ABSTRACT

For this project, we reimplemented the image analogies framework, with applications to artistic filters, texture synthesis, texture transfer, and texture-by-numbers.

Index Terms— non-photorealistic rendering, texture synthesis, texture-by-numbers, texture transfer

1. BACKGROUND

Image analogies provide a way to learn relationships between images, using a single training pair. From images $A$ and $A'$, we can learn the relationship between these images, and from a new image $B$, we can generate an analogous image $B'$. Image analogies can be applied to many different applications, including artistic filters, texture synthesis, texture transfer, texture-by-numbers, super-resolution, and simple filter learning. For this project, we have reimplemented the methods discussed in Hertzmann et al. [1].

2. METHODS

To implement image analogies, we create feature vectors from image pairs $A:A'$ and $B:B'$. We loop over each pixel of $B'$ in scan-line order and find the best match between the $B:B'$ feature vector and the $A:A'$ feature vector, and we copy the pixel value from $A'$ to $B'$, as shown in Algorithm 1. Here we discuss in further detail the method proposed by Hertzmann et al.

Algorithm 1 Create analogous image

function CREATEIMAGE
  for each level $\ell$ do:
    for each pixel $q$ in $B'$ do:
      $p^* \leftarrow$ BestMatch($q, \ell, A, A', B, B', s_\ell$)
      $B'_\ell(q) \leftarrow A'_\ell(p^*)$
      $s_\ell(q) \leftarrow p^*$
  return $B'_L$

2.1. Multi-scale Representation

We first create Gaussian pyramids from each image. We loop over the levels of the pyramid and synthesize the pixels of $B'$ from the coarsest level to the finest level. The multi-scale representation allows the algorithm to represent high-level features, without requiring a large increase in dimensionality of the feature vectors.

2.2. Feature Vectors

Feature vectors are formed from the 5 by 5 neighborhood of the pixel at the current level $\ell$ and 3 by 3 neighborhood of the coarser level $\ell - 1$. We concatenate the neighborhood pixels from $A$ and $A'$ to form a feature vector for a single pixel, and the neighborhood pixels from $B$ and $B'$ to form another feature vector. We denote the feature vector corresponding to pixel $p$ as $F(p)$.

As suggested by Hertzmann et al., we work in the YIQ color space. The feature vectors may include luminance only, or all channels. For all examples shown here except texture-by-numbers, we use only luminance in the feature vectors. For artistic style transfer, we also apply a linear luminance remapping, to match the mean and standard deviation of the luminance in images $A$ and $B$.

2.3. Approximate and Coherent Matches

For each pixel in $B'$, we determine the best matching pixel from $A'$ using approximate and coherent matches, as shown in Algorithms 2-4. We find the nearest neighbor feature vector match using Approximate Nearest Neighbors (ANN). In this project, we used the Spotify Annoy library [2]. Often, the nearest neighbor match does not provide visually pleasing results. We also find a coherent match, using the following procedure. First, find source pixel locations in $A'$ of neighborhood pixels, and among the pixels that have the same relative location as the current pixel of $B'$, find the nearest neighborhood match.

We use a weighted distance metric between feature vectors. The vectors are pre-weighted by a Gaussian, such that nearby pixels have higher weight than farther pixels in the neighborhood.

To determine the best match, we then compare the distance to the coherent match and the distance to the approximate match weighted by $1 + K2^{L-L}$, where $K$ is a coherence parameter specifying the size of texture features and $L$ is the number of levels. We choose the match with the lower dis-
tance as the best match. We can then copy the luminance and/or color from $A'$ to $B'$.

Algorithm 2 Best match

```
function BESTMATCH(q, ℓ, A, A', B, B', sℓ)
    papproximate ← ApproximateMatch(q, ℓ, A, A', B, B')
    pcoherent ← CoherentMatch(q, ℓ, A, A', B, B', sℓ)
    dapproximate ← ||F(papproximate) − F(q)||₂
    dcoherent ← ||F(pcoherent) − F(q)||₂
    if dcoherent < dapproximate(1 + K2ℓ−L) then
        return pcoherent
    else
        return papproximate
```

Algorithm 3 Approximate match

```
function APPROXIMATEMATCH(q, ℓ, A, A', B, B', sℓ)
    p⋆ ← approx arg min_p ||F(p) − F(q)||₂
    return p⋆
```

Algorithm 4 Coherent match

```
function COHERENTMATCH(q, ℓ, A, A', B, B', sℓ)
    r⋆ ← arg min_r∈N(q) ||F(sℓ(r) + (q − r)) − F(q)||₂
    return s(r⋆) + (q − r⋆)
```

3. RESULTS AND DISCUSSION

We applied the image analogy to artistic filters, texture synthesis, texture transfer, and texture-by-numbers.

3.1. Artistic Filters

We use artistic filters with three different styles: pointilism, Van Gogh, and watercolor (Figures 1-4).

The training pairs for pointilism and Van Gogh styles were created by applying a filter [3] to a photograph, while the training pair for watercolor style was created by smoothing the texture of a watercolor painting [4].

3.2. Texture Synthesis

For texture synthesis, $A$ and $B$ are images with constant value, while $A'$ is a patch of texture. We can generate textures of different sizes from the original texture patch (Figure 5).

3.3. Texture Transfer

In texture transfer, $A$ and $A'$ are the same patch of texture, and $B$ is a grayscale image. The resulting image $B'$ is the image $B$ with the texture from $A'$ applied (Figure 6).

3.4. Texture-by-numbers

In texture-by-numbers, we manually label images with a colored mask and create a new color mask. Using image analogies, we can generate a new image with the textures specified by the color mask (Figure 7-8). Interestingly, the boundaries between different textures in the generated $B'$ images are realistic, since we sample these boundary pixels from the original boundaries in $A'$.

3.5. Discussion

The resulting images are overall visually pleasing. We can emulate various artistic filters and perform different operations with textures.
However, the algorithm has a slow runtime and requires a few minutes to synthesize images with height and/or width of approximately 300 pixels. In addition, the coherence parameter $K$ must be manually chosen and is dependent on the scaling of the image.

4. CONCLUSION

Overall, image analogies are a surprisingly effective, yet simple way to learn filter relationships between images. Future work can be done to speed up the algorithm through using patch-based approach or other methods, rather than synthesizing images pixel-by-pixel.

5. REFERENCES


Fig. 4: Watercolor. We generate images with the watercolor style, from the training pair in the first two images.

Fig. 5: Texture synthesis. We generate different sized rectangular patches of the flower and Ritz cracker textures from the original square patches.

Fig. 6: Texture transfer. We transfer the flower and Ritz cracker textures to the grayscale image.

Fig. 7: Texture-by-numbers
Fig. 8: Texture-by-numbers. The original map image comes from Google Maps. We use image analogies to generate the bottom map.