Dynamic Split-Screen for Visualizing Events in Augmented Reality

Luca Becheanu
Responsible Professor: Elmar Eisemann
Supervisor: Mark Winter
EEMCS, Delft University of Technology, The Netherlands

June 19, 2022

A Dissertation Submitted to EEMCS faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering

Abstract

This paper introduces the concept of the Voronoi Split-Screen in Augmented Reality inside a Sailing Regatta visualization application. We are making use of existing methods in 2D environments and modifying them to treat the implications of merging the screen where a user has complete camera control (3D/AR/VR).

This is done in three phases which take into account the 3D coordinates of the cameras, while considering the distance between objects, and mapping them from world space coordinates to screen space. Another important aspect analyzed is the orientation of the main camera compared to where the event is taking place.

Furthermore, the algorithm will also give the user guidance on reaching key events by pointing towards them using an arrow, as well as possibilities of choosing the amount of screen size they would like to have when an event is happening by using a slider. The proposed method provides a good starting point for tackling the problem of multiple key events happening at the same time, but it requires large enough displays such that the cells can be properly visualized.

Keywords: Dynamic Split-Screen, Voronoi Diagrams, Event Visualization, Augmented Reality

1 Introduction

1.1 Background

Sports have a long tradition of providing spectators with exciting moments throughout their competitions. With the current technological advances, these events can be visualized in multiple environments, such as 2D/3D on a computer, Augmented Reality using a mobile device, or even Virtual Reality by wearing a headset.

Once these events happen, a user should not feel constrained or compelled to watch them, if he chooses to view other parts of the race. Therefore, this paper showcases a method that emphasizes user interaction and helps them to smoothly engage with such events, without losing their full control of the camera.

Envisioning race events clearly can be difficult, even more in AR/VR environments where the camera cannot be completely controlled programmatically and the user is free to roam with all degrees of freedom, as discussed by Christie et al. (2008). This creates ergonomic constraints on the distance the camera must be pointed in the direction of the real-world scene to be augmented, called the "magic lens" metaphor (Kurkovsky et al., 2012, p.71).

Moreover, the field of view of the user is limited to the size and resolution of the smartphone's display screen, and the ability to use the mobile device while walking might have a negative impact on image visualization and clarity. Drascic and Milgram (1996) discuss the same problem in a virtual reality setting, where a narrow field of view makes the user wearing a head-mounted display unable to see important parts of the world and have incomplete and inaccurate depth perception.

The Computer Graphics and Visualization Group at TU Delft, in collaboration with the Sailing Innovation Center, is developing a new Augmented Reality/Virtual Reality (AR/VR) application (Figure 1a) for experiencing sailing regattas, which are sporting events consisting of a series of boat or yacht races. This application is meant to engage and inform those new to the sport of sailing, as well as those who already know the sport well. A captivating way is therefore needed for visualizing the competitors and key moments of an event (Figure 1b).

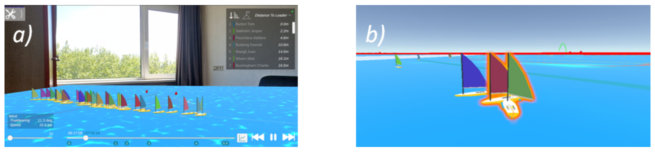


Figure 1: (a) Main Camera View in AR that follows the user's phone movement. (b) Event Camera View that displays a collision. The proposed method will discuss showing both images in an equal proportion when the user is not close enough to visualize the event. Original images from the Sailing+ AR Application.

1.2 The Voronoi Approach

This research aims to answer the question "How can a virtual camera system engagingly exhibit interesting race events, in an interactive AR/VR setting?".

In order to approach this, one may consider delving deeper into how can multiple mandatory/primary points of focus be handled. Taking into account the many issues related to AR/VR settings described in the previous section, the user camera is hard to be tweaked and modified. However, making use of additional cameras that can be automated will let the user keep full control of movement whilst still having a way to visualize key events.

Having a split-screen is, therefore, a possible solution and the method is dividing the screen horizontally or vertically, such that both points of interest will be on the screen at the same time. This has the drawback that when the primary points approach each other, both screen subdivisions would display the same environment twice, effectively wasting half of the valuable screen.

Previous work optimizes the screen space using the well-known Voronoi diagrams (Senechal et al., 1995) to implement a "dynamic split-screen". A line as a separator between points changes its orientation dynamically according to the position of the points while they move (Figure 2a,b). This line vanishes when the points get too close to each other (Figure 2c).

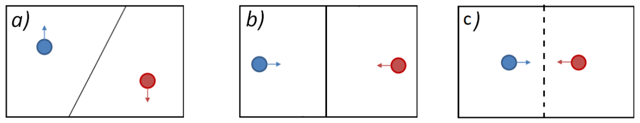


Figure 2: (a) The screen is split to show both points (red and blue) when they are far away from one another. (b) The separator line changes its orientation dynamically based on the points' positions in world space while they move (arrows). (c) The separator line vanishes once the points move towards each other and are close by.

In that sense, the objective of this research is to provide a solution for implementing the Voronoi split-screen in the context of the Sailing+ application from a 2D setting to 3D/AR. Since most dynamic split-screens use 2D points in screen space to map pixels, merging in terms of 3D camera rotation and translations will have some implications that will be discussed further. The solution will focus on handling two cameras on the same screen at the same time, but there is a possibility of extending to multiple cameras if the need arises.

The upcoming sections are structured in the following way: Section 2 provides the methodology used to tackle the problem and Section 3 presents the main results of the implementation in the context of a video demonstration. Following this, Section 4 discusses the performance and the drawbacks that the method has when multiple key events are happening at the same time, with open issues for improvement in future research. Finally, Section 5 draws a conclusion with an overview of the extent the research question has been answered.

2 Methodology

This section contains a description of the research techniques used to answer the research question and details the algorithm and its three phases. Section 2.1 introduces the notion of Voronoi diagrams with regards to the Sailing+ application and details the split phase where the screen is split into two equal parts, Section 2.2 describes the transition phase between the split-screen and merge, where the user receives more screen space when it approaches an event. Finally, Section 2.3 discusses how the user camera takes over the entire screen in order for the event to be engagingly visualized, and how we are solving the orientation problems found in an environment with more than two dimensions.

The Euclidean distance between the 3D world space coordinates of the user and event camera objects defined in Equation 1 determines the split-screen phases. In this equation, (m_1, m_2, m_3) are the 3D (x, y, z) coordinates of the main user camera position and (e_1, e_2, e_3) the 3D (x, y, z) coordinates of the event camera position. The table in Appendix B displays the mathematical notation used in this paper.

$$d(\mathbf{m, e}) = \sqrt{(m_1 - e_1)^2 + (m_2 - e_2)^2 + (m_3 - e_3)^2} \quad (1)$$

The computed Euclidean distance is bound in the $[0, \infty)$ range. We define an interval $[min, max]$ in Equation 2 which will delimit the transition phase. If the distance $d(\mathbf{m, e})$ determined in Equation 1 is higher than max, the split phase occurs. Finally, if the distance is lower than min, the merge phase takes place. The two edges (min, max) that establish the three phases can be modified in the algorithm.

$$\text{phase}(d(\mathbf{m, e})) = \begin{cases} \text{merge} & d(\mathbf{m, e}) < min \\ \text{transition} & min \leq d(\mathbf{m, e}) < max \\ \text{split} & d(\mathbf{m, e}) \geq max \end{cases} \quad (2)$$

2.1 The Split Phase

The method applies the concept of a Voronoi diagram by generating it using Graphics Hardware for fast computations as seen in Hoff et al. (1999). The Voronoi diagram (Figure 3) is a partition of space in regions called Voronoi cells. These cells consist of finitely many points of the Euclidean plane closer to that cell center than any other.

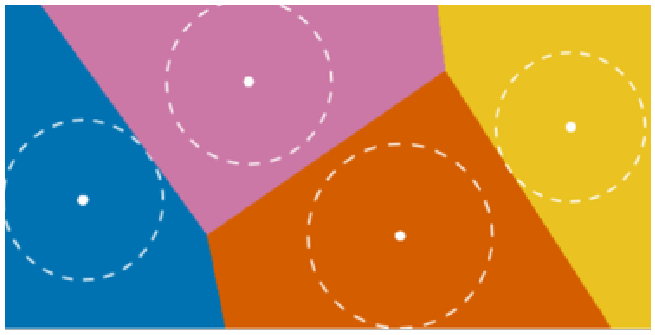


Figure 3: Voronoi diagram with multiple Voronoi cells, which are regions obtained from the intersection of half-spaces. Each has a different color and consists of every point in the Euclidean plane whose distance is less than or equal to its distance to any other cell center. Image by Eiserloh (2016).

Our focus is on two such cells, with their respective cell centers representing the user (Figure 1a) and event (Figure 1b) camera positions. The points that need to be closer to the cell centers in our case are on-screen pixels. In order to display both cells simultaneously (Figure 4), we are adjusting the $O(n \log n)$ Euclidean Space geometric sweepline algorithm for Voronoi diagrams as described by Fortune (1987).

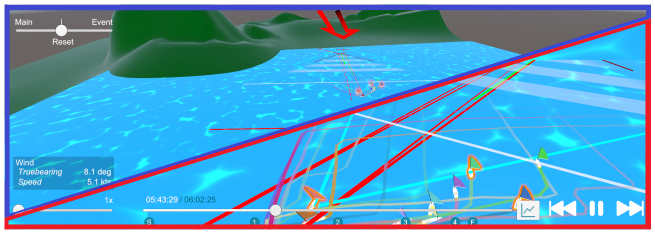


Figure 4: A dynamic separator line (middle) changes its orientation based on the position of the user's camera (highlighted with blue, left), which is far away from the race, relative to the event camera (highlighted with red, right) in order to have an equal screen distribution. Original image from the Sailing+ 3D Application.

Since every point is independent of another, a part of the algorithm is implemented inside the fragment shader, the stage that will process a Fragment generated by the rasterization into a set of colors (Akenine-Möller et al., 2018, Chapter 3.6).

Normalizing camera positions

In order to work with pixel coordinates, we need to compute the normalized camera positions from the 3D world space coordinates to screen space (Figure 5), according to the aspect ratio, which is being done inside a script called per frame that is attached to the user's camera game object.

The normalization is done by converting the $\mathbf{world_{xy}}$ camera coordinates which are in a seemingly infinite range into a $[0, 1]$ interval.

$$\mathbf{diff} = \begin{cases} \begin{pmatrix} (\mathbf{Max_y} - \mathbf{min_y}) \cdot r - (\mathbf{Max_x} - \mathbf{min_x}) \\ 0 \end{pmatrix} & \mathbf{Max_x} < \mathbf{Max_y} \cdot r \\ \begin{pmatrix} 0 \\ (\mathbf{Max_x} - \mathbf{min_x})/r - (\mathbf{Max_y} - \mathbf{min_y}) \end{pmatrix} & \mathbf{Max_x} > \mathbf{Max_y} \cdot r \end{cases} \quad (3)$$

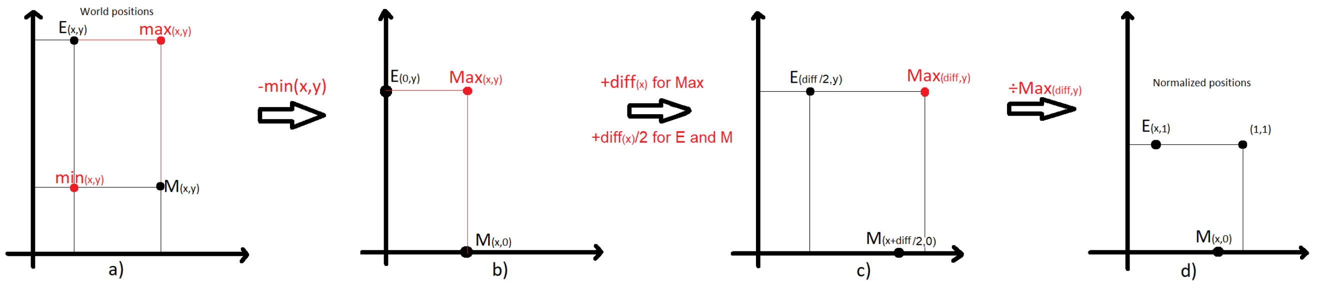


Figure 5: Normalization of camera positions **M** (Main camera) and **E** (Event camera) from world space to screen space. (a) The maximum and minimum values of x and y ($\mathbf{max}, \mathbf{min}$) between the $\mathbf{world}_{\boldsymbol{xy}}$ camera coordinates are computed. (b) Subtracting **min** from all points, gives us the upper right corner of the viewport (**Max**). (c) The $\mathbf{diff}_{\boldsymbol{x}}$ (in case of **Max** or $\mathbf{diff}_{\boldsymbol{x}}/2$ for **E** and **M**) from the first case of Equation 3 is added to the result from (b) in order to correct the camera positions for the screen aspect ratio. (d) Dividing all points to the **Max** computed at point (c) yields the normalized coordinates.

This is done by finding the maximum and minimum values of x and y ($\mathbf{max}, \mathbf{min}$) between the $\mathbf{world}_{\boldsymbol{xy}}$ camera coordinates (Figure 5a).

Then, we compute the upper right corner of the viewport (**Max**) by subtracting **min** from **max** and we do the same for both cameras coordinates (Figure 5b).

The 2D Vector **diff** corrects all the points for the screen aspect ratio r as seen in Equation 3, where r it the screen width divided by height (Figure 5c).

Half of this **diff** is added to the camera positions, whilst the entire **diff** is added to **Max**. Finally, the normalization of camera coordinates will be computed by dividing them to the new **Max** (Figure 5d).

The steps shown in Figure 5 are included in Equation 4, where **diff** is defined in Equation 3. The **normalized** result of the camera positions 2D coordinates will be in our case always on the edges of the screen opposite to one another (Figure 6).

$$\mathbf{normalized} = \frac{\mathbf{world}_{\boldsymbol{xy}} - \mathbf{min} + \frac{\mathbf{diff}}{2}}{\mathbf{Max} + \mathbf{diff}} \quad (4)$$

We are using the Euclidean distance defined in Equation 5, to compute which pixel is closest to what camera, where (p_1, p_2) are the 2D (x, y) coordinates of the pixel being rendered and (c_1, c_2) the 2D (x, y) coordinates of each camera.

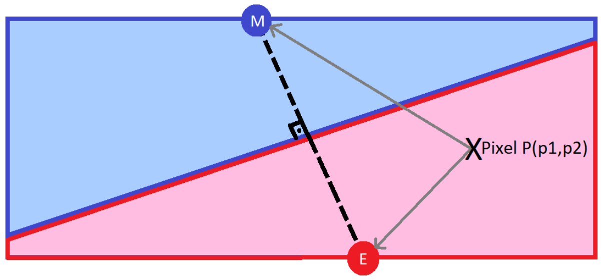


Figure 6: Pixel **P** with coordinates (p_1, p_2) belongs to the Voronoi cell with center in **E** (Event camera) because it is at a smaller Euclidean distance than the Voronoi cell with center in **M** (Main camera). The pixels that are on the blue/red line between the two cells are at the same distance between the center of these cells and form the separator line.

$$d(\mathbf{p}, \mathbf{c}) = \sqrt{(c_1 - p_1)^2 + (c_2 - p_2)^2} \quad (5)$$

Camera layering and separator line

In order to mimic the cell generation, instead of computing cells from one side of the screen to the other as done by Fortune (1987), the event cameras are layered to render before the main camera each time an event is happening. This means that the main camera will be a layer below the event camera. To create the Voronoi cells, if a pixel is closer to the event camera (Figure 6), it will be rendered on screen. All pixels that are closer to the main user's camera defined by the 2D Euclidean distance in Equation 5 will be discarded from the event camera, thus showing the user camera rendered below it in the other half of the screen (Figure 4).

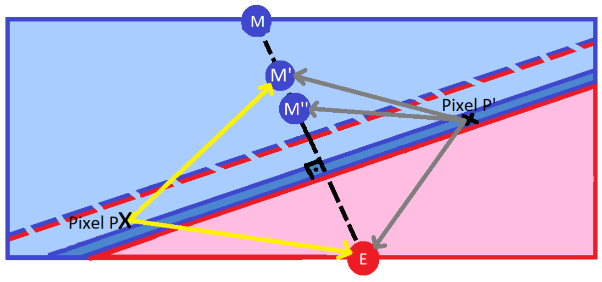


Figure 7: The cell center **M** (Main camera) is translated in the point **M'** (new cell center of the Main camera for the transition phase) and **M''** (used for computing the line) respectively, in the direction of the event camera along the line that ties both cameras. Pixel **P** (bottom left) is closer to **M'**, rather than **E** (Event camera), therefore it belongs to the Main cell. Pixel $\mathbf{P}'$ (middle right) is closer to **E** than **M'**, therefore it belongs to the Event cell. Because it is closer to **M''** than **E**, it is colored in dark blue to form the separator line between the **M** (or **M'** in transition) and **E** cells.

The pixels that are at an equal distance to both of the cameras will be colored to represent the separator line between them. Since a pixel is not enough to create the separator line, we will translate the center of the cell from the main camera towards the event camera position along the line which ties both cameras, on a distance equal to the thickness of the line we would like to display in that frame (**M”** in Figure 7, computed in Equation 6).

$$\mathbf{M''} = \mathbf{m} + lt \cdot \frac{\mathbf{e} - \mathbf{m}}{d(\mathbf{m}, \mathbf{e})} \tag{6}$$

In this equation, $(\mathbf{m}, \mathbf{e})$ represent the 2D positions of the main and event cameras, lt represents the separator line thickness which can be set from inside the algorithm, and $d(\mathbf{m}, \mathbf{e})$ is the 2D Euclidean distance between cameras.

This way, the pixels that belong to the event, but are closer to the new center of the main cell will be colored as a separator line between cells.

Centering the event camera inside the cell

The centers of the Voronoi cells follow the changes in the positions of the cameras in the 2D screen space such that the center of the camera is always translated into the center of its respective cell. This way the split will always change based on where a camera is in the plane relative to the other. Thus the user knows if his camera is on the top/bottom/left/right side of the event camera.

In order to always show the event in the middle of its cell, we compute the pixel deviation from the center of the event camera along the line connecting the main camera to the event camera (Figure 7) at a distance of 75% from the main camera as seen in Equation 7, where $(\mathbf{m}, \mathbf{e})$ are the main and event 2D camera positions in screen space.

$$\mathbf{pixel\_deviation} = \mathbf{m} + 0.75 \cdot (\mathbf{e} - \mathbf{m}) \tag{7}$$

We will apply this deviation to each pixel that belongs to the event camera cell. We add or subtract this deviation to the pixel position depending on the location of the pixel in screen space.

Guiding the user towards the event location

During this phase, an arrow at the top of the main user's camera screen points towards the position of the event camera, to guide the user's way to the event (Figure 8, highlighted with purple). This is achieved by creating a 3D arrow object that looks into the direction of the event location and based on how the split is oriented, it will shift its position on the edge of the main camera view to either be on top/bottom/left/right, following the main camera position.

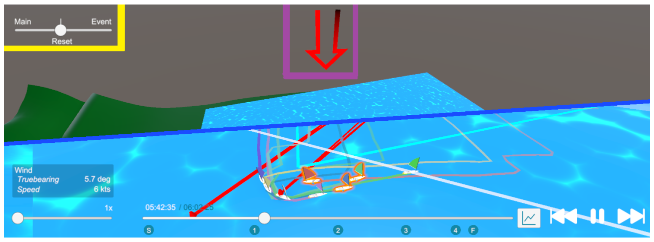


Figure 8: An arrow (highlighted with purple, top middle) is placed on the main camera view and points towards the event which in this case is at the bottom. The slider (highlighted with yellow, top left) is used to increase or decrease the camera views image, depending on the direction of the camera they would rather see (main to the left or event to the right). The reset button in the middle will revert the slider to its original position. Original image from the Sailing+ 3D Application.

Another guidance feature is a slider located at the top left of the screen (Figure 8, highlighted with yellow) which can be moved in the direction of the camera that the user would rather see. Sliding all the way to the right will result in only the event being displayed and sliding to the left will increase the main camera view. The reset button in the middle will put the slider back to its original position with both cameras having the same screen size.

When we increase the main cell, the center of the cell is translated in the direction of the event camera along the line that ties both cameras (Figure 7), or the opposite direction along the same line when the user's shifts the slider to increase the event view. This way, we increase the number of pixels that belong to one cell compared to the other. This principle of a 'weighted' Voronoi cell will be further explained in the next section.

2.2 The Transition Phase

The transition is the intermediate phase between the split phase and the merge phase and consists of raising the screen space allocated to the main user's camera detrimental to the event camera (Figure 7) using the Weighted Generalization of the Voronoi Diagram described by Ash and Bolker (1986).

In order to achieve this, we use an interval between the lower and upper bound of the Voronoi Weight (Equation 2), which can be modified inside the algorithm and is based on the world space distances between cameras. The merge phase happens when the distance is below the lower bound of the interval, whilst the split phase commences if the distance between cameras is higher than the upper bound.

If the Euclidean distance between cameras in 3D world space coordinates (Equation 1) is inside the range of the interval (Equation 2), we compute a smooth Hermite interpolation between 0 and 1 as seen in Equation 8 (Tatarchuk, 2003, p.94) which is transposed in screen space. In this equation, x represents the 3D Euclidean distance between cameras, while min and max are the lower and upper bounds of the Weight interval.

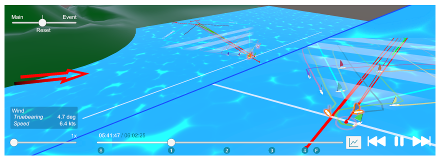
$$\text{smoothstep}(x) = \begin{cases} 0 & x < min \\ 3x^2 - 2x^3 & min \leq x < max \\ 1 & x \geq max \end{cases} \tag{8}$$

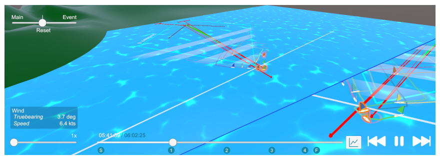
The result is used in Equation 9 to compute the translated position of the main camera found on the line between the main and event camera (Figure 7, **M’**). In this equation, $(\mathbf{m}, \mathbf{e})$ represent the 2D positions of the main and event cameras, $\text{smoothstep}(dist)$ is the interpolation defined in Equation 8, where $dist$ is the 3D Euclidean distance between the main and event cameras.

$$\mathbf{M'} = \mathbf{m} + (1 - \text{smoothstep}(dist)) \cdot \frac{\mathbf{e} - \mathbf{m}}{d(\mathbf{m}, \mathbf{e})} \tag{9}$$

The translation will be closer to the event camera, to be able to attract like a magnet more pixels to the user camera. In this sense, the weighted cell of the main camera is increased as seen in Figure 9.

The separator line between cells (Figure 9b) will shorten depending on the result of the interpolation from Equation 8.


(a)


(b)

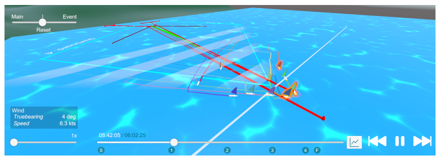

(c)

Figure 9: Magnetism effect of the main camera, which attracts more pixels to it as the distance between itself and the event camera decreases. (a) Main camera view (left) increases as the distance between itself and the event camera (right) decreases. (b) The separator line as well as the event (bottom left) shrink smoothly as the main camera takes over. (c) Main camera view is visualizing the event on the entire screen space. Original images from the Sailing+ 3D Application.

We achieve this by translating the main camera from **M'** in point **M''** towards the event camera as seen in Figure 7, by using Equation 10. In this equation, smoothstep$(dist)$ is the interpolation defined in Equation 8, where $dist$ is the 3D Euclidean distance between the main and event cameras. Finally, lt represents the separator line thickness which can be set from inside the algorithm, and $d(\mathbf{m}, \mathbf{e})$ is the 2D Euclidean distance between cameras.

$$\mathbf{M''} = \mathbf{M'} + \text{smoothstep}(dist) \cdot lt \cdot \frac{\mathbf{e} - \mathbf{m}}{d(\mathbf{m}, \mathbf{e})} \tag{10}$$

The center of the event camera is translated to keep the event centered no matter how small the Voronoi cell becomes, as described in Section 2.1, Equation 7.

In this phase, the arrow will disappear as the event is close enough to be engagingly visualized by the user, but the slider will remain on the top left as in Figure 8, to let the user have full control of what he would rather see.

2.3 The Merge Phase

The merge phase happens when the 3D Euclidean distance is smaller than the min value defined in Equation 2. However, despite being in close range to the event camera, the user can still choose to look in a different direction than where the event is taking place. Due to this fact, we need to establish a new interval (Equation 11) that takes into account multiple key points:

1. The 3D Euclidean distance between the point where the forward vector of the main camera intersects the regatta field and the competitor that the event camera is pointing towards (Figure 10, dark blue and red lines).
2. The world space angle formed between two 3D vectors, the main camera to the center of the viewport and the main camera to the point where the event competitor in the regatta field is (Figure 10, angle α).
3. The 2D normalized Euclidean distance between the main camera center of the viewport and the event competitor (Figure 10, purple line).

If the 3D Euclidean distance from the first point (Figure 10, between **H** and **E**) is greater than a maximum visible distance, and the world space angle (second point) is greater than a maximum visible angle (both of which are defined in the algorithm), then the M_split phase occurs.

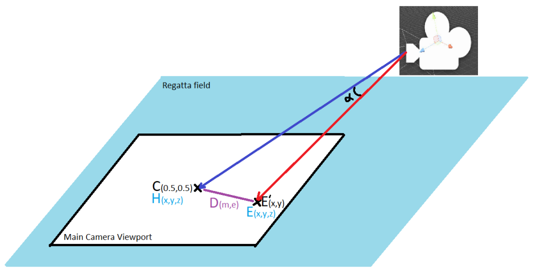


Figure 10: Main camera pointing at the regatta field (light blue area). **H** represents the point with world space coordinates where the forward vector (dark blue) going from the camera intersects the regatta field. **E** is the point with the world space coordinates of the competitor participating in an event. **C** is the center of the viewport (white) with coordinates $(0.5, 0.5)$, and **E'** is the point with the coordinates of the event competitor.

The computed 2D Euclidean distance between cameras $D(\mathbf{m}, \mathbf{e})$ described at the third point is bound in the $[0, \infty)$ range. The interval $[Min, Max]$ in Equation 11 will delimit the M_transit phase. If the distance is higher than Max, the M_split phase occurs. Finally, if the distance is lower than Min, the M_merge phase takes place, where the user's camera will take over the entire screen. The two edges (Min, Max) that establish the three phases can be modified in the algorithm.

$$\text{Phase}(D(\mathbf{m}, \mathbf{e})) = \begin{cases} \text{M\_merge} & D(\mathbf{m}, \mathbf{e}) < Min \\ \text{M\_transit} & Min \leq D(\mathbf{m}, \mathbf{e}) < Max \\ \text{M\_split} & D(\mathbf{m}, \mathbf{e}) \geq Max \end{cases} \tag{11}$$

Handling the transit and split phases

In the M_transit phase, we compute the smoothstep(x) from Equation 8. In this equation, x is the $D(\mathbf{m}, \mathbf{e})$ distance, while min and max are the lower and upper bounds of the interval (Min, Max).

$$[\mathbf{M}, \mathbf{E}] = \begin{cases} [\mathbf{M}_{(0,1-\mathbf{E}_y)}, \mathbf{E}_{(1,\mathbf{E}_y)}] & \mathbf{E}_y < \mathbf{E}_x \ and \ \mathbf{E}_y \geq 1 - \mathbf{E}_x \\ [\mathbf{M}_{(1,1-\mathbf{E}_y)}, \mathbf{E}_{(0,\mathbf{E}_y)}] & \mathbf{E}_y < 1 - \mathbf{E}_x \ and \ \mathbf{E}_y \geq \mathbf{E}_x \\ [\mathbf{M}_{(1-\mathbf{E}_x,1)}, \mathbf{E}_{(\mathbf{E}_x,0)}] & \mathbf{E}_y < 1 - \mathbf{E}_x \ and \ \mathbf{E}_y < \mathbf{E}_x \\ [\mathbf{M}_{(1-\mathbf{E}_x,0)}, \mathbf{E}_{(\mathbf{E}_x,1)}] & \mathbf{E}_y \geq \mathbf{E}_x \ and \ \mathbf{E}_y > 1 - \mathbf{E}_x \end{cases} \tag{12}$$

In both phases (M_transit and M_split), we simulate the movement of the camera positions along the edges of the screen opposite to one another as seen in Figure 6 using the 2D coordinates of the center of the viewport and the competitor's position (Figure 10, **C** and **E'**).

To achieve this simulation, we will translate both points on the edges of the screen by using Equation 12, depending on the event position of the competitor **E** relative to the center of the viewport **M**. The result of this translation can be seen in Figure 11, where **M** and **E** will be the center of the Voronoi Cells for these two phases.

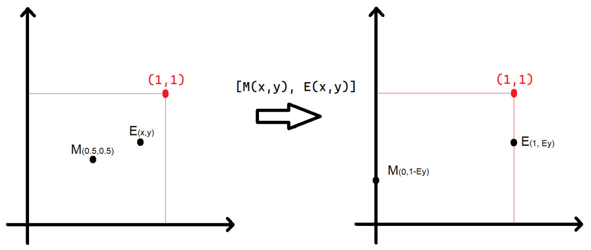


Figure 11: Translation of the center of the viewport **M** and the competior's position that participates in the event **E**. After applying Equation 12, we obtain a list of two 2D vectors representing the center of the Voronoi cells, which will always be on the edges of the screen at opposite ends of each other (**M** and **E** as seen previously in Figure 6).

If the competitor's position is outside of the main camera viewport $[0, 1]$, when the user changes its camera orientation, or if the event will continue outside of the viewport, then we will normalize this new position using the previous Equation 4.

In these phases, the behaviour of the Voronoi cells and the separator line between, as well as the event camera cell centering, and the mentioned guidance features are the same as described in Sections 2.1 and 2.2.

Once the 2D distance $D(\mathbf{m}, \mathbf{e})$ is below the transition interval, the user's camera will take over the entire screen (Figure 9c). As the user distances himself from the event, the algorithm will smoothly return to the transition and split phases consequently. If the event ends in this phase, the user will simply remain in complete control of the camera, and the split-screen is removed. At the beginning of the race, or if a new event happens, the screen will be in one of the three phases, depending on the position of the event compared to where the user camera is at that current moment in time.

3 Results

Events are an important part of the race and there should be a way to showcase them even though the user is not directly looking in their direction. That is where the split-screen algorithm comes into play. In this section, we showcase the results of the technique presented in the three phases.

The screen will display two Voronoi Cells, one for the main user camera and one for the event. Depending on the position of the user's camera relative to the event, the split along with the separator line between will shift their direction to show where the event is happening (Figure 4).

If the user's camera is above the event camera, the event will be displayed at the bottom of the screen, if the user's camera is on the left of the event camera, the event will be displayed on the right side of the screen. This will happen similarly in the other two directions.

During this time an arrow is pointing towards the location of where the event is currently happening, and the user only needs to follow it to see everything from the point of view of the event camera. Another guidance feature, in case the user chooses not to move their camera at all, is moving the slider on the top left of the screen in the direction of the camera that they would rather see. If they choose the main camera, the event cell will slowly shrink until the entire main camera view takes over. If they would rather see the event, then the event camera cell will increase in size depending on the movement of the slider. The reset button in the middle will put the slider back at its original position (Figure 8).

In the transition phase that happens when the user approaches the event, the center of the Voronoi cell corresponding to the main camera is shifted closer to the center of the event cell similar to the split-phase, based on the main camera position relative to the event camera (Figure 9a). The event cell will always be centered in the remaining screen, even though the view starts shrinking (Figure 9b). The user will still be able to choose what he would rather like to see in this phase, by shifting the top left slider towards its desired camera.

Finally, we have reached the merge phase, where the user camera is in the range of the event camera, therefore the only active camera is now the one of the user's (Figure 9c). However, they can still choose to rotate their camera away from the event, even though the camera positions are close to each other.

In that case, the screen will start to shift inside a transition phase again, which will result in a full split with the arrow that guides the user in the direction of the event. This time, the split follows the direction of where the user is pointing his camera relative to the event position and he can also choose to shift the slider to choose the camera he would like to see most. He can also perform full rotations around the event and the split will work in the same way, by shifting depending on the location of the event to the center of the main camera viewport.

Thus, the solution solves the Augmented Reality issue of the user having complete control of the smartphone equipped with a camera on the opposite side of the display, and allows the user to engage with the event regardless of his camera movement.

Since the split-screen effect is highly dynamic, the results are best visualized inside a video, rather than in figures. It shows the user engaging with an event by moving through all three phases of the algorithm. The demo is presented in the 3D environment of the Sailing+ application but the camera movement mimics the Augmented Reality behavior of a user with a phone camera.

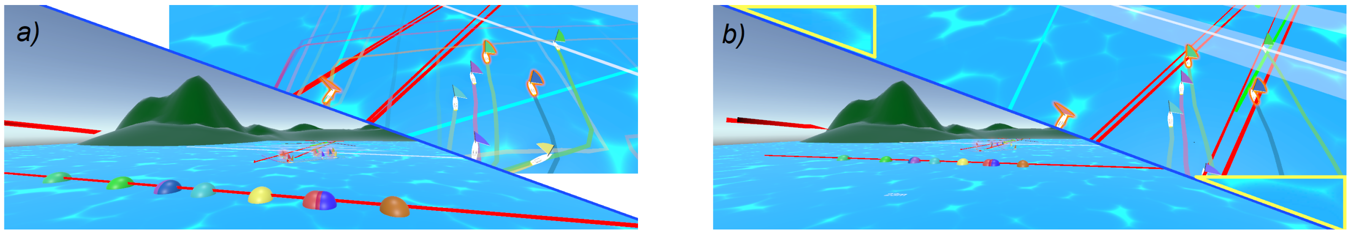


Figure 12: Center deviation of the event camera. (a) There is no information to render in the corners of the event Voronoi cell (white). (b) The empty edges are replaced with a texture similar to the water inside the regatta field (top left and bottom right, yellow).

4 Discussion

4.1 Performance

Our algorithm performs in $O(m \cdot n)$ time complexity where m is the number of active cameras (Voronoi cells) and n is the number of pixels. However, this is a negligible issue for two cameras in a fragment shader ($O(2 \cdot n)$) and it will not affect the framerate, making the algorithm fast enough for the required seconds in which the event is taking place.

The technique allows the customization of controlling the split-screen via several parameters. Firstly, the edges of the distance intervals from Equation 2 and 11 can be modified to better delimit when the event is still visible enough to be engagingly visualized and when the transition leading to the full split should begin and end. Moreover, the colors of the separator line and the arrow pointing toward the event as well as the line thickness can also be changed.

4.2 Shortcomings and Future Work

With the center displacement of the event cell and deviation of the pixels, there will be areas in the cell where there is no render information (Figure 12). In order to solve this problem, we render a texture similar to the one of the water where the event is taking place. If the pixel coordinates x, y (computed after the addition or subtraction of Equation 7) are not in the range $(0, 1)$, then the water texture will be rendered.

Another limitation of the algorithm happens when the Voronoi cell of the event camera shrinks in the transition phase. The line separator will shrink along with the event cell and the smooth computed in Equation 8 will reach a very low value. The line thickness is influenced by this smooth equation, therefore the line will disappear completely before the event reaches the end. The solution to this is to set up a minimum line thickness that can be modified in the algorithm. In this way, the line will remain visible even if the event Voronoi cell is at a small scale.

The solution presents a split for two cameras, but it can be extended for multiple events happening at the same time by adding more event camera cells. The difficulty is in creating a fair size balance when more points of focus come into play (Figure 3), and an even ratio solution for more than two points has been proven to not exist as discussed in Lenz (2018). This means that the method can be extended to multiple key events, in case competitors are scattered far away in the plane but there will be ”spatial adjacency” issues (Gold, 1991, p.66). It does not necessarily mean that it cannot be done, however, a larger screen is needed in order to visualize each camera properly, and given the fact that the application targets mostly mobile phones, few of them have a large enough display for more than two splits, as it will be hard for the user to understand what is happening in all the different camera views.

The method showcased in the figures and the accompanying video is inside a 3D environment, yet it applies to any setup, including mobile devices and AR environments. The same can be said for projecting the application on a television display. That will provide enough room for the cells to be understood independently and it could encompass at least three more additional splits, before the views become unintelligible. Moreover, the split is transitioning smoothly enough to not provoke motion sickness in a Virtual Reality setting, therefore another possible extension of the algorithm is to test it inside this setting and conduct a user study to see if the addition is feasible with fast head tilt movements. For environments that have the possibility of touching the screen, the slider on the top left of the viewport can be completely removed and replaced with user gestures, by sliding with a finger the camera that the users would like to maximize.

5 Conclusions

We presented a technique for adding a split-screen effect to the Sailing+ application in order for the user to engagingly visualize key events happening throughout a sailing regatta. The technique is based on concepts of Voronoi diagrams as well as the Weighted Voronoi Generalization, applied to a 3D/AR environment.

The common usage of the algorithm is inside 2D video games, to showcase cooperative players on the same screen. Our approach demonstrates that the algorithm can be applied in any context other than 2D, as seen in the previous section (3D/AR/VR).

The split-screen algorithm is highly customizable, as everything from the phase intervals to the line thickness and color can be changed for the most engaging results.

Acknowledgments

We would like to thank Dr. Elmar Eisemann and Dr. Mark Winter for their support during the project. We would also like to thank the academic communication skills lecturers and the responsible research coaches of the TU Delft. Thanks also go to Berend Baas for providing insights and advice to solving the problem in an efficient manner and to Dr. Ricardo Marroquim for the feedback given.

References

Tomas Akenine-Möller, Eric Haines, Naty Hoffman, Angelo Pesce, Michał Iwanicki, and Sébastien Hillaire. 2018. *Real-Time Rendering 4th Edition*. A K Peters/CRC Press, Boca Raton, FL, USA. 1200 pages.

Peter F. Ash and Ethan D. Bolker. 1986. Generalized Dirichlet tessellations. *Geometriae Dedicata* 20, 2 (April 1986), 209–243. https://doi.org/10.1007/bf00164401

Marc Christie, Patrick Olivier, and Jean-Marie Normand. 2008. Camera Control in Computer Graphics. *Computer Graphics Forum* 27, 8 (2008), 2197–2218. https://doi.org/10.1111/j.1467-8659.2008.01181.x

David Drascic and Paul Milgram. 1996. Perceptual issues in augmented reality. In *SPIE Proceedings*, Mark T. Bolas, Scott S. Fisher, and John O. Merritt (Eds.). SPIE. https://doi.org/10.1117/12.237425

Squirrel Eiserloh. 2016. Math for Game Programmers: Juicing Your Cameras with Math. In *Game Developers Conference*.

Steven Fortune. 1987. A sweepline algorithm for Voronoi diagrams. *Algorithmica* 2, 1-4 (1987), 153–174. https://doi.org/10.1007/bf01840357

Christopher M. Gold. 1991. Problems with handling spatial data — the voronoi approach. *CISM journal* 45, 1 (1991), 65–80. https://doi.org/10.1139/geomat-1991-0005

Kenneth E. Hoff, John Keyser, Ming Lin, Dinesh Manocha, and Tim Culver. 1999. Fast computation of generalized Voronoi diagrams using graphics hardware. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques - SIGGRAPH '99*. ACM Press. https://doi.org/10.1145/311535.311567

Stan Kurkovsky, Ranjana Koshy, Vivian Novak, and Peter Szul. 2012. Current issues in handheld augmented reality. In *2012 International Conference on Communications and Information Technology (ICCIT)*. IEEE. https://doi.org/10.1109/iccitechnol.2012.6285844

Tobias Lenz. 2018. Fair Voronoi Split-Screen for N-Player Games. https://conference.imp.fu-berlin.de/eurocg18/download/paper_34.pdf

Wendy E. Mackay. 1995. Ethics, lies and videotape.... In *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '95*. ACM Press. https://doi.org/10.1145/223904.223922

Marjorie Senechal, Atsuyuki Okabe, Barry Boots, and Kokichi Sugihara. 1995. Spatial Tessellations: Concepts and Applications of Voronoi Diagrams. *The College Mathematics Journal* 26, 1 (Jan. 1995), 79. https://doi.org/10.2307/2687299

Natalya Tatarchuk. 2003. Advanced Real-Time Shader Techniques. AMD.

A Responsible Research

Research involves ethical implications when it is being conducted. The intended purpose of our algorithm is to showcase videos from different cameras at the same time. This can be used unethically (Mackay, 1995) to distort the event maliciously and manipulate the result to provide misleading information.

Another concern for this type of research is the reproducibility of results. If the method is not clearly detailed, future research will be hard to be conducted by others. Since the source code is written inside a licensed application, readers have to request access by contacting one of the paper authors. The lines have been thoroughly documented to remove ambiguity as much as possible.

B Mathematical Notation

The following table summarizes the mathematical notation used in this paper.

Type	**Notation**	**Examples**
angle	lowercase Greek	$\alpha_i, \phi, \rho, \eta, \gamma_{242}, \theta$
scalar	lowercase italic	a, b, t, u_k, v, w_{ij}
vector or point	lowercase bold	$\mathbf{a}$,$\mathbf{u}$, $\mathbf{v}_s$ $\mathbf{h}(\rho)$, $\mathbf{h}_z$

The angles and the scalars are real numbers from $\mathbb{R}$.

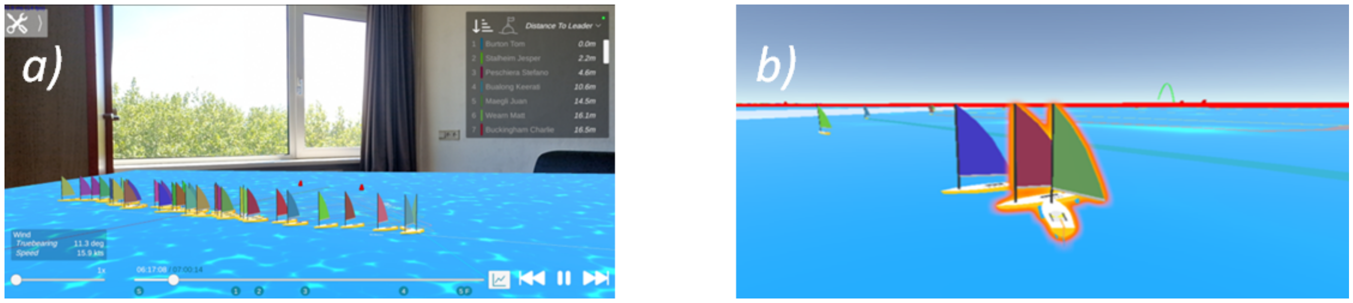
Vectors and points are denoted by bold lowercase letters, and the components are accessed in column vector format as

$$\mathbf{v} = \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix}$$

for a 3D vector.

C Enlarged Figures

The following section provides the figures used in the paper at a larger scale.



1	Burton Tom	0.0m
2	Stalheim Jesper	2.2m
3	Peschiera Stefano	4.6m
4	Bualong Keerati	10.6m
5	Maegli Juan	14.5m
6	Weam Matt	16.1m
7	Buckingham Charlie	16.5m

Figure 13: (a) Main Camera View in AR that follows the user's phone movement. (b) Event Camera View that displays a collision. The proposed method will discuss showing both images in an equal proportion when the user is not close enough to visualize the event. Original images from the Sailing+ AR Application.

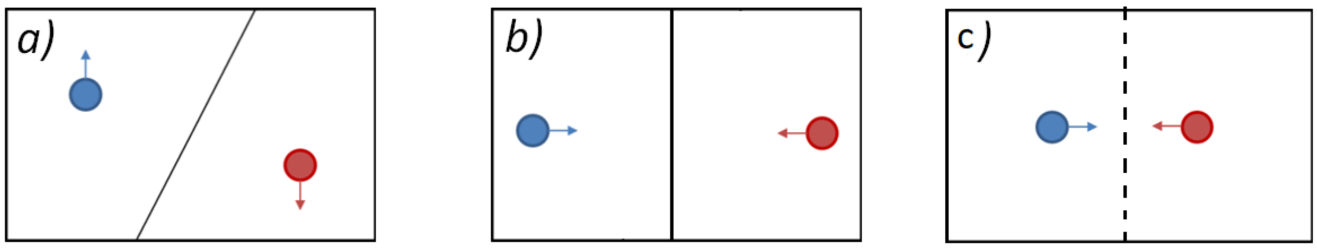


Figure 14: (a) The screen is split to show both points (red and blue) when they are far away from one another. (b) The separator line changes its orientation dynamically based on the points' positions in world space while they move (arrows). (c) The separator line vanishes once the points move towards each other and are close by.

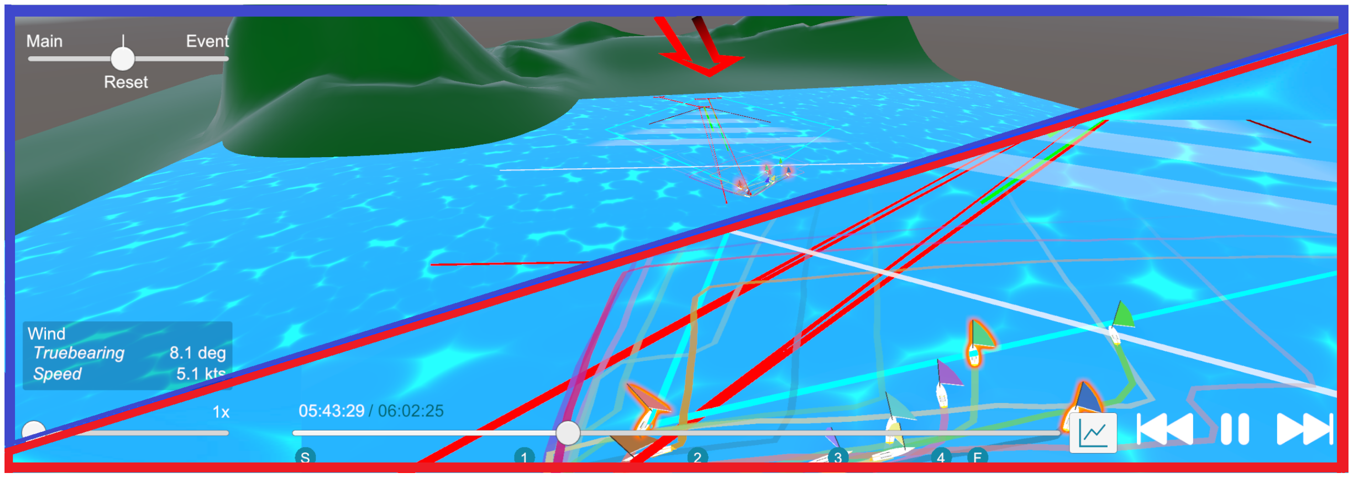


Figure 15: A dynamic separator line (middle) changes its orientation based on the position of the user's camera (highlighted with blue, left), which is far away from the race, relative to the event camera (highlighted with red, right) in order to have an equal screen distribution. Original image from the Sailing+ 3D Application.

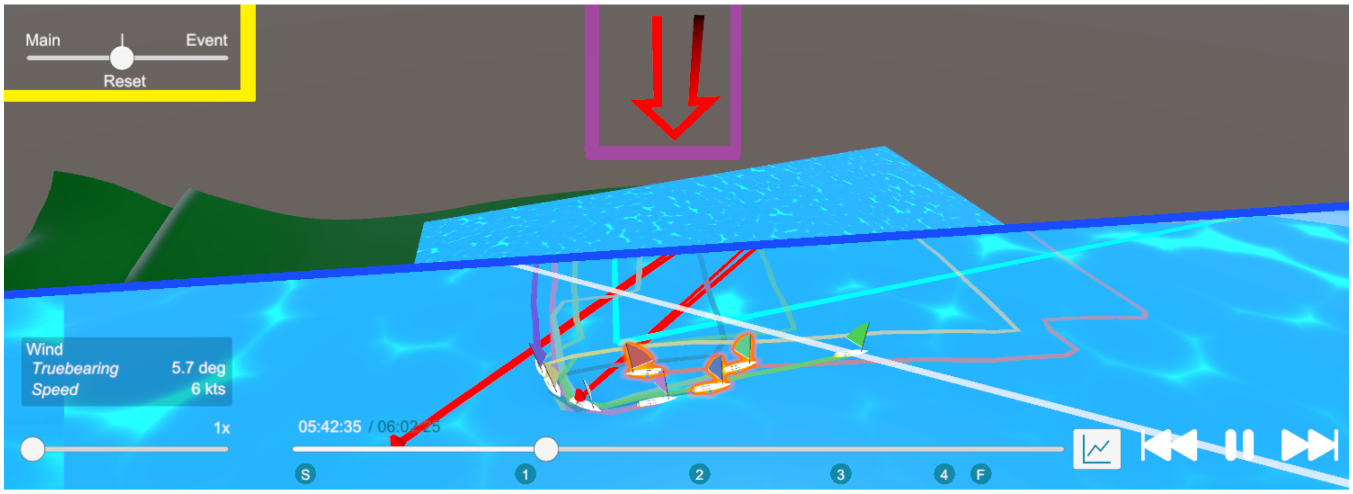


Figure 16: An arrow (highlighted with purple, top middle) is placed on the main camera view and points towards the event which in this case is at the bottom. The slider (highlighted with yellow, top left) is used to increase or decrease the camera views image, depending on the direction of the camera they would rather see (main to the left or event to the right). The reset button in the middle will revert the slider to its original position. Original image from the Sailing+ 3D Application.

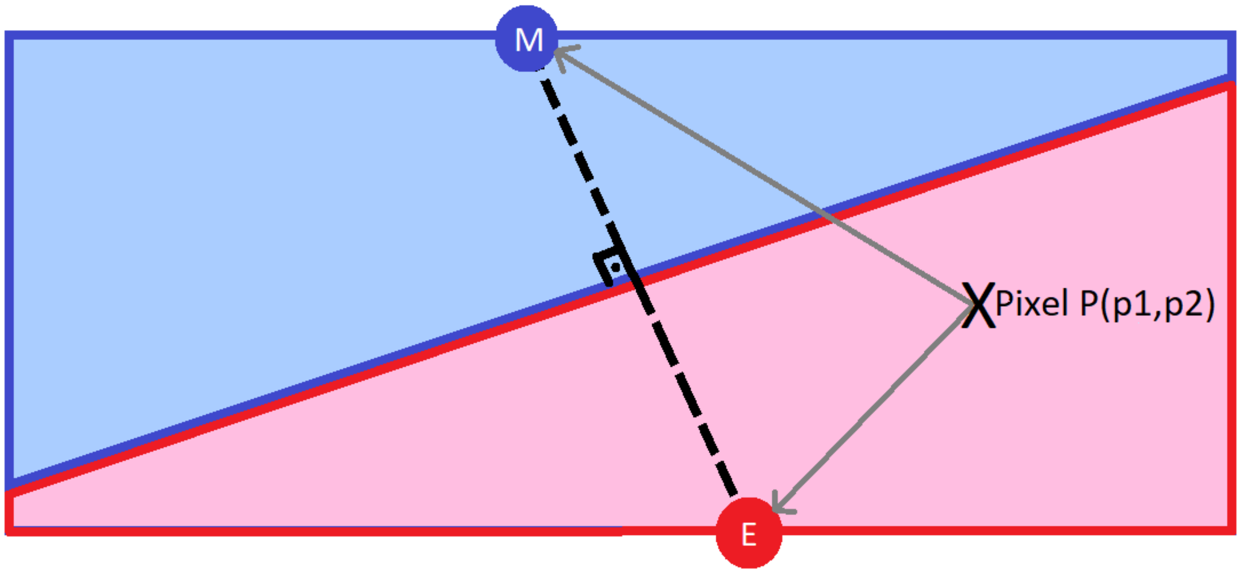


Figure 17: Pixel **P** with coordinates (p_1, p_2) belongs to the Voronoi cell with center in **E** (Event camera) because it is at a smaller Euclidean distance than the Voronoi cell with center in **M** (Main camera). The pixels that are on the blue/red line between the two cells are at the same distance between the center of these cells and form the separator line.

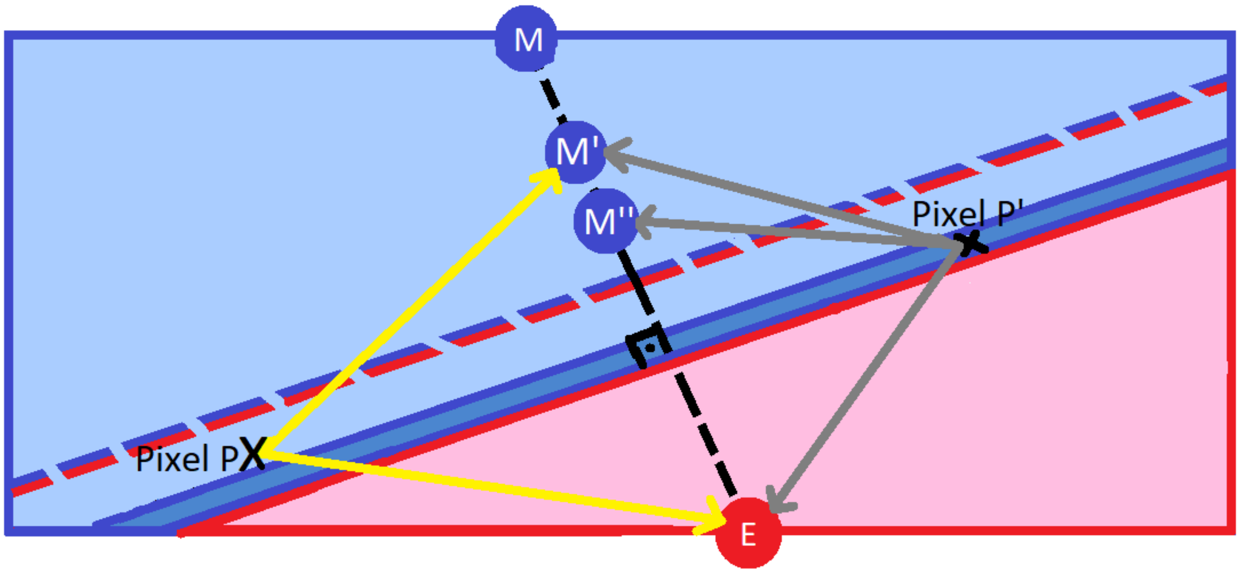


Figure 18: The cell center **M** (Main camera) is translated in the point **M'** (new cell center of the Main camera for the transition phase) and **M"** (used for computing the line) respectively, in the direction of the event camera along the line that ties both cameras. Pixel **P** (bottom left) is closer to **M'**, rather than **E** (Event camera), therefore it belongs to the Main cell. Pixel $\mathbf{P}'$ (middle right) is closer to **E** than **M'**, therefore it belongs to the Event cell. Because it is closer to **M"** than **E**, it is colored in dark blue to form the separator line between the **M** (or **M'** in transition) and **E** cells.

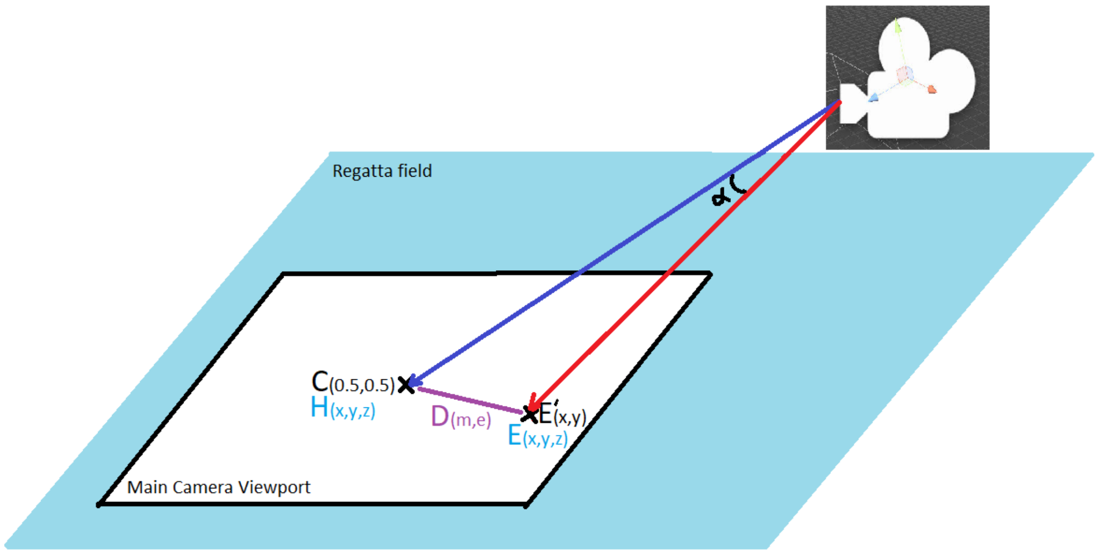


Figure 19: Main camera pointing at the regatta field (light blue area). **H** represents the point with world space coordinates where the forward vector (dark blue) going from the camera intersects the regatta field. **E** is the point with the world space coordinates of the competitor participating in an event. **C** is the center of the viewport (white) with coordinates $(0.5, 0.5)$, and **E'** is the point with the coordinates of the event competitor.

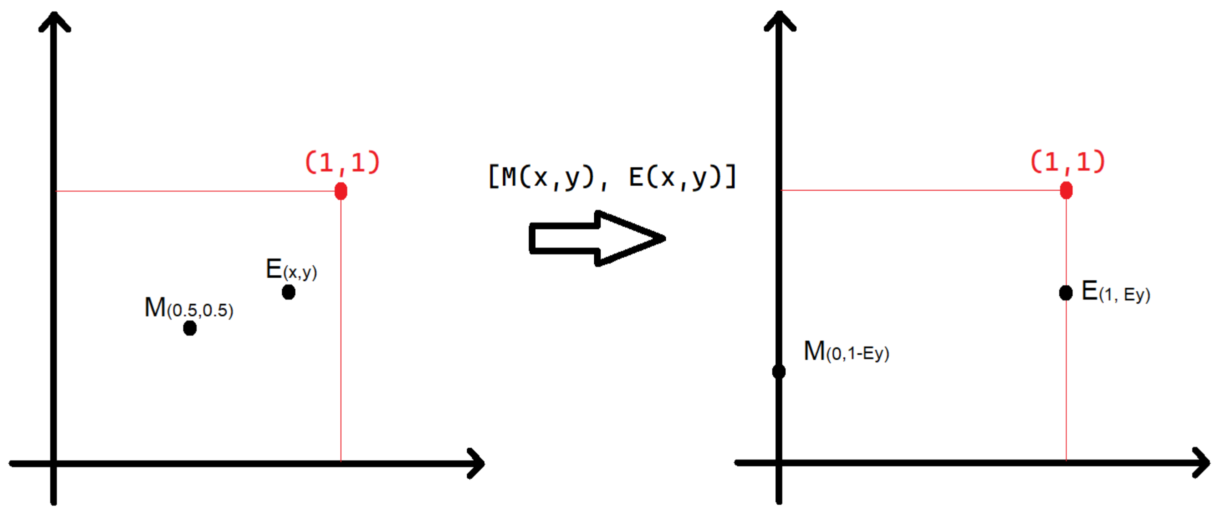


Figure 20: Translation of the center of the viewport **M** and the competior's position that participates in the event **E**. After applying Equation 12, we obtain a list of two 2D vectors representing the center of the Voronoi cells, which will always be on the edges of the screen at opposite ends of each other (**M** and **E** as seen previously in Figure 6).

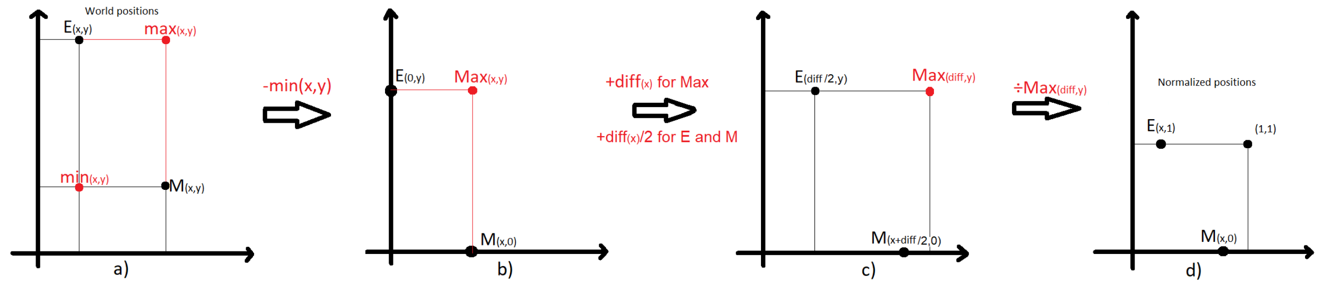


Figure 21: Normalization of camera positions **M** (Main camera) and **E** (Event camera) from world space to screen space. (a) The maximum and minimum values of x and y (**max**, **min**) between the $\mathbf{world}_{xy}$ camera coordinates are computed. (b) Subtracting **min** from all points, gives us the upper right corner of the viewport (**Max**). (c) The $\mathbf{diff}_x$ (in case of **Max** or $\mathbf{diff}_x/2$ for **E** and **M**) from the first case of Equation 3 is added to the result from (b) in order to correct the camera positions for the screen aspect ratio. (d) Dividing all points to the **Max** computed at point (c) yields the normalized coordinates.

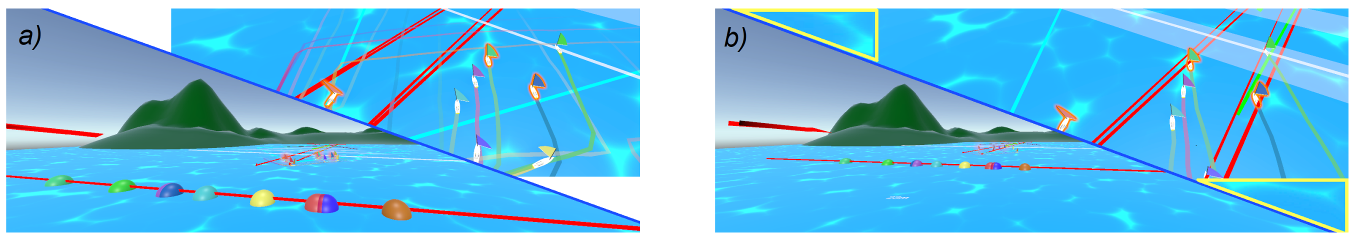
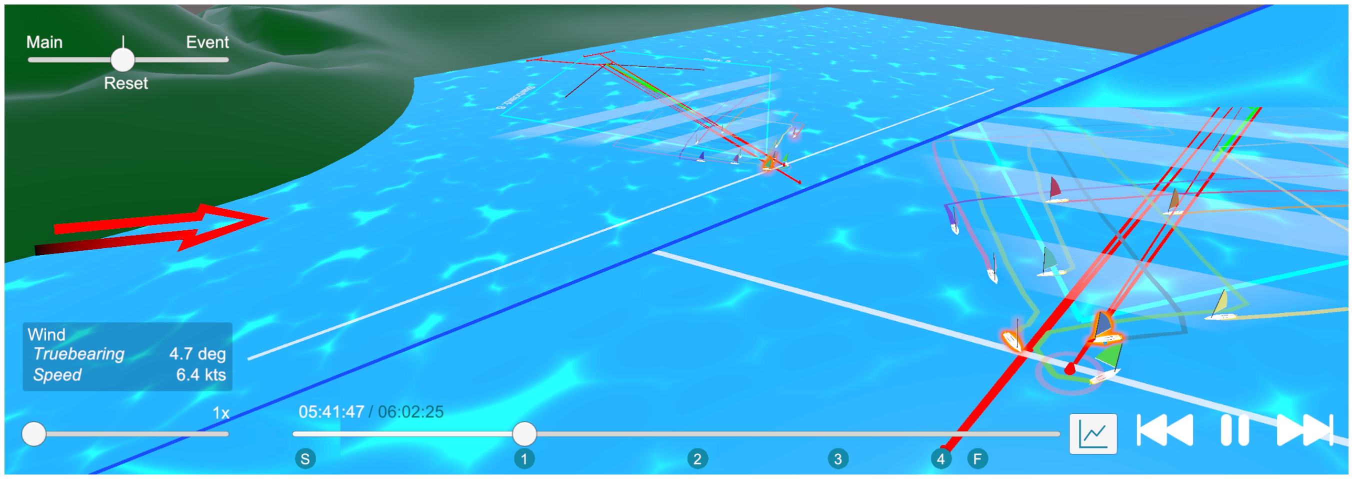
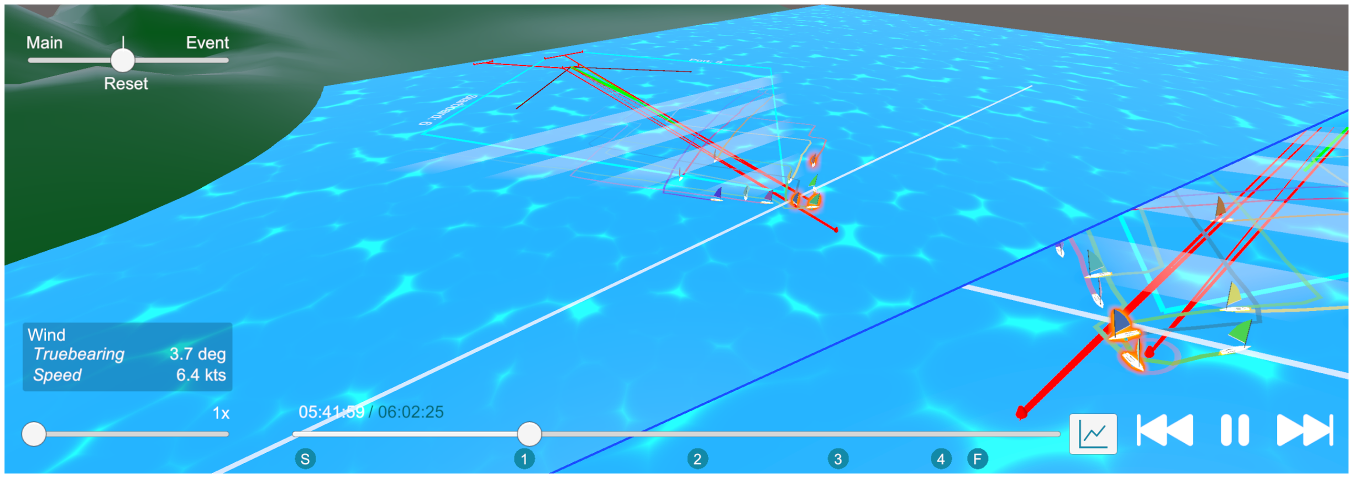


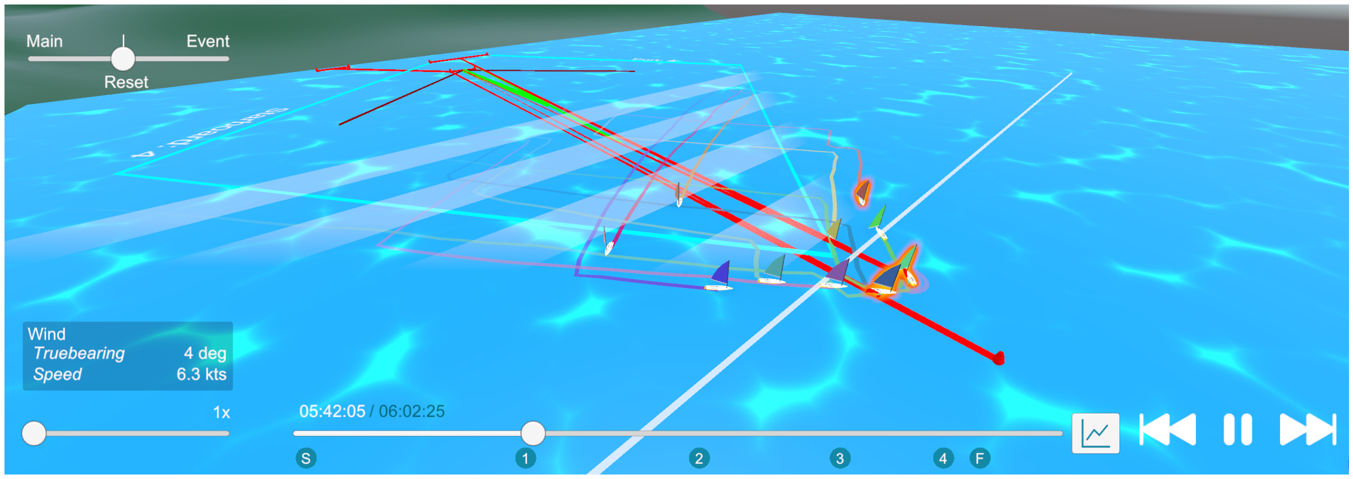
Figure 22: Center deviation of the event camera. (a) There is no information to render in the corners of the event Voronoi cell (white).
(b) The empty edges are replaced with a texture similar to the water inside the regatta field (top left and bottom right, yellow). Original images from the Sailing+ 3D Application.



(a)



(b)



(c)

Figure 23: Magnetism effect of the main camera, which attracts more pixels to it as the distance between itself and the event camera decreases. (a) Main camera view (left) increases as the distance between itself and the event camera (right) decreases. (b) The separator line as well as the event (bottom left) shrink smoothly as the main camera takes over. (c) Main camera view is visualizing the event on the entire screen space. Original images from the Sailing+ 3D Application.