[**?**]2848

A Planetarium on the Web

Robert Cox - Supervised by Steffen Van Bakel

June 17, 2014

Abstract

The goal of this project is to build a Planetarium application that shows an accurate, realistic, and beautiful representation of the night sky on compatible Android devices. The night sky in the application should be generated from a user input of any date, time and location on the Earth's surface and should have a sufficient amount of accuracy to be able to reproduce the transit of Venus and the eclipses.

Contents

List of Figures

1 Introduction

We have had access to data about the celestial objects arranged in the visible universe for a long time. However it is only relatively recently that the performance of devices has reached a point where is is possible to provide interactive exploration of the universe to average users on both desktop and mobile devices.

Existing solutions that provide an interactive experience for exploring the arrangement of stars and planets suffer from similar issues. None of these solutions can provide an experience that is sufficiently visually appealing and engaging to appeal to the average user whilst also having sufficient accuracy and usability to be a useful tool to current or potential astronomers. An example of this can be seen in Google Sky Map [1] in Figure 1.

Figure 1: Planet scale in Google Sky Map

Despite managing to represent Neptune at the right position in the sky based on the user's location, the apparent size of the planet is not correct. It would not be possible at any time of day to see Neptune as it is displayed in the app. A further example of this lack of accuracy can be shown when the 'Transit of Venus' phenomenon is attempted to be recreated in Figure 2.

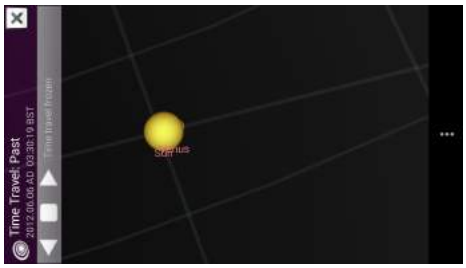


Figure 2: Transit of venus in Google Sky Map

The app manages to display the correct locations of the Sun and Venus during the phenomenon, however Venus is incorrectly displayed behind the Sun and the relative sizes of the celestial objects are also incorrect.

The objective of this project is to implement a fluid, engaging and visually appealing

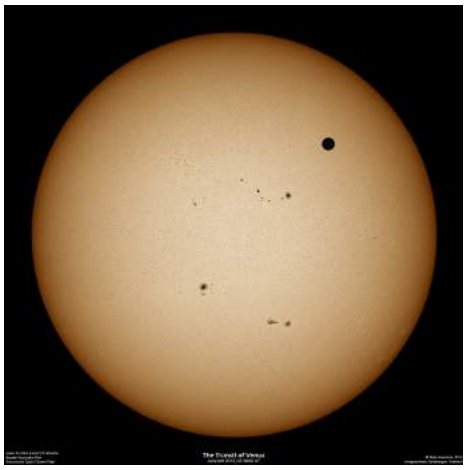


Figure 3: Real Transit of venus

application with a large degree of accuracy. In order to be a useful tool for astronomers, whatever is displayed on the app at a given time should be a reasonably accurate representation of what the user can see in the sky at the time.

I aim to achieve this using graphics programs that can create a proportional and realistic representation of the night sky along with the use of mathematical calculations to accurately display celestial objects of different sizes and relative brightnesses at given dates, times and locations.

1.1 Likely Benefit

An Android application that can present a map of stars and planets in a way that is visually appealing, engaging and accurate would have numerous benefits. By being visually appealing and engaging the app would reach an audience who would otherwise not be interested in Astronomy. By also being accurate and having the ability to display a large amount of information, the app would be particularly useful to users already interested in Astronomy who require a tool that can be used to aid their hobby or profession.

1.2 Challenges

To provide an application that fulfils the requirements previously outlined, the following challenges need to be addressed:

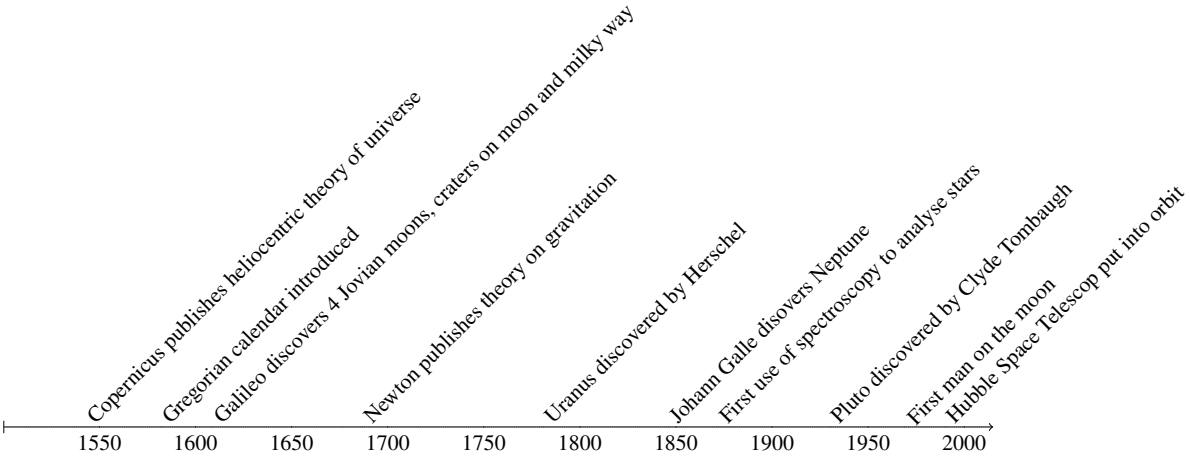
1. **Latency -** The application has to deal with a database of over 100,000 stars and will have to display a subset of these based on the the area of the sky the user is viewing at the time. When a user makes a change to their view such as zooming in or moving the camera, the new viewable stars will need to be displayed within a reasonable timeframe.

2. **Accuracy -** In order to be a viable resource for Astronomers and other professional users, the location and orientation of celestial objects needs to be calculated to a reasonable accuracy so that phenomenons such as the Transit of Venus and Solar Eclipses are displayed correctly and that the arrangement of these objects in the application mirrors the actual arrangement in the night sky.
3. **Aesthetics -** The app needs to be able to visually represent the night sky in all its glory and so an emphasis has to be placed on creating graphics that are visually appealing to the user.

2 Background

2.1 History

Astronomy is the oldest natural science and although our ideas about the Universe changed so drastically with time, night sky's ability to make an observer seem insignificant and awe inspired remains unchanged. From the first observations by early man tens of thousands of years ago, through to modern day imaging and satellites, our thirst for exploring the expanse of the night sky has kept us steadily discovering more and more.



2.1.1 Origins

For the early man with little knowledge about the world, the night sky held a significant presence and was associated with fear and the unknown. Some primitive drawings from this era exist, but the first written records are attributed to the Babylonians in about 1600 B.C. These were astronomical records of the positions of planets and times of eclipses.

About 1000 years later, the Ancient Greeks took the records of the Babylonians and used them to achieve practical goals. For example, in 480 B.C, Thales used the records to predict the occurrence of eclipses and in 220 B.C Eratosthenes, together with the knowledge that the earth was a sphere deduced the approximate circumference of the earth. The Ancient Greeks are also attributed with developing the first star catalog and recording the names of constellations (Hipparchus 100 B.C.).

2.1.2 Geocentric vs Heliocentric

Around 300 B.C, two alternative and conflicting models for the Solar System were introduced. The Geocentric Solar System Model, introduced by Heraclides in 330 B.C, suggested that the earth was at the centre of the solar system and so all the observed planets, along with the sun, orbited the earth. The Heliocentric Solar System Model introduced by Aristarchus in 280 B.C however, suggested that the sun was actually at

the centre of the solar system and that the earth as well as all the other planets orbited it. The two theories are shown in figure 4.

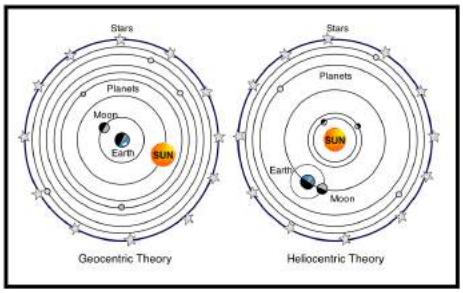


Figure 4: Geocentric and Heliocentric Theories

Both of these models deduced that the orbits associated with planets were perfectly circular and it would be 2000 years before this idea was proven incorrect and the correct model of the universe was proven.

2.1.3 Medieval Astronomy

After a rapid development in the understanding of the universe by the Ancient Greeks, the years following saw little development across Europe as the dark ages bought about a strong religious presence and a period of intellectual stagnation.

In the Arabic and Persian world however rapid developments continued due to a highly cultured society and the translation of the the work done by the Greek astronomers into Arabic. Observational astronomy saw the largest development at this time with Abd al-Rahman al-Sufi carrying out observations to produce the Book of Fixed Stars that described the positions, magnitudes, brightness and colour of stars as well as observing and documenting the Andromeda Galaxy and the Large Magellanic Cloud for the first time.

2.1.4 Renaissance

Following a relatively stagnant 1000 years of astronomical development in Western Europe, in the 1500's Copernicus challenged the status quo be redeveloping the heliocentric theory. Through naked eye observations, he concluded that the Earth actually orbited the Sun, determined the correct order of planets from the Sun and developed a better mathematical description of the Earth's motion. Copernicus's ideas had huge social and political implications as it challenged the common religious ideal of the Earth at the centre of the universe, they caused a paradigm shift to a new wave of astronomy.

The renaissance also bought about a breakthrough in observational astronomy as in 1610 Galileo developed a 20x refractor telescope and observed the four largest moons of Jupiter: Ganymede, IO, Europa and Callisto- the first sighting of satellites orbiting a planet. Most significantly, Galileo observed that Venus exhibited lunar phases and argued that this supported the Heliocentric theory suggested by Copernicus.

2.1.5 The Birth of Modern Astronomy

Up until the 17th century the motions of celestial bodies had been qualitatively explained but no progress had been made in deriving mathematical formulae to represent this. In the early 1600's Johannes Kepler attempted to derive these formulae through the combination of physical insights and naked eye observations and in doing so discovered the three laws of planetary motion. An important aspect in Kepler's laws is he deduced orbits to be ellipses rather than circles as had been the norm for the previous thousand years.

This link between physics and astronomy was further developed when Isaac Newton, after developing his law of universal gravitation, realised that gravity could also be responsible for keeping planets in orbit. In his Philosphiae Naturalis Principia Mathematica, Newton was able to explain all known gravitational phenomena. Despite not initially being accepted, Newton's discoveries has arguably had the single greatest influence on modern astronomy.

2.1.6 New Breakthroughs

The next 300 years saw a large number of astronomical discoveries such as the discovery of more distant objects such as Uranus and Pluto as the quality and accuracy of observational equipment increased. Developments in astronomy often coincided with breakthroughs in science for example the development in the study of spectroscopy (decomposing the light given out by objects in the night sky into spectral lines) was due to the discovery of quantum physics, which could be used to understand these observations.

More recently, breakthroughs in technology have allowed us to send both objects as well as people into space. This has had a significant effect on observational astronomy as it means that telescopes such as the Hubble Space Telescope can take high resolution images of space without the distortion of the earth's atmosphere which allowed us to capture hugely detailed images of the far reaches of space. Such images have provided us with new insights about the nature of the universe such as determining the rate of expansion of the universe.

2.2 Existing solutions

There are several existing solutions currently available for viewing the night sky. Despite none of these managing to meet the specification outlined in my introduction, it was useful to analyse the areas where these solutions excelled and the areas where they failed and to incorporate these into the design of my app.

2.2.1 Google Sky Map

Google Sky Map is the most popular of the existing mobile solutions I have looked at with over 10,000,000 downloads, and this is surprising given that the app is so poorly designed.

The first issue with Google Sky Map is the unrealistic representation of the night sky. In order to engage users a planetarium app should be visually striking in the same way as the real night sky and this app fails to do so on many levels.

The stars in the app are displayed as small grey circles that do not look particularly like stars. I think the reason for this is that the edges of the circles are well defined whereas stars in the night sky have a specular blur around the edges. Therefore my app should represent stars more realistically by using a better texture that blurs the edges of stars.

In the Google app the celestial sphere is divided so that half is a dark blue colour and the other half is lighter and has a yellowish colour. I think this detracts from the experience as I'm confused as to what the purpose of it is as it does not seem to correlate with the Sun's position. I think both the colours used are not good representations of the night sky as they are too bright, I should aim to use a darker colour.

Google Sky Map has an option to either rotate the camera manually by swiping on the screen or to use the built in accelerometer. This works fairly well but is possibly too juddery. It allows the user to rotate and tilt the device to point in a given direction in the sky and this would be very useful in my app.

The manual rotation however is very poorly implemented and can be very confusing to use as there seems to be no definitive direction for up and down. The camera does not rotate around the zenith and nadir nor does it rotate around the celestial poles and so it is very easy to end up with the camera upside down and all the text to appear the wrong way up. In my solution I will need to lock the rotations around fixed references to counter this issue.

Despite its relatively well implemented accelerometer feature the app lacks real world use as it is not possible for a user to select objects on the screen to bring up more information. To have any use to an astronomer, things like the star name, coordinates, distance and magnitude should be available to the user upon selecting a star.

Another issue with the app is the scale. In the app all the planets of the Solar System are easily visible in full colour without any zoom. This of course is incorrect and again makes the app far less useful from an astronomer's perspective, as they want the sky without any zoom to be representative of what they can actually see in the sky. My app should have all planets and stars to scale in order to present the night sky as accurately as possible.

One thing the app does well is its location, date and time support. When the user first loads the app, the location of the user is determined using the phone GPS and then the night sky is calculated based on this information. This process is seamless and does not require any extra effort from the user, which works well and I should aim for my app to work in a similar way. Selecting a date and time is also very well implemented on the app, with an easy to use date and time selector being shown that on confirmation will change the arrangement of the night sky to the selected time.

Overall Google Sky Map is lacking greatly in accuracy and detail and so has a limited use for astronomers. It really only has a use as an entertainment application.

2.2.2 Star Chart

Star Chart [2] is an alternative to Google Sky Map that is also available on the Play Store and has significantly less downloads, around 41,000.

Firstly the aesthetic quality of Star Char is far better than Google Sky Map with the stars looking very realistic and the app having a very polished feel. The issue is however, that it has taken the aesthetics too far and has opted to include a lot of superfluous graphics (Figure 5). For example, in front of constellations there are renderings of the animals or people that the constellation is named after and this is unnecessary for an application to be used by astronomers. Moreover, many galaxies and systems are visible at the standard zoom which, although might be useful when the user zooms in on a portion of the sky, it makes the app less realistic and useful for star gazing.

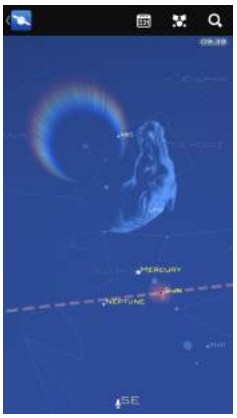


Figure 5: Graphics in Star Chart App

Many of the extra graphics are available to turn off in the menu but I found this difficult to find and it would make more sense to have the features as an opt-in as opposed to opt-out. The ability to adjust what is displayed through the menu is a useful feature, however and I should try and give users of my app this option.

Manual user rotations on the Star Chart app work much better than the Google app as they work about a fixed axis of the zenith and nadir and so it is not possible to flip the app upside down. I also prefer the use of the accelerometer in the app compared with the Google version as it is far less juddery and this is probably due to some sort of moving average on the values produced by the accelerometer.

Similar to the Google app, Star Chart falls short in displaying planets to the correct scale at the default zoom with Pluto, and Uranus easily visible without any zoom. One good point is that planets at the default zoom are presented in a similar fashion to stars as they would actually appear in the night sky with a visual magnitude.

If a planet or star is selected in the app then an info box is presented to the user with information about the star such as apparent magnitude, distance and position as seen in Figure 6, and this would be particularly useful in my app. When using the app I did find that the box often got in the way a little bit so in my solution I may need to think about a better location to present the information, like the corner of the screen.

Figure 6: Displaying information

If the user zooms in on a portion of the sky, more stars pop into view for the part of the sky the user is viewing and these can also be selected to reveal more information. In my implementation this would be an important feature but I feel that the number of stars that pop into view should be significantly increased as the amount in the Star Chart app was underwhelming and at a high zoom the sky becomes bare.

Star Chart improves upon Google Sky Map in a number of respects, most notably in the graphics but has too many unnecessary and detracting features as well as a real lack of accuracy and scale.

2.2.3 Stellarium

Stellarium [24] is a planetarium application available for Windows, Linux and OS X. Although it is not a mobile application it is useful to analyse a solution that does not suffer from the constraints of a mobile device such as screen size, an inconsistent internet connection and low processing power.

Stellarium gives the user a huge amount of options to change what is displayed such as a large number of different grid lines, changing the number of labels of stars, changing the scale and even down to changing the amount of light pollution (Figure 7. For someone that has a keen interest the number of options are very interesting and useful but for a beginner or a casual user the number of options would be overwhelming and confusing.

Unlike the previous two solutions, everything in the application is completely to scale and viewing the night sky at a default zoom provides a pretty accurate representation of the night sky for the time and location of the user as seen in Figure 8).

The colour chosen for the night sky is a very dark blue as opposed to complete black and I think this works particularly well in making the app look like the night sky so I should look to use a similar colour.

The number of stars being rendered in Stellarium is also significantly greater than the other two solutions. This could be due to the greater amount of processing power available to a desktop computer but I feel it greatly adds to the user experience, and so

Figure 7: Options in Stellarium

if I can work around the limitations of a mobile device this would greatly enhance my app.

Stellarium however, does suffer from a few usability issues. Zooming does not work particularly well with a mouse or trackpad. I often had to use the keyboard shortcuts instead which are far less intuitive. Interaction with the sky generally feels less fluid than with the other solutions and this is a definite limitation in a desktop application. Moreover, being a desktop application it can only be used whenever a desktop is available, which is far less often than a mobile device and would be problematic if it was needed to be used in a rural environment with lower light pollution.

It seems that there is a trade off between mobile applications and desktop applications. Mobile applications are far more intuitive to use but lack in accuracy and aesthetics whereas desktop applications produce accurate representations of the sky but have a poor user experience and can not be used on the move. I should aim to find a middle ground that can achieve both benefits.

2.2.4 Sky Survey

Sky survey [25] is a web based application to view all the visible stars at once. A sky made up from a huge number of different layered images can be rotated and zoomed in. From an astronomers perspective it has little use as a utility for stargazing, however it provides a spectacular way of viewing everything in the night sky.

It would be extremely difficult to recreate an app with a similar amount of detail on a mobile device due to the sheer amount of processing power in downloading and rendering all the textures of the sky together. However it does act as inspiration for how I could go about rendering graphics in my application.

Figure 8: Correct scale in Stellarium

Figure 9: Sky Surveyi

2.3 API's and Technicals

2.3.1 Android

I have developed my app for the Android [11] mobile operating system and have chosen to do this for two reasons. Firstly, Android has the largest market share of mobile devices and so I would like my app to be available to as many users as possible. Secondly, it is possible to distribute the app .apk file to specific users without going through a distribution service like the Play Store - therefore it is possible to receive feedback quickly and directly from users.

2.3.2 OpenGL ES2

OpenGL [12] is a cross language, multi-platform API used for rendering 2D and 3D computer graphics. OpenGL ES is a subset of the OpenGL designed for use on

embedded systems. I have used OpenGL ES2 as it is the most widely used version of the embedded OpenGL API. A newer version - ES 3 does exist with added features however this is only available to a small number of Android devices and so would reduce the number of devices that could use my application.

2.3.3 SQLite

SQLite [13] is a relational database management system that is the most popular for embedded systems and is the default database system used in Android. All the code concerning my database of stars has been written using an SQLite database.

3 Theory

3.1 Planets

The solar system contains 8 planets, all of which orbit the sun.

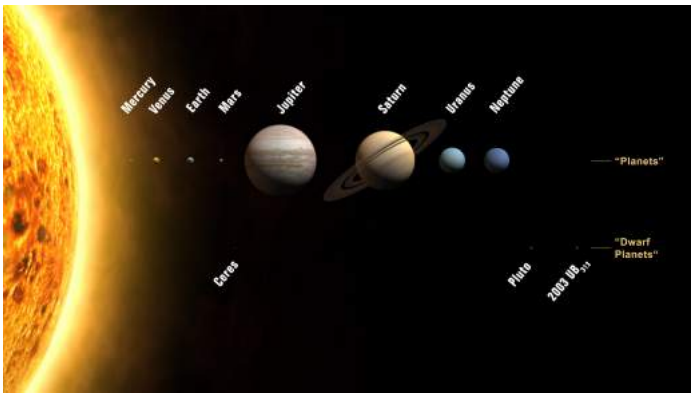


Figure 10: Planets of the Solar System

Mercury

Mercury is the smallest planet in the solar system and is the closest to the sun. It was first observed in the 17th century by Galileo and in 1639 it was discovered that it exhibited orbital phases. At its brightest, Mercury can be observed with the naked eye.

Venus

Venus is the second brightest object in the sky after the moon and can frequently be viewed by the naked eye, with its maximum brightness occurring at shortly before sunrise or shortly after sunset.

Mars

Mars is the second smallest planet in the solar system and is characterised by its distinctive coloration when viewed by the naked eye. Despite being close to earth, it can be difficult to observe due to its small size.

Jupiter

Jupiter is the largest planet in the solar system and similar to Venus, it can be frequently viewed by the naked eye at twilight and is always brighter than the brightest star - Sirius. Jupiter also has a number of moons, the four largest of which - IO, Europa, Ganymede and Callisto - can be viewed from a telescope.

Saturn

Saturn is the second largest planet in the solar system and is a gas giant known for its distinctive ring system. Similar to Jupiter it also has a number of moons, the largest of which, Titan, is 50% larger than the Earth's moon. Saturn can be observed with the naked eye but in order to view its rings, a telescope is needed.

Uranus

Uranus is the largest of the two ice giants (the other being Neptune) and is characterised by its featureless pale blue face. It is the furthest planet from Earth that is still visible to the unaided eye, however this is only possible in the a very dark sky due to its low apparent magnitude.

Neptune

Neptune is the furthest planet from the the sun and is too faint to be seen with the naked eye. When viewed through binoculars it appears as a dim star.

3.2 Moons

A moon is a natural satellite that orbits another body (its primary) and our Solar System contains 173 know natural satellites. The inner planets - Mercury and Venus - have no natural satellites and the Earth has one. Further out from Earth, Mars has two small moons - Phobos and Deimos and the gas giants further out having a large number of satellites, as subset of which are comparable to the Earth's moon in size.

3.3 Phases

As the Earth orbits the sun, the position of the Moon with respect to both the Sun and the Earth changes the amount of light that shines on its visible face and these are called phases. For example, when the Sun is placed directly behind the Moon with respect to the earth, no light shines on the visible face of the Moon so it appears black and this is called the new Moon. Conversely, if the Moon is positioned such that its visible face receives the maximum amount of light then the moon is said to be Full. In between these phases there are a number of stages when the face is partially lit and these are shown in Figure 11.

Other planets orbit the sun at a different rate and shape to the Earth and so for this reason it is possible for phases occur in these as well. Although the phase of all planets do change, only the planets in between the Sun and the Earth - Mercury and Venus- change significantly enough from a full phase to be easily observed. Similarly to the Moon, when Mercury and Venus lie directly in between the Earth and Sun (at their transit) they will appear to be completely black as all the light from the sun will be directed to their back faces.

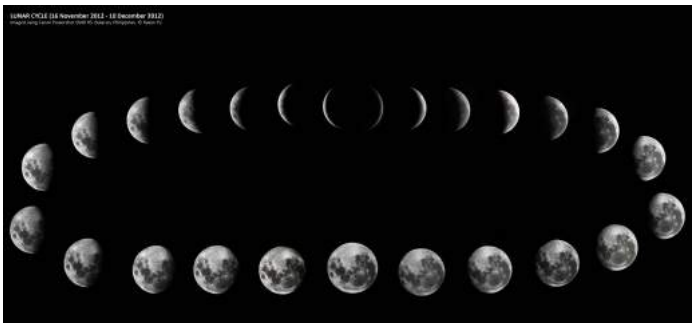


Figure 11: Phases of the moon [31]

3.4 Stars

Stars are massive, dense spheres that are far outside the solar system that provide energy to satellites orbiting them. The closest star to Earth (besides the Sun), is Proxima Centauri at a distance of 4.243 light years. Despite being a considerable distance away, stars are still viewable because of their massive size and the considerable amount of energy they emit. To an observer stars are stationary and only travel across the sky due to the rotation of the Earth.

3.5 Magnitude

The brightness of an object in the night sky is dependant on a number of factors including its size, distance from a light source, angle or the amount of light it emits. In order to quantify this brightness, the logarithmic scale of magnitudes were introduced.

3.5.1 Absolute Magnitude

The absolute magnitude of an object is a measure of its intrinsic brightness, that is, its hypothetical magnitude if viewed from a standard distance (10 parsecs or 32.6 light years). It works on a logarithmic scale where brighter objects have smaller absolute magnitudes. For example, the star Rigel has an absolute magnitude of -6.69.

3.5.2 Apparent Magnitude

A more useful way of interpreting the brightness of a star is to use apparent magnitude. This is a measure of the brightness of a star when viewed by an observer on earth. Again, this works on a reverse logarithmic scale where a smaller apparent magnitude value corresponds with a brighter star. Apparent magnitudes are based relative to the star Vega which has an apparent magnitude of approximately 0. The brightest object in the night sky - the Sun - has an apparent magnitude of -26 and the Moon has a mean apparent magnitude of -12.

3.6 Geographical coordinates

Many calculations of the positions of celestial objects in the night sky depend on the location of the observer and this is expressed through the geographic coordinate system- longitude and latitude.

The **latitude** of a point on the Earth's surface is the angle between the equatorial plane and the straight line that passes through that point. The north pole is at 90 degrees north and the south pole is at 90 degrees south with the equator at 0 degrees.

The **longitude** of a point on the Earth's surface is the angle east or west of a reference meridian to another meridian. For my app I will use the Greenwich Meridian.

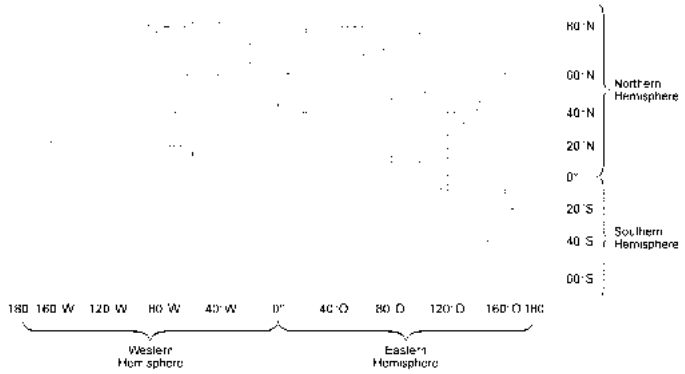


Figure 12: Longitude and Latitudes of Earth [3]

3.7 Celestial Sphere

In order to understand how the night sky is arranged the idea of a celestial sphere is used. It is an imaginary sphere that contains the Earth, with the Earth being treated as a point in the centre of the sphere.

3.7.1 Zenith, Nadir and Horizon

For an observer standing on the Earth's surface, if they were to draw a line that goes directly above them that passes through the celestial sphere, the intersection point would be the zenith. Similarly, the point below would be the nadir. The division between the earth and the sky is the horizon and so the horizon forms a circle with a circumference that spans the celestial sphere. To an observer, only the celestial objects that lie above the horizon are visible.

The zenith, nadir and horizon positions are all relative to the observers location and this is an important point that distinguishes between the equatorial coordinate system and the horizontal coordinate system, both of which are covered later.

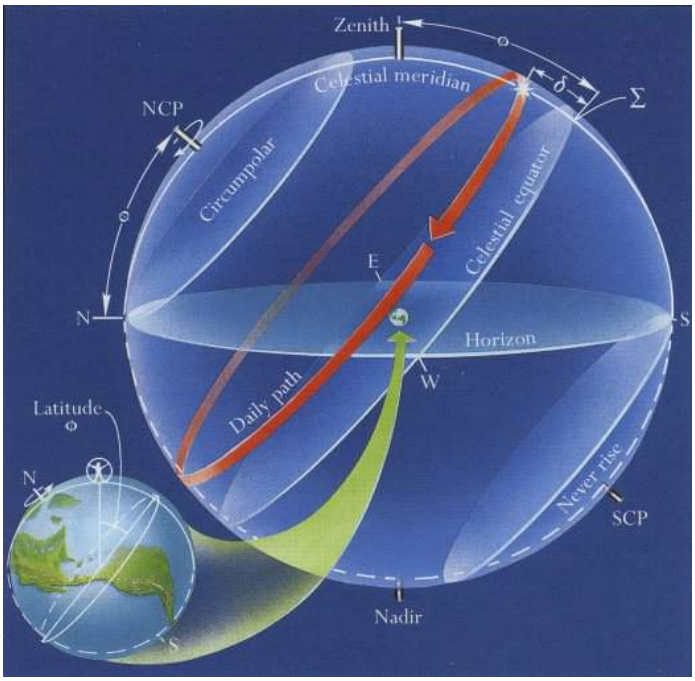


Figure 13: Celestial Sphere [7]

3.7.2 Celestial Poles and Celestial Equator

The celestial sphere is tipped on its axis in a similar way to the Earth. If the axis that the Earth rotates around (the line which joins the north and south poles) is extended upwards, the point where this intersects the celestial sphere is the north celestial pole and similarly the intersection when the line is extended downwards is the south celestial pole. Polaris (the North Star) is located very close to the north celestial pole and appears to be stationary, therefore it is particularly useful as a reference point, especially in navigation.

If the circle defined by the equator is expanded outwards to intersect the celestial sphere, this is defined as the celestial equator. The celestial equator is particularly important in astronomy as all the planets of the solar system will tend to lie very close above or below this line.

Unlike the zenith, nadir and horizon, the celestial poles and celestial equator are not relative to the observer but are fixed.

3.7.3 Celestial Meridian

The circle passes through the north celestial pole, south celestial pole, zenith and nadir is called the celestial meridian. The intersection of this meridian with the horizon defines north and south and the intersection of the celestial equator with the horizon defines east and west.

If you were to imagine the sky with the stars and planets located on the inside surface of the celestial sphere then as the Earth rotates on its axis from west to east the sky would seem to rotate in the other direction. This is what gives the impression of objects moving across the sky.

3.7.4 Equatorial Coordinate System

The positions of celestial objects in the night sky are represented through equatorial coordinates which are represented by two angles, the right ascension and the declination.

The right ascension is an angle measure east along the equator from the vernal equinox (the intersection of the equator and the celestial meridian). It can be represented as an angle or more commonly as a unit of time in hours, minutes and seconds with 24 hours being a whole circle. This way of measuring means the position of the celestial object can be timed along the sky.

The declination is an angle measured from the equator and is the vertical component of an object in the night sky. The units of degrees, minutes and seconds are used to represent the angle.

The equatorial coordinate system is independent of the observer's location.

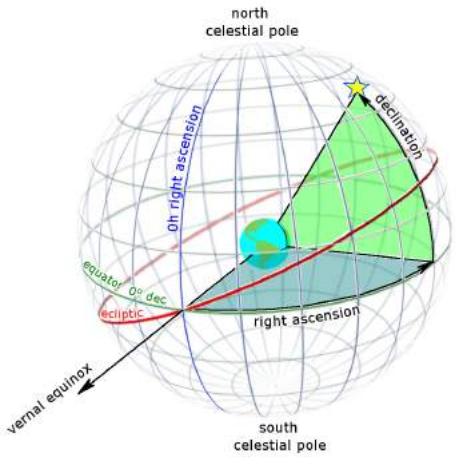


Figure 14: Right Ascension and Declination [4]

3.7.5 Horizontal Coordinate System

Horizontal coordinates are the positions of the celestial objects in the sky with the observers location taken into account. The horizon of the user's current location is used as the fundamental plane (which for equatorial coordinates was the equator). Horizontal coordinates are made up from an azimuth and an altitude.

The azimuth of an object in the sky is the horizontal angle on the fundamental plane between the object and a reference point (normally north) and is measure in degrees.

The altitude of an object (also known as the elevation) is the angle between the object and the observers local horizon and is between 0 degrees and 90 degrees for visible objects.

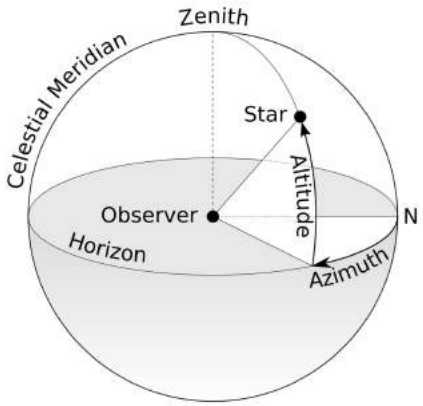


Figure 15: Azimuth and Altitude [5]

3.8 Time

In order to observe, calculate and predict the locations of celestial objects, a time reference is needed

3.8.1 Universal Time

The earth is constantly rotating on its axis which gives rise to the concept of day and night and because of this, some portions of the earth will be in night time while others will be in day. To get around this issue, the idea of time zones was introduced so that roughly speaking daytime would occur at the same local time for any location on the planet.

Time zones are represented as an offset from GMT (Greenwich Mean Time), that is, the the mean solar time on the prime meridian in Greenwich, UK. For astronomers throughout the world, rather than performing calculations to the local time, it is often desirable to use a standard universal time and for this reason UTC (an extension of GMT) is used.

In my application the vast majority of calculations are performed using universal time as opposed to local time.

3.8.2 Epoch

An epoch is an instant in time that is used as a reference point from which time may be measured to remove ambiguity when performing time based calculations. Events taking place before a given epoch can be described as negative values. Two often used epochs are 00:00:00 UTC on the 1st January 1970 - used by Unix Systems and

programming languages and J2000 (approximately noon UTC January 1, 2000). J2000 is the current standard epoch but in my app I use both the 1970 and the J2000 epoch depending on the formula.

3.8.3 Day Number

In working out the positions of planets or moons in their orbit an epoch of J2000 is often used, where the current position of a satellite can be determined by a start position at the epoch and a given amount of movement per day from that initial position. Therefore the number of days since the epoch need to be calculated and can be done using the following formula:

$$\begin{aligned} D = 367 \times year - 7 \times (year + (month + 9) \div 12) \div 4 + \\ 275 \times month \div 9 + day - 730531.5 + h \div 24 \end{aligned} \tag{1}$$

where h is defined as:

$$h = hour + \frac{minute}{60} + \frac{second}{3600} \tag{2}$$

3.8.4 Sidereal Time

Sidereal time is a system used by astronomers to work out the direction to point their telescopes and is based on the Earth's rotation to fixed stars with a mean sidereal day being about 23.9344699 hours. Sidereal time therefore divides on full spin of the Earth into 24 sidereal hours and these correspond with the 24 hours of right ascension.

The **Local Sidereal Time** indicates the right ascension on the sky that is currently crossing the local meridian. Therefore if an object in the sky has a right ascension of 4h 20m 10s it will be on the observers local meridian at local sidereal time = 04:20:10. If an object has a right ascension of 5h 20m 10s and the local local sidereal time = 04:20:10 then the object is an hour away from passing the local meridian and this corresponds to an angle of 15° to the west.

The angular distance between the local meridian and the object at a given time is the object's **Hour Angle**.

3.9 Orbital Motion

3.9.1 Eliptic

The ecliptic is defined as the apparent path taken by the Sun on the celestial sphere. It can be difficult to observe that the Sun is following such a well defined path due to the constant rotation of the Earth. An important result is that the ecliptic is at an angle of 23.4° and this is called the obliquity of the ecliptic.

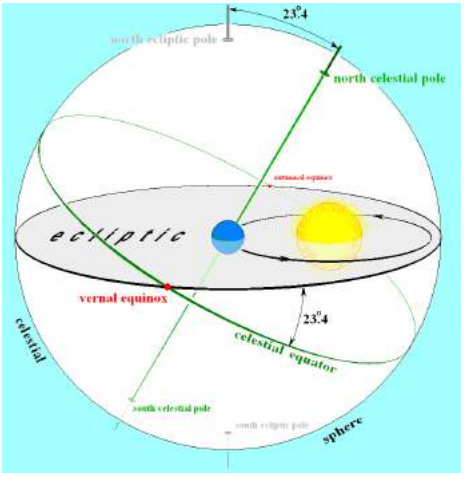


Figure 16: Ecliptic [22]

3.9.2 Eccentricity e

As Keplar discovered, orbits are not perfect circles but are ellipses. The eccentricity e of an orbit is a non negative number that defines its shape - it is a measure of how much the orbit deviates from a perfect circle.

- Circular orbit: $e = 0$
- Elliptic orbit: $0 < e < 1$
- Parabolic trajectory: $e = 1$
- Hyperbolic trajectory: $e > 1$

The orbits of planets generally have an eccentricity close to 0, for example the Earth's orbit has an eccentricity of 0.0167 and the orbit with the greatest eccentricity is Mercury with a value of 0.2506. Asteroids in the solar system have orbital eccentrics between 0 and 0.35 and the eccentricity of comets are close to 1.

3.9.3 Perigee and Apogee

The apogee refers to the furthest distance and the perigee refers to the closest distance that an orbiting satellite can be from the body it is orbiting. Depending on the context these two terms can also refer to the points of furthest and least distance respectively.

3.9.4 Semi-major axis

For an elliptic orbit the major axis is the longest diameter of the ellipse and the semi-major axis is one half of the major axis. In the special case of a perfect circular orbit the semi-major axis is the radius.

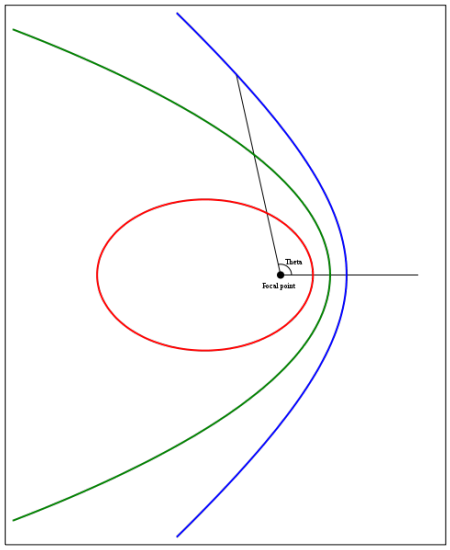


Figure 17: Eccentricity of orbits [20]

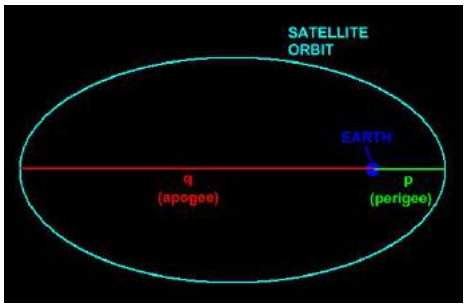


Figure 18: Perigee and Apogee

3.9.5 Inclination

The inclination along with the longitude of ascending node is used to define the orientation of an orbit. The inclination is the angle between a reference plane such as the ecliptics and the orbital plane. For example the inclination to the ecliptic for Mercury is $7.01°$ and by definition, the inclination to the ecliptic for the earth is $0°$.

3.9.6 Ascending node

Any given orbit (that is not contained in a reference plane) will cross a reference plane in two places. For geocentric and heliocentric orbits (the only orbits used in my calculations) the ascending node is the one of the two crosses with the reference plane where the object is moving north in its orbit where for the descending node the object is moving south.

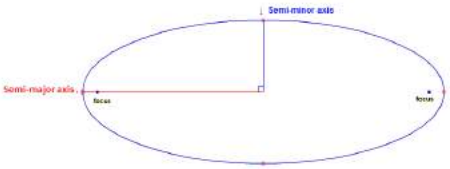


Figure 19: Semi-major axis [21]

3.9.7 Longitude of ascending node O

With the inclination being the first value to define the orientation of an orbit the second is the longitude of the ascending node. It is the angle from a reference direction (origin of longitude) to the ascending node around the plane of reference. For geocentric orbits the reference plane is the equatorial plane and the vernal equinox is the reference direction and for heliocentric orbits the reference plane is the ecliptic and again the vernal equinox is the reference direction.

3.9.8 Orbit Anomalies

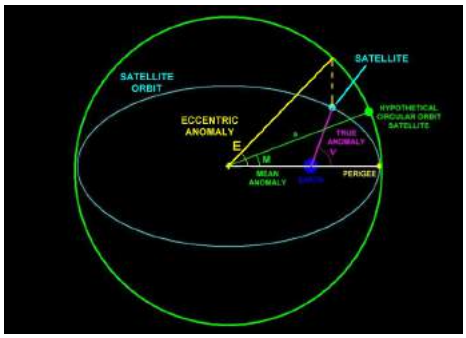


Figure 20: Orbit Anomalies [29]

The **mean anomaly** is one of three angles that describe the position of a satellite in its orbit. It is the angle measure from the perigee if the orbit were completely circular.

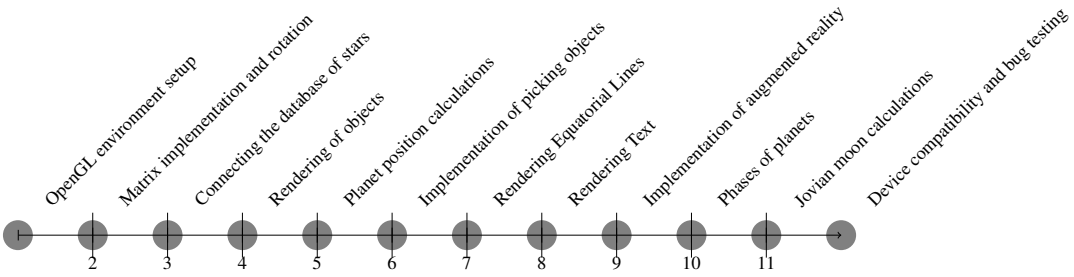
The **eccentric anomaly** is the second angle that describes the position of a satellite in its orbit and like the mean anomaly it is measured from the perigee. If a line is drawn perpendicular to the major axis of the orbit that passes through the true position of the satellite and intersects with the circular orbit then the eccentric anomaly is the angle from the perigee to this intersection.

The **true anomaly** is the third of the three angles and is true angle between the line from the body being orbited to the perigee and the line from the body to the satellite.

4 Implementation

4.1 General Order of Implementation

The general order in which I went about implementing my app is as follows. As you will see later in this section, I frequently made alterations and adjustments to different features but generally speaking this was the order in which I worked.



4.2 Algorithms

4.2.1 Computation of planets

The algorithm used in computing the positions of the planets is outlined by Stephen R. Schmitt on his website [10].

In order to render the planets, I first needed to work out their location in equatorial coordinates - right ascension and declination and then convert these to horizontal coordinates - azimuth and altitude.

In order to calculate a planet's position in its orbit I needed to work out the amount of time that had passed since a given reference point and for this I used J2000 and the calculation of the day number that I discussed section 3.8.3. Using the day number, I could then work out the number of centuries cy that had passed:

$$cy = \frac{D}{36525.0} \tag{3}$$

The position of a planet in its orbit can then be determined by working out how its orbital elements have changed with respect to the J2000. The values to do this are different for each planet, for example the orbital elements of Venus are worked out by:

$$a = 0.72333199 + 0.00000092cy \tag{4}$$
$$e = 0.00677323 - 0.00004938cy \tag{5}$$
$$i = 3.39471 - \frac{2.86cy}{3600} \tag{6}$$
$$O = 76.68069 - \frac{996.89cy}{3600} \tag{7}$$
$$w = 131.53298 - \frac{108.8cy}{3600} \tag{8}$$
$$L = 181.97973 + \frac{210664136.06cy}{3600} \tag{9}$$

After doing this I could then compute the mean anomaly of the planet:

$$M = L - w \tag{10}$$

Using the mean anomaly M along with the eccentricity e computed with the orbital elements, the eccentric anomaly E of the planet can be computed by:

$$E = M + e\sin(M)(1.0 + e\cos(M)) \tag{11}$$

Then iterate:

$$E' = E \tag{12}$$
$$E = \frac{E' - e\sin(E') - M}{1 - e\cos(E")} \tag{13}$$

Until E-E' is close enough to 0.

Then the true anomaly is:

$$V = 2\arctan(\sqrt{(}\frac{(1+e)\tan(0.5E)}{1-e}) \tag{14}$$

The heliocentric radius R can then be computed using the true anomaly V and the orbital elements a and e:

$$R = \frac{a(1-e^2}{1 + e\cos(V)} \tag{15}$$

Using the heliocentric radius the heliocentric coordinates of the planet can be computed:

$$X_H = R[\cos(O)\cos(V + w - O) - \sin(O)\sin(V + w - O)\cos(i)] \tag{16}$$
$$Y_H = R[\sin(O)\cos(V + w - O) - \cos(O)\sin(V + w - O)\cos(i)] \tag{17}$$
$$Z_H = R[\sin(V + w - O)\sin(i)] \tag{18}$$

If the calculation is being performed for the Sun then naturally the heliocentric coordinates would be (0, 0, 0) as the heliocentric coordinates are based on the position compared with the Sun.

Next the heliocentric coordinates of the Earth need to be calculated and so the previous process can be carried out again but using the orbital elements of the Earth instead.

Once the heliocentric coordinates of both the planet and Earth have been computed the geocentric coordinates of the planet can be computed by:

$$X_G = X_H - X_E \tag{19}$$
$$Y_G = Y_H - Y_E \tag{20}$$
$$Z_G = Z_H - Y_E \tag{21}$$

These geocentric coordinates are computed for the geocentric ecliptic as opposed to the geocentric equatorial and so need to be adjusted by rotating around the obliquity of the ecliptic for J2000:

$$ecl = 23.439281 \tag{22}$$
$$X_{EQ} = X_G \tag{23}$$
$$Y_{EQ} = Y_G \cos(ecl) - Z_G \sin(ecl) \tag{24}$$
$$Z_{EQ} = Y_G \sin(ecl) + Z_G \cos(ecl) \tag{25}$$

Finally the right ascension RA, declination DEC and the distance can be computed. The right ascension is computed by:

$$RA = \begin{cases} \arctan(Y_{EQ} \div X_{EQ}) + 180, & \text{if } X_{EQ} < 0 \\ \arctan(Y_{EQ} \div X_{EQ}) + 360, & \text{if } X_{EQ} < 0 \wedge Y_{EQ} < 0 \\ \arctan(Y_{EQ} \div X_{EQ}), & \text{otherwise} \end{cases} \tag{26}$$

The declination is computed by:

$$DEC = \arctan \frac{Z_{EQ}}{\sqrt{X_{EQ}^2 + Y_{EQ}^2}} \tag{27}$$

The distance (in AU) is computed by:

$$Distance = \sqrt{X_{EQ}^2 + Y_{EQ}^2 + Z_{EQ}^2} \tag{28}$$

4.2.2 Computation of moon

Due to the proximity of the earth to the Moon a different calculation must be carried out to the planets that has a higher degree of accuracy. The algorithm I used is outlined by Keith Burnett on his website [32].

Similarly to the planets, the day number d is computed and from that the number of centuries cy are determined.

Using this the geocentric longitude l of the moon can be calculated by:

$$l = 218.32 + 481267.883cy \tag{29}$$
$$l = l + 6.29\sin(134.9 + 477198.85cy) \tag{30}$$
$$l = l - 1.27\sin(259.2 - 413335.38cy) \tag{31}$$
$$l = l + 0.66\sin(235.7 + 890534.23cy) \tag{32}$$
$$l = l + 0.21\sin(269.9 + 954397.7cy) \tag{33}$$
$$l = l - 0.19\sin(357.5 + 35999.05cy) \tag{34}$$
$$l = l - 0.11\sin(186.6 + 966404.05cy) \tag{35}$$

the geocentric latitude bm of the moon can be calculated by:

$$bm = 5.13\sin(93.3 + 483202.03cy) \tag{36}$$
$$bm = bm + 0.28\sin(228.2 + 960400.87cy) \tag{37}$$
$$bm = bm - 0.28\sin(318.3 + 6003.18cy) \tag{38}$$
$$bm = bm - 0.17\sin(217.6 + 407332.2cy) \tag{39}$$

and the paralax gp can be computed by:

$$gp = 0.9508 \tag{40}$$
$$gp = gp + 0.0518\sin(134.9 + 477198.86cy) \tag{41}$$
$$gp = gp + 0.0095\sin(259.2 - 413335.38cy) \tag{42}$$
$$gp = gp + 0.0078\sin(235.7 + 890534cy) \tag{43}$$
$$gp = gp + 0.0028\sin(269.9 + 954397.7cy) \tag{44}$$

The parallax can be used to find the radius vector rm

$$rm = \frac{1}{\sin(gp)} \tag{45}$$

Using the geocentric latitude and longitude with the radius vector the geocentric coordinates can be computed:

$$xg = rm\cos(l)\cos(bm) \tag{46}$$
$$yg = rm\sin(l)\cos(bm) \tag{47}$$
$$zg = rm\sin(bm) \tag{48}$$

Similarly to the planets, the equatorial coordinates can be computed by:

$$xe = xg \tag{49}$$
$$ye = yg\cos(ecl) - zg\sin(ecl) \tag{50}$$
$$ze = yg\sin(ecl) + zg\sin(ecl) \tag{51}$$
$$\tag{52}$$

Where ecl is defined as:

$$ecl = 23.4393 - 3.563 \times 10^{-7}d \tag{53}$$

Unlike the planets, the right ascension and declination of the Moon depends upon the location of the observer and these are topocentric coordinates computed from the local sidereal time lst, the equatorial coordinates and the latitude $glat$ of the observer:

$$xtop = xe - \cos(glat)\cos(lst) \tag{54}$$
$$ytop = ye - \cos(glat)\sin(lst) \tag{55}$$
$$ztop = ze - \sin(glat) \tag{56}$$

Finally the right ascension RA, declination DEC and distance R (in lunar diameters) can be computed by:

$$RA = atan2(ytop, xtop) \tag{57}$$
$$DEC = atan(\frac{ztop}{\sqrt{xtop^2 + ytop^2 + ztop^2}}) \tag{58}$$
$$R = \sqrt{xtop^2 + ytop^2 + ztop^2} \tag{59}$$

4.2.3 Computation of phase

For computation of both the phases of the planets and the apparent magnitudes I used the algorithms outlined by Paul Schlyter on his website [8].

The Moon, along with Mercury and Venus exhibit phases depending on their position in relation to the Earth and the Sun. As an observer on earth these phases are very apparent and so it would be useful to have them in my app.

To compute the phase angle for Mercury and Venus we need to know the heliocentric distance r (distance from the sun) and the geocentric distance R (distance from earth) for the planet as well as the distance from the sun for the observer s. All these values are already computed for the other planets and so no more calculation needs to be carried out to reach these values.

The elongation of the planet $elong$ can then be computed by:

$$elong = \cos^{-1}(\frac{s^2 + R^2 - r^2}{2sR}) \tag{60}$$

and the phase angle FV can be computed by:

$$FV = \cos^{-1}(\frac{r^2 + R^2 - s^2}{2rR}) \tag{61}$$

The Moon, similar to its position calculation requires a higher degree of accuracy and so a different calculation is used. This involves the use of the Sun's ecliptic longitude $slon$ as well as the the Moon's ecliptic longitude $mlon$ and its ecliptic latitude $mlat$:

$$elong = \cos^{-1}(\cos(slon - mlon)\cos(mlat)) \tag{62}$$

and the phase angle FV can be computed by:

$$FV = 180 - elong \tag{63}$$

4.2.4 Computation of apparent magnitude

In order to calculate the apparent magnitude of the planets in the night sky, the phase angle FV needs to be computed and then along with the geocentric and heliocentric distance the apparent magnitude can be computed. This calculation is different for each planet, for example the calculation for Mercury is:

$$mag = -0.36 + 5\log_{10}(rR) + 0.027FV + 2.2 \times 10^{-13}FV^6 \tag{64}$$

For the Moon the magnitude is:

$$mag = 0.23 + 5\log_{10}(rR) + 0.026FV + 4.0 \times 10^{-9}FV^4 \tag{65}$$

No calculation is needed for the Sun as by definition the magnitude of the Sun is -26.74.

4.2.5 Computation of Jovian moons

The algorithm, used in determining the positions of the moons of Jupiter is outlined by Keith Burnett on his website [26].

The positions of the four brightest Jovian moons depend on the location of Jupiter in its orbit- which was calculated previously.

Using the previously calculated mean longitude for Jupiter bj the elongations of each moon of Jupiter can be calculated by:

$$u1 = 163.8067 + 203.4058643d' + \phi - bj \tag{66}$$
$$u2 = 358.4108 + 101.2916334d' + \phi - bj \tag{67}$$
$$u3 = 5.7129 + 50.2345179d' + \phi - bj \tag{68}$$
$$u4 = 224.8151 + 21.4879801d' + \phi - bj \tag{69}$$

where ϕ is calculated from the Earth-Sun distance res and the Jupiter-Sun distance rej and the corrected different between the mean longitude of earth and jupiter k:

$$\phi = \sin^{-1}(\frac{res}{rej\sin(k)}) \tag{70}$$

and d' is the light travel time correction and is computed by:

$$d' = d - \frac{reg}{173} \tag{71}$$

The distances of each moon from Jupiter can then be calculated by:

$$r1 = 5.9073 - 0.0244\cos(2(u1 - u2)) \tag{72}$$
$$r2 = 9.3991 - 0.0882\cos(2(u2 - u3)) \tag{73}$$
$$r3 = 14.9924 - 0.0216\cos(Gj) \tag{74}$$
$$r4 = 26.3699 - 0.1935\cos(Hj) \tag{75}$$

where Gj and Hj are constants defined by:

$$Gj = 331.18 + 50.310482d' \tag{76}$$
$$Hj = 87.4 + 21.569231d' \tag{77}$$

The x coordinates of each moon can then be easily computed by:

$$x1 = r1\sin(u1) \tag{78}$$
$$x2 = r2\sin(u2) \tag{79}$$
$$x3 = r3\sin(u2) \tag{80}$$
$$x4 = r4\sin(u4) \tag{81}$$

Finally, by using the right ascension of Jupiter jra we can work out the right ascension of each planet:

$$ra1 = jra + \frac{1.6375x1}{60rej} \tag{82}$$
$$ra2 = jra + \frac{1.6375x2}{60rej} \tag{83}$$
$$ra3 = jra + \frac{1.6375x3}{60rej} \tag{84}$$
$$ra4 = jra + \frac{1.6375x4}{60rej} \tag{85}$$

4.2.6 Converting to Horizontal Coordinates

All calculations for the stars, moons and planets give positional information in equatorial coordinates: right ascension and declination. These only provide a position with relation to the celestial equator, therefore in order to present the night sky as it appears to the user at their location I had to convert these equatorial positions into local horizontal coordinates - azimuth and altitude.

The first step was to compute the mean sidereal time. The mean sidereal time is based on both the user's longitude and the time and date.

Like the positional calculations for the planets and moons the day number d from J2000 again needs to be calculated and from this the number of centuries cy can be determined.

Using the day number, the number of centuries and the longitude lon of the user the mean sidereal time mst can be calculated by:

$$mst = 280.46061837 + 360.98564736629 * d + 0.000387933cy^2 - \frac{cy^3}{38710000} + lon \tag{86}$$

The mst is a value in degrees and so needs to be clamped to be between the values of 0 and 360.

The hour angle can then be determined using the mean sidereal time and the RA:

$$ha = mst - RA \tag{87}$$

The azimuth AZ and altitude ALT can then be determined by:

$$sin(ALT) = \sin(DEC)\sin(LAT) + \cos(DEC)\cos(LAT)\cos(HA) \tag{88}$$

$$cos(A) = \frac{\sin(DEC) - \sin(ALT)\sin(LAT)}{\cos(ALT)\cos(LAT)} \tag{89}$$

$$AZ = \begin{cases} A, & \text{if } \sin(HA) < 0 \\ 360 - A, & \text{otherwise} \end{cases} \tag{90}$$

With the horizontal coordinates it is possible to render planets in the correct position for the user's current time and location.

4.3 Graphics

In order to satisfy the specification of the app being both aesthetically pleasing as well as accurate a significant part of the project was concerned with the development of the graphics.

In order for the app to provide the best user experience possible, two criteria were kept in mind:

- Frame rate
- Visual quality

In building my app, there would be a trade off between these two criteria as doing things that would improve the visual quality, such as increasing the number of viewable stars and using high resolution textures, would also have an effect on the performance that could be detrimental to the user experience.

4.3.1 Matrices

Unlike many other graphics programs OpenGL ES 2.0 requires the developer to keep track of all matrices involved in the graphics pipeline as opposed to letting the API look after them. This had both positives and negatives when designing the app. On the one hand it added to the code complexity but on the other it meant that there was complete control over any matrix calculations and this was particularly useful when I wanted to tweak the matrices involved in the camera rotation.

View Matrix The view matrix defines the orientation and the position of the camera in the world and is set when the OpenGL environment is initialised. The app works by keeping the camera stationary and allowing the user to change the camera orientation by rotating. This simplifies the view matrix in that the camera position can always be set to the origin (0, 0, 0) and only the vector describing its forward direction and its up direction needs be defined.

Projection Matrix The projection matrix defines how the 3D world of stars and planets can be converted to a 2D screen viewable by the user. It is defined by a viewing frustum which comprises 6 planes: top, bottom, left, right, near and far. Objects that fall within this viewing frustum are visibile to the user and as the camera rotates, so does the viewing frustum. The projection matrix is particularly important for the zooming functionality involved in the app as the zoom is carried out by skewing the viewing frustum by a given amount.

Rotation Matrix The rotation matrix is used to keep track of the amount of rotation applied to the camera and to adjust the view matrix accordingly upon the rendering of each frame. This matrix is used for both manual and automatic rotation but the calculations involved differ slightly.

4.3.2 Rotations

My app had to be able to handle the the user rotating the camera orientation by swiping on the screen. When building the rotation feature I had a choice whether I wanted the camera to remain at a given orientation and have the graphics objects rotate around it, or instead to keep the objects stationary and have the camera change its orientation. I decided upon the former purely because I predicted there would be a significant overhead in recalculating the positions of objects when the user rotates the screen and therefore it would be difficult to achieve a reasonable frame rate.

I wanted to use the same type of rotation seen in the other apps where the camera only rotated around two axis: its vector describing up (yaw) and its vector describing right (pitch). By only using pitch and yaw it would only be possible for the camera to rotate

left and right and up and down. The advantage of this is that it would be impossible for the camera to end up being flipped upside down and for the user to lose their sense of orientation.

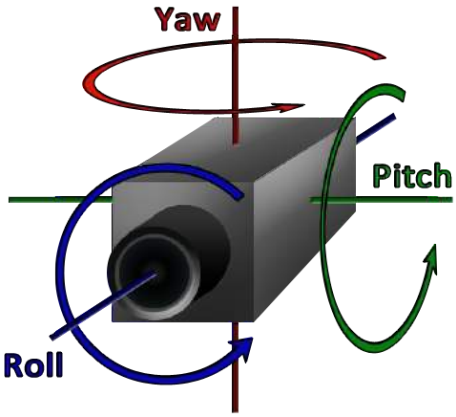


Figure 21: Camera pitch, yaw and roll [14]

On my first iteration of this scheme, I came across the issue that the order of rotations influenced the way in which the camera rotated. This was problematic, as despite only rotating around two axis, the combination of these two rotations resulted in a roll rotation which I did not want. The solution to this was to rotate the pitch by a given amount round the right vector axis and then to rotate the yaw around the up vector axis and this gave the result I wanted.

Another consideration was that there should be a limit to how far up the camera should be able to point, otherwise it would be possible for the user to continue to rotate the pitch upwards and end up flipping the camera upside down, the same principle was true for downward rotations. I therefore clamped the pitch value to an angle between -90 and 90 degrees. I did not want left and right yaw rotations to be treated the same as I wanted to the user to be able to freely rotate left and right infinitely, so a solution to this was to take the modulus of 360 of the yaw angle.

Up until this point all rotations had been done with the horizontal coordinate system in mind, where the pitch would be up and down from the horizon and a pitch of 90 degrees would point to the zenith. This would seem a logical way of rotating the camera if the camera were to represent the view of an observer on earth. However it would make made more sense for an astronomer to be using an equatorial coordinate system where rotations about the horizon were instead rotations about the equator with the celestial poles directly above and below the camera.

In order to achieve this I did not need to change a large amount of my rotation code, but instead all I needed to do was redefine the up vector of the camera so that it pointed toward the north celestial pole and the right vector so that it pointed west.

4.3.3 Building objects vs. building textures

When I started to focus on how I would represent the night sky two solutions became immediately apparent.

Using the concept of the celestial sphere discussed earlier, I could build a large sphere that enclosed the camera so that by rotating the camera the user would be looking at a different point on the inside of the sphere. I could then build an image that represented the night sky or retrieve one from an external source such as NASA and then paint that image as a texture on the inside of the sphere.

Figure 22: Example sky texture [15]

This solution has the advantage that it would produce a realistic and aesthetically pleasing representation of the sky. However as I started to think of the practicalities of using this method there were a number of issues. First, this method gave me limited control over what I could present to the user, if I wanted to add or remove elements from the night sky this would not be trivial. Second, it would be difficult to implement zooming using this method as new images would need to be loaded for zooming in close on planets, moons and stars. Finally, I wanted the app to be highly interactive and I felt that this method would give the user the impression of just interacting with an image as opposed to a 3D world.

I therefore came to the conclusion that building a representation of the night sky using individually rendered objects would provide flexibility and that it would still be possible to provide an aesthetically pleasing solution by using high quality texturing for the objects.

4.3.4 Scale

Despite not using the textured celestial sphere method outlined previously, I applied a similar concept in my version. With the camera placed at the origin (0, 0, 0) simulating the location on earth of the user, I would then the render the stars viewable to the user all at the same distance from camera but at their correct orientations.

The difference in distances from Earth of the planets and the stars is gigantic and if I were to render everything completely to scale the planets would be right up close to the camera and all the stars would be almost infinitely far away from it and impossible to see. Instead I decided that the distance of each star from the camera would be the

same but their magnitudes would be different and it would only be the planets and the moons that would be rendered to scale.

To keep values reasonable I used a scale of Astronomical Units when dealing with size and distance in my application so that one unit in my application would be equal to 149597871 (1 astronomical unit). For example the distance to the Sun would be 1 unit in my application.

4.3.5 Planets

My first iteration of building the planets involved rendering them as spheres and this was achieved through the use of spherical polar coordinates. Where a point on the surface of the sphere is defined by a radius r and azimuth and altitude angles θ and ϕ.

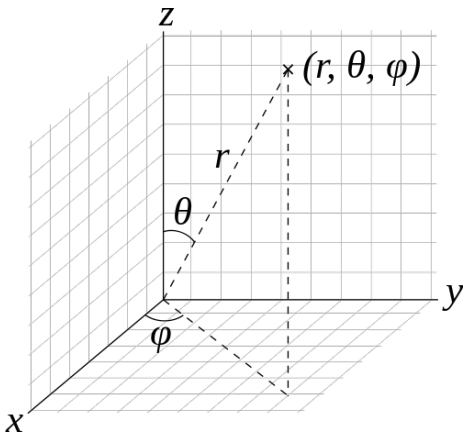


Figure 23: Spherical Coordinates [16]

$$r \geq 0 \tag{91}$$
$$0 \leq \theta \leq \pi \tag{92}$$
$$0 \leq \phi < 2\pi \tag{93}$$

To build a sphere I could build a nested loop through the two angles defined above and compute the cartesian coordinates using the formulae:

$$x = r \sin\theta \cos\phi \tag{94}$$
$$y = r \sin\theta \sin\phi \tag{95}$$
$$z = r \cos\phi \tag{96}$$
$$\tag{97}$$

This would give me the cartesian coordinates with in relation to the centre of the object. To get the coordinates in relation to the camera I would need to add the position of the planet to the x, y and z components.

To produce a sphere with sufficient smoothness required, the steps of azimuth and altitude that I was looping through to be sufficiently small, so if I were to set these as:

$$\frac{\pi}{64} \tag{98}$$

then the resulting sphere would be made up from 1152 vertexes. Though this was not completely infeasible I was keen to minimise the graphic complexity of my app.

As the camera would always be located at the origin and so would only ever see an object from one direction, it seemed wasteful to render an object as a sphere with only a portion of it viewable, so instead I decided to render planets as 2D circles.

The most apparent problem with this was the implications of drawing a 2D object in a 3D world. In the real world no object exists that is fundamentally 2D and so I would need to draw the circles in such a way that would give the impression of 3D.

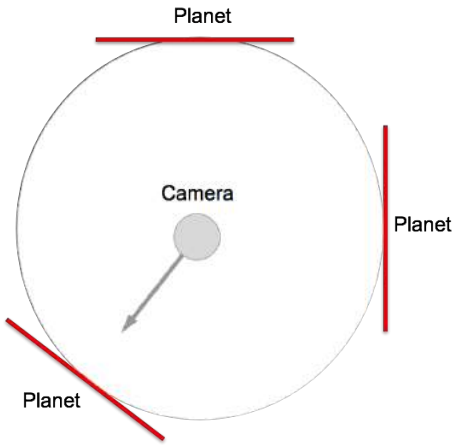


Figure 24: 2D Planet Orientation

The planets had to be built so that whatever position they were in the sky they would always face the camera directly. This involved a large amount of vector algebra.

The position of a planet, like any other object in my application can be described through the use of an azimuth angle and altitude angle along with a distance from the camera. The centre $\vec{c}$ of the object can then be defined as:

$$x = distance \sin(azimuth) \cos(altitude) \tag{99}$$
$$y = distance \sin(altitude) \tag{100}$$
$$z = distance \cos(azimuth) \sin(altitude) \tag{101}$$
$$\vec{c} = (x, y, z) \tag{102}$$

If we then define a direction vector $\vec{a}$ which goes from the centre of the object through the top and centre point of the circle as :

$$x = -\sin(altitude)\cos(azimuth) \tag{103}$$
$$y = \cos(altitude) \tag{104}$$
$$z = -\sin(altitude)\sin(azimuth) \tag{105}$$
$$\vec{a} = (x, y, z) \tag{106}$$

We can easily define a second direction vector $\vec{n}$ as the normal of the circle towards the camera position and this is simply:

$$\vec{n} = -\vec{c} \tag{107}$$

If we then take the cross product of the two direction vectors we find a third vector $\vec{b}$ that lies in the plane defined by the circle.

$$\vec{b} = \vec{a} \times \vec{n} \tag{108}$$

We now have two direction vectors $\vec{a}$ and $\vec{b}$ that define the plane that the circle lies in. Using these vectors as the axis we can use polar coordinates to draw a circle in the plane. If θ represents the angle of the circle and lies in the range $0 \leq \theta < 2\pi$ then the cartesian coordinates of a point on the edge of the circle with radius r can be defined as:

$$x = \vec{c}.x + (r\cos(\theta)\vec{a}.x) + (r\sin(\theta)\vec{b}.x) \tag{109}$$
$$y = \vec{c}.y + (r\cos(\theta)\vec{a}.y) + (r\sin(\theta)\vec{b}.y) \tag{110}$$
$$z = \vec{c}.z + (r\cos(\theta)\vec{a}.z) + (r\sin(\theta)\vec{b}.z) \tag{111}$$

The results is a circle that is oriented towards the camera.

By using a circle as opposed to a sphere, the number of vertexes needed is close to the square root of the number of vertexes needed for a sphere which is a significant improvement.

With the vertex positions of a planet calculated, a 2D texture can be applied to the circle. As you can see in Figure 25 this gives a very realistic representation and it is not immediately obvious that the object itself is 2D.

4.3.6 Lunar and Planetary Phases

For Mercury, Venus and the Moon the amount of the face that is visible to an observer changes in relation to time as the angles between the Sun, the planets/Moon and the earth change and my app needed to represent this. These changes in phase could be simulated through the use of a global light system or through changing textures.

A global light system would work by having the Sun act as a light source in the 3D environment and Mercury, Venus and the Moon would need to be rendered as spheres.

Figure 25: 2D Planet Texture

If the relative scale of everything in my app was correct then depending on the position of Mercury, Venus and the Moon objects in relation to the light source (the Sun), the amount of each object lit up would change and the user would observe them to have phases.

The benefit of this solution would be that all the phases would be dynamic and so it should be possible to recreate every phase for each planet. For my application however I determined that this would not be feasible as adding in global lighting to my application together with the use of spheres as opposed to circles would have a significant impact on the performance, which for the sake of a small feature did not seem worth it.

My solution was to alter the texture being applied to each planet semi-dynamically. This worked by creating about 5 or 6 textures for Mercury, Venus and the Moon for each of their distinct phases and then depending on what the calculation for the phase of each returned, the appropriate texture would be applied to the planet/moon. This required a bit of work in collecting the images for each phase of each planet/moon but the overhead in changing the texture was very small and had no real impact on performance.

4.3.7 Stars

Unlike the planets, where the number of objects would be relatively small, I needed to efficiently build and render upward of 10,000 stars. Moreover, unlike the planets, where the position calculations would be done once and then only redone if the time or location was changed, calculations for the positions of stars would need to be done every time the user zoomed in and more stars came into view. Therefore the vertex building of all the stars would need to be completed in a small amount of time so that the user would not be hindered by a lag.

For this reason, rendering stars as spheres was not feasible as it would require upwards of 10,000,000 vertex's to be computed and rendered. I then decided to use the same

process I had used for the planets and to test the performance of that. I initially tried the process with the 200 brightest stars in the night sky and naturally this increased the initial loading time as the first set of stars needed to be built. However, this only increased the initial loading time by a few seconds and so did not detriment the user experience too much.

A key difference between viewing planets and stars in my app was that stars would only ever be seen as small circles at reasonable distance from the camera and would never require the accuracy needed by zoomed in planets. I decided to simplify the process of building stars by using an OpenGL primitive called Point Sprites- 2D graphic quads that are defined by a position vector and a size. Using point sprites as opposed to circles provided significant performance gains for a number of reasons.

Firstly, where circles would require a large number of position vertices to be specified in order to be rendered properly, point sprites required only one which meant that a sky filled with 10,000 stars would only require 10,000 vertices. Secondly, the large amount of vector algebra to orient circles so that they always faced the camera was not needed, as by definition point sprites always face the camera.

One consideration with using point sprites is that they are rendered as rectangles rather than circles. To deal with this I created a separate fragment shader that would be used to render the point sprites and used the formula for polar coordinates of a circle to eliminate pixels of the point sprite leaving just a circle.

After switching to point sprites the rendering times of my app decreased drastically and I was able to greatly increase the number of stars being generated both initially and when the user zoomed in.

It did not make sense to draw the distances of the stars to scale as distance between planets are measured in millions of kilometres and distances between stars are measured in light years. To get around this issue I decided to make use of the apparent magnitude data available in the HYG database stored on the device. Using this data I could then alter the size of a star point sprite depending on its apparent magnitude so that brighter stars appear larger and dimmer stars appear smaller. Every star would still be rendered at the same distance away from the camera with just differing orientations, and so an apparent celestial sphere was formed as shown in Figure 26.

An issue I had with displaying planets as circles and stars as point sprites was that stars were far brighter and easier to see as the circles for planets were very small and a large distance from the camera. This did not satisfy my specification as in the real night sky many of the planets have an apparent magnitude that is significantly greater than most stars and are easy to spot.

When I looked at how existing planetarium applications were addressing this problem I noticed that without zoom all the planets that should be visible were actually being rendered in a similar fashion to stars as a bright white point and then as the zoom was increased they would gradually change into their proper representation.

The advantage that this had for my app was that I had already written code to display a large number of stars. Planets largely had the same attributes as stars - right ascension, declination, azimuth and altitude - and so could easily be added in the existing code to render stars. The only attribute that was missing was that to be displayed as a star they would need an apparent magnitude but this could be calculated easily using the formulae described earlier.

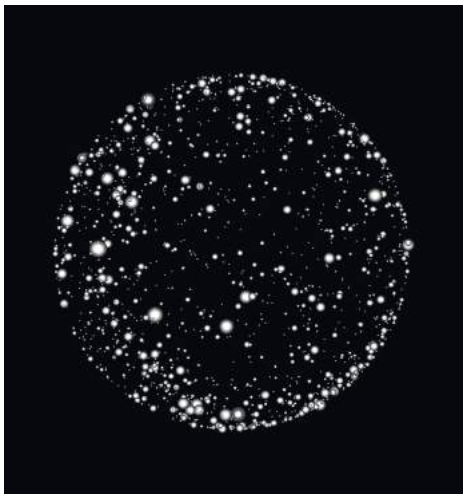


Figure 26: Stars forming a celestial sphere

4.3.8 Vertex Buffer Objects/Index Buffer Objects

The rendering of all objects involved in the 3D environment of application involved the use of the OpenGL feature of vertex and index buffer objects. This works so that when the objects in the environment are being built the data that is associated with each vertex such as the position, colour and texture would be uploaded into the graphics card of the device as opposed to the main system memory. This has significant advantages for performance as the objects can then be rendered directly from the graphics card as opposed to being retrieved from the system memory.

Vertex buffer objects became extremely important in building and rendering stars in a short period of time as was required. Instead of creating a vertex buffer object for each star I instead created one vertex buffer object for all the stars at once. Each vertex in the buffer corresponded to a different star and the index of each vertex was stored in an index buffer object. This significantly reduced the overhead in making calls to the OpenGL API to bind and create buffers, producing a drastic decrease in the build times for stars. When rendering the stars all that was needed was to loop through the index buffer object drawing each vertex as though 1000's of stars made up one object.

The time taken to build the array of stars could be reduced significantly again through the use of rebuilding buffers. When zooming in and more stars needed to be generated, instead of creating a whole new vertex buffer object, the vertex data of the new set of stars could be stored in an array and then uploaded into the existing vertex buffer object on completion. This again reduced the overhead in creation and deletion of buffers and significantly reduced the lag involved with zooming in and creating more stars.

4.3.9 Zoom

When it is not zoomed in at all the app should display only the brightest stars in the sky. I worked out that about 1000 stars was a good amount to produce an accurate portrayal of the sky. When the user zooms in, the number of stars should increase. By zooming

in on a relatively barren part of the night sky a number of dimmer stars should appear that would not ordinarily be seen in the sky with the naked eye.

The first thing I had to decide was how I wanted the user to access the zoom function and I could think of multiple ways to go about this. One way was to have a slider on the side of the screen that the user could move up and down to zoom in and out in a simple fashion similar to Google Maps. Creating such a slider would be relatively simple, however I felt it would give the user less fine tune control over how much the camera was zoomed. Moreover with the zoom already being implemented by swiping on the screen I felt that using a slider along with a swipe gesture would not feel particularly natural.

I decided to implement zoom in my application by using the pinch gesture described in the Android API and used in a large number of existing Android applications. The implementation of the pinch to zoom function in itself was relatively simple however linking it in with all the other objects in my app bought about a few issues.

On the first iteration of the pinch to zoom feature, I found that the zoom amount was far too large and was very difficult to use. The reason was I was dividing the current zoom by the amount returned by the pinch gesture (where a zoom of 1 would be no zoom and 0 would be an infinite zoom). After investigating the issue I learnt that the zoom worked in a logarithmic scale and so I had to considerably reduce my scale factor to produce a more controllable zoom.

Another consideration was that when the camera was zoomed right in on a very distant object, camera rotations were not adjusted for this new zoom and so trying to move the camera at a very high zoom was extremely difficult. To rectify this I changed the camera rotations to be inversely proportional to the zoom amount so a camera would only rotate 1/10 of the actual amount at a zoom of 10x.

I implemented my zoom function so that every time a zoom occurs a call to the database is made that retrieves a number of the brightest stars that is proportional to the amount of zoom and then builds and renders them. When I first implemented stars as circles the build stage took a significant amount of time due to the large number of vertices and so I had to reduce the amount of stars that could be rendered on a zoom. However, when I instead used point sprites the build time decreased hugely and so I could significantly increase the amount of new stars that could be displayed in every zoom.

One issue I still had however, was that while the user was pinching to zoom, the frame rate would drop and the zooming would seem to have a lag. This occurred because while the pinch action was occurring multiple calls would be made to the database to retrieve more stars in the background and this created a performance drop in the graphics and UI tasks. To rectify this I created an extra function that would change the zoom of the camera in the app but would not update the amount of stars to render. I then had the pinch to zoom function call this function, while the pinch action was occurring and only when the action had completed would a call to the original zoom function be made to update the number of stars.

4.3.10 Object Picking

In order for my app to be interactive for the user I needed to find ways for the user to interact with the night sky through selecting different things and unlike selecting

objects in regular 2D applications, selecting objects in a 3D OpenGL environment is non trivial.

The first step was to find the pixel on the screen that the user had selected and its offset in both the x and y directions from the top corner of the screen. The viewport of the camera is then found. The viewport is the 2D rectangle which is used to project the 3D scene to the 2D image. By using the OpenGL function gluUnProject() with arguments of the x and y position, the view matrix, the projection matrix and the viewport it is possible to find the 3D coordinates that the 2D position is projected to.

The obvious issue with this method is that it is impossible to determine at what depth the user is selecting on the screen, are they selecting an object which is 2 units away from the camera or one which is 1000?

To solve this I first needed to normalise the 3D coordinates and as the camera was placed at the origin these coordinates could define a direction vector and therefore a line. In order to find out what object the user was selecting I could loop through all the objects and work out their minimum distance to the line; the one with the smallest value would be the object selected by the user.

If we call the line created by the users selection $\vec{l}$ and the the position vector of the object we are measuring the distance from is p then the minimum distance d from p to $\vec{l}$ is defined as:

$$d = \frac{|\vec{l} \times p|}{|\vec{l}|} \tag{112}$$

An alternative method I explored for selecting objects was the use of a colour buffer. The principle was to render each object in the 3D world twice- the first time with all textures and lighting and the second time as simple as possible with just one unique colour for each object. Then when a user selects an object the colour value in the buffer for the selected pixel is read and lookup can occur to find out which object it corresponds to. Although this method would remove all the vector calculations involved in determining the closest object, the overhead in creating duplicates for over 10000 objects would have a significant effect on performance and my method, even when carried out with over 10000 stars, had no noticeable lag.

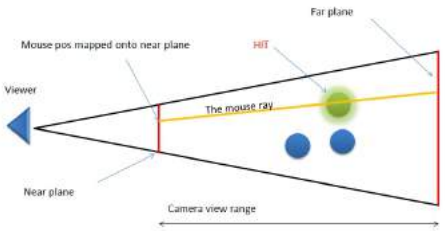


Figure 27: Object picking with rays

4.3.11 Equatorial Lines

In order for the app to be useful for potential astronomers to find objects in the night sky there needed to be some fixed frame of reference. This reference would be a grid of angles as seen in Figure 29 where the camera would be in the centre and the lines would depend upon whether the user wanted to work in terms of horizontal or equatorial coordinates. For a horizontal coordinate system the horizontal lines would be the various altitudes and the vertical lines would be the azimuths. The circumference line half way up the sphere would define the horizon with the intersection of all the azimuth lines at the top being the zenith and the bottom being the nadir.

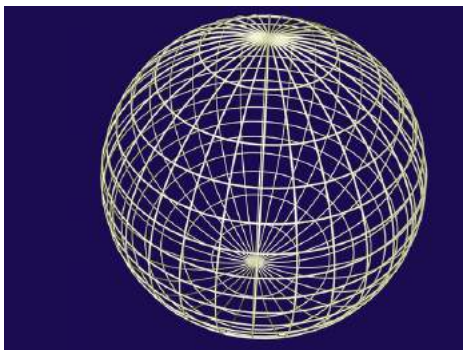


Figure 28: Equatorial grid

This method of defining the lines was used in the first iteration as to me it made sense to have the horizon as the frame of reference. After showing my app to some users it became apparent that from an astronomers perspective the equator is a far more useful frame of reference as planets will never lie much more than 30° away from it. Where a horizontal grid works in terms of azimuth and altitude, an equatorial grid works in terms of right ascension and declination with the top intersection of all the right ascension lines forming the north celestial pole and the bottom forming the south celestial pole. This makes finding the location of the Polar Star particularly easy as it will always be located very close to the intersection of all the equatorial lines at the north celestial pole.

My application is designed in such a way that the locations of all objects are described in terms of horizontal coordinates, therefore drawing the initial grid was particularly easy.

To draw the altitude (horizontal) lines I could just created a nested loop that started with an altitude of -90 and went through in steps of 15 up to +90 and at each step drawing a circle by fixing the altitude and drawing a point for every azimuth. The same process could then take place again but with the azimuth and altitude swapped and a sphere of lines would be formed.

To draw a grid that was defined through the use of equatorial coordinates meant that the same process would be carried out as before but this time with right ascension and declination and before drawing each point I would first transform them to horizontal coordinates. This naturally bought about an added complexity in the calculations but as the grid only needs to be constructed whenever the location is changed this was not

a problem.

I decided to draw the grid with dotted lines to distinguish it from the rest of sky but also to make it less obtrusive. I also constructed extra lines at 23° above and below the equator (Tropic of Cancer and Capricorn) and coloured them yellow along with the equator as these are particularly useful reference points in the night sky.

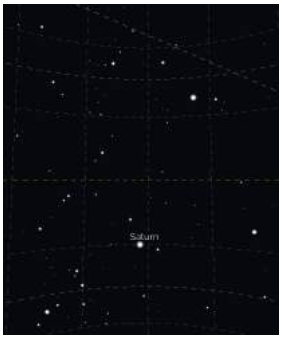


Figure 29: Equatorial lines

4.3.12 Text

OpenGL doesn't provide dedicated methods or primitives for rendering text in a 3D environment, so I had to make a few choices about how I wanted to do this. The first option was to create letters to make up a word by physically drawing all the vertices of each letter and then filling in the text using a colour.

There are a number of issues that arise from rendering text in such a way. Firstly, I would have to manually work out the vertex positions that make up each letter of the alphabet (Figure 30) and this would take a significant amount of time as to create the smooth curves on letters like 'c' and 'o' the number of vertices would need to be large. Secondly, after performing this step I would still have the same issue I encountered with the circles, in that they would always need to face the camera and this would require extra computation.

Figure 30: Constructing text with vertices

Instead I chose to create a simple shape at the position I wanted the text and then to use a texture of that text on the shape. This meant that for each word I would need to create a new word texture but given the small amount of text actually needed in the 3D environment of my app the extra work was minimal.

The first shape I chose was a rectangle which would be defined by 4 vertices in the 3D environment and then a texture for the given text could be used to fill in the shape. An issue with using a rectangle was that it was complex to work out the values for each vertex when the rectangle needed to be placed at a given location. Once I had managed to work out the vertices, it was not trivial to move the rectangle to another point in the 3D world as I would need to recalculate all the vertices.

The most significant drawback however, was that often I wanted the size of the text to remain constant even when the camera was zoomed in, for example the labels above each planet. By specifying 4 vertices of the rectangle it was not possible to do this without recalculating the vertices whenever the zoom was increased.

I then took inspiration from my use of point sprites for the stars, but rather than making the point sprites circular, I kept them as rectangles and just applied the text texture onto them. The same effect could then be achieved as with the original rectangles but rather than specifying four vertices all I needed to do was give an azimuth and an altitude to calculate the vertex and an integer to change the size. Moreover, by using point sprites it meant that the text always faced directly at the camera and would scale with zooming.

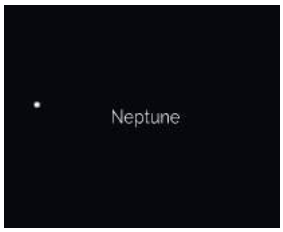


Figure 31: Rendering text using point sprites

4.3.13 Augmented Reality

For my app to be useful for observing the sky at a given time the user needs to have a method of matching up what they can see on their device with what is actually in the sky. I went about solving this problem by using the same principle as was presented in existing solutions. The device could be physically tilted and rotated so that it would face a particular point in the sky and that portion of the sky would be presented on the screen.

I had designed existing rotations in my app with this feature in my mind as all rotations were carried out using a yaw (rotate right or left) and a pitch (rotate up or down). From my research I determined that physical rotations of a device could be specified using yaw, pitch and roll. For manual rotations I omitted roll as I did not want the user to have the ability to flip the camera upside down but for a natural augmented reality experience it would be needed.

To get the pitch, roll and yaw values of the phone I had to access two sensors in my device- the accelerometer and the magnetic field measure. The accelerometer works out the change in the local orientation values for the device and then these values can then be mapped to global coordinates by using the direction of north from the magnetic field measure.

Once I had retrieved the values I substituted the yaw and pitch values into the rotation matrix for my application. This did not work particularly well as the rotation was far too sensitive and would jump around uncontrollably making focusing on a point difficult. In order to decrease the sensitivity of the device I decided to take a moving average of the last 3 readings and then pass that into my rotation matrix and this produced a slightly better result. After trying a range of values I decided upon 10 readings as a compromise between having a smooth experience and too much of a lag in updating the orientation of the device.

Without using the roll value my rotations were only occurring around two axis where the physical rotations of the device worked in 3. For this reason I decided to implement a third rotation about the z axis that would only be used in augmented reality for the roll of the device and this felt far more natural.

When I was working on the manual rotations of my device, my final implementation worked by rotating around the equatorial grid as opposed to the horizontal grid as it had more use for astronomers.

For augmented reality, however, this did not make sense as when the device was facing directly forward and straight it would be facing the horizon. Therefore I concluded that augmented reality should work in one coordinate system and manual rotation should work in another and so the up and look vectors of the camera should be changed whenever the user changed modes.

4.4 Database

When designing my app I needed to make several decisions about whether it was possible to calculate information about given celestial objects on the fly or whether I needed to store the data and if so, where would the information be gathered from and in what format would it be stored.

I concluded that information about planets and moons such as positions could easily be computed directly using mathematical formulas and due to the small number this would be feasible. Due to the large number of stars in the night sky I determined I would need to gather the data from an external source and store it.

It is far easier to view the night sky in remote places due to the significantly lower light pollution, therefore it is likely that users of my app would have limited or no access to the internet. For this reason I concluded that it would not be feasible to collect information from the internet every time the app is used.

4.4.1 HYG Database

The HYG Database is a collection of multiple star catalogues with a combined total of 120,000 stars. It is subset of data from the three major star catalogues- the Hipparcos

Catalogue, the Yale Bright Star Catalogue and the Cliese Catalog of Nearby Stars.

I collected the HYG database as a .csv text database file which I could then bundle with my application to be used later. As the database was stored as a .csv file, the file was only 9mb in size and therefore did not increase the size of the app by a noticeable amount.

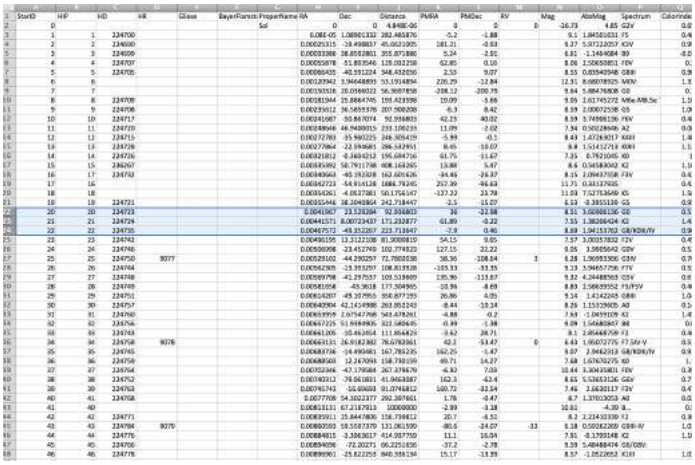


Figure 32: .csv HYG Database

4.4.2 SQLite

The tradeoff with storing the database as a .csv file was that trying to sort and access the data in a reasonable amount of time would be impossible until it was loaded into a real SQL database. For this I used an SQLite database as it is the official database program available to Android devices. Had the database implementation for my app been more involved, I may have looked into alternatives but the number of commands and general complexity was low and SQLite satisfied my requirements.

4.4.3 Database Building

SQLite databases are persistent on Android devices, so the transfer of the .csv file into the SQLite database needed only to be done when the user opened the app for the first time. When the app is uninstalled by the user the SQLite database is deleted and will need to be recreated when the app is installed again.

From the .csv database I collected star ID's and common names, positional information in the form of the right ascension, declination and distance and extra information about each star such as the colour index and apparent magnitude. I decided to not load in the other attributes in the .csv database as I had no use for them in my app and by eliminating them I could reduce the access time in retrieving records from my SQLite database at runtime and provide a smoother experience for the user.

However, when I distributed my app to users for testing who had devices that were not as powerful as my test device the initial database loading phase took over 15 minutes

compared with the 90 seconds on my device. I felt that this was far too long for the user to wait and so I needed to find a way to reduce the time.

I worked out that I could extract the database that had been created while testing and bundle it with my app when distributed. Then when the user starts the app the database file is copied into the correct database folder to be used. To carry this out I used a modified SQLite Android library created by Jeff Gilfelt and distributed on Github [27]. This was a simple change to use the modified SQLite version rather than the inbuilt android SQLlte implementation. Then the prebuilt database was placed in a folder of my app that is used to store assets for the app such as fonts.

5 User Interface Walkthrough

5.1 Font

The main font used throughout my app is a Sans Serif font called Raleway [28], created by Matt McIerney and expanded on by Pablo Impallari and Rodrigo Fuenzalida. It is completely free to use for both personal and commercial use.

Figure 33: Raleway Font

5.2 Splash screens

When the user opens the application they are presented with a splash screen displaying the name of the app- Galitarium and the logo. At this stage the pre-built database stored along side the app is copied into the correct place to be used.

Once the process has completed the screen changes to another splash screen with a display saying that the sky is being generated. At this stage, in the background the equatorial coordinates of all the celestial objects are being computed. If the user has their GPS turned on the application will attempt to retrieve the location of the user and use this to work out the locations of the objects in the night sky specific to the users location. If this is successful, the longitude and latitude of the user are displayed and if not the default coordinates of London are used.

5.3 Main screen

Once the graphics of all the objects have been built the user is presented with the main screen and this is the platform from which the user can begin to interact with the sky. The three horizontal yellow lines from top to bottom represent the tropic of cancer, the equator and the tropic of capricorn respectively and the white line represents the users horizon which changes depending on the location. In the corner of the screen the current location is displayed along with the date and time in UTC.

To explore the sky the user can swipe up, down, left or right. If the user rotates the camera up as far as they are able they are able to see the north celestial pole with the Polar Star very close to it.

The user is able to observe some of the planets by finding a point that is indicated by a text box with the name of a planet and looking directly below the text. Uranus, Neptune and Pluto cannot be observed at the default distance as they are not bright enough in the night sky but all the other planets will appear as bright points of various sizes depending

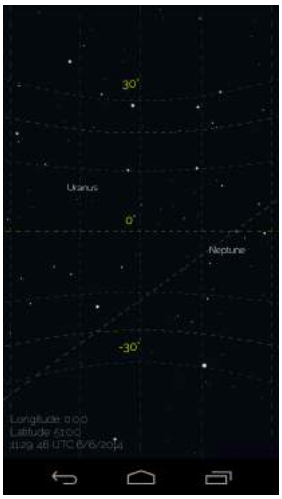


Figure 34: Main Screen

on their magnitude. For example figure 36 shows Venus being displayed without any zoom.

5.4 Selecting planets and stars

Any object in the night sky can be selected by the user tapping on it. When an object is selected a dotted circle will form around the selected object and an information box appears in the corner of the screen. Figure 37 shows Sirius, the brightest star in the night sky after being selected by the user.

The information box contains different information depending on what type of object is selected. All objects will have positional information in both horizontal and equatorial coordinates as well as a distance. If a star does not have a common name it will show an ID instead.

5.5 Zooming

If the user pinches the screen they can zoom in on a portion of the sky. As the zoom is increased the number of stars being displayed in the app will increase and more distant stars will pop into the view of the camera. Once the zoom passes a given threshold value, planets and moons will instead be rendered as they are as opposed to how they are viewed by an observer from earth. For example, Figure 38 shows the sun when zoomed in. All the stars that pop into view when zooming can be selected to bring up more information about them. The amount of zoom allowed in the app has an upper bound as this is the amount of zoom needed to view Pluto up close.

A good example of the zoom function in action is zooming in on Orion's Belt and this can be seen in Figure 39.

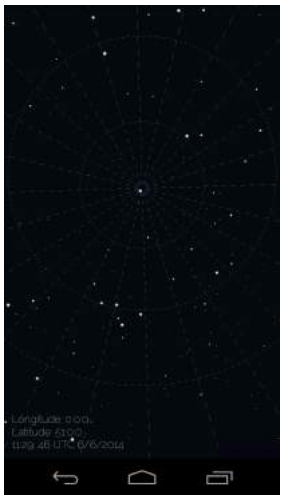


Figure 35: Polar Star at the North Celestial Pole

5.6 Side drawer

In order to make the menu as unobtrusive as possible the user can access a side drawer with the menu by swiping from the left of the screen and this is shown in Figure 40.

From here the user can access the different features of the app from one central point.

5.7 Find planets

If the user selects the 'Find Planet' option at the menu they are taken to another menu with a number of celestial objects as shown in Figure 41.

If the user selects one of the options the menu will close and the camera will rotate from its current position to find the selected object in the night sky and zoom in close on it. Figure 42 shows Jupiter after being selected in the Find Planet menu.

5.8 Change Time

The second option in the menu allows the user to change the date and time for which the night sky is displayed. After selecting the change time option in the main menu the user is presented with a new menu for the date and time as shown in Figure 43.

If the user selects the 'change to current' option the current date and time will be set for the user. If the user selects the 'change date' option a fragment is revealed that gives the user the option to select a day month and year. Any date within 100 years of J2000 will work but with the accuracy of decreasing the further the date is from J2000. After selecting a date, the night sky will refresh to show the night sky on that date using the most recently selected time of day by the user. If the user then wants to change the time of day they can do so by selecting the corresponding option in the menu. In the

Figure 36: Venus as seen without zoom

fragment that appears, the time being selected is in local time and once chosen will be converted to UTC time for the purpose of the calculations.

5.9 Change Location

Similarly to changing the time if the user chooses the 'change location' option in the main menu they are bought to a new menu with various location options. If the user selects the option to 'use current location' then the app will attempt to determine the location of the user in longitude and latitude and then rebuilt the night sky to match this location. If the user's location can not be determined a message will alert the user of this.

The user also has the ability to specify a specific location using the set latitude and set longitude options. In the set latitude function the user can either select the coordinate to be north or south of the equator using the switch and then the degrees (0 to 90), minutes (0 to 60) and seconds (0 to 60) can be chosen. Similarly for choosing the longitude the user can select the coordinate to be either west or east of the meridian using the switch and then the degrees (0 to 180), minutes (0 to 60) and seconds (0 to 60) can be selected.

After changing either the longitude and/or the latitude the night sky will rebuild itself to be accurate for the specified position.

5.10 Settings

If the user selects the 'settings' option in the menu they are taken to a new menu which gives them extra options to change aspects of the app. If the user checks the 'augmented reality' option they will turn on the accelerometer in the phone and all rotations in the app will be based upon the phones orientation and this is a useful way in finding stars and planets in the actual night sky. While in this mode it is not possible to use the find planet option in the menu however it is still possible to select things on the screen to

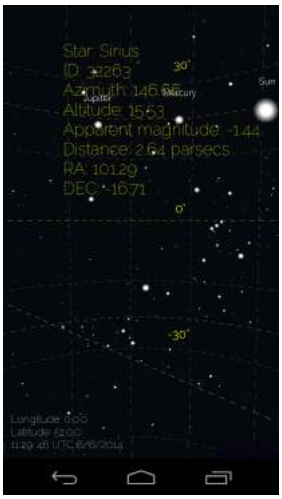


Figure 37: Selecting a star

bring up more information. Figure 45 shows the app along with the augmented reality feature being used to show the sun setting.

5.11 Phenomena

Due to the high accuracy of calculations in the app and a correct scale it is possible to recreate a number of celestial phenomena by changing the location and time in the app.

5.11.1 Transit of Venus

To see Venus passing in front of the sun the user should do the following. Change the date to the 6th June 2012, the time to 03:30, the location to 51° N and 0.1° E and select Venus in the 'find a planet' menu. This phenomenon occurs very infrequently and its representation is shown in figure 46. If the user observes Venus itself they will notice that it is completely black and this is because its phase its correct as the sun is directly behind it and so no light would reach the side facing earth.

5.11.2 Transit of Mercury

To see Mercury passing in front of the sun the user should do the following. Change the date to the 9th May 2016, the time to 14:57, the location to 51°N and 0.1° W and select Mercury in the 'find a planet' menu. Similarly to the Transit of Venus, this phenomenon occurs very infrequently and its representation is shown in figure 47. Again, similarly to the transit of Venus the face of Mercury appears black due to the phase it is in.

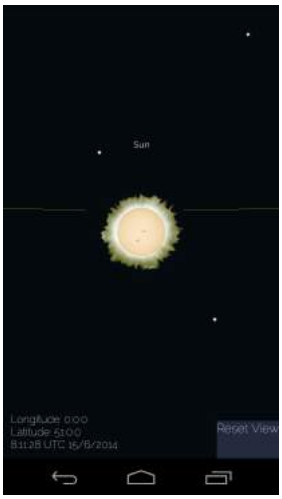


Figure 38: Sun zoomed in

5.11.3 Solar Eclipse

To view a solar eclipse the user should do the following. Change the date to the 11th August 1999, the time to 12:03,the location to $45.1°$N and $24.3°$E and select the Moon or the Sun in the 'find a planet' menu. Figure 48 shows its representation in Galitarium. The moon appears black as again, its phase is a new moon due to the sun being located behind it in relation to the earth.

5.11.4 Arrangement of Jovian Moons

A feature unique to my app is the ability to view the arrangement of the moons of Jupiter at any point in time. An example of this is shown in Figure 49.

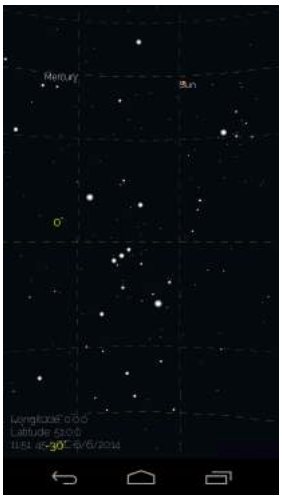


Figure 39: Orion zoomed in

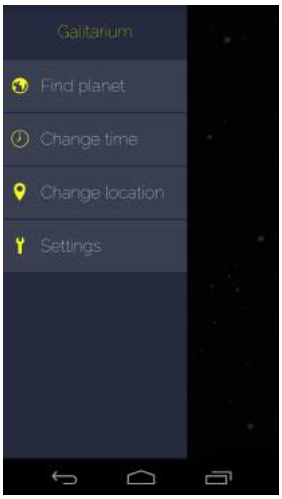


Figure 40: Menu

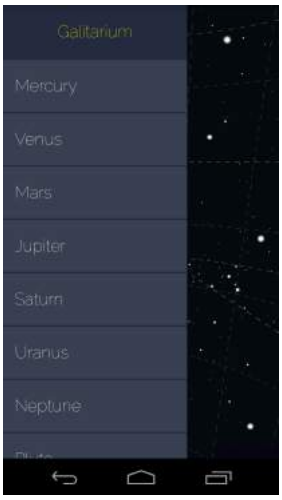


Figure 41: Find Planet Menu

Figure 42: Jupiter up close

Figure 43: Date and Time Menu

Figure 44: Location Menu

Figure 45: Augmented Reality being used to view a sunset

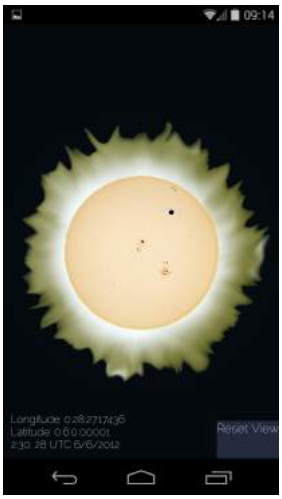


Figure 46: Transit of Venus in Galitarium

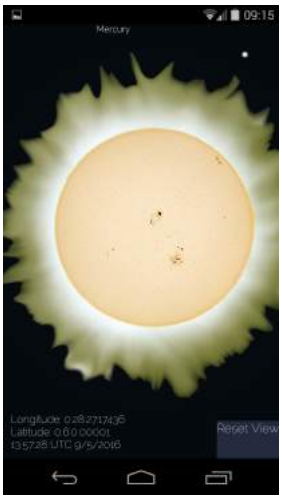


Figure 47: Transit of Mercury in Galitarium

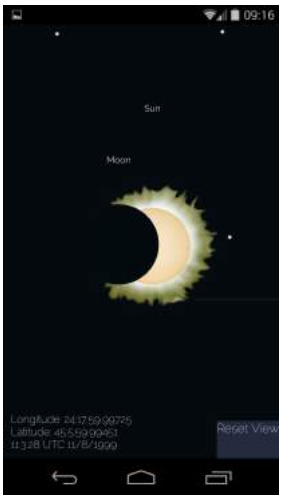


Figure 48: Solar Eclipse in Galitarium

Figure 49: Jovian Moons in Galitarium

6 Evaluation

To evaluate the success of my app I tested it in 5 ways:

- To what extent has the app met the outlined specification?
- How does the app perform with regards to latency and loading times?
- To what degree of accuracy are calculations for positions and scale carried out?
- How closely does the night sky in my app resemble the actual night sky?
- What feedback do real world users have about the app?

I will then go on to talk about the limitations of the app and how future extensions could counter these or enhance the app further.

6.1 Satisfaction of goals

The specification for the project was as follows:

Planetaria have existed for many years, but few allow interactive stargazing over the web.

The project involves the implementation of some existing mathematical formulas (describing the movement of objects in the sky), building a star / planet database, and setting up a fast, user friendly interface. The main requirement is speed: one problem there is that the actual picture of the sky needs to be generated as a transferable file, rather than a bitmap, the other is that good database construction and management should deal with a database of millions of stars.

The first point concerns the construction of a star database as a transferable file that deals with a very large number of stars. As I showed in my implementation, each star in my app is generated as a single object with positional information extracted from a database of 100,000 stars rather than just using a pre rendered image of the night sky. Therefore I feel I have met this point of the specification.

The specification also makes reference to a fast user-friendly interface. Although I have tried my best to provide such an interface in my app, I cannot say with any certainty whether this was successful without analysing user feedback, which I will do later on in the evaluation.

Another requirement is the accurate calculation of the orbits of planets and their moons, and thereby the correct positioning of those objects in the generated pictures. The formulae for this can be found in the literature. This means that you must translate a mathematical formulae (containing cos, sin, etc) into a java program.

In the implementation section I discussed how I went about computing the positions of the planets and moons using set algorithms programmed in Java. I was able to do this for all the planets of the Solar System as well as both Earth's moon and the moons of Jupiter. In order to quantify the accuracy of these calculations I will further explore my results later on in the evaluation.

I want you to build a virtual web-based telescope, that will accurately show the sky for any place on earth, at any time, and at any viewing angle. The telescope should be

able to zoom in and out, displaying the weaker starts and galaxies when it zooms in. In view of the size of the star database, this has to be managed well to keep response time down. Planets (and their moons) need to plotted correctly. A correctness test will be in showing the transfer of Venus, and eclipses, that should be displayed correctly.

As discussed in both the implementation and walk through sections of the report it is possible for the user of the app to select any longitude and latitude on the earth along with any time and date, and the night sky generated will be accurate for those parameters. The user can swipe on the screen to change the viewing angle and by pinching the screen the zoom can be increased to display objects in the night sky that are less visible. I feel that I have satisfied these points by demonstrating them in my walkthrough.

The specification makes reference to reducing latency when extracting new stars from the database. The success of this in my app will be further explored quantitively in the performance section and qualitatively by user feedback.

In my walk through I demonstrated the transit of Venus correctly and so the correctness test for planetary positions outlined in the specification can be confirmed.

6.2 Performance

The next area is a quantitative evaluation of the performance of the app carried out on the test device - a Nexus 5 running Android 4.2.2.

6.2.1 Load Times

For my app to be useable it should be capable of being opened and the graphics displayed within a reasonable amount of time. A good way to measure this is to compare the opening time with that of existing solutions. I tested this by measuring the time taken to load up the representation of the night sky from selecting the icon at the phone menu. The results were as follows:

Google Sky Map: 5.57 seconds

Sky Chart: 8.684 seconds

Galitarium: 9.135 seconds

These results show that out of the 3 apps, Galitarium was the slowest at opening, but I don't see any great significance in this result. The difference between Google Sky Map and Galitarium is just under 4 seconds and I feel this extra time isn't too detrimental to the user experience. Also the significant extra number of stars and the added features of Galitarium more than justifys the extra time needed for the start up.

6.2.2 Database Latency

In the specification it was mentioned that the latency in database queries should be kept to a minimum so that when zooming in and more stars need to be generated, the lag in doing so would be small.

I inserted a timer into my code which measured the latency of retrieving different numbers of stars and the results were as follows:

1200 stars - 197ms

2000 stars - 225ms

4000 stars - 473ms

7500 stars - 888ms

12000 stars - 1532ms

17000 stars - 4085ms

31200 stars - 7876ms

For the base number of stars the latency is very small and feels nearly instantaneous. It is possible to increase the number of stars to about 8000 while still keeping the latency below 1 second. This works very well as with 8000 stars it is possible to fill the night sky very densely with stars and provide a good experience for the user when zoomed in.

At high zoom levels (such as viewing Pluto zoomed in) the number of stars is very large and so naturally the database query times are greater. With the maximum amount of stars the latency is almost 8 seconds. In the first implementation of database queries in my app, the query locked the user interface which meant the user had to wait until the query was completed before rotating the camera or selecting anything. If I had left the implementation like this the user experience would have been greatly hindered - especially with devices that are not as fast as the test device I used.

However, as I changed the database queries to run in the background, I feel that latency has become a less important factor. The user can still interact with my app during these wait times, and as soon as the stars are loaded they will pop into view.

It could be argued that the latency times for very large numbers of stars are too long. However I feel that it is unlikely that the user will be frequently viewing stars at such high zoom levels. The most interesting constellations and stars are generally visible without needing too much zoom.

6.3 Computational Accuracy

In order for my app to be useful to astronomers studying the night sky, it needs to have a reasonable amount of accuracy in terms of the relative scale of the night sky along with the positional information.

6.3.1 Stars

The HYG database provides the right ascension and declination of stars to 8 decimal places and when extracting this information I stored the values as floating point numbers in order to preserve this level of accuracy. When converting the right ascension and declination to azimuth and altitude, naturally some accuracy was lost as this calculation

is dependant on both the location of the user and the current time, but I did my best to avoid rounding errors.

The position of the graphics object representing a star is generated using the unrounded values of the azimuth and altitude, but when the user selects a star in the app the resulting values for its position are rounded to 2 decimal places.

The apparent size of a star is determined by its apparent magnitude and this is is to 2 decimal places in the database. Because there is no way to compute the size of the star graphic from a given apparent magnitude I had to use trial and error in generating a formula that worked this out and gave the best representation.

6.3.2 Planets

Positions of the planets for UTC time: 12:00:00 on the 06/06/2014 in my app are as follows:

Mercury: RA = 6h 12.7m, DEC = 23° 10.2’

Venus: RA = 2h 31.7m, DEC = 12° 49.6’

Mars: RA = 12h 39.5m, DEC = -4° 4.1’

Jupiter: RA = 7h 32.0m, DEC = 22° 5.6’

Saturn: RA = 15h 4.5m, DEC = -14° 49.2’

Uranus: RA = 0h 58.1m, DEC = 5° 29.4’

Neptune: RA = 22h 37.4m, DEC = -9° 27.7’

Pluto: RA = 18h 54.3m, DEC = -20° 11.8’

Sun: RA = 4h 57.1m, DEC = 22° 39.1’

The actual positions calculated by NASA are as follows:

Mercury: RA = 6h 12.73m, DEC = 23° 10.49’

Venus: RA = 2h 31.51m, DEC = 12° 48.51’

Mars: RA = 12h 39.39m, DEC = -4° 3.72’

Jupiter: RA = 7h 32.12m, DEC = 22° 5.47’

Saturn: RA = 15h 4.67m, DEC = -14° 49.73’

Uranus: RA = 0h 58.08, DEC = 5°29.37’

Neptune: RA = 22h 37.43, DEC = -9°27.63’

Pluto: RA = 18h 54.34m, DEC = -20° 11.83’

Sun: RA = 4h 56.94m, DEC = 22° 38.70’

So the differences are:

Mercury: RA = 0.03m, DEC = 0.29’

Venus: RA = 0.19m, DEC = 0.09’

Mars: RA = 0.11m. DEC = 0.38’

Jupiter: RA = 0.12m, DEC = 0.13’

Saturn: RA = 0.17m, DEC = 0.54’

Uranus: RA = 0.02m, DEC = 0.03’

Neptune: RA = 0.03m, DEC = 0.07’

Pluto: RA = 0.07m, DEC = 0.03’

Sun: RA = 0.16m, DEC = 0.4’

As you can see, for all planets the right ascension lies within 0.2 minutes of the correct value and the declination lies within 0.6 minutes. For most purposes this is far beyond the accuracy needed, however as I will discuss later, the discrepancy in values for the Sun can have an impact.

6.3.3 Moon

The position of the Moon for UTC time: 07:17:05 on the 11/06/2014 and position 51 N, 0 E in my app is as follows:

Right Ascension: 278.58 Declination: -19.10

The actual position of the Moon is:

Right Ascension: 278.55 Declination: -19.052

Therefore, the discrepancy in the positions are:

Right Ascension: 0.03 Declination: 0.048

As you can can see, like the planets these values are very nearly the same but there is some small discrepancy. For most purposes this is fine but as the next section shows, this small discrepancy can have an effect.

6.3.4 Eclipses and Transits

Recreating the eclipse is particularly tricky since the position calculations for the Moon and the Sun have to be very accurate for them to line up correctly. In figure 50 I have shown a real total eclipse next to a recreated eclipse in my app to show the difference.

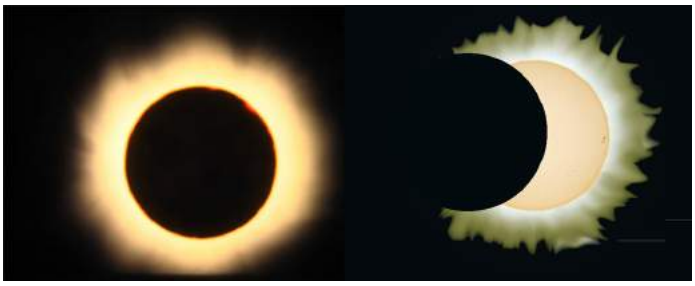


Figure 50: Real eclipse vs Galitarium

Note that in the real eclipse the Moon totally covers the Sun due to the apparent diameter of both objects being the same. In my app, the apparent diameter of each planet is the same, however due to the slight discrepancy in the calculations of both objects they do not completely align with one another. Despite this, I am very happy with the outcome as recreating the near total eclipse in my app requires a very high degree of accuracy in itself.

6.4 Graphical Accuracy

While the accuracy for the positions of objects in the night sky is important, it is also important to present a recreation of the night sky which can engage the user by looking very close to the actual thing. I will evaluate this by comparing objects in my app vs. their real life counterparts.

Stars

Figure 51 shows a comparison of a real image of the night sky with my app.

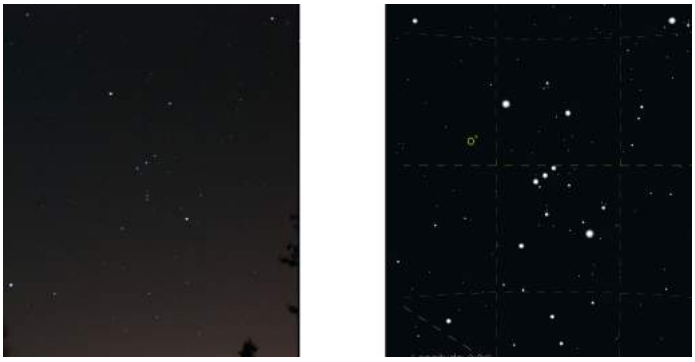


Figure 51: Real Night Sky vs Galitarium

I think the overall representation of the night sky in Galitarium is successful. It manages to strike the balance between having too many stars and being unrealistic and having too little and the sky seeming bare. Moreover, I feel that the relative brightness of each star seems very realistic and I have mostly managed to blur the edges of stars to give a specular feel. I would have liked to have more specular light coming off of each star but with the time I had I could not find a better way to present this.

Mercury

Figure 52 shows a comparison of a real image of Mercury with Mercury on my app.

When creating image textures for both Mercury and Venus in phase I had to alter and adjust a number of images as it was difficult to find many clear pictures for each phase. Given this fact, I am very happy with the result and I think Mercury looks particularly visually appealing.

Figure 52: Real Mercury with Phase vs Galitarium

Moon

Figure 53 shows a comparison of a real image of the Moon with phase with the Moon in Galitarium.

Figure 53: Real Moon with Phase vs Galitarium

The image on the left is the moon in its Waxing Gibbous phase and the image on the right is its representation in Galitarium. Given that the colour of the shadow of the Moon had to be adjusted to match the colour of the night sky in Galitarium and that the texture is being applied to a circle in 3D space I am very happy with the result.

6.5 User feedback

The final method used to evaluate the success of the app is user feedback.

6.5.1 Distribution

I distributed my app to users in two different ways. I either got them to use the app on my device there and then or I distributed it online to download on their own device. I made sure that initially I would not give them any guidance in how to navigate or use

the app. The reason for this is that I wanted to test that the user interface was intuitive enough that the app could be used without any guidance.

6.5.2 Common Issues

Working in this way raised couple of points. Firstly, getting to the navigation drawer was not intuitive and nearly all the test users did not realise that the menu existed. This was important as the vast majority of the features were accessed from this menu and by not realising it was there, the user would have a lesser experience.

A similar problem was encountered in trying to select objects in the sky. Only by accidentally holding down the screen did a user bring up the pop up displaying information about a selected object. To counter these issues I made a dialog appear the first time the user ran the app that explained a couple of key points to get the most out of the app.

6.5.3 Consensus

Users of my app who did not have any experience in astronomy liked that my app is particularly accessible and even though it can perform a lot of functions, the interface is easy to navigate and the options are straight forward. One suggestion was that even though the app is very easy to use, an introductory demo explaining how to get the most out of the app (such as recreating eclipses and transits) might be useful. With more time, I would aim to implement this by adding in pointers on screen that could guide the user through a first time run. Users that were not familiar with geographical coordinates suggested that I could provide options to set the location to landmarks around the world such as Tokyo or the North Pole without having to look up the longitude and latitude.

Users of my app with experience in astronomy were impressed with the extra features specific to my app such as the Jovian Moons and the phases of the planets and generally agreed that they would prefer to use my app in order to access these features.

6.6 Difficulties

6.6.1 Device Support

The most significant issue encountered while developing my app came with supporting a variety of devices. I chose the Android platform due to its large market share and its ease in distribution but when I neared the final weeks of the project and I began to test my app on other devices, I came across numerous compatibility and performance issues.

Some of the issues encountered were small and were due to minor differences in devices such as screen resolution and physical key placement. Before working with multiple devices a large part of the design of my UI was based in pixel offsets and although this worked fine for the test device, on a different device at a different resolution this would put everything out of place. I solved this by working in scaled pixels that change depending on the dimensions of the device in use.

Some of the parts of the Android API used in my app were only supported on devices above certain firmware thresholds such as the navigation drawer. This meant that I had to cut off some devices from supporting my app which is not what I had initially intended. The app is only able to be installed by devices running Android API 14 or above - that is Android 4.0.0.

The problem was that having an Android device with an API above this threshold was necessary but not sufficient to run my app. Android devices in their nature come from a large number of manufacturers, all with their own components and alterations in software. This made determining whether my app would run on the device difficult and determining where a fault lay was even trickier.

One such device in my development was the Samsung Galaxy Fame running Android 4.1.2. According to the API's outlined by Google, theoretically this device should be supported. However, when it came to testing on this device I found numerous compatibility issues.

The first of these was due to the graphics component in the device. My app is built in OpenGl ES2 and the device did support this version so upon running the app there seemed to be no errors. The issue was that only a subset of all the objects in the OpenGl environment were being displayed, with some of the objects also being displayed incorrectly. After investigating the issue I found that the number of texture units in the device was only 8 compared with the 32 in the test device. I wrote my code under the assumption that I would have 32 units available to me and so I used all of them. The only way to combat this issue would be to load different textures into the 8 available units every frame as opposed to at the beginning. This would require a significant overhaul in how my program ran - something I did not have the time to do.

A second issue encountered came with the accelerometer in the device. The augmented reality function works by using both the accelerometer and the magnetic field measurer in the device to work out where north is and rotate the screen accordingly. The Galaxy Fame only had the accelerometer without the magnetic field and to support this phone I would need to investigate an alternative method of determining north - and again this is something I did not have the time to do.

6.6.2 Time

One of the most confusing parts of developing my app was the use of different measure of time. As I have previously discussed, most of the calculations for the positions of objects in the night sky are based on UTC. For working out the azimuth and altitude of an object for a particular location and time, things became very confusing.

Firstly, I had to decide how user input for a given time would work. I decided that from a user perspective, it made more sense to have the input in local time as opposed to UTC. For example if the user is trying to recreate the sun setting, they would be much more likely to know the time that this occurs in local time not UTC.

I decided to keep track of both UTC time and local time separately so that when a user selects a local time, the UTC time would be adjusted accordingly depending on the time zone difference. Calculations for positions could then use this adjusted UTC time.

One confusion that does arise from this method is when locations are used in combination with a time. For example, if a user in London selects a longitude and latitude that correspond to Los Angeles which is 7 hours behind UTC, and then selects a time of 12:00, are they trying to select the time local to Los Angeles as 12:00 (UTC 20:00) or the time local to their position (UTC 13:00 in BST)? I feel that by making my app work in the latter way it is more consistent as the time always corresponds to the timezone of the user's phone.

6.7 Limitations

Device Performance

As discussed previously, supporting a wide variety of devices has been one of the most challenging parts of developing my app. Due to the lack of devices available to me for testing, I can only say with any real certainty that the app runs correctly on the test device, a Nexus 4 running 4.4.2 and a Samsung Galaxy S4. I have tried to design my code in such a way that other devices are supported, for example by implementing the loading of stars in the background, my app should run more smoothly on older devices. But given the time constraints I have had, it has only been possible to completely support a few devices.

Accuracy

As presented previously, although positions and sizes in my app are fairly accurate there is a small discrepancy that becomes more apparent in certain cases. At small amounts of zoom this will be completely unnoticeable but once the user reaches very high zoom levels the small difference in the position of objects in my app compared with the real world becomes magnified.

Connectivity and Power compromises

When researching existing ways of viewing the night sky online I was very impressed by Sky Survey. Being a web application it utilise the power of potentially very fast internet connections and fast devices to show a hugely detailed, high resolution version of the night sky. Choosing to develop my app for a mobile device meant I had to make compromises about the way the sky was presented in order for it to run smoothly. If I was not burdened with temperamental internet connections and slower graphics and processing units, I would have looked into downloading textures at run time in order to present a far more stunning night sky.

6.8 Further extensions

6.8.1 Extension of device support

Now that I provided a complete app to a subset of devices I would work on trying to support as many devices as possible. In order to do this it would be useful to distribute

my app on the Play Store and to use the feedback to add support for a greater number of devices.

6.8.2 Extension of Augmented Reality

Although I feel the augmented reality feature currently implemented in my app runs well I feel it has the opportunity to be extended. Currently, the user tilts the device in a given direction and can match up what they see on the device with what is in the night sky. As is, the user is limited by the representation on the device and it can be hard to actually match up individual stars and planets.

A really exciting extension, and one which as far as I can tell has not been implemented, would be to use the camera of the device to display the actual night sky on the device screen with an overlay with markers presenting the actual locations of objects in the sky. Providing such a feature would require a huge amount of image analysis but I feel it would provide a unique and immersive way to view the night sky.

6.8.3 Inclusion of other objects

What is unique to my app is that I include the Jovian moons in my representation of the night sky. If I had more time I would like to set apart my app even further by including more objects. Such examples could be the moons of other planets, comets, the asteroid belt and distant galaxies.

7 Bibiology graphy

References

[1] Google Inc, Google Sky Map App, 05/01/2014 https://play.google.com/store/apps/details?id=com.google.android.stardroid&hl=en_GB

[2] Escapist Games Limited, Star Chart App, 06/01/2014 https://play.google.com/store/apps/details?id=com.escapistgames.starchart&hl=en_GB

[3] Siyavula Uploaders, OpenStax CNX, Longitude and Latitude Image, 03/03/2014 http://cnx.org/content/m22756/latest/Picture%2010.png

[4] Wikipedia, Right Ascension and Declination Image, 03/03/2014 http://upload.wikimedia.org/wikipedia/commons/9/98/Ra_and_dec_on_celestial_sphere.png

[5] Wikipedia, Azimuth and Altitude Image, 03/03/2014 http://upload.wikimedia.org/wikipedia/commons/9/98/Ra_and_dec_on_celestial_sphere.png

[6] R. Nermiroff J. Bonnel, Planets of the Solar System Image, 27/05/2014 http://apod.nasa.gov/apod/image/0608/planets_iau_big.jpg

[7] J. B. Kaler, Celestial Sphere Image, 26/05/2014 http://stars.astro.illinois.edu/cel-sph.jpg

[8] P. Schlyter, Phase and Magnitude Calculations, 08/03/2014 http://www.stjarnhimlen.se/comp/ppcomp.html

[9] D. Nash, HYG Star Database, 05/01/2014 http://www.astronexus.com/node/34

[10] S. R. Schmitt, Planet position calculation using mean orbital elements, 02/01/2014 http://mysite.verizon.net/res148h4j/javascript/script_planet_orbits.html#thesourcecode

[11] Google Inc, Android Mobile Operating System, 28/12/2013 http://www.android.com/

[12] Khronos Group, OpenGL ES2, 28/12/2013 http://www.khronos.org/opengles/2_X/

[13] D. R. Hipp, SQLite, 28/12/2013 http://www.sqlite.org/

[14] B. Planche, Pitch, Yaw, Roll Image, 08/05/2014 http://aldream.net/img/blog/articles/cam-yaw-pitch-roll.png

[15] R. Nermiroff J. Bonnel, NASA Image of Sky, 09/05/2014 http://apod.nasa.gov/apod/image/1011/pleiadesSky_JOhn.jpg

[16] Wikipedia, Spherical Coordinates Image, 15/05/2014 http://upload.wikimedia.org/wikipedia/commons/thumb/4/4f/3D_Spherical.svg/500px-3D_Spherical.svg.png

[17] Unknown ImageShack User, Text Vertices Image, 16/05/2014 http://img690.imageshack.us/img690/2475/textextrusion.png

[18] Unknown Blogspot User, Ray Picking Image, 16/05/2014 http://2.bp.blogspot.com/-65I7JP_CkgA/UYVZZEPlgmI/AAAAAAAABtg/GtJE_GWQgqM/s1600/mouse_picking_1.jpg

[19] Wikipedia, Orbital Elements Image, 17/05/2014 http://upload.wikimedia.org/wikipedia/commons/thumb/e/eb/Orbit1.svg/500px-Orbit1.svg.png

[20] Wikipedia, Eccentricity of orbits Image, 20/05/2014 http://upload.wikimedia.org/wikipedia/commons/thumb/b/b7/Kepler_orbits.svg/500px-Kepler_orbits.svg.png

[21] Wikipedia, Semi-major axis Image, 19/05/2014 http://en.wikipedia.org/wiki/File:An_image_describing_the_semi-major_and_semi-minor_axis_of_eclipse.png

[22] Wikipedia, Ecliptic Image, 20/05/2014 http://en.wikipedia.org/wiki/File:Earths_orbit_and_ecliptic.PNG

[23] TurboSquid, Sphere Grid Image, 23/05/2014 http://previewcf.turbosquid.com/Preview/Content_2009_07_15__16_24_53/sphere_grid0000.jpgb63cf695-435d-46bf-82c8-c3cb94b64ba0Larger.jpg

[24] GNU GPL, Stellarium, 25/05/2014 http://www.stellarium.org/en_GB/

[25] N. Risinger, Sky Survey, 28/05/2014 http://media.skysurvey.org/interactive360/index.html

[26] K.Burnett, Algorithm for calculation of Jovian Moons, 29/05/2014 http://www.stargazing.net/kepler/galileo.html

[27] J. Gillfelt, Android SQLlite asset helper library, 30/05/2014 https://github.com/jgilfelt/android-sqlite-asset-helper

[28] M. McInterney, Raleway Font, 20/03/2014 https://www.theleagueofmoveabletype.com/raleway

[29] M. A. Earl, Anomalies of Orbit Image, 17/05/2014 http://www.castor2.ca/03_Mechanics/01_Basics/03_Anomalies/index.html

[30] M. A. Earl, Perigee and Apogee Image, 17/05/2014 http://www.castor2.ca/03_Mechanics/01_Basics/01_Parameters/index.html#perigee

[31] R. Yu, Lunar Phases Image, 17/05/2014 http://journeytothestars.files.wordpress.com/2012/12/lunar-phases-1-copy.jpg

[32] K. Burnett, Moon Position Algorithm, 25/03/2014 http://www.stargazing.net/kepler/moon.html