School of GeoSciences

Dissertation
For the degree of

MSc in Geographical Information Science

Callum Scougal

August 2014

Is a Puffersphere an effective medium for visualising real-time global earthquake events?

"I declare that this dissertation represents my own work, and that where the work of others has been used it has been duly accredited. I further declare that the length of the components of this dissertation is 4976 words for the Research Paper and 5090 words for the Technical Report. "

Acknowledgements

Many thanks to Jay and the team at Puffersphere for their aid throughout the process, to Bruce for his help and guidance and to the staff and volunteers at RSGS in Perth.

PART I: ABSTRACT

Little research has been conducted on the effectiveness of real time mapping on three dimensional surfaces such as Pufferspheres. This research aims to assess the effectiveness of using a 3D sphere for visualising real-time geographical information by mapping live earthquakes across the globe for a weekly period. In order to conduct this research, a Python script was developed to automatically retrieve and plot United States Geological Survey data against a satellite base map. The resulting output was then processed using a selection of tools to create a Puffersphere compatible visualisation. The visualisation was subjected to a user trial at the Royal Scottish Geographical Society in Perth, in order to determine its effectiveness in displaying geographical data and developing understanding and interest in geographical topics like earthquakes among the public. The results strongly imply that the visualisation was very effective in visualising earthquakes and in helping members of the public learn about the dynamics and distribution of global earthquakes in a real-time context.

PART I: Research Paper

Introduction

The real-time mapping and monitoring of earthquakes has seen significant advances since its inception in the 1980's. Whilst significant research has been conducted on live mapping and visualisation of earthquakes on 2D surfaces, very little work has been conducted on the effectiveness of real-time mapping on 3D spherical surfaces like the Puffersphere (Wang et al, 2007). Puffersphere's and similar devices like Japan's Geo-Cosmos globe(Geo-Cosmos,2014) are essentially digital globes which allow 360 degree visualisations of geophysical and environmental datasets on a global scale. They are commonly used for education, commercial and scientific applications as they are a modern and attractive way of conveying global data (Riedl, 2009). With the rise of accessible data and the continual demand for real-time data in society (Gellar, 2007), there is potential that an implementation of real-time earthquakes on a Puffersphere could greatly benefit scientific, educational and commercial communities.

Visualisations have long been used to aid in learning and to allow easier understanding of complex issues (Larkin and Simon, 1987). Studies have shown that humans are primarily 'ocular centric, i.e. we predominantly learn via our eyes. Research indicates the majority of humans learn new information much faster and more coherently via visual aids and models (Lloyd, 1995). Whilst a person may be told that the distribution of global earthquakes is heavily focused around the pacific region, a map showing this distribution is far more beneficial, as it shows exactly where they are occurring and any patterns which may be apparent. Whilst we are able to read and listen to information, it is often not fully understood until a process or phenomena has been visualised. Studies have shown that the vast majority of people are able to process information much more efficiently by studying an image/visual cue rather than reading or listening to information (Ainsworth and Loizou, 2003). The true foundation of visualisation is the ability to depict a phenomena or process in a way which is easy to understand. It is paramount that the visualisation is accessible to all members of society but particularly those with limited knowledge of earthquakes. This research aims to assess whether a real-time earthquake visualisation on a Puffersphere is an effective medium for promoting learning and interest in geography.

Primary Research Question

Is a Puffersphere an effective medium for visualising real-time global earthquake events?

Research Objectives:

1) Automatically retrieve United States Geological Survey earthquake data as a CSV file.

2) Develop a system which can accurately and precisely map real-time occurrences of earthquake events upon a Puffersphere.

3) Ensure that the earthquake data effectively conveys the distribution, magnitude and frequency of real-time earthquake events.

4) Gain user feedback on the effectiveness of the visualisation

Literature Review

The ability to interact and model spatial data in three dimensions, allows users to view and create unique visualisations which would otherwise be impossible or ineffective via traditional 2D mapping. Whilst virtual globes such as Google Earth allow users to effectively interact and map data within three dimensions, they are not ideal for public displays or exhibitions. Traditional globes have been used for centuries, but the rise of 3D technology and data availability has helped create a new generation of digital globes known as hyper globes (Cartwright and Peterson, 2007).

The primary advantages of hyper globes include the ability to display real world spatial relationships without distortion, at a global scale (Riedl, 2007). The capacity to visualise geographical data upon a spherical surface allows effective visuals to be created, ideal for museum and educational exhibitions. Visuals aside, being able to interact with spatial data, via touch recognition, tablet controls or a mouse ensures a level of interaction ideal for public situations. Other advantages of digital globes include the ability to store and display multiple data themes as well as being able to share globe media across multiple units. The application of digital globes in displaying geographic data is renowned for having a multitude of advantages. Ainsworth and Loizou, 2003 report that studies using NASA's *Science on Sphere Project*, resulted in users having a far greater understanding of difficult scientific topics, such as plate tectonics, meteorology and climate change (Hruby et al, 2008). Further studies on the *SOS* project showed that by including animation and interactive capabilities, public interaction and involvement in learning about scientific topics was greatly increased (Tvsersky, 2001). Schratt and Riedl, 2005 state that spheres are the ideal media to view

dynamic global phenomena, far surpassing two dimensional and virtual 3D outputs for global geographical visualisations.

Science on a Sphere was created by the National Oceanic and Atmospheric Administration (NOAA, 2014) and is based upon visualising high resolution geographical data upon a suspended globe/sphere. Sphere based visualisations like the Globe 4D project and Science on a Sphere are extensively used within public exhibitions and museum environments (Companje, 2007). Each truly highlights how traditional globes have been replaced by digitally augmented spheres in order to create striking, interactive geographical visualisations. Other models such as *The Sphere* (Benko et al, 2008) and the *Viball* (Kettner et al, 2004) are focussed upon allowing users to interact with high resolution imagery and data as a means of increasing curiosity and learning within geographical environments.

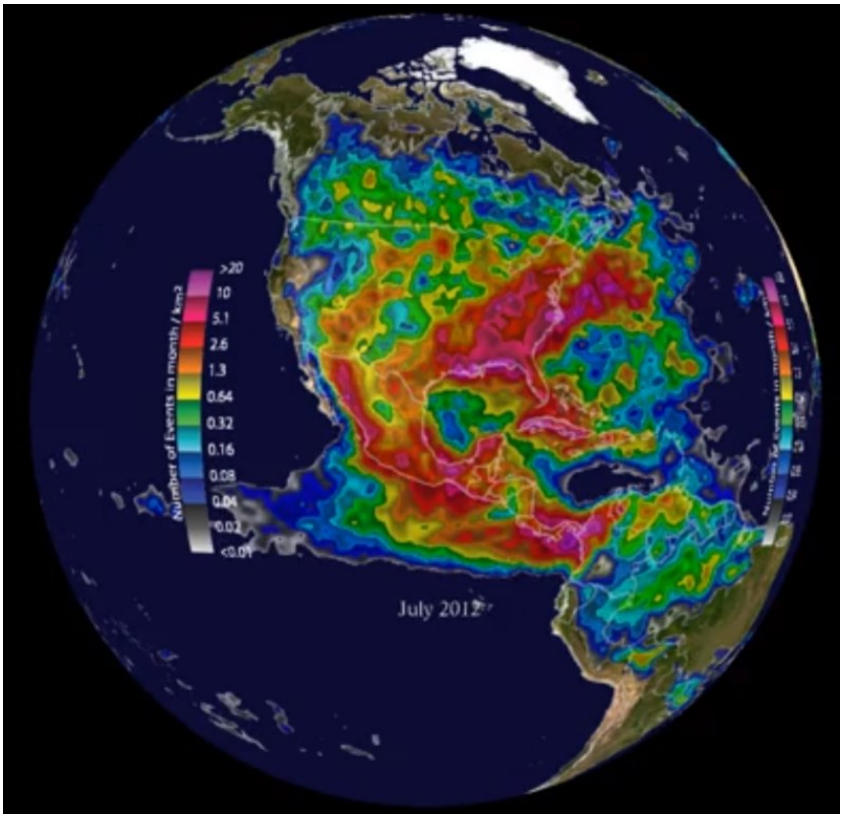

Figure 1: An example of a Science on a Sphere data set visualising thunderstorms in the USA.

The work of Goldman et al, (2010) strongly indicates that sphere based visualisations are well received in exhibition environments. However, minimal work has been conducted on whether there are specific advantages in using three-dimensional visualisations over two-dimensional ones. Despite this new wave of digital globes providing high resolution images of dynamic earth processes, there is a clear gap in the current research involving real-time displays of geographical phenomena upon three dimensional spheres. The closest sphere which incorporates real-time data is Japans Geo-*Cosmos* sphere. What sets this globe apart is its ability to use real-time scientific data from a range of sources, and the ability to stream live images and data directly from satellites. This ensures that the sphere can be used for near

real time representations of the earth's surface whilst also incorporating aspects such as real-time cloud movements taken from weather satellites as seen in Figure 2 (Geo-Cosmos, 2014).

Each sphere has the capacity to display real-time data, yet only the Geo-Cosmos has successfully done so. It has done this predominantly via satellite feeds and as such is subject to potential delays of imagery. This means that the data is not always real-time data, instead it is simply relaying data from footage captured hours before. This highlights one of the key barriers in real-time visualisations. The time a phenomena or aspect is recorded, entered into a database then made available for transfer undoubtedly takes time and as such the subsequent mapping of data could be viewed as simply being near real-time (Worboys, 1995).

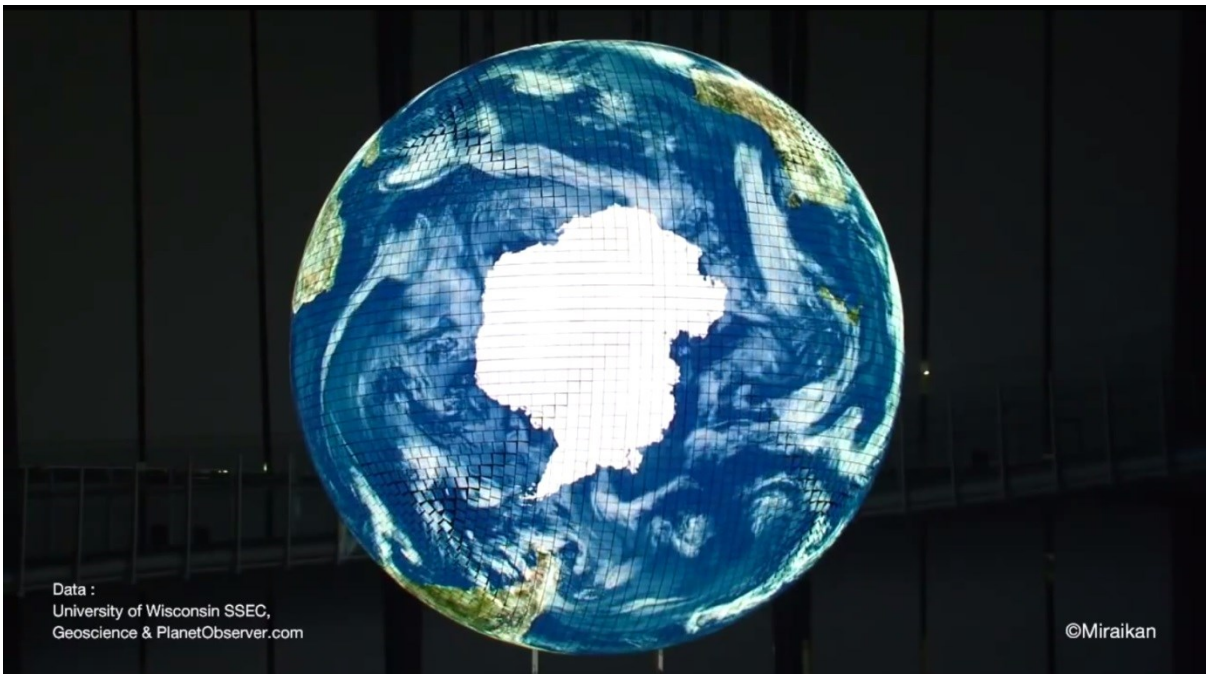

Figure 2: Real-time weather patterns across the world on Japan's Geo-Cosmos sphere.

For many common sphere applications such as desertification across the globe, real-time data is not a major issue as it is visualising a process which occurs over millions of years. For other applications it may be that displaying real-time images would not aid in the actual visual output desired. For example, forest cover change over a number of years would undoubtedly create a more striking visual than real time changes. This highlights one potential reason why little development has been conducted in regards to real time visualisations. The applications being developed and displayed within the current literature are not yet utilising real-time data. This author would argue that the recent shift in society towards expecting real-time information will lead to a new wave of three-dimensional real-time applications for hyper globes, as has been seen in the ever expanding two-dimensional field of mapping (Gellar, 2007).

There is a lack of studies within the literature which review the effectiveness of real-time dynamic mapping upon a three dimensional sphere. Yip et al (2006) review work on visualising internet traffic with the ViBall sphere in 2006 showing that visually appealing and educationally stimulating applications can be made easily upon static data sets. The lack of real time mapping studies highlights a clear gap in current research on dynamic spherical modelling. The paper aims to reduce this gap in the research by providing an effective three-dimensional visualisation of real-time earthquake events across the world using a Puffersphere.

The Puffersphere primarily caters to the commercial sector as well as the scientific community and as such have been used for a wide range of unique visualisations. Visualisations such as continental drift and global sea surface temperatures are displayed on spheres located within the National Museum of Scotland and the Royal Scottish Geographical Society as seen in Figure 3.

Figure 3: Sea-surface temperature visualisation at the Royal Scottish Geographical Society in Perth.

Methodology

To fulfil the primary research question, the specified research objectives must be fulfilled. The structure of the overall methodology is displayed below (Figure 4).

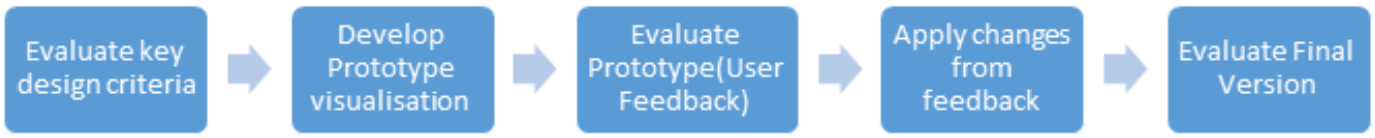

Figure 4: Research development stages

To display a visualisation upon the Puffersphere, a 2000x1000(pixel) image or video must be provided to the PufferWarp box within a Puffersphere unit. This box takes an equi-rectangular image and warps it to a projection suitable for a spherical device. This section briefly describes each of the core steps in creating a real-time map of global earthquake event's over a weekly period. Python and an array of modules, such as Basemap and Numpy were used in combination with ImageMagick and FFMPEG to develop this visualisation (Scougal, 2014).

The process of creating the visualisation can essentially be split into three separate sections:

1) Evaluation of key design criteria
2) The retrieval and plotting of earthquake data upon a satellite base image using Python
3) The rolling of images to ensure global rotation upon a sphere and the compression of these images into an mp4 video format via Python, FFMPEG and ImageMagick.

Each key stage of the methodology can be seen in Figure 4.

Evaluation of key design criteria

In order to create the most effective earthquake visualisation possible, major cartographic principles were evaluated (Scougal, 2014). These were based upon data classification, symbolisation and visual variables. Data classification relies upon the accurate symbolisation and portrayal of data upon a cartographic output (MacEachren, 2004). This visualisation aims to do this by appropriate use of colour, size, shape and tone for earthquake events. Quakes will be separated into three categories as a means of clearly symbolising low, medium and large seismic events. These bands will range from magnitudes 4-5, 5-6 and 6+. This delineation will help to create a clear and concise visual hierarchy allowing users to instantly determine patterns of seismic activity across the world .Circles will be used for symbolisation due to the ease at which they can visualise quantitative differences between varying magnitudes (Bertin, 1978).

This visualisation aims to use core visual variables like colour and shape to distinguish between the three earthquake bands effectively (Monmonier, 1996). Figure 5 below shows low magnitude events symbolised as small green circles and larger events as big red circles. In many earthquake visualisations such as the USGS's web map (USGS, 2014), colour denotes depth. This is not the case in this visualisation, as it was felt using both colour and size to portray quakes offered a simple and clearer way of illustrating the nature and distribution of earthquakes around the world.

Figure 5: Classifications and colours used for earthquakes.

One core concept of map design is "less is more" (Tufte, 1978). Whilst explanatory labels for countries, continents, oceans could be provided and would potentially help users become more geographically aware, it could also detract from the primary message of the visualisation. The major aim of this research is to effectively convey the strength and distribution of real time global earthquakes as a means of educating and encouraging interest in geography among members of the public. With this in mind, this development will experiment with various graphical and contextual options as a means of improving the visualisation. However, the final visualisation will only incorporate elements which actively contribute in fulfilling the primary goal of effectively conveying real time earthquakes upon a Puffersphere.

Retrieval and Plotting of USGS Earthquake data

Python was used to download weekly earthquake data from the USGS (Scougal, 2014). The downloaded CSV data was then parsed into explicit lists to hold latitude, longitude and magnitude data. This was achieved by using Python's inbuilt CSV parser (Scougal, 2014). Having successfully downloaded and parsed the data into a usable format, the next stage requires the plotting of data and warping of a satellite image.

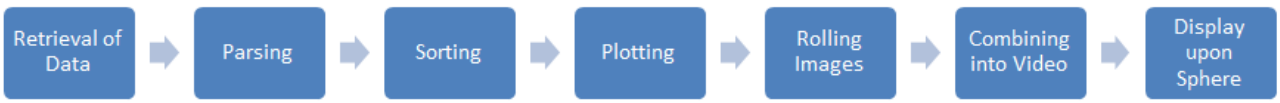

Figure 6: Diagram illustrating the overall methodological process

The matplotlib Basemap module is a toolkit/library which allows the plotting of 2D data on images/maps via Python. Its primary function is to allow the transformation of coordinates to

a number of different map projections. One very useful aspect of Basemap is the ability to display an external image as a map allowing easy plotting of locational data upon an equi-rectangular image. A series of functions were created to allow easy plotting of earthquake data with a variety of graphical options. Aspects such as varying symbol size and colour depending on magnitude size were implemented as was functionality for altering transparency. One of the major issues encountered in the early stages of development was the prevalence of large clusters of earthquakes around the pacific and western seaboard of the USA. This posed a major visualisation problem, as these areas were at the edges of the base map as seen in Figure 7. This problem meant that when the final image was being rolled to allow for sphere rotation, many earthquakes were being cut off and not being fully represented by the visualisation. This problem could be easily solved by moving the starting coordinates of the map, to an area with low seismic activity. Whilst this could alleviate the problem for one weeks' worth of earthquakes, there would be no guarantee that future quakes would follow the exact same pattern. As this entire script is based around automating the process of plotting and creating a video suitable for a Puffersphere display, it was clear that an alternative automatic solution was required.

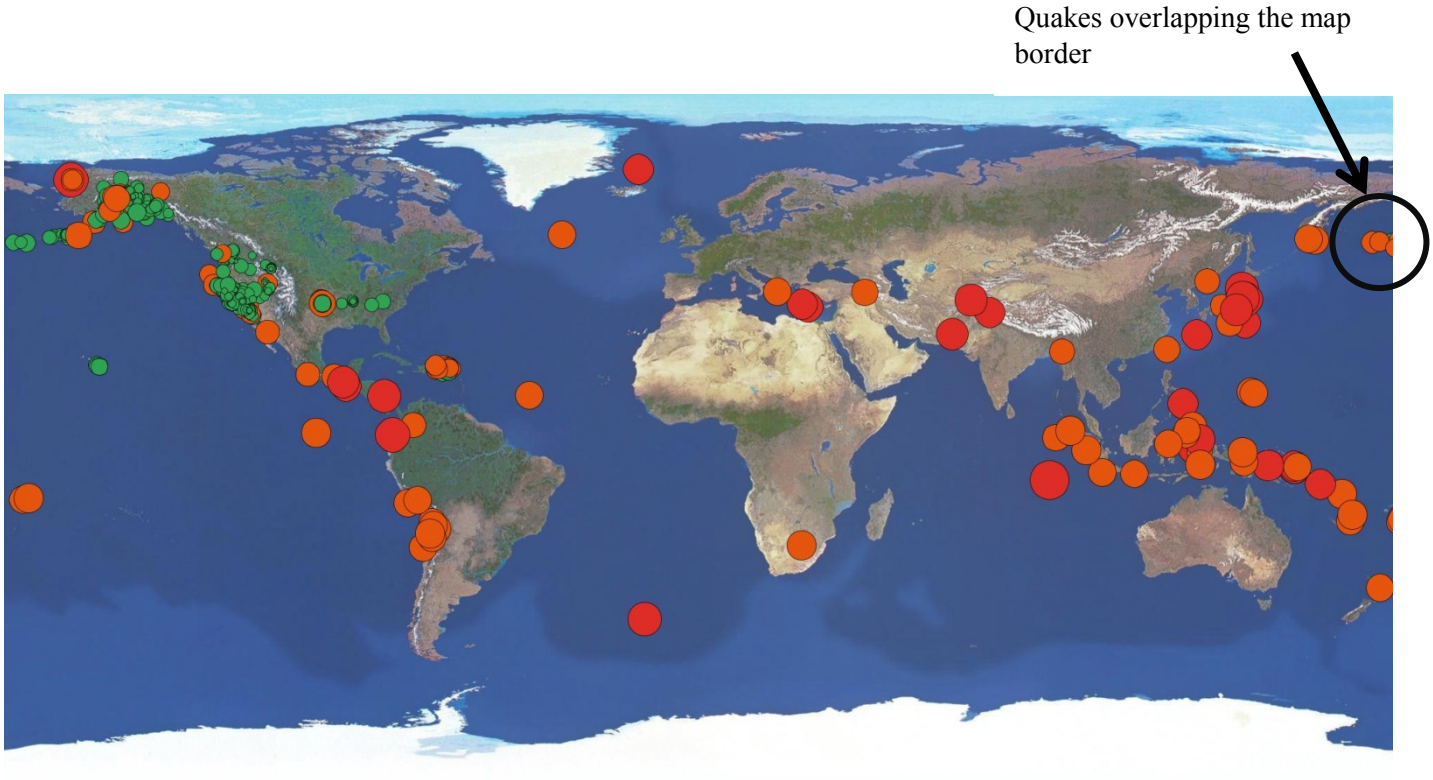

Figure 7: Sub-optimal global earthquake map with edge quakes being poorly represented.

In its default form, the starting longitudes of -180 to 180 degrees are clearly not suitable for the final visualisation. A specialised function was created to combat this problem. The function essentially finds the longitudinal difference between every earthquake within the data set. The largest longitudinal difference is then returned, along with the index positions of the two quakes. This difference is then divided by 2 to return the position which is furthest from the two earthquakes. This result is then used as the starting longitude of the map, and

the final edge longitude is simply calculated as 360 degrees plus the returned value. This function ensures that all earthquakes will be represented effectively for visualisation.

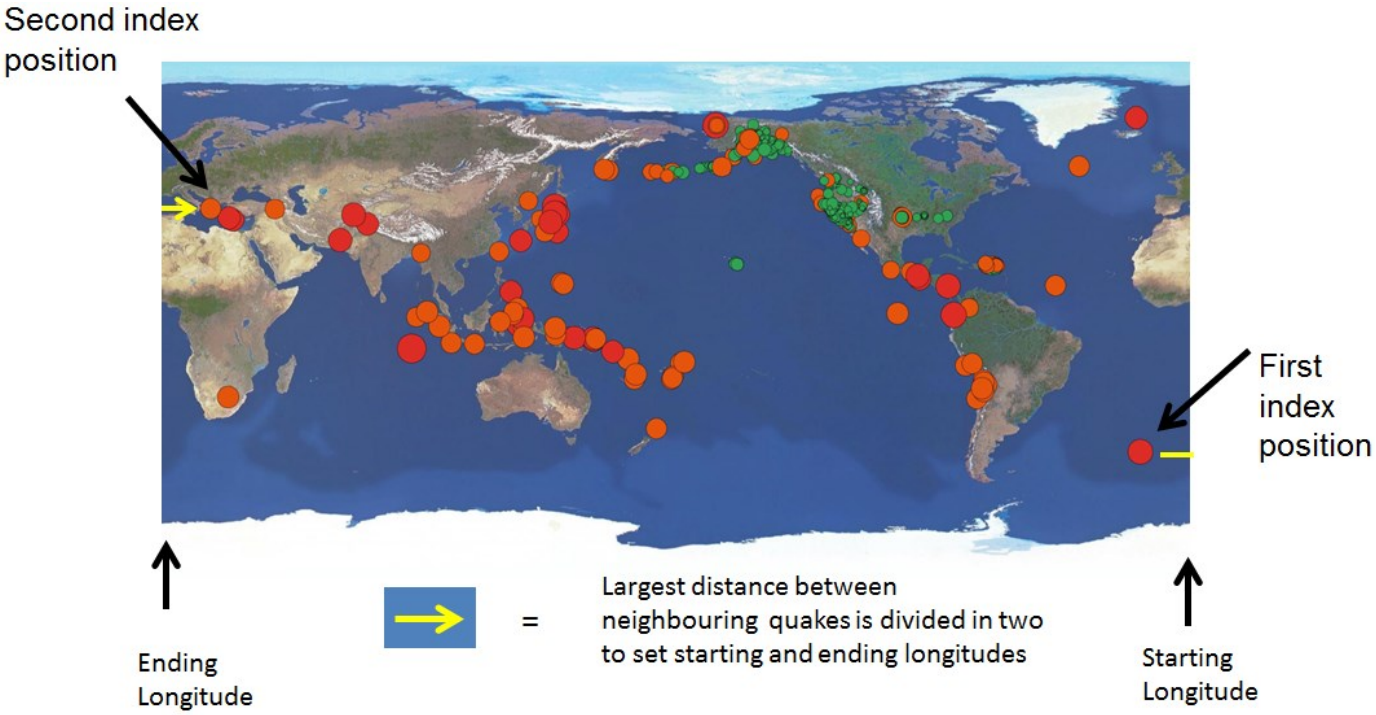

Figure 8: Earthquake data plotted with the developed longitudinal function.

It is clear from Figure 8, that no earthquakes are clipped at the edges and all are represented as expected. After extensive testing on monthly data sets, the function operates efficiently even with large numbers of earthquakes (tested with more than 8000 quakes). Unsurprisingly, the function nearly always sets an area of Europe as the starting and ending longitudes as Europe and Africa tend to have a low seismicity compared to the rest of the world.

Animation processing

To create a revolving animation, pixels from one side of the image are moved to the other side, to simulate a moving image via ImageMagick commands (ImageMagick, 2014). The resulting stacks of images created from this "rolling" process are then combined together using FFMPEG to provide a video of global earthquakes for the last week. Initial testing proved successful, however it was clear the visualisation lacked interactivity. The Python script was used to create separate images for all of the earthquakes occurring on each day of the week (Figures 9, 10 and 11). The animation and video functions developed were then run on each of the saved images to create a series of videos for the earthquakes occurring on each day of the week (Figures 9, 10, 11). These were then concatenated using FFMPEG to produce one final video of earthquake visualisations for a weekly period. This addition allowed the final output to have a level of dynamism instead of simply being a revolving base

map with a static data upon it. The script can be easily run for hourly, half day, or even monthly operations if required.

The figures below illustrate the process involved in creating animation for the visualisation.

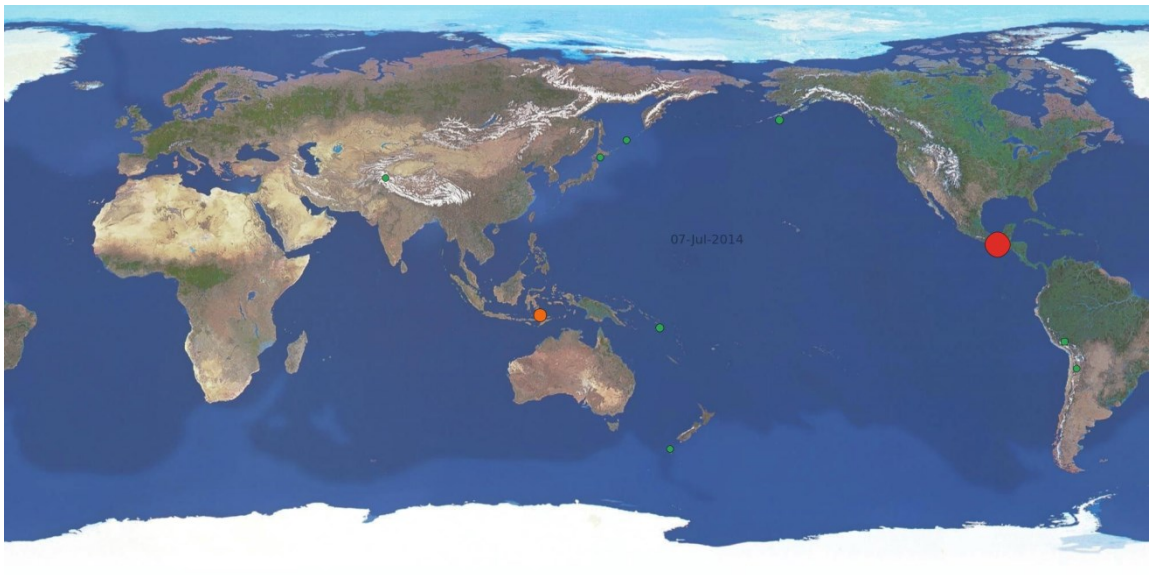

Figure 9: This image is showing earthquakes from the 6-7th July.

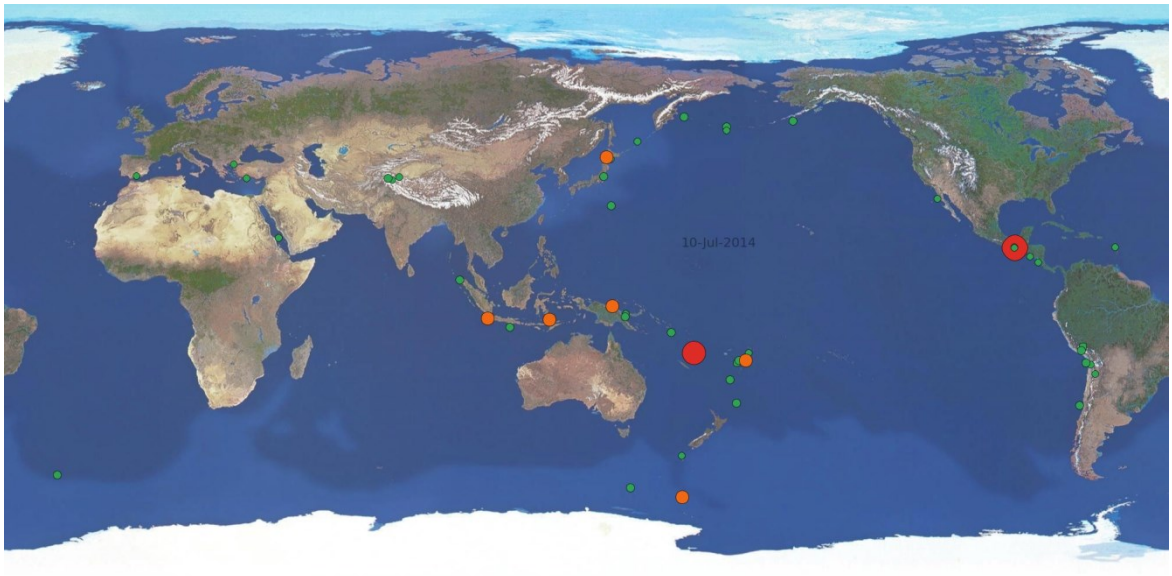

Figure 10: Earthquakes from the 6th- 10th July. Each image contains the earthquakes for that day and each of the previous days.

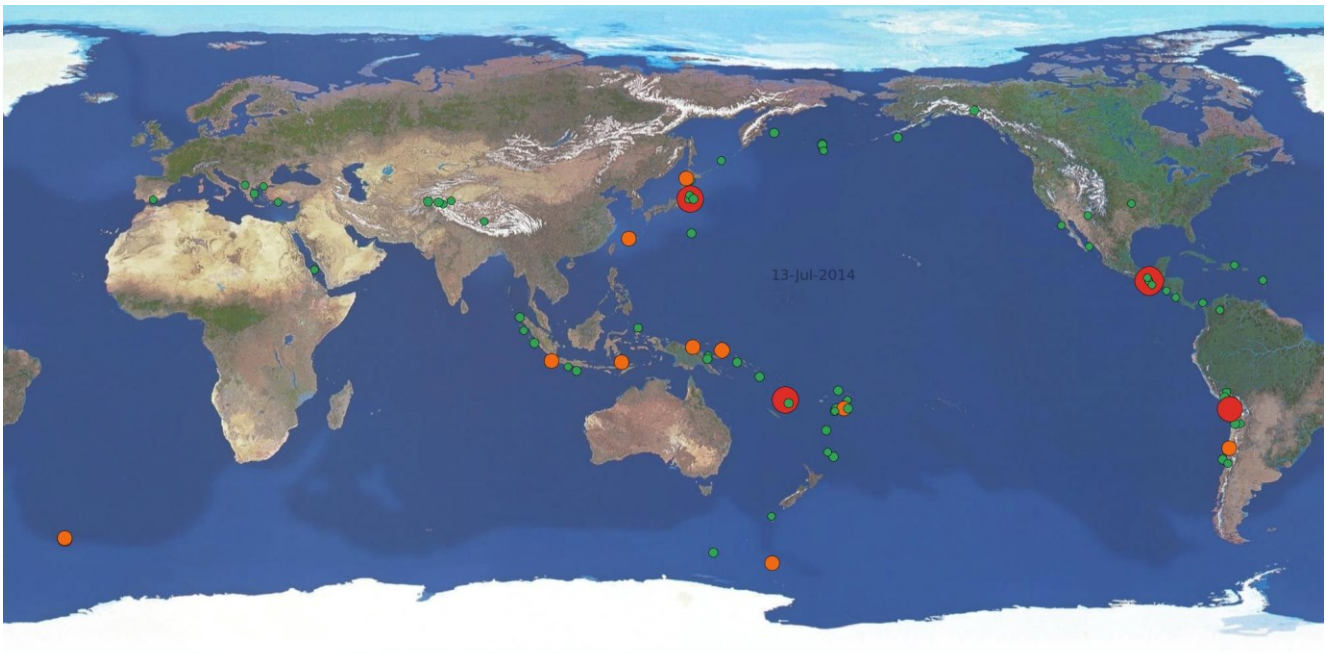

Figure 11: Earthquakes from the 6th-13th July.

It became apparent that the modularity of the script could allow for a variety of data and images to be used for alternative visualisations. With this in mind, an adapted script was created which allowed the use of user defined variables via a configuration text file. This file holds key information such as: location of CSV data, base image (WMS/local), save directories, colours, symbols etc. This ensures the program is accessible to those with no programming knowledge, and essentially allows any user to easily create a Puffersphere video in a fast and efficient manner. The script was used to test out a number of alternative visualisations with a wide variety of data, to ensure robustness. These visualisations are a small example of how easily the script can be used to develop a range of unique and automatic geographical visualisations (Figures 12 and 13).

Figure 12: Sample visualisation displaying the distribution of European Starlings across the globe upon a WMS layer.

Figure 13: Sample visualisation showing customisable symbols for cities across the world

Results

The development of an automated system for mapping real-time earthquakes upon a Puffersphere was successful in creation and operation. However, it is clear that the effectiveness of the visualisation must be validated. This was done by sampling two user groups and assessing the feedback received. The site chosen for showcasing the visualisation was the Royal Scottish Geographical Society in Perth. The earthquake visualisation was demonstrated to a group of RSGS volunteers and staff, who then filled out a survey regarding the effectiveness of the visualisation. The second user group tested were members of the public who visited the RSGS during a weekly showcasing of the visualisation. It is important to note that the earthquake visualisation was created as a means of encouraging interest and knowledge within geography. As such it can be expected that certain elements of feedback gained from each group may differ significantly, given that group one's personnel have typically worked in a variety of geographical professions such as National mapping library archivist, Ministry of Defence Cartographer and teaching roles.

This next section will detail the results gained from the visualisation surveys conducted at RSGS.

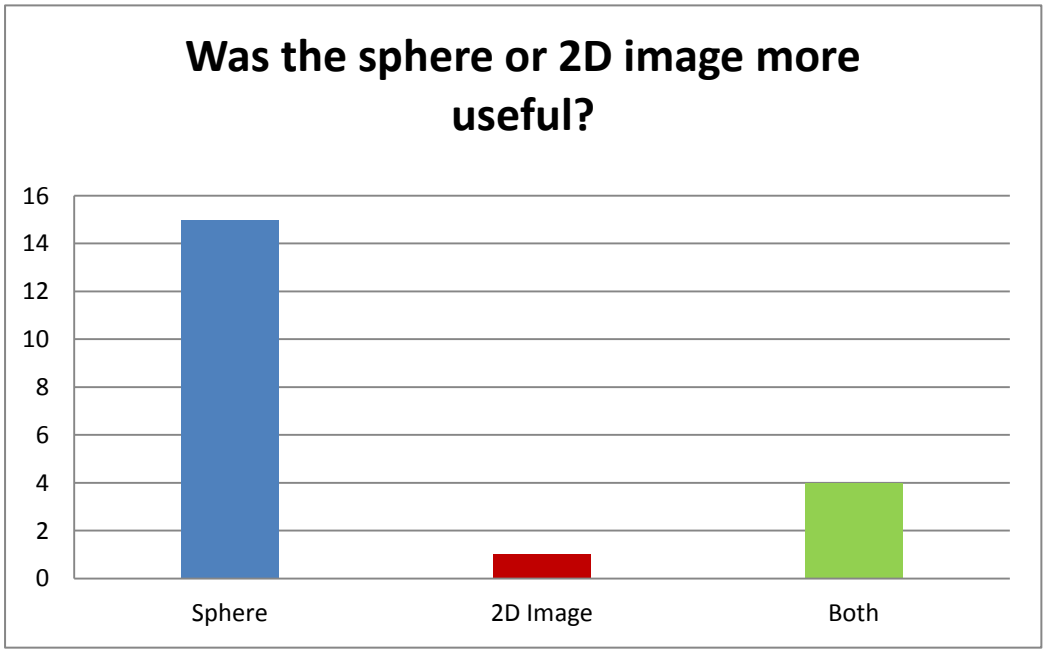

Figure 14: Aggregated results from the first question of the survey.

The vast majority of users felt that the sphere was far superior to the two dimensional image. One participant stated that he found the sphere to be useful as "eye candy" but found the 2D image more practical for studying the patterns of earthquakes. This sentiment was echoed by a few others who felt that they both offered different perspectives of the same issue.

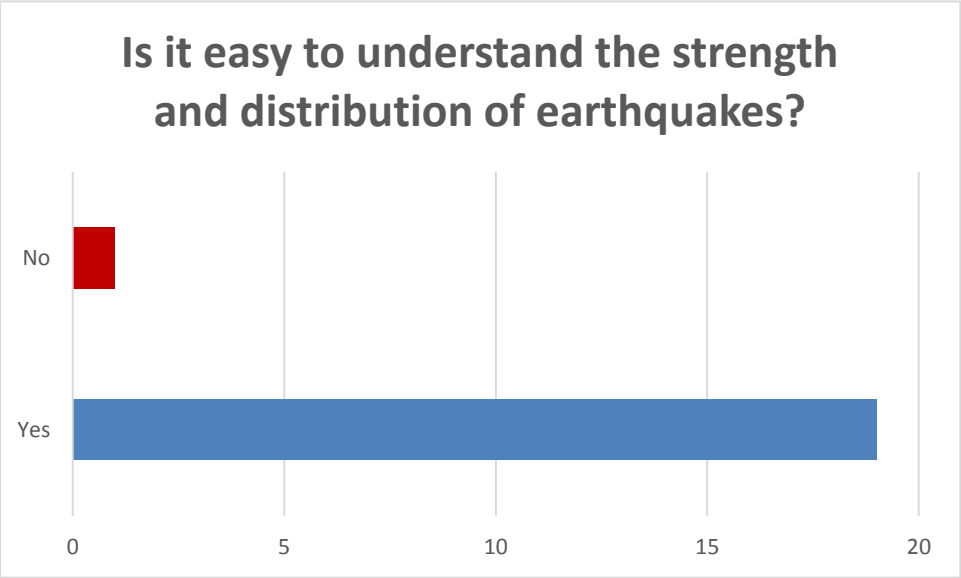

Figure 15: Results from the second survey question regarding the accessibility and ease in understanding the earthquake information displayed.

95% of users found the visualisation easy to interpret in terms of understanding the strength and distribution though many reported that it could be significantly improved. These issues and improvements can be seen in Tables 1 and 2.

Table 1: Summarised version of the issues encountered with the visualisation. They are ordered by frequency each issue was mentioned.

Problems/disadvantages of the visualisation	
1	Needs more contrasting colours
2	Needs flashing quakes / different colour for each quake on each day
3	Needs contextual information(plate tectonics map)
4	Revolving animation made it difficult to study particular areas in detail
5	Difficult to know what dots represent if not at start of visualisation
6	Date difficult to make out, not linked to quakes on each day

Table 2: Includes information about how users felt the visualisation could be improved.

How could this visualisation be improved?	
1	More intense colours
2	More highlighting/ flashing of new quakes
3	Needs tectonic plates map
4	Add in pauses to allow studying
5	Constant legend
6	Add audio commentary
7	Add ocean and continent names
8	Magnitude labels
9	High Definition
10	Touch screen interaction

The majority of users felt that greater use of colours and animation for quakes would help create a more visually pleasing and interactive visualisation. Another key element reported by users was the need for contextual information in the form of a tectonic plates map. Interestingly the second group had no problems with the colours used and instead were only interested in having more information relating to ocean and continental labelling.

Table 3: Results from the visual effectiveness question

Do you think this is an effective way of visualising global earthquakes	
Yes	No
20	0

The entire user group believed the overall visualisation was an effective way of visualising global earthquakes, albeit with room for improvements (Table 2).

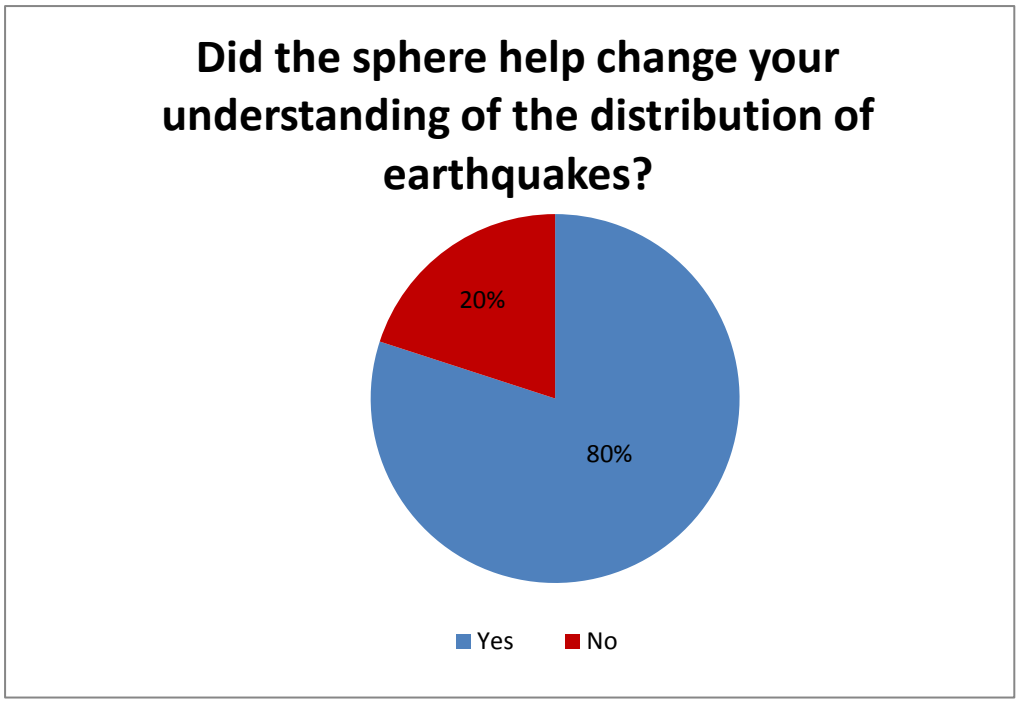

Figure 16 shows the overall result as to whether users found the visualisation to have informed or aided them in learning about the distribution of global earthquakes.

Over 80% found that the sphere did aid them in their understanding of quakes, whilst 20% found it did not. This is promising as it strongly suggests that the sphere is a useful tool for providing understanding and knowledge in geographical topics. The four participants, who found it did not help, reported that it did not aid them as they had previously studied or already knew of the patterns of earthquakes. One of these users stated that if he had no prior knowledge, he believed it would have been very useful.

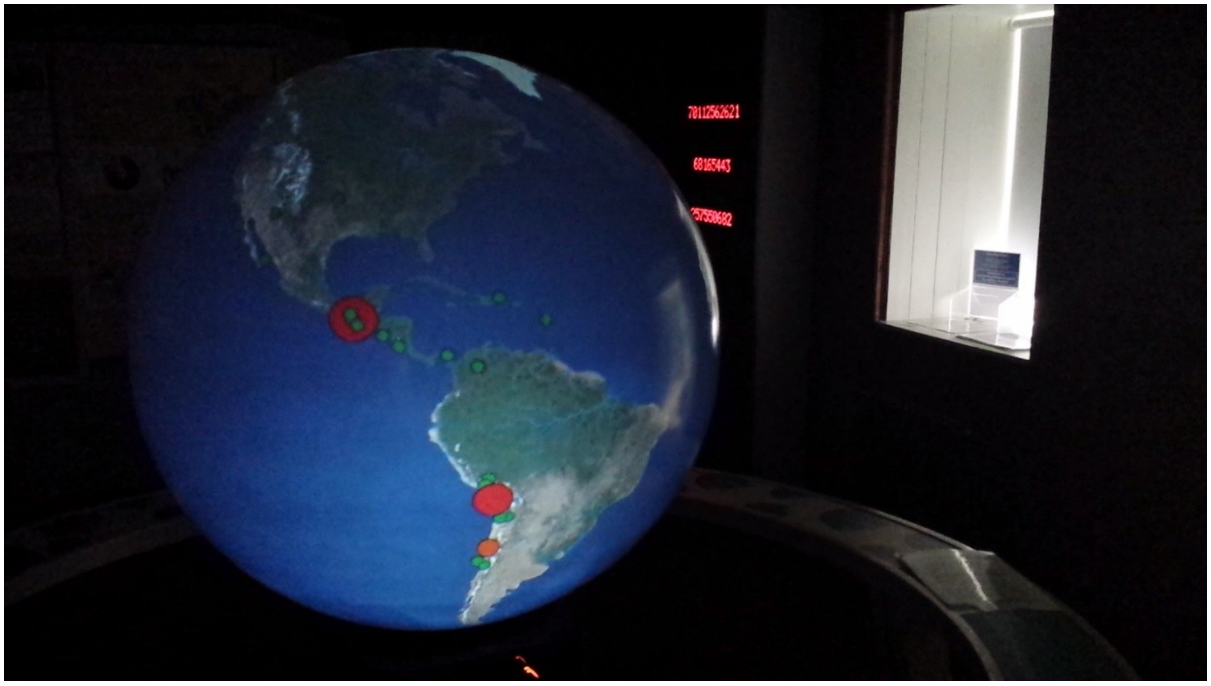

Figure 17 shows a snapshot of the prototype visualisation in action at the RSGS in Perth.

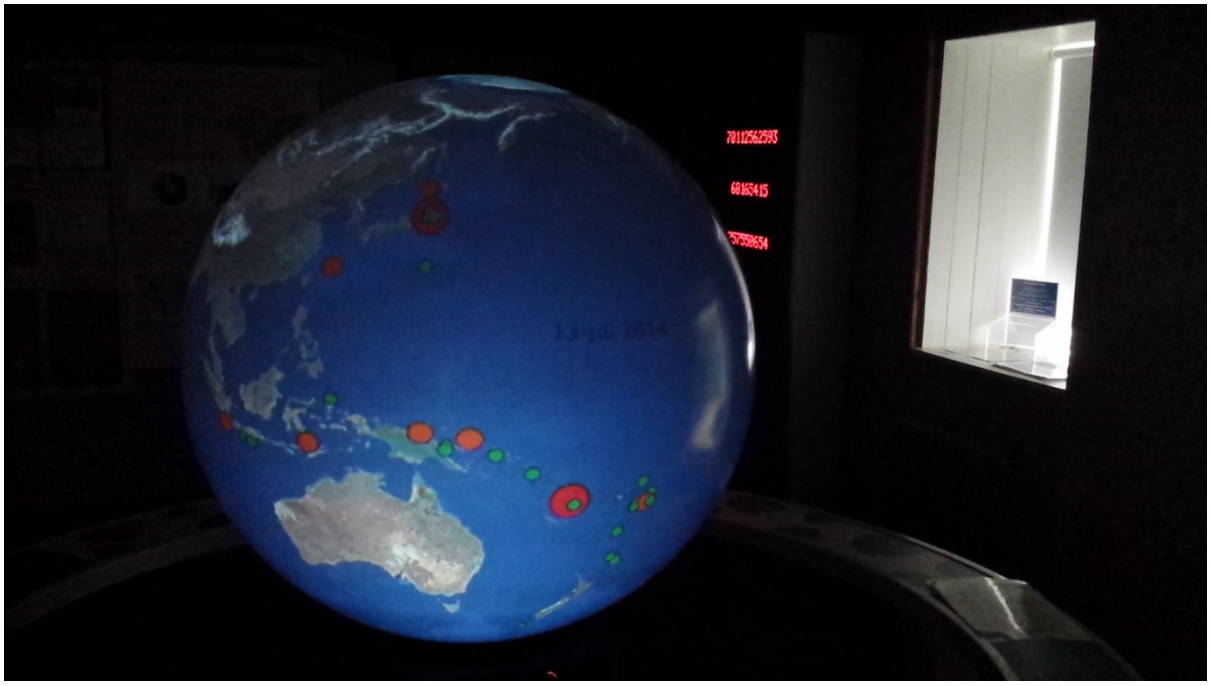

Figure 18 shows another angle of the visualisation.

Discussion

The results shown in Figure 14 clearly show the bulk of users found the three dimensional visualisation to be superior to its two dimensional counterpart. Users stated that the sphere was a useful and engaging way of displaying information and enjoyed the use of an innovative technology to aid in learning. One user echoed Korakakis view in that spheres are underused tools in education, and that with the correct balance of visuals and information they have the potential to be fantastic educational aids (Korakakis et al, 2008). Whilst most stated they preferred the sphere's visuals, considerable feedback was provided as to how improvements could lead to a greater visualisation.

Figure 15 shows 95% of users found that the sphere was easy to understand in terms of magnitude and distribution. However, they also stated that key cartographic principles such as colour and tone (Bertin, 1978) could be altered to improve the visualisation. It is clear from Tables 1 and 2 that users found the colours were not diverse or vibrant enough and that smaller magnitude events were too difficult to differentiate with the green terrain of the world. Another element not previously considered was support for those who are colour blind. Future versions of the visualisation will be adapted to ensure vibrant and distinct colours suitable for all (Figures 19, 20). Users found that whilst each rotation of the sphere led to a new days' worth of data, there was no way of knowing which specific earthquake occurred on each day. Suggestions ranged from colour coding each individual day, using flashing symbols for each new event or having a coloured band linking each new quake to the date marker. Whilst each suggestion has its merits, the logistics of implementing flashing symbols is out with the scope of this project. One fix, would be to simply code each days

earthquakes in different colours. Whilst this could be easily achieved, it would most likely lead to a confused juxtaposition of colours detracting from the visualisation overall. The final suggestion of implementing a colour band for each day's quakes is technically possible and would allow the current colour scheme of three major colours to be used. This is imperative as the visualisation needs to be simple and easy to understand in order to appeal to everyone (Crampton, 2009). Having three core colours and altering radii for each quake provides a visually effective method of symbolising varying quake magnitudes.

One persistent view was the lack of context toward the tectonic plates. This visualisation aims to be a simple representation as to where earthquakes are occurring in the world and what size they are, yet the introduction of a simplified plates map would greatly boost the learning potential of the visualisation as it allows users to know why earthquakes are occurring in specific regions, like the *Pacific Ring of Fire*. Following on from this, was the belief that the animation should incorporate pauses or be allowed to stop in order to allow inspection upon areas of interest. Three users backed the idea of not having a spinning visualisation and simply having a static video which would allow users to walk around and view each particular zone of interest. Whilst the animation could easily be stopped or paused for exhibitions via a remote device, the feedback from the general public suggested having a static video would most likely detract interest from members of the public. This is not surprising as movement and animation is a key driver in creating curiosity in visualisations (Goldman et al, 2010). A compromise may be that a static representation is shown directly after each day, allowing users to stop and study specific areas at their leisure.

One recommendation implied that extreme earthquakes should be represented better due to the significantly greater risk they pose to people. Whilst a 6+ quake is a significant event, a 7+ is exponentially more powerful and therefore has a much greater risk. With this in mind a new classification band was added to the latest visualisation to allow extreme quakes to be effectively differentiated from large quakes (Figure 19 and 20). Whilst the introductory part of the video contains a legend, users felt that if someone came in mid visualisation they would be confused as to what they are actually viewing. An easy solution to this would be to incorporate an inbuilt legend throughout the visualisation. Whilst simple, it does pose a slight risk in terms of clarity, as the position of a legend could potentially conflict with actively occurring earthquakes. For this reason it would need to be placed in an area of low seismic activity but easily seen by users.

One user suggested a detailed commentary could be provided via headphones, which would guide a user through the process. Whilst a great idea, it may be difficult to implement in the sense that each new user would need a different audio track depending on when they started viewing. Future work could definitely look into incorporating audio into the visualisation, as the use of visual imagery, animation and audio are all key factors in influencing someone's cognitive abilities and interest (Loftin,2003).

The overall view received from this feedback was that a greater differentiation between events was needed, greater linkages to time and more complexity in the form of a tectonic plate map. Whilst Tables 1 and 2 reported the various ways in which this visualisation could be improved, it is important to note that nearly all of these suggestions came from the RSGS user group. The majority of the user's from this group have been based within teaching or geographical professions. Whilst their feedback is very useful and informative in improving

the overall visualisation, it is imperative to note that their beliefs did not echo the opinions of the wider public. The second user test conducted on members of the public at RSGS found that all of the users found the visualisation to be effective and that they felt it could only be improved with greater information in regards to labelling of oceans, continents and magnitudes.

These results are very interesting in that both groups share similar responses to the key questions of sphere effectiveness (Figure 14), understanding of magnitude and distribution (Figure 15), yet they have very different responses to the problems and improvement aspects of the survey. This is very good as it shows that members of the public found the sphere to be a finished article which helped them to learn and understand more about the dynamics behind global earthquakes (Figure 16). Whilst the public users were generally much happier and content, it is vital not to disregard the views from the RSGS group. These views are very important and the majority of the suggestions had strong arguments behind them key to improving the final output. The core suggestions were to implement:

1) Greater colour differentiation between quakes
2) Greater linkages between each days quakes
3) Larger magnitude range
4) Inbuilt legend
5) Tectonic plates base map

These key suggestions were then implemented in the final version of the visualisation, as seen in Figure 19 and 20.

Figure 19: Final visualisation with suggested adaptations

Figure 20: Alternative view of final visualisation

Limitations

A number of limitations were found during both the development and the evaluation stages. The primary limitation in terms of development was the conflict between video quality and run time. It was clear that in order to get a smooth rotating globe animation, a high number of images/frames is required, ideally 3400 images. Large numbers of frames/images increases the run time of the video significantly. This is a problem as many people may not wish to view a visualisation which lasts longer than five minutes. Code was developed which allowed video output to be sped up, this was ideal for reducing run time, but also resulted in the play back speed being too fast when displayed on a sphere. In the end a balance was found between quality, time and speed but it is clear that further testing and a greater understanding of FFMPEG settings may well result in a higher quality output. This limitation is of particular significance in terms of the user friendly script developed, as without careful consideration of key settings, the output video created could be less than ideal. Future work would most certainly be based upon understanding the balance between these core settings better in order to optimise the final output. The primary limitation during the evaluation stages was the number of participants involved within the survey. Whilst a number of people did participate, the overall results would have much greater significance if the user group was larger.

Future work

This work could be easily expanded in a number of directions. The most beneficial developments would undoubtedly be found in utilising touch screen /tablet technology to allow users direct interaction with data upon the Puffersphere. Ideally a user would be able to spin the globe with their fingers, select earthquakes by touch and discover a range of information from this e.g. magnitude, depth, nearest settlement etc. Whilst touch screen integration was explored in relation to this project, it was clear that for this project it was not suitable due to limited access to touch screen capabilities and difficulties implementing it with a real time system. In order to implement touch screen abilities, an XML layer is used to store the locations of where a touch screen window/item will be placed. For static visualisations unlikely to change this is a relatively simple process to implement. However, with a dynamic data set like weekly earthquakes, a new file would have to be manually created for each new visualisation. Pufferfish's software is reliant upon the use of XML for all touch screen based applications. If future work was to be conducted, then ideally an automated process which generates dynamic touch screen windows for each earthquake could be created. This would then link to key data from USGS allowing not only spatial information to be represented but also key attribute data.

From the feedback gained, it was clear that the animation within the visualisation was well received by many. It stands to reason that by implementing touch screen capabilities, it would allow users to be able to explore and understand data in a much more dynamic and user controlled manner. The ability to directly touch events like earthquakes on a screen and have information pop up via windows, would provide a much more interactive and potentially information rich experience for users.

The work of Goldman et al (2010) strongly suggests this would lead to users having greater enjoyment and interaction with science which in turn, would hopefully lead to improved learning and understanding. The development of the modular script allowing the input of any coordinate based data and base image, allows a variety of visualisations to be made (Figure 12, 13). Future work could look at creating a playlist of visualisations all based on helping people learn more about geographical topics and issues.

Conclusion

Ultimately, this research has seen the creation of an automated mapping system based on real time earthquakes across the world for Puffersphere displays. Whilst this system was originally designed for this purpose, the adaptability of the coding has allowed any number of informative visualisations to be created with great ease. The evaluation of the visualisation showed that over 75% of users found the Sphere to be more useful than its two dimensional counterpart, 95% of users found it easy to understand the strength and location of earthquakes and finally 80% of users found seeing the quakes on the sphere helped them have a much greater understanding of the distribution and dynamics of earthquakes. Overall the results from this research strongly indicate that the primary and secondary research objectives have been achieved and that the Puffersphere is an effective medium for visualising real-time global earthquake events. Feedback gained has clearly shown that the visualisation is not perfect, and that it can be improved in a number of ways such as greater colour differentiation and more contextual information such as a tectonic plate base map. Overall the suggestions provided by users were relatively small, easy fixes which have been implemented in order to help the visualisation become more effective (Figures 19, 20). The next major step in development would be to incorporate more sensory capabilities, such as touch screen capabilities and an audio based commentary. These combined with the current visualisation would allow the creation of a much more complete and exhibition ready visualisation which could help truly highlight the Puffersphere's effectiveness for visualising geographical information.

References

Ainsworth, S and Loizou, A. (2003) The effects of self-explaining when learning with text or diagrams. *Cognitive Science, Vol. 27, pp. 660-681.*

Benko, H., Wilson, A and Balakrishnan, R. (2008) Sphere: Multi-touch Interactions on a Spherical Display. *UIST '08 Proceedings of the 21st annual ACM symposium on user interface software and technology, pp. 77-86.*

Bertin, J. (1978) Theory of communication and theory of 'the graphic'. *The international yearbook of Cartography Vol. 18, pp 118-126*

Card, S., Mackinlay, J., Shneiderman, B and Kaufmann, M. (1999) Readings in information visualisation: Using vision to think. Morgan Kaufmann Publishers Inc. San Fransicsco, CA.

Cartwright, W and Peterson, M. (2007) Multimedia Cartography, Second Edition, Springer, Berlin.

Companje, R., van Dijk, N., Hogenbirk,H and Mast,D. (2007) Globe4D, Time-travelling with an interactive four-dimensional globe. *ACM Multimedia, pp. 959-960.*

Crampton, J. (2009) Cartography: maps 2. 0. *Progress in Human Geography, Vol. 33, pp. 91-100.*

DiBiase, D., MacEachren, A., Krygier,J and Reeves, C. (1992) Animation and the role of map design in scientific visualisation. Cartography and Geographic Information Systems, Vol. 19, pp. 201-214.

Gellar, T. (2007) Imagining the world: The state of online mapping. *IEEE Computer Graphics, Vol. 27 pp. 8-13*

Geo-Cosmos (2014) Miraikan "Tsunagari" Project. Accessed from, https://www.miraikan.jst.go.jp/en/sp/tsunagari/geocosmos.html, on the 26th of February, 2014

Goldman, K., Kessler, C and Danter, E. (2010) Science on a Sphere: Cross-site summative evaluation. Accessed from http://www.oesd.noaa.gov/network/SOS_evals/, on the 26th February, 2014.

Hruby, F., Kristen, J and Riedl, A. (2008) Global stories on tactile hyperglobes- visualising global change research for global change actors. *Proceedings, Digital Earth Summit on Geoinformatics: Tools for global change research, Potsdam, Germany.*

ImageMagick, (2014) Command line processing. Accessed from http://www.imagemagick.org/script/command-line-processing.php, on the 25th June, 2014.

Kettner, S., Madden, C and Ziegler, R. (2004) Direct rotational interaction with a spherical projection. *Creativity and cognition symposium on interaction: Systems, Practice and theory.*

Korakakis, G., Pavlatou, E., Palyvos, J and Spyrellis, N. (2008) 3D visualisation types in multimedia applications for science learning: A Case study for 8th grade students in Greece. *Computers and Education, Vol. 50, pp. 390-401*

Lloyd, P. (1995) A historical review of visualisation in human cognition. *Educational Technology Research and Development., Vol. 43, pp. 45-56.*

Loftin, R. (2003). *Multisensory perception. Beyond the visual in visualisation. IEEE Computer, Sci. Eng, Vol. 5, pp. 56-58*

MacEachren, A. (2004) How maps work: representation, visualisation and design. The Guilford Press, New York.

Monmonier, M. (1996) How to Lie with Maps. University of Chicago Press.

National Oceanic and Atmospheric Administration (2014) Science on a Sphere. Accessed from http://sos.noaa.gov/What_is_SOS/index.html, on the 26th of February, 2014.

Pufferfish Displays (2014) Accessed from http://www.pufferfishdisplays.co.uk/services/, on the 25h February, 2014.

Riedl, A. (2007) Digital Globes. In Multimedia Cartography, pp. 256-266

Riedl, A. (2009) State of the art tactile hyper globes. *In: Buchroithner,M, True 3D in Cartography: Autostereoscopic and solid visualisation of geodata. Lecture Notes in Geoinformation and Cartography, pp. 215-226.*

Schratt, A and Riedl, A. (2005) The potential of three-dimensional display technologies for the visualisation of geo-virtual environments. *ICA Cartographic Conference, 2005, A Coruna, Spain.*

Scougal, C. (2014) Is a Puffersphere an effective medium for visualising real-time global earthquake events? Part II: Technical Report, Unpub. MSc. Dissertation, Univ. of Edinburgh

Tufte, E. (1978). The visual display of quantitative information. Graphics Press, Cheshire, Connecticut

Tufte, E. (1983). The visual display of quantitative information. Graphics Press, Cheshire, Connecticut

Yip, B., Goyette, S and Madden, C. (2006) Visualising Internet Traffic Data with Three-Dimensional Spherical Display. *Conferences in Research and Practice in Information Technology, Vol. 45.*

United States Geological Survey (2014) Accessed from http://earthquake.usgs.gov/earthquakes/feed/v1.0/, on the 13th January, 2014.

Wang, H., Chang, C and Li, T. (2007) The comparative efficacy of 2D- versus 3D-based media design for influencing spatial visualisation skills. *Computers in Human Behaviour, Vol. 23, pp. 1943-1957.*

Worboys,M(1995) *GIS: A Computing Perspective*, Taylor & Francis, London

Williams, R. (2003). The non-designers design book, second edition. Peachpit Press, Berkeley, CA.

PART II: Technical Report

Table of Contents

Table of Figures

Table of Tables

Introduction

Part I briefly covered the steps required to create an automated system for real-time monitoring of earthquakes upon a Puffersphere. This section will go into great detail how this system was developed and will also highlight additional methods which were created but ultimately not used in the final system. The first stage of this report will cover additional literature relating to cartography and visualisations in general. It will then go into detail on the methodology employed to create the visualisation.

Cartographic Literature Review

The work of Card et al, (1999) reports that visualising information is a beneficial process for influencing cognitive processes and understanding data. MacEachren (1992) suggests that visual representations allow spatial patterns to be discovered and can highlight potential issues and problems, which force people to use the most powerful information processing ability they have, that of their vision. The importance of visual variables in cartography has been established for many years and is a cornerstone in the creation of any good visualisation. Bertin, 1978 initially proposed the importance of key "retinal variables" within map design as seen in Figure 1.

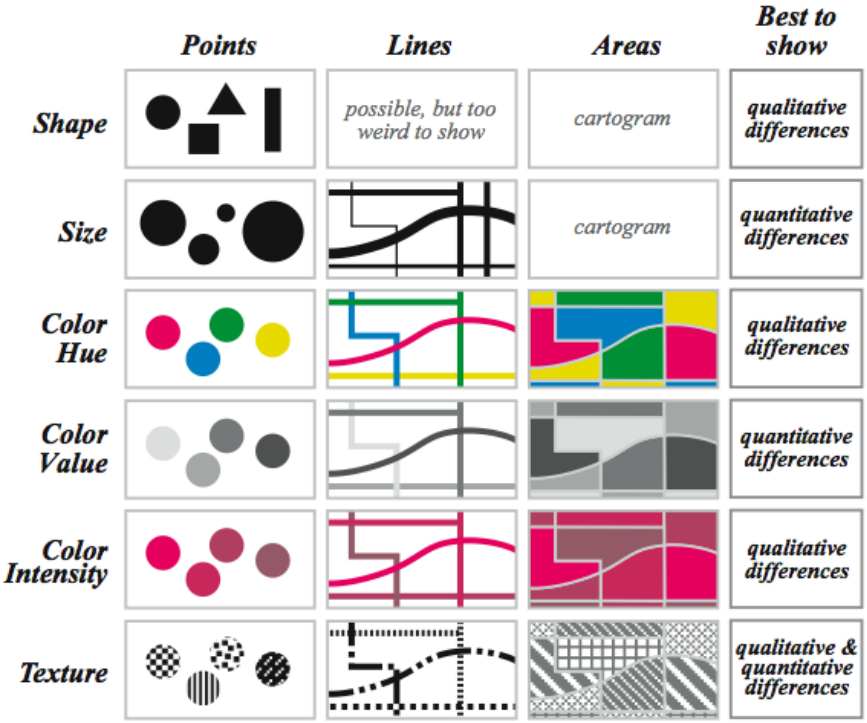

Figure 1 shows Bertins classification of visual variables.

Aspects such as colour, shape, size, hue, intensity and texture are all deemed core components in helping a visualisation maximise its effectiveness. The overall message from the literature is the importance of a clear and well established visual hierarchy, which accurately portrays the information it is representing. The use of the major visual/retinal

variables is core in producing an effective visualisation. The use of inappropriate symbols, colours, and shapes can immediately render a visualisation ineffective, if design elements conflict in conveying information across to users. Williams, 2004 argues that repetition is a key design principle and that classifying areas of similarity can help to create visual consistency within a visualisation. This is echoed by the well-established Gestalt theory (Koch, 2001) which is based upon how humans will instinctively organise visual elements into groups if displayed optimally. This in turn helps people to cognitively process information faster and more effectively (Card et al, 1999). By symbolising/displaying similar information, users can instinctively view patterns within visualisations. In the context of an earthquake visualisation, if a large number of quakes are over 6.0 and are displayed in medium sized orange circles, it is easy to see that they are similar in nature and most probably occurring for the same reasons. However, the second a smaller or larger individual quake event is in proximity to this group it can immediately be distinguished from the others. This can be seen as being a dissimilar entity/anomaly (MacEachren, 2004). Effective earthquake visualisations show this principle very well in that users will be able to immediately identify areas of high and low seismic activity via the clustering of earthquake events (Figure 2).

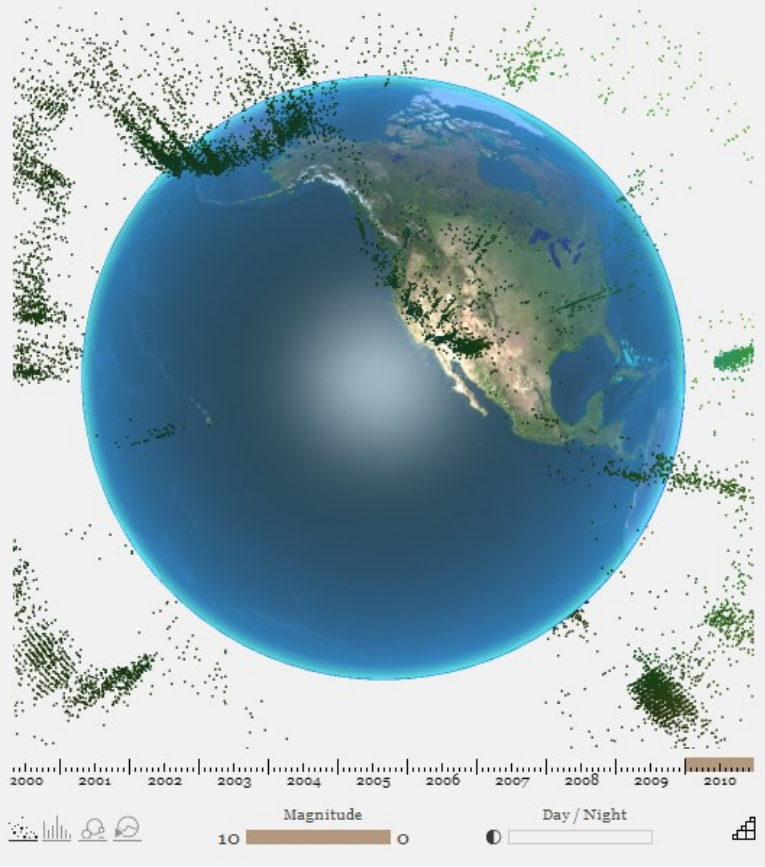

Figure 2: Ninepointfive, (2014) WebGL visualisation displaying quakes over the last 30 years

Methodology

The programming used for this research was all conducted using Python. This was due to its ease of use, wide range of well supported modules and its object orientated nature. Figure 3 shows the major steps of the methodology which will subsequently be explained.

This entire script has been constructed to be modular and to be installed and operated easily on a Puffersphere PC unit. To aid in this, certain key variables such as save directories and video names have been specified via the use of a text based configuration file. This essentially makes it easy to specify key names and directories where a user wishes images/videos etc. to be saved. This was done via the use of the Python configParser module which essentially reads specified text strings as arguments for variables within the Python script.

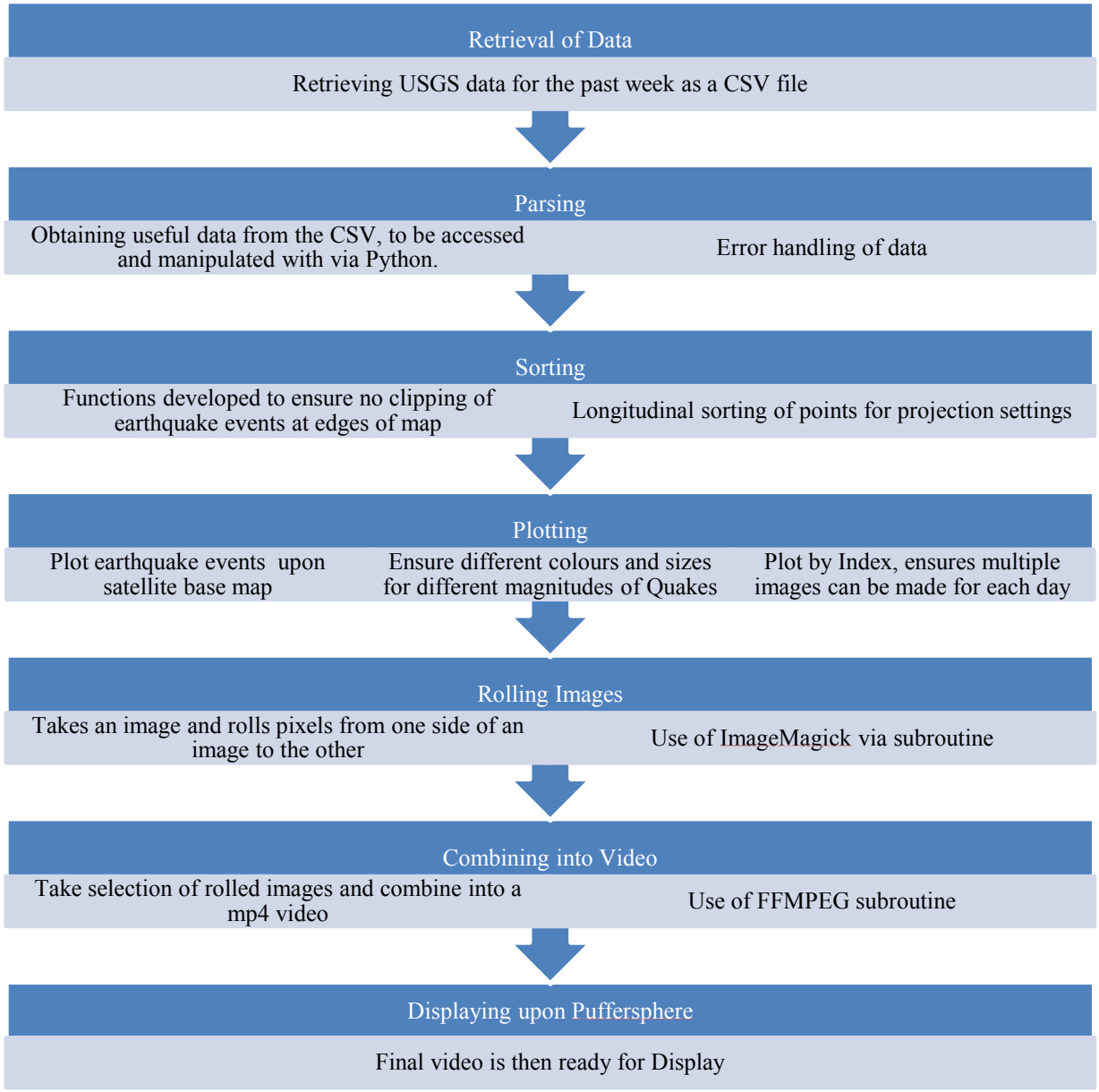

Figure 3: Detailed methodological process

Retrieval of Data

The Python module urllib and urllib2 were used to automatically retrieve United States Geological Survey data for global earthquakes. These modules essentially connect to an external URL and then access it within a programming environment. The Urllib2.urlopen command was used to directly open earthquake data for the last week to allow parsing and sorting of the data, as seen in Figure 4. Urllib.urlretrieve was used to download a hardcopy of the data to a designated directory. This hardcopy is necessary in case any errors appear in the live USGS stream of data retrieved by urllib2. The hardcopy will be run through a function known as **insurance** in order to rectify any errors present and allow a functional data set to be worked with.

```
#data = urllib2.urlopen('http://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/all_day.csv')
data = urllib2.urlopen('http://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/all_week.csv')
#data = urllib2.urlopen('http://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/all_month.csv')
#data=urllib2.urlopen('http://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/significant_month.csv')
urllib.urlretrieve('http://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/all_week.csv','safe.csv')
```

Figure 4: This code retrieves USGS earthquake data directly. Each line commented out represents a different data file e.g. 30 days, one day or significant quakes in the past month. The final line of code downloads a hardcopy of the data in case the live streamed version is corrupted or incomplete.

Parsing of Data

The **parsing** function was developed to allow access to rows of data within CSV files. The function takes a series of empty lists as its arguments and the live data and hardcopy as well. The function essentially pans through a CSV and appends the information within the rows of the file to the subsequent lists seen in Figure 5. If the live stream of data does not work, perhaps due to an empty value within a column, the **insurance** function is activated before any parsing of data occurs.

```
def parsing(data1,safety,latst,lonst,lons2t,lons3t,lons4t,lemont,magt,placet,timet):
    '''Parses CSV data, if any errors with online data, downloads it and repairs any blank values'''
    csv_data =csv.reader(data1)
#reads as csv
    next(csv_data)

    try:
        for row in csv_data:
                #print row
                latst.append(float(row[1]))
                lonst.append(float(row[2]))
                lons2t.append(float(row[2]))
                lons3t.append(float(row[2]))
                lons4t.append(float(row[2]))
                lemont.append(float(row[2]))
                magt.append(float(row[4]))
                placet.append(str(row[13]))
                timet.append(row[0])
    except:
        insurance(safety)
        with open(safety) as f:
            csv_data =csv.reader(f)
            next(csv_data)
            for row in csv_data:
                print row
                latst.append(float(row[1]))
                lonst.append(float(row[2]))
                lons2t.append(float(row[2]))
                lons3t.append(float(row[2]))
                lons4t.append(float(row[2]))
                lemont.append(float(row[2]))
                magt.append(float(row[4]))
                placet.append(str(row[13]))
                timet.append(row[0])
```

Figure 5: Code developed for parsing USGS CSV data.

The **insurance** function takes the downloaded hardcopy of data, and checks every row within a specified column for blank values. If blank values are found, it is then replaced with a zero value and the CSV is saved as a new file (Figure 6). This new file can then be parsed normally and the data is then ready for sorting and plotting processes. As the overall aim of this script is to provide automatic plotting of real-time earthquakes, it is vital that a safeguard like this is in place. Whilst it is not accurate in the sense that blank values all become zeroes, its role is to simply allow the rest of the data to be usable and all other methods to function correctly if there are any issues with the source data.

```
def insurance(data):
    '''Takes downloaded copy of data and edits for errors and replaces blank
    cells with 0's. Ensures visualisation can always be made, if errors with USGS data'''

    output="safety.csv"
    with open(data, "rb") as infile, open(output, "wb") as outfile:
        r = csv.DictReader(infile)
        w = csv.DictWriter(outfile, r.fieldnames)
        w.writeheader()
        for row in r:
            if not row["latitude"].strip():
                row["latitude"] = "0"
            #w.writerow(row)
            if not row["longitude"].strip():
                row["longitude"] = "0"
            if not row["mag"].strip():
                row["mag"] = "0"
            w.writerow(row)
```

Figure 6: Insurance function

The lists returned from the **parsing** function, are subsequently reversed (Stack Overflow, 2014) to ensure that the data is ordered chronologically (Figure 7).

```
parsing(data,safety,latst,lonst,lons2t,lons3t,lons4t,lemont,magt,placet,timet)
#

lats=latst[::-1]
lons=lonst[::-1]
lons2=lons2t[::-1]
lons3=lons3t[::-1]
lons4=lons4t[::-1]
lemon=lemont[::-1]
mag=magt[::-1]
place=placet[::-1]
time1=timet[::-1]
```

Figure 7: Example of the parsing function in action and the reversal of data within the core lists.

Sorting of Data

One of the first problems encountered in this research was the occurrence of earthquakes in regions which were essentially at the edges of the map. Quake events would be plotted as circles, only for half of the circles to be clipped of the image, as seen in Figure 8. Whilst initially this problem seemed quite limited, it became clear that with earthquake data over longer time periods e.g. week/months, there were serious visualisation problems. These were exacerbated by the fact that the edges of the base map being used were essentially the west coast of America and the Far East, typically areas of high seismic activity.

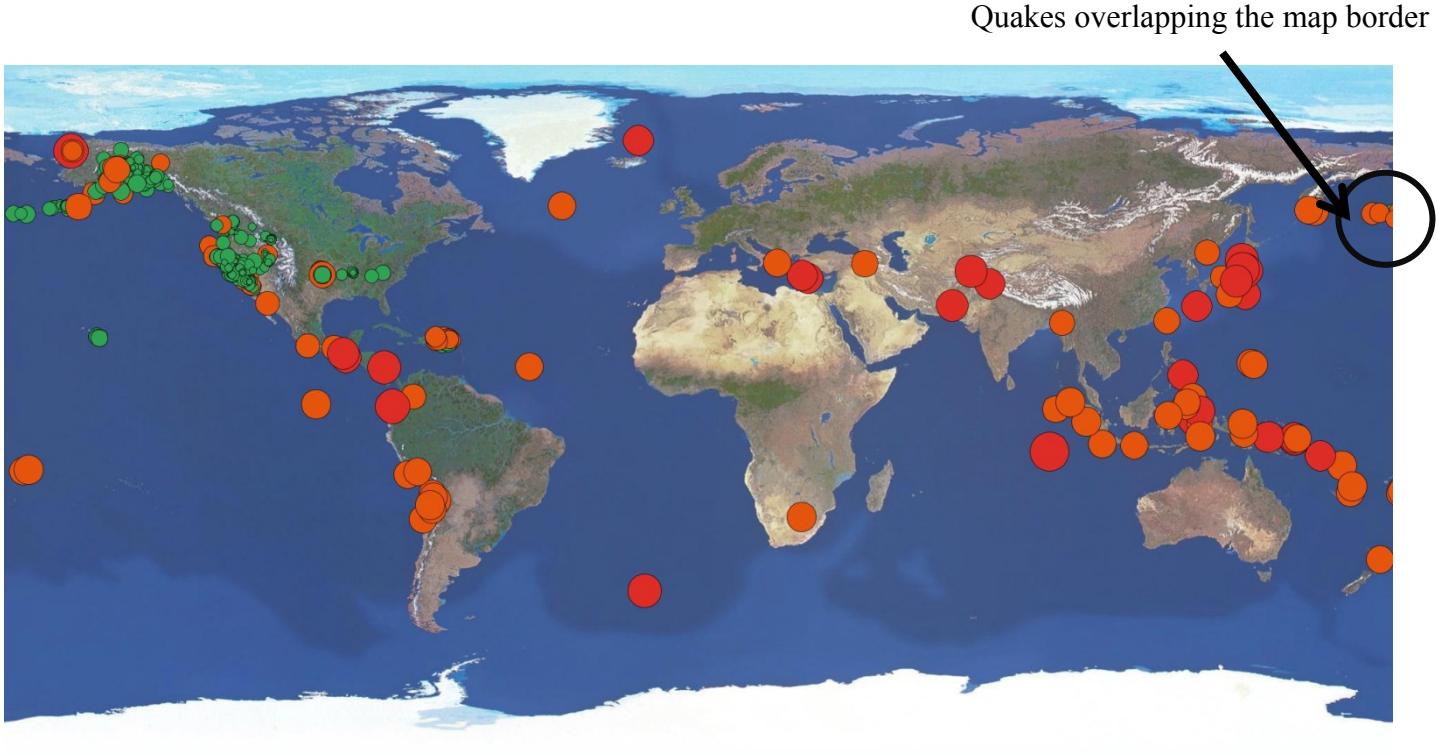

Figure 8: Sub-optimal global earthquake map

In order to combat this, two functions known as **sorting1 and sorting2** were developed (Figure 9). **Sorting1** essentially takes a list of longitudes and sorts them from the lowest longitude to the highest, then adds them to a Numpy array, which allows the differences between each longitudinal point to be calculated. Numpy arrays are memory efficient containers within Python which allow fast numerical operations to be conducted (Scipy, 2014). The largest difference between two neighbouring longitudes is then recorded, as are the index positions of the associated quakes. Whilst for this research it allows us to know where the lowest region of seismic activity is on a longitudinal scale, it can easily be applied to any coordinate based data. The **sorting2** function is then used to determine the best position for the starting longitude of the base map to ensure no clipping occurs. Essentially, it takes the two index positions with the largest difference and assesses whether the starting longitude should be positive or negative (e.g. if the first index position is -6 and the second position is 12, it will determine that it should be a positive value of 6). Once this has been done, it half's the value so the starting longitude of the base map will be directly in the middle of the two index positions, as seen in Figure 10.

```
def sorting1(lons):
    '''Sorts longitudes, finds difference between every point and reports the
    max difference value and index position as well as partnering index position'''

    lons.sort()
    v=np.diff(lons)
    index, value = max(enumerate(v), key=operator.itemgetter(1))
    return lons[index],lons[index+1], value


def sorting2(x,y):
    '''This function takes the difference between the 2 index's and then halfs it.
    This ensures the maximum distance between the 2 points is found and can then be
    used to set the optimal figure break point. Ultra ignores negative aspects, so this
    function ensures negative values are negative and postive are positive.'''

    if x <0 and y <0:
       a=x + y
       return a/2

    elif x <0 and y >0:
        a=x + y
        return a/2

    else:
        x>0 and y >0
        a=y + x
        return a/2
```

Figure 9: Illustrates the two developed functions to combat the problem of edge based earthquakes

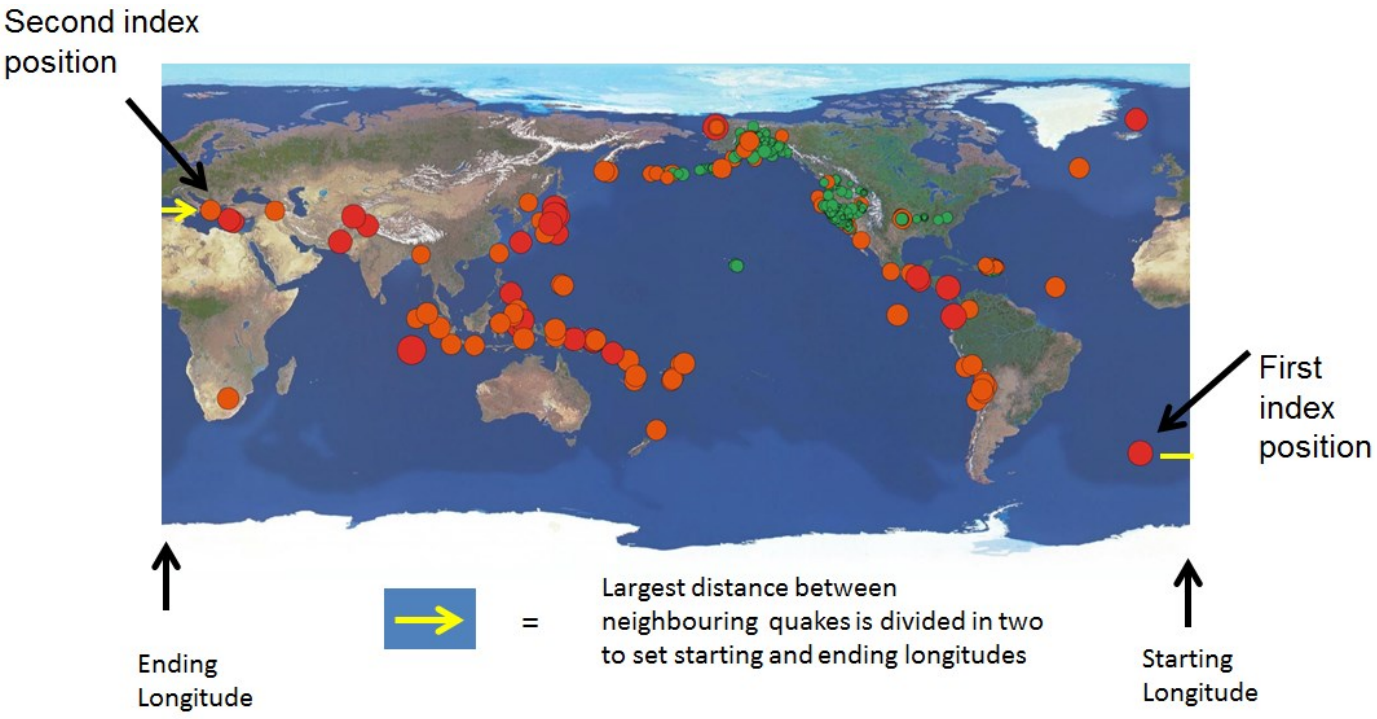

Figure 10: Optimal earthquake map after sorting functions.

Plotting of Data

Before any plotting of earthquakes occur, a map projection must be setup within Python. The Python module Basemap was used for this purpose and for warping the base satellite image (Matplotlib, 2014).

The **proj** function was developed to ensure the correct projection is established (Tosi, 2014) the correct longitudes are set and a suitable equi-rectangular base image is present (Figure 11).

```
def proj(a):
    '''Sets the equirectangular projection, and ensures correct start/end longitudes'''
    if a<0:
        m = Basemap(projection='cyl',llcrnrlat=-90,urcrnrlat=90,\
        llcrnrlon=(a),urcrnrlon=360+(a),resolution='c')
        m.warpimage(image="base.jpg")
        return m
    if a>0:
        m = Basemap(projection='cyl',llcrnrlat=-90,urcrnrlat=90,\
        llcrnrlon=(a),urcrnrlon=360+(a),resolution='c')
        m.warpimage(image="base.jpg")
        return m
```

Figure 11: Illustrates the code developed for ensuring optimal longitudes and the correct projection.

The value obtained from the previous **sorting2** function(i.e. the position furthest from any horizontal quakes) is used as a the starting longitude value (llcrnlon=(a)), and subsequently the end longitude is simply 360 degrees + the (a) value. This ensures that the map always starts and finishes at the optimal position for displaying all earthquake events. It has been thoroughly tested with a range of data and has been proven to work with very high numbers of earthquakes (8000+). The majority of testing was done on the larger monthly data set of earthquakes to ensure the function performed correctly. Figures 8 and 10 show a before and after optimisation of a monthly data set in July. These figures show how the function has designated the best possible position for the visualisation to begin.

In the vast majority of cases the function typically sets the starting longitude within western Europe as it along with Africa have a low seismicity compared to the rest of the world on a longitudinal basis.

The **mapPlot** function ensures that all earthquakes are plotted upon the base map and that they are appropriately symbolised and coloured, according to their magnitude. The function requires 5 arguments. These are a list of longitudes, latitudes, magnitudes, the alpha/transparency value of the quake symbol and the appropriate projection reference (m) as seen in Figure 12. The zip operator essentially links the longitude, latitude and magnitude lists together. When it states, **for lon, lat, mag in zip (Lons2, lats, mag)** it's simply saying that every action which follows this statement will be applied to every element within each of these lists.

```
def mapPlot(lons2,lats,mag,alpha1,m,thick):
    '''Takes lat, long and mag and plots quake events on map, varying
    colour & size for magnitude'''

    for lon, lat, mag in zip(lons2, lats, mag):
        mp.axis('off')

        x,y = m(lon+360, lat)
            #ensures all points plotted when origin map placement is altered.
        z,j =m(lon,lat)
#        msize = mag * 4,5,6.25
        msize = mag * 2
        msize1=mag * 3
        msize2=mag * 4.5

        if mag >4.0 and mag <5.0:
            m.plot(x, y, '#31a354',marker="o",alpha=alpha1, markersize=msize,mew=thick)
            m.plot(z, j, '#31a354',marker="o",alpha=alpha1, markersize=msize,mew=thick)

        elif mag >5.0 and mag <6.0:
            m.plot(x, y, '#f16913',marker="o", alpha=alpha1, markersize=msize1,mew=thick)
            m.plot(z, j, '#f16913',marker="o", alpha=alpha1, markersize=msize1,mew=thick)

        elif mag >6.0:
            m.plot(x, y, '#de2d26',marker="o", alpha=alpha1, markersize=msize2,mew=thick)
            m.plot(z, j, '#de2d26',marker="o", alpha=alpha1, markersize=msize2,mew=thick)
```

Figure 12: Code for the primary plotting function **mapPlot.**

Following on from this, every **longitude** and latitude is converted into a equi-rectangular coordinate (**x,y=m(lon+360, lat).** This is not entirely necessary as the coordinates are already in the correct form for this projection, however it simplifies the subsequent plotting commands to do this now. If converting from a different projection, it is imperative this is done before any specific plotting commands (e,g,**m.plot**). Each earthquake is plotted twice **(z,j=m(lon,lat).** This is to ensure that when the starting longitude is changed every point is plotted in the correct place. Only plotting the quakes in **x,y=m(lon+360,lat)** can lead to some quakes not being represented depending on the starting longitude. For this reason two sets of coordinates are used to ensure accurate and precise plotting. The use of **msize** variables simply denotes a magnifier effect, so for magsize1, it simply means that every earthquake with a magnitude between 4 and 5 should have a marker that is its own magnitude multiplied by 3. This simply allows for better visualisation and representation of larger and smaller earthquakes across the globe. The final three statements of the function simply state that earthquakes of certain magnitudes should be plotted with different characteristics e.g marker size and colour, transparency and line thickness.

The **plotIndex** function is a very simple function, which makes use of the **mapPlot** function to allow plotting of earthquakes within a specified range/index of a list. It simply requires a

list of longitudes, latitudes, magnitudes, a suitable map projection and the number of quakes needing to be plotted. For the script developed, the indexing element was used in combination with the **time3** and **uDay** functions. The **time3** function essentially takes a list of dates and time from the data, strips away the time element, and formats into a day/month/year format as seen in Figure 13.

```
def time3(self):
    '''Formats date information nicely'''
    a=[]
    for x in self:
        d=dateutil.parser.parse(x)
        #c=d.strftime('%d-%m-%Y %H:%M:%S')
        c=d.strftime('%d-%b-%Y')
        a.append(c)
    return a
```

Figure 13: Date formatting

This list is then used by the **uDay** function (Figure 14) which finds the number of times a unique date appears. This allows us to know the precise number of earthquakes which occur each day.

```
def uDate(date):
    '''Takes a list of dates and determines how many unique dates are
    present and the number of quakes occuring within each day'''
    c=[]
    for i in date:
        c.append(date.count(i))
    return c
```

Figure 14: Function for counting number of earthquakes for each day

```
p=time3(time1)

c=uDate(p)
c= list(OrderedDict.fromkeys(c))
```

```
[123, 227, 241, 271, 219, 210, 63]
['23-Jul-2014', '24-Jul-2014', '25-Jul-2014', '26-Jul-2014', '27-Jul-2
014', '28-Jul-2014', '29-Jul-2014', '30-Jul-2014']
```

Figure 15: This code shows the **time3** function being used on the time data (time1) and then the **uDate** function being used to find the unique counts of each date. The last piece of code simply creates an ordered dictionary list of all the dates in the correct chronological order. This is important, as native sorting of dates in Python can encounter issues when changing from one month to the next. Using a dictionary based list, ensures the exact order of dates is maintained.

The number of quakes which have occurred each day, acts as the index variables for the **plotIndex** function seen in Figure 16. The function is then used 8 separate times for each day of the week to allow 8 separate images with the earthquakes of each day plotted upon them.

```
def plotIndex(x,y,j,i,m,thick):
    '''Uses mapPlot for specified indiceis within a list, e.g 20-40'''
    if len(x)> i:
        mapPlot(x[0:i],y[0:i],j[0:i],1,m,thick)

    elif len(x) <i:
        mapPlot(x[:],y[:],j[:],1,m,thick)
```

Figure 16: This method plots earthquakes from one index point to another. The first statement simply means if there are more earthquakes than specified in the **I** variable (the quakes for a day) then it will plot all quakes from zero to that number. Otherwise, it will simply plot all earthquakes present.

Rolling Images

The rolling process is essentially focused on taking an image and moving the pixels from one end of the image to another to mimic a rotation effect. The **vid** function was developed for this process and has 6 primary arguments (Figure 17). These are the input image**(x)**, the number of frames required**(y),** the distance or width of the image (**z**) (2000 is required for Puffersphere visualisations)**,** name of image folders, save directory and an output name (Figure 17). The function begins by creating a series of directories or folders for each image which is to be rolled. From this point a basic counter is started (**frame=0**) and a list is created for the range of the number of frames specified e.g 200 frames, means there will be a list from 1-200. A loop is then used for every element within the range of the frames list **(p)**. **Frame = frame+z1** refers to the previous variable, where z1= z/y. z1 is essentially the width of the image(2000) divided by the number of frames specified. So for example, if z=2000 and y=1400, z1 will be 1.428. So **frame=frame+z1** is essentially saying that for every element within (**p**), the frame is plus 1.428. So the starting frame will be 1.428, then 2.85, for the next and so on. This ensures that the specified number of frames are moved accordingly. The following variable **a=str(frame) and a1= '+'str(a)** are simply used to ensure that the value is a string value with a plus sign in front of it. This is vital for the next stage where the variables are passed onto an ImageMagick sub process. The subprocess.Popen command is used to externally access and run commands from the command prompt within a Python script. In this case it is used to call ImageMagick and roll each of the original 8 earthquake images. The command has a few primary arguments such as action type=**convert**, the name of the image being rolled(**x**), the actual command(**-roll**) , the degree by which an image should be rolled(**a1),** the scale of the image(**2000*1000)**, and what the output image should be called and where it should be saved.

The wait command is used after the sub process to ensure that the sub process's actions are completely finished before continuing. Failure to specify the wait command will result in videos being made from an incomplete selection of images.

```
def vid(x,y,z,name,directory,output):
    '''Creates dir for images which are rolled via imagemagick subprocess,
    images then compiled into video via ffmpeg subprocess'''
    print x

    route=directory+str('\\framesFor')+str(name)+str('\\')
    #simply acts as a path variable to where rolled images are located
    print route
    raw = route+str('%d.jpg')
    #again acts as a path variable to images
    print raw
    z1=z/y
    #This is simply number of frames divided by distance of image(2000 for Puffersphere)

    print "ANIMATION PROCESS"
    if not os.path.isfile("images\\framesFor"+str(name)) and not os.path.isdir("images\\framesFor"+str(name)):
        os.mkdir("images\\framesFor"+str(name))
    #essentially if there is not a  folder for the selection of rolled images, will make folders for each

    frame = 0
    p=list(xrange(y))
    #makes list the length of frames specified
    for i in p:
        frame=frame+z1
        #counter used to determine distance for each rolled image
        a= str(frame)
        a1='+'+str(a)
        o=subprocess.Popen(["convert", x,"-roll",a1, "-scale", "2000x1000", route+str(i)+'.jpg'],shell = True)
        #This subprocess accesses imageMagick via popen, and carries out the neccessary roll commands.
        o.wait()
        #IMP. Tells system to finish this before starting next process. If not present vids made of incomplete images

    c=subprocess.Popen(["ffmpeg", "-f", "image2",
                            "-i", raw, "-c:v", "libx264",
                            "-pix_fmt", "yuv420p", "-preset",
                            "ultrafast", "-tune", "fastdecode",
                            "-profile:v", "main", "-vf", "scale=2000:1000",
                            "-r", "30", "-coder", "0", "-g", "6", "-crf", "20",
                            "-y", output],shell=True)
    c.wait()
    #This subprocess accesses FFMPEG and compiles all rolled images within a dir into a mp4
    #Various settings determine quality, speed and compression.
```

Figure 17: Detailed code illustrating the creation of rolling and video compiling functions.

The following figures show a brief demonstration of the rolling process.

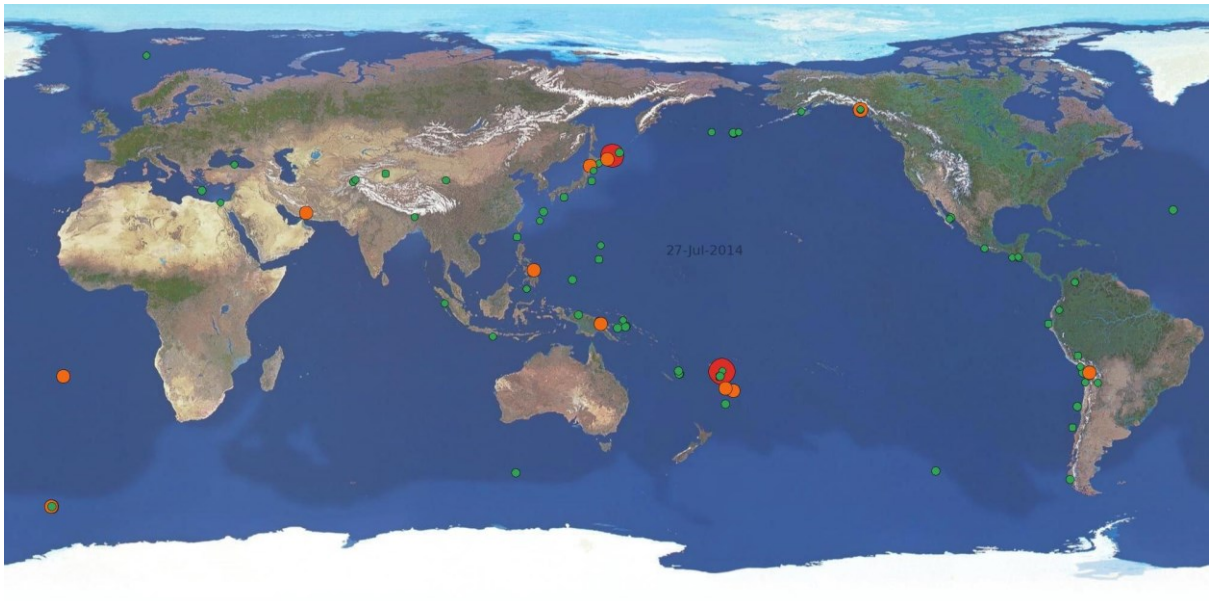

Figure 18: This is the first image of the rolling process. The width of the image is 2000 and the number of frames set is 200. This means that 200 images will be created, and each image will progressively move ten pixels each time.

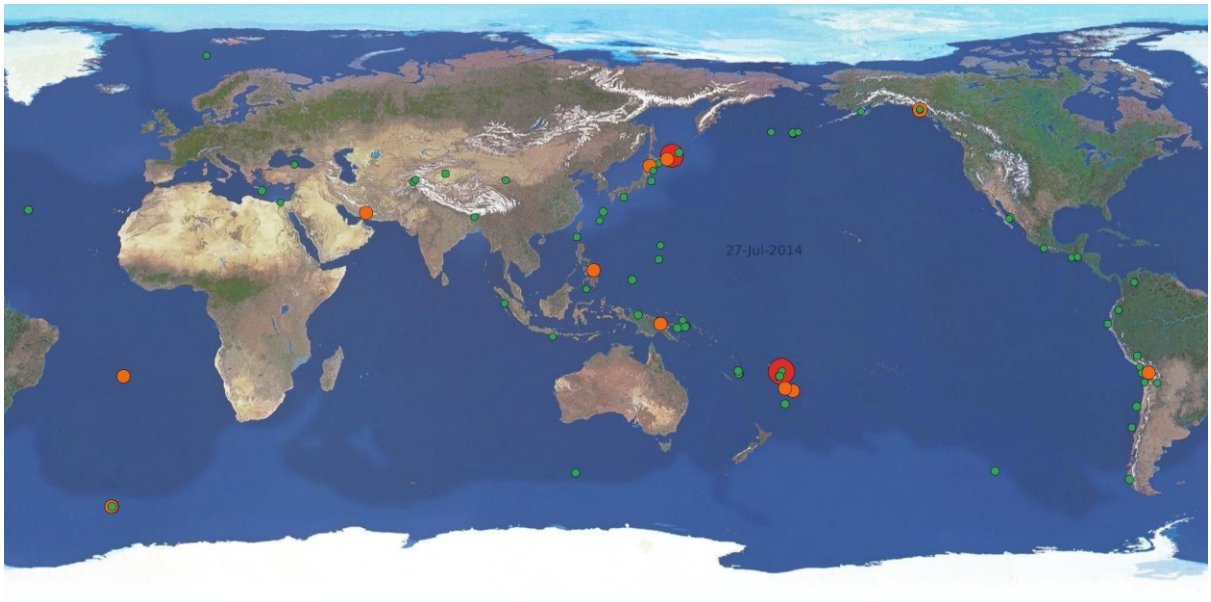

Figure 19: 10th image of the rolling process.

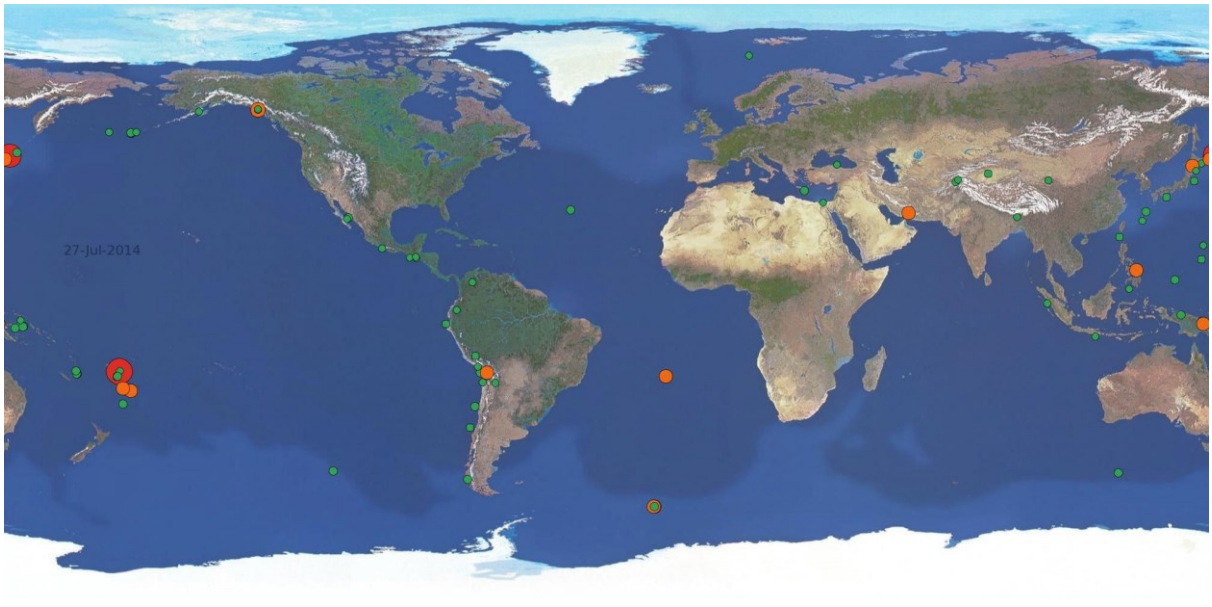

Figure 20: 100th image of the rolling process.

Figures 18, 19 and 20, clearly illustrate the dynamics involved within the rolling function. This example used 200 frames as a means of clearly illustrating the way the function actually operates. Having a large number of frames ensures the distance moved by each subsequent image is relatively small. This ensures that when the images are compiled as a video, the output is as smooth as possible. The simplified example highlights the potential problems of using too small a number of frames as larger distances are rolled in each subsequent image, resulting in a very fast and choppy output video when compiled. This example is moving 10 pixels for every image, when in reality a smooth output should be closer to 0.588 pixels/image.

The next stage of the **vid** function occurs outside of the loop and is a sub process calling upon FFMPEG. This process essentially takes the stacks of images created by the previous rolling process and combines them into an mp4 movie file (Figure 21). A full description of each argument used can be seen in Table 1. Further commands can be found at ***https://ffmpeg.org/ffmpeg.html***.

```
    c=subprocess.Popen(["ffmpeg", "-f", "image2",
                        "-i", raw, "-c:v", "libx264",
                        "-pix_fmt", "yuv420p", "-preset",
                        "ultrafast", "-tune", "fastdecode",
                        "-profile:v", "main", "-vf", "scale=2000:1000",
                        "-r", "30", "-coder", "0", "-g", "6", "-crf", "20",
                        "-y", output],shell=True)
    c.wait()
    #This subprocess accesses FFMPEG and compiles all rolled images within a dir into a mp4
    #Various settings determine quality, speed and compression.

#veryslow
```

Figure 21: Detailed look at FFMPEG settings for **vid** function.

Table 1: Detailed look at FFMPEG commands used

FFMPEG Command table	
-f	Forces input file format
image2	Specifies creation of video from a selection of images
-i	Specifies Input filename
-c:v libx264	Encodes video with x264 codec
-pix_fmt	Shows available pixel formats
yuv420p	420p video format
-preset	Sets encoding preset(encoding speed vs compression ratio
ultrafast	Very fast encoding speed at cost of high disk usage
-tune	Sets tuning of encoding parameters
fastdecode	Fast decode speed at cost of disk space
profile:v	Limits output to specific H.264 profile(mp4)
-vf	Specifies video filters
scale=2000:1000	Scale of video
-r	Sets frame rate
crf	Sets output quality for whole file(optimal range 18-28)
-y	Overwrite output file
output	Sets output name
concat	Joins videos together
setpts	Alters playback speed

The primary arguments are specifying the output file, the scale (**2000:1000)**, and the FPS value **("-r", "30")**. Again a wait command is applied to ensure each action is successfully completed before moving onto the next. It is important to note that every argument within a sub processes command must be encapsulated by quotation marks, unless you are directly passing a Python defined variable which is already formatted as a string value. In which case quotation marks or apostrophes are not necessary.

The **vid** command was run 8 times for each of the base images, resulting in eight mp4 videos for each of the days earthquakes within a week. In order to merge these videos, another sub process using FFMPEG is required. This process is based on using the **concat** command in combination with a text file. The command merges all the videos specified within a text file as one, to create an output video. For this reason the **makemylist** function was created (Figure 22).

```
def makemylist(directory):
    '''Makes a txt list of every mp4 within a directory, to allow concatination of mp4's'''
    f= os.listdir(directory)
    test=open("mylist.txt","w")
    for i in f:
        if i.endswith('output.mp4'):
            pass
        elif i.endswith('.mp4'):
            test.write('file \'')
            test.write(directory+str('/'))
            test.write(i+str('\'') +'\n')
```

Figure 22: This function essentially creates and writes to a text file the name of every mp4 file within a directory. This can then be passed as an argument to the FFMPEG sub process for concatenation.

After this point another FFMPEG based sub process is run which is based upon speeding the final video playback up (Figure 23). This command simply requires an inputfile, output file and the degree to which one wants a video sped or slowed down. **Setpts= 0.5*PTS** essentially means that the original video will be halved or sped up twice as fast. Conversely if it was set to 2, it would make the final video twice as long.

```
makemylist(viddir)
#Function finds every mp4 file and adds name to a list

concat=subprocess.Popen(["ffmpeg", "-f", "concat", "-i","mylist.txt", "-c", "copy","-y", "output.mp4"])#,shell = True
concat.wait()
#THis FFMPEG subprocess combines all mp4's as one

fast=subprocess.Popen(["ffmpeg", "-i", "output.mp4", "-vf","setpts=0.3*PTS","-y", outputvid])#,shell = True
fast.wait()
#Command to alter speed/runtime of final video
```

Figure 23: FFMPEG commands for **concatenation** and for speeding the output playback.

Other Functions and settings within the primary script

Three similar functions were developed to ensure that only current data is used for each new earthquake visualisation created. These were **deleteDir, deleteJpg** and **deleteMp4**. Essentially each of these functions is directed to the appropriate save directory and then deletes out of date images folders and mp4's which have been created. It does this using the OS and system functions native to Python (Figure 24). Giving read and write access to programs and scripts can potentially be disastrous, as one directories root location may differ between systems, culminating in possible deletion of core system files. For this reason, safeguards were implemented which ensure that only folders and files with a certain string within their name can be deleted. For example each quake image created goes into a framesFor a001 etc. folder, this function ensures that only folders/Dir's with framesFor in the folder name will be deleted. The use of the configParser and the external configuration file makes it easy to guide these functions directly to the necessary directories where they will be used upon.

```
def deleteDir(save):
    '''Deletes any Dir's beginning with framesFor, ensures new data always used'''
    path =[x[0] for x in os.walk(save)]
    #peach =apple[0]
    #print apple[2:66]
    path1=path[1:66]
    print path1
    for f in path1:
        if 'framesFor' in f:
            shutil.rmtree(f)

def deleteJpg(save):
    '''Deletes all images within save directory, ensures new images always used'''
    files = glob.glob(save+str('/*.jpg'))
    for f in files:
        if not 'base.jpg' in f:
            if not 'intro.jpg' in f:
                os.remove(f)

def deleteMp4(vidloc,outputvid):
    '''Deletes all mp4's in vid dir, ensures new vids always used'''
    files2 = glob.glob(vidloc+str('/*.mp4'))
    for g in files2:
        if not 'output.mp4' in g:
            if not outputvid in g:
                os.remove(g)
```

Figure 24: Deletion functions created to streamline automatic procedures.

Key display settings

These settings are small but important in ensuring the final static images are of a high quality**. Figure.frame= "false"** and **mp.axis('off')** are commands used to turn off any graph axis or grids which may be on by default in matplotlibs settings. It is vital that these are turned off, as not doing so will result in a black seam along the edge of each image created and will be visible upon the final visualisation on the Puffersphere. Two other important settings are that the **figure.set_size_inches** setting is at 20, 10 and the **mp.tight_layout** is set to **pad=0**. The first setting essentially states that the image will have a 2000*1000 resolution and the second setting allows only the image to be saved, with no whitespace surrounding it. This is essential as if any white space appears around the image it will be displayed in the final visualisation.

The use of **mp.clf()** and repeated use of **m.warpimage(image= "base.jpg")** are required between the plotting of each new base image of data. Without using **mp.clf()**, data which has been previously plotted will show up on subsequent plots. The command **mp.clf()** essentially clears all aspects of the current figure. However, in doing this it also removes the base image as well, so one needs to recall the **m.warpimage** function after clearing the figure, to ensure the image is correctly displayed again. A number of basic functions and settings were also used within the script to create each of the elements on the introductory slide of the visualisation. Aspects such as titles, names, earthquake symbols and dates are created using the built in **mp.text** and **m.plot** functionality.

Unused functions

A series of functions were developed to allow the plotting of magnitude labels for each earthquake. These functions were created in order to see if the visualisation could be improved with the addition of in depth magnitude information.

The first function developed was **labelMag** which essentially takes the longitude, latitude and magnitude of an earthquake and writes a magnitude label for it, if the magnitude is over a certain size (Figure 25). Whilst this function works, it is in no way useful to the overall visualisation as in nearly all cases earthquakes are overlapping each other leading to a muddled and confused output. To remedy this, a series of linking functions were created to only allow aesthetically pleasing labelling of earthquakes. The overall aim was to prohibit conflicting labels and to provide magnitude labels to quakes where it would look appropriate.

```
def labelMag(x,y,magnitude,i):
    '''Adds magnitude numbers to map, i = mag > than'''
    for  lon, lat,mag in zip( x, y,magnitude):
        if mag > i:
            mp.text(lon+358.5,lat-.5,mag)
            mp.text(lon+-2,lat-.5,mag)
```

Figure 25: Magnitude labelling function

The first function developed for this goal was the **nn** function, which essentially finds the distances between nearest neighbours in an array(Figure 26).

```
def nn(array):
    '''Function to calculate nearest neighbour between point array
    k=number of neighbours,d_upper=distance)'''
    c=[]
    tree=scipy.spatial.cKDTree(array,leafsize=100)
    for item in array:
        result=tree.query(item,k=2,distance_upper_bound=10)
        c.append(result)
    return c
```

Figure 26: Nearest neighbour method

This can be configured for multiple neighbours and for specified ranges. The **difference** function was then created to append the returned array from the **nn** function as a list (Figure 27).

```
def difference(newlist):
    '''Simply accesses the point distances and returns as list'''
    a=[]
    for i in newlist:
        a.append(i[1])
    return a
```

Figure 27: Simple code to return array as a list

The **distSorter** function requires the point distances of the nearest neighbours, longitude, latitude and magnitude data as its arguments. The function is designed so that only the longitudes, latitudes and magnitudes of earthquakes are returned if the point difference between the nearest neighbour is over a specified value (Figure 28). In testing a distance of 6 was found to be most optimal for display whilst not conflicting with other labels.

```
def distSorter(x,y,z,u):
    '''Essentially returns the long,lat and mag of points over specified distance'''
    p=[]
    c=[]
    d=[]
    for i,j,k,l in zip(x, y,z,u):
        if i >6:
            p.append(j)
            c.append(k)
            d.append(l)
    return p, c,d
```

Figure 28: Specified point distance function

A final function known as **behemoth** was used to run each individual sub function developed as one, in order to allow easy magnitude labelling (Figure 29).

```
def behemoth(i,j,longitude,latitude,magnitude):
    '''Combines numerous functions to allow aesthetic magnitude labelling for quakes'''
    selection = zip(longitude[i:j], latitude[i:j])
    myarray=np.array(selection)
    nn1=nn(myarray)
    nnlist=list(nn1)
    index1= map(operator.itemgetter(0), nnlist)
    new=difference(index1)
    a1=distSorter(new,longitude[i:j],latitude[i:j],magnitude[i:j])[0]
    a2= distSorter(new,longitude[i:j],latitude[i:j],magnitude[i:j])[1]
    a3= distSorter(new,longitude[i:j],latitude[i:j],magnitude[i:j])[2]
    labelMag(a1,a2,a3,4)
```

Figure 29: Combined function to enable clear magnitude labelling.

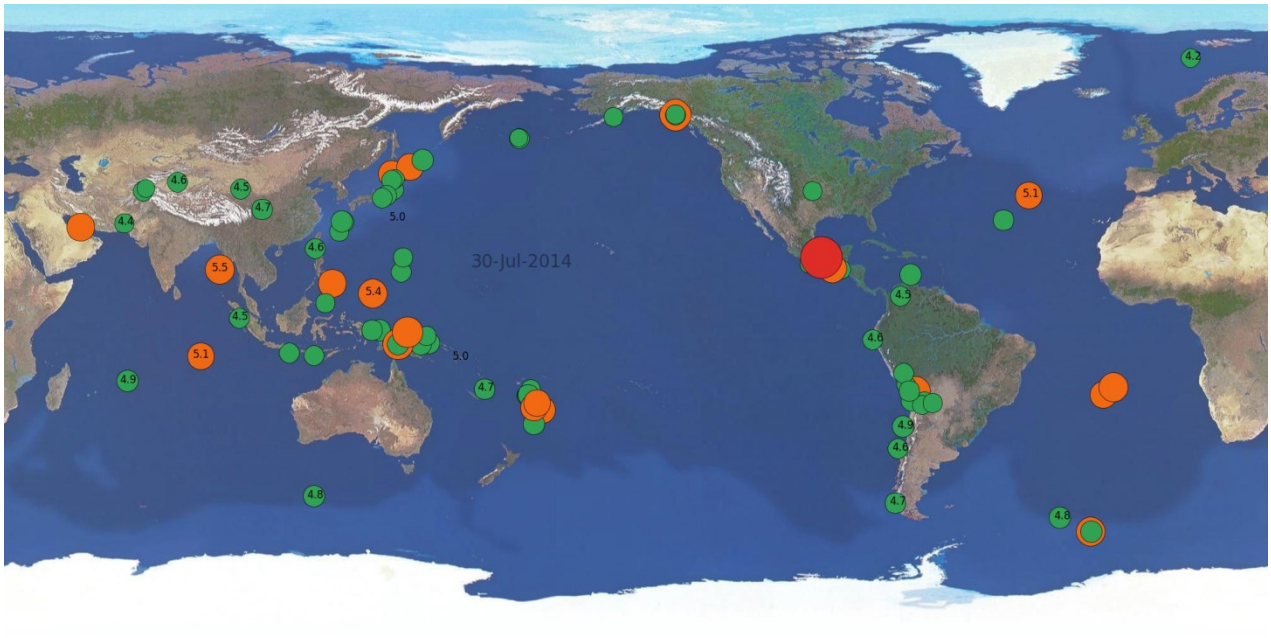

Figure 30: Earthquake visualisation with dynamic magnitude labelling

The overall result of these functions was successful in terms of implementation (Figure 30), but in reality was not particularly appropriate for the earthquake visualisation. The labelling of some earthquakes and the omission of others led to a disjointed final output where users were more likely to be concerned as to why some quakes were labelled and why others weren't instead of focusing on the overall visualisation. This function could be used to only show large earthquakes but it would likely suffer from the same issues as before, with later quakes in close proximity blocking the numbering .Whilst not ideal in this situation, the underlying functionality is sound, and the functions developed would be ideal for numbered or text based labelling of any coordinate based phenomena.

Alternative visualisations

The scripting of the global earthquake visualisation was done in a way which encouraged modularity whilst being specialised to allow optimal visualisation of earthquakes. After finishing the script it was easy to see that the script could be adapted to allow for alternative data sets, with customisable options for general users not necessarily versed in programming. To this end, a new script was created which relies heavily upon user submitted commands via a configuration text file.

The script reuses much of the code developed for the quake visualisation, with a few vital additions. The major aspects of this new script, is the ability to use any CSV file either online or on a local drive, use of a base image or WMS image with layer capabilities and to allow user based plotting indices with varying colours, symbols etc. Whilst primarily following the same framework as the previous script, this one differs in that user's must define certain variables before running the program. In its most basic form, these are a CSV file (online/offline), the column numbers of the latitude and longitude of a phenomena, a base image(local/WMS), colour , marker type and size of a point and a title. Additional options can be specified if users wish to segregate their data into different categories with varying symbology etc. The script supports up to three divisions of the original source data.

A small number of functions were created to accommodate this new user friendly script. The **locWeb** function simply tells the script whether the user is using an online CSV or a local CSV (Figure 31). The configuration text file, which holds the key variables etc. offers the opportunity to select online or offline modes. If no file location is typed into the offline setting, the function will automatically select the online based CSV file.

```
def locweb(lats,lons,lons2,x,y):
    '''Essentially used to determine whther the csv file being used is web/locally based
    and subsequently deals with either type'''


    if not '.csv' in data:
        quakes1 = urllib2.urlopen(quakes)
        csv_data =csv.reader(quakes1)
        next(csv_data)
        for row in csv_data:
            lats.append(float(row[x]))
            lons.append(float(row[y]))
            lons2.append(float(row[y]))

    else:
        with open(data) as f:
            csv_data =csv.reader(f)
            next(csv_data)
            try:
                for row in csv_data:
                #print row
                    lats.append(float(row[x]))
                    lons.append(float(row[y]))
                    lons2.append(float(row[y]))
            except:
                    insurance(data)
                    with open(safety) as f:
                        csv_data =csv.reader(f)
                        next(csv_data)
                        for row in csv_data:
                            print row
                            lats.append(float(row[x]))
                            lons.append(float(row[y]))
                            lons2.append(float(row[y]))
```

Figure 31: Function to determine whether a local or online based csv is used.

The **proj1** function is a slight adaptation of the original **proj** function, which simply selects a WMS image for the projection if a WMS URL is specified, else it will select a user inputted local image, or a default base image (Figure 32).

```
def proj1(a):
    '''Sets the equirectangular projection, and ensures correct start/end longitudes'''

    if  'http://' in serverurl:
        #ensures WMS layer used if in config
        m = Basemap(projection='cyl',llcrnrlat=-90,urcrnrlat=90,\
        llcrnrlon=-180,urcrnrlon=180,resolution='c')
        m.wmsimage(serverurl,layers=[layers1],xpixels=2000,ypixels=1000)
        return m

    else:
        #ensures base image used if wanted or no WMS present
        m = Basemap(projection='cyl',llcrnrlat=-90,urcrnrlat=90,\
        llcrnrlon=(a),urcrnrlon=360+(a),resolution='c')
        m.warpimage(image=basei)
        return m
```

Figure 32: Slight adaptation from previous **proj** function to allow for WMS selections.

The **plotting** function is again a simple adaptation of the previous **mapPlot** function. It differs in allowing users to separate the source data into 3 divisions. The function checks to see whether a user has inputted indices for separation and if so, plots each separate selection of data accordingly. If no indices are specified, all data is plotted as one with the user specified settings for colour, size and symbology (Figure 33).

```
def plotting(lons,lats,m):
    '''Simply plots points depending on whether indicies are used or not.
    Simpler in functionality than mapPlot, as designed for widespread use
    not bound to earthquakes'''

    if 'NA' in plot1:
        for lon, lat in zip(lons,lats):
            x,y = m(lon, lat)
            z,j =m(lon+360,lat)
            m.plot(x, y, colour,marker=mark,alpha=1, markersize=int(msize))
            m.plot(z, j, colour,marker=mark,alpha=1, markersize=int(msize))
    else:
        for lon, lat in zip(lons[int(plot1):int(plot2)],lats[int(plot1):int(plot2)]):
            x,y = m(lon, lat)
            z,j =m(lon+360,lat)
            m.plot(x, y, colour1,marker=mark1,alpha=1, markersize=int(msize1))
            m.plot(z, j, colour1,marker=mark1,alpha=1, markersize=int(msize1))

    if 'NA' in plot3:
        if 'NA' in plot1:
            for lon, lat in zip(lons,lats):
                x,y = m(lon, lat)
                z,j =m(lon+360,lat)
                m.plot(x, y, colour,marker=mark,alpha=1, markersize=int(msize))
                m.plot(z, j, colour,marker=mark,alpha=1, markersize=int(msize))
    else:
        for lon, lat in zip(lons[int(plot3):int(plot4)],lats[int(plot3):int(plot4)]):
            x,y = m(lon, lat)
            z,j =m(lon+360,lat)
            m.plot(x, y, colour2,marker=mark2,alpha=1, markersize=int(msize2))
            m.plot(z, j, colour2,marker=mark2,alpha=1, markersize=int(msize2))
#
    if 'NA' in plot5:
        if 'NA' in plot3:
            if 'NA' in plot1:
                for lon, lat in zip(lons,lats):
                    x,y = m(lon, lat)
                    z,j =m(lon+360,lat)
                    m.plot(x, y, colour,marker=mark,alpha=1, markersize=int(msize))
                    m.plot(z, j, colour,marker=mark,alpha=1, markersize=int(msize))
    else:
        for lon, lat in zip(lons[int(plot5):int(plot6)],lats[int(plot5):int(plot6)]):
            x,y = m(lon, lat)
            z,j =m(lon+360,lat)
            m.plot(x, y, colour3,marker=mark3,alpha=1, markersize=int(msize3))
            m.plot(z, j, colour3,marker=mark3,alpha=1, markersize=int(msize3))
```

Figure 33: Adapted plotting function

Example Visualisations

The visualisation in Figure 34 is using a local based CSV file for European Starling populations and is incorporating a live WMS image with a topographic layer. One problem discovered with using WMS, is an incompatibility with the **sorting1** and **sorting 2** functions which determine optimal starting longitudes. This is because the starting positions of the image cannot be changed when directly accessing a WMS image. If a static data set is being used and has no clipping this is not an issue. If data is being clipped at the edges then a local base image should be used instead. Further work is required to find a solution for optimising WMS image settings.

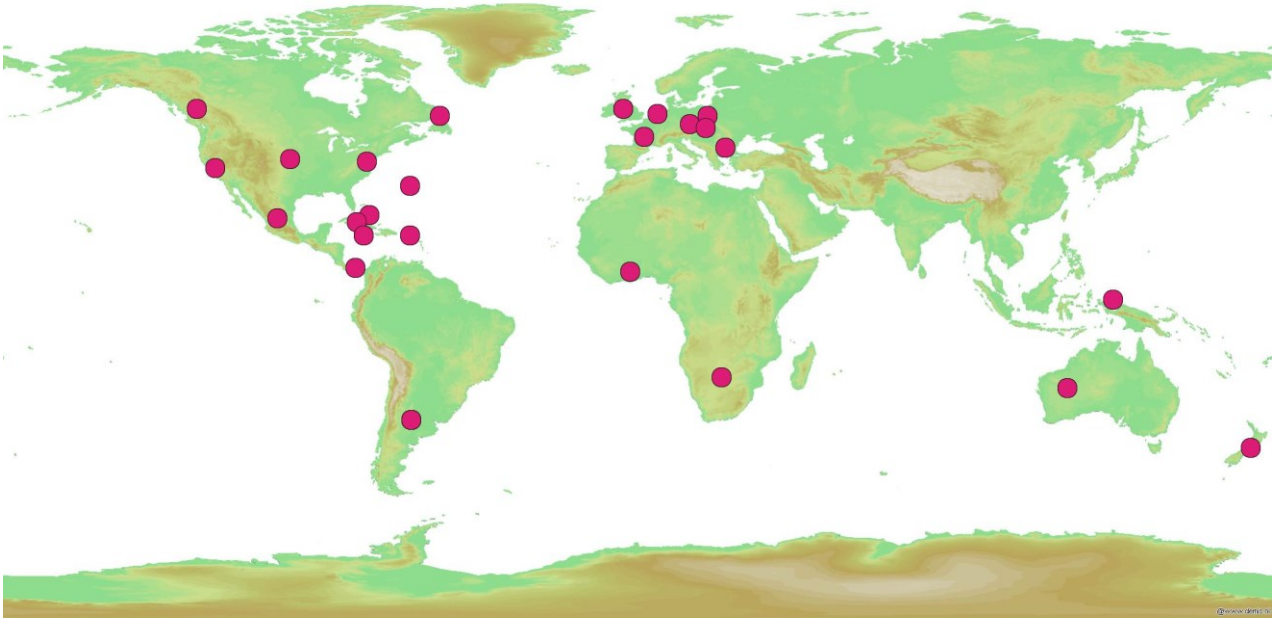

Figure 34: This visualisation is showing the distribution of Sturnus vulgaris populations (European Starlings) across the world.

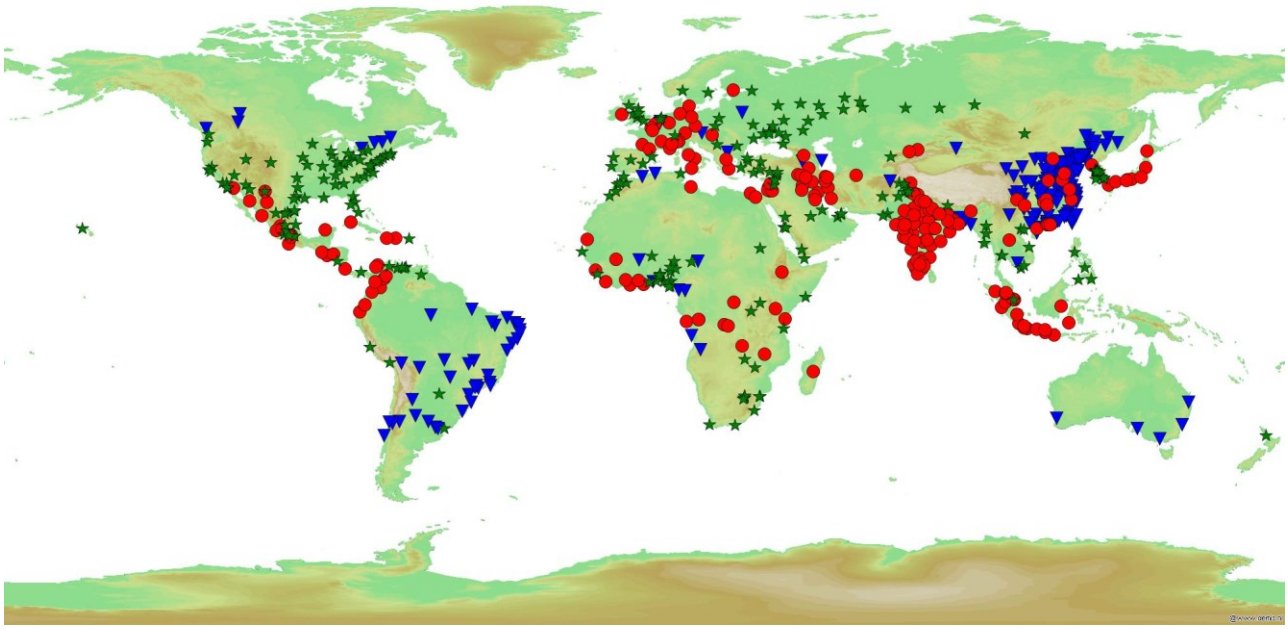

Figure 35: This test visualisation is displaying global city locations in various colours and symbols. It is a simple visualisation designed to ensure key customisable settings such as symbol colours, size and type are all working correctly.

As seen in Figures 36 and 37, the text file is fully commented with the necessary instructions for operation. The primary settings are a data file, image, coordinate locations and save directories. The following figures, detail the code required both in the configuration file and the code needed for use in Python (Figure 38).

```
[myvars]

localdata: C:\Users\Callum\Desktop\Translate\city.csv
#Use this for local data set, leave blank if connecting to URL csv. Must have header then data, no blank rows between

webdata:

lat: 4
#latitude column number(First Column = 0) latitude and longitude must have no capitals in csv file.

lon: 5
#longitude column number

baseimage: C:\Users\Callum\Desktop\Puffersphere\base.jpg
#local image dir

wmsserver: http://www2.demis.nl/WMS/wms.ashx?wms=WorldMap&
# WMS server, if data is directly on border of image and being cut off, use Baseimage. Local images lat long can be altered WMS will not.

layers: Borders,Coastlines,Topography
# WMS Layers

base image example = C:\Users\Callum\Desktop\Translate\base.jpg
online csv example = http://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/all_week.csv

frames: 3400
#Quality/frames/number of images


savedir: C:\Users\Callum\Desktop\Translate\images
#saved image directory

viddir: C:\Users\Callum\Desktop\Translate\vids
#video directory

outputvid: finaloutput.mp4
#save video name

title: Invasive Sparrows/Swallows
#intro title
```

Figure 36 shows the configuration text file required to run the script for alternative data sets.

```
#Default Settings for plotting all events as one

marker: o
#Marker Type: o=circle, v=triangle, s=square, *=star,(http://matplotlib.org/api/markers_api.html)
markersize: 22
colour: #dd1c77
#Colour of Marker, hexdecimal colours require # symbol, else use green,red ,blue etc.

#Multiple Plots

plot1: 0
plot2: 200
colour1:blue
marker1:v
markersize1:15

#USE NA for plot sections if not using them

plot3: 200
plot4: 400
colour2:red
marker2:o
markersize2:15

plot5: 400
plot6: 650
colour3:green
marker3:*
markersize3:15

#plot 1 to plot 2 = range of lat,lons of an event e.g. 1st earthquake to 20th.
#plot 3-4 and 5-6 follow this pattern as well.

safetycopy: C:\Users\Callum\Desktop\Translate\safety.csv
```

Figure 37 shows the customisable settings for plotting data

```
config = ConfigParser.ConfigParser()
con=config.read("config.txt")
savedir = config.get("myvars", "savedir")
viddir = config.get("myvars", "viddir")
title = config.get("myvars", "title")
data = config.get("myvars", "localdata")
quakes =config.get("myvars", "webdata")
latcol =config.get("myvars", "lat")
longcol =config.get("myvars", "lon")
basei =config.get("myvars", "baseimage")
serverurl=config.get("myvars", "wmsserver")
layers1=config.get("myvars", "layers")
colour=config.get("myvars", "colour")
colour1=config.get("myvars", "colour1")
colour2=config.get("myvars", "colour2")
colour3=config.get("myvars", "colour3")

mark=config.get("myvars", "marker")
mark1=config.get("myvars", "marker1")
mark2=config.get("myvars", "marker2")
mark3=config.get("myvars", "marker3")

msize=config.get("myvars", "markersize")
msize1=config.get("myvars", "markersize1")
msize2=config.get("myvars", "markersize2")
msize3=config.get("myvars", "markersize3")

plot1=config.get("myvars", "plot1")
plot2=config.get("myvars", "plot2")
plot3=config.get("myvars", "plot3")
plot4=config.get("myvars", "plot4")
plot5=config.get("myvars", "plot5")
plot6=config.get("myvars", "plot6")

outputvid=config.get("myvars", "outputvid")
val=config.get("myvars", "frames")
safety = config.get("myvars", "safetycopy")
val=int(val)
```

Figure 38: Illustrates the necessary code for taking variables from a configuration file and applying them to the core Python script. This could be adapted and developed in a number of ways in both functionality and customisation.

References

Bertin, J. (1978) Theory of communication and theory of 'the graphic'. *The international yearbook of Cartography Vol. 18, pp 118-126*

Card, S., Mackinlay, J., Shneiderman, B and Kaufmann, M. (1999) Readings in information visualisation: Using vision to think. Morgan Kaufmann Publishers Inc. San Fransicsco, CA.

FFMPEG, (2014) Accessed from *https://ffmpeg.org/ffmpeg.html*, on the 25th June, 2014.

Koch, W. (2001) Jacques Bertin's theory of graphics and its development and influence on multimedia cartography. *Information Design Journal, Vol. 10, pp. 37-43.*

Lloyd, R. (2003) Visual search processes used in map reading. *Cartographica, vol. 34, pp. 11-12*

MacEachren, A. (2004) How maps work: representation, visualisation and design. The Guilford Press, New York.

Matplotlib, (2014) Accessed from http://matplotlib.org/basemap/users/geography.html, on the 25th June 2014.

Monmonier, M. (1996) How to Lie with Maps. University of Chicago Press.

PythonProgramming, (2014) Geographical plotting with Basemap tutorial. Accessed from http://pythonprogramming.net/geographical-plotting-basemap-tutorial/, on the 25th June, 2014.

Scipy, (2014) Accessed from http://www.scipy.org/, on the 25th June, 2014.

Stack Overflow, (2014) Accessed from http://stackoverflow.com/questions/3940128/how-can-i-reverse-a-list-in-python on the 25th June, 2014.

Tosi, S, (2014). Matplotlib for Python Developers. Packt Publishing, Birmingham.

Tufte, E. (1978). The visual display of quantitative information. Graphics Press, Cheshire, Connecticut

United States Geological Survey, (2014). Earthquakes Hazards Program. Accessed from http://earthquake.usgs.gov/earthquakes/feed/v1.0/csv.php, on the 25th June, 2014.

Williams, R. (2003). The non-designers design book, second edition. Peachpit Press, Berkeley, CA.

Appendices

Appendix A: Required Programs for running earthquake visualisation

Required Programs

Program	Purpose	Link
Python x,y (ver. 2.7.3.1)	Programming language	https://code.google.com/p/pythonxy/wiki/Downloads?tm=2
Basemap 1.07	Mapping Module	http://matplotlib.org/basemap/
Numpy 1.8.1	Numeric module required	http://sourceforge.net/projects/numpy/files/NumPy/1.8.1/numpy-1.8.1-win32-superpack-python2.7.exe/download
OWSLib-0.87	Allows use of WMS	https://pypi.python.org/pypi/OWSLib
ImageMagick (Ver. 6.8.8-10)	Image manipulation software	http://www.imagemagick.org/download/windows/
FFMPEG (ver. M-50911 – g9efcfbe)	Video creation software	https://www.ffmpeg.org/

Appendix B: Installation procedure for Earthquake and alternative visualisations

Installation Procedure for Earthquake Visualisation

Ensure required programs are all installed first.

Step 1: Open config1.txt file

Step 2: Ensure savedir is targeted to images folder within Puffersphere e.g.

M:\DissFinal\Puffersphere\images

Step 3: Ensure viddir is targeted to vids folder within Puffersphere e.g.

M:\DissFinal\Puffersphere\vids

Step 4: Ensure safety copy is targeted to Puffersphere folder (file will be created when run)

Step 5(Optional): Change output video name

Step 6: Double click driver.py /Run via CMD line or integrate into batch script, to run program.

Installation Procedure for Alternative Visualisation

Exact same procedure, except provide locations for alternative base images/data within Puffersphere folder.

Appendix C: Primary driver code for earthquake visualisation(driver.py)

```
# -*- coding: utf-8 -*-
"""
Created on Thu May 01 12:26:58 2014

@author: Callum
"""
import matplotlib.pyplot as mp
from mpl_toolkits.basemap import Basemap
import dateutil.parser
import urllib2
import urllib
import csv
import numpy as np
import operator
import os
import subprocess
import glob
import shutil
import scipy.spatial
import time
import sys
from functions import *
import ConfigParser
from collections import OrderedDict

np.set_printoptions(threshold='nan')
horse=time.clock()

config = ConfigParser.ConfigParser()
con=config.read("config1.txt")
savedir = config.get("myvars", "savedir")
viddir = config.get("myvars", "viddir")
outputvid = config.get("myvars", "outputvid")
safety = config.get("myvars", "safetycopy")

#RESET FOLDERS AND FILES FOR REPEATED USE


deleteDir(savedir)
deleteJpg(savedir)
deleteMp4(viddir,outputvid)


#DOWNLOADING CSV'S AND PARSING

#data = urllib2.urlopen('http://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/all_day.csv')
data = urllib2.urlopen('http://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/all_week.csv')
#data = urllib2.urlopen('http://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/all_month.csv')
#data=urllib2.urlopen('http://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/significant_month.csv')
urllib.urlretrieve('http://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/all_week.csv','safety.csv')

latst =[]
lonst =[]
lons2t=[]
magt = []
placet =[]
timet = []
lons3t = []
lons4t =[]
lemont=[]

parsing(data,safety,latst,lonst,lons2t,lons3t,lons4t,lemont,magt,placet,timet)
```

```
#Runs parsing function

lats=latst[::-1]
lons=lonst[::-1]
lons2=lons2t[::-1]
lons3=lons3t[::-1]
lons4=lons4t[::-1]
lemon=lemont[::-1]
mag=magt[::-1]
place=placet[::-1]
time1=timet[::-1]

#Essentially reverses the data so that older data will be plotted first
print max(mag)

#SORTING FUNCTIONS FOR LAT/LONG OF FIGURES

#Settings to determine starting and ending longitude

a=sorting2(sorting1(lons)[1],sorting1(lons)[0])
print a

#Sets the Projection type
m=proj(a)


figure = mp.gcf()

#Save directories for each day of earthquakes within the week.
testdir1= savedir+str("\\a001.jpg")
testdir2= savedir+str("\\a002.jpg")
testdir3= savedir+str("\\a003.jpg")
testdir4= savedir+str("\\a004.jpg")
testdir5= savedir+str("\\a005.jpg")
testdir6= savedir+str("\\a006.jpg")
testdir7= savedir+str("\\a007.jpg")
testdir8= savedir+str("\\a008.jpg")
testdir9= savedir+str("\\a009.jpg")
#testdir10= savedir+str("\\a010.jpg")

p=time3(time1)

c=uDay(p)
c= list(OrderedDict.fromkeys(c))
#retrieves the count/number of times a date appears
#i.e. tells how many quakes for each day
d= list(OrderedDict.fromkeys(p))
print c
print d

day1=c[0]
day2=day1+c[1]
day3=day2+c[2]
day4=day3+c[3]
day5=day4+c[4]
day6=day5+c[5]
day7=day6+c[6]
day8=day7+c[7]
```

```
121
122
123 #general figure settings
124 figure.frameon="false"
125 mp.axis('off')
126 figure.set_size_inches(20,10)
127 mp.tight_layout(pad=0)
128 image2='base.jpg'
129
130 #Figures
131
132 m1=3
133 m2=4.5
134 m3=5.5
135 m4=6.5
136 min_marker_size=7
137 msize = m1 * min_marker_size
138 msize2 = m2 * min_marker_size
139 msize3 = m3 * min_marker_size
140 msize4= m4 * min_marker_size
141
142
143 print testdir2
144
145 #plots quakes for day1
146 plotIndex(lons2,lats,mag,day1,m,0.5)
147 plotIndex2(lons2,lats,mag,0,day1,m,1.8)
148 #Plots inbuilt legend,text, date info
149
150 mp.text(165,10,("%s"  % d[0]),color='black',size='16',alpha=1)
151 m.plot(210,20,'#e6f413',marker='o',markersize=msize)
152 m.plot(210,11,'#F94D00',marker='o',markersize=msize2)
153 m.plot(210,0,'#e7298a',marker='o',markersize=msize3)
154 m.plot(210,-13,'#e31a1c',marker='o',markersize=msize4)
155 mp.text(208,19,(">4"),color='k',size='13')
156 mp.text(208,10,("5-6"),color='k',size='14')
157 mp.text(208,0,("6+"),color='k',size='14')
158 mp.text(208,-13,("7+"),color='k',size='14')
159 mp.text(198,26,("Magnitude"),color='k',size='18')
160
161 mp.savefig(testdir2,dpi=100)
162 #saves image
163 mp.clf()
164 #clears figure
165 m.warpimage(image=image2)
166 #replots baseimage
167 #behemoth(0,200)
168
169
170 plotIndex(lons2,lats,mag,day2,m,0.5)
171 plotIndex2(lons2,lats,mag,day1,day2,m,1.8)
172 #Plots inbuilt legend and text
173 mp.text(165,10,("%s"  % d[1]),color='black',size='16',alpha=1)
174 m.plot(210,20,'#e6f413',marker='o',markersize=msize)
175 m.plot(210,11,'#F94D00',marker='o',markersize=msize2)
176 m.plot(210,0,'#e7298a',marker='o',markersize=msize3)
177 m.plot(210,-13,'#e31a1c',marker='o',markersize=msize4)
178 mp.text(208,19,(">4"),color='k',size='13')
179 mp.text(208,10,("5-6"),color='k',size='14')
180 mp.text(208,0,("6+"),color='k',size='14')
181 mp.text(208,-13,("7+"),color='k',size='14')
182 mp.text(198,26,("Magnitude"),color='k',size='18')
183
184 mp.savefig(testdir3,dpi=100)
185 mp.clf()
186 m.warpimage(image=image2)
187
```

```
#behemoth(0,400)
plotIndex(lons2,lats,mag,day3,m,0.5)
plotIndex2(lons2,lats,mag,day2,day3,m,1.8)
#Plots inbuilt legend and text
mp.text(165,10,("%s"  % d[2]),color='black',size='16',alpha=1)
m.plot(210,20,'#e6f413',marker='o',markersize=msize)
m.plot(210,11,'#F94D00',marker='o',markersize=msize2)
m.plot(210,0,'#e7298a',marker='o',markersize=msize3)
m.plot(210,-13,'#e31a1c',marker='o',markersize=msize4)
mp.text(208,19,(">4"),color='k',size='13')
mp.text(208,10,("5-6"),color='k',size='14')
mp.text(208,0,("6+"),color='k',size='14')
mp.text(208,-13,("7+"),color='k',size='14')
mp.text(198,26,("Magnitude"),color='k',size='18')

mp.savefig(testdir4,dpi=100)
mp.clf()
m.warpimage(image=image2)

#behemoth(0,600)
plotIndex(lons2,lats,mag,day4,m,0.5)
plotIndex2(lons2,lats,mag,day3,day4,m,1.8)
#Plots inbuilt legend and text
mp.text(165,10,("%s"  % d[3]),color='black',size='16',alpha=1)
m.plot(210,20,'#e6f413',marker='o',markersize=msize)
m.plot(210,11,'#F94D00',marker='o',markersize=msize2)
m.plot(210,0,'#e7298a',marker='o',markersize=msize3)
m.plot(210,-13,'#e31a1c',marker='o',markersize=msize4)
mp.text(208,19,(">4"),color='k',size='13')
mp.text(208,10,("5-6"),color='k',size='14')
mp.text(208,0,("6+"),color='k',size='14')
mp.text(208,-13,("7+"),color='k',size='14')
mp.text(198,26,("Magnitude"),color='k',size='18')

mp.savefig(testdir5,dpi=100)
mp.clf()
m.warpimage(image=image2)

#behemoth(0,800)
plotIndex(lons2,lats,mag,day5,m,0.5)
plotIndex2(lons2,lats,mag,day4,day5,m,1.8)
#Plots inbuilt legend and text
mp.text(165,10,("%s"  % d[4]),color='black',size='16',alpha=1)
m.plot(210,20,'#e6f413',marker='o',markersize=msize)
m.plot(210,11,'#F94D00',marker='o',markersize=msize2)
m.plot(210,0,'#e7298a',marker='o',markersize=msize3)
m.plot(210,-13,'#e31a1c',marker='o',markersize=msize4)
mp.text(208,19,(">4"),color='k',size='13')
mp.text(208,10,("5-6"),color='k',size='14')
mp.text(208,0,("6+"),color='k',size='14')
mp.text(208,-13,("7+"),color='k',size='14')
mp.text(198,26,("Magnitude"),color='k',size='18')

mp.savefig(testdir6,dpi=100)
mp.clf()
m.warpimage(image=image2)
```

```
244
245 #behemoth(0,1000)
246 plotIndex(lons2,lats,mag,day6,m,0.5)
247 plotIndex2(lons2,lats,mag,day5,day6,m,1.8)
248 #Plots inbuilt legend and text
249 mp.text(165,10,("%s"  % d[5]),color='black',size='16',alpha=1)
250 m.plot(210,20,'#e6f413',marker='o',markersize=msize)
251 m.plot(210,11,'#F94D00',marker='o',markersize=msize2)
252 m.plot(210,0,'#e7298a',marker='o',markersize=msize3)
253 m.plot(210,-13,'#e31a1c',marker='o',markersize=msize4)
254 mp.text(208,19,(">4"),color='k',size='13')
255 mp.text(208,10,("5-6"),color='k',size='14')
256 mp.text(208,0,("6+"),color='k',size='14')
257 mp.text(208,-13,("7+"),color='k',size='14')
258 mp.text(198,26,("Magnitude"),color='k',size='18')
259
260 mp.savefig(testdir7,dpi=100)
261 mp.clf()
262 m.warpimage(image=image2)
263
264 #behemoth(0,1200)
265
266 plotIndex(lons2,lats,mag,day7,m,0.5)
267 plotIndex2(lons2,lats,mag,day6,day7,m,1.8)
268 #Plots inbuilt legend and text
269 mp.text(165,10,("%s"  % d[6]),color='black',size='16',alpha=1)
270 m.plot(210,20,'#e6f413',marker='o',markersize=msize)
271 m.plot(210,11,'#F94D00',marker='o',markersize=msize2)
272 m.plot(210,0,'#e7298a',marker='o',markersize=msize3)
273 m.plot(210,-13,'#e31a1c',marker='o',markersize=msize4)
274 mp.text(208,19,(">4"),color='k',size='13')
275 mp.text(208,10,("5-6"),color='k',size='14')
276 mp.text(208,0,("6+"),color='k',size='14')
277 mp.text(208,-13,("7+"),color='k',size='14')
278 mp.text(198,26,("Magnitude"),color='k',size='18')
279
280 mp.savefig(testdir8,dpi=100)
281 mp.clf()
282 m.warpimage(image=image2)
283
284 #behemoth(0,200000,lons2,lats,mag)
285
286 plotIndex(lons2,lats,mag,3000,m,0.5)
287 plotIndex2(lons2,lats,mag,day7,day8,m,1.8)
288 #Plots inbuilt legend and text
289 mp.text(165,10,("%s"  % d[7]),color='black',size='16',alpha=1)
290 m.plot(210,20,'#e6f413',marker='o',markersize=msize)
291 m.plot(210,11,'#F94D00',marker='o',markersize=msize2)
292 m.plot(210,0,'#e7298a',marker='o',markersize=msize3)
293 m.plot(210,-13,'#e31a1c',marker='o',markersize=msize4)
294 mp.text(208,19,(">4"),color='k',size='13')
295 mp.text(208,10,("5-6"),color='k',size='14')
296 mp.text(208,0,("6+"),color='k',size='14')
297 mp.text(208,-13,("7+"),color='k',size='14')
298 mp.text(198,26,("Magnitude"),color='k',size='18')
299
300 mp.savefig(testdir9,dpi=100)
301 mp.clf()
302 m.warpimage(image=image2)
303
```

```
304
305 #INTRO SLIDE SETTINGS
306
307 #general settings for plotting titles, lables and legends etc.
308 mp.clf()
309 #basic projection for intro slide
310 m= Basemap(projection='cyl',llcrnrlat=-90,urcrnrlat=90,\
311     llcrnrlon=-180,urcrnrlon=180,resolution='c')
312 m.warpimage(image="title2.jpg")
313 m1=3
314 m2=5
315 m3=6
316 m4=7.5
317 min_marker_size=8
318 msize = m1 * min_marker_size
319 msize2 = m2 * min_marker_size
320 msize3 = m3 * min_marker_size
321 msize4= m4 * min_marker_size
322
323
324 m.plot(25,-10,'#e6f413',marker='o',markersize=msize)
325 m.plot(45,-10,'#F94D00',marker='o',markersize=msize2)
326 m.plot(65,-10,'#e7298a',marker='o',markersize=msize3)
327 m.plot(85,-10,'#e31a1c',marker='o',markersize=msize4)
328
329 m.plot(-165,-10,'#e6f413',marker='o',markersize=msize)
330 m.plot(-145,-10,'#F94D00',marker='o',markersize=msize2)
331 m.plot(-125,-10,'#e7298a',marker='o',markersize=msize3)
332 m.plot(-105,-10,'#e31a1c',marker='o',markersize=msize4)
333
334 figure.frameon="false"
335 mp.axis('off')
336
337 m.warpimage(image="title2.jpg")
338
339 mp.text(-177,30,("Global Earthquakes"),color='w',size='36')
340 mp.text(-177,20,("from %s to"  % p[0]),color='w',size='30')
341 mp.text(-177,10,("%s" %  p[-1]),color='w',size='30')
342 mp.text(-80,30,("Callum Scougal"),color='w',size='24')
343 mp.text(-80,20,("GIS"),color='w',size='24')
344 mp.text(-80,10,("School of GeoSciences"),color='w',size='24')
345
346 mp.text(10,30,("Global Earthquakes"),color='w',size='36')
347 mp.text(10,20,("from %s to"  % p[0]),color='w',size='30')
348 mp.text(10,10,("%s" %  p[-1]),color='w',size='30')
349 mp.text(110,30,("Callum Scougal"),color='w',size='24')
350 mp.text(110,20,("GIS"),color='w',size='24')
351 mp.text(110,10,("School of GeoSciences"),color='w',size='24')
352
353 mp.text(25,0,("Magnitude"),color='w',size='24')
354 mp.text(20,-22,("> 4"),color='w',size='18')
355 mp.text(41,-22,("5-6"),color='w',size='18')
356 mp.text(63,-22,("6+"),color='w',size='18')
357 mp.text(84,-22,("7+"),color='w',size='18')
358
359 mp.text(-160,0,("Magnitude"),color='w',size='24')
360 mp.text(-170,-22,("> 4"),color='w',size='18')
361 mp.text(-148,-22,("5-6"),color='w',size='18')
362 mp.text(-129,-22,("6+"),color='w',size='18')
363 mp.text(-109,-22,("7+"),color='w',size='18')
364 mp.savefig(testdir1,dpi=100)
365 #
366
367 print max(mag)
```

```
368
369 #ROLLING IMAGES AND MP4 Creation
370
371 val=3400
372 #number of images or frames required for final video
373 #test= viddir+str('//0001.mp4')
374 #runs vid function for 3400 images across image 2000 pixels wide.
375 #saves images in images folder, vids in vids folder
376 test= viddir+str('//0001.mp4')
377 vid(testdir1,val,float(2000),'0001.jpg',savedir,viddir+str('//0001.mp4'))
378 vid(testdir2,val,float(2000),'0002.jpg',savedir,viddir+str('//0002.mp4'))
379 vid(testdir3,val,float(2000),'0003.jpg',savedir,viddir+str('//0003.mp4'))
380 vid(testdir4,val,float(2000),'0004.jpg',savedir,viddir+str('//0004.mp4'))
381 vid(testdir5,val,float(2000),'0005.jpg',savedir,viddir+str('//0005.mp4'))
382 vid(testdir6,val,float(2000),'0006.jpg',savedir,viddir+str('//0006.mp4'))
383 vid(testdir7,val,float(2000),'0007.jpg',savedir,viddir+str('//0007.mp4'))
384 vid(testdir8,val,float(2000),'0008.jpg',savedir,viddir+str('//0008.mp4'))
385 vid(testdir9,val,float(2000),'0009.jpg',savedir,viddir+str('//0009.mp4'))
386 vid(testdir10,val,float(2000),'0010.jpg',savedir,viddir+str('//0010.mp4'))
387
388 makemylist(viddir)
389 #Function finds every mp4 file and adds name to a list
390
391 concat=subprocess.Popen(["ffmpeg", "-f", "concat", "-i","mylist.txt", "-c", "copy","-y", "output.mp4"])#,shell = True
392 concat.wait()
393 #THis FFMPEG subprocess combines all mp4's as one
394
395 fast=subprocess.Popen(["ffmpeg", "-i", "output.mp4", "-vf","setpts=0.3*PTS","-y", outputvid])#,shell = True
396 fast.wait()
397 #Command to alter speed/runtime of final video
398
399
400 finish = (time.clock()-horse)
```

Appendix D: Functions developed for earthquake visualisation (functions.py)

```
# -*- coding: utf-8 -*-
"""
Created on Tue Jun 24 10:52:23 2014

@author: Callum
"""
import matplotlib.pyplot as mp
from mpl_toolkits.basemap import Basemap
import dateutil.parser
import urllib2
import csv
import numpy as np
import operator
import os
import subprocess
import glob
import shutil
import scipy.spatial
import time
import sys


def deleteDir(save):
    '''Deletes any Dir's beginning with framesFor, ensures new data always used'''
    path =[x[0] for x in os.walk(save)]
    #peach =apple[0]
    #print apple[2:66]
    path1=path[1:66]
    print path1
    for f in path1:
        if 'framesFor' in f:
            shutil.rmtree(f)

def deleteJpg(save):
    '''Deletes all images within save directory, ensures new images always used'''
    files = glob.glob(save+str('/*.jpg'))
    for f in files:
        if not 'base.jpg' in f:
            if not 'intro.jpg' in f:
                os.remove(f)

def deleteMp4(vidloc,outputvid):
    '''Deletes all mp4's in vid dir, ensures new vids always used'''
    files2 = glob.glob(vidloc+str('/*.mp4'))
    for g in files2:
        if not 'output.mp4' in g:
            if not outputvid in g:
                os.remove(g)

```

```
50
51 def parsing(data1,safety,latst,lonst,lons2t,lons3t,lons4t,lemont,magt,placet,timet):
52     '''Parses CSV data, if any errors with online data, downloads it and repairs any blank values'''
53     csv_data =csv.reader(data1)
54 #reads as csv
55     next(csv_data)
56
57     try:
58         for row in csv_data:
59                 #print row
60                 latst.append(float(row[1]))
61                 lonst.append(float(row[2]))
62                 lons2t.append(float(row[2]))
63                 lons3t.append(float(row[2]))
64                 lons4t.append(float(row[2]))
65                 lemont.append(float(row[2]))
66                 magt.append(float(row[4]))
67                 placet.append(str(row[13]))
68                 timet.append(row[0])
69     except:
70         insurance(safety)
71         with open(safety) as f:
72             csv_data =csv.reader(f)
73             next(csv_data)
74             for row in csv_data:
75                 print row
76                 latst.append(float(row[1]))
77                 lonst.append(float(row[2]))
78                 lons2t.append(float(row[2]))
79                 lons3t.append(float(row[2]))
80                 lons4t.append(float(row[2]))
81                 lemont.append(float(row[2]))
82                 magt.append(float(row[4]))
83                 placet.append(str(row[13]))
84                 timet.append(row[0])
85
86
87 def insurance(data):
88     '''Takes downloaded copy of data and edits for errors and replaces blank
89     cells with 0's. Ensures visualisation can always be made, if errors with USGS data'''
90
91     output="safety.csv"
92     with open(data, "rb") as infile, open(output, "wb") as outfile:
93         r = csv.DictReader(infile)
94         w = csv.DictWriter(outfile, r.fieldnames)
95         w.writeheader()
96         for row in r:
97             if not row["latitude"].strip():
98                 row["latitude"] = "0"
99             #w.writerow(row)
100             if not row["longitude"].strip():
101                 row["longitude"] = "0"
102             if not row["mag"].strip():
103                 row["mag"] = "0"
104             w.writerow(row)
105
106
107 def sorting1(lons):
108     '''Sorts longitudes, finds difference between every point and reports the
109     max difference value and index position as well as partnering index position'''
110
111     lons.sort()
112     v=np.diff(lons)
113     index, value = max(enumerate(v), key=operator.itemgetter(1))
114     return lons[index],lons[index+1], value
```

```
115
116
117 def sorting2(x,y):
118     '''This function takes the difference between the 2 index's and then halfs it.
119     This ensures the maximum distance between the 2 points is found and can then be
120     used to set the optimal figure break point. Ultra ignores negative aspects, so this
121     function ensures negative values are negative and postive are positive.'''
122
123     if x <0 and y <0:
124        a=x + y
125        return a/2
126
127     elif x <0 and y >0:
128         a=x + y
129         return a/2
130
131     else:
132         x>0 and y >0
133         a=y + x
134         return a/2
135
136
137 def proj(a):
138     '''Sets the equirectangular projection, and ensures correct start/end longitudes'''
139     if a<0:
140         m = Basemap(projection='cyl',llcrnrlat=-90,urcrnrlat=90,\
141         llcrnrlon=(a),urcrnrlon=360+(a),resolution='c')
142         m.warpimage(image="base.jpg")
143         #m.warpimage(image="plates.jpg")
144         return m
145     if a>0:
146         m = Basemap(projection='cyl',llcrnrlat=-90,urcrnrlat=90,\
147         llcrnrlon=(a),urcrnrlon=360+(a),resolution='c')
148         m.warpimage(image="base.jpg")
149        # m.warpimage(image="plates.jpg")
150         return m
151
```

```
152
153 def mapPlot(lons2,lats,mag,alpha1,m,thick):
154     '''Takes lat, long and mag and plots quake events on map, varying
155     colour & size for magnitude'''
156
157     for lon, lat, mag in zip(lons2, lats, mag):
158         mp.axis('off')
159
160         x,y = m(lon+360, lat)
161             #ensures all points plotted when origin map placement is altered.
162         z,j =m(lon,lat)
163 #        msize = mag * 4,5,6.25
164         msize = mag * 2
165         msize1=mag * 3
166         msize2=mag * 4.5
167         msize3=mag * 5
168
169         if mag >4.0 and mag <5.0:
170             m.plot(x, y, '#e6f413',marker="o",alpha=alpha1, markersize=msize,mew=thick)
171             m.plot(z, j, '#e6f413',marker="o",alpha=alpha1, markersize=msize,mew=thick)
172
173         elif mag >5.0 and mag <6.0:
174             m.plot(x, y, '#F94D00',marker="o", alpha=alpha1, markersize=msize1,mew=thick)
175             m.plot(z, j, '#F94D00',marker="o", alpha=alpha1, markersize=msize1,mew=thick)
176
177         elif mag >6.0:
178             m.plot(x, y, '#e7298a',marker="o", alpha=alpha1, markersize=msize2,mew=thick)
179             m.plot(z, j, '#e7298a',marker="o", alpha=alpha1, markersize=msize2,mew=thick)
180
181         elif mag >7.0:
182             m.plot(x, y, '#e31a1c',marker="o", alpha=alpha1, markersize=msize3,mew=thick)
183             m.plot(z, j, '#e31a1c',marker="o", alpha=alpha1, markersize=msize3,mew=thick)
184
185
186
187 def mapPlot2(lons2,lats,mag,alpha1,m,thick):
188     '''Takes lat, long and mag and plots quake events on map, varying
189     colour & size for magnitude'''
190
191     for lon, lat, mag in zip(lons2, lats, mag):
192         mp.axis('off')
193
194         x,y = m(lon+360, lat)
195             #ensures all points plotted when origin map placement is altered.
196         z,j =m(lon,lat)
197 #        msize = mag * 4,5,6.25
198         msize = mag * 2
199         msize1=mag * 3
200         msize2=mag * 4.5
201         msize3=mag * 5
202         me='black'
203         if mag >4.0 and mag <5.0:
204             m.plot(x, y, '#e6f413',marker="o",alpha=alpha1, markersize=msize,mew=thick,mec=me)
205             m.plot(z, j, '#e6f413',marker="o",alpha=alpha1, markersize=msize,mew=thick,mec=me)
206
207         elif mag >5.0 and mag <6.0:
208             m.plot(x, y, '#F94D00',marker="o", alpha=alpha1, markersize=msize1,mew=thick,mec=me)
209             m.plot(z, j, '#F94D00',marker="o", alpha=alpha1, markersize=msize1,mew=thick,mec=me)
210
211         elif mag >6.0:
212             m.plot(x, y, '#e7298a',marker="o", alpha=alpha1, markersize=msize2,mew=thick,mec=me)
213             m.plot(z, j, '#e7298a',marker="o", alpha=alpha1, markersize=msize2,mew=thick,mec=me)
214
215         elif mag >7.0:
216             m.plot(x, y, '#e31a1c',marker="o", alpha=alpha1, markersize=msize3,mew=thick,mec=me)
217             m.plot(z, j, '#e31a1c',marker="o", alpha=alpha1, markersize=msize3,mew=thick,mec=me)
218
219
```

```
def plotIndex(x,y,j,i,m,thick):
    '''Uses mapPlot for specified indiceis within a list, e.g 20-40'''
    if len(x)> i:
        mapPlot(x[0:i],y[0:i],j[0:i],1,m,thick)

    elif len(x) <i:
        mapPlot(x[:],y[:],j[:],1,m,thick)

def plotIndex2(x,y,j,z,i,m,thick):
    '''Uses mapPlot for specified indiceis within a list, e.g 20-40'''
    if len(x)> 1:
        mapPlot2(x[z:i],y[z:i],j[z:i],1,m,thick)

    elif len(x) <i:
        mapPlot2(x[:],y[:],j[:],1,m,thick)

def time3(self):
    '''Formats date information nicely'''
    a=[]
    for x in self:
        d=dateutil.parser.parse(x)
        #c=d.strftime('%d-%m-%Y %H:%M:%S')
        c=d.strftime('%d-%b-%Y')
        a.append(c)
    return a

def uDay(date):
    '''Takes a list of dates and determines how many unique dates are
    present and the number of quakes occuring within each day'''
    c=[]
    for i in date:
        c.append(date.count(i))
    return c


def nn(array):
    '''Function to calculate nearest neighbour between point array
    k=number of neighbours,d_upper=distance)'''
    c=[]
    tree=scipy.spatial.cKDTree(array,leafsize=100)
    for item in array:
        result=tree.query(item,k=2,distance_upper_bound=10)
        c.append(result)
    return c

def difference(newlist):
    '''Simply accesses the point distances and returns as list'''
    a=[]
    for i in newlist:
        a.append(i[1])
    return a

def distSorter(x,y,z,u):
    '''Essentially returns the long,lat and mag of points over specified distance'''
    p=[]
    c=[]
    d=[]
    for i,j,k,l in zip(x, y,z,u):
        if i >6:
            p.append(j)
            c.append(k)
            d.append(l)
    return p, c,d
```

```
283
284 def labelMag(x,y,magnitude,i):
285     '''Adds magnitude numbers to map, i = mag > than'''
286     for  lon, lat,mag in zip( x, y,magnitude):
287         if mag > i:
288             mp.text(lon+358.5,lat-.5,mag)
289             mp.text(lon+-2,lat-.5,mag)
290
291
292 def behemoth(i,j,longitude,latitude,magnitude):
293     '''Combines numerous functions to allow aesthetic magnitude labelling for quakes'''
294     selection = zip(longitude[i:j], latitude[i:j])
295     myarray=np.array(selection)
296     nn1=nn(myarray)
297     nnlist=list(nn1)
298     index1= map(operator.itemgetter(0), nnlist)
299     new=difference(index1)
300     a1=distSorter(new,longitude[i:j],latitude[i:j],magnitude[i:j])[0]
301     a2= distSorter(new,longitude[i:j],latitude[i:j],magnitude[i:j])[1]
302     a3= distSorter(new,longitude[i:j],latitude[i:j],magnitude[i:j])[2]
303     labelMag(a1,a2,a3,4)
304
305
306
307 def vid(x,y,z,name,directory,output):
308     '''Creates dir for images which are rolled via imagemagick subprocess,
309     images then compiled into video via ffmpeg subprocess'''
310     print x
311
312     route=directory+str('\\framesFor')+str(name)+str('\\')
313     #simply acts as a path variable to where rolled images are located
314     print route
315     raw = route+str('%d.jpg')
316     #again acts as a path variable to images
317     print raw
318     z1=z/y
319     #This is simply number of frames divided by distance of image(2000 for Puffersphere)
320
321     print "ANIMATION PROCESS"
322     if not os.path.isfile("images\\framesFor"+str(name)) and not os.path.isdir("images\\framesFor"+str(name)):
323         os.mkdir("images\\framesFor"+str(name))
324     #essentially if there is not a  folder for the selection of rolled images, will make folders for each
325
326     frame = 0
327     p=list(xrange(y))
328     #makes list the length of frames specified
329     for i in p:
330         frame=frame+z1
331         #counter used to determine distance for each rolled image
332         a= str(frame)
333         a1='+'+str(a)
334         #print a1
335         o=subprocess.Popen(["convert", x,"-roll",a1, "-scale", "2000x1000", route+str(i)+'.jpg'],shell = True)
336         #This subprocess accesses imageMagick via popen, and carries out the neccessary roll commands.
337         o.wait()
338         #IMP. Tells system to finish this before starting next process. If not present vids made of incomplete images
339
340     c=subprocess.Popen(["ffmpeg", "-f", "image2",
341                             "-i", raw, "-c:v", "libx264",
342                             "-pix_fmt", "yuv420p", "-preset",
343                             "ultrafast", "-tune", "fastdecode",
344                             "-profile:v", "main", "-vf", "scale=2000:1000",
345                             "-r", "30", "-coder", "0", "-g", "6", "-crf", "20",
346                             "-y", output],shell=True)
347     c.wait()
348     #This subprocess accesses FFMPEG and compiles all rolled images within a dir into a mp4
349     #Various settings determine quality, speed and compression.
350
351 #veryslow
352
353 def makemylist(directory):
354     '''Makes a txt list of every mp4 within a directory, to allow concatination of mp4's'''
355     f= os.listdir(directory)
356     test=open("mylist.txt","w")
357     for i in f:
358         if i.endswith('output.mp4'):
359             pass
360         elif i.endswith('.mp4'):
361             test.write('file \'')
362             test.write(directory+str('/'))
363             test.write(i+str('\'') +'\n')
```

Appendix E: Primary driver code for alternative visualisations (driver.py)

```
# -*- coding: utf-8 -*-
"""
Created on Sat Jun 21 14:15:54 2014

@author: Callum
"""
import csv
import ConfigParser
import urllib2
from mpl_toolkits.basemap import Basemap
import matplotlib.pyplot as mp
import subprocess
from functions import *
#from owslib.wms import WebMapService

#Config parser settings
config = ConfigParser.ConfigParser()
con=config.read("config.txt")
savedir = config.get("myvars", "savedir")
viddir = config.get("myvars", "viddir")
title = config.get("myvars", "title")
data = config.get("myvars", "localdata")
quakes =config.get("myvars", "webdata")
latcol =config.get("myvars", "lat")
longcol =config.get("myvars", "lon")
basei =config.get("myvars", "baseimage")
serverurl=config.get("myvars", "wmsserver")
layers1=config.get("myvars", "layers")

colour=config.get("myvars", "colour")
colour1=config.get("myvars", "colour1")
colour2=config.get("myvars", "colour2")
colour3=config.get("myvars", "colour3")

mark=config.get("myvars", "marker")
mark1=config.get("myvars", "marker1")
mark2=config.get("myvars", "marker2")
mark3=config.get("myvars", "marker3")

msize=config.get("myvars", "markersize")
msize1=config.get("myvars", "markersize1")
msize2=config.get("myvars", "markersize2")
msize3=config.get("myvars", "markersize3")

plot1=config.get("myvars", "plot1")
plot2=config.get("myvars", "plot2")
plot3=config.get("myvars", "plot3")
plot4=config.get("myvars", "plot4")
plot5=config.get("myvars", "plot5")
plot6=config.get("myvars", "plot6")

outputvid=config.get("myvars", "outputvid")
val=config.get("myvars", "frames")
safety = config.get("myvars", "safetycopy")
val=int(val)

#Reset settings

deleteDir(savedir)
deleteJpg(savedir)
deleteMp4(viddir,outputvid)


print plot1
print data
print quakes
```

```
67
68 #Ensures coordinates are integers
69 x=int(latcol)
70 y=int(longcol)
71
72 lats=[]
73 lons=[]
74 lons2=[]
75
76 #CSV parsing for local/online data
77 locweb(lats,lons,lons2,x,y)
78 print lats
79
80 #Longitude figure settings
81 a=sorting2(sorting1(lons2)[1],sorting1(lons2)[0])
82 print a
83
84 #projection settings
85 m=proj1(a)
86 print m
87
88
89 #figure settings
90 figure = mp.gcf()
91 figure.frameon="false"
92 mp.axis('off')
93 figure.set_size_inches(20,10)
94 mp.tight_layout(pad=0)
95
96 #plots data
97 plotting(lons,lats,m)
98
99 #save dir#s for data and intro slide
100 testdir= savedir+str("\\a001.jpg")
101 testdir2= savedir+str("\\a002.jpg")
102
103 mp.savefig(testdir,dpi=100)
104
105 #clears figure
106 mp.clf()
107 figure.frameon="false"
108 mp.axis('off')
109
110 #basic projection for intro slide
111 c= Basemap(projection='cyl',llcrnrlat=-90,urcrnrlat=90,\
112     llcrnrlon=-180,urcrnrlon=180,resolution='c')
113
114 #replots basemap
115 c.warpimage(image="intro.jpg")
116
117 #inserts user defined title from config
118 mp.text(-80,25,(title),color='w',size='46')
119 mp.savefig(testdir2,dpi=100)
120
121 print testdir
122
123 #runs rolling and video process
124 vid(testdir2,val,float(2000),'a001.jpg',savedir,viddir+str('//0001.mp4'))
125 vid(testdir,val,float(2000),'a002.jpg',savedir,viddir+str('//0002.mp4'))
126
127 makemylist(savedir)
128 #concatenates videos together
129 concat=subprocess.Popen(["ffmpeg", "-f", "concat", "-i","mylist.txt", "-c", "copy","-y", "output.mp4"])#,shell = True
130 concat.wait()
131
132 #speeds videos up
133 fast=subprocess.Popen(["ffmpeg", "-i", "output.mp4", "-vf","setpts=0.4*PTS","-y", outputvid])#,shell = True
134 fast.wait()
135
```

Appendix F: Functions developed for alternative visualisations(functions.py)

```
# -*- coding: utf-8 -*-
"""
Created on Tue Jun 24 10:52:23 2014

@author: Callum
"""
import matplotlib.pyplot as mp
from mpl_toolkits.basemap import Basemap
import dateutil.parser
import urllib2
import csv
import numpy as np
import operator
import os
import subprocess
import glob
import shutil
import scipy.spatial
import time
import sys
import ConfigParser

#Config parser used to import variables/information from a text based configuration file
config = ConfigParser.ConfigParser()
con=config.read("config.txt")

data = config.get("myvars", "localdata")
quakes =config.get("myvars", "webdata")
basei =config.get("myvars", "baseimage")
serverurl=config.get("myvars", "wmsserver")
layers1=config.get("myvars", "layers")

colour=config.get("myvars", "colour")
colour1=config.get("myvars", "colour1")
colour2=config.get("myvars", "colour2")
colour3=config.get("myvars", "colour3")

mark=config.get("myvars", "marker")
mark1=config.get("myvars", "marker1")
mark2=config.get("myvars", "marker2")
mark3=config.get("myvars", "marker3")

msize=config.get("myvars", "markersize")
msize1=config.get("myvars", "markersize1")
msize2=config.get("myvars", "markersize2")
msize3=config.get("myvars", "markersize3")

plot1=config.get("myvars", "plot1")
plot2=config.get("myvars", "plot2")
plot3=config.get("myvars", "plot3")
plot4=config.get("myvars", "plot4")
plot5=config.get("myvars", "plot5")
plot6=config.get("myvars", "plot6")

safety = config.get("myvars", "safetycopy")

def deleteDir(save):
    '''Deletes any Dir's beginning with framesFor, ensures new data always used'''
    apple =[x[0] for x in os.walk(save)]
    #peach =apple[0]
    #print apple[2:66]
    orange=apple[1:66]
    print orange
    for f in orange:
        if 'framesFor' in f:
            shutil.rmtree(f)
```

```
67
68 def deleteJpg(save):
69     '''Deletes all images within save directory, ensures new images always used'''
70     files = glob.glob(save+str('/*.jpg'))
71     for f in files:
72         if not 'base.jpg' in f:
73             if not 'intro.jpg' in f:
74                 os.remove(f)
75
76 def deleteMp4(vidloc,outputvid):
77     '''Deletes all mp4's in vid dir, ensures new vids always used'''
78     files2 = glob.glob(vidloc+str('/*.mp4'))
79     for g in files2:
80         if not 'output.mp4' in g:
81             if not outputvid in g:
82                 os.remove(g)
83
84 def sorting1(lons):
85     '''Sorts longitudes, finds difference between every point and reports the
86     max difference value and index position as well as partnering index position'''
87
88     lons.sort()
89     v=np.diff(lons)
90     index, value = max(enumerate(v), key=operator.itemgetter(1))
91     return lons[index],lons[index+1], value
92
93
94 def sorting2(x,y):
95     '''This function takes the difference between the 2 index's and then halfs it.
96     This ensures the maximum distance between the 2 points is found and can then be
97     used to set the optimal figure break point. Ultra ignores negative aspects, so this
98     function ensures negative values are negative and postive are positive.'''
99
100    if x <0 and y <0:
101       a=x + y
102       return a/2
103
104    elif x <0 and y >0:
105        a=x + y
106        return a/2
107
108    else:
109         x>0 and y >0
110         a=x+y
111         return a/2
112
113
114 def insurance(data):
115     '''Takes downloaded copy of data and edits for errors and replaces blank
116     cells with 0's. Ensures visualisation can always be made, if errors with USGS data'''
117
118     output="safety.csv"
119     with open(data, "rb") as infile, open(output, "wb") as outfile:
120        r = csv.DictReader(infile)
121        w = csv.DictWriter(outfile, r.fieldnames)
122        w.writeheader()
123        for row in r:
124            if not row["latitude"].strip():
125                row["latitude"] = "0"
126            #w.writerow(row)
127            if not row["longitude"].strip():
128                row["longitude"] = "0"
129 #           if not row["mag"].strip():
130 #               row["mag"] = "0"
131            w.writerow(row)
```

```
132
133
134 def locweb(lats,lons,lons2,x,y):
135     '''Essentially used to determine whther the csv file being used is web/locally based
136     and subsequently deals with either type'''
137
138
139     if not '.csv' in data:
140         quakes1 = urllib2.urlopen(quakes)
141         csv_data =csv.reader(quakes1)
142         next(csv_data)
143         for row in csv_data:
144             lats.append(float(row[x]))
145             lons.append(float(row[y]))
146             lons2.append(float(row[y]))
147
148     else:
149         with open(data) as f:
150             csv_data =csv.reader(f)
151             next(csv_data)
152             try:
153                 for row in csv_data:
154                 #print row
155                     lats.append(float(row[x]))
156                     lons.append(float(row[y]))
157                     lons2.append(float(row[y]))
158             except:
159                     insurance(data)
160                     with open(safety) as f:
161                         csv_data =csv.reader(f)
162                         next(csv_data)
163                         for row in csv_data:
164                             print row
165                             lats.append(float(row[x]))
166                             lons.append(float(row[y]))
167                             lons2.append(float(row[y]))
168
169
170 def proj1(a):
171     '''Sets the equirectangular projection, and ensures correct start/end longitudes'''
172
173     if  'http://' in serverurl:
174         #ensures WMS layer used if in config
175         m = Basemap(projection='cyl',llcrnrlat=-90,urcrnrlat=90,\
176         llcrnrlon=-180,urcrnrlon=180,resolution='c')
177         m.wmsimage(serverurl,layers=[layers1],xpixels=2000,ypixels=1000)
178         return m
179
180     else:
181         #ensures base image used if wanted or no WMS present
182         m = Basemap(projection='cyl',llcrnrlat=-90,urcrnrlat=90,\
183         llcrnrlon=(a),urcrnrlon=360+(a),resolution='c')
184         m.warpimage(image=basei)
185         return m
186
187
188
189 #min_marker_size = 6.25
190
```

```
191
192 def mapPlot(lons2,lats,mag,alpha1,m):
193     '''Takes lat, long and mag and plots quake events on map, varying
194     colour & size for magnitude'''
195
196     for lon, lat, mag in zip(lons2, lats, mag):
197         mp.axis('off')
198
199         x,y = m(lon+360, lat)
200             #ensures all points plotted when origin map placement is altered.
201         z,j =m(lon,lat)
202         msize = mag * 4
203         msize1=mag * 5
204         msize2=mag * 6.25
205
206         if mag <3.0:
207             m.plot(x, y, '#31a354',marker="o",alpha=alpha1, markersize=msize)
208             m.plot(z, j, '#31a354',marker="o",alpha=alpha1, markersize=msize)
209         elif mag <5.0:
210             m.plot(x, y, '#e6550d',marker="o", alpha=alpha1, markersize=msize1)
211             m.plot(z, j, '#e6550d',marker="o", alpha=alpha1, markersize=msize1)
212         else:
213             m.plot(x, y, '#de2d26',marker="o", alpha=alpha1, markersize=msize2)
214             m.plot(z, j, '#de2d26',marker="o", alpha=alpha1, markersize=msize2)
215
216 def plotIndex(x,y,j,i,m):
217     '''Uses mapPlot for specified indiceis within a list, e.g 20-40'''
218
219     if len(x)> i:
220         mapPlot(x[0:i],y[0:i],j[0:i],1,m)
221
222     elif len(x) <i:
223         mapPlot(x[:],y[:],j[:],1,m)
```

```
224
225 def plotting(lons,lats,m):
226     '''Simply plots points depending on whether indicies are used or not.
227     Simpler in functionality than mapPlot, as designed for widespread use
228     not bound to earthquakes'''
229
230     if 'NA' in plot1:
231         for lon, lat in zip(lons,lats):
232             x,y = m(lon, lat)
233             z,j =m(lon+360,lat)
234             m.plot(x, y, colour,marker=mark,alpha=1, markersize=int(msize))
235             m.plot(z, j, colour,marker=mark,alpha=1, markersize=int(msize))
236     else:
237         for lon, lat in zip(lons[int(plot1):int(plot2)],lats[int(plot1):int(plot2)]):
238             x,y = m(lon, lat)
239             z,j =m(lon+360,lat)
240             m.plot(x, y, colour1,marker=mark1,alpha=1, markersize=int(msize1))
241             m.plot(z, j, colour1,marker=mark1,alpha=1, markersize=int(msize1))
242
243     if 'NA' in plot3:
244         if 'NA' in plot1:
245             for lon, lat in zip(lons,lats):
246                 x,y = m(lon, lat)
247                 z,j =m(lon+360,lat)
248                 m.plot(x, y, colour,marker=mark,alpha=1, markersize=int(msize))
249                 m.plot(z, j, colour,marker=mark,alpha=1, markersize=int(msize))
250     else:
251         for lon, lat in zip(lons[int(plot3):int(plot4)],lats[int(plot3):int(plot4)]):
252             x,y = m(lon, lat)
253             z,j =m(lon+360,lat)
254             m.plot(x, y, colour2,marker=mark2,alpha=1, markersize=int(msize2))
255             m.plot(z, j, colour2,marker=mark2,alpha=1, markersize=int(msize2))
256 #
257     if 'NA' in plot5:
258         if 'NA' in plot3:
259             if 'NA' in plot1:
260                 for lon, lat in zip(lons,lats):
261                     x,y = m(lon, lat)
262                     z,j =m(lon+360,lat)
263                     m.plot(x, y, colour,marker=mark,alpha=1, markersize=int(msize))
264                     m.plot(z, j, colour,marker=mark,alpha=1, markersize=int(msize))
265     else:
266         for lon, lat in zip(lons[int(plot5):int(plot6)],lats[int(plot5):int(plot6)]):
267             x,y = m(lon, lat)
268             z,j =m(lon+360,lat)
269             m.plot(x, y, colour3,marker=mark3,alpha=1, markersize=int(msize3))
270             m.plot(z, j, colour3,marker=mark3,alpha=1, markersize=int(msize3))
271
272
273 def time3(self):
274     '''Formats date information nicely'''
275
276     a=[]
277     for x in self:
278         d=dateutil.parser.parse(x)
279         #c=d.strftime('%d-%m-%Y %H:%M:%S')
280         c=d.strftime('%d-%m-%Y')
281         a.append(c)
282     return a
```

```
283
284
285
286 def nn(array):
287     '''Function to calculate nearest neighbour between point array
288     k=number of neighbours,d_upper=distance)'''
289
290     c=[]
291     tree=scipy.spatial.cKDTree(array,leafsize=100)
292     for item in array:
293         result=tree.query(item,k=2,distance_upper_bound=10)
294         c.append(result)
295     return c
296
297 def difference(newlist):
298     '''Simply accesses the point distances from array and returns as list'''
299
300     a=[]
301     for i in newlist:
302         a.append(i[1])
303     return a
304
305 def distSorter(x,y,z,u):
306     '''Essentially returns the long,lat and mag of points over specified distance'''
307     p=[]
308     c=[]
309     d=[]
310     for i,j,k,l in zip(x, y,z,u):
311         if i >6:
312             p.append(j)
313             c.append(k)
314             d.append(l)
315     return p, c,d
316
317 def labelMag(x,y,magnitude,i):
318     '''Adds magnitude numbers to map, i = mag > than'''
319     for  lon, lat,mag in zip( x, y,magnitude):
320         if mag > i:
321             mp.text(lon+358.5,lat-.5,mag)
322             mp.text(lon+-2,lat-.5,mag)
323
324
325
326 def behemoth(i,j,longitude,latitude,magnitude):
327     '''Combines numerous functions to allow aesthetic labelling for quakes'''
328     selection = zip(longitude[i:j], latitude[i:j])
329     myarray=np.array(selection)
330     nn1=nn(myarray)
331     nnlist=list(nn1)
332     index1= map(operator.itemgetter(0), nnlist)
333     new=difference(index1)
334     a1=distSorter(new,longitude[i:j],latitude[i:j],magnitude[i:j])[0]
335     a2= distSorter(new,longitude[i:j],latitude[i:j],magnitude[i:j])[1]
336     a3= distSorter(new,longitude[i:j],latitude[i:j],magnitude[i:j])[2]
337     labelMag(a1,a2,a3,4)
```

```
338
339
340 def vid(x,y,z,name,directory,output):
341     '''Creates dir for images which are rolled via imagemagick subprocess,
342     images then compiled into video via ffmpeg subprocess'''
343     print x
344
345     route=directory+str('\\framesFor')+str(name)+str('\\')
346     #simply acts as a path variable to where rolled images are located
347     print route
348     raw = route+str('%d.jpg')
349     #again acts as a path variable to images
350     print raw
351     z1=z/y
352     #This is simply number of frames divided by distance of image(2000 for Puffersphere)
353
354     print "ANIMATION PROCESS"
355     if not os.path.isfile("images\\framesFor"+str(name)) and not os.path.isdir("images\\framesFor"+str(name)):
356         os.mkdir("images\\framesFor"+str(name))
357     #essentially if there is not a  folder for the selection of rolled images, will make folders for each
358
359     frame = 0
360     p=list(xrange(y))
361     #makes list the length of frames specified
362     for i in p:
363         frame=frame+z1
364         #counter used to determine distance for each rolled image
365         a= str(frame)
366         a1='+'+str(a)
367         o=subprocess.Popen(["convert", x,"-roll",a1, "-scale", "2000x1000", route+str(i)+'.jpg'],shell = True)
368         #This subprocess accesses imageMagick via popen, and carries out the neccessary roll commands.
369         o.wait()
370         #IMP. Tells system to finish this before starting next process. If not present vids made of incomplete images
371
372     c=subprocess.Popen(["ffmpeg", "-f", "image2",
373                             "-i", raw, "-c:v", "libx264",
374                             "-pix_fmt", "yuv420p", "-preset",
375                             "ultrafast", "-tune", "fastdecode",
376                             "-profile:v", "main", "-vf", "scale=2000:1000",
377                             "-r", "30", "-coder", "0", "-g", "6", "-crf", "20",
378                             "-y", output],shell=True)
379     c.wait()
380     #This subprocess accesses FFMPEG and compiles all rolled images within a dir into a mp4
381     #Various settings determine quality, speed and compression.
382
383
384 def makemylist(directory):
385     '''Makes a txt list of every mp4 within a directory, to allow concatination of mp4's'''
386
387     f= os.listdir(directory)
388     test=open("mylist.txt","w")
389     for i in f:
390         if i.endswith('output.mp4'):
391             pass
392         elif i.endswith('.mp4'):
393             test.write('file \'')
394             test.write(directory+str('/'))
395             test.write(i+str('\'') +'\n')
396
```

Appendix G: Earthquake Visualisation survey

Earthquake Visualisation survey:

Date: _____ Age: _____

1) In comparing the globe visualisation to the provided 2D image, which do you find to be more useful and why?

2) Is it easy to understand where earthquakes are occurring in the world and their associated strength/magnitude?

3) Do you think this is an effective way of visualising global earthquakes?

4) Did you encounter any disadvantages or problems with the globe display?

5) How could this visualisation be further improved?

6) Do you think seeing geographic information like global earthquakes on a sphere changed your understanding or led to a better understanding of the distribution of earthquakes?

Appendix H: User feedback results table

Earthquake Visualisation Survey						
User	Q.1	Q.2	Q.3	Q.4	Q.5	Q.6
1	Very interesting, can see the changes	Yes, would like different colours each day.	Yes, but rather quakes appear individually	Can't tell the date, can't see whole world at once but better than a boring still screen	See above	Useful and interactive way of showing it, great way to use technology to improve learning
2	Prefer globe, underused tools in education, map raises issues as to where it is focussed, projection dependant	Ability to freeze globe would enhance and strengthen perceptions of the main concentration of tectonic activity	Overall yes, but would argue that there is a need to contextualise the distribution against plate boundaries	Perhaps issue of colour but would your basic globe show vegetation as in the fmh. I assume relief/submarine relief would be used as a background	Possibly more highlighting e.g. flashing light (cost permitting) to emphasise particular strength	I see the globe as an add on to the 2D view should enhance awareness of global distribution of both physical and human data
3	More user-friendly for general public	No, minimal visual impact because of limit on range involved	Yes	not clear enough, prefer more contrasting colours	Needs more visual impact, too static	No, would like extra links e.g tectonic plates rather than just extrapolating frequency and size
4	Both, globe is eye candy, 2D better to study	Yes, but change of colours used would be useful	Yes	When it had my interest, I could not study further because of continual revolving, I lost track	Stop rotating occasionally, change colours used show plates at end	Yes, first time iv'e thought about how many and their distribution. Show legend constantly

5	The 3D image provides a new aspect, projecting the earthquakes onto a globe, in many ways makes It seem more relevant, the 2D image can be seen anywhere	Yes, intro screen could be more common instead of a full cycle as most members of public would not wait to see the scaling	Yes	No	With a scale to tell what the dots represent	Yes, but plate tectonics would certainly improve
6	Globe more useful(but I'm not sure your comparing like for like(i.e. the glove was bigger, revolving and the symbols gradually appeared over a time frame	Yes	Yes but…	Difficult to see which occurred on each of the 7 days	Flashing symbol for each day, stop/start globe	Not really, being a geographer I was already aware of the distribution of earthquakes
7	The globe-3D- if your project becomes more developed, this could be excellent in 3D	Would be better if any newly occurring earthquakes could 'wink/flash' or be another colour for an hour/day	Yes	Commentary, maybe headphones guiding user through visualisation(like a gallery)	We gave the views of other things we'd like to see	Jury out though the U of E Geog. Website earthquakes are perhaps better because 2D. I think this could be better when developed further
8	colour distinctions of dots on 2D are difficult to see in earth room. Orange too close to red.2D just a snapshot	Yes and Yes	Yes, not sure rotation of globe helps would a static display not be simpler and easier to function	Not sure about timeline between new dots appearance	Text relating to timeline, perhaps new dots could flash a few times	Yes quite helpful

9	Neither, both offer different things. Globe better for showing how different parts of the world really join up without distortion. Map is better for showing whole structure though of course without the time lapse	Yes, though a location name could flash up briefly at the same time as the dot appeared.	Yes, though would be more effective perhaps if it could look more explosive.e.g bright jagged shape flashing then settling as calm dot	Not exciting but interesting	Explosions	No, but its an interesting way of reinforcing it. And its particularly interesting that were seeing real and very recent earthquakes, makes it very immediate and relevant
10	Had to say one is more useful than the other. instead they are complimentary, offering different perspectives on the same issue	Yes	Yes	In the context of RSGS, the never cannot pause the globe display	again pause capability	The globe removes the need to manually adjust from the known shortcomings of any flat map of the globe
11	2D easier to view global distribution, colours clearer	Colour difference between orange and red difficult to distinguish , possibly projector related (dead bulb)	On very basic level yes. Shows distribution clearly but would be better if the globe spun more slowly and/or be paused.	Too fast, earthquake markers perhaps too large	Ability to pause for closer inspection would be helpful	No, but only because I have previously studied earthquakes If I had no background info I think it would be very useful.
12	Both have their merits, 2D provides quicker	Yes, very	Yes, think good for children as globe has	No	Maybe put in names of seas + oceans	Definitely

	"at a glance"		fun value			
13	Globe, easier to relate to	Yes	Yes	No	Don't know, not technical enough	Yes
14	The Globe	yes	yes	no	in HD	Yes
15	The globe is more realistic than a map	yes, because of the tectonic plates	certainly	No		Yes, as you can see where they are distributed, never realised there were so many
16	The globe, 3D is more interesting learning tool	Yes although it would be good to see the strength details displayed as well	Yes	no	More info on globe, and may be touch sensitive for more detailed info	Yes
17	The globe, provides better perspective	yes	yes	took a while to get my head around the sequence of programmes (NA seq. of RSGS vids)	Signal each program	Yes, very much so
18	Globe, gives a better visualisation of where earthquakes are happening and also shows lots of things, whereas the 2D map can only show	yes	yes	no	more labels which tell you exact magnitude of each Earthquake. And also tells you what the globe is showing you	Yes

	one thing					
19	The Globe provides a clearer and more impacting picture	Yes, this globe set up is clearly defining the earthquake and magnitude	Yes, it immediately allows you to see the global eruptions for a variety of global	No, it clearly speaks for itself , whichever mode is being shown	suggested names of oceans/countries	Most definitely
20	Globe visualisation more readily appreciated	very much so- very clear	yes, simple and straight forward	No	too clever for me	Yes, globe leads to better understanding

Appendix I: Index of Files

For Earthquake Visualisation

Type	Name	Location	Purpose	Size (KB)
Directory	images	M:\DissFinal\Puffersphere\images	Stores all images created by script	0
Directory	vids	M:\DissFinal\Puffersphere\vids	Stores all videos created by script	0
JPEG	base	M:\DissFinal\Puffersphere\base.jpg	Default satellite image	770
JPEG	title2	M:\DissFinal\Puffersphere\title2.jpg	Introductory image for visualisation	109
TXT file	config1	M:\DissFinal\Puffersphere\config1.txt	Configuration file for installation settings	1
py	driver	M:\DissFinal\Puffersphere\driver.py	Primary python script	9
py	functions	M:\DissFinal\Puffersphere\functions.py	Developed methods script	11
pyc	functions	M:\DissFinal\Puffersphere\functions.pyc	Developed methods script(combined python file)	12

For Alternate Visualisations

Type	Name	Location	Purpose	Size (KB)
Directory	images	M:\DissFinal\alternate\images	Stores all images created by script	0
Directory	vids	M:\DissFinal\alternate\vids	Stores all videos created by script	0
JPEG	base	M:\DissFinal\alternate\base.jpg	Default satellite image	770
JPEG	title2	M:\DissFinal\alternate\title2.jpg	Introductory image for visualisation	109
TXT file	config	M:\DissFinal\alternate\config1.txt	Configuration file for installation settings	1
py	driver	M:\DissFinal\alternate\driver.py	Primary python script	9
py	functions	M:\DissFinal alternate\functions.py	Developed methods script	11
pyc	functions	M:\DissFinal\alternate\functions.pyc	Developed methods script(combined python file)	12
CSV	SV	M:\DissFinal\alternate\SV.csv	Test data for visualisation(European Starlings)	3
CSV	city	M:\DissFinal\alternate\city.csv	Test data for visualisation(Global Cities)	118