ChartFlight – From Spreadsheets to Computer-Animated Data Flights

Rainer Lutz
Computer Science Department
University of Trier, Germany
lutzr@uni-trier.de

Stephan Diehl
Computer Science Department
University of Trier, Germany
diehl@uni-trier.de

Abstract

In business as well as science a clear and professional presentation of quantitative information is often required and helps to efficiently communicate new insights. The predominant approach is to integrate charts into slide shows created with standard presentation programs. In this paper, we introduce the chart flight metaphor for visualizing spatially distributed statistical data as a computer-generated three-dimensional camera flight over a map with animated charts. Our web application leverages the Blender 3D modeling and animation tool to allow end users to submit their data sets, and easily generate chart flight videos without profound knowledge of computer graphics methods and systems. The generated videos can be included into slide show presentations, put on web pages or shared via file hosting sites, and even displayed on low-performance hardware devices like mobile phones or netbooks.

CR Categories: H.5.1 [Information Interfaces and Presentation]: Multimedia Information Systems—Animations;

Keywords: animation, web, video

1 Introduction

Without visualization complicated data cannot speak for themselves. Newspapers or news programs on TV mostly show quantitative information like demographic surveys, stock market indices, or sales numbers in form of charts.

In business and science a clear and professional presentation of quantitative information is often required and helps to efficiently communicate new insights. The predominant approach is to integrate charts into slide shows created with presentation programs like Microsoft Powerpoint or OpenOffice.org Impress.

Statistical charts created with such office tools usually do not look as professional and are not animated as those shown on news shows on TV. While it is possible for the end user to make good charts, she must either spend a lot design effort or spend both a lot of money and time for professional tools and on learning how to use them.

In this paper, we introduce a totally different approach for presenting charts utilizing the idea of generated three-dimensional camera flights.

Imagine that you want to present election results of different parts of your country, e.g. for federal states, provinces, etc. In general, if you do not have access to professional tools, the straightforward approach is to create a sequence of images or slides, each showing a chart of the results for one of the states or provinces. During the presentation one slide is shown after the other possibly using an animated transition like flipping or blending between subsequent slides. In contrast, our approach creates a three-dimensional, animated video, which may be used for presentations or shared on the web, e.g., via video hosting websites like YouTube.

For instance, the election results would be shown as a kind of 3D flight, henceforth called a *chart flight*, over the map of your country with intermediate stops at each federal state. At each stop the local results would be shown as a chart right in place. Such an approach has the advantage of depicting graphically the location the data is related to, rather than just showing the location as plain text.

Thus, our work makes the following contributions:

- A generic metaphor for animated, three-dimensional presentations of spatially distributed statistical data.
- An approach for generating these presentations using the open source 3D computer animation software Blender [Blender 2009].
- An end-user friendly frontend for making our approach accessible on the web.
- Generation of high quality 3D content without requiring high-end computer graphics hardware on the client-side, which makes it accessible to mobile phones, PDA's, or netbooks.

Moreover, in the conclusions we briefly explain how you can adapt our approach to leverage other visualization techniques.

2 End-user Visualization

There are many other areas where spatially distributed statistical data occur, for instance, economical data of different countries, sales number at different branches, cancer incidence statistics related to different parts of the body, weather data at different cities, etc.

As this kind of statistical data is so widespread, the question arises how we can enable end users to easily produce chart flight videos from their own data sets.

Obviously, the typical end user lacks the skills to create three-dimensional animations or animations at all using 3D content creation suites (such as Blender). Therefore, our goal was to find a user-friendly approach requiring no computer graphics background on the part of the user. Typically, 3D content creation suites like Blender provide a large number of different features through their graphical user interfaces. But the question is how to decide which options should be available for the end user. Should it be possible to specify the amount of specularity of a chart or the letter spacing of text? Questions like these are crucial when it comes to usability. Too many options could discourage a user to work with this form of data visualization and on the other hand if we provide too few settings our system will not be flexible enough for different types of data.

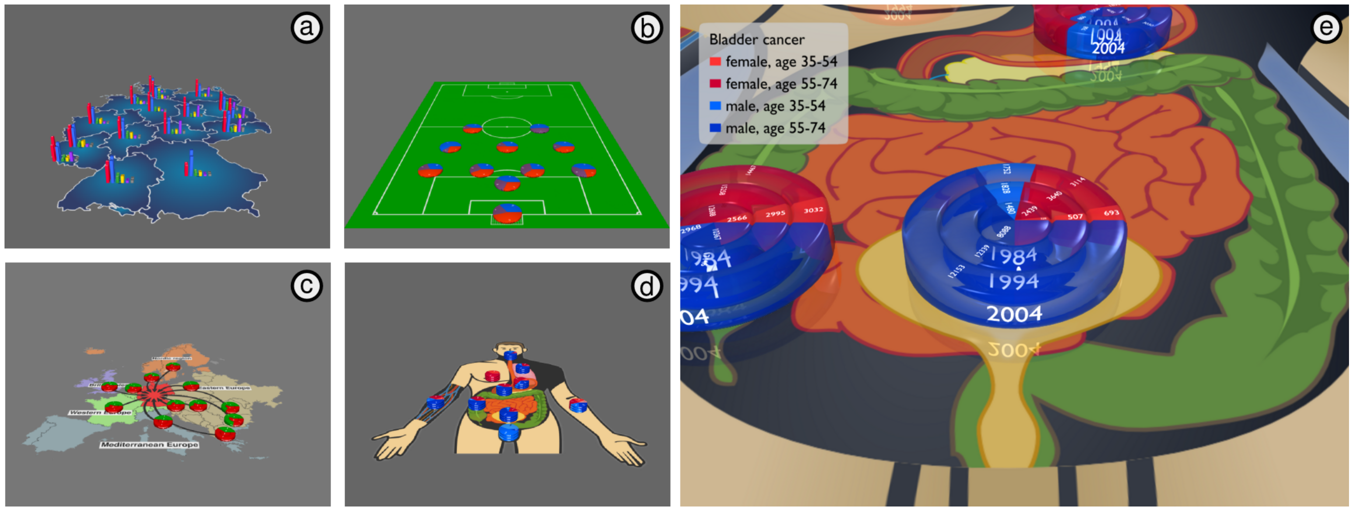


Figure 1: *Selection of different real-world examples. Results of German parliamentary elections for its federal states (a), touches of the ball for players of a soccer team (b), migration in Germany (c), cancer incidence statistics for the most common cancer types (d and e).*

2.1 The ChartFlight metaphor

Returning to our election results example, to create a chart flight video the user has to provide at least three different inputs: A map of the respective country, the locations, which in our case were defined by a point close to the center of each federal state, and the data set, which contains the election results for each location. Note, that none of these inputs require computer graphics knowledge on part of the user.

As another example, consider cancer incidence statistics. Imagine you want to provide information about common cancer types and at same time show where these cancer types are normally located in the human body. Based on our approach we would suggest either to draw a sketch of the human body or to try to find a usable image on the internet, provide the data set with data for each cancer type, and define the locations where to display the information.

While the above examples greatly differ in the contents of the images and the interpretation of the data sets, the generated videos follow the same scheme.

2.2 The ChartFlight data model

Based on observations like the one discussed above, we developed a generic model for chart flight visualizations. Basically, a chart flight is a generated 3D camera movement along a given path that needs three different inputs:

- the ground plane to display a given image
- an ordered set of locations to define where charts will be displayed and to generate the camera movement
- a data set that contains a separate table for every chart location

The chart flight model can be applied to many real world examples. For instance, we visualized freight traffic and passenger transport statistics of the European Union based on data sets from [European Commission 2009]. Besides election results and cancer incidence statistics we also generated chart flights presenting German migration statistics in 2007 [Statistisches Bundesamt 2009]. Figure 1 shows an overview of different real world examples.

2.3 The ChartFlight web application

Our first stable prototype implementing the chart flight metaphor is a web application called *ChartFlight*. In general, a web application has the advantage that users do not have to download and install a separate application or connect to a rendering server directly, which could also imply problems with firewall settings.

Basically, ChartFlight consists of three separate parts: a web client, a web server, and one or more rendering servers in the background.

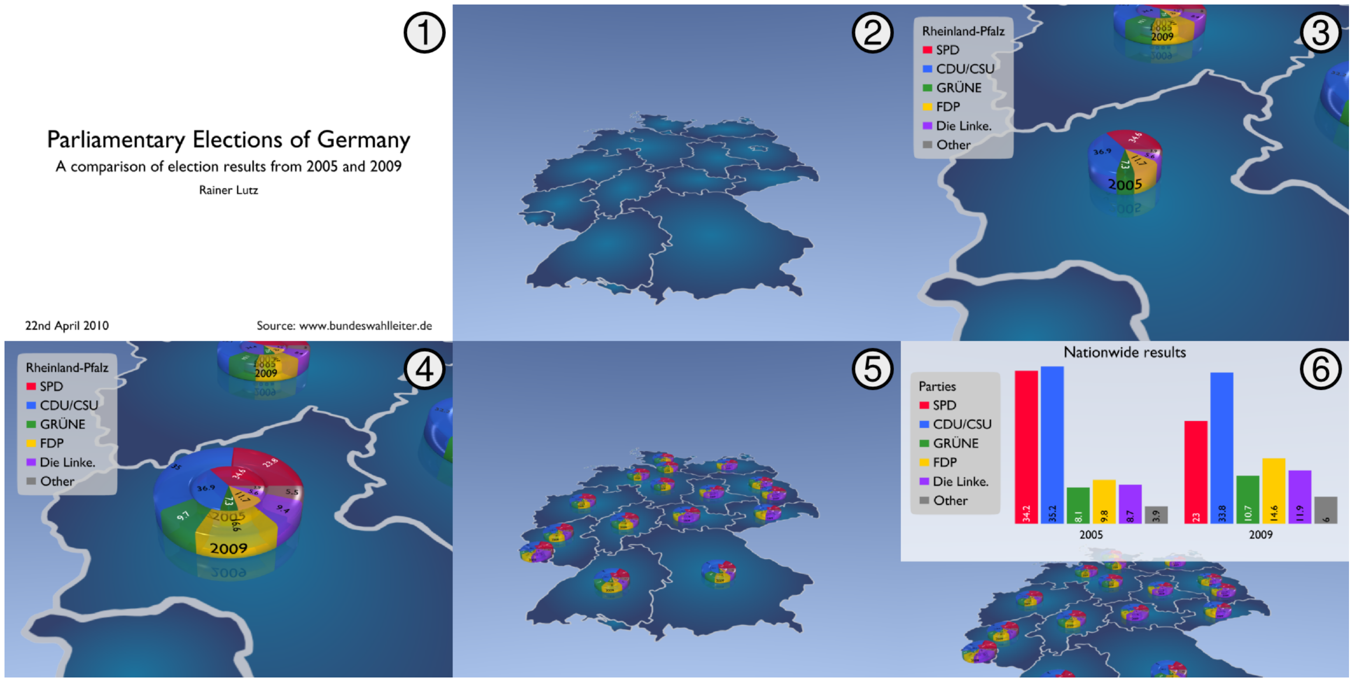
The web client provides different webforms to create a settings file, which is then transferred through the web server to the rendering engine to generate a chart flight video.

In our current prototype implementation web server and rendering server are located on the same computer, but, in general, it is possible to distribute them and provide more than one rendering server for ChartFlight.

2.4 Videos for end users

Data visualizations for end users rendered as video files have several advantages as compared to real-time applications. First of all such files can be easily embedded in presentation slides and websites or uploaded to video file hosting services. Even if users want to play a generated video on a local machine they may either use a pre-installed video player or decide to install their favorite player, which they are familiar with. That means they do not have to learn how to interact with a real-time visualization. Finally, for playing a video less resources are required than for running a real-time presentation, which especially concerns laptops and low-performance hardware devices like mobile phones or netbooks because there it is often not that common to have access to powerful graphics hardware.

Our idea to present a video generated with ChartFlight is close to presentations shown at [Rosling et al. 2004]. The lecturer plays the video or just a part of it and while the audience is watching the animation she gives them additional information about the shown statistics. The video can be paused if a description of more specific facts is required. This scenario maintains our choice to prefer rendering video files over creating real-time applications because it is easier to integrate videos into presentations. Alternatively, users



1
Parliamentary Elections of Germany
A comparison of election results from 2005 and 2009
Rainer Lutz
22nd April 2010
Source: www.bundeswahlleiter.de
2
3
4
5
6
Rheinland-Pfalz
SPD
CDU/CSU
GRÜNE
FDP
Die Linke.
Other
Nationwide results
Parties
2005
2009

Figure 2: *A sequence of frames showing a video's five phases in the following order: Title screen, intro view, chart animation (two frames), final overview, and summary chart. The scene, automatically generated and rendered by ChartFlight, visualizes parliamentary election results of the years 2005 and 2009 for all 16 federal states of Germany. Finally, the summary chart compares the nationwide results of both years.*

could upload videos to a file hosting service to provide visualizations of their statistics for a larger audience. The generated videos can also be post-processed with video editing software to add textual annotations or audio comments.

3 Features of ChartFlight

Next we provide a more detailed view of our prototype implementation. In this section we discuss the various features of ChartFlight from a user's point of view. In the subsequent Section 4 we provide technical details of ChartFlight, in particular, how the user interface is connected to the rendering engine. Furthermore, in Section 5, we present some performance results.

The common structure of a generated video is shown in Figure 2. A full video consists of the following five sections:

Title screen. The title screen displays general information about the video like title, author, or data source. Title screens are always two-dimensional and shown at the beginning of a video before the actual three-dimensional camera flight.

Intro view. The camera is placed above the whole scene and shows the map from a bird's eye perspective. From this position the actual camera flight starts by moving the focus of the camera down to the location of the first chart.

Camera flight. The main part of each video is a camera animation based on our generic model. The focus of the camera moves from location to location where the individual charts are shown. Rather than moving with constant speed from one position to another the camera animation uses a more natural slow-in/slow-out approach. Furthermore, the camera moves along a straight line to allow the user to focus on the location the next diagram will appear. The timing of the camera flight is set by default such that it is fast but not hectic. A presenter should be able to say a short sentence during the camera animation, for instance, to announce what data will be shown next.

Final overview. Similar to the intro view the final overview, also called outro view, shows the whole map. But at this time, all charts are displayed as created during the camera flight allowing the user to compare the charts.

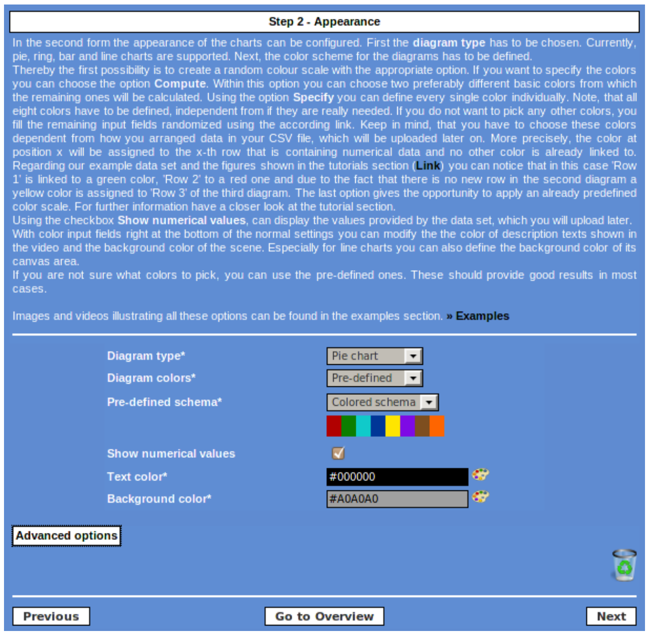
Summary chart. This last part of a rendered video is optional and may be used to show global or summary information. Similar to the title screen a summary chart is two-dimensional and displayed on a plane orthogonal to the camera's view direction.

3.1 The web-based user interface

Basically, the web client consists of eight webforms and several additional web pages that provide information about the current progress. Furthermore, there are many examples and tutorial pages to help the user to select the right parameters and provide her data in the right format. By filling in these webforms the user defines the settings to be used for generating the video.

Most of the webforms consist of two parts: one with basic settings that are important or even required for creating a video and an optional part containing advanced settings. Users decide individually whether they want to modify these settings or use default values and just care about the basic input fields. Furthermore, each webform checks the correctness of all inputs, if required settings are missing, notifies the user by changing the color of the input field, annotating it with a small exclamation mark as well as displaying a global error message.

For the process of creating a video the user goes through our webforms in the following order:



Step 2 - Appearance

In the second form the appearance of the charts can be configured. First the **diagram type** has to be chosen. Currently, pie, ring, bar and line charts are supported. Next, the color scheme for the diagrams has to be defined.
Thereby the first possibility is to create a random colour scale with the appropriate option. If you want to specify the colors you can choose the option **Compute**. Within this option you can choose two preferably different basic colors from which the remaining ones will be calculated. Using the option **Specify** you can define every single color individually. Note, that all eight colors have to be defined, independent from if they are really needed. If you do not want to pick any other colors, you fill the remaining input fields randomized using the according link. Keep in mind, that you have to choose these colors dependent from how you arranged data in your CSV file, which will be uploaded later on. More precisely, the color at position x will be assigned to the x-th row that is containing numerical data and no other color is already linked to. Regarding our example data set and the figures shown in the tutorials section (**Link**) you can notice that in this case 'Row 1' is linked to a green color, 'Row 2' to a red one and due to the fact that there is no new row in the second diagram a yellow color is assigned to 'Row 3' of the third diagram. The last option gives the opportunity to apply an already predefined color scale. For further information have a closer look at the tutorial section.
Using the checkbox **Show numerical values**, can display the values provided by the data set, which you will upload later.
With color input fields right at the bottom of the normal settings you can modify the the color of description texts shown in the video and the background color of the scene. Especially for line charts you can also define the background color of its canvas area.
If you are not sure what colors to pick, you can use the pre-defined ones. These should provide good results in most cases.

Images and videos illustrating all these options can be found in the examples section. **» Examples**

Diagram type*	Pie chart
Diagram colors*	Pre-defined
Pre-defined schema*	Colored schema
	🟥🟩🟦🟦🟨🟪🟫🟧
Show numerical values	✓
Text color*	#000000
Background color*	#A0A0A0

Figure 3: *A description text and a example page lead through all available settings in the bottom part of the appearance form. There users simply choose their favored options. Additionally, advanced options for further adjustments may be opened.*

Title form. Users choose between two different types of title screens. Either they upload an image, which will be displayed as title, or they decide to generate a title screen based on their text input, for instance, title, subtitle, author etc. In the advanced settings users may customize colors or determine the layout of the title screen.

Appearance form. Here, users determine global settings like background and font color, but also the type and color palette of the charts (Figure 3). Currently, as shown in Figure 4, four chart types are available: Pie charts, ring charts, bar charts, and line charts. These chart types have different advantages with respect to occlusion, perspective distortion, relative size, and readability of labels. The advanced settings include options for the size and material of the charts.

Animation form. This form provides only a few basic settings for the animation, for example, users may choose to fade in charts slowly. In contrast, there are a lot of advanced settings including different animation intervals, for instance, how long the title screen is displayed or how long it takes to move the camera from one chart to another.

Summary chart form. If users opt for a summary chart, they can, similar to title form, either upload an image or provide data to generate a flat 2D chart. In the advanced section users may change animation intervals.

File uploads form. Here, users specify and upload up to four files, namely the map as image file, the data set as text or CSV file, and images for the title screen and the summary chart. Uploading a map and a data set is required for creating a video, while the other uploads depend on settings made in the previous webforms. Thus, only if users decide to use an image for title screen an additional upload box will appear in this form. During the uploading process file types and sizes are checked and if accepted information about those files are displayed. In addition, the file containing the data set will be validated. If it does not match the format introduced in Section 3.2, it will be rejected and users will be informed about the errors. Each error description is listed along with the line, in which it occurred.

Diagram locations form. This webform provides two ways to link every single table to a location on the map. Users choose to either select the locations interactively on a map using a Java applet or enter their coordinates as numbers in an input form. Figure 5 shows the Java applet displaying the map of Germany and 16 marked locations.

Miscellaneous form. This webform covers all settings that do not fit one of the other categories. For instance, users might change the video resolution suggested by ChartFlight. Furthermore, they must enter their e-mail address in this form, such that ChartFlight is able to contact them once the rendering of the preview images or the video is finished.

Overview form. This final webform provides an overview of all webforms and indicates whether they have been successfully completed. Users may return to each of the forms or confirm their settings and make ChartFlight start rendering preview images.

When users submits their data, a *job* with a unique identifier is initiated. It contains all information needed to generate a chart flight, namely the map, the data set linked to an ordered set of locations, and all user settings. By means of the job identifier users are able to download the rendered video, look at preview images, or get additional information about the current state of the job.

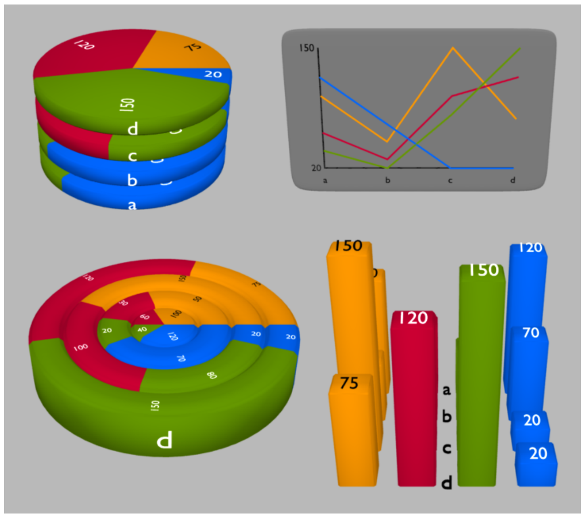


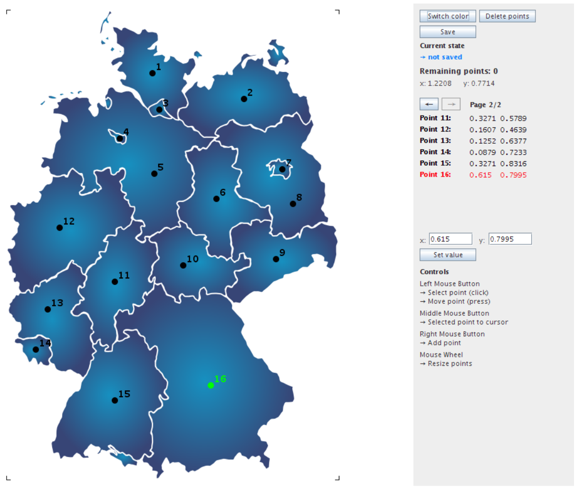
Figure 4: *Different chart types: pie chart, line chart, ring chart, and bar chart with a basic shading*

Once preview images are available ChartFlight notifies the respective user via e-mail. Such an e-mail contains among others a link to the preview web page of the job where users find snapshots of different states of their video. This preview mechanism provides a first overview of how the generated video will look like. Users can either modify the settings and re-render all images, confirm it for final rendering, or simply delete the job.

While the rendering of preview images only takes seconds, the rendering of the video usually takes several hours, see Section 5. Again, when the rendered video becomes available ChartFlight sends a notification e-mail.

Both generated and uploaded data is kept on our server for a fixed period of time (currently one month). Users access these data via the given identifier. Moreover, they have the possibility to modify and re-render their jobs.

Before we discuss the architecture and implementation of ChartFlight, we briefly return to our example — the presentation of election results. Figure 2 depicts a sequence of six frames extracted from a video that was generated by ChartFlight. It shows a comparison of the German parliamentary elections of the years 2005 and 2009. The left upper frame displays a title screen. Since we just input strings for title, subtitle, author, etc., ChartFlight created this title screen automatically using a standard layout. The second frame illustrates an overview of the map of Germany, which we also call intro view. It shows the situation just before the actual chart flight when so far no charts have been displayed. The third and fourth frame give you an idea of how the actual charts look like. In this particular case we chose election results of Rhineland-Palatinate visualized by ring charts. A final overview of all charts is presented in Frame 5. Comparing the charts in the western part of Germany with those in the east, one can easily see that the party colored purple got more votes in Eastern Germany. Finally, the sixth and last image shows a generated summary chart visualizing nationwide election results of 2005 and 2009. In contrast to the three-dimensional diagrams where we preferred ring charts, we decided to use bars instead, which facilitate a comparison of both years.



Current state
→ not saved

Remaining points: 0
x: 1.2208 y: 0.7714

Page 2/2

Point 11:	0.3271	0.5789
Point 12:	0.1607	0.4639
Point 13:	0.1252	0.6377
Point 14:	0.0879	0.7233
Point 15:	0.3271	0.8316
Point 16:	0.615	0.7995

x: 0.615 y: 0.7995

Controls
Left Mouse Button
→ Select point (click)
→ Move point (press)
Middle Mouse Button
→ Selected point to cursor
Right Mouse Button
→ Add point
Mouse Wheel
→ Resize points

Figure 5: *Java applet displaying a map of Germany. In this example a user has already set all available locations, which got an individual numbering.*

3.2 Simple file format for data sets

ChartFlight expects data sets to be stored in CSV files, i.e. text files with comma separated values. This is a very simple, but widely used format to exchange data between different spreadsheet or database applications. Table 1 provides a small example covering two data tables for the actual chart flight, which implies that we want to create only two three-dimensional diagrams, and a third one for the summary chart. These tables can be easily converted to a CSV file using commas as column separators.

The resulting CSV file contains a set of tables that are separated by an empty line. The first table in the data set will be assigned to the first user-defined location and considered first during a chart flight, and so on. Users who want to create summary charts as well have to append these data to the end of the same file.

The three tables in Table 1 contain both column and row headers

Diagram 1	*Column 1*	*Column 2*	
Row 1	50	80	
Row 2	50	20	
Row 3	30	50	
Diagram 2	*Column 1*	*Column 2*	*Column 3*
Row 1	21	70	50
Row 2	66	9	40
Summary chart	*Column 1*	*Column 2*	
Row 1	70	24	
Row 2	54	56	
Row 3	26	129	
Row 4	99	12	

Table 1: *A small example of a data set accepted by ChartFlight, which provides data for two three-dimensional diagrams on the map and a two-dimensional summary chart.*

(printed in italics). The column headers are used to name the individual charts, for instance, one pie out of all stacked pie charts at a chosen location. Each column of a table provides the data for such an individual chart. In contrast, a row header describes a single element of an individual chart, for instance, one sector of a pie chart. In addition, the row header defines the color of the element. This means that elements with equal row headers will be colored the same. For example, if you assign the color red to the row header *Row 1*, every row with that description will be colored red. Finally, the upper left entry of a table might be used to describe the whole diagram or its location.

4 Architecture of ChartFlight

The rendering server consists of two components. For generating and rendering a scene we chose Blender, a free open source 3D content creation suite that provides a Python scripting interface [Blender 2009]. The second component, which is actually connecting the web server and Blender is a small Java console application called *JobListener*. This Java application is essential for providing ChartFlight to more than one user at a time, because it manages the job database.

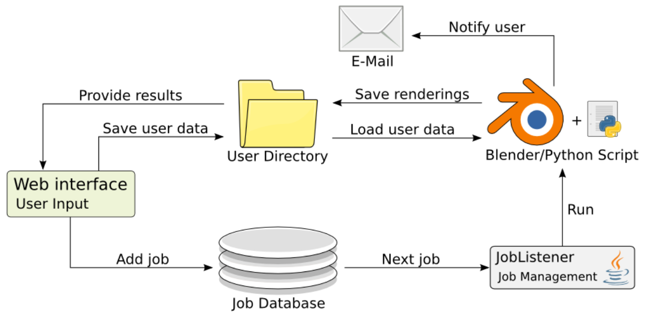


Figure 6: *Background operations when processing a job*

Figure 6 illustrates how web server and rendering server cooperate and what happens in the background when users confirm their job to render preview images or a video. The web interface is linked to two other nodes — the user directory and the job database. The former is created directly after users have started a new job and is used to save all necessary information. More precisely, all uploaded files will be moved to this directory and after users have confirmed their job for rendering preview images an XML file containing the settings will be saved there. Furthermore, when users confirm their job it will be added to the database as indicated in Figure 6.

Now that the database contains the job, it is competing with other jobs in a waiting queue. Basically, this means the earlier a job has been added the sooner it will be processed. But using only this approach causes several problems. For instance, while ChartFlight is rendering a full video, it is not possible to render preview images because Blender is already in use. This means users have to wait until all jobs added before their own are completely rendered, even if they, for example, just want to change one letter of the title screen and re-render the preview images. To solve these problems we decided to use two different queues. One for jobs that are waiting for preview images and another for those the full video shall be rendered. But this leads to the problem that we would have to decide which queue is more important and will be processed first. Finally, we kept the idea of having two different waiting queues, but in our current prototype it is possible to run two instances of Blender at a time — one for creating preview images and one for rendering full videos. Altogether, that means we completely separated both tasks.

Returning to Figure 6 the JobListener searches for the next job using the discussed waiting queue mechanism. If jobs were found and no Blender instance for the respective task (preview images or full video) is already running, the JobListener creates a new instance and passes the required information. Creating a Blender instance in our case means that Blender itself is started and a Python script for generating the three-dimensional scene together with all animations is executed. Thus, Blender has to access the user directory in order to load images, the data set and all user settings. After the scene is created Blender renders the required frames for either the preview images or the full video and saves the result to the user directory. The last task of the Python script is to send a notification e-mail to the user.

4.1 Generating videos with Python & Blender

As already mentioned, we use Blender to generate and render chart flights. Both can be done by utilizing the Python API of Blender, which provides access to the most common features via the programming language Python. Basically, Blender uses scenes to describe a set of three-dimensional objects and their animations. Note, that all objects even if they are not displayed for a certain amount of time, i.e. a certain number of frames, have to be available in the scene before the actual animation is applied and rendered. In other words, a dynamic creation of an object at a specific frame is not possible. Instead, we resort to different techniques like modifying the alpha values of object to make them appear.

We implemented several Python scripts to automatically create such scenes and animations based on the user settings and data sets. These scripts provide classes to send notification e-mails, to parse the CSV files and the user settings, to store them in internal data structures, and to use these to generate, animate, and render a scene.

Before describing these classes in more detail, we briefly describe how animating 3D objects in Blender, henceforth called Blender objects, works.

Basically, a Blender object can be considered as a set of different properties like geometry, location, or material settings (e.g. color, specularity, ...). All these properties are relevant when rendering images showing a Blender object. In order to animate a Blender object you have to use so called IPO curves to change the value of a property over time. Therefore, each animated property has it's own individual IPO curve. In general, an IPO curve can be considered as a two-dimensional curve that links the value of a property (y-axis) to a specific frame index (x-axis). For instance, if we want to fade in a Blender object, we set its alpha value at the first frame index of

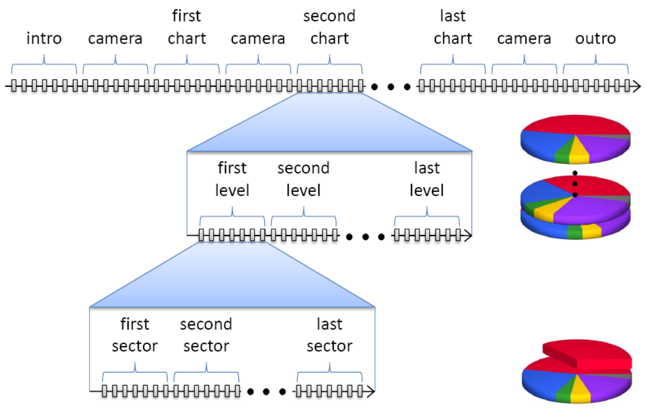


Figure 7: *Plot of a ChartFlight video.*

the animation to 0.0 (transparent) and at its last frame index to 1.0 (opaque). Given these two values Blender is able to interpolate a value for every frame index in between the start and end frame index of the animation. For all other frames indices, i.e. those before the start index and after the end index of the animation constant values based on the first and last value will be assigned. In summary, an IPO curve exactly defines the value of a property at each point in time.

This said, we can now have a closer look at the classes provided by our Python scripts.

Main class. For every single job the main class will be initialized and the scene generated each time Blender is started. The main class links all other parts to load data, create an animated scene, and render it to produce the final video. For all these tasks Python classes are provided.

Title and map classes. The `Title` class is responsible for the title screen, the layout of all text objects on it and its animations. The `Map` class creates a plane for displaying the uploaded image. The image is mapped onto this plane using both classes of the Python API and the utility classes considered below to generate materials and textures and assign them to that plane. In addition, special properties like alpha mapping may be set individually.

3D charts. The `BasicChart` class contains the main functionality for every kind of 3D chart. These are among others member variables and methods for maximum width and height to define the maximum space in which such a chart is displayed. Furthermore, basic animation settings are stored here. Classes like `PieChart`, `RingChart`, `BarChart`, and `LineChart` implementing specific chart types are derived from the `BasicChart` class. Each of these classes has its own method to create charts for the full data set. In other words, all charts are created from the start but are not visible.

During the generation of all charts the information about where and when charts appear is computed based on the ordered chart locations provided by the user. Finally, the indices of the start and end frames of the animation of each chart are stored in a list of frame tuples. This list is accessible by other classes, for instance, the `Legend` class uses the frame tuples to generate the animation of each chart legend, i.e. to define the IPO curve of its alpha value.

2D summary charts. The `Basic2DChart` class covers the main properties of all two-dimensional charts and the specialized classes for the different chart types are derived from it. We implemented a top level class that uses these classes in order to create a summary chart. In particular, this gave us the possibility to just initialize the top level class with the respective values to generate the full summary chart.

Scene classes. There are two classes that are relevant for rendering the complete scene — `Lighting` and `CameraAnimation`. For the lighting it is important to have a setup that both illuminates every region of the map as well as all displayed charts. Considering these restrictions we decided to use sun lamps in Blender, which have a constant light falloff. To illuminate the whole scene from all directions the `Lighting` class puts four sun lights at each corner of the map and one pointing towards the cameras view direction. The camera itself is created, positioned, and animated by the `CameraAnimation` class based on the chart locations, the list of frame tuples, and a predefined camera offset that provides the distance between the position of a 3D chart and the camera itself.

Figure 7 shows the plot of a complete ChartFlight video. Camera animations are used as transitions between intro, outro, and chart animations. The chart animations are hierarchically subdivided into animations of the levels (at every location several charts can be shown) and at each level, the parts of the chart, for instance, the sectors of a pie chart, are animated. In order to show the full map in the intro or outro view a global position is calculated based on the width of the map and the movement from/to this location is added as a camera animation.

Utility classes. Our utility classes only provide static methods, thus that one can easily apply them to different kinds of objects or use them within other classes without having to create an instance. While most of these methods use functions from Blender's Python API to generate, for example, materials or IPO curves for animation, some provide mathematical calculations or create basic objects. An important utility class is the the `Ipo` class, which provides methods to compute IPO curves for animation. Given a reference to an object and some animation settings, e.g., a sector of a pie chart or the indices of the start and end frames, those methods create the specific animation. For instance, a method for creating a fading animation assigns an IPO curve to the alpha property of a given object.

5 Performance

Next we present and discuss results of our experimental studies, which deal with two basic questions:

- How do user settings, for instance, different resolutions effect rendering times?
- How many videos can a single server render per day?

For our evaluation we used two different hardware setups. The first is named setup D for development because it was also utilized for developing our prototype and the second, assigned with the token S for server, shows the specifications of the current ChartFlight server.

Hardware Setup D

CPU	Intel Core2Duo 6400 @ 2.13GHz
MEM	1024MB DDRII 667MHz
GFX	NVIDIA GeForce 7900 GTO 512MB
OS	Windows XP Professional SP 3

Hardware Setup S

CPU	Intel Core2Quad Q9300 @ 2.50GHz
MEM	3894MB DDRII 800MHz
GFX	NVIDIA Quadro FX 570 256MB
OS	Ubuntu 8.04.3

5.1 Experiment: Parameter settings

For comparing rendering times of different settings we had to set up a basic test scene first. For our test scene ChartFlight had to render exactly four stacked pie charts with four segments each. For the map itself we took the image of Germany as seen in Figure 2. Using this basic scene we measured rendering times starting with that frame the legend of the chart is shown the first time until the animation of the chart is completed and the camera moved to the next location. For the camera movement we assumed five Blender units which is half of the maps width and therefore a good average value. Considering these frames gives us an average duration for rendering exactly one chart and hence, it may be used to extrapolate rendering times for whole chart flights displaying several diagrams.

Our first experiment was to compare rendering times for four different resolutions. Based on our test scene and hardware setup D we measured the values shown in Table 2.

	Video 1	**Video 2**	**Video 3**	**Video 4**
Resolution	512x384	640x480	800x600	1024x768
Pixels	196608	307200	480000	786432
Rendering time	18:42 min	26:07 min	36:35 min	56:08 min
Extrapolation	2:29:36 h	3:28:56 h	4:52:40 h	7:29:04 h
per frame	1.045 s	1.459 s	2.044 s	3.136 s

Table 2: *Comparison of rendering times for different resolutions. All videos were rendered using pie charts and a test scene of 1074 frames, which is equivalent to a video length of 42.96 seconds.*

Not surprisingly, the rendering time increases with a higher resolution. But if you compare the amount of pixels that were rendered for one frame regarding the first and the last video you get a ratio of 1:4 while the rendering time is only three times higher. This shows that in our case rendering times do not increase as fast as the amount of pixels when rendering higher resolutions with the internal renderer of Blender. Based on the measurements in Table 2 we also extrapolated the rendering times for a video with eight locations, i.e. a video consisting of camera flights to eight different locations and animations of the test charts at these locations.

Due to the fact that all charts have a different geometry and thus visualize data differently we ran a second experiment that examined if and how the available diagram types effect rendering times. For this experiment, we used a fixed resolution of 800x600 pixels and varied the diagram type. Table 3 shows the results of our second experiment.

	Video 1	**Video 2**	**Video 3**	**Video 4**
Diagram type	Pie chart	Ring chart	Bar chart	Line chart
Frames	1074	1074	1074	344
Video length	42.96 s	42.96 s	42.96 s	13.76 s
Rendering time	36:35 min	35:55 min	37:47 min	11:02 min
per frame	2.044 s	2.007 s	2.111 s	1.924 s

Table 3: *Comparison of rendering times for different diagram types. All videos were rendered in a resolution of 800x600 pixels.*

The rendering times as well as the file sizes of the first three videos with pie, ring, and bar charts differ only slightly. It took around 36 to 37 minutes to render 1074 frames. The differences are in the range of milliseconds, if we calculate average times for creating a single frame.

The fourth video, which displays line charts, needs 68 percent less frames (344 compared to 1074) and consequently less time (13.76 s). As a single line in the line chart represents several values at the same time while, a segment of a pie chart or a bar of a bar chart only represents a single value. Hence, the animation of a single line takes less time than animating the sectors of pie chart or the bars of a bar chart one after another. Obviously, less frames lead to shorter videos and smaller file sizes. Rendering a single frame (1.924 seconds) takes only 7 percent less time compared to the other chart types.

5.2 Experiment: Throughput

When setting up the current ChartFlight server we wanted to know how fast this hardware configuration (Setup S) renders full videos and therefore, we measured rendering times of different example scenes. In contrast to the experiments discussed above, for this experiment we did not use test scenes, but real-world examples, which means we searched the web for freely available statistics to visualize them using ChartFlight. Altogether, we rendered six videos in two different resolutions. Table 4 shows our measurements ordered by video length.

	Video 1	**Video 2**	**Video 3**	**Video 4**	**Video 5**	**Video 6**
Video length	0:02:34	0:04:02	0:04:09	0:04:16	0:05:18	0:05:55
Rendering						
800x600	1:04:56	1:41:21	1:43:50	1:49:32	2:24:35	2:28:04
1024x768	1:32:32	2:24:49	2:27:10	2:36:06	3:24:42	3:27:59
Ratio	142.5%	142.8%	141.7%	142.5%	141.6%	140.5%

Table 4: *Rendering times for six different real-world examples of various video length. Therefore, we considered the results for two different resolutions and calculated their ratio.*

Based on our experimental data we extrapolated how many videos ChartFlight is able to render per day. Our videos have video length of 4:22 minutes on average. For rendering all jobs in a row the rendering engine would take 11:12:18 hours for a video size of 800x600 pixels and 15:53:18 hours for 1024x768 pixels. Thus, considering average videos of 4:30 minutes, ChartFlight is able to render about 12 videos of smaller and 8 videos of higher resolution per day. Videos of smaller resolution are often sufficient, for example, when integrating them into presentations or uploading them to video hosting web sites.

ChartFlight currently uses the rendering engine of Blender 2.46. Recent builds of the upcoming versions 2.5/2.6 implement optimized ray tracing techniques [Blender Ray Tracing 2009], i.e. rendering is up to ten times faster. To find out how much these optimizations will affect scenes generated by ChartFlight, we manually modified a high quality version of the cancer incidence example (Figure 1, e) in such a way that both versions of Blender (2.49 and 2.5 Alpha 2) are able to render it. Finally, we chose hardware setup D and measured slight improvements per frame. When a stable release is available, it simply requires some extra effort to port our Python scripts to the new Blender Python API.

Independently from current or upcoming versions, Blender provides a variety of different render settings, which may further improve the rendering speed. In a small pre-study we investigated the most important settings, for instance, those that improve performance on multi-core CPUs. For each option we measured rendering times for different inputs. As test scene we used the election results example shown in Figure 2. We found that while keeping the quality of the rendered video, with adapted settings the ray tracer of Blender saved between 12 and 25 % of the time.

6 Related Work

Related work can be roughly categorized into tools for creating visualizations, videos that have been generated, and interactive applications that provide a data set a user can explore. Furthermore, according to the categories presented in [Ghanam and Sheelagh 2008], the visualization ChartFlight produces may be considered as a combination of an abstract 3D visualization due to the displayed charts and, in a wider sense, a Virtual Environment visualization due to the real-world ground plane.

Although ChartFlight visualizations do not support user interaction and due to ray tracing frames cannot be rendered in real-time, it still has similarities—especially in terms of system architecture—with interactive approaches like [Yoon and Neumann 2000] or [Noimark and Cohen-Or 2003]. In comparison to ChartFlight these approaches provide general techniques for remote rendering and enable interactive 3D graphics for low-performance hardware devices. While the former approach also requires image rendering on the client side the latter one uses remote rendering exclusively. Rendered frames are then translated via MPEG-4 streaming.

code_swarm is a tool that produces videos of software evolution. More precisely, it visualizes the history of commits in a software archive focusing on developers and the files they committed [Ogawa and Ma 2009].

Gapminder is a project that is based on a web application called *Trendalyzer* (Gapminder World), which visualizes statistical time series of different nations. Using Trendalyzer one can interactively compare different statistics from a given pool, relate them to each other, and animate their changes over time. Moreover, Gapminder provides videos of lecturers who present their observations using Trendalyzer [Rosling et al. 2004].

The idea of *gCensus* is to utilize Google Earth for visualizations. Users simply select statistical data from a given pool via a webform and generate a file. After importing this file into Google Earth they can browse the selected data interactively [gCensus 2007].

By combining Google Earth with GPS data the *GPS Visualizer* helps users to create maps and profiles. In addition, geographic data can be visualized, which basically means that users define locations on a map and connect data to it (e.g. business locations, geotagged photos) [GPSVisualizer 2009].

A commercial tool that is probably closest to the idea of ChartFlight is called *Easy Chart 3D GEO*. In contrast to our approach this product is a desktop application that mainly produces overview images of a map and the provided data displayed as charts. Nevertheless, users of Easy Chart 3D GEO may create interactive standalone visualizations as Windows executables or videos showing charts separately [Easy Chart 3D GEO 2009].

The *CyberNet* project is an interactive application developed for visualizing dynamic data in a three-dimensional virtual world. While ChartFlight works with a fixed amount of data, CyberNet constantly gathers new data and updates the visualization. Therefore, it uses techniques for automatically mapping information onto visual metaphors. These may be both real-world (city, solar system) and abstract (cone-tree) metaphors depending on the users' choice [Santos et al. 2000].

Finally, the videos produced with ChartFlight also bear some similarity to weather forecast flights, which are sometimes used in weather forecasts on TV. They show animated weather effects during a flight over a 3D map typically from one city to another or from coast to coast.

7 Conclusion

In this paper we have introduced the chart flight metaphor and our web application, which allows end users to generate presentation videos for their own data sets. We also presented some implementation details and the results of two experiments to assess performance of our current implementation.

While our current hardware setup is able to render several videos per day, it does not scale for access by larger numbers of users. However, to scale our web application we have several possibilities:

- Increase number of rendering servers: With slight modifications it is possible to distribute renderings of full videos over different computers and keep the main server for creating preview images and providing the web interface.
- Enhance rendering process: For instance, those frames that just show a chart for a certain period of time without any animation could be rendered only once and afterwards duplicated. According to the small pre-study at the end of Section 5.2), running a detailed study with different render and also material settings in Blender could further improve rendering speed.
- Replace Ray Tracer: As described in Section 5.2 replacing the old rendering engine of Blender may improve the performance of ChartFlight. Due to the scripting functions of Blender, it is also possible to use other rendering engines, which might outperform even the new ray tracer of Blender 2.5.

While we have used charts to depict information at the different locations on a map, our approach can be adapted to many other kinds of diagram types and data visualization techniques. Thus, for instance, at each location relations could be visualized using graph visualizations or multivariate data could be visualized with parallel coordinates. Thus, the chart flight metaphor can be seen as an instance of the more general metaphor of data flights.

ChartFlight has not extensively been used in practice so far. Therefore, future goals are to conduct two different user studies. Firstly, a usability study may give insights on how test persons make use of the web-interface of ChartFlight. Secondly, a study that investigates the effectiveness and usage of videos generated by ChartFlight and compares it to common static alternatives could help users to choose an appropriate visualization technique for their data.

References

BLENDER RAY TRACING, 2009. Blender - Ray Tracing Optimization. http://www.blender.org/development/release-logs/blender-250/ray-tracing-optimization/ (accessed Apr 2010).

BLENDER, 2009. Blender. http://www.blender.org (accessed Sep 2009).

EASY CHART 3D GEO, 2009. Easy Chart 3D GEO. http://www.geobrush.com/ (accessed Sep 2009).

EUROPEAN COMMISSION, 2009. European Commission Eurostat. http://ec.europa.eu/eurostat (accessed Aug 2009).

GCENSUS, 2007. gCensus - Free Online GIS. http://gecensus.stanford.edu/gcensus/index.html (accessed Sep 2009).

GHANAM, Y., AND SHEELAGH, C. 2008. A Survey Paper on Software Architecture Visualization. Technical report, University of Calgary, June. http://hdl.handle.net/1880/46648.

GPSVISUALIZER, 2009. GPS Visualizer - Do-It-Yourself Mapping. http://www.gpsvisualizer.com/ (accessed Sep 2009).

NOIMARK, Y., AND COHEN-OR, D. 2003. Streaming Scenes to MPEG-4 Video-Enabled Devices. *IEEE Computer Graphics and Applications 23*, 1, 58–64.

OGAWA, M., AND MA, K.-L. 2009. A Design Study in Organic Software Visualization. *Proceedings of IEEE Transactions on Visualization and Computer Graphics (InfoVis 2009) 15*, 6.

ROSLING, H., RÖNNLUND, A. R., AND ROSLING, O. 2004. New software brings statistics beyond the eye. In *Proceedings of OECD World Forum on Key Indicators, Palermo*.

SANTOS, C. R. D., GROS, P., ABEL, P., LOISEL, D., TRICHAUD, N., AND PARIS, J. P. 2000. Mapping Information onto 3D Virtual Worlds. In *Proceedings of IEEE International Conference on Information Visualization*, 379–386.

STATISTISCHES BUNDESAMT, 2009. Statistisches Bundesamt Deutschland. http://www.destatis.de (accessed Aug 2009).

YOON, I., AND NEUMANN, U. 2000. Web-Based Remote Rendering with IBRAC (Image-Based Rendering Acceleration and Compression). *Comput. Graph. Forum 19*, 3.