VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

VELLORE INSTITUTE OF TECHNOLOGY, CHENNAI

PROJECT MENTOR:- Dr. Saraswathi D

Project: Image Quilting and Texture Synthesis

Subject Title: Digital Image Processing
Subject Code: SWE1010

Submitted by:

Raj Koyani-21MIS1017
Aneesh.V-21MIS1030

TABLE OF CONTENTS

Page No

Chapter 1

1. Abstract, Problem Statement & Introduction……. **01**

Chapter 2

2. Literature Survey……..... **06**

Chapter 3

3. Objective and Methodology…….. **09**

- Objective
- Methodology

Chapter 4

4. System Design……... **15**

- System Architecture
- Use Case diagram
- Sequence diagram
- Algorithms used

Chapter 5

5. Implementation and Design…….. **20**

- Implementation of LSB
- Results

Chapter 6

6. Hardware and Software requirement…… **29**

- Hardware requirement
- Software requirement

Conclusion

References

Chapter:1
Abstract

We present a simple image-based method of generating novel visual appearance in which a new image is synthesized by stitching together small patches of existing images. We call this process image quilting. First, we use quilting as a fast and very simple texture synthesis algorithm which produces surprisingly good results for a wide range of textures. Second, we extend the algorithm to perform texture transfer – rendering an object with a texture taken from a different object. More generally, we demonstrate how an image can be re-rendered in the style of a different image. The method works directly on the images and does not require 3D information.

Image Synthesis:

Image synthesis is a crucial problem in the field of digital image processing. It involves generating new images that are similar to a set of given images. This process is commonly used in various applications, such as computer graphics, image editing, and medical imaging. In recent years, deep learning-based approaches have shown promising results in image synthesis tasks. These approaches utilize neural networks to learn the statistical distribution of the given images and generate new images that follow the same distribution. This abstract gives an overview of image synthesis techniques and their applications in various fields.

Image Quilting:

Image quilting is a technique used in digital image processing for texture synthesis and image generation. It involves dividing a source image into small blocks and reassembling them to form a new image. The blocks are selected based on their similarity to the neighboring blocks, resulting in a visually coherent image. This technique has various applications, such as image compression, texture mapping, and computer graphics. This abstract provides an overview of image quilting techniques and their applications in digital image processing.

Introduction:

1.1 Steganography

In the past decade computer graphics experienced a wave of activity in the area of image-based rendering as researchers explored the idea of capturing samples of the real world as images and using them to synthesize novel views rather than recreating the entire physical world from scratch. This, in turn, fueled interest in image-based texture synthesis algorithms. Such an algorithm should be able to take a sample of texture and generate an unlimited amount of image data which, while not exactly like the original, will be perceived by humans to be the same texture. Furthermore, it would be useful to be able to transfer texture from one object to anther (e.g. the ability to cut and paste material properties on arbitrary objects). In this paper we present an extremely simple algorithm to address the texture synthesis problem. The main idea is to synthesize new texture by taking patches of existing texture and stitching them together in a consistent way. We then present a simple generalization of the method that can be used for texture transfer. In the past decade, computer graphics experienced a wave of activity in the area of image-based rendering as researchers explored the idea of capturing samples of the real world as images and using them to synthesize novel views rather than recreating the entire physical world from scratch. This, in turn, fueled interest in image-based texture synthesis algorithms. Such an algorithm should be able to take a sample of texture and generate an unlimited amount of image data which, while not exactly like the original, will be perceived by humans to be the same texture. Furthermore, it would be useful to be able to transfer texture from one object to another (e.g. the ability to cut and paste material properties on arbitrary objects). In this paper, we present an extremely simple algorithm to address the texture synthesis problem. The main idea is to synthesize new textures by taking patches of existing textures and stitching them together in a consistent way. We then present a simple generalization of the method that can be used for texture transfer.

Image quilting is a technique used for texture synthesis, which involves generating new textures by combining small patches of existing textures. It has several applications in computer graphics, computer vision, and image processing. Here are some of the applications of image quilting for texture synthesis:

Video games and virtual environments: Image quilting can be used to generate realistic textures for objects, terrains, and environments in video games and virtual reality applications. Developers can create diverse and visually appealing game worlds by synthesizing textures based on existing

samples.

Augmented reality: Image quilting can be employed in augmented reality applications to seamlessly blend virtual objects with real-world scenes. By synthesizing textures that match the surrounding environment, the virtual objects can appear more realistic and integrated into the scene.

Visual effects in movies and animations: Texture synthesis using image quilting is valuable in the creation of visual effects for movies and animations. It allows artists to generate detailed and coherent textures for various elements like characters, creatures, and environments, enhancing the overall visual quality.

Texture transfer and style transfer: Image quilting can be used to transfer the texture of one image onto another. This technique finds applications in image editing and artistic rendering. It enables the transfer of textures from a reference image to a target image, resulting in the target image having a similar texture to the reference image.

Image in painting and restoration: Image quilting can assist in image inpainting, which involves filling in missing or damaged regions in an image. By synthesizing texture from surrounding areas, image quilting can help restore the missing regions and make the in painted image visually coherent.

Problem statement:

- The task of image quilting involves generating visually coherent and high-quality texture synthesis by stitching together smaller patches from a given set of sample images. However, existing image quilting algorithms face challenges in efficiently and accurately selecting appropriate patches, ensuring seamless blending, and handling complex textures. The problem statement aims to develop an improved image quilting technique that effectively addresses these challenges, resulting in realistic and visually appealing texture synthesis.
- Texture synthesis refers to the process of generating new textures that exhibit similar visual characteristics as a given input texture. Current texture synthesis methods encounter difficulties in capturing fine details, preserving global structure, and maintaining realism in the synthesized textures. The problem statement seeks to develop an advanced texture synthesis algorithm that overcomes these limitations, enabling the creation of high-quality synthetic textures that are visually consistent with the input texture and possess rich detail,

global coherence, and realistic appearance.

Texture Synthesis

- This aims to generate new textures that resemble a given source texture.
- It is often used to create larger textures from a small sample or to generate variations of existing textures.
- The process involves analyzing the statistical properties of the source texture and then procedurally generating new pixels that match those properties.

Image Quilting

- Image quilting and texture synthesis are techniques used in computer graphics and computer vision to generate realistic textures or fill in missing regions in images.
- The process involves selecting patches from a source texture and placing them in a target region, while ensuring seamless blending with neighboring patches.
- Image quilting involves stitching together small patches of textures to create a larger texture. It is often used for texture synthesis and texture transfer tasks.
- The objective is to create a visually appealing and creates a image that displays high resolution.

Process of Image Quilting

- Patch selection: Randomly select patches from the source texture that are similar in appearance to the target region.
- Overlapping: Place the selected patches in the target region, allowing for some overlap between patches.
- Overlap blending: Perform blending or feathering along the overlapping boundaries of the patches to ensure seamless transitions.
- Optimization: Adjust the placement and blending of patches to minimize visible artifacts such as discontinuities or repetitive patterns.

Efros &freeman:

The algorithm:

- Very simple
- Surprisingly good results
- Synthesis is easier than analysis.

1.2 Image quilting for texture synthesis principle

1.2.1 METHODOLOGY

Texture Synthesizing:

Sampling: The first step is to sample the given texture to obtain a set of texture patches.

Feature Extraction: The next step is to extract features from the texture patches using various techniques, such as wavelet transform, Gabor filters, or deep learning-based approaches.

Modeling: A statistical model is created based on the extracted features, such as a Markov Random Field (MRF) or a Gaussian Mixture Model (GMM).

Synthesis: The new texture is generated by sampling from the statistical model using various sampling techniques, such as Metropolis-Hastings or Gibbs sampling.

Image Quilting:

Patch Extraction: The first step is to extract patches from the input image, typically using a sliding window approach.

Overlapping Patches: The patches are then overlapped to create a larger image, which may result in visible seams.

Seams Elimination: The seams are eliminated by selecting the best patch at each overlap based on various criteria, such as texture similarity or color coherence.

Quilt Completion: The final step is to complete the image quilting process by filling in the remaining pixels using various techniques, such as Poisson blending or optimization-based approaches.

Texture Synthesizing:

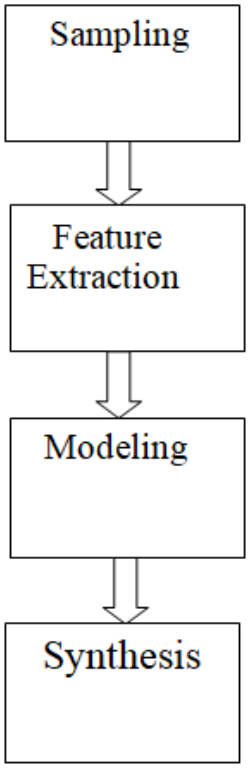
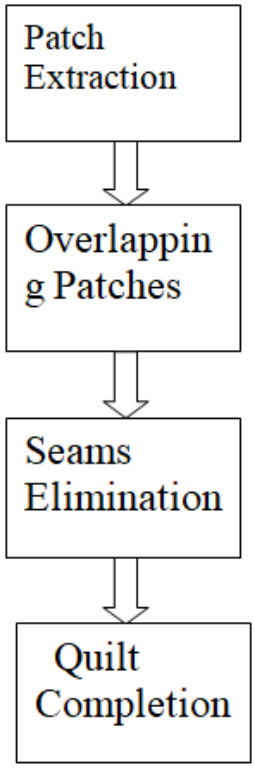


Image Quilting:



Chapter:2

Literature Survey:

J.T.O'Brien, D.S.Wickramanayake, E.A.Edirisinghe, H.E.Bez Image Quilting for Texture Synthesis: A Revisit & A Variation

(2003)

Based on the provided information, "Image Quilting for Texture Synthesis: A Revisit & A Variation" appears to be the title of a research paper or academic article. The study type is not explicitly mentioned in the given title, but it is likely to be a revisit and variation of the existing technique of image quilting for texture synthesis.

Alexei Efros and William T. Freeman Quilting for Texture Synthesis and Transfer

(2001-17)

As a research paper, the study is likely to present a detailed analysis of the proposed quilting technique, including its strengths, limitations, and potential applications. It may also compare the results with existing approaches or algorithms in the field. The authors might discuss the underlying concepts, theory, and methodologies employed in their study.

Afrin Fathima*1, Shoby Sunny*2 IMAGE QUILTING ALGORITHM FOR TEXTURE SYNTHESIS

(2021)

The sequential spot-based algorithm includes steps like initialization, patch selection, calculation of minimum error margins and the blending method along the margins. The approach for stitching a fresh patch in the sequentially created output texture to eliminate discontinuities as much as feasible is the method's key .

Prasad Tanaji Satpute1, Prof. Deipali Gore2“ Image Quilting in Steganography using Reversible

Texture Formation (2017)

Hardware based circuit is constructed by utilizing the features of covolution neural network and that circuit is also tested for fault free method which helped in smoothing of texture streaming.

Pepijn van Heiningen “Evaluation of Image Quilting algorithms”

(2014)

Neural networks can be ”trained” to solve problems that are difficult to solve by conventional computer algorithms.

A.A. Efros and T.K. LeungTexture synthesis by non-parametric sampling

(2010)

To gain a comprehensive understanding of the study's tools, methodology, and results, it would be necessary to refer to the research multiresolution strategy.

Vladimir V. Gurenko, Vyacheslav A. Eliseev,Maxim O. Usmanov Analysis of Texture Synthesis Algorithms in Computer Graphics

(2022)

Hardware based circuit is constructed by utilizing the features of neural network and that circuit is also tested for fault free method.

A survey on texture synthesis and its approaches“P. Malathi,N.V. Shibu,P. Malathi”

(2016)

Neural networks can be ”trained” to solve problems that are difficult to solve by conventional computer algorithms.

Wei Xin; Jiang Huawei; Yang Teng fei study on constraint-based texture synthesis
(2010)

This paper emphasizes on the idea of implementation of neural networks using FPGAs. The

resultant neural networks are modular, compact.

Gang Liu; Yann Gousseau; Gui-Song Xia Texture synthesis through convolutional neural networks and spectrum constraints

(2016)

The study focuses on texture synthesis and proposes a method that combines convolutional neural networks (CNNs) with spectrum constraints. CNNs are a type of deep learning architecture commonly used for image analysis and processing tasks. Spectrum constraints refer to imposing constraints on the frequency spectrum of synthesized textures**.**

Chapter 3

OBJECTIVES:

The objectives involved in this problem are:

- **Synthesizing textures:** The primary objective of synthesizing textures is to generate new textures that are similar to a given texture. This can be useful in various applications, such as creating realistic 3D models, generating high-resolution images, and designing computer games. The goal is to create textures that are visually appealing and can be used in various contexts.

- **Image quilting:** The objective of image quilting is to generate new images that have similar properties to a given image. This technique is particularly useful for generating large images from smaller ones, such as creating panoramic images or generating high-resolution images from low-resolution ones. The goal is to create an image that is visually coherent and seamless, without visible seams or artifacts.
  In general, the objective of both techniques is to create new digital content that is visually appealing and can be used in various contexts. These techniques have various applications in digital image processing, computer graphics, and other fields

Image Quilting:

Uses:

Texture Synthesis: Image quilting is used to synthesize textures by combining small patches from an input texture to generate a larger output texture with seamless transitions.

Texture Transfer: It can be used to transfer the texture of one image onto another, which is valuable in various applications like style transfer, image editing, and graphics design.

Texture Sampling: Image quilting can be used to efficiently sample textures from a source image for various purposes in computer graphics.

Benefits:

Seamless Textures: Image quilting ensures that the generated textures are seamless, meaning the transitions between patches are not visible to the human eye. This is important to create realistic and visually appealing results.

Local Adaptation: Image quilting considers the local features of patches during the synthesis process. As a result, it can better preserve the details and structures of the input texture in the output.

Scalability: Image quilting allows for generating textures of arbitrary sizes, making it suitable for various applications that require textures of different resolutions.
Real-Time Performance: Depending on the algorithm used, image quilting can be computationally efficient and can be performed in real-time, making it applicable in interactive applications and video games.

Texture Synthesis:

Uses:

Image and Video Compression: Texture synthesis can be used to compress images and videos more efficiently by representing textures with less data.

Computer Graphics: Texture synthesis is widely used in computer graphics for generating realistic textures on 3D models and scenes.

Virtual Reality and Augmented Reality: Texture synthesis plays a crucial role in creating immersive virtual and augmented reality experiences by providing detailed and realistic textures for virtual environments.

Procedural Content Generation: Texture synthesis can be employed in procedural content generation for video games and simulations to create diverse and visually appealing environments.

Benefits:

Realism: Texture synthesis algorithms aim to replicate the appearance of real-world textures, leading to visually convincing results.

Non-Periodic Patterns: Unlike traditional tiling methods, texture synthesis can generate non-periodic patterns that avoid the repetitive look often associated with regular tiling approaches.

Efficiency: Texture synthesis techniques can efficiently generate large amounts of texture data, reducing the storage and memory requirements in various applications.

Artistic Expression: Texture synthesis algorithms can be combined with other techniques to create unique and artistic visual effects in digital art and design.

Overall, image quilting and texture synthesis are essential tools for generating realistic and seamless textures, enabling numerous applications in computer graphics, image processing, and computer vision. Their ability to create visually convincing textures while preserving local features makes them valuable assets in various industries and creative fields.

Chapter:4
System Design

System Architecture

1. Designing a system for image quilting for texture synthesis involves planning the overall architecture, components, and workflows necessary to implement the technique effectively. Here's a high-level system design for image quilting:

2. User Interface: Design a user interface that allows users to interact with the system. This can include options to select input texture samples, set parameters (e.g., patch size, overlap amount), and visualize the synthesized texture.

3. Input Texture Samples: Provide a mechanism for users to input a set of texture samples that will be used as the basis for synthesis. These samples can be a collection of images or a single image with multiple texture regions.

4. Patch Extraction: Implement a module to extract patches from the input texture samples. This involves dividing the images into smaller patches, typically square or rectangular in shape, with a user-defined patch size.

5. Patch Selection: Develop a module to select patches from the input samples based on their similarity to the neighboring patches. This module compares patches using a distance measure (e.g., SSD, cross-correlation) and selects the best-matching patches.

6. Optimal Seam Finding: Implement a module to find the optimal seams along which the selected patches will be stitched together. This module utilizes dynamic programming or graph-cut algorithms to determine the minimum-cost seams that minimize visible boundaries or artifacts between patches.

7. Texture Synthesis: Develop a module that performs the iterative texture synthesis process. Start with an initial seed patch and iteratively expand the synthesized texture by adding new patches that seamlessly fit with the existing texture. Use the patch selection

and seam finding modules to guide the synthesis process.

8. Post-processing: Include a post-processing module to enhance the quality of the synthesized texture. This can involve noise reduction, color correction, or filtering techniques to improve visual appearance and coherence.

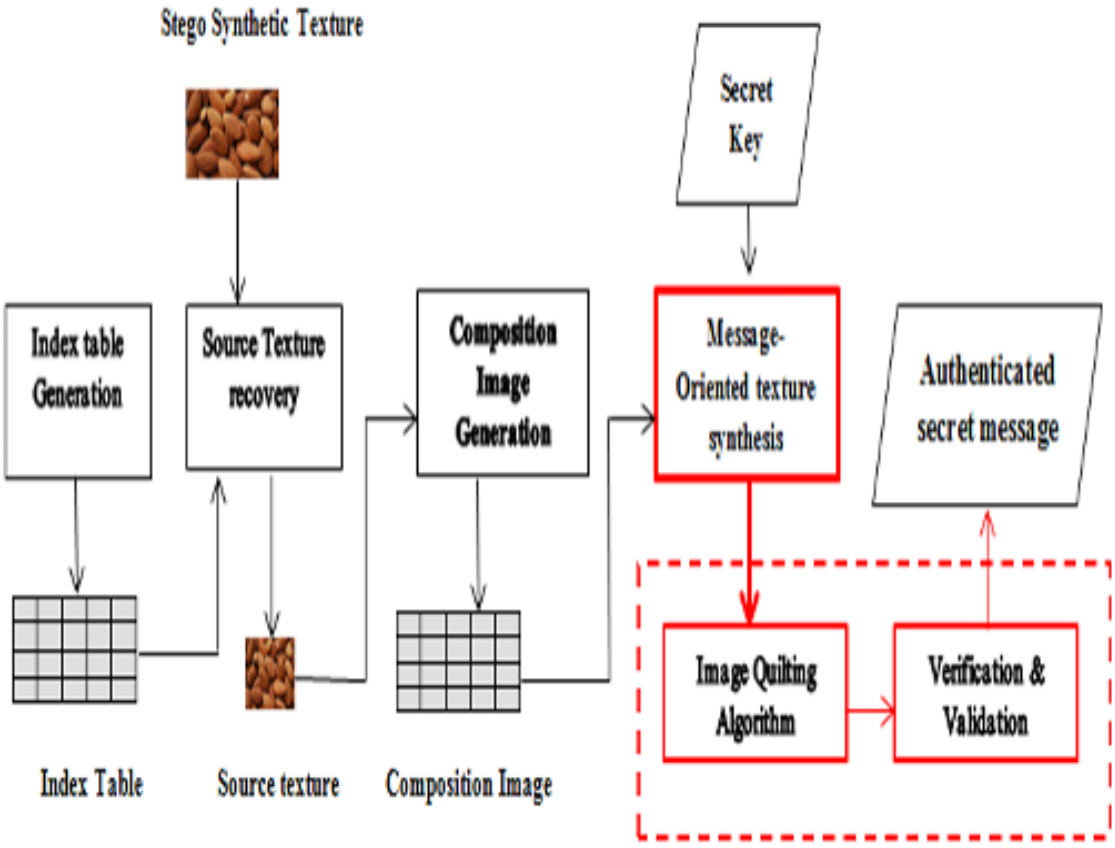
9. Output and Visualization: Design a mechanism to display the synthesized texture to the user. This can include visualizing the texture in real-time during the synthesis process or providing an output image once the synthesis is complete.

10. Performance Optimization: Implement optimization techniques to improve the efficiency and performance of the system. This can involve parallel processing, memory optimizations, or algorithmic improvements to reduce computation time.

11. Error Handling and Validation: Incorporate error handling mechanisms and validation checks to ensure the system operates smoothly. This can include handling invalid user inputs, detecting and resolving conflicts during patch selection or seam finding, and validating the synthesized texture against desired criteria.

12. Integration and Deployment: Consider how the system will be integrated into the target environment or application. This includes determining the programming languages, libraries, and frameworks to be used, as well as any dependencies or platform-specific considerations.

Use Case Diagram :



Algorithm Used

The Efros and Freeman algorithm, also known as the "Image Quilting for Texture Synthesis" algorithm, is a popular method for synthesizing textures using image quilting techniques. It was introduced by Alexei A. Efros and William T. Freeman in their seminal paper in 2001. Here's an overview of the algorithm:

1. Patch Extraction: The algorithm starts by extracting patches from the input texture samples. These patches serve as the basic units for synthesizing the new texture. The patch size is typically small and user-defined.

2. Overlapping Patches: The algorithm uses overlapping patches to ensure smooth transitions between neighboring patches in the synthesized texture. The amount of overlap is usually around 30-50% of the patch size.

3. Patch Selection: The algorithm selects patches from the input texture samples based on their similarity to the neighboring patches. It compares the patches using a distance measure, such as the sum of squared differences (SSD) or cross-correlation, to find the best matching patches.

4. Optimal Seam Finding: The algorithm determines the optimal seams along which the selected patches will be stitched together. It aims to minimize the visible boundaries or artifacts between patches. This is achieved by finding the path with the minimum cost through the overlap regions using dynamic programming.

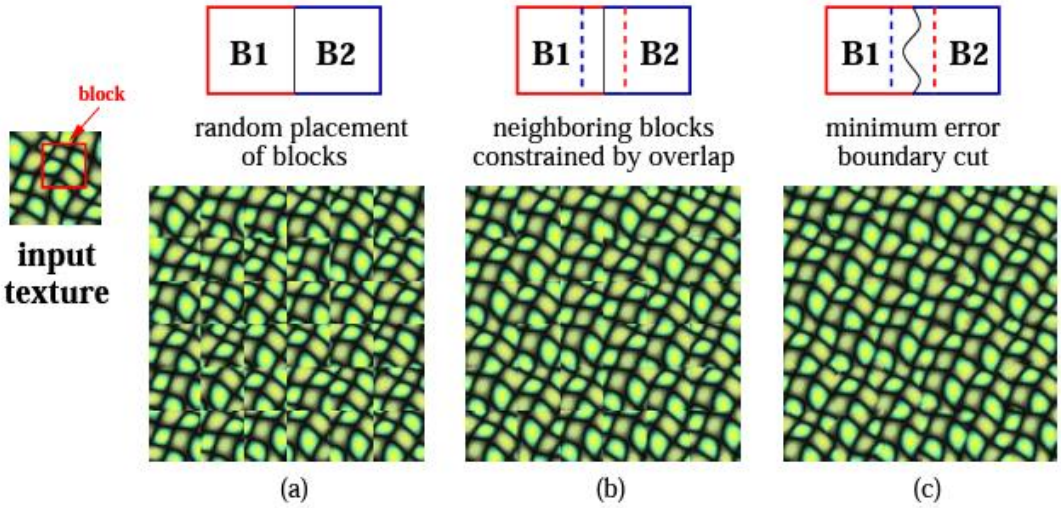
5. Texture Synthesis: The algorithm starts with an initial seed patch and iteratively expands the synthesized texture by adding new patches. The patches are selected based on their similarity to the existing texture, and the optimal seams are used to seamlessly blend them into the texture.

6. Texture Post-processing: After the synthesis process, post-processing steps can be applied to enhance the quality of the synthesized texture. This may include noise reduction, color correction, or filtering techniques to improve the visual appearance.

The Efros and Freeman algorithm is known for its ability to generate visually appealing textures with coherent structures and details. It effectively combines patch selection, optimal seam finding, and iterative texture synthesis to create seamless textures. The algorithm can be extended and optimized with additional techniques, such as spatial hashing or texture annealing, to improve its efficiency and performance.

Implementing the Efros and Freeman algorithm involves coding the steps mentioned above, including patch extraction, patch selection based on similarity measures, optimal seam finding using dynamic programming, iterative texture synthesis, and optional post-processing steps. The algorithm can be implemented using various programming languages and image processing libraries, depending on the developer's preference and requirements.

```
procedure bubbleSort(arr: array of elements)
    n = length(arr)
    for i = 0 to n-1
        // Set a flag to check if any swap occurred in this pass
        swapped = false

        for j = 0 to n-i-1
            // Compare adjacent elements
            if arr[j] > arr[j+1]
                // Swap elements if they are in the wrong order
                swap(arr[j], arr[j+1])
                swapped = true

        // If no swaps occurred in this pass, the array is already sorted
        if not swapped
            break
    end for
end procedure
```


block
input texture
B1
B2
random placement of blocks
B1
B2
neighboring blocks constrained by overlap
B1
B2
minimum error boundary cut
(a)
(b)
(c)

Chapter 5

Implementation and Results

Implementation of Image Quilting for Texture Synthesis

Implementing image quilting for texture synthesis involves several steps and considerations. Here is a high-level overview of the implementation process:

Patch Extraction: The first step is to extract patches from the input texture samples. These patches serve as the basic building blocks for synthesizing the new texture. The size of the patches can vary depending on the desired level of detail.

Overlapping Patches: Image quilting involves overlapping patches to ensure seamless blending between neighboring patches. The amount of overlap determines the smoothness and coherence of the synthesized texture. Typically, the overlap is around 30-50% of the patch size.

Distance Measure: Choose an appropriate distance measure to compare patches and determine their similarity. Common distance measures include Euclidean distance, SSD (sum of squared differences), or cross-correlation. The distance measure helps identify the best matching patches for the synthesis process.

Patch Selection: Select patches from the input texture samples based on their similarity to the neighboring patches. The goal is to find the best matching patches that provide a smooth transition between neighboring regions.

Optimal Seam Finding: Determine the optimal seams where the selected patches will be stitched together. This process involves finding the seams that minimize the visible boundaries or artifacts between patches. Algorithms such as dynamic programming or graph cuts can be used for seam finding.

Texture Synthesis: Begin the texture synthesis process by starting with an initial seed patch. Then iteratively expand the synthesized texture by adding new patches that seamlessly fit with the existing texture. The process continues until the desired texture size is achieved.

Texture Post-processing: Perform any necessary post-processing steps to enhance the quality of the synthesized texture. This may involve noise reduction, color correction, or filtering to further refine the texture and improve its visual appearance.

Parameter Tuning: Fine-tune the parameters of the image quilting algorithm to achieve the desired results. Parameters such as patch size, overlap amount, distance measure weights, or patch selection strategies can be adjusted to influence the texture synthesis process.

Optimization: Implement optimization techniques to improve the efficiency and performance of the image quilting algorithm. This may involve utilizing parallel processing, optimizing data structures, or implementing caching mechanisms to speed up the synthesis process.

Validation and Testing: Finally, validate the synthesized textures by comparing them with the original input samples and assessing their visual quality. Perform thorough testing to ensure the algorithm works correctly in different scenarios and handles various texture types effectively.

Screenshots:

Texture Synthesis:

```
import numpy as np
import matplotlib.pyplot as plt
from skimage.util import view_as_windows
from skimage.color import rgb2gray, rgb2lab, lab2rgb

def texture_synthesis(image1, image2, output_shape, patch_size):
    h, w, _ = output_shape  # Updated to handle color images
    output_texture = np.zeros(output_shape)

    # Generate patches from both images
    patches1 = view_as_windows(image1, (patch_size, patch_size, 3), step=patch_size)
    patches2 = view_as_windows(image2, (patch_size, patch_size, 3), step=patch_size)

    # Randomly select patches from both images and stitch them in the output texture
    for y in range(0, h, patch_size):
        for x in range(0, w, patch_size):
            if np.random.rand() < 0.5:
                patch = patches1[np.random.randint(patches1.shape[0]), np.random.randint(patches1.shape[1])]
            else:
                patch = patches2[np.random.randint(patches2.shape[0]), np.random.randint(patches2.shape[1])]

            output_texture[y:y + patch_size, x:x + patch_size] = patch

    return output_texture

# Example usage
image1 = plt.imread('D:/dip/3.jpg')
image2 = plt.imread('D:/dip/256x256_Dissolve_Noise_Texture.png')
output_shape = (512, 512, 3)  # Updated to specify color output shape
patch_size = 16

# Resize the input images to 256x256
image1 = image1[:256, :256]
image2 = image2[:256, :256]

output_texture = texture_synthesis(image1, image2, output_shape, patch_size)

# Display the synthesized texture
plt.imshow(output_texture)
plt.axis('off')
plt.show()
```

Image Quilting:

```
import numpy as np
import math
from skimage import io, util
import heapq

def randomPatch(texture, patchLength):
    h, w, _ = texture.shape
    i = np.random.randint(h - patchLength)
    j = np.random.randint(w - patchLength)

    return texture[i:i+patchLength, j:j+patchLength]

def L2OverlapDiff(patch, patchLength, overlap, res, y, x):
    error = 0

    if x > 0:
        left = patch[:, :overlap] - res[y:y+patchLength, x:x+overlap]
        error += np.sum(left**2)

    if y > 0:
        up   = patch[:overlap, :] - res[y:y+overlap, x:x+patchLength]
        error += np.sum(up**2)

    if x > 0 and y > 0:
        corner = patch[:overlap, :overlap] - res[y:y+overlap, x:x+overlap]
        error -= np.sum(corner**2)

    return error


def randomBestPatch(texture, patchLength, overlap, res, y, x):
    h, w, _ = texture.shape
    errors = np.zeros((h - patchLength, w - patchLength))

    for i in range(h - patchLength):
        for j in range(w - patchLength):
            patch = texture[i:i+patchLength, j:j+patchLength]
            e = L2OverlapDiff(patch, patchLength, overlap, res, y, x)
            errors[i, j] = e

    i, j = np.unravel_index(np.argmin(errors), errors.shape)
    return texture[i:i+patchLength, j:j+patchLength]


def minCutPath(errors):
    # dijkstra's algorithm vertical
    pq = [(error, [i]) for i, error in enumerate(errors[0])]
    heapq.heapify(pq)
```

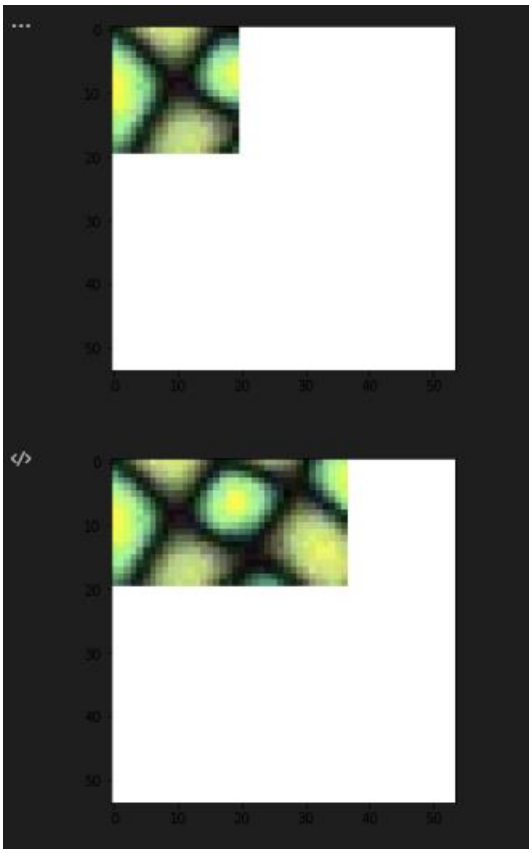
```
    h, w = errors.shape
    seen = set()

    while pq:
        error, path = heapq.heappop(pq)
        curDepth = len(path)
        curIndex = path[-1]

        if curDepth == h:
            return path

        for delta in -1, 0, 1:
            nextIndex = curIndex + delta

            if 0 <= nextIndex < w:
                if (curDepth, nextIndex) not in seen:
                    cumError = error + errors[curDepth, nextIndex]
                    heapq.heappush(pq, (cumError, path + [nextIndex]))
                    seen.add((curDepth, nextIndex))

def minCutPath2(errors):
    # dynamic programming, unused
    errors = np.pad(errors, [(0, 0), (1, 1)],
                    mode='constant',
                    constant_values=np.inf)

    cumError = errors[0].copy()
    paths = np.zeros_like(errors, dtype=int)

    for i in range(1, len(errors)):
        M = cumError
        L = np.roll(M, 1)
        R = np.roll(M, -1)

        # optimize with np.choose?
        cumError = np.min((L, M, R), axis=0) + errors[i]
        paths[i] = np.argmin((L, M, R), axis=0)

    paths -= 1

    minCutPath = [np.argmin(cumError)]
    for i in reversed(range(1, len(errors))):
        minCutPath.append(minCutPath[-1] + paths[i][minCutPath[-1]])

    return map(lambda x: x - 1, reversed(minCutPath))

def minCutPatch(patch, patchLength, overlap, res, y, x):
    patch = patch.copy()
    dy, dx, _ = patch.shape
    minCut = np.zeros_like(patch, dtype=bool)

    if x > 0:
        left = patch[:, :overlap] - res[y:y+dy, x:x+overlap]
```

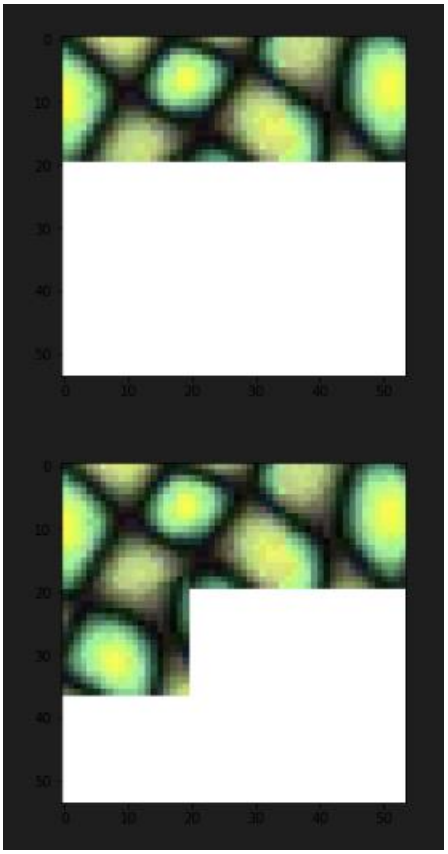
```
        leftL2 = np.sum(left**2, axis=2)
        for i, j in enumerate(minCutPath(leftL2)):
            minCut[i, :j] = True

    if y > 0:
        up = patch[:overlap, :] - res[y:y+overlap, x:x+dx]
        upL2 = np.sum(up**2, axis=2)
        for j, i in enumerate(minCutPath(upL2.T)):
            minCut[:i, j] = True

    np.copyto(patch, res[y:y+dy, x:x+dx], where=minCut)

    return patch

def quilt(texture, patchLength, numPatches, mode="cut", sequence=False):
    texture = util.img_as_float(texture)

    overlap = patchLength // 6
    numPatchesHigh, numPatchesWide = numPatches

    h = (numPatchesHigh * patchLength) - (numPatchesHigh - 1) * overlap
    w = (numPatchesWide * patchLength) - (numPatchesWide - 1) * overlap

    res = np.zeros((h, w, texture.shape[2]))

    for i in range(numPatchesHigh):
        for j in range(numPatchesWide):
            y = i * (patchLength - overlap)
            x = j * (patchLength - overlap)

            if i == 0 and j == 0 or mode == "random":
                patch = randomPatch(texture, patchLength)
            elif mode == "best":
                patch = randomBestPatch(texture, patchLength, overlap, res, y, x)
            elif mode == "cut":
                patch = randomBestPatch(texture, patchLength, overlap, res, y, x)
                patch = minCutPatch(patch, patchLength, overlap, res, y, x)

            res[y:y+patchLength, x:x+patchLength] = patch

            if sequence:
                io.imshow(res)
                io.show()

    return res

def quiltSize(texture, patchLength, shape, mode="cut"):
    overlap = patchLength // 6
    h, w = shape

    numPatchesHigh = math.ceil((h - patchLength) / (patchLength - overlap)) + 1 or 1
    numPatchesWide = math.ceil((w - patchLength) / (patchLength - overlap)) + 1 or 1
    res = quilt(texture, patchLength, (numPatchesHigh, numPatchesWide), mode)
```

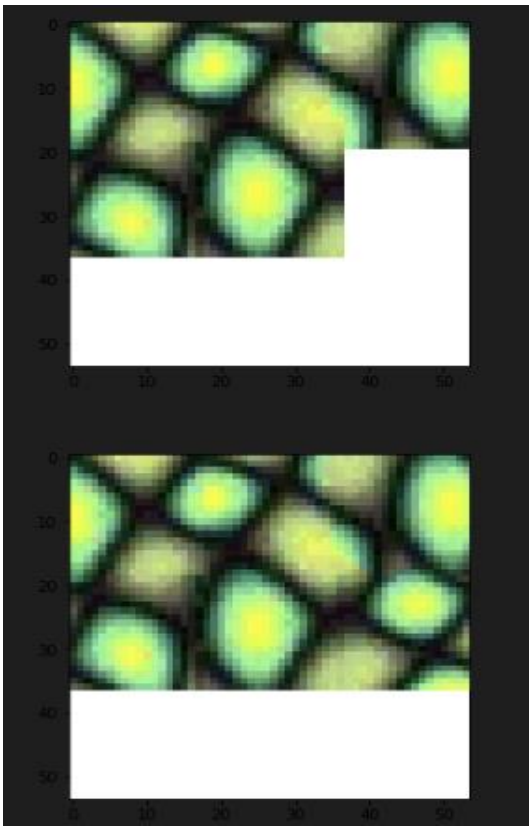
```
    return res[:h, :w]
s = "https://raw.githubusercontent.com/axu2/image-quilting/master/"

texture = io.imread(s + "test.png")
io.imshow(texture)
io.show()

io.imshow(quilt(texture, 25, (6, 6), "random"))
io.show()

io.imshow(quilt(texture, 25, (6, 6), "best"))
io.show()

io.imshow(quilt(texture, 20, (6, 6), "cut"))
io.show()
```

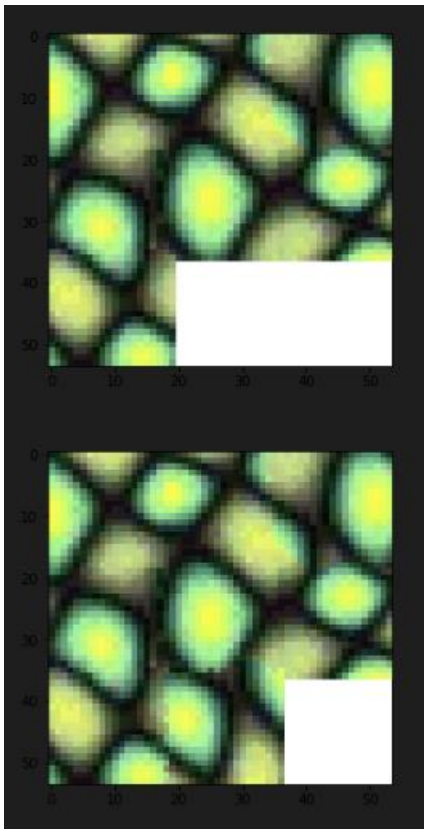
Results

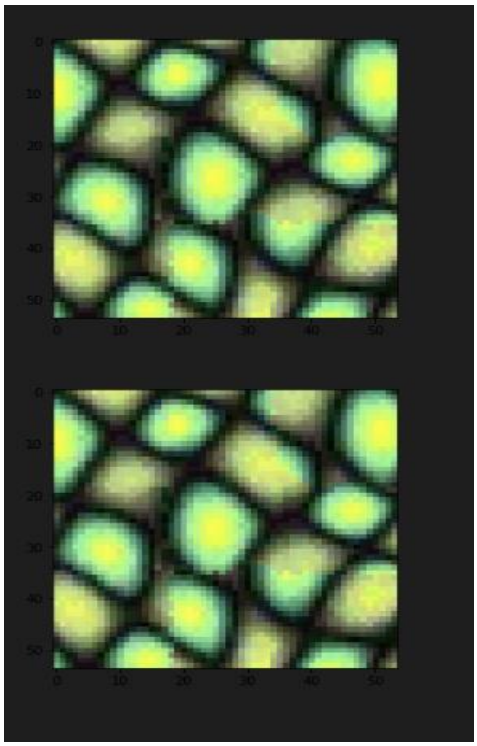
Image quilting:

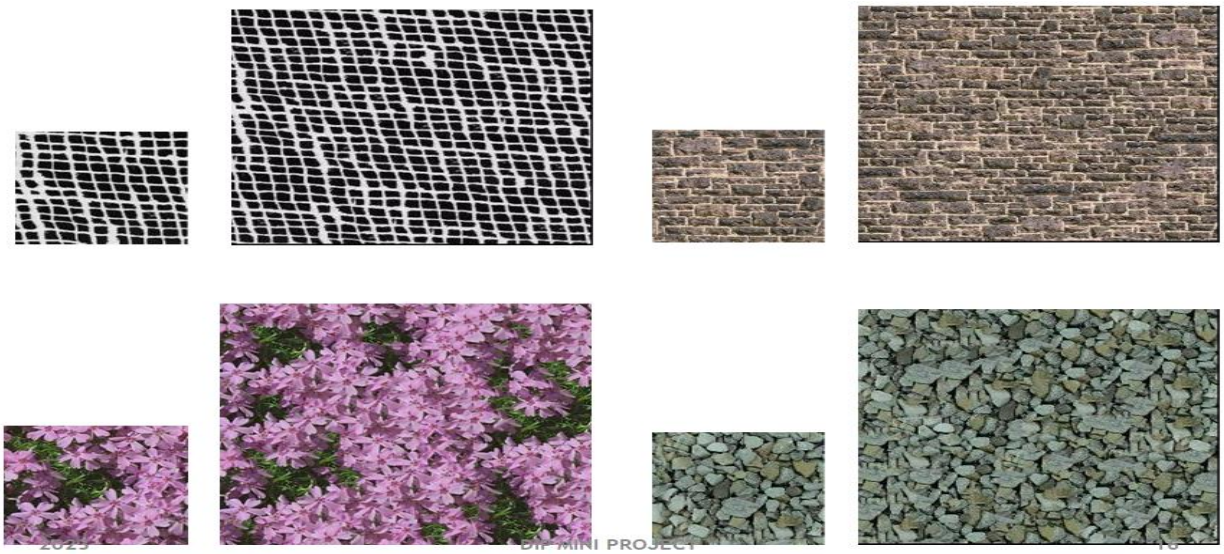






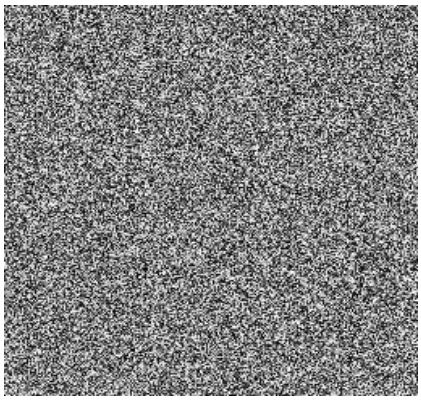




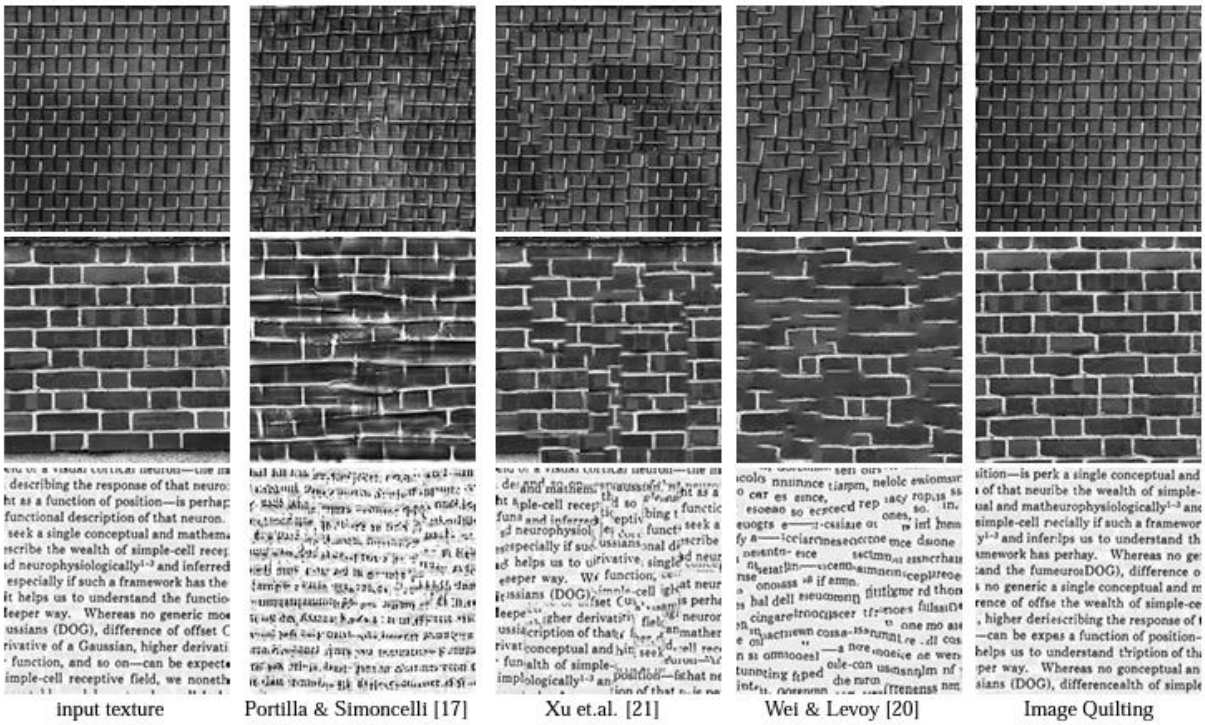


Texture Synthesis:

Input:



Ouput:



input texture | Portilla & Simoncelli [17] | Xu et.al. [21] | Wei & Levoy [20] | Image Quilting

Figure 6: Comparison of various texture synthesis methods on structured textures. Our results are virtually the same as Efros & Leung [6] (not shown) but at a much smaller computational cost.

Chapter: 6

Non-Functional Requirements

- Image quilting is a technique that revolutionizes the process of texture synthesis by combining small patches of existing textures to create new ones. It has gained significant attention in computer graphics, computer vision, and image processing due to its ability to generate visually appealing and realistic textures. The technique offers several advantages and applications in different fields.

- In the realm of video games and virtual environments, image quilting allows developers to generate diverse and visually captivating textures for objects, terrains, and environments. By synthesizing textures based on existing samples, they can create immersive game worlds that enhance the overall gaming experience.

- Computational Efficiency: Image quilting should be computationally efficient to ensure real-time or near-real-time performance, especially in applications that require interactive user interaction, such as video games or augmented reality. The algorithm should be optimized to reduce processing time and memory usage.

- Scalability: The image quilting technique should be able to handle large-scale textures and work effectively with high-resolution images. It should be capable of synthesizing textures of varying sizes without a significant impact on performance or quality.

- Quality and Realism: The synthesized textures should exhibit high-quality and realistic characteristics. The technique should strive to generate textures that seamlessly blend with the existing environment or target image, avoiding visible artifacts, inconsistencies, or obvious repetition patterns.

- Flexibility and Adaptability: The image quilting algorithm should be adaptable to different texture styles, types, and input sources. It should be able to synthesize textures that match specific requirements, such as the desired level of detail, color distribution, or visual appearance.

- User Control: In certain applications, it may be essential to provide users with control over the texture synthesis process. The technique should support user-defined constraints or parameters that allow users to guide the synthesis process according to their preferences**.**

System Requirements

SOFTWARE REQUIREMENTS

OS: Windows 10 /11.

HARDWARE REQUIREMENTS

- Digital microscope
- AMD /INTEL Pentium Processor
- 8GBRAM

Conclusion

Image quilting is a technique that revolutionizes the process of texture synthesis by combining small patches of existing textures to create new ones. It has gained significant attention in computer graphics, computer vision, and image processing due to its ability to generate visually appealing and realistic textures. The technique offers several advantages and applications in different fields.

In the realm of video games and virtual environments, image quilting allows developers to generate diverse and visually captivating textures for objects, terrains, and environments. By synthesizing textures based on existing samples, they can create immersive game worlds that enhance the overall gaming experience.

Augmented reality applications also benefit from image quilting. By synthesizing textures that match the surrounding real-world environment, virtual objects seamlessly blend with the scene, appearing more realistic and integrated. This enhances the illusion of virtual objects coexisting with the real world.

In the domain of movies and animations, image quilting plays a crucial role in creating visual effects. Artists can use this technique to generate detailed and coherent textures for various elements like characters, creatures, and environments. The result is visually stunning effects that captivate the audience and enhance the overall visual quality of the production.

Image quilting finds applications in image editing and artistic rendering as well. It enables texture transfer and style transfer, allowing the texture of one image to be applied to another. This facilitates the creation of artistic effects and enables the generation of images with textures inspired by specific references.

Furthermore, image quilting assists in image inpainting and restoration. When there are missing or damaged regions in an image, the technique can synthesize textures from the surrounding areas to fill in the gaps. This helps in restoring the image's visual coherence and produces a more complete representation.

Lastly, image quilting is valuable in the field of 3D modeling. It enables the generation of textures for 3D models, such as buildings, vehicles, or objects. By synthesizing textures based on existing

samples, it allows the creation of realistic and visually appealing textures that enhance the overall quality of the 3D models.

Overall, image quilting for texture synthesis offers flexibility, versatility, and realistic results across various applications. Its ability to seamlessly blend textures, transfer styles, inpaint missing regions, and generate visually appealing textures makes it an invaluable tool in fields that rely on high-quality texture synthesis. Through its continued development and utilization, image quilting contributes to advancements in computer graphics, computer vision, and image processing, pushing the boundaries of what is possible in texture.

References

[1]Efros, Alexei A., and William T. Freeman. "Image quilting for texture synthesis and transfer." In Proceedings of the 28th annual conference on Computer graphics and interactive techniques, pp. 341-346. 2001.

[2]Song, S., Yu, H., Miao, Z., et al. “Domain Adaptation for Convolutional Neural Networks- Based Remote Sensing Scene Classification,” IEEE Geoscience and Remote Sensing Letters 16.8 (2019): 1324-1328.

[3]Hoppe, Hugues. (1997). View-dependent refinement of progressive meshes.. SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques. 189-198. 10.1145/237170.237216.

[4]Abderrahim, Zeineb & Bouhlel, Med. (2021). Compression and Visualization Interactive of 3D Mesh. International Journal of Applied Mathematics and Informatics. 15. 85-92. 10.46300/91014.2021.15.14.]

[5]Payan, Frédéric & Antonini, Marc. (2005). An Efficient Bit Allocation for Compressing Normal Meshes with an Error-driven Quantization. Computer Aided Geometric Design. 22. 466-486. 10.1016/j.cagd.2005.04.001.

[6]Lee, Ho & Lavoué, Guillaume & Dupont, Florent. (2012). Rate-distortion optimization for progressive compression of 3D mesh with color attributes. The Visual Computer. 28. 137-153. 10.1007/s00371-011-0602-y.

[7]Lee, A. & Sweldens, W. & Cowsar, Lawrence & Dobkin, D.. (1998). MAPS: Multiresolution Adaptive parameterization of Surfaces. ACM SIGGRAPH.