Analytics of Realtime Soccer Match Sensor Data with JavaScript and WebGL—Reprocessed and Visualized for Web Browser or Command Line Consumption

Martin Kleppe
Ubilabs GmbH
Juliusstr. 25
22769 Hamburg, Germany
kleppe@ubilabs.net

ABSTRACT

This project demonstrates the applicability of JavaScript to provide complex analytics over high velocity sensor data along the example of analyzing a soccer match. Hybrid code—a program that runs at the command line or in the browser—is used to visualize results and stream aggregated statistics in real-time. During the game, the data is being collected using wireless sensors embedded in the ball and the players' shoes. Real-time analytics include the continuous computation of statistics relevant to the audience (ball possession, shots on goal) as well as to the coaches and team managers (running analysis, position heat maps).

Keywords

Real-Time, Complex Analytics, JavaScript, Data Streams, Sport

1. INTRODUCTION

Detailed sports game analytics are of high interest and relevance in today's professional sports leagues. Spectators are provided with additional statistics, such as the number of shots, movement analysis, or percentage of ball possession per player or team. Furthermore, statistics provide useful information for coaches and team managers about a players' performance during the match or in certain situations, and give insights about opponents which could lead to modification in tactics.

Although automated solutions such as high resolution video analysis are desirable, as they generate more detailed information more quickly, most sport game statistics are processed manually to date. Unfortunately, insights gained through image-based solutions are limited due to the lack of higher image resolution and frame rate..

For the ACM DEBS 2013 Grand Challenge the Fraunhofer IIS set up a real-time locating system on a soccer field in the Nuremberg Stadium (Germany). Every player and the ball were equipped with wireless sensors that produce high velocity sensor data at a total rate of about 15,000 position events per second.

In the following I will outline my submission that uses continuous computation of statistics in JavaScript to generate real-time analytics and interactive visualizations of the game such as ball possession, shots on goal, running analysis of all players and the two teams. This approach will be called "hybrid" because the same system is used to visualize the game in a web browser and to output multiple streams on the file system.

2. RELATED WORK

Automated sport analyses heavily depend on video systems that capture the game, compute differences between images and then use the remaining color information to track players and ball. Another approach is to equip the players with sensors that collect position data over time. That data can be combined with video processing for instance, to select and zoom in on a situation where a certain player is within the opponent's penalty area. Additionally, biometric sensors that collect information about the players condition—such as heartbeat and body temperature—provide data that is used to analyze the performance during the game. Spatial game analytics (*e.g.*, heat maps that display the players' distribution over time) are one of the most useful applications, because they can be used to optimize the team distribution for a specific game.

Using JavaScript to analyze that amount of data in real-time was not possible until Node.js, which is built on top of Google's open source JavaScript engine V8 has proven that it is suitable to build high-performance network programs because of its event-driven and non-blocking nature. Furthermore, HTML5 features, such as Canvas and WebSockets are used for tools like real-time monitoring systems. When visualizing three dimensional content, WebGL—a web standard for a low-level 3D graphics API based on OpenGL ES 2.0—is ideal to render the objects in the browser. It is used for MMOGs (Massively Multiplayer Online Games), efficient rendering of 3D models and interactive visualization of volumetric data.

3. METHODOLOGY

The following submission for the ACM DEBS 2013 Grand Challenge started with the question: "Is it possible to read and analyze the provided input data stream and visualize

it in the browser"? The first tests turned out that parsing the file and simply displaying the positions of all players and balls in 3D is possible with twenty times the actual speed, that means, a minute of the real game is replayed within just three seconds. This leaves enough time to make additional calculations. The two teams and the ball can easily be recognized by using different colors for each. Additionally the playing field and goals are drawn to visually check different game situation.

The second step was to implement the required queries. For every information a visual element was added to the browser interface to keep track of the computations and avoid gross errors. These include: A tracing line for the ball movement, the list of players with stats about ball possession, colorized sparklines for running analyses, the precalculated ball path, the possible hit target, highlight of current player, an animated acceleration bar and the current time.

As the application has to deal with large data sets, it is critical to observe the internally used memory. The Chrome Developer Tools Timeline Panel was used to detect if the scripts result into an increasing usage and then the memory leaks were evaluated and removed.

After all parts were included, the original code was extended to also run via the command line without the need of a browser. Therefore a bridge was created to either render HTML or output several streams if executed via the command line. These streams can written to a file or distributed via WebSockets. The result is a hybrid application that does both: visualize the match in a browser or share aggregated results via the network.

4. IMPLEMENTATION DETAILS

Data: The original data-stream was captured during the game and results in a 4.62 GB CSV input file. Position updates for sensors in players' shoes and goalkeepers' hands are provided with a frequency of 200Hz. The sensors in all balls update with 2000Hz. A record contains the following data: sensor id, timestamp in picoseconds, position (in a three-dimensional coordinate system) of the sensor in mm, |v| (in µm/s), vx, vy, vz describe direction by a vector, and |a| (in m/s□), ax, ay, az describe the absolute acceleration and its constituents in three dimensions.

Queries: Based on this data, the following queries are required for the ACM DEBS 2013 Grand Challenge: Running analysis, ball possession, heat maps and shots on goal. Furthermore, goal detection and when the ball is out were included.

Analysis: All entries of the original data stream are distributed to several JavaScript objects based on a mapping table that includes more information about the sensor type. Whenever a ball position update was detected, the system uses the last known position to check for a goal or whether the ball has left the field. If the ball acceleration peaks, it detects the associated player and computes the shot target based on the current speed vector and gravity. Position updates for all players are collected to result into running statistics and heat map calculations.

Performance: To optimize performance and to avoid long running scripts, plain JavaScript with almost zero dependencies was used. The only exceptions are the library fishbone.js and Three.js—a wrapper that handles WebGL. The code was organized into two kinds of modules: Simple class-like modules with prototype-based inheritance that were used for fast-changing game objects like players or ball and modules that handle events between these objects and the streams. A centralized runner script handles most of the time-consuming calculations within a flat lexical scope to avoid nested function and variable lookups. This was especially critical for large loops that occur when the input stream emits new records.

Aggregations: Whenever an update of one of the balls is detected, the program evaluates if this ball is active by comparing the position with the field's bounds. Then the nearest player is selected and if the ball's acceleration peaks, shots on goal and ball possessions (per player and team) are evaluated. With a frequency of 50Hz, the player's current position is recorded into a large array to generate running statistics and heat maps based on different time frames (1, 5, 20 minutes and the whole game). This is done by looping through the records multiple times per interval and comparing all dimensions to create aggregated values.

Visualization: To visualize the results in the browser, position properties of JavaScript objects are updated as new data arrives—and because WebGL is a state machine, it handles these updates very fast. Geometries for sensors are displayed as colored cubes, the field and ball paths are simple polylines and the heat map is a particle system. It uses rectangular sprites, because the rendering performance of 2D sprites is way better than updating complex geometries. The list of players is drawn as an unordered HTML list and colors are assigned via CSS. The color-coded sparkline graph the end of each list item is drawn using HTML5 Canvas, which results in a better performance than plain HTML, SVG and WebGL, as it generates stateless bitmaps.

Streams: Output for the command-line version was implemented using file streams: For every calculation that emits events, a writable stream is created in Node.js. The current implementation pipes the output to multiple files on the hard disk, each for every type: Player running analysis (1 stream), aggregate running statistics (4 time frames), player ball possession (1 stream), team ball possession (1 stream), heatmaps (4 time frames) and shot on goals (1 stream). This results into a total of 12 files written to disk.

5. CONCLUSION AND FUTURE WORK

In the first evaluation phase it turned out that modern JavaScript engines such as V8 are well suited to process large amount of data in real-time. With the hybrid model it is possible to analyze a full soccer match on the command line and visualize it in the browser. HTML5 features such as Canvas and WebGL draw graphics without any noticeable performance degradation and the event-driven, non-blocking I/O model of Node.js is an efficient way to read, process and write data.

As mentioned above, the current implementation outputs all streams to the file system, but using the abstract stream pattern in Node.js, they can be piped to other types of streams such as WebSockets. A possible use case are mobile devices with limited storage and computing power. They connect to the main server via a socket connection and receive only small chunks of data that are necessary to render relevant information. Coaches and team members could benefit from such a solution for mobile devices, because they are small and portable.

This available type of system can also be ported to other ball team sports, *e.g.*, American football, rugby or basket-

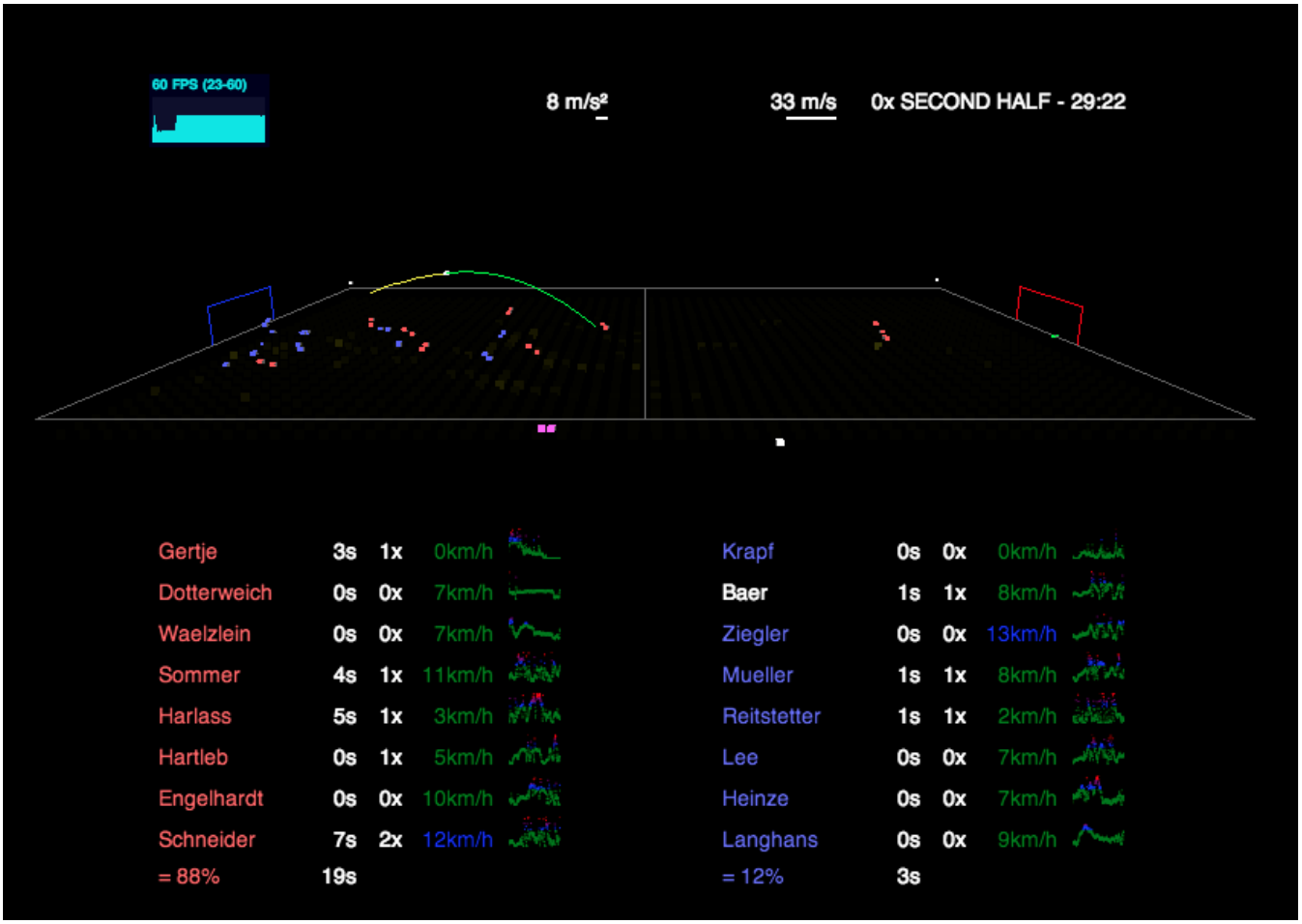

Figure 1: Screenshot

ball. Further work might include more automatic analyses, such as number of corner shots, passes and duel statistics. If combined with additional biometric sensor data (pulse, power) it will give insights to the current player's condition or even his progress during a whole season.

Acknowledgments

Ubilabs? Thomas Steiner, Klaus Trainer, Michael Pletziger, Jens Wille, Samuel Oey, Robert Katzki

6. REFERENCES