Application of Convolutional Filters to Enhance Minimap Visualization in Games

Rafizan Muhammad Syawalazmi - 13523034[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]*rafi.syawalazmi@gmail.com*, *13523034@std.stei.itb.ac.id*

***Abstract*—Minimap are essential components of a video game user interface providing players the information of the surrounding areas. This paper explores the application of convolutional filters to enhance minimap visualization in games, mainly focusing on the fog of war feature. Using the game engine Unity, the implementation uses visibility masks which then convolve using a gaussian kernel for smoothing the visible areas and be used to create the fog of war effect.**

***Keywords*— Fog of War, Games, Minimap, Unity.**

I. Introduction

A minimap is a miniature map Heads Up Display (HUD) element in video games. This miniature map is often placed at the corner of the screen so players could view it while still do actions uninterrupted. Being a miniature version of the map, they must be selective on what elements or details they need to show. These elements varies between different genres of video games. Some games feature the position of the player character, the enemies, quest location, point of interest, etc.

For instance, a role-playing game might emphasize quest markers, while competitive player versus player (PvP) games prioritize enemy locations and objectives. This selective elements balances information density so the player won't be overwhelmed.

Image 1. Example of a minimap from Xaero's mod in Minecraft. Source: *https://www.curseforge.com/minecraft/mc-mods/xaeros-minimap*

A common feature in many minimaps is the "Fog of War," which restricts visibility to explored areas. The minimap will draw itself incrementally as the game progresses often by the player exploring the area. Some games like story games, after the player explored the area, it will reveal that area even after you exit that area later on. Some other games like Massively Multiplayer Online Role-Playing Game (MMORPG) or Competitive PvP games, after you cease to see the area, that area then become unknown to the player on what activity is inside that said area.

Image 2. Example of a fog of war minimap in League of Legends. Source: https://maknee.github.io/blog/2021/League-ML-Minimap-Detection2/

Convolutions are mathematical operations widely used in image processing to analyze and transform visual data. They work by applying a filter, or kernel, to an input image, creating a modified output that emphasizes specific features such as edges, textures, or patterns. By applying convolutional filters to visual elements, it becomes possible to dynamically process and enhance images or patterns, making convolutions a versatile tool for real-time visualization tasks.

This paper explores on how convolutional filters, a tool often associated with image processing, can be applied to enhance minimap visualization, specifically the fog of war visualization. By optimizing the presentation of map elements and dynamic features like fog of war, convolutional filters offer the potential to improve clarity, usability, and gameplay immersion.

II. BACKGROUNDS

A. Minimaps

Minimaps are a vital component of the Heads Up Display (HUD) in video games, offering players a simplified, real-time overview of their surroundings. Typically located in a corner of the screen, minimaps help players navigate and strategize effectively without interrupting gameplay. They provide essential spatial information while remaining unobtrusive, allowing players to focus on the primary game view. Despite their simplicity, minimaps are carefully designed to balance clarity and functionality.

Minimap designs and features vary widely across game genres, each tailored to meet specific gameplay requirements. For instance, role-playing games (RPGs) often include quest markers, non-playable character (NPC) locations, and treasure chests to assist exploration and storytelling. Strategy games prioritize a broader overview of the battlefield, highlighting units, resources, and strategic objectives. In competitive player-versus-player (PvP) games, minimaps focus on providing real-time updates, such as ally and enemy positions, to support tactical decision-making. Open-world games emphasize detailed representations of the environment, showcasing landmarks, terrain, and navigation routes to guide players through expansive game worlds.

A well-designed minimap must strike a delicate balance between information richness and clarity. Overloading a minimap with excessive data risks overwhelming players, while displaying too little information can leave them disoriented. Designers face challenges in abstracting complex 3D environments into concise 2D representations, selecting which elements to display, and ensuring smooth real-time updates as the game evolves. These challenges are compounded by the need to maintain a clear and intuitive interface that enhances the gameplay experience.

In many games using a mini-map, the mini-map begins completely blank, while the map is automatically drawn as the player discovers new areas of the game world. After players discover new areas, the terrain of the discovered area often remains visible on the mini-map. If the player's characters or units cease to see the area, the area might be covered by a fog of war, so that unit or structure movements in that area will not be shown. Things in a fog of war portion of a mini-map may not be updated until they are rediscovered [1].

B. Convolution Filters

Convolution Filters (also known as kernels) are used with images for blurring, sharpening, embossing, edge detection, and more. This is accomplished by doing a convolution between a kernel and an image [2]. The convolution process can be expressed mathematically like this:

$$g(x,y) = \omega * f(x,y) = \sum_{dx=-a}^{a} \sum_{dy=-b}^{b} \omega(dx,dy) f(x+dx, y+dy)$$

Image 3. Convolution expression. Source: https://medium.com/@ianormy/convolution-filters-4971820e851f

Where g(x, y) is the output filtered image, f(x, y) is the original image and w is the filter kernel.

To understand how convolutions transform images, imagine a grayscale image represented as a 2D grid of pixel intensities. A kernel, also represented as a small 2D matrix, is applied to a region of the image.

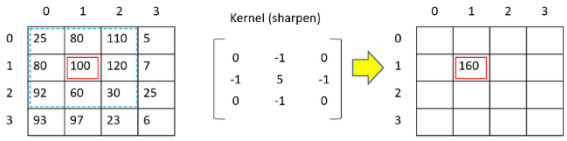

Image 4. Convolution process visualization. Source: https://medium.com/@ianormy/convolution-filters-4971820e851f

Each value in the kernel multiplies with the corresponding pixel value in the selected region of the image. Then the results of the multiplications are summed up to produce a single output value. After for the corresponding pixel is processed, the kernel slides across the image, repeating the process until all regions have been processed.

The end results is a transformed image where the effect of the kernel is visible. In the example case, it will transform the original image to a more sharper one.

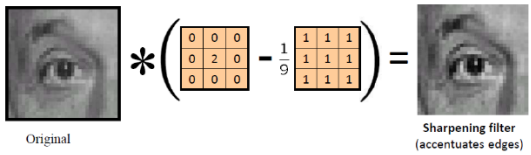

Image 5. Example end results of a sharpening filter. Source: https://jeheonpark93.medium.com/vc-convolution-based-image-denoising-sharpening-332bbe6293ff

In the context of minimap visualization in video games, convolutions can be employed to enhance features like the fog of war. By applying convolutional filters, it's possible to dynamically process and highlight important map elements, improving clarity and usability for players.

In this paper, the convolution filter will be used for the smoothing out of the lit area in the fog of war feature. By doing so, it will produce a masking layer complete of all the areas the player could see and cannot see on the minimap.

There are certain advantages by using convolutions instead of anything else. First is that convolutions are computationally efficient and can be optimized for further improvements. It is also versatile and supports a wide range of transformations and effects if needed. And lastly, convolutions can handle maps of varying size making them ideal for dynamic minimap effects.

C. Unity

Unity is a versatile and widely-used game development platform that allows developers to create 2D and 3D games and experiences across various platforms, including PC, mobile, console, and virtual reality. Launched by Unity Technologies in 2005, Unity has grown into one of the most popular game

engines globally due to its accessibility, robust feature set, and active developer community. It provides an integrated development environment (IDE) with tools for scripting, asset management, physics simulation, and more.

One of Unity's most powerful features is its support for C# scripting, which enables developers to define game mechanics, create interactive systems, and manage complex behaviors. The engine's visual editor offers a user-friendly interface for designing scenes, animating characters, and adjusting game settings. Unity also supports a component-based architecture, where objects in a game are built by attaching reusable components, such as scripts or physics behaviors, to GameObjects. This modular design simplifies game development and enhances flexibility.

Additionally, Unity's extensibility and asset pipeline are key strengths. The Unity Asset Store provides a vast library of ready-to-use assets, including 3D models, animations, scripts, and tools, which can accelerate development. Developers can also create custom editor tools and plugins to tailor Unity to their project's needs.

Unity supports a broad array of platforms, allowing developers to build and deploy their games seamlessly to Windows, macOS, Android, iOS, PlayStation, Xbox, and more. This cross-platform capability makes it an ideal choice for both indie developers and large studios seeking to reach a diverse audience.

The engine's versatility extends beyond traditional game development. Unity is used in fields such as architecture, automotive, film, and education for creating interactive simulations and visualizations. Its ability to render high-quality real-time visuals and support emerging technologies like augmented reality (AR) and virtual reality (VR) further cements its position as a leading development platform.

In this paper, Unity serves as the development environment for implementing and demonstrating the application of convolutional filters in minimap visualization. Its ease of use, combined with robust graphics and scripting capabilities, provides an ideal platform for prototyping and testing innovative game mechanics.

III. Implementation and Results

The Unity Scene that this paper will cover is just a normal scene with a player character that can move, rotate, and jump and some red blocks to represent objects in the game.

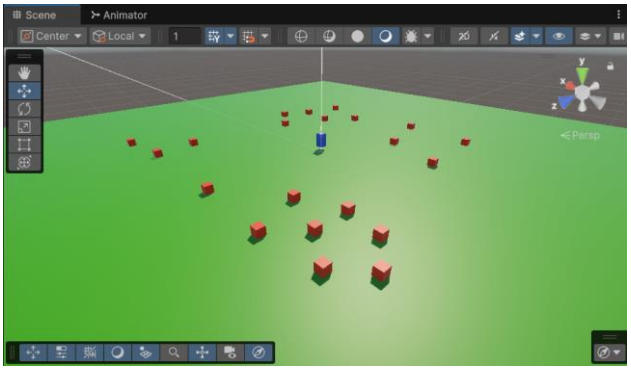

Image 6. The Implementation Scene

The main minimap uses an orthographic projection camera placed above the player viewing the entire scene. That camera then outputs the render to a render texture placed inside the assets folder. The render texture then be used for an image User Interface (UI) element inside the scene resulting in the map seen below.

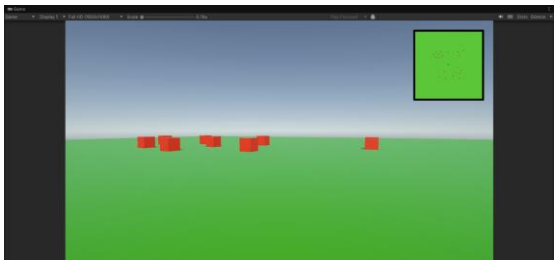

Image 7. The Minimap Implementation

Now, is for the fog of war feature implementation to come in. The fog of war feature will be an overlay image placed on top of the original minimap. This fog overlay is built using another render texture that starts fully dark representing unexplored areas.

The render texture is updated dynamically as the player character moves. The process involves creating a visibility mask that defines which areas around the player that is visible. This is done by calculating the player's location on the map and mark that spot as to be visible in the visibility mask.

After a pixel is mark to be visible, then the Gaussian Kernel is applied at the visibility mask to generate a smooth fog effect. Which the source code can be viewed below.

```
1 reference
float[,] DirectConvolve(float[,] input, float[,] kernel)
{
    int kSize = kernel.GetLength(0);
    int kRadius = kSize / 2;
    float[,] output = new float[width, height];

    for (int x = 0; x < width; x++)
    {
        for (int y = 0; y < height; y++)
        {
            float sum = 0f;

            for (int kx = -kRadius; kx <= kRadius; kx++)
            {
                for (int ky = -kRadius; ky <= kRadius; ky++)
                {
                    int ix = x + kx;
                    int iy = y + ky;

                    if (ix >= 0 && ix < width && iy >= 0 && iy < height)
                    {
                        int kernelX = kx + kRadius;
                        int kernelY = ky + kRadius;
                        sum += input[ix, iy] * kernel[kernelX, kernelY];
                    }
                }
            }

            output[x, y] = sum;
        }
    }

    return output;
}
```

Image 8. Convolution Source Code.

The result of the convolution is then applied to the render texture as a transparency value of the fog mask. The more transparent meant that the area is more visible on the map compared to non-transparent which means the mask will completely cover the area of the minimap user interface (UI).

The previous are that the player has explored are made to be less transparent than the player visible area but more transparent than unexplored areas. This is meant to show that the player has explored that certain area.

After all of that has been implemented the results are the

following.

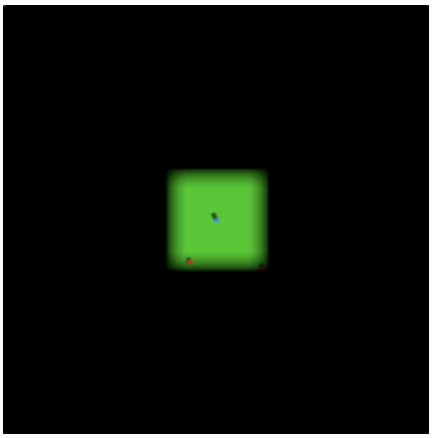

Image 9. Unexplored Minimap.

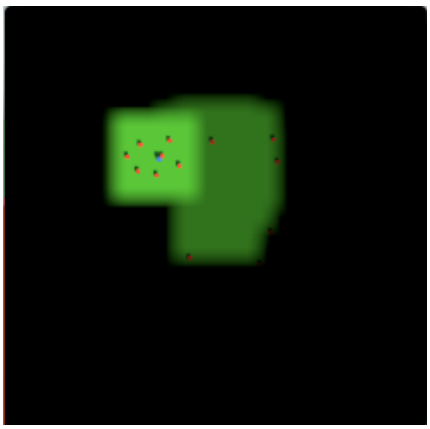

Image 10. Half-explored Minimap.

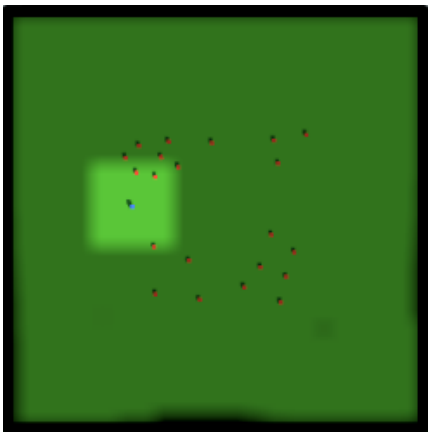

Image 11. Fully Explored Minimap.

IV. Conclusion

This paper successfully demonstrates how convolutional filters can be used to enhance the visualization of minimaps, specifically through an implementation of a fog of war system in Unity. The Implementation highlights some advantages of using convolutional system in game design that is their ability to process complex transformation in real time and with possible varying map sizes. This implementation is quite versatile too and can be used for other features like heat maps. Also this implementation proves that Unity or the game engine can easily calculate and supports mechanics using convolutions.

Further projects can explore more uses of the convolutional system in enhancing the minimap effects and also finding a more optimized way to do the convolutions.

V. Acknowledgment

The Author would like to express our heartfelt gratitude to everyone who contributed to the successful completion of the paper.

The Author extend his deepest appreciation to his linear and geometric algebra teacher, Mr. Rinaldi Munir, for their teachings to make this paper possible and encouraging us to make this paper in the first place.

This paper is collaborative effort and shared knowledge of all those involved. Thank you.

References

[1] Adams, Ernest (2014). Fundamentals of Game Design. New Riders. ISBN 9780133435719.

[2] https://medium.com/@ianormy/convolution-filters-4971820e851f accessed on 1st January 2025 at 16.31.

Pernyataan

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 1 Januari 2025

Rafizan Muhammad Syawalazmi - 13523034