Image Blending: Project Report for CS-672 Advanced Topics in Computational Vision

Ioannis Kaziales*
Department of Computer Science
University of Crete
Voutes, 715 00, Heraklion Crete
kaziales@csd.uoc.gr
https://github.com/JohnnyKaz/ComputerVision/tree/main/Image-Bleding

Abstract

Editing digital images is considered one of the cornerstones of computer vision. Among the different image editing tasks, this report focuses on image blending. Captivating both artists and researchers, image blending allows the seamless merging of multiple images, resulting in visually compelling compositions. We will delve deeper into this interesting task, exploring the core principles behind classical blending techniques, examining the capabilities and limitations of each method, and highlighting their effectiveness in different scenarios. Finally, we will discuss promising research directions that could push the boundaries of image blending.

Index Terms:
computer vision, image composition, blending, feathering, Poisson, Laplacian pyramid

1 Introduction & Problem Definition

In computer vision, image composition [6] is defined as the process of merging a foreground from one image with the background from another image, creating a unified and believable final product.

This task remains an active research area, filled with interesting applications in various fields. For instance, in the fields of artistic creation and entertainment, image composition can assist people in crafting entirely novel artworks, realistic panoramic images by stitching different photos, or self-portraits with an exciting, yet believable background. Additionally, there are multiple commercial benefits, such as allowing clients to try on clothes, furniture, or paint colors virtually before purchase, or assisting in the creation of personalized advertisements where products or logos seamlessly blend into the desired setting. Finally, image composition could create synthetic images that closely resemble real-world examples following a certain data distribution, serving as valuable training data for neural network tasks, like object detection and segmentation.

However, achieving a natural-looking composite image can be quite challenging. To tackle this complex task, researchers often break image composition down into smaller, more manageable sub-tasks. The initial step of the composition process typically involves deciding the desired location, scale, and rotation of the foreground object in the blended image and then extracting the foreground object using image segmentation algorithms [5]. Nevertheless, these algorithms are not always perfect, often leaving behind rough edges and imperfections at the object's borders. When simply pasting this "cut-out" onto the background, these jagged edges create obvious visual artifacts. The inconsistency in appearance between the foreground object and the background is one of the major issues in realistic composition and can manifest as obvious color discrepancies or uneven lighting distribution where the objects meet.

*Graduate student [personal webpage]

To address this issue, the subtask of **image blending**, which is the focus of our research, aims to smooth out these transitions and seamlessly integrate the foreground object into the background. Recently, the impressive image generation capabilities of neural networks, particularly diffusion models, have led some researchers [12], [9] to use these models for performing multiple composition subtasks concurrently. Essentially, they build a single, unified model that directly produces the final composite image, with the foreground seamlessly blended within the background. These "generative" composition methods regenerate the foreground object inside the target image, using image inpainting techniques, instead of making targeted adjustments to it. However, in this work, we choose to focus exclusively on "traditional" techniques that do not rely on neural networks. As we will demonstrate, these established methods can achieve compelling results, even surpassing generative approaches in certain scenarios.

2 Related Work

Traditional image blending techniques aim to create seamless transitions between a foreground object and its background over a given region by minimizing the visibility of the seams. This is achieved by merging information from the overlapping region of two images in such a way that the boundaries of the images involved become imperceptible. In this section, we will explore some popular image blending methods developed over the years. A table summarizing their strengths and weaknesses is provided in Table 1.

First, we need to define some key terms (visualized in Figure 1):

- Blended Image (I_B): The final image resulting from the blending process.
- Source Image (I_S): The image containing the foreground object.
- Target Image (I_T): The image containing the desired background.
- Foreground Region (Ω): The specific area in the source image that contains the object to be blended.
- Boundary ($\partial\Omega$): The edge of the foreground region.
- Mask (M): An array of the same size as the source and target images, with values ranging from 0 to 1 indicating the origin of the pixel values in I_B. A value of 1 corresponds to the foreground object, while 0 signifies the background.

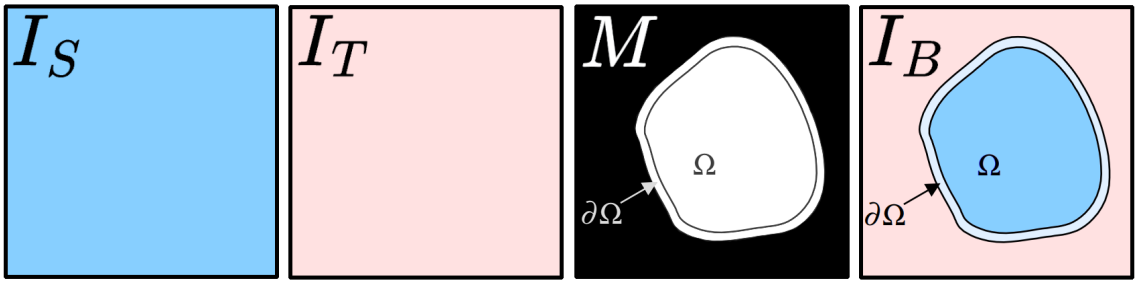


Figure 1: Image blending terms. From left to right: the source image I_S which contains the foreground object, the target image I_T with the desired background, the mask M with values $\in [0, 1]$, and the final composite image I_B.

Table 1: Comparison of different "traditional" image blending methods

Method	Strengths	Weaknesses
Alpha/Feathering	fast, simple	visible seam, fuzziness, ghosting, 'halo'
Laplacian Pyramid	smooth seam, preserves details	artifacts, computationally expensive
Poisson Equation	seamless transition, changes color/texture, extra editing operations	color bleeding, sensitive to initialization, the most computationally expensive

2.1 Alpha blending

Alpha blending, also known as ***feathering*** [8], is a technique that assigns alpha values to pixels along the boundary between foreground and background. These alpha values, which range from 0 (fully transparent) to 1 (fully opaque), indicate the contribution of the foreground from I_S to the final color. However, users need to manually define these alpha values.

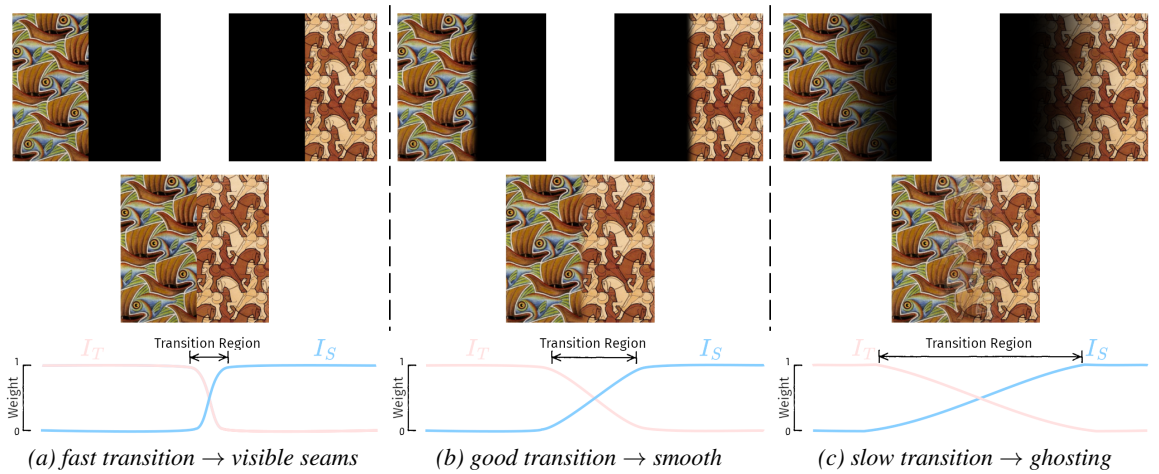
The blended image is calculated using the following equation:

$$I_B = \alpha \cdot I_S + (1 - \alpha) \cdot I_T$$

where for each pixel p: $\alpha = \begin{cases} 1 & \text{if } p \in \Omega \quad \text{(foreground)} \\ 0 & \text{if } p \notin \Omega \quad \text{(background)} \\ \text{between 0 and 1} & \text{if } p \in \partial\Omega \quad \text{(transition region)} \end{cases}$

As shown in Figure 2, the quality of the blended image is influenced by the size and type of the transition zone. Specifically, if the transition between the foreground and background is abrupt (small window size), the boundaries between them become noticeable, creating unrealistic results. However, a slow transition (large window size) leads to "ghosting" effects, where semi-transparent versions of both the foreground and background appear simultaneously. Determining the optimal window size can be challenging but, if chosen correctly, the resulting image should have a smooth, yet not ghosted, transition.

Despite its simplicity and speed, alpha blending presents several limitations, including a potential loss of detail since it blurs fine details, the introduction of unpleasant visual effects like "halos" (bright or dark rings surrounding the foreground), and the fact that users need to find the balance between narrow and wide window sizes. Thus, it is considered a baseline technique but is often not ideal for high-quality results. It works well when the transition region is well-defined and the images have similar colors and textures.



(a) fast transition → visible seams *(b) good transition → smooth* *(c) slow transition → ghosting*

Figure 2: Impact of the size of the Transition Region on Alpha Blending

2.2 Laplacian pyramid blending

The concept of ***Laplacian pyramid blending*** [1] involves constructing multi-scale Laplacian pyramids for two images and applying *alpha blending* at each scale. The final result is achieved by adding the blended outcomes from the different scales. This method addresses the limitations of traditional *alpha blending*, based on two important observations.

When directly blending images with *alpha blending*, the size of the transition zone between them is directly linked to the size of the image features. To eliminate visible seams, the width of the zone needs to be at least as large as the largest prominent features. However, to avoid a ghosting effect, the zone shouldn't be considerably larger than the smallest features. Therefore, a suitable transition width can only be determined if the images share a similar, narrow spatial frequency range. The first observation is that images can first be decomposed into multiple band-pass component layers, each capturing specific frequency details. This allows for independent blending at each layer, offering finer control over the transition regions. Coarse structures should blend slowly between images (using a wide transition zone), while fine details should transition quicker (using a narrow transition zone). Then, the individual blended layers can be reassembled to form the final blended image.

The second observation is that this multi-resolution blending can be defined rather simply and elegantly, using two different types of pyramids and their properties:

A ***Gaussian pyramid*** with N levels is created by repeatedly applying a Gaussian (smoothing) filter K to an image I and downsampling the result to half the resolution. Each level of the pyramid contains a progressively coarser representation of the original image that reduces the high-frequency details while retaining the overall structure. The resulting sequence of low-pass filtered (blurry) images $G_0, G_1, \ldots, G_N$ forms the Gaussian pyramid, which is formally defined as:

$$G^I = \begin{cases} G_0 = I \\ G_i = (K * G_{i-1})_{\downarrow 2} \quad \text{for } i = 1, 2, \ldots, N \end{cases}$$

A ***Laplacian pyramid*** with N levels is constructed to extract the band-pass components necessary for multi-resolution blending. This process involves subtracting an upsampled version of each level in the Gaussian pyramid from the level above it, creating images that capture the high-frequency details and edges that are lost when going from one level of the Gaussian pyramid to the next. This process yields a sequence of band-pass images $L_0, L_1, \ldots, L_N$ capturing specific frequency bands, forming the Laplacian pyramid:

$$L^I = \begin{cases} L_N = G_N \\ L_i = G_i - (G_{i+1})_{\uparrow 2} = G_i - ((K * G_i)_{\downarrow 2})_{\uparrow 2} \quad \text{for } i = 0, 1, \ldots, N-1 \end{cases}$$

An ingenious aspect of this method, as illustrated in Figure 3, is the recognition that the steps involved in constructing the Laplacian pyramid L^I can be reversed to precisely recover the original image I by iteratively upsampling each level of L^I and adding it to the level above it:

$$I = \sum_{i=0}^{N} (L_i)_{\uparrow \text{ to full size}}$$

Building on the previous concepts, the core of the **Laplacian pyramid blending** method lies in blending separately the corresponding levels of the Laplacian pyramids from both images I_S and I_T. This is achieved using *alpha blending*, but with a crucial twist. A Gaussian pyramid of the mask M is used to determine the transition region for each level. This allows for wider transition regions at lower levels (capturing coarse structures) and narrower transitions at higher levels (preserving fine details). Once the individual layers are blended, we can create the final blended image by iteratively adding and upsampling the blended levels back together. This process effectively recombines the low and high-frequency details, resulting in a seamless blend that preserves details across different scales. The precise algorithm formulation is as follows:

Step 1: Build Laplacian pyramids L^S and L^T for the source and target image respectively.
Step 2: Build a Gaussian pyramid G^M for the mask M
Step 3: Form a Laplacian pyramid L^B by combining L^S and L^T with *Alpha Blending* using G^M as weights separately for each level: $L_i^B = G_i^M \cdot L_i^S + (1 - G_i^M) \cdot L_i^T, \qquad i = 0, \ldots, N$
Step 4: Obtain the final blended image I_B by upsampling and summing the levels of L^B.

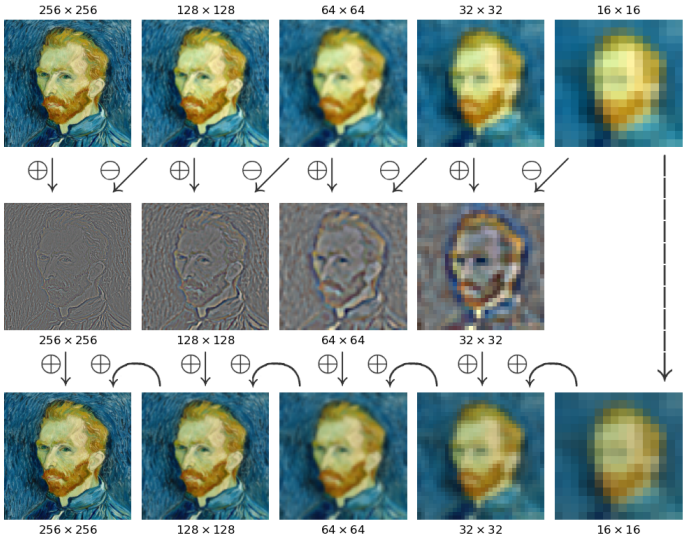


Figure 3: Image Reconstruction. Gaussian pyramid (top), Laplacian pyramid (middle), Reconstructed image (bottom).

Although it requires more computational resources as a result of the pyramid construction and manipulation steps, Laplacian pyramid blending offers superior control and reduces visible artifacts compared to alpha blending. The result is often more visually pleasing, as the transition appears smoother and less jarring, even though subtle texture differences might still be noticeable.

2.3 Poisson blending

Laplacian pyramid blending simply smooths the transition region $\partial\Omega$ between I_S and I_T, leaving the interior of the source image unchanged. This works well when the colors of the images are similar near the blending region. However, if the colors differ substantially, an obvious color mismatch can occur in the blended image I_B, reducing the realism. ***Poisson image blending*** [7] offers a solution by modifying the color of the source image to better match the color of the target, while preserving the important details of the foreground. An example is illustrated in Figure 4.

Figure 4: ***Blending images with different colors****. Laplacian pyramid (left) creates an unnatural image with a blue "halo" around the airplane, while Poisson blending (right) modifies its color, giving it an orange hue to adapt to the background.*

This approach leverages the insight from psychometric studies [4] that the human eye is more sensitive to contrast rather than absolute intensity values in images, so the edges (that can be extracted using a Laplacian operator) are perceptually the most significant. Consequently, it focuses on blending in the gradient domain, which captures those image edges and visual details. This shift in focus from intensity to gradient information allows *Poisson blending* to seamlessly integrate images with different color palettes, producing visually compelling and realistic results.

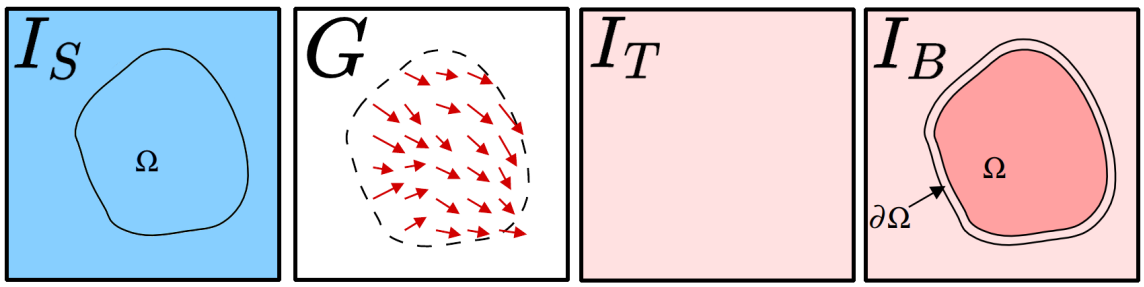


Figure 5: ***Guided interpolation notation****. The unknown values of the blended image I_B in domain Ω interpolate the known values of the target image I_T, under the guidance of the vector field G, which could be (but is not limited to) the gradient of a source image ∇I_S.*

The notation used here has some differences compared to the one used in the previous sections, as visualized in Figure 5. The core aim of this technique is to create a blended image (I_B) where the gradients (∇I_B) resemble as closely as possible a guidance vector field (G). This guidance field can be constructed using the gradients of the source image (∇I_S). Simultaneously, the intensity values (I_B) on the boundary ($\partial\Omega$) of the blended image must match the target image (I_T) to ensure a seamless transition. To achieve these goals, the following constrained minimization problem needs to be solved:

$$\min_{I_B(x,y)\in\Omega} \iint_{\Omega} ||\nabla I_B(x,y) - G(x,y)||^2 dx\, dy$$
$$\text{s.t.} \quad I_B(x,y) = I_T(x,y) \qquad \text{on } \partial\Omega$$

Here, the equation minimizes the difference between the blended image's gradient ∇I_B and the guidance field G, while ensuring that the intensity values on the boundary $I_B|_{\partial\Omega}$ match the target image $I_T|_{\partial\Omega}$.

This minimization problem can also be interpreted as a guided interpolation task. Essentially, a constraint is specified on the boundary of the blended image and the gradient information of the foreground object is propagated inwards from those boundary pixels. The solution to the minimization problem takes the form of a Poisson equation:

$$\begin{aligned} \nabla^2 I_B(x,y) &= \nabla G(x,y) \quad \text{in } \Omega \\ I_B(x,y) &= I_T(x,y) \quad \text{on } \partial\Omega \end{aligned}$$

In the field of differential equations, it is well known that a scalar function (like the intensity values I_B in our case) within a bounded domain (Ω) is uniquely determined by its values on the boundary ($\partial\Omega$) and its Laplacian within the interior ($\nabla^2 I_B$). This ensures the existence of a unique solution to the Poisson equation, leading to a well-defined algorithm: given methods for crafting the Laplacian and boundary conditions for an unknown image over some domain, the Poisson equation can be solved numerically to achieve seamless filling of that domain. This process can be applied independently to each color channel of an image. The specific details of discretizing this problem are further discussed in Section 3.4.

Moreover, Poisson blending offers great flexibility by allowing creative control over the guidance field (G) that shapes the outcome of the blending process. The most basic approach, "**Normal Clone**", directly uses the gradient field of the source image (∇I_S), enabling seamless removal of unwanted objects or blemishes or insertion of new elements as if they were part of the original scene. However, the true strength of Poisson blending is its acceptance of "non-conservative" guidance fields. These fields don't follow the traditional rules of image gradient formation (they don't need to be defined as the gradient of an existing image), allowing for even more compelling visual effects. For instance, there are situations where combining properties of I_S with those of I_T is desirable, such as blending objects with partial transparency or holes on top of a textured background, or blending an object near another object in the target image while reducing the color bleeding that would otherwise occur. A simple but somewhat limited approach would involve creating a *linear combination* of the source and target gradient fields ($\alpha\nabla I_S + \beta\nabla I_T$). However, this method may "wash out" textures and reduce details, so it is usually avoided. The "**Mixed Clone**" offers greater refinement by selecting the strongest variations from either the source or target image at each point (x, y) within the blending region Ω. Formally, this is expressed as:

$$\begin{array}{l|l} \textit{Normal Clone}: & \textit{Mixed Clone}: \\ G(x,y) = \nabla I_S(x,y) & G(x,y) = \begin{cases} \nabla I_T(x,y) & \text{if } \; ||\nabla I_T(x,y)|| > ||\nabla I_S(x,y)|| \\ \nabla I_S(x,y) & \text{otherwise} \end{cases} \end{array}$$

While previous methods often struggle with visible seams and color inconsistencies, Poisson blending generally produces more pleasing results. This technique excels at adapting the foreground object to the background's color and texture, making it ideal for seamlessly integrating objects into new environments. However, it also comes at a cost. Solving the Poisson equation, which involves a complex system of linear equations with every pixel in Ω as an unknown variable, can be computationally expensive and time-consuming. Additionally, the colors of the background may seep through the foreground too much (color bleeding), potentially distorting the foreground object and compromising its details.

3 Technical Details & Experiments

This section dives into the methodology, technical details, and the experiments conducted as part of our project. Our primary goal was to create an efficient and accurate technique for image blending. Thus, we implemented the established "traditional" methods from relevant literature, as discussed in Section 2. While implementing these algorithms may seem straightforward since they require only a few lines of code, the devil truly lies in the details. Despite the apparent simplicity, even seemingly minor oversights and subtle nuances can lead to frustrating and challenging bugs. Here we will describe our implementation choices, giving special emphasis to some of these often-overlooked details that can cause issues. Figure 14 showcases images blended with the implemented techniques.

3.1 Preparing the Data

This project focuses solely on image blending. Therefore, the images involved must be of identical size, and the location, scale, and rotation of the foreground object in the final blended image need to be predetermined (it is considered a separate subtask of image composition, namely *object placement*). To expedite and simplify the creation of images in the correct format for our experiments, we developed a graphical user interface (GUI) program. This GUI Editor program allows users to visualize both I_S and I_T with the foreground object superimposed onto the background with adjustable scale, orientation, location, and opacity. It is also tasked

with creating and modifying binary masks through two options: (1) drawing/erasing polygons with points indicated by mouse clicks and (2) freehand drawing/erasing using strokes of a brush with varying shapes and sizes. The blending algorithms discussed require only loose lasso selections for the region Ω in the mask. Then, it can save the resulting images in a user-friendly format for further processing. This GUI program is showcased in Figure 6.

(a) placing the foreground with the desired scale and rotation

(b) editing the mask with polygons from selected points (mouse clicks)

(c) editing the mask manually with brush strokes for greater control

(d) the generated pre-processed images, ready for blending

Figure 6: Using the Editor program to prepare the images I_S, I_T, M for image blending.

3.2 Alpha Blending

Alpha blending is a simple, and straightforward method. We are given a binary mask M, where a value of 0 represents the background and 1 represents the foreground. In its most basic form, we could directly use this mask as the alpha channel, thus copying the foreground of I_S exactly onto the background of I_T in the region Ω. However, this approach creates visible seams and an unnatural transition at the edges where the images meet. This abruptness occurs due to the sudden change in the mask, similar to using a unit-step window where the values jump directly from 0 to 1. In practice, to achieve a more natural and visually pleasing blend, a Gaussian filter is applied to the mask. This filter essentially blurs the mask, smoothing the edges and creating a gradual transition between the foreground and background. The resulting smoother version of the mask M' is then used as the alpha channel. The width and extent of this transition are controlled by the standard deviation σ of the Gaussian filter. Finding the ideal value for σ is crucial. Setting it too low results in a rough and noticeable transition. Conversely, setting it too high creates a smooth blend but can lead to a loss of detail in the foreground object, causing a "ghosting" effect where the foreground appears faded. Therefore, striking a balance between smoothness and detail preservation is essential. This trade-off is demonstrated in Figure 7.

3.3 Laplacian Pyramid Blending

This section explores Laplacian pyramid blending, a technique for seamlessly merging two images. Figure 9 provides a visual breakdown of the process. To construct the Gaussian and Laplacian pyramids, we leverage OpenCV functions for image downsampling and upsampling.

One crucial aspect to consider is the image shape at each pyramid level. During downsampling, in theory, the resulting image becomes a quarter the size of the image of the previous level. However, to prevent unwanted visual artifacts at the blended image's borders in practice, it is essential to round up the division during downsampling. For an image with dimensions $m \times n$ at level i, the dimensions at level $i+1$ should be $\lceil \frac{m}{2} \rceil \times \lceil \frac{n}{2} \rceil$. This ensures that every level in the pyramid contains at least a single pixel (an image with dimension 1×1), regardless of the chosen number of levels. It is also worth noting that we can calculate an upper bound for the number of pyramid levels using the formula $max\left(\lceil log_2(m) \rceil, \lceil log_2(n) \rceil\right)$. Adding more levels beyond this limit offers no improvement to the final result and only increases computational cost and memory usage because these additional levels would all contain the same single-pixel image.

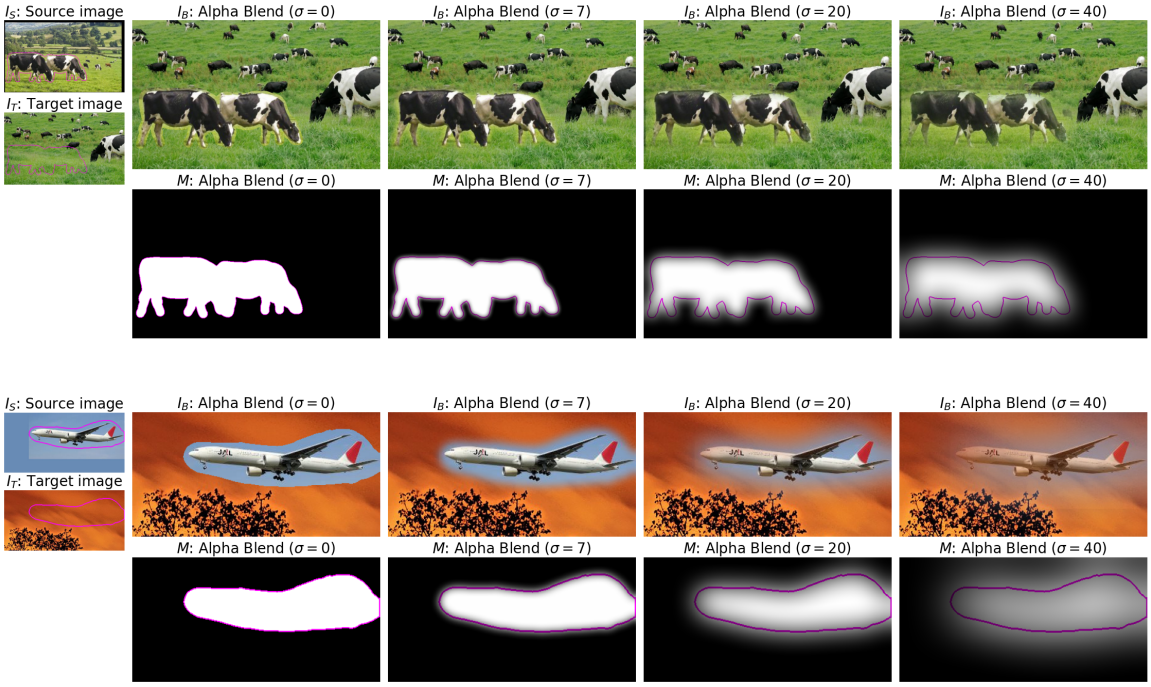


*Figure 7: Impact of different σ on **Alpha blending**. As σ increases, both the blended image I_B (top) and the mask M' (bottom) have a wider, smoother, and more natural transition region, but the foreground starts to fade.*

Figure 8 showcases an example where two images are blended with varying numbers of pyramid levels. As the number of levels increases, the transition between the images becomes smoother and a subtle change in their colors becomes more notable. However, this comes at the expense of higher computational demands and memory requirements due to the extra pyramids.

Figure 8: Blending two self-portraits of Van Gogh using Laplacian pyramids with different level count. As the number of levels increases, the transition gets less noticeable (e.g. the red area around the hat progressively fades away), while slight color adjustments occur (e.g. both the hat's warm purple hue and the pipe shift to a cooler blue color, while the vibrant blue background becomes more muted with brownish hues).

3.4 Poisson Blending

To implement Poisson Blending, we first need to discretize the underlying mathematical formulation. This involves calculating the Laplacian operator ∇^2. A fast and efficient way to approximate this is by using a small 3x3 filter: $\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$

If we assume that the guidance gradient field G is the gradient of a known image g, we need to solve the Poisson equation $\nabla^2 I_B = \nabla G = \nabla^2 g$. To discretize this equation, we build a system of linear equations from each pixel in the blended image using the Laplacian filter.

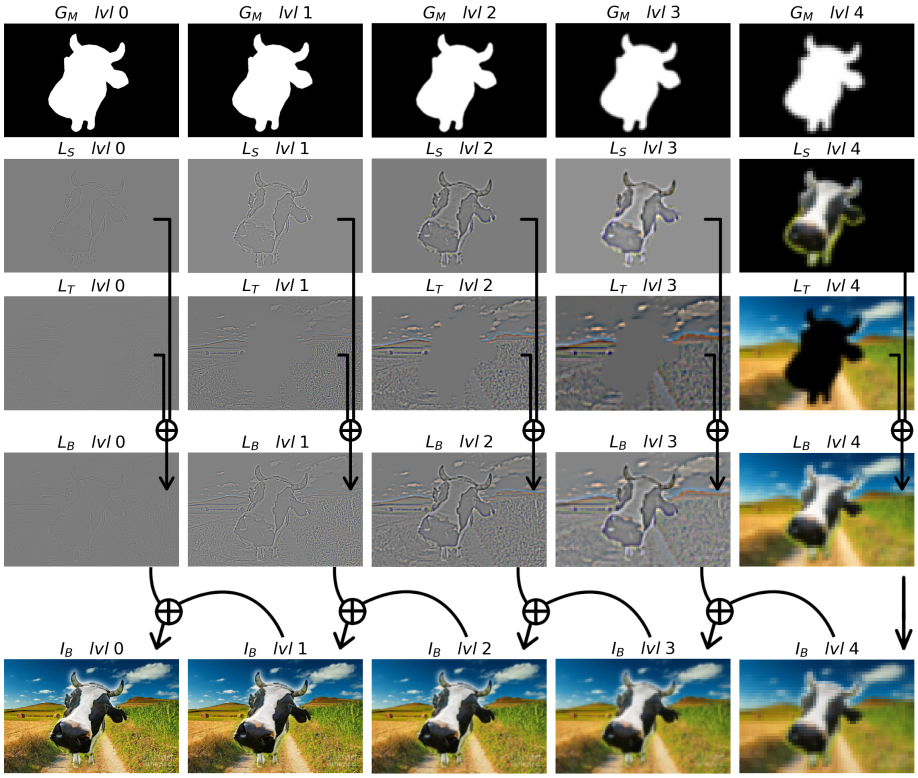


Figure 9: Forming the image shown in Figure 6 using Laplacian pyramid blending with 4 levels. Each column represents a level in the pyramid, with images becoming progressively blurrier as we move to the right. Blurred versions of the masks M' from the Gaussian pyramid (top row) are independently used for feathering each level of the source and target Laplacian pyramids (second and third rows respectively), crafting the Laplacian pyramid of the blended image (fourth row). The final image is obtained by adding all the individual Laplacian levels.

There are two key cases to consider (also shown in Figure 10):
For pixels (x, y) whose **neighborhood is fully inside** Ω, we only need to satisfy the condition $\nabla^2 I_B = \nabla^2 g$. We can achieve this with the equation:
$I_B(x-1,y) + I_B(x+1,y) + I_B(x,y-1) + I_B(x,y+1) - 4 \cdot I_B(x,y) =$
$g(x-1,y) + g(x+1,y) + g(x,y-1) + g(x,y+1) - 4 \cdot g(x,y)$
This way, we get a set of linear equations:

$$\begin{bmatrix} & & & & & & & & & & \\ \dots & 0 & 1 & 0 \dots & 1 & -4 & 1 & 0 \dots & 1 & 0 \dots \\ & & & & & & & & & & \end{bmatrix} \begin{bmatrix} I_{B,1} \\ I_{B,2} \\ \vdots \\ I_{B,N} \end{bmatrix} = \begin{bmatrix} \\ \square \\ \\ \end{bmatrix}$$

For Pixels (x, y) with **neighborhood not fully inside** Ω, we also need to incorporate the boundary condition $I_B|_{\partial\Omega} = g|_{\partial\Omega}$. This is achieved by replacing the terms $I_B(x', y')$ in the previous equation with the actual target image values $I_T(x', y')$ in our system of equations for all pixels (x', y') on the boundary.

By combining these cases, we arrive at a large, sparse system of linear equations. This system can be represented as a sparse matrix, where most of the elements are zero. We can then leverage efficient sparse solvers from libraries like SciPy to solve this system and obtain the final blended image I_B.

To make Poisson Blending more efficient, we can leverage a few implementation tricks: Instead of processing the entire image, we can identify the specific region to be blended using the given binary mask M. We then calculate the bounding box of this region. All the calculations are performed only for pixels within this bounding box. This significantly reduces computation time because each additional pixel in the calculation translates to an unknown variable in the linear system. Solving a system with fewer unknowns translates to a smaller matrix equation, requiring less processing power and time.

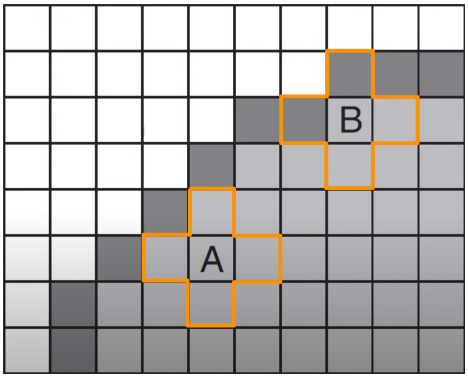


Figure 10: ***Discretizing the Poisson equation*** *the pixels inside Ω is shown as gray pixels and the boundary $\partial\Omega$ is a darker gray. The neighborhood of pixel A lies fully on Ω, while some pixels in the neighborhood of B are in $\partial\Omega$, so they need to satisfy the boundary conditions.*

Extracting the boundary of the masked region $\partial\Omega$ can be achieved efficiently using morphological operations. We opted to define the boundary as the pixels just inside the region Ω, rather than the thin outline outside it. Here's a breakdown of the steps involved:

1. To handle masks that touch the image borders and ensure those borders are included in the boundary, we pad the image with one extra pixel of zeros all around.
2. We apply erosion to the binary mask with the structuring element $E = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$
3. The final boundary $\partial\Omega$ is obtained by performing a bitwise XOR operation between the original and the eroded mask: $\partial\Omega = M \; XOR \; (M \ominus E)$

While Poisson Blending typically transfers the entire content from the source image, there might be situations where we only want a specific aspect, like the texture (pattern) but not the color. A simple solution is to convert the source image to grayscale (monochrome) before using it in the blending process. This ensures that only the intensity variations (texture) are incorporated into the final image, as demonstrated in Figure 11.

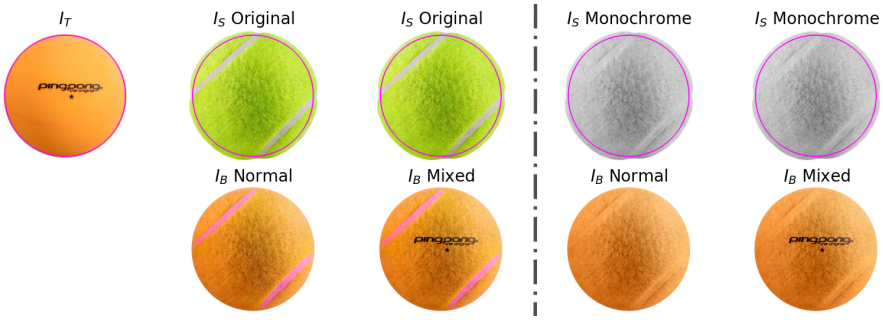


Figure 11: ***Texture transfer with monochrome source*** *In tasks like texture transfer, the part of the source color remaining after seamless cloning might be undesirable (e.g. green hue on the blended ping-pong ball, or pink stripes). This can be fixed by turning the source image monochrome beforehand.*

3.5 Gradient Domain Editing

Beyond seamless image composition, Poisson Blending offers can be used as a tool for image editing. It allows us to manipulate images based on a guidance field derived entirely from the original image, leading to some interesting effects.

One such application is localized color manipulation. Given the original color image and the region Ω, Poisson Blending lets us seamlessly blend two different color variations of the same image, where one variation serves as the background outside Ω and the other provides the desired colors for the foreground inserted within Ω. For instance, we could use this technique to selectively desaturate an image, leaving only a specific object in

its original color. This can be achieved by using the original color image as the source I_S and its grayscale version as the target I_T. Figure 12 showcases this technique in action.

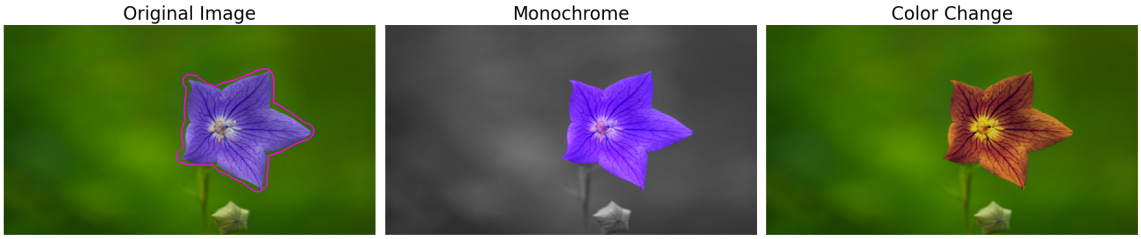


Figure 12: ***Local color adjustments****. (a) Original image with Ω loosely surrounding an object. (b) Background desaturation. (c) Object recoloring by adjusting RGB channels of I_S (multiplying them by 1.8, 1.5, 0.2 respectively).*

4 Discussion & Future Work

While the methods we presented in this project rely on older segmentation assumptions, the field has witnessed a significant leap forward with the rise of neural networks. Notably, the "Segment Anything Model" (SAM) [3] demonstrates impressive amortized real-time zero-shot promptable object segmentation capabilities. This raises the question of the relevance of these older methods. We argue that blending remains important. Segmentation algorithms, despite their progress, are not infallible. Traditional blending methods offer a valuable framework to address segmentation imperfections. Additionally, these techniques are simple, efficient, and produce satisfactory results. Advancements in segmentation can enhance these methods, improving the visual quality of the results and opening new possibilities.

Several improvements can be made to enhance the efficiency of the proposed blending algorithms and final image quality. More specifically, the selection of the boundary $\partial\Omega$ significantly impacts the blended image, and even minor variations can alter the outcome. Currently, mask selection is often a manual, imprecise process. Literature has explored algorithms like shortest closed-path approaches to optimize the boundary selection [2], [10]. An example is shown in Figure 13. Another area of focus is minimizing color bleeding and halo effects in the blended images. To that end, [11] proposes a promising two-step approach we could implement that involves processing boundary gradients and then reconstructing the image using a weighted integration scheme. Also, accelerating Poisson image blending [10] is desirable, as solving the Poisson equation is computationally expensive. Additionally, we would like to make image blending more accessible. We envision a user-friendly GUI program that enables real-time (or amortized real-time) promptable image blending. This GUI would offer options for manual control over blending algorithms and parameters, while also providing automatic settings. Users would simply load source and target images, use a mouse hover or a few clicks to designate the foreground object, and the blended image would be instantly generated. Behind the scenes, SAM would segment the object, providing a mask Ω_{obj}. If a user doesn't specify a different mask Ω_0, we could generate a more inclusive mask Ω_0 using dilation and subsequently optimize the mask boundary before feeding it into the chosen blending algorithm. We hypothesize that even simpler blending methods like feathering or Laplacian pyramid blending could suffice when combined with high-quality segmentation results. However, further experimentation is required to validate this.

5 Conclusion

This project investigated various image blending algorithms, each offering unique advantages and considerations. *Alpha blending* provides a straightforward and efficient approach. By incorporating a carefully chosen Gaussian blur, we can often achieve a visually pleasing balance between simplicity and quality. However, this method may result in less natural transitions or unwanted ghosting effects. *Laplacian pyramid blending* strikes a good compromise between speed and detail. It delivers smoother transitions between images with slight color adjustments for improved coherence, and avoids color bleeding. This technique is particularly useful for combining foreground and background elements with contrasting colors, but may struggle with more significant color inconsistencies.

Poisson image editing tackles this challenge and seamlessly integrates the foreground object by adjusting its color to match the background at the boundary. Laplacian pyramid blending attempts to insert a fast approximation of the source Laplacian in the destination region. This method goes beyond that and inserts the source Laplacian exactly by precisely solving a Poisson equation. It excels at creating seamless transitions for

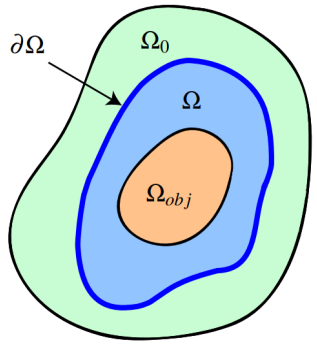


Figure 13: ***Selecting the best boundary for blending****. A rough, manual region Ω_0 is provided, which completely encloses the object of interest Ω_{obj} (extracted using a segmentation algorithm). The optimized boundary $\partial\Omega$ lies inside $\Omega_0 \setminus \Omega_{obj}$.*

complex scenarios due to its focus on gradient smoothness. When employing Poisson blending, the choice between "Normal Cloning" and "Mixed Cloning" depends on the desired outcome: Normal Cloning preserves the foreground object's texture while seamlessly integrating it into the background. This is ideal when the color palettes and textures between images are similar (e.g. for blending facial features onto another face). Mixed Cloning retains textures and details from both images in the final composition. This proves valuable when the desired effect is to superimpose an image to a background and retain information about the background (e.g. blending a painting on top of a brick wall). It is also useful when blending text onto an image (e.g. adding a watermark) or incorporating partially transparent objects like smoke or glass.

In conclusion, while each technique has its limitations, the optimal choice depends on the specific application and the desired visual effect. Understanding these strengths and weaknesses empowers users to select the most suitable method for crafting seamless and visually captivating image compositions.

References

[1] Peter J Burt and Edward H Adelson. A multiresolution spline with application to image mosaics. *ACM Transactions on Graphics (TOG)*, 2(4):217–236, 1983.

[2] Jiaya Jia, Jian Sun, Chi-Keung Tang, and Heung-Yeung Shum. Drag-and-drop pasting. *ACM Transactions on graphics (TOG)*, 25(3):631–637, 2006.

[3] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, et al. Segment anything. *arXiv preprint arXiv:2304.02643*, 2023.

[4] Edwin H Land and John J McCann. Lightness and retinex theory. *Josa*, 61(1):1–11, 1971.

[5] Shervin Minaee, Yuri Boykov, Fatih Porikli, Antonio Plaza, Nasser Kehtarnavaz, and Demetri Terzopoulos. Image segmentation using deep learning: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 44(7):3523–3542, 2021.

[6] Li Niu, Wenyan Cong, Liu Liu, Yan Hong, Bo Zhang, Jing Liang, and Liqing Zhang. Making images real again: A comprehensive survey on deep image composition. *arXiv preprint arXiv:2106.14490*, 2021.

[7] Patrick Pérez, Michel Gangnet, and Andrew Blake. Poisson image editing. In *Seminal Graphics Papers: Pushing the Boundaries, Volume 2*, pages 577–582. ACM New York, NY, USA, 2023.

[8] Thomas Porter and Tom Duff. Compositing digital images. In *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 253–259, 1984.

[9] Yizhi Song, Zhifei Zhang, Zhe Lin, Scott Cohen, Brian Price, Jianming Zhang, Soo Ye Kim, and Daniel Aliaga. Objectstitch: Object compositing with diffusion model. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18310–18319, 2023.

[10] Richard Szeliski, Matthew Uyttendaele, and Drew Steedly. Fast poisson blending using multi-splines. In *2011 IEEE International Conference on Computational Photography (ICCP)*, pages 1–8. IEEE, 2011.

[11] Michael W Tao, Micah K Johnson, and Sylvain Paris. Error-tolerant image compositing. *International journal of computer vision*, 103:178–189, 2013.

[12] Binxin Yang, Shuyang Gu, Bo Zhang, Ting Zhang, Xuejin Chen, Xiaoyan Sun, Dong Chen, and Fang Wen. Paint by example: Exemplar-based image editing with diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18381–18391, 2023.

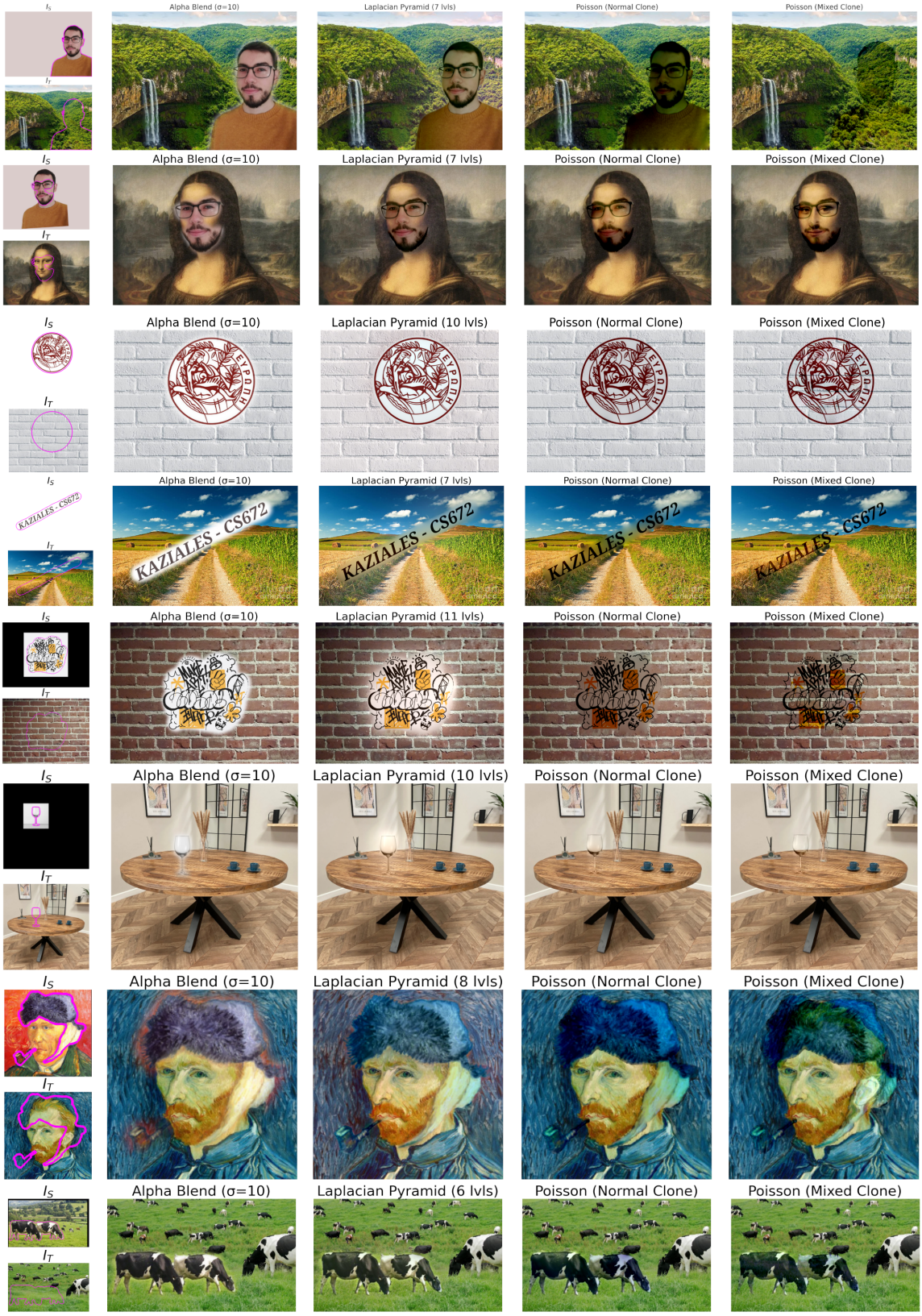


Figure 14: ***Exploring Image Blending Techniques****. Various image blending methods as discussed throughout this document. The original source and target images are displayed in column (a). Subsequent columns demonstrate the results of different blending techniques, namely (b) Alpha blending with standard deviation $\sigma = 10$, (c) Laplacian pyramid blending with various numbers of pyramid levels, and Poisson Blending with "Normal Clone" (d) or "Mixed Clone" (e).*