Avaruusrekka expo demonstrator enable subsequent additions in a relatively fast workflow.

[10] multiple VR-enabled cameras were introduced in a new version of Unity

3.3 Additional development

Additional technical developments , esp. with remote sensing, would include streaming textures for the globe, to accommodate resolutions at the 100 m scale (or even further). Unity engine constraints limit textures to 8192x8192 pixels, maximum, without the use of sparse textures[11]. The Granite library for Unity is being considered as a solution to enable streaming textures, with less overhead for actually implementing the streaming features. Using other streaming solutions, such as Google Earth APIs or NASA WorldWind should be looked into. Sadly, the WorldWind C♯ implementation[25] has been discontinued. Google Earth Engine API works on JavaScript, so this could be potentially useful, although the API is restricted to evaluators only at this point, and use restrictions may apply - and it's not very clear whether or not the API can be interfaced with Unity, either.

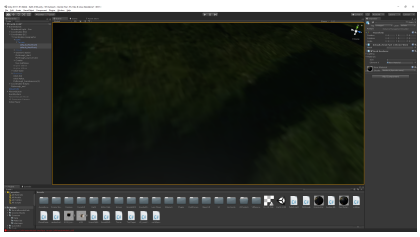
Interfacing to remote sensing APIs such as NASA WorldView/WorldWind or Google Earth Engine, would be very beneficial, as one could directly interface to huge Earth observation datasets through these APIs. Both terrain models and textures could be readily obtained through the Internet, reducing the size of the demonstrator build immensely. By default, the services provide robust GIS data, reducing developer overhead. Lack of good Internet connectivity might be a problem at certain settings, but the benefits would outweight the problem, and some lower-resolution data can always be retained as a fallback option.

Granite would enable using textures with dimensions of 262144x262144, reaching a global resolution at the equator of roughly 40000 km/262144 px$\approx$ 150 m/px, while the NASA Blue Marble datasets have a corresponding resolution of about 500 m/px. However, additional custom solutions are required to implement e.g. monthly datasets (probably via re-mapping UV coordinates onto a concatenated texture atlas of monthly datasets), and the library does have some technical restrictions with respect to the number of textures accessible from a single tileset. See figure 5 on the next page for resolution comparisons.

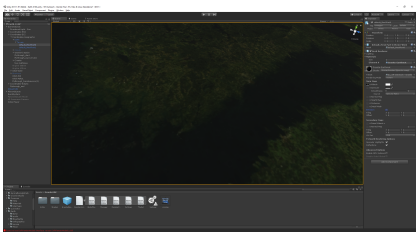
Scales, distances and floating points don't mix very well, and some more flexible solution than now employed (scaling only the user avatar or the scene) is required for the smooth inclusion of the large scale differences and distances in space. Scaling the scene proved to be problematic for proper rendering of translucent objects, so either a combined or avatar-based scaling system is to be developed further. Some floating-point errors are already resolved via the use of scene recentering, so that the user avatar always resides very close to the origin, where the floating-point precision is the highest.

A hierarchical set of floating-point coordinates could be a solution for the problem, some of which has already been implemented in the demonstrator for

[11] Unity includes a sparse textures feature, that taps into DX11.2 tiled resources, but its use requires plenty of manual coding

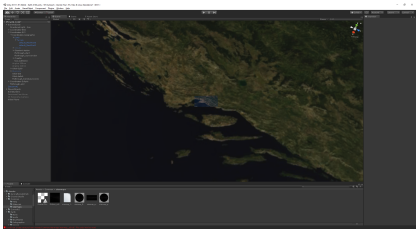


(a) Southern Finland, seen at 8192 pixels/parallel.



(b) Southern Finland, seen at 86400 pixels/parallel.

(c) Croatian coastline and overlaid forest fires imagery, seen at 8192 pixels/parallel.



(d) Croatian coastline and overlaid forest fires imagery, seen at 86400 pixels/parallel.

Figure 5: Granite library improvements to resolution, from Unity-constrained to full Blue Marble resolution.

easy handling of different celestial and geographic coordinate systems. However, the implemented coordinate systems do not (yet) contain scaling information. For reference of a working, Unity-based solution, the reader is directed to the PC game Kerbal Space Program, which actually has its own education branch for space and aerotech purposes, as well.

Orbits of satellites, including the Moon, are already generated from the set of orbital parameters for each object, for some given epoch, and can be readily attached to corresponding coordinate systems. Interfacing to Space-Track or SPICE kernels and having an automated epoch system would introduce a fine addition to the system.

3.3.1 Multi-platform compatibility and distribution

Using the Unity software, the demonstrator is readily deployed to almost all platforms, from HTC Vive on Windows PC to Google Cardboard, and with some modification to a less immersive environments, to AR applications on smartphones and web browsers. Essentially, the user interface and visual settings need to be adjusted to account for deployment on e.g. touchscreen systems and comparatively lightweight hardware.

The space truck outreach demo could be distributed as-is, available to all

users HTC Vive access, or suitable middleware to use Vive software with e.g. Oculus or other devices (which would be at the user's discretion, with no developer support required). Publication and free availability of the demonstrator would be advisable for significantly improved outreach impact, but some additional work should be put into the current demo, e.g. localization to English, for example. In a solely user-driven environment, there are no additional, external issues of available demonstration time, as in the case of the space truck setting with significant queues observed.

3.4 Expo usage and observations

The VR system has proven to be quite a popular attraction during the truck tour, with quite a lot of positive feedback: the demonstrator has been described as very informative, entertaining and captivating, and as such could be called an unmitigated outreach success. Even with the suboptimal VIMS-avoidance of the touring system, instances of symptoms are quite rarely reported, and manageable when reported.

The main problem of the system is the single-user limitation of the gear, which can and will lead to long queues. Queue mitigation procedures, usually effective, include showing only select portions of the demonstrator, and in the extreme cases, only a very general overview with no user interaction. User feedback has been quite good even in this case, but from the view of the presenter, this sort of heavy attendance is quite tiring and unfulfilling, as a lot of the material is discarded for the hurried viewers. Especially in the case of (young) children, it is better to not divulge the controls to the young user, with the presenter controlling the demo manually. This is very handy in the case of large school groups. Care must be taken when using the VIMS-unfriendly touring function without the dashboard menus visible. Going through the whole demonstrator with an inspired audience member, however, is very rewarding. These claims can be corroborated by the permanent expo staff.

As an extreme case of heavy attendance, the Avaruusrekka event on the 29th of September, during the European Researcher's Night in Jyväskylä, Finland, drew an estimated audience of over 5000 viewers in the truck expo itself, with an unestimated attendance at the VR demonstrator–suffice to say, that there was a lengthy (10-20 people) queue for almost the whole duration of the night, starting immediately from 16 o'clock and lasting to 23 o'clock, with the action subsiding towards midnight and the cessation of the day's expo. Even with heavy use of the aforementioned queue mitigation techniques, it is probable that a significant amount of potential users yielded from participating in the demonstrator.

3.5 Outreach: conclusions

The outreach demonstrator has produced a good audience response, even in the limited scale that the users have been able to explore the demonstrator. With the growing prevalence of VR gear, especially with smart phones, having the demonstrator software publicly available could amount to a significant impact

Figure 6: Avaruusrekka expo setup after the European Researcher's Night 2017 in Jyväskylä. One lighthouse is installed on a light stand and the other on the wall, visible in the upper left corner of the image. Lighthouses are connected with the sync cable. The VR laptop is on the side table, with HTC Vive link box beside it. A Pico projector is mounted on the wall, mirroring the laptop display. I recommend using a large info screen instead of projectors for both better quality and less interference by projector beaming. A nightside auroral arc is visible in the projector screen.

on the public audience in terms of space science outreach. Free distribution of the built software is therefore recommended[12], for as many platforms as feasible.

4 Diving to Data: VR for Space Research

Having a true, immersive 3-D view of three-dimensional datasets could be seen advantageous in interpreting complex simulation results, such as magnetic topology and the structure of flow fields. Although 3-D visualization software, such as ParaView, VisIt and VTK are very useful in producing 3-D plots, a usual *modus operandi* is to extract 2-D slices of the whole simulation volume, or by tracing magnetic field lines from seed points along some given line. This sort of analysis might overlook significant physical processes, such as non-axial draping patterns at comets[26]. VR systems, as such, do not create new tools to analyze data, but enable a more direct view of, and an immersive interface to the data.

However, the workflow employed in the demonstrator software is somewhat rigid, as datasets have to be pre-analyzed and results exported into the VR software with some steps in between. Directly modifying plots from within the VR software would be essential for fluent analysis of data in a VR environment, and thankfully, there are libraries for interfacing between VR engines and visualization libraries. For example, VtkUnity interfaces Unity to the VTK library, through the use of ActiViz[27] (C♯ bindings for VTK) and Vectrosity (vector graphics library for Unity). Another (more costly and more unexplored) solution would be the use of MiddleVR. For future developments, client-server configurations in a cluster or supercomputing environment should be kept in mind, and both VTK and MiddleVR support cluster deployment.

Interactable plotting in a VR environment opens up a huge amount of intuitive plotting options, besides seeing regular static visualizations in true 3-D (which is not too shabby, either). For example, the user could adjust an isocontour surface by changing the contour value on the fly from within the VR environment and locate some interesting feature. After locating the feature, one could seed streamlines (either some flow lines or magnetic field lines) from a precisely and easily defined point in 3-D to examine the local topology, as one can pinpoint points of 3-D space using the VR controller with ease. Likewise, test particle tracing could be performed "from hand", by tracing particles under the Lorentz force, sampled from some parametrized or simulated distribution and injected and oriented by the user and the hand-held controller[13].

Proper interfacing of data handling libraries would also enable intuitive use of virtual detectors and/or satellites: The user could spawn a satellite with given instruments and drag it along or through potentially interesting features in 3-D space, with the software fetching simulated data points on the fly from the interfaced dataset. A virtual dashboard, as seen in the demonstrator, could be

[12] especially when the *Suomi 100* satellite data products start to arrive and are included in the software

[13] Also, 3-D mouse compatibility with current visualization software should be looked into!

used to display a plot of virtual data along the user-defined trajectory (see e.g. the outreach demonstrator's live spectrogram). Satellite and sensor orientation with respect to the environment would be easy to inspect given the immersive nature of the display. Tools, such as Touko Haunia's Rosetta tool, that interface to SPICE kernels and produce satellite ephemeris could be incorporated to show actual locations and orientations of probes and sensor suites.

Implementing these science tools into the existing demonstrator software would be reasonably straightforward: include the interface libraries to the software and expose select parts of the VTK library for the VR user. User interface construction in Unity is relatively straightforward as well. Additional tools, such as test particle tracers, require a bit more effort, but existing algorithms are readily available.

5 Software tips & tricks

5.1 ParaView to Unity

A suitable workflow for exporting ParaView meshed data to Unity entails exporting the ParaView scene in `.x3d` format, which exports vertex colors and normal data. Importing the data into MeshLab[28], the mesh can be then retouched, e.g. with a decimate operation. Exporting from MeshLab in the COLLADA `.dae` format transfers vertex colors properly to Unity, completing the pipeline.

In ParaView one may construct the plots as usual. With a proper pipeline and file formats, the plot colors will be transferred to Unity automatically - this is true for at least RGB values. Alpha channel export has not been tried, as alpha has been configured in Unity to produce the wanted visuals. Streamline plots have been exported, thus far, through the use of the *tube filter* to produce a triangulated mesh for each streamline, which can be expensive in terms of polygons.

Pipeline:

ParaView Generate meshes from plots. Note that ParaView will export all visible objects in a scene, so make sure to export only the parts you need by hiding the unwanted pipeline phases and plots.

- **Export** `File -> Export scene...`, use the Extensible 3D Graphics format (`.x3d`)

MeshLab Convert and retouch meshes to work properly with Unity. Strictly, only the import and export steps are needed, and decimate and Poisson surface features are included for reference.

- **Import** `File -> Import Mesh...`

Decimate `Filters -> Remeshing, Simplification and Reconstruction -> Quadric Edge Collapse Decimation`

Poisson surf. `Filters -> Point Set -> Surface Reconstruction: Poisson`

Export `File -> Export Mesh As...`, COLLADA .dae format

Unity It is advisable to re-calculate mesh normals during import, otherwise a smooth mesh may appear with a flat shading. Also, it is useful to scale the mesh using the import inspector.

5.2 Unity

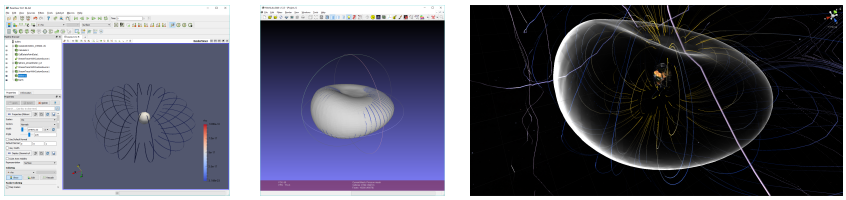
A handy tool with lots of free (and more-or-less reasonably priced) assets and scripts. A collection of notes.

Splatting imagery works through using a camera object at the imaging location, set to a proper attitude. The imaging plane can be construed at e.g. the camera near plane, after which one can raycast the plane vertices from the camera onto a suitable (sphere) collider. The image plane can be textured either with a *render texture* from a virtual camera, or read from a file (see .NET `System.IO`). One could even animate the "splatting" process through a mesh deformation, as with the raytrace surfaces. The special case of over-the-horizon imaging has to be handled separately, though, and this is not yet implemented (only disabled).

Rim shading was implemented as having both the emission and alpha values of the surface modified by factor of the form $(1 - \hat{n} \cdot \hat{v})^{\gamma}$, where $\hat{n}$ is the surface normal, $\hat{v}$ the normalized view direction, and γ a real parameter giving the strength of the rim effect, usually with $\gamma > 1$. The factor is additionally saturated to the range [0, 1] for sanitarity.

A method for creating L-shell surfaces was examined during the development (and displayed in figure 7 on the following page), although it was not used in the demonstrator to constrain the scope of the demo. First, using ParaView, field lines from a constant equatorial radius were extracted using the streamline filter. On top of the streamlines, a ribbon filter was applied to transform the line mesh to a set of narrow, triangulated meshes, with associated normal information. ParaView successfully computed reasonable normal data for the L-shell streamlines, orienting the surfaces of the ribbons consistently and in a sufficiently intuitive manner[14]. The resulting mesh of ribbons can be exported to MeshLab for surface reconstruction. MeshLab can perform a Poisson surface reconstruction, using the point and normal data of the exported mesh, and the results are very satisfactory, as long as the ribbons do not overlap and do not

[14] I haven't found the definition for ParaView-computed normals, but they would look to be perpendicular to the curvature and the actual vector field, which is exactly what would be needed for this to work properly.



(a) Constructing a set of streamlines on a shell in ParaView

(b) Constructing the surface from stream ribbons

(c) The L = 8 shell in Unity, with B field lines and translucent rim shading

Figure 7: Phases in constructing an L-shell surface with ParaView and MeshLab

contain spurious elements. As a disclaimer, the test data had its magnetic field cropped to zero below 3 Earth radii, so the produced L-shell had somewhat awkwardly capped polar regions. Producing toroidal surfaces, or ones clipped only inside one Earth radius to hide the discontinuity, have not been tested yet, and some care might have to be taken when having relatively tight cusp-type geometries.

Normal extrusion refers to using a vertex shader function to modify the vertex positions in the render pipeline, using the GPU (so it is considerably cheaper than using mesh deformations). This is useful with tube-type streamline plots, as the tubes tend to disappear in the distance much too quickly, when the developer wishes to retain streamline visibility. In the vertex shader, the distance of the view point to the vertex is calculated, and the vertex position is translated along the vertex normal in proportion to the view point–vertex distance. This enables the far-away portions of streamlines to expand, and retain visibility. A shader was implemented to provide this functionality, but in some cases (scale transformations by scaling the scene, not the user avatar), mesh deformations to provide the same functionality had to be applied as well. This is a point of future optimization, perhaps through vector graphics libraries.

Attachements

A. Prototype source

The prototype was built using Unity 5.4 personal edition, and the source code suffers from code and asset bloat. Due to the use of the personal edition license terms limiting the use of Unity or derived products to education use, the prototype shall not be publicly presented or distributed. The prototype source code is included in the database (folder `VR bloatware`), and can be opened with a compatible Unity version (5.4.2f2 or more recent).

The source code is arranged as a Unity project, with all assets (3-D models, textures, scripts, etc.) included in the Assets folder, with the remainder of the

folders consisting of Unity project housekeeping. The code (not accounting for included assets possibly written in JavaScript) is written in C♯and ShaderLab.

B. Outreach demonstrator source and build

The outreach demonstrator was developed with the know-how from the prototype version, using a new Unity version (2017.1.1f1), taking advantage of new features. Both the demonstrator source code and a prepared version are attached in the given database. The source code (in the folder `VR-outreach`) can be opened with a compatible Unity version (2017.1.1f1 or more recent), and the pre-compiled build (in the folder `VR outreach Avaruusrekka build 4`) can be executed on Windows PCs (using a HTC Vive headset is required). The Outreach demonstrator was developed using a commercial license of Unity, clearing the source code and built products for public and research use. NB: Developing the Outreach demonstrator with a personal edition is not allowed, per license terms.

The source code is arranged as a Unity project, with all assets (3-D models, textures, scripts, etc.) included in the Assets folder, with the remainder of the folders consisting of Unity project housekeeping. The code (not accounting for included assets possibly written in JavaScript) is written in C♯and ShaderLab. The compiled software in the build folder can be executed by running the provided `VR Outreach.exe` in the root folder.

References

[1] This Is How Valve's Amazing Lighthouse Tracking Technology Works. Accessed 2017-10-23. [Online]. Available: https://gizmodo.com/this-is-how-valve-s-amazing-lighthouse-tracking-technol-1705356768

[2] Microsoft's Windows Mixed Reality: everything you need to know. Accessed 2017-10-23. [Online]. Available: https://www.theverge.com/2017/10/17/16487936/microsoft-windows-mixed-reality-vr-headsets-guide-pricing-features

[3] What HoloLens' field of view really looks like. Accessed 2017-10-23. [Online]. Available: http://newatlas.com/hololens-fov-field-of-view-illustrated/44903/

[4] Unity Technologies, "Unity." [Online]. Available: https://unity3d.com/

[5] Epic Games, "Unreal engine," 2007. [Online]. Available: https://www.unrealengine.com

[6] R. Hess, *The essential Blender: guide to 3D creation with the open source suite Blender*. No Starch Press, 2007, software available at https://www.blender.org/.

[7] U. Ayachit, "The paraview guide: a parallel visualization application," 2015, software available at https://www.paraview.org/.

[8] W. J. Schroeder, B. Lorensen, and K. Martin, *The visualization toolkit: an object-oriented approach to 3D graphics*. Kitware, 2004.

[9] H. Childs, E. Brugger, B. Whitlock, J. Meredith, S. Ahern, D. Pugmire, K. Biagas, M. Miller, C. Harrison, G. H. Weber, H. Krishnan, T. Fogal, A. Sanderson, C. Garth, E. W. Bethel, D. Camp, O. Rübel, M. Durant, J. M. Favre, and P. Navrátil, "VisIt: An End-User Tool For Visualizing and Analyzing Very Large Data," in *High Performance Visualization–Enabling Extreme-Scale Scientific Insight*, Oct 2012, pp. 357–372, software available at https://wci.llnl.gov/simulation/computer-codes/visit/.

[10] G. Ferrand, J. English, and P. Irani, "3D visualization of astronomy data cubes using immersive displays," *ArXiv e-prints*, Jul. 2016.

[11] B. Su. unity-ray-marching. Accessed 2017-10-23. [Online]. Available: https://github.com/brianasu/unity-ray-marching/tree/volumetric-textures

[12] Transparency Sorting. Khronos group. Accessed 2017-10-23. [Online]. Available: https://www.khronos.org/opengl/wiki/Transparency_Sorting

[13] Vectrosity. Starscene Software. Accessed 2017-10-23. [Online]. Available: https://starscenesoftware.com/vectrosity.html

[14] Keyframe Animation. Khronos group. Accessed 2017-10-23. [Online]. Available: https://www.khronos.org/opengl/wiki/Keyframe_Animation

[15] J. E. Bos, W. Bles, and E. L. Groen, "A theory on visually induced motion sickness," *Displays*, vol. 29, no. 2, pp. 47 – 57, 2008, health and Safety Aspects of Visual Displays. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0141938207000935

[16] M. Imura, P. Figueroa, and B. Mohler, Eds., *Visually Induced Motion Sickness Estimation and Prediction in Virtual Reality using Frequency Components Analysis of Postural Sway Signal*, 10 2015. [Online]. Available: https://www.researchgate.net/publication/283856134_Visually_Induced_Motion_Sickness_Estimation_and_Prediction_in_Virtual_Reality_using_Frequency_Components_Analysis_of_Postural_Sway_Signal

[17] R. Stöckli, E. Vermote, N. Saleous, R. Simmon, and D. Herring, "The blue marble next generation-a true color earth dataset including seasonal dynamics from modis," *Published by the NASA Earth Observatory*, 2005, datasets available at https://visibleearth.nasa.gov/view_cat.php?categoryID=1484. [Online]. Available: ftp://169.154.132.40/bluemarble/bmng/bmng.pdf

[18] Esa copernicus open access hub. Copernicus Sentinel data 2017. [Online]. Available: https://sentinels.copernicus.eu/web/sentinel/home

[19] F. Scholten, J. Oberst, K.-D. Matz, T. Roatsch, M. Whlisch, E. J. Speyerer, and M. S. Robinson, "GLD100: The near-global lunar 100 m raster DTM from LROC WAC stereo image data," *Journal of Geophysical Research: Planets*, vol. 117, no. E12, pp. n/a–n/a, 2012, E00H17. Raster available at https://astrogeology.usgs.gov/search/map/Moon/LRO/LROC_WAC/Lunar_LROC_WAC_GLD100_79s79n_118m_v1_1. [Online]. Available: http://dx.doi.org/10.1029/2011JE003926

[20] LRO LROC-WAC Global Mosaic 100m June2013. Accessed 2017-10-23. [Online]. Available: https://astrogeology.usgs.gov/search/map/Moon/LRO/LROC_WAC/Lunar_LRO_LROC-WAC_Mosaic_global_100m_June2013

[21] D. A. Gurnett. Space audio recordings. Accessed 2017-10-23. [Online]. Available: http://www-pw.physics.uiowa.edu/space-audio/

[22] Rosetta 3D model. Accessed 2017-10-23. [Online]. Available: https://nasa3d.arc.nasa.gov/detail/eoss-rosetta

[23] M. Malmer. Accessed 2017-10-23. [Online]. Available: http://mattias.malmer.nu/category/rosetta/

[24] M. Alho. VR prototype recording of the comet 67P. Accessed 2017-10-23. [Online]. Available: https://drive.google.com/open?id=0B97WoSiWSq27WEEtS0FmNVFwdmc

[25] Worldwind C♯. Accessed 2017-10-23. [Online]. Available: https://www.openhub.net/p/wwc

[26] C. Koenders, C. Goetz, I. Richter, U. Motschmann, and K.-H. Glassmeier, "Magnetic field pile-up and draping at intermediately active comets: results from comet 67p/churyumovgerasimenko at 2.0au," *Monthly Notices of the Royal Astronomical Society*, vol. 462, no. Suppl 1, pp. S235–S241, 2016. [Online]. Available: http://dx.doi.org/10.1093/mnras/stw2480

[27] Activiz. Kitware. Accessed 2017-10-23. [Online]. Available: https://www.kitware.eu/product/activiz

[28] P. Cignoni, M. Callieri, M. Corsini, M. Dellepiane, F. Ganovelli, and G. Ranzuglia, "MeshLab: an Open-Source Mesh Processing Tool," in *Eurographics Italian Chapter Conference*, V. Scarano, R. D. Chiara, and U. Erra, Eds. The Eurographics Association, 2008, software available at: http://www.meshlab.net/.