AVPR Oral Presentation: Presenter's Notes

Subject: Multiscale Texture Synthesis
Student: Theofanis Petkos

This document is used to gather all the information I would like to present during the Oral Presentation exercise. I have chosen as paper the multiscale texture synthesis. Additionally, as the concept of texture synthesis was new to me I have listed as references other sources I looked into during my preparations for the presentation.

Paper Overview

From an 1000 feet overview of the concepts, we can identify the following key points discussed in the contents of the paper:

- The example-based texture synthesis, or more precisely the single scale texture synthesis in comparison to the multiscale approach.
- The multi scale texture synthesis use case, with the suggested approach that allows us to take advantage of features like infinite zoom, inconsistency corrections etc.

Example-Based Texture Synthesis

(Comment on image 8) The example based algorithms usually take as input an exemplar image and create a coherent, non-periodic texture, using a small portion (an exemplar) of the input image. They follow a hierarchical approach.

However this behaviour comes with specific limitations. Sometimes, the input image has specific resolution that doesn't allow to capture texture features that might are larger than the exemplar and as a result they will be missed. That applies also in the real world where many images involve features features at widely varying spatial scales.

Multiscale Texture Synthesis: Introduction

The suggested approach of multiscale texture synthesis is taking an input of an exemplar graph (link to image 1) of n samples and can generate a brand new texture (link to image 2) using multiple and different scales of the given images.

On the other hand, for cases with one image we use loops to iterate over the different scales of one image to produce the final coherent texture.

Multiscale Texture Synthesis: Key Concepts

- *Exemplar Graph (Image 3):* is a reflexive, directed, weighted graph and its vertices are exemplars. We have a source exemplar pointing to destination exemplar(s). In the contents of the paper we consider source/destination exemplars on the graph to have scaling relations only and loops (same source/destination exemplar) are allowed.
- *Gaussian Stacks:* Are associated to each exemplar, with each stack level being an image mxm obtained by filtering with a Gaussian kernel of radius 2^{L-k}, with L being the number of levels.
- *Admissible Candidates (Image 4):* The stack levels of exemplars that sharing equivalent scale. For each exemplar they are calculated with:

$$\mathcal{A}(E_k^i) = \{E_l^j \mid \exists (i, j, k-l) \in \mathbb{E},\ 0 \le l < L\},$$

Multiscale Texture Synthesis: Overview

Using an exemplar graph instead of a single exemplar introduces a lot of potential in the texture synthesis process. The current Paper follows a hierarchical approach on the input/output analysis. Each exemplar image is represented by coordinates (and not the usual color value). Each synthesized pixel can point to different exemplars from the exemplar graph, as for neighboour pixels we take an admissible candidate.

Eeach stack level *S* passes through the processes of upsampling and Jitter. Upsampling is used to refine the coordinates of S to point to the next-finer stack level of the Gaussian stack. This is done with:

$$S_t\left[2p + \Delta + \left(\tfrac{1}{2}, \tfrac{1}{2}\right)\right] := \left(i, k+1, u + \lfloor h_k \Delta \rfloor \pmod m\right),$$

$$\text{where } \Delta \in \left\{\left(\pm\tfrac{1}{2}, \pm\tfrac{1}{2}\right)\right\}.$$

Where S_t is the stack level we want to upsample. On the other hand, jittering is introduced to ensure randomness over the synthesized pixels. Jittering step is formed with:

$$S_t[p] := \left(i, k, u + J_t(p) \pmod m\right), \text{ where } J_t(p) = \left\lfloor h_k \mathcal{H}(p) \rho_t + \left(\tfrac{1}{2}, \tfrac{1}{2}\right)\right\rfloor.$$

where the original coordinates of S_t[p] are *(i, k, u)*.

Now after we have upsampled and jittered *S* we proceed to corrections so the synthesized texture is smooth. In a sense we look over the admissible candidates to re-arrange the pixels with the best matching neighbours.

In order to achieve an accelerated matching the *k-coherence* algorithm is used. So given the graph, for each stack level texel $E_k^j[u]$ we identify $E_k^j[u]$ the texels that minimize the error functional:

$$\sum_{\delta \in \{-2 \ldots +2\}^2} \left\| E_k^i[u + \delta h_k] - E_l^j[v + \delta h_l] \right\|^2$$

Normally for this process the K is 2 an set a candidate set: $\tilde{\mathcal{A}}(E_k^i[u])$

From this set we get the minimum of the precomputed candidates: $\bigcup_{d\in\{-1\ldots1\}^2} \tilde{\mathcal{A}}(*S_l[p+d])$

Finally to align the neighbourhoods in source and destination we replace:
$E_l^j[v+\delta h_l]$ (with) $E_l^j[v+(\delta-d)h_l]$

Multiscale Texture Synthesis: Inconsistency Correction

(Image 4) As the exemplar graph usage tries to welcome data from different sources is possible that inconsistencies on the scales or even the colors between the graph and the synthesized texture can occur. For example, we may want to get a synthesized neighbour of a 5x5 region with color from a candidate in grayscale. Dealing with such inconsistencies is also computational expensive.

(Image 5) In an overview such inconsistencies are being handled with appearance transfer functions *r: RGB -> RGB* which tries to capture the overall change in the appearance between the source exemplar and the destination exemplar. We focus on interexemplar changes as for self-exemplar cases we don't have such inconsistencies.

In the analysis process we need a function that will show us the transition that happens in the current step. To narrow down the number of possible destinations we use the candidates we discussed in the matching process. Now, having in mind that we use 5x5 neighborhoods for the matching and *r(c) = c + b* as the function, *r* is optimized:

$$\mathbf{b} = \frac{1}{25}\sum_{\delta\in\{-2\ldots+2\}^2}\left(E_k^i[u+\delta h_k]-E_l^j[v+\delta h_l]\right)$$

The synthesis process tries to inverse the color space of the synthesized texture back to the one used in the analysis process and applied to the *transformed neighborhood,* the one that is matched as the neighborhood of the current step. This is done by:

$$\sum_{\delta\in\{-2\ldots+2\}^2}\left\|\mathbf{R}_p^{-1}(\mathbf{R}_{p+\delta}(*S_t[p+\delta]))-\mathbf{r}_v\left(E_l^j[v+\delta h_l]\right)\right\|^2$$

(comment on image 5) so upon finding the best match neighborhood, we invert its color space to the previous one, we make the correction and then we synthesize.

Optimizations

Three optimizations are applied in the context of the paper combined with the usage of GPU for better performance:

- *Superexemplar*: (comment on Image 6) A super exemplar mechanism tries to:
  - Unroll the graph loops into trees with root E^0.
  - Expand each graph vertex into a chain of vertices.
  - Link edges to stack level pairs.
- *PCA Projection*: The PCA is the mechanism used for the matching of the neighborhoods described in the previous slides. An important note is that in the PCA process all levels are considered for the synthesis. Here PCA uses the

superexemplar extended tree output to identify all those levels and find the next neighbor. To avoid the inconsistencies mentioned earlier the transformations of the target neighborhoods are applied prior to the projection to PCA.

- *Texture Packing:*
    - One RGB texture for all the stack levels.
    - Three RGBA textures for the candidate neighborhoods (PCA-reduced)
    - One RBGA 16-bit texture to store the links for the transfer functions.

Results

- (*Comment Image 2)* **Gigapixel Textures:** We can see that we can quickly get an 16k x 16k map texture from a set of eleven only exemplars with size 256x256.

- *(Comment Image 7)* **Coherent Infinite Zooms:** Through the usage of this multiscale texture synthesis approach we are able to apply coherent zooms

- *(Comment Image 5)* **Inconsistency Correction:** It was explained earlier and it remains a useful feature that this approach adds on the table.

- **Synthesis performance:** It was explained earlier and it remains a useful feature that this approach adds on the table.

Resources

Image 1: Input exemplar graph

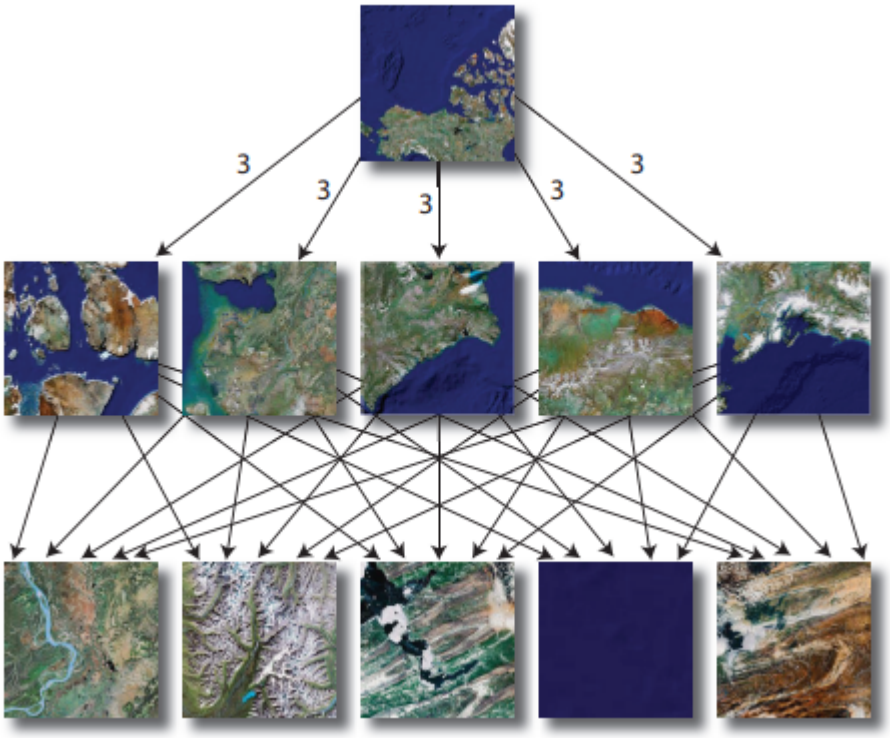


Image 2: Output Multiscale Texture

Image 3: Simple exemplar graph

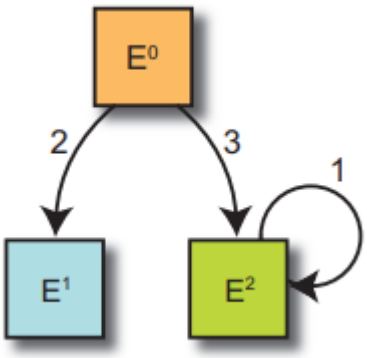


Image 4: Graph Inconsistencies

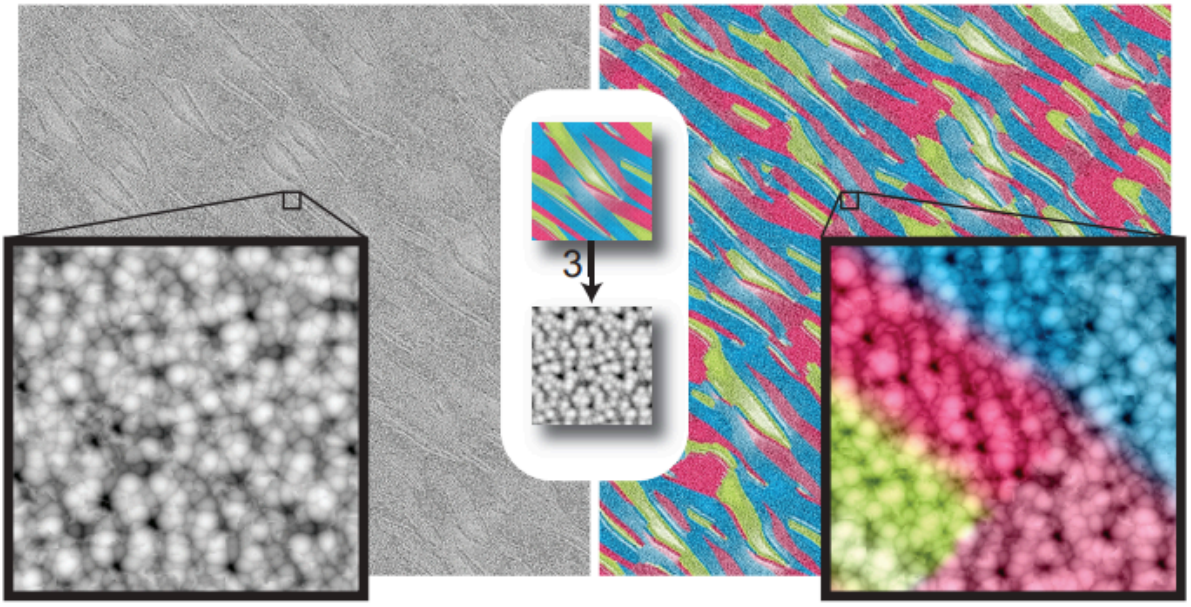


Image 5: Transfer Functions

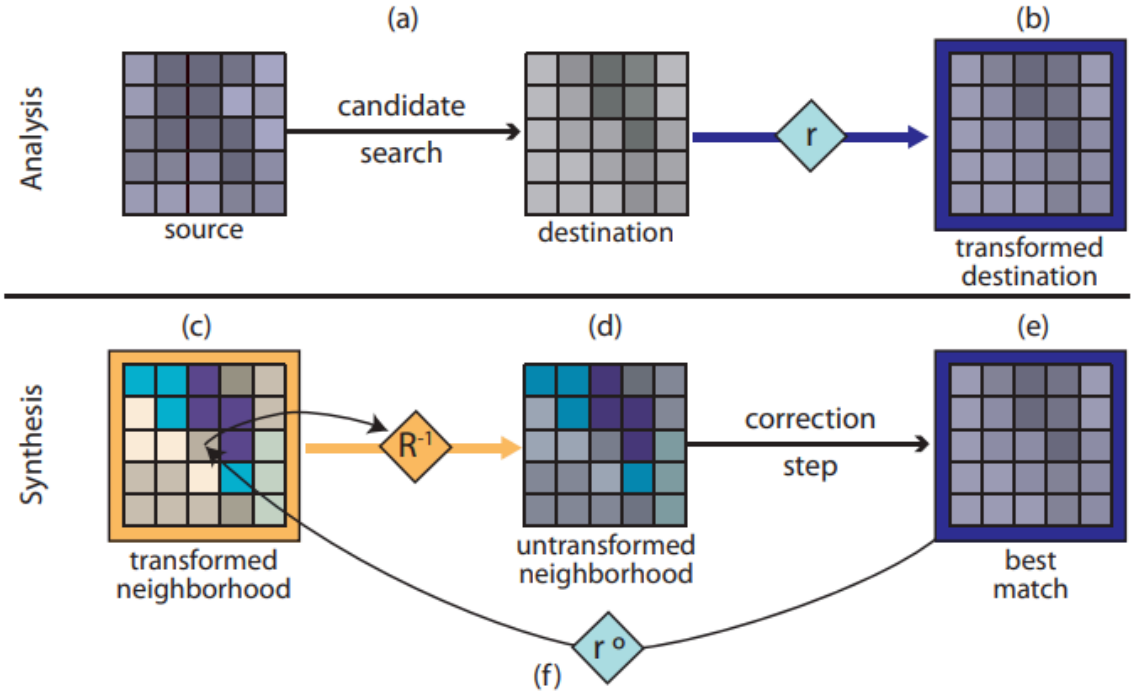


Image 6: Superexemplar

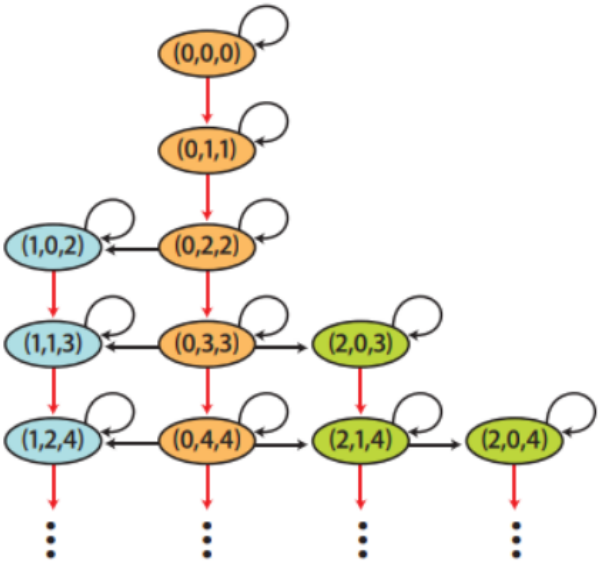


Image 7: Pixel scale

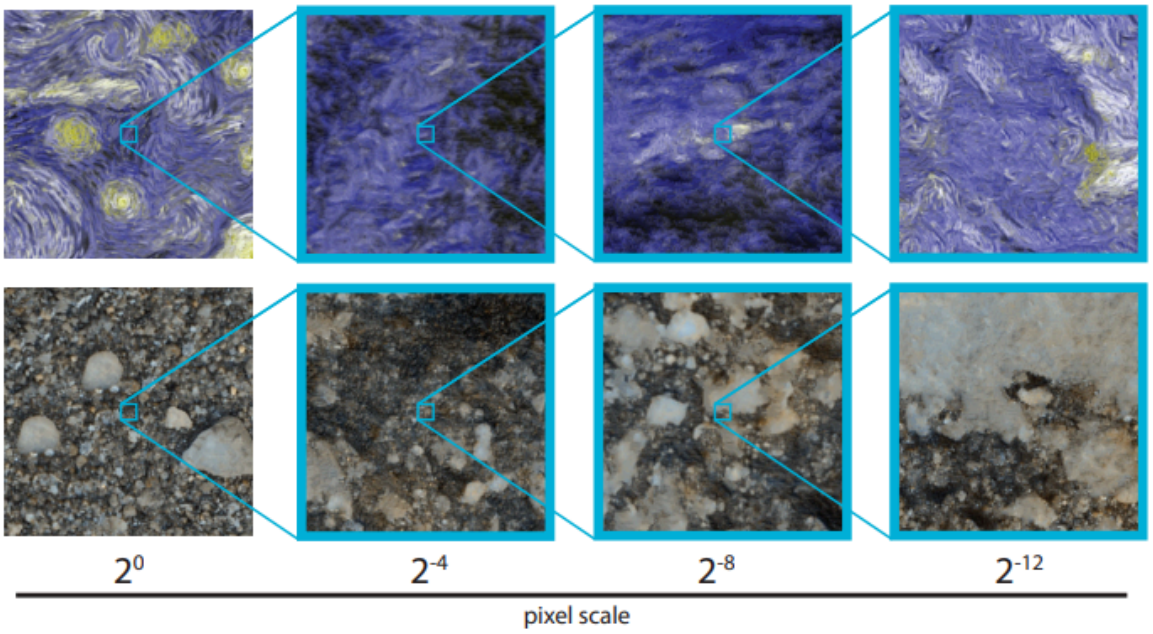
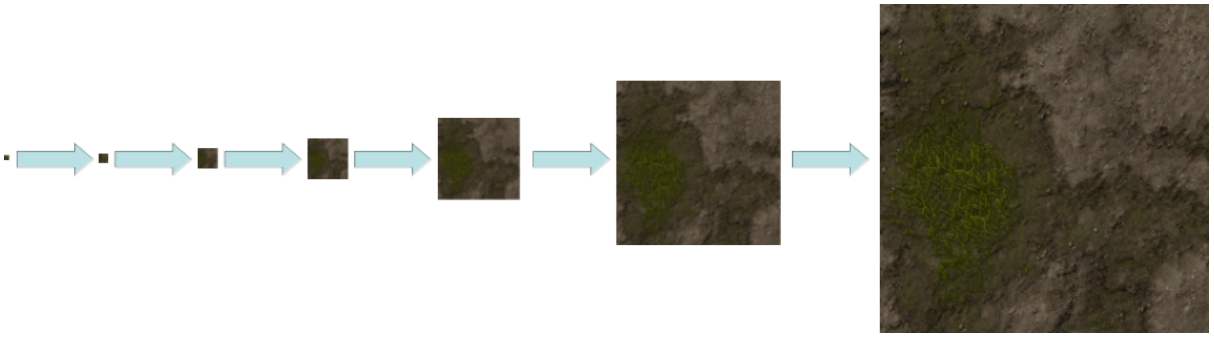


Image 8 (from the slides): Hierarchical approach



References

[1] C.Han et al., *Multiscale Texture Synthesis*,

[2] L. Raad et al., *A survey of exemplar-based texture synthesis*, 2018.

[3] C. Aguerrebere, *Exemplar-Based Texture Synthesis: the Efros–Leung Algorithm*

[4] X. Snelgrove, *High-Resolution Multi-Scale Neural Texture Synthesis*, 2017