Advanced Shot Chart

Augustinas Dankevicius
The University of Nottingham
Nottingham, UK
Psxad4@nottingham.ac.uk

INTRODUCTION

Probably the most important aspect of the game of basketball is player/team shooting tendencies. To win, a team has to score more points than their opponents and it is important to maintain a high shooting percentage. To do so, it is necessary to know the locations that have the highest chance of player making a basket, also known as shooting "Hot-Spots". To determine "Hot-Spot" locations, player statistics need to be analyzed by comparing number of total shots and number of made shots for a given location. After best shooting positions are known, the game plan can then be altered to ensure that each player attempts a shot from locations with a high chance of making the basket. Players take many shots over the course of the season, each one from a different location than the previous. Because of this, visualizing shooting tendencies is very useful to gain required insight from the information. One common way of visualizing basketball shooting information is the Shot Chart.

A typical Shot Chart contains a 2D outline of the court onto which each shot attempt/make is plotted with an appropriate shape or color (scatterplot). While this is useful to review performance for a single game, it can be difficult to see trends in the data over time due to over-plotting of points. This can be overcome by splitting the shots into bins of given size. The bins can then be plotted and encoded with size and color to indicate total shots and shooting percentages respectively. Another approach is to use heat map, where each zone of the court is encoded with color based on shot frequency for a given location. Both of these approaches present information well, however, often they do not offer filtering options such as statistics for a given shot distance or quarter. As a result, the aim of this project is to create an interface for visualizing basketball statistics, while also exploring novel ways of plotting them.

RELATED WORK

The Wizards' shooting stars [1]

This visualization was inspired by the fact that the Washington Wizards attempted the most mid-range shots of any NBA team in 2013-2014 season, while shooting below league average from that range. The aim of the project was to visualize this statistic by plotting curves of every shot attempt from the season (Figure 1 a). The plot is presented on a side view of the court chart. Curves have a slight opacity, which makes ranges with more shot attempts stand out. The interface contains options to filter the plot by player, made/missed shots. It also contains the classic top-down shot chart as well as shooting by zone chart. To help interpret the curve plot, shot distribution and shooting percentage by distance statistics are presented under the chart. On the other hand, the visualization only offers statistics for a single team. The main point of interest from this visualization to the project is the curve chart. While it does a good job at emphasizing the frequency of shots from a given distance, it has a disadvantage of being two-dimensional side-court view. This makes it difficult to see the distribution of shots from all angles of the court.

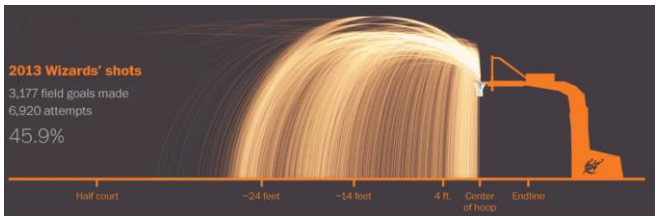

Figure 1 a. Wizards' shooting stars visualization.

BallR: Interactive NBA Shot Charts with R and Shiny [2]

This API offers different types of NBA shot visualization for every player since 1996. It has several advantageous qualities available. 1) Filtering by: regular season/playoffs, season, shot zones, shot angles and distances, made/total shots. 2) User can select chart from: a) Scatter plot, b) Hexagonal [3], (Figure 1 b), c) Heat Map. Main point of interest from this project is the hexagon chart. The hexagon plot successfully encodes two variables (shot frequency and percentage) into a single plot by using area and colour encodings. On the down side, it can only plot statistics of individual players and does not show exact statistics for a given hexagon. These shortcomings will be addressed in this report.

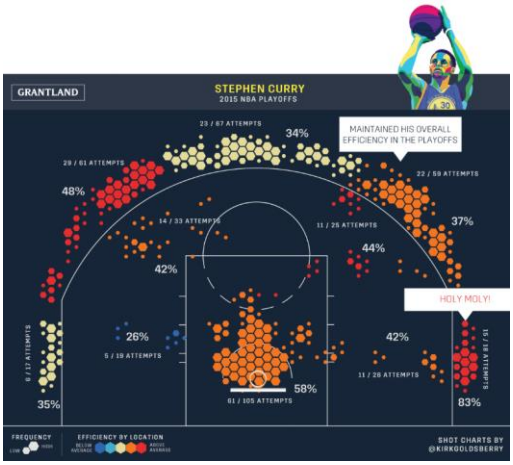

Figure 1 b. Showing the hexagon plot.

METHODOLOGY

As mentioned previously, the chief goal of this project is to create a novel basketball shot visualization technique and provide an interface for user interaction. This section aims to explain the design choices made and methodology used to implement it. First, design ideas will be presented, followed by implementation and pseudocode of each algorithm.

In both cases, background of a court was added by importing and skewing an SVG image of its' outline. This proved to be a much easier approach than drawing the outline using SVG draw commands.

Design of Curve plot

The main idea behind the curve plot is simple: Take the plot seen in [1] and create a three-dimensional view for it. Reasoning behind it is simple as well: a 3D view will allow a more comprehensive view of shot frequency, by including all angles of the court.

First step in creating the curve plot was to skew each data point by a chosen angle, resulting in a 3D effect (Figure 2).

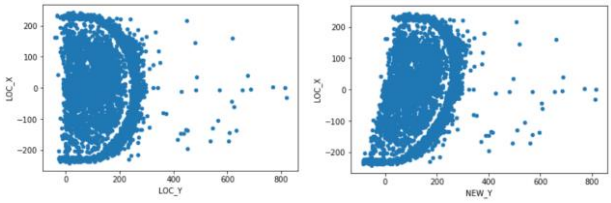

Figure 2. Showing data points before and after skewing

Next, from each data point, a curve would be drawn. The control points of the curve path were calculated as follows (Figure 3):

1. Start at shot location (X, Y)
2. Go up 200px (X, Y + 200)
3. Go above the basket (0, 200)
4. Go to the basket (0, 100)

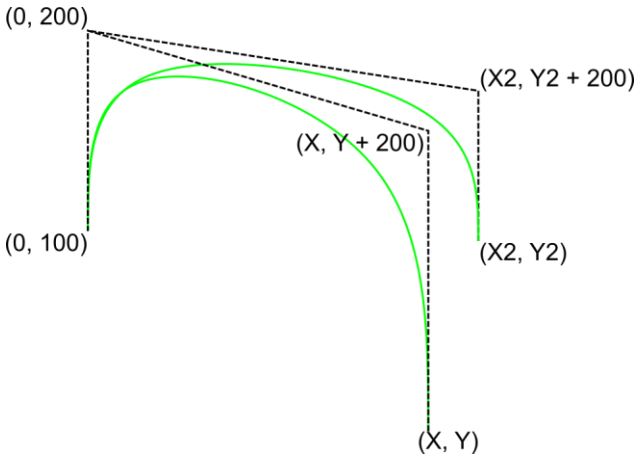

Figure 3. Showing curve control points example.

Note, that arc height is chosen arbitrarily and does not represent actual trajectory of each shot. Now, the curves can be plotted with chosen colour and lowered opacity. Opacity is lowered to allow stacking of curves, which would make zones with higher shot frequency brighter. This is done to reveal the shooting patterns of different teams/players (As seen in Figure 1).

It was suspected early on, that curves from further distance shots would cover closer distance curves, making them difficult to see. As a result, appropriate filtering option was needed to reveal points that are difficult to see.

To have a simple and easy to use interface, user interaction was designed in the following way:

1. User selects the team and all data for the chosen team is plotted.
2. User can then (optionally) interact with the plot by filtering:
    a. Individual player shots
    b. Made/All shots
    c. Shots from chosen distance only

Design of Column plot

Main goal of the column plot is to combine two variables (shot frequency and shooting percentage), that would normally require two separate plots, into a single interactive plot. This visualization technique is based on the hexagon plot seen in [2]. In the hexagon plot, shooting percentage is encoded with colour, while shooting percentage is encoded with area of the hexagon. In case of the column plot, shooting percentage would also be encoded with colour and shot frequency - with column height. As well as overcoming weaknesses of the previously seen hexagon plot, the column chart also presents a novel way of visualizing basketball statistics.

As with the curve plot, first step was skewing the data. Next, data is binned into an array of chosen cell size. To create a 3D effect, the columns are created by combining two ellipses (opacity = 1) with a rectangle (opacity = 0.5, height = n. shot attempts) between them (Figure 4).

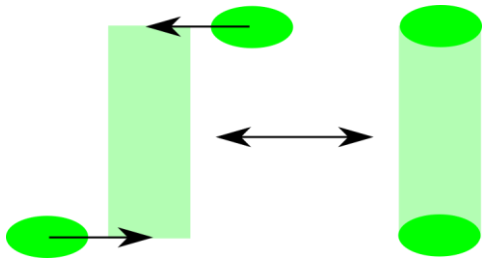

Figure 4. Showing how 3D column effect was created.

As with curve plot, initially all data for a chosen team is plotted with the column plot. The user can then select to display individual players plot and filter the chart by shooting percentage. Because colour encoding is used, a legend indicating values for each colour was needed. Optional interaction was also added to show tooltip with exact statistics for a given column, when mouse is hovered on it.

Pseudocode of plots

```
Function plotCurves (data) {
    For each point in data:
        Create a path element between points:
        {[data.Y, data.X],
         [data.Y, data.X + 300],
         [0, 300],
         [0, 200]}
        Set parameters:
        Style:
          stroke, stroke-width, fill, opacity
        Animation:
          Stroke-dashoffset, stroke-dasharray
}

Function plotCols (data) {
    For each point in data:
      Create ellipse element at the center of the data point;
        Set fill colour based on data value;
      Create rect element, align bottom edge with center
  axis of the ellipse;
        Set fill colour based on data value;
        Set opacity to 0.5;
      Create ellipse element at the center of the data point,
align center axis with top edge of rect;
        Set fill colour based on data value;
}
```

IMPLEMENTATION

Data pre-processing

Initially, the plan was to fetch the statistics directly from NBA. However, due to recent browser restrictions it was not possible and data had to be processed and saved locally. Because of the vast amount of data available (yearly statistics from 1996 – present), it was decided to use data from a single season only. Data was obtained using Python's nba_api [4] and processed using Jupyter Notebook. Following data pre-processing steps were then taken:

1. Team names and ID's were fetched and saved.
2. Shot statistics (made/attempted) of every game from the 2017-2018 season were fetched.
3. A new Y coordinate column was created in the data frame to add skew effect to the data.
4. Columns that would not be used for visualization were deleted, to reduce the size of data and improve loading times.
5. Data frame containing all the shots was saved in csv format for visualization purposes.
6. Using R the csv file was used to create a grid of cells containing league averages of that cell.

D3.js implementation

First, basic user interface was created (styling was applied at the very end of the project and will not be discussed in this paper), using CSS grid (Figure 5).

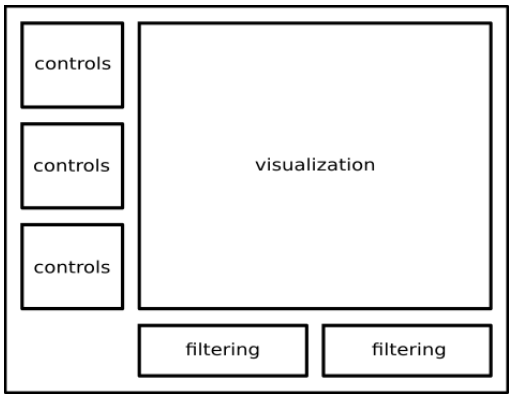

Figure 5. Showing grid arrangement of the interface.

Loading and getting data

As mentioned already, a single dataset containing all shots from 2017-18 season was used. This dataset contains more than 210,000 rows and has a size of 18Mb. It would be very unpractical to try to visualize it at once. Thankfully, D3 has a built-in filtering function, which allows us to extract rows of interest from our large dataset. For this purpose, two helper functions were created: one for team data and another for individual player data. Filtering a dataset of over 200,000 rows in JavaScript was a concern at first. Alternative plan here was to split data into smaller team datasets in the pre-processing stage, and load them on demand. However, filtering process took less than a second in each case, allowing us to load one dataset and use it as required by filtering out unnecessary data. In every case, the dataset is loaded and a block of functions to modify and visualize it are created.

Data point scaling

The original data has a range from -250 to 250 and from 0 to ~500 in X and Y coordinate respectively. Note, that Y coordinate does not have many data points beyond values of 250, which is more or less the three point line. As a consequence, Y coordinate is cut-off at 300. To be able to fit these points into our visualization window, a scaling transformation of data was needed. Following [5], two functions were created for both coordinates. Each function takes data points as input and maps them to appropriate visualization window dimension. Note, X scaling function scales Y values from data, while Y function uses X coordinates. This is because the shot chart was flipped to represent side view. Both functions were later fine-tuned to ensure consistent placement of graphs for different plots.

Plotting curves

First step in plotting the curves was to create a curve generator function. Curve generator is a D3 line generator function [6], with a specified curve function. For this, curveBundle function was used with a beta parameter of 0.8, which was determined by experimentation. The goal of this function is to take data points as input and draw a spline (See green spline in Figure 3.) between them.

Next step in producing the plot is to use the curve generator on each instance of a dataset. To do this, D3 data.enter() function [7] was used, which works as follows:

1. Select DOM element to be created (in our case, this is path)
2. Specify dataset to be used for creation of each element.
3. Use enter() function to enter the dataset. This determines, how many elements need to be added.
4. Append element of interest (path - curve).
5. Set attributes, based on data (in our case, this is the path coordinates, or the output of the curve generator function).
6. Set remaining attributes, such as fill, opacity and stroke.

While this methods works great for smaller datasets (<500 instances), the dataset for each team has around 7000 rows. This means that each time plot function was called, 7000 SVG elements needed to be generated. Even on a relatively powerful PC (i7, 16 GB) this would take around 10 seconds, freezing whole interface in the process. The technique that was used to overcome this flaw is progressive rendering.

Main idea behind progressive rendering is to chunk a large dataset into smaller ones, which can then be plotted much faster individually. To do this, a function was written, which takes a dataset and a slice size as input and returns an array with data slices as its' elements. This new array is then iterated through using JavaScript's setInterval function. Each slice is plotted at intervals of 50ms resulting in a much smoother performance as well as visually pleasing effect of curve accumulation.

Lastly, for more visual appeal, a transition was needed for drawing the curves. The idea here was to transition the curve from the starting point to the basket, leaving a trail behind it. To do this, Stroke Dash Interpolation method [8] was used. This method works in the following way:

1. Set the stroke of a curve to dashed type.
2. Make length of dashes and solid parts equal to original length of the curve. (total length = 2 * original length)
3. Offset the curve, so only dashed part is visible.
4. Transition from dashed to solid, creating a moving spline effect.

Plotting columns

The visual aspect of the column plot is relatively simple and will be discussed later in this segment. More important and difficult to deal with was the process of getting data ready for plotting.

Firstly, the data needed to be binned into cells. To do this, a function was created with the inputs of dataset and cell dimensions. This function iterates through the data in increments of cell size, and saves the following parameters of a given cell in a json object:

- xMin, xMax, yMin, yMax – coordinates of cell
- totalShots – total number of shots of cell
- madeShots – number of made shots in cell
- shotPercentage – made shots divided by total shots

These parameters are the same as the league averages datasets', created in the pre-processing stage. This is done to ensure easy comparison. The case of cell size in this chart is similar to choosing the bin width in a classic histogram. Higher cell size means loss of resolution, and as a result – information. Smaller cell size makes the chart unreadable. Hence, cell size was chosen, by trial and error as, 10, which means that we would end up with an array size of 50 by 30 (X – 500px, Y – 300px). Values of these cells are objects with variables described above. Next step is to use these variables to create visuals.

Similar to the procedure seen in plotting the curves, we use the data.enter method to produce the column chart. One exception is that in this case we have to create three different elements for each data point. This is simply done by calling data.enter method three times for different shapes (two ellipses and a rectangle). Parameters from the shot grid are mapped to graphics in the following way:

- xMin and xMax are used to calculate the center of the cell in the Y coordinate.
- yMin and yMax are used to calculate the skewed center of the cell in the X coordinate.
- totalShots is height of rectangle.
- shotPercentage – difference of this and the league average is used to determine the fill colour of the three elements.

Now the elements could be plotted. While plotting was much faster (less data than curve plot) it revealed a significant issue. Because elements were plotted all at once in no particular order, there was a lot of overlap as random columns were simply plotted on top of each other. To overcome this, progressive rendering was used again. This time, to deal with the order of elements on the screen. Again, we use JavaScript's setInterval and plot the grid row by row, starting from the furthest one. This method creates the desired 3D effect. Another issue was that some cells were simply empty, but would still be plotted as white ellipses. To overcome this, a filter was added before entering each shape, to remove instances containing zero shots.

To add colour scale to the chart, D3 scaleSequential [9] function was used. This function takes a number between specified domain and returns a colour code from a chosen scheme based on the value. In our case the domain range is -20 to 20, which is the range of shooting percentage difference between a player/team and league average. Chosen colour scheme is Viridis, which goes from purple on the lower end to yellow at the high end. Because we used a colour scheme for this plot, a legend was needed. For this, a function was written, which would simply use the same colour scheme to add a row of rectangles, with text indicating their value

above. This function would then be called each time a column chart was produced.

Lastly, to make the plot more interactive, a tooltip is added. The tooltip provides more precise information about the shooting statistics for a given cell. The script to create a tooltip was used from [10]. Interactivity works in the following way:

1. User hovers mouse pointer on a selected column.
2. Rectangle element's from select column opacity is increased to 1, emphasizing the selected column.
3. Tooltip appears above the columns, showing statistics.

Filtering implementation

The first available and perhaps the most straightforward filtering option is choosing a player. Once a player button is clicked (for both curve and column plots), a function is called. This function has two cases, one for each plot type. In case of the curve plot, all curves on the screen are selected, and ones that don't match the chosen player are hidden with an appropriate CSS class. Because there are a lot less shots from individual players, the curves with the default opacity were quite difficult to see. As a result, the opacity of a highlighted player was doubled. On the other hand, if chosen plot is column, the existing column plot is completely removed and a new shot grid, containing chosen players' statistics is created and then plotted.

Next option is selecting the made/all shots buttons, for curve plot. Each button has a corresponding function built for it. Similarly to player filtering, in both cases all curves are selected. Next, curves are filtered out by made shot flag variable, which leaves us with only missed shots selected. Finally, using Stroke Dash Interpolation method again, curves indicating a made or a missed shot are hidden/revealed accordingly.

Next filtering option implemented for the curve plot is the shot distance filter. As mentioned in the methodology section, over plotting of the curves is an issue. Once every shot is plotted, shots closer to the basket are quite difficult to see due to large amount of shots from the three point line. To deal with this, a simple HTML slider is implemented. The range of the slider is from 0 to 25, which is approximately the distance from the basket to just above three point line. On change of slider, a function is called with sliders' value as input. Very similarly to the player filtering, the shots that match the distance argument remain on the screen, while others are hidden with a CSS class selector. This makes the plot much easier to interpret.

Final filtering option was implemented for the column chart. The form of the filter is two sliders which allow choosing boundaries of the shooting percentages to be displayed. This filter is very useful for emphasizing the best and the worst shooting locations. To implement it, values from two sliders are passed into appropriate functions, which hide/reveal columns that match the criteria.

RESULTS

The aim of this chapter is to present and describe the resulting plots. First, curve plot will be presented, followed by the column plot. Both team and filtered out results will be discussed. In both cases, the visualization window is the same, and has the same background graphic of the court outline.

Curve plot

The curve plot displaying all shots of the Golden State Warriors can be seen in Figure 6. The chart contains over 6000 splines with lowered opacity, which makes zones with higher shot frequency stand out more. Overall, the plot looks visually pleasing, although slightly overwhelming. On the downside, unpopular shot locations are barely visible. It can be seen that the vast majority of the shots come from distances very close to the basket as the plot is brightest around that area. The chart also makes it difficult to interpret the mid-range area due to over plotting. There is also evidence of potentially misleading information, where curves are stacked because of the view angle (top right corner, three point line). This makes the illusion of that area being more popular than, say, same location on the opposing side (bottom right corner three point line), which looks a lot less dense.

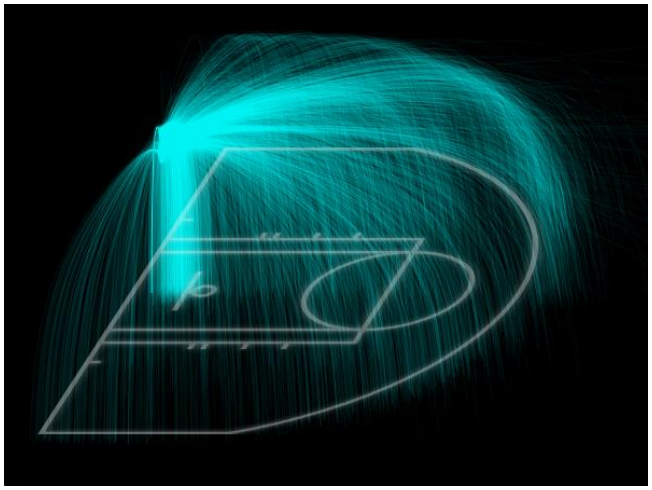

Figure 6. Showing the curve plot of all Golden State Warriors shots.

After filtering, the plot looks less overwhelming. The visualization showing all shots of Kevin Durant can be seen in Figure 7. From the plot, it is clear, that Kevin can shoot from anywhere on the court and is a very versatile player. The plot shows that, when it comes to three point shots, Kevin prefers to take one from either shoulder (top/bottom right of plot) of the court, as brightness of the curves is clearly more pronounced in those two areas. Mid-range shots and even shots close to the basket can also be interpreted more easily in an individual player plot.

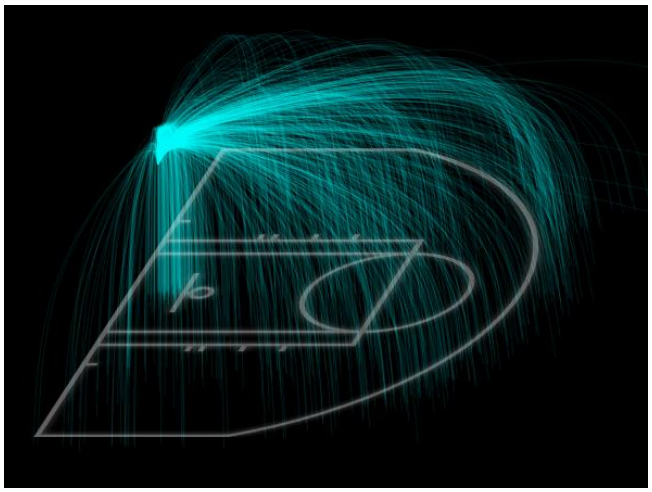

Figure 7. Showing the curve plot for Kevin Durant.

Next filtering option applied is removing the missed shot attempts. Results are quite similar to individual player filter described above. Mid-range is more visible, but still difficult to make sense of. Three point range is more pronounced, and several brighter areas indicate higher number of made shots for that location. On the down side, we do not know the shooting percentage from that location, which makes it difficult to determine efficiency of that zone.

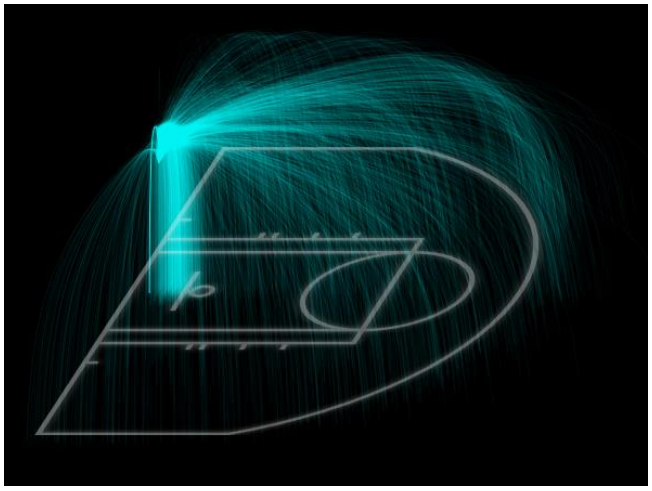

Figure 8. Showing the curve plot of made Golden State Warriors shots.

Filtering shots by shot distance allows closer inspection of shot distributions from any range. This filter emphasizes the information of interest by removing the curves that were not selected. On the downside, only the exact shot range can be chosen, which makes inspecting shot frequency of ranges (e.g. close to mid-range) impossible.

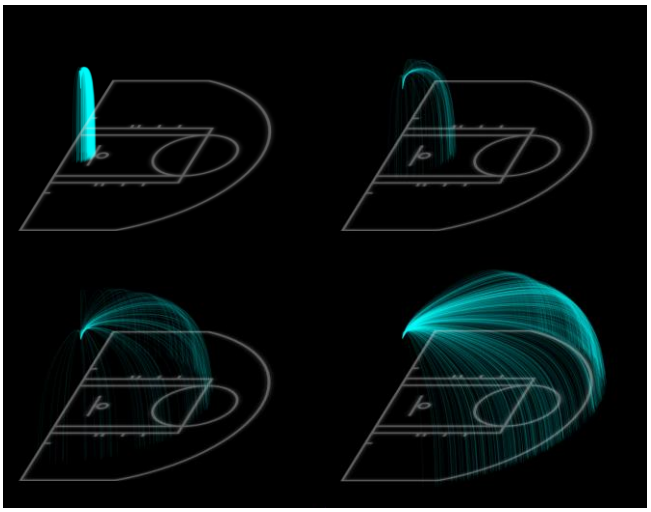

Figure 9. Showing curve plots of all Golden State Warriors shots filtered by different shot distances.

Column plot

Figure 10 shows, essentially, the same data as seen in curve plot of all Golden State Warriors shots (Figure 6). Only difference is the combination of made and total shot attempts to a single variable (shooting percentage), splitting of data into cells and of course, the plot type itself. This means, that both charts can be easily compared, which will be done later in the discussion section.

The chart itself looks rather comprehensible. Although, there are close to 1500 elements in the chart, it is still easily readable. Higher columns, indicating higher shot frequency stand out clearly, especially those close to the basket, again. One exception is, as before, the top right shoulder behind the three point line, where columns seem to be bunching up quite closely, due to the view angle. Nonetheless, chart interaction allows us to easily overcome this: when mouse is hovered, the columns immediately stand out. A minor flaw here is that smallest rectangles are hidden under their ellipses, which prevents the activation of hovering and tool tip.

Colour coding, indicating difference from the league average of the shot zone works great too. The colours on the opposite ends of the spectrum are contrasting each other well and all the elements are clearly visible. Interpreting the chart is quite straightforward as well. Highest columns, with colour code above 0 (green-yellow) indicate the best shooting locations for a given team. On the other hand, highest columns, with colour code below 0 (magenta-green) indicate locations that teams like to take shots, but can't seem to make them. In general, the more green - yellow elements on the chart, the better the team is shooting. This will be explored in more detail in the discussion section.

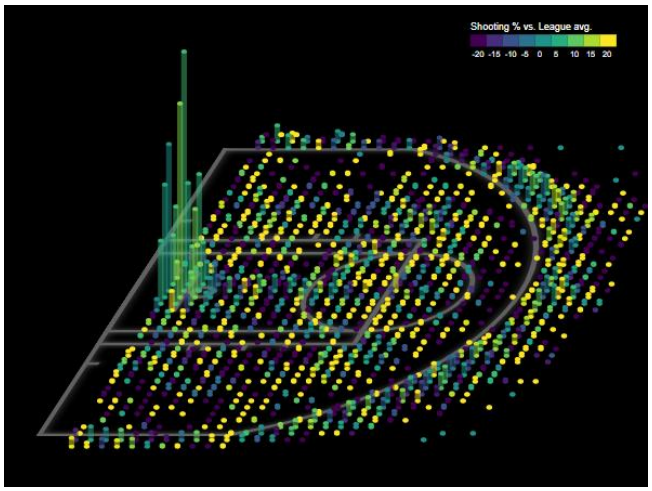

Figure 10. Showing the column plot for Golden State Warriors.

Filtering the chart by shooting percentage works well too (Figure 11). By reducing the amount of data points, we emphasize the shooting zones that the team is the most/least efficient at.

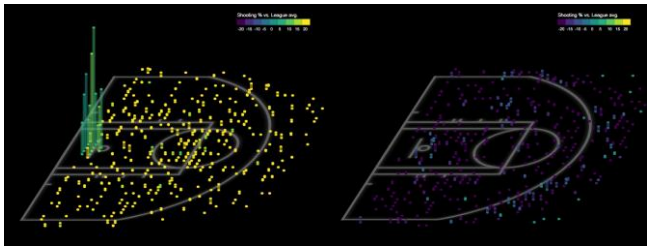

Figure 11. Showing two filtered column plots for Golden State Warriors. Left – above 70 % made shots, right – below 30 %.

EVALUATION

The aim of this chapter is to evaluate the performance of the interface. A user study would have been very useful to determine the efficiency of the user experience, however, time constraint did not allow this. As a consequence, a critical reflection will be performed instead, with as little bias as possible. Two main topics will be covered: User Experience in a form of cognitive walkthrough; Performance of the visualization – loading times and general website performance.

User Experience

Upon opening the website, we see the chart area, control buttons for chart type, team and player as well as corresponding text paragraphs. The text boxes are numbered, suggesting a sequence in which buttons should be clicked. At this point, user has several options (response of the action in the brackets):

- Click one of the chart type buttons (depending on the button clicked, appropriate filtering options appear below the visualization window, button remains selected until next mouse click)
- Click select team button (team dropdown menu opens, and a user can select a team, name of which then replaces text on the button)
- Click select player button (same as above)

Clearly, team and player buttons should not be clickable as at this point they have no purpose and can be confusing to a user. After selecting the chart type, filter options appear, but again, serve no purpose and user may assume that next logical action is to click one of the filter options. After chart type button is clicked, it should remain toggled to indicate that this actions has been completed, otherwise, user is not aware.

Assuming user selects the Curves chart type and then selects the team, the visualization is rendered in the visualization window on the right. Now, user has the following options:

- Click select player button (if a player is chosen from the dropdown, the chart updates, else – nothing happens)
- Click made shots filtering button (chart is updated with an animation)
- Click all shots button (chart returns to previous state with an animation)
- Drag the slider on the right (chart is immediately updated to show curves at proportional distance to the slider's location, exact value of slider is not shown)
- Click reset button below the slider (chart return to original state)
- Click chart type button again (nothing happens, user needs to click team button again for visualization to update)

The user interface has several faults at this point: Lack of feedback on user actions; Appropriate next action is almost never clear; Unclear how to restart the process without reloading the website.

Selecting Columns chart type and then a team, different visualization is rendered. Same UI issues described above are present, with the addition of an unintuitive slider filter method.

To compensate for shortcomings of the UI and make user experience slightly more clear, an instructional text box is added to the page.

Performance

To measure performance of the interface, Google Dev tools in Chrome were used. Results of Curve and Column plots are summarized in Figures 12 and 13 respectively.

Because of significantly larger number of elements, the Curve chart type is clearly more resource-intensive. Curve chart took nearly five times longer from start to finish than the Column chart. Most intensive process was the rendering of the SVG elements (setting of styling attributes). It was followed by scripting, as the 200,000 long dataset was being filtered through. Lastly, painting of the graphics itself took 3.2 seconds to achieve.

On the other hand, producing Column plot took only 3.7 seconds, slightly longer than just the scripting process of the Curve plot. In this case, the longest process was the scripting – iteration through and binning of the data.

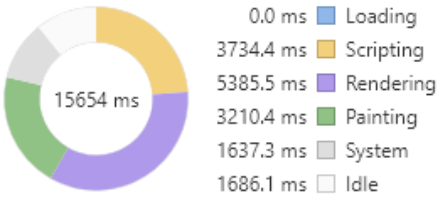

Figure 12. Google Dev summary of rendering of curve plot.

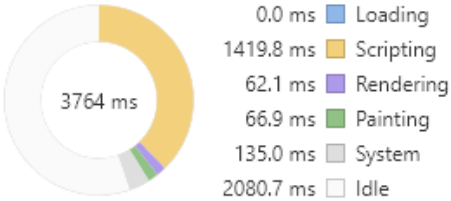

Figure 13. Google Dev summary of rendering of column plot.

DISCUSSION

The aim of this section is to give the reader a critical overview of the project, including what went well and what could have been done better. First, user interface will be briefly discussed. Second and third, both types of visualization will be addressed, with discussion on advantages and disadvantages of each.

User Interface

As discussed in the evaluation section, the current user interface has a few faults. However without actual user feedback it is quite difficult to judge just how ineffective it really is. On the bright side, even though it is not perfect, everything works and there are no critical errors. More effort could have been spent on designing and fine-tuning it, but instead that effort was put into the visualizations, which in writer's opinion was the most important part of the project.

Curve plot

To determine effectiveness of the curve plot, it was decided to compare teams that had very different seasons. Six teams for chosen for comparison, two with best records in each conference, two with the worst, and two with intermediate records from each conference. First two teams are Houston Rockets with the best overall record (65 Wins, 17 Losses) and Toronto Raptors (59/23); Second pair is Phoenix Suns with the worst overall record (21/61) and Atlanta Hawks (24/58); Third – Denver Nuggets, with an intermediate record (46/36) and Washington Wizards (43/39). Plots only show made shots, because, as seen earlier, it is more informative than total shots. All plot pairs can be seen in Figures 14-16 respectively.

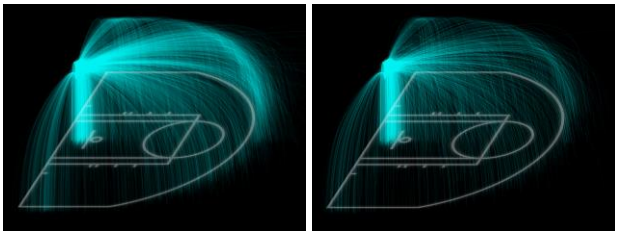

Figure 14. Showing made shots by Houston Rockets (left) and Raptors (right).

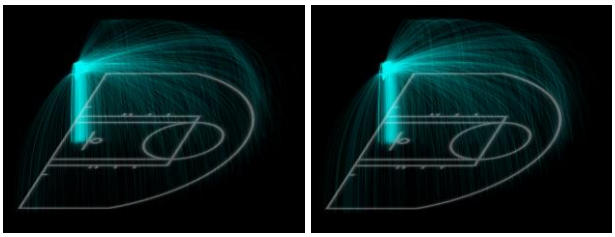

Figure 15. Showing made shots by Phoenix Suns (left) and Hawks (right).

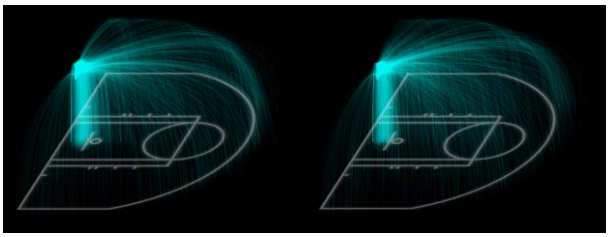

Figure 16. Showing made shots by Denver Nuggets (left) and Wizards (right).

The difference in shooting tendencies of some teams are very clear. Houston Rockets shot and made the most three pointers in the season and it is clearly evident in the plot. On the other hand, the Raptors', which also had a great season, plot does not look very different from Suns (worst record). The Raptors were a better shooting team, but it is hardly noticeable in the visualization, making it less effective for team comparison. In general, all but Houston Rockets plots look very similar to each other. Nonetheless, the chart still shows trends in team shooting and looks visually pleasing. It could be used in situations where style, rather than statistical knowledge, is more important. Inefficient visualization of information coupled with high usage of computer resources makes possible uses of this chart quite limited. Potential applications: basketball video games, illustrations in sports magazines.

Column plot

Following the same approach seen in discussion about the curve plot, the column charts can be seen in Figures 17-19.

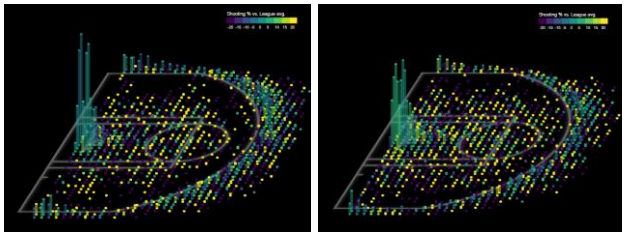

Figure 17. Showing made shots by Houston Rockets (left) and Raptors (right).

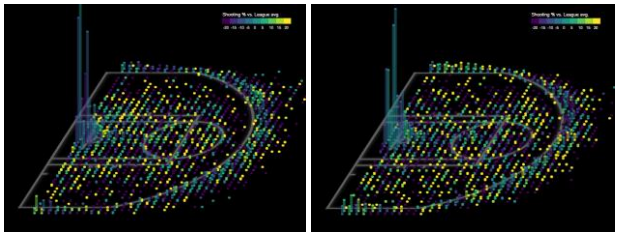

Figure 18. Showing made shots by Phoenix Suns (left) and Hawks (right).

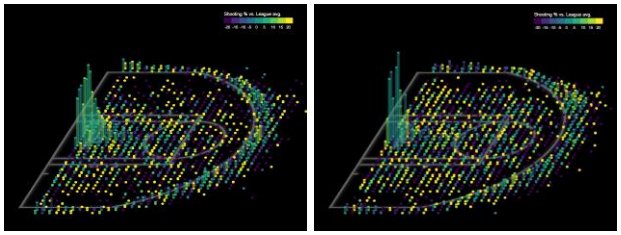

Figure 16. Showing made shots by Denver Nuggets (left) and Wizards (right).

The results from Column chart comparison are much better. This is reflected in both colour distribution as well as frequency of "tall" columns across charts. As before, the Rockets' three point range is much more pronounced, having a much bigger proportion of taller columns than the remaining charts. Two of the worst teams (Figure 18) have attempted more shots from close range than other teams, but also shot below league average, indicating poor performance. Overall shooting tendencies are much clearer when compared to the curve plot. For example, Denver Nuggets have been very efficient from close range, while also taking a variety of shots from that range. On the other hand, the column chart does not offer much more insight than the hexagon plot. It essentially portrays the same information in a different way. Potential uses here include: video games and sports magazine illustrations, as before, and also as a replacement for classic shot chart in areas, where novelty is acceptable.

CONCLUSIONS AND FUTURE WORK

To summarise, this project presented two novel ways of visualizing the shot chart statistic of basketball: the Curve plot and the Column plot.

The Curve plot has a novel visual appeal, but does not portray information very well. Nonetheless, it could potentially be used in industries that do not require exact information and rely more on style.

The Column plot, on the other hand, has a similar visual appeal as the Curve plot, but also does a much better job at visualizing information. As a result, in addition to the same areas as the Curve plot, it can also be used as a replacement for the shot chart in statistical areas.

In terms of future work, there is plenty of room for improvement. The bare bone algorithms of both visualizations work well and can be easily modified. Firstly, the user interface of the project was mostly focused on functionality and was not designed properly. Secondly, both charts could be implemented with different statistics, for example curves could be modified to represent shot accuracy with colour, or different height of the spline. Experimentation with different approaches is encouraged. Column chart can be improved by adding a different statistic, such as actual shooting percentage, rather than difference from league averages only. In both cases, more filtering options could easily be added, such as statistics by quarter or time remaining, to emphasize performance in clutch times. Option to choose the year of the season and regular/playoff options would also be logical to add. Optimization of the plotting for better performance, especially in case of the curve plot, would be very useful.

REFERENCES

1. Todd Lindeman, L.G. *The Washington Wizards' shooting stars.* 2014; Available from: http://www.washingtonpost.com/wp-srv/special/sports/wizards-shooting-stars/.
2. Schneider, T.W. *BallR: Interactive NBA Shot Charts with R and Shiny*. 2016; Available from: https://toddwschneider.com/posts/ballr-interactive-nba-shot-charts-with-r-and-shiny/.
3. Goldsberry, K. *Golden State Warriors Illustrated*. 2015; Available from: http://grantland.com/the-triangle/golden-state-warriors-illustrated/.
4. *nba_api*. Available from: https://github.com/swar/nba_api.
5. Scale functions. Available from: https://www.d3indepth.com/scales/
6. Shapes. Available from: https://www.d3indepth.com/shapes/#line-generator
7. Enter and Exit. Available from: https://www.d3indepth.com/enterexit/

8. Bostock, M. *Stroke Dash Interpolation.* 2019; Available from: https://bl.ocks.org/mbostock/5649592

9. Managing colors in d3.js. Available from: https://www.d3-graph-gallery.com/graph/custom_color.html

10. Gotz, D. *Using d3-tip with D3.js Version 4.* 2017; Available from: http://bl.ocks.org/davegotz/bd54b56723c154d25eedde6504d30ad7