VESTA – VIRTUAL ENVIRONMENT FOR SPACE AND TERRESTRIAL APPLICATIONS – AN OPEN SOURCE 3D GRAPHICS ENGINE

S. Weikert [1], C. Laurel [2]

[1] Astos Solutions GmbH, Meitnerstr. 10, 70563 Stuttgart, Germany, Email: sven.weikert@astos.de
[2] Periapsis Visual Software, 626 Randolph Place, Seattle, WA 98122, USA, Email:claurel@gmail.com

ABSTRACT

This paper presents the Virtual Environment for Space and Terrestrial Applications, VESTA. VESTA is an open-source graphical 3D engine for animation of space scenarios, visualization of mission analysis results and managerial presentations. VESTA has been designed to offer a freely available graphics engine that can be incorporated into any software package, no matter if it is open-source or commercial closed-source software.
The paper presents the capabilities of VESTA, gives an overview on its architectural design and explains the licensing conditions of VESTA.
The paper concludes with a summary of the current development status and the upcoming development activities.

1 INTRODUCTION

Worldwide there are a numerous number of astrodynamic tools for simulation, analysis and optimization of space missions. Most of these tools are in-house developments. Only a few tools are commercially available (like ASTOS). All these tools would gain from a real-time visualisation, either for result presentation purpose or to get a better understanding of the numerical results. A graphical representation of the results could also serve as a plausibility check. Most of these tools lack adequate visualisation means. Especially in-house developments concentrate on the algorithms, graphical result processing is most-often not considered. In this case third-party software like Matlab or STK is often used for visualisation. Since licenses for these commercial tools are quite expensive and pure visualisation cannot max out these tools, this is not the best approach from an economical point of view. However, an own development of a realistic real-time 3D visualisation software will most often exceed the budget of the projects.
As alternative to commercial third-party software, free tools like Celestia or Orbiter are sometimes used for real-time visualisation. These tools are not designed for commercial applications, there license conditions does not allow to incorporate them into a commercial product or it is not easy to link astrodynamic models to them, mainly because these tools come with their own astrodynamic algorithms and the code structure was not designed to replace parts of these models.
So what was missing is a freely available visualisation module that can be linked to own models, that can be sold as part of a commercial product and that is open-source so it can be adapted to special needs.
With VESTA such a module is available now. VESTA comprises a programming interface that is compatible with a variety of languages like C++, Java, Python and many more. With VESTA space scenarios can be presented in a lifelike manner, with an accurate representation of celestial bodies, spacecrafts, propulsion exhausts, shadows, lights and clouds.
In order to display analysis results the realistic picture can be superimposed by a variety of visual aids. Vector arrows can be used to show coordinate frames, velocities, accelerations or forces. Markers for points and areas can show ground stations and observation zones. Antenna beams of any shape can be rendered and will show the current antenna coverage on-ground. Trajectory curves can give an impression of the mission arc.
Uncertainty volumes can be used to show the estimated error of the underlying propagation or analysis. Alternative textures for celestial bodies may be used to add additional information, e.g. a map of population density in conjunction with a re-entry safety analysis.
An important capability of VESTA, and what makes it so suitable for visualizing space environments is that it can display objects at a huge range of scales. A VESTA demonstration application shows a spacecraft just a few meters in size orbiting the Earth, which – with a diameter of over 12,000 km – is roughly one million times the scale of the spacecraft. Camera motion through this environment is completely seamless – there is no abrupt transition from the spacecraft close-up view to the 'whole Earth' view. It's difficult to make this work, because graphics hardware uses single precision (32-bit) floating point arithmetic for transformation calculations, and typically just 24-bit precision for depth buffers. Careful software techniques are required to reduce the large double precision coordinates to single precision (or lower) without producing rendering artefacts. The VESTA library manages the coordinate reduction so that the application doesn't have to. The interface exposed to the application is simple and flexible: spacecraft, planets, and cameras can be positioned arbitrarily using double precision coordinates, and VESTA will render everything correctly. Trajectory plots are an especially troublesome

case. It is reasonable for the camera can be placed within a meter of the plotted trajectory of a Mars-bound spacecraft, even though the trajectory may span over 300 million kilometres. The size of the visible portion of the trajectory will be less than 10^{-11} times the total extent – well beyond the precision of the GPU's single precision arithmetic. Yet VESTA will render even this pathological case correctly.

2 REALISTIC VISUALISATION

A realistic presentation of the mission scenario in real-time was the main requirement for the development of VESTA (see example screenshot in Fig. 1).

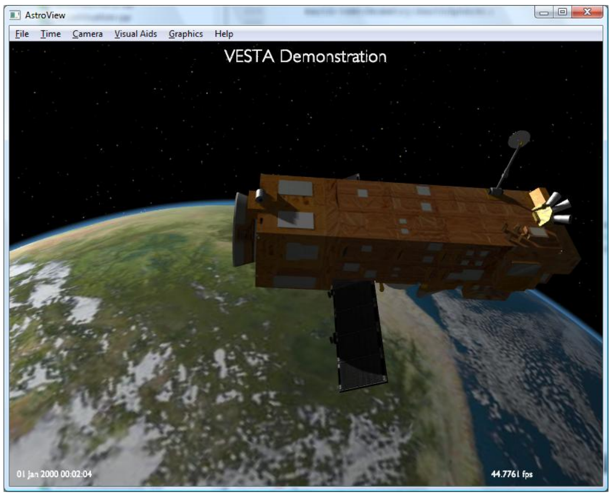

Figure 1. AstroView example application of VESTA

Two different techniques are available to reach this goal:

- Graphics card accelerated scanline rendering
- Ray tracing

The first is commonly used by game developers and produces quite realistic scenarios. With OpenGL [2] and DirectX two hardware-independent interfaces are available to use the 3D rendering capabilities of modern graphics cards.
Ray tracing is a technique for generating the image by tracing the path of light through pixels in an image plane and simulating the effects of its encounters with virtual objects. This technique is capable of producing a superior visual realism, usually higher than that of typical scanline rendering methods, but at a greater computational cost, so it cannot be used in real-time. Since DirectX is only available on Windows platforms, OpenGL has been selected as the graphics API. However, VESTA has an internal abstraction layer on top of the underlying graphics API in order to ease porting to other graphics systems. The abstraction layer is already used to support two different rendering modes in VESTA: fixed-function and shader based. Fixed function mode is required for VESTA to support older GPUs without programmable shading. Certain effects such as shadows, atmospheric scattering, and bump mapping are unavailable on computers configured with older graphics hardware. When using the shader based render mode, all visual effects can be enabled.

2.1 Shadows

The calculation of shadows is not directly supported by the OpenGL language and the rendering engine of the graphics card even though OpenGL supports so-called shadow maps.
The illumination of an object depends only on the angle with respect to the light source and its colour settings.
Shadows due to objects that cover the light source will not be considered unless they are otherwise calculated.
VESTA renders shadows using shadow maps with percentage closer filtering (PCF). To generate the shadow maps for some geometry, VESTA first draws the objects from the point of view of the light source and records the depth at each pixel. In the second pass, the geometry is drawn normally except that at each pixel, a test is performed to see if the corresponding shadow map pixel is closer to the light source. If it is, then the pixel being drawn is in shadow. With percentage closer filtering, multiple shadow tests are performed at slightly offset positions. The result is that some pixels near the edges of shadows will have intermediate lighting levels, i.e. they are located in the penumbral region of the shadow. When there is more than one light source, additional shadow maps may be required; fortunately, the number of light sources casting visible shadows is usually quite low, especially in space environments.

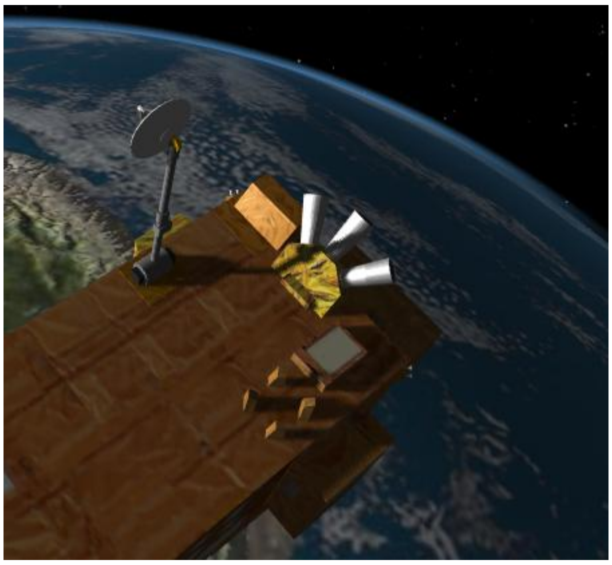

Figure 2. Shadows rendered by VESTA

Only if an object is illuminated by the sun it will become a secondary light source. These additional light sources must be considered to obtain an adequate rendering result.

2.2 Particle System

The particle system of VESTA allows the visualisation of gases and fluids like exhausts (see Fig. 3). Realistic rendering of these entities is a hard task that needs a lot of computation time. This is caused by the fact that the gaseous stream is represented by numerous particles, whose motion needs to be calculated every frame.

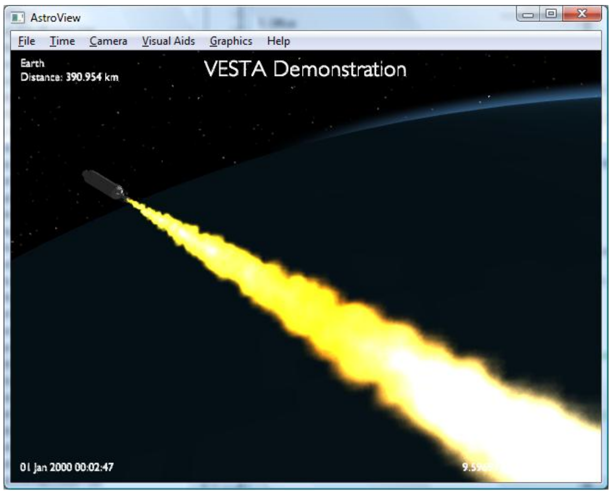

Figure 3. VESTA particle system

VESTA provides different types of particle emitters: point sources, circular sources and box sources.

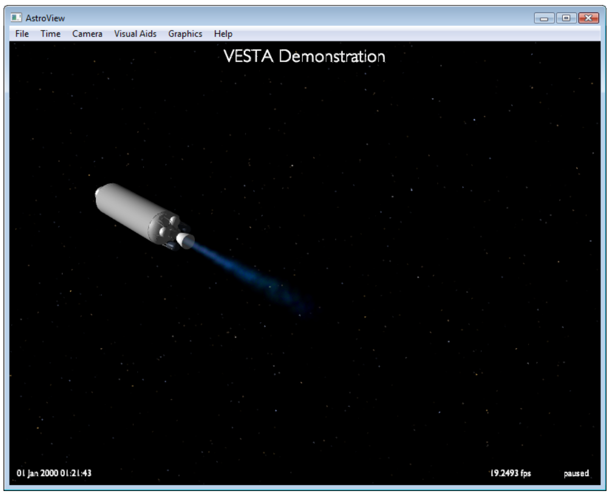

Figure 4. Particle system with different settings

The following parameters allow an adaption of the particle system to different applications like different engine types and different environmental conditions:

- the nominal initial velocity vector
- the variation of the initial velocity vector
- the life-time of the particles
- a constant force applied to all particles
- an array of colours and opacities the particles have from creation till end-of-live
- the initial and final size of each particle

Changing these parameters, the exhaust shown in Fig. 3 looks more like one produced by an electric propulsion system (see Fig. 4). Also other dynamic processes of completely different kind, e.g. water falling down a waterfall can be visualised with this technique.

3 OVERLAID INFORMATION

Besides the realistic view on the mission it is required to overlay the scenario with additional information. VESTA provides a lot of these visual aids:

- Celestial grid (Fig. 5)
- Planes, e.g. for the ecliptic (Fig. 7)
- Coordinate frames (Fig. 5)
- Vectors, e.g. for velocities or forces (Fig. 5)
- Cones, e.g. for antenna lobes or a sensor's field of view (Fig. 6)
- Overlaid textures for planets, e.g. to show the population density (Fig. 7)
- Markers for points of interest
- Uncertainty volumes
- Trajectories (Fig. 5)
- Icons and labels in the screen pane, e.g. to display the current time (all screenshots)

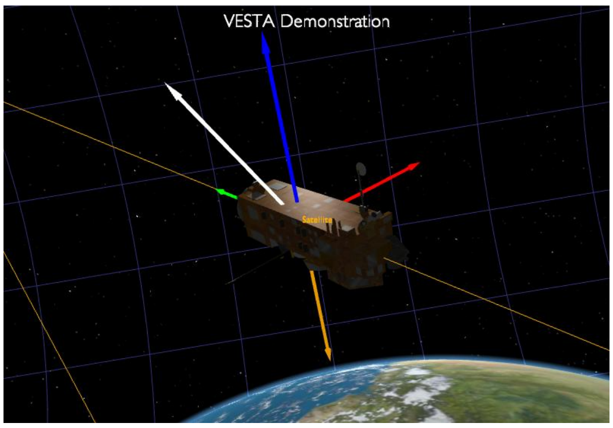

Figure 5. Celestial grid, vectors and trajectories

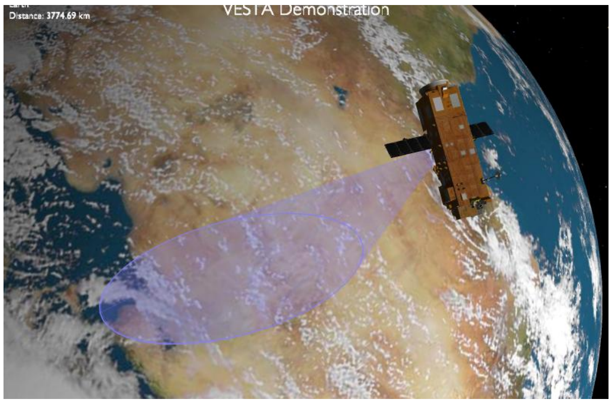

Figure 6. Antenna lobe and coverage

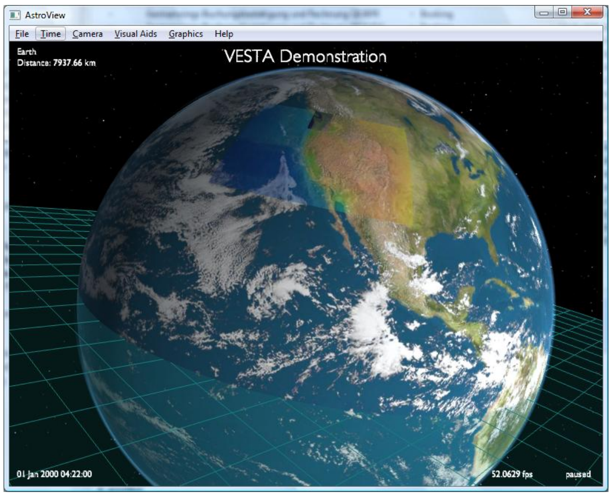

Figure 7. Equatorial plane and overlaid textures

All of these visual aids may be adapted to the user's requirements, either by the existing class methods or by creating a derived *Geometry* class.

4 ARCHITECTURE

VESTA has an object-oriented class hierarchy, programmed in C++. VESTA depends on the some external libraries: Glew is used as interface to OpenGL, the Eigen library is used for vector transformations and the lib3ds library for the import of 3ds model files.
All relevant classes of VESTA are derived from the abstract parent class *Object* (see Fig. 8). To explain all classes goes beyond the aim of this paper, why in the following only the most interesting and relevant classes are discussed.

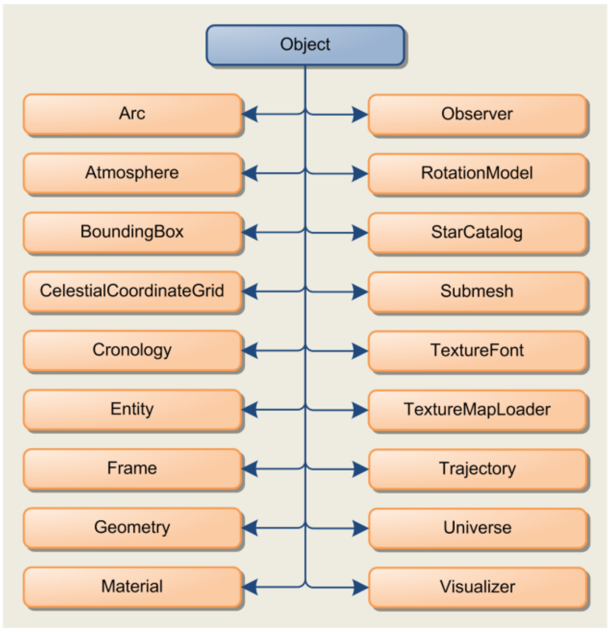

Figure 8. VESTA class hierarchy

Each object in VESTA is represented by an instance of the *Body* class (which is derived from the *Entity* class). Each body has a *Chronology* that specifies the position of the object and a *Geometry* that specifies its visual representation. Body related visual aids may be added by child class instances of *Visualizer*. Celestial bodies may have an optional *Atmosphere*.
An *Arc* is one segment of a *Chronology*. Within an *Arc*, a single *Trajectory* expresses translational motion relative to the centre object and in the trajectory *Frame*. Similarly, rotational motion is described by a *RotationModel* object. The frame for rotational motion is given by the Arc's *bodyFrame*, which can be distinct from the trajectory frame. This class relationship is illustrated in Fig. 9.

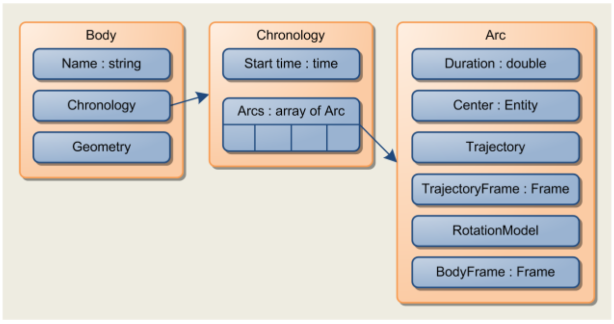

Figure 9. Classes that define an object in VESTA

Three types of trajectories are available (see Fig. 10). *FixedPointTrajectory* describes a point that remains at a fixed position within the reference frame. This could be used for a launch pad within the Earth rotating frame.

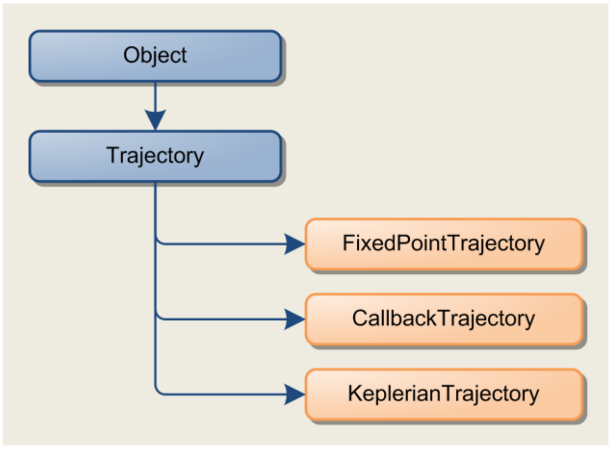

Figure 10. Available trajectory classes

The *KeplerianTrajectory* is self-explaining; the user specifies the Keplerian by providing periapsis radius, eccentricity, inclination, RAAN, argument of periapsis the epoch and the mean anomaly at epoch. Since no mass is associated to the central body, also the mean motion at epoch has to be provided.
The above mentioned classes may be derived to create arbitrary trajectories. The third class *CallbackTrajectory* may be used where the above mentioned classes cannot

be derived due to cross-language inheritance limitations. It calls a user-defined call-back method, providing the current time. Then the call-back method returns the current position in the reference frame of the trajectory.

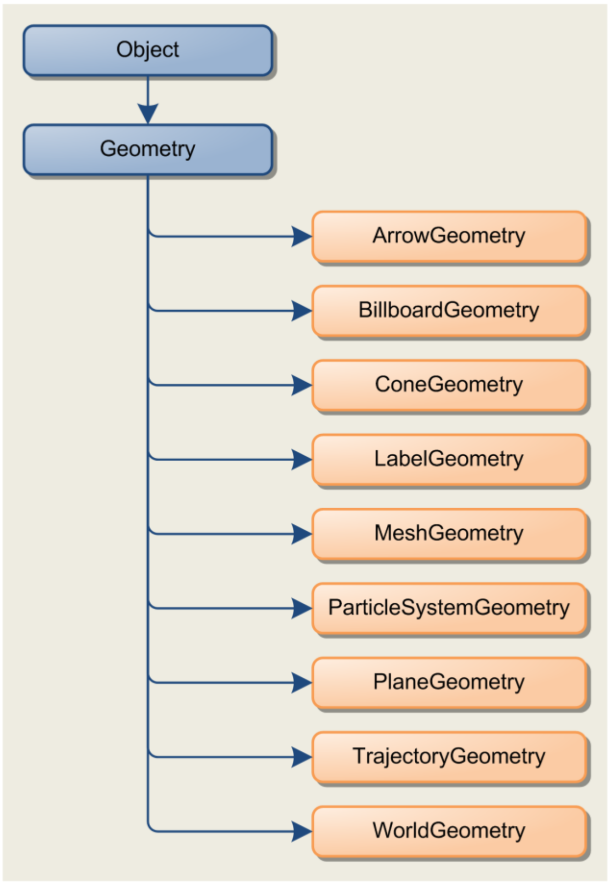

Figure 11. Hierarchy of geometry classes

The *Geometry* object is the visual representation of an *Entity* in VESTA. This base class is abstract; several derived classes define the visual scenario (see Fig. 11):

- *ArrowGeometry* is a *Geometry* object used for visualizers with one or more arrows: body axes, frame axes, direction arrows, etc.
- *BillboardGeometry* is a geometry type used for drawing screen aligned, textured squares, for icons and similar items.
- *ConeGeometry* is a *Geometry* object with a single cone. The cone is intended to be used to show instrument fields of view rather than physical objects. Thus, no surface normals are generated and material properties other than colour and opacity may be set.
- *LabelGeometry* is a geometry type used for single-line screen aligned text and/or icons.
- *MeshGeometry* is a *Geometry* object for triangle meshes, typically loaded from a 3D model file.
- *ParticleSystemGeoemtry* is a *Geometry* object that contains one or more particle emitters.
- *TrajectoryGeometry* is used for visualizing the paths of bodies through space.
- *WorldGeometry* is used for rendering spherical (or ellipsoidal) worlds. Optionally, a *WorldGeometry* object may have a cloud layer, an atmosphere, a ring system, and one or more map layers. Areas, lines and points of interest may be added to the surface, illustrating any kind of data.

The *CelestialCoordinateGrid* and *StarCatalog* classes are responsible to paint the sky with grid and star background. The *Observer* class specifies position, view direction and field of view of the observer.

5 INTEGRATION OF VESTA INTO ASTOS

One of the main goals for the development of VESTA was the improvement of the Aerospace Trajectory Optimization and Simulation Software ASTOS. Up to now ASTOS was able to display its results as reports, diagrams and as two and three dimensional maps. Animation capabilities were not available, but only export filters to tools like Celestia, Orbiter or STK.
With the integration of VESTA, ASTOS will become more independent from these tools. Furthermore it will be possible to show results in a much more flexible way than it was possible via export to third-party software.
One major issue for the engineer using tools like VESTA is he task to create the 3D model of the satellite or launcher. In later design phases the engineer might have access to CATIA models, but probably not to a textured 3D model, e.g. in 3ds format.
Therefore ASTOS will comprise a database of typical launcher and satellite shapes that can be used and adapted for early design phases. For detailed designs and interface to CAD software like CATIA is planned. A graphical editor let the user associate textures to these CAD models. Typical surfaces like of solar panel arrays or beta cloth will be available and can be extended by user-defined surfaces.

6 LICENSE CONDITIONS

The goal behind the VESTA project was to create freely available software that can be incorporated into any commercial product. The project shall benefit from the expertise of numerous potential voluntary developers that might contribute to the project. The code tree shall be made available through a repository.
To avoid different branches of the software, which cannot be maintained anymore, the software shall be available on a single website. To assure the quality of the code it is required that every update is revised by a team of administrators.
Existing license conditions like LGPL [3] and BSD were investigated whether they fit the requirements or not.
It was figured out that these license conditions were designed to protect the open-source development against commercial abuse of the code. Especially LGPL

is virulent: code that links to LGPL licensed code is automatically LGPL licensed. Commercial closed-source code can only be linked with LGPL code if the LGPL code is compiled into a separated library in such a way that the library may be replaced by the user.
This is often not wanted or not possible, e.g. the library need to be modified and contains confidential interface code or enhancements.
In consequence it was decided to create a new license for VESTA that gives full flexibility on the definition of license conditions. These license conditions are detailed in the following as far they are worth to be mentioned here. For the full license text see [1]. However, it has to be stated that only the full license text matters legally.
The core of this license agreement is that Astos Solutions grants to the licensee a royalty free, worldwide right to use, copy, distribute VESTA and to make derivative work of the software. On the other hand the licensee grants the same rights to Astos Solutions for the derivative work created by the licensee. The licensee is only allowed to create derivative work if he follows these rules:

1. He inserts a prominent notice in each changed file stating how and when he changed that file
2. He sends a notice of modification to the email address *licensee@astos.de*
3. He does at least one of the following:
   a. He provides his modified source code to Astos Solutions, so it can be added to the repository.
   b. He uses the modified package only within his corporation or organization.

The distribution of VESTA and modified versions of VESTA is only allowed if the licensee inserts a copyright notice of Astos Solutions during installation of his software. Modified source code of VESTA *must not* be delivered to the user. Irrespective of whether the licensee has modified the VESTA code or not, he has to place a statement about where to get the VESTA source code in the user manual of his software or an equivalent place. This rule is contrary to the rules defined by the GPL or LGPL license, where the licensee has to provide the source code or at least a link to it. With this regulation it shall be prevented that different branches of VESTA are publically available. In consequence only reviewed code modifications that were identified to be useful and reliable will become public.
Comparable to LGPL, paragraph 6 of the Astos Solutions Free Public License grants the licensee the explicit right to include VESTA (as binary) into a commercial product and to charge a fee for it. The software package and its input and output will not fall automatically under the Astos Solutions Free Public License.
Besides the rules mentioned above, the license text allows different individual arrangements between Astos Solutions and the licensee.

7 FUTURE DEVELOPMENT

Currently VESTA has reached the end of its incubation phase. Now a basic version with all required features exist why it had been made available to public in the past days. Anyhow, there is always room for improvements. With future hardware developments also new features of the graphics API will be available that should be utilized. Maybe, ray tracing techniques could be introduced, e.g. for accurate reflection modelling.
The effect of light-scattering in the atmosphere is not yet realistic at dusk and dawn and will be improved.
Currently the particle system does not consider other objects when calculating the motion of particles, i.e. the particles are not deflected by other objects. In particular this becomes visible when rendering an ignited rocket on a launch pad. Future versions of VESTA will consider the launch pad and the planet’s surface when calculating the motion of particles.

8 REFERENCES

1. Astos Solutions GmbH (2010). Astos Solutions Free Public License. Online at *http://www.astos.de/↩ Astos_Solutions_Free_Public_License.html*.

2. Rost, R. J. et al. (2009). *OpenGL Shading Language*, Addison-Wesley Longman, 3rd edition, Amsterdam, The Netherlands.

3. Free Software Foundation, Inc. (2007). GNU Lesser General Public License (LGPL). Online at *http://www.gnu.org/licenses/lgpl.html.*