OBLIQUE DECISION TREES AS AN IMAGE MODEL FOR CUBIST IMAGE RESTYLING

Edric Chan[†] *Magzhan Gabidolla*[⋆] *Miguel Á. Carreira-Perpiñán*[⋆]

[†] Great Oak High School, Temecula, CA, USA
[⋆] Dept. Computer Science & Engineering, University of California, Merced. Merced, CA, USA

ABSTRACT

We propose the use of oblique decision trees and forests, originally a supervised machine learning approach, as a model of a single image. This results in a hierarchical partition into constant-color polygons ("poxels") that best approximates the source image. The tree or forest can be trained using a recent algorithm that applies alternating optimization over the nodes of a tree and the trees of a forest. In this paper, we use this for aesthetic effects, such as restyling an image to look like a cubist painting or a stained-glass window. We illustrate this with multiple examples and compare it with the results achieved by Generative AI and Neural Style Transfer.

***Index Terms*—** Decision trees, hierarchical piecewise-constant function, poxel, image representation, non-photorealistic rendering.

1. INTRODUCTION

Image processing techniques have long been used to manipulate images and achieve various effects. Recent years have seen the widespread development of machine learning models for this purpose, mostly based on deep neural networks. Here, we propose, we believe for the first time, to use *oblique trees and forests as a model for a single image*. Decision trees have existed for a long time in machine learning as predictive models for supervised settings, but traditional tree learning algorithms are quite suboptimal and limited in the form of the trees they can train for practical use (usually axis-aligned). Having a model that can better approximate images and being able to optimize it well is critical, even if the goal is purely aesthetic, because we need to have a sufficiently faithful representation of the source image that we can then manipulate for artistic goals. Our idea is motivated by the recent development of effective algorithms to learn tree-structured models for arbitrary choices of loss function and tree type [1, 2].

An oblique tree approximates a source image (considered as function from 2D to color space) by a hierarchical piecewise-constant function, where each piece is a convex, colored polygon that we will call *poxel* (for *polygon pixel*), and whose fidelity can be controlled by the model size (tree depth and number of trees). Such a representation has multiple uses, such as image segmentation and compression, but here we focus on aesthetic effects, which have applicability in software packages such as PhotoShop or for game and film production effects. The output image, which is a vector graphics object rather than a bitmap, is strikingly reminiscent of cubist paintings such as those of Picasso and other artists (see fig. 2). Other effects are also possible, such as the style of stained-glass windows. In general, our work opens the door for further manipulations based on the poxel representation. However, we emphasize that we are not trying to imitate faithfully these styles, each of which is created using specific tools and procedures (originally not digital). The images we create are the result of a specific image model and optimization algorithm and have their own style.

Throughout the paper, we will use the term *cubist tree or forest* to refer to our model or to images generated with it.

2. RELATED WORK

Our work has relations with multiple areas, which we briefly review.

Machine learning (ML). Our work is primarily an adaptation of a supervised ML model for image representation and manipulation. Decision trees and forests have a long history in statistics and ML, where they have been used mostly with the goal of prediction on test data, e.g. classification. They have been traditionally learned using greedy recursive partitioning (such as CART [3] or C5.0 [4]). This does not optimize a global loss function and is practically limited to axis-aligned trees (which split based on a single feature), so it produces large, unbalanced trees with low accuracy. The Tree Alternating Optimization (TAO) algorithm, reviewed later, does optimize an arbitrary combination of loss and tree model (such as oblique decision nodes), which produces higher accuracy with smaller trees.

Style transfer is a form of non-photorealistic rendering that manipulates a "content" image so it resembles the visual style of a "style" image. It can be used for aesthetic effects. Early work was based on patch-based texture synthesis [5, 6]. Neural Style Transfer (NST) [7] and its variations [8, 9] define the result by optimizing a weighted loss with the content and style images, but where each image is represented by features produced by a certain layer (capturing different image statistics) of a pretrained convolutional neural net.

Generative AI (GenAI) is a very recent, fast moving area, which includes different techniques, such as generative adversarial networks, variational autoencoders and diffusion models [10, 11]. They use deep neural nets to define a probability distribution over images from which samples can be generated, e.g. conditioned on a text prompt. This results in impressive images, though they often display unrealistic features and different samples show huge variability, so it is difficult to control the result obtained, even after iterated prompting. Training the neural nets in both NST and GenAI requires huge costs in terms of data, GPU hardware, storage and time.

Our approach has a restricted scope: it learns from a single image in a few seconds, which strongly determines the output image, and the user has significant control via a few hyperparameters.

Image processing. Many approaches exist to process images with different goals, from blurring, denoising or edge detection, to more complex ones such as segmentation or registration, and using techniques from local filters to partial differential equations [12, 13, 14]. Closer to us are quadtrees [15, 16, 17], but they are limited in both

*E. Chan is a senior at Great Oak High School. His contribution to this work took place during an internship at UC Merced in summer 2024. The last two authors are partially supported by NSF award IIS–2007147.

their shape and ability to fit an image. Our work can be seen as a sophisticated form of image processing that seeks a hierarchical piecewise-constant representation over polygons of a source image by optimizing a global objective function.

Computer graphics. Non-photorealistic rendering (NPR) [18, 19, 20, 21, 22] consists of techniques that enable expressive styles for digital art, inspired by other artistic modes such as painting and drawing, technical illustration and scientific visualization, and animated cartoons, among others. A specific type of NPR is stroke-based rendering (SBR) [23], which is based on (semi)automatically placing discrete elements such as paintbrush strokes or stipples to achieve some goal, for example to resemble the style of an oil painting. SBR methods have as primary application the manipulation of images for aesthetic effects. They have been implemented in many commercial packages and used in entertainment, such as game and film production effects. Some SBR techniques use greedy, local algorithms that place strokes on a real image as they go; this is fast, but achieves worse results than optimizing a global objective function that trades off the error of the resulting image vs the source image with the number of strokes. This produces results spanning a spectrum between abstract and realistic. Other approaches use low-level image processing, such as the Sobel operator and anisotropic and edge-preserving filters; other rendering primitives beyond strokes (regions, tiles, hatch marks); and combine it with segmentation [19]. Several approaches are based on Voronoi tessellations [24, 25], e.g. by placing a set of centroids in the image plane according to some criterion to define a Voronoi partition of the image and then coloring each cell. This tends to create honeycomb-like patterns with uniform-sized cells, due to the definition of Voronoi cells. A final type of approaches are photomosaics and jigsaw image mosaics (e.g. [26]), in which an image is approximated by a collection of smaller images. In [27] a kd-tree (a multidimensional data structure) is used to partition video into blocks.

Our work can be seen as a new SBR technique that, given a source image, uses hierarchical poxels as "strokes" to minimize the image error subject to a maximum number of strokes. Note our poxels can have a highly variable shape which adapts to the underlying image color and edges.

3. IMAGE MODEL: HIERARCHICAL POXELS

3.1. An image model as a regression problem

An image can be seen as a function from $\mathbb{R}^2$ (sampled on a 2D grid) to some other space: $\mathbb{R}$ if grayscale, $\mathbb{R}^3$ if color (RGB or a perceptual space such as LAB), or even a higher-dimensional space if representing each pixel as, say, a texture or SIFT feature vector. Throughout, we will focus on a 3D color space for definiteness. Thus, the N-pixel image can be considered a labeled dataset $\{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^N \subset \mathbb{R}^2 \times \mathbb{R}^3$, and a model of the image may be obtained by a least-squares regression of a mapping $\boldsymbol{\tau}$: $\mathbb{R}^2 \rightarrow \mathbb{R}^3$ with parameters $\boldsymbol{\Theta}$:

$$\min_{\boldsymbol{\Theta}} \sum_{n=1}^N \|\mathbf{y}_n - \boldsymbol{\tau}(\mathbf{x}_n; \boldsymbol{\Theta})\|^2. \quad (1)$$

$\boldsymbol{\tau}$ can then be applied to any point in $\mathbb{R}^2$, not just the grid points, and thus color the Euclidean 2D space. Fig. 1 illustrates the approach.

3.2. Oblique decision trees and forests

As model $\boldsymbol{\tau}$, we use either a single oblique regression tree or an ensemble (forest) of T of them, $\mathbf{F}(\mathbf{x}) = \frac{1}{T}\sum_{t=1}^T \boldsymbol{\tau}(\mathbf{x}; \boldsymbol{\Theta}_t)$. We consider binary, complete trees of depth Δ, thus having 2^Δ leaves. An oblique regression tree contains two types of nodes: decision nodes and leaves. A *decision node* i sends an input instance $\mathbf{x}$ down its right child if $\mathbf{w}_i^T\mathbf{x} + w_{i0} > 0$, else down its left child, where $(\mathbf{w}_i, w_{i0}) \in \mathbb{R}^2 \times \mathbb{R}$ are the weight vector and bias of a hyperplane (actually a line, in 2D). A *leaf* j does the actual prediction for $\mathbf{x}$, which is a constant vector $\boldsymbol{\mu}_j \in \mathbb{R}^3$, regardless of $\mathbf{x}$.

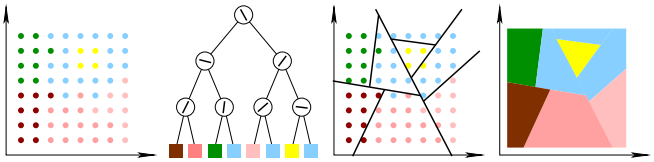

Fig. 1. Learning an oblique regression tree to represent a single image. *Plot 1*: an input image, as a grid of points $\mathbf{x}_n \in \mathbb{R}^2$ (input features) each with a color $\mathbf{y}_n \in \mathbb{R}^3$ (output labels). *Plot 2*: an oblique regression tree of depth 3 learned on this dataset. *Plot 3*: the partition of the 2D space induced by the tree. *Plot 4*: the partition with each leaf polygon colored by the leaf label (poxels).

Geometrically, the tree defines a nested, hierarchical partition of the 2D grid, with each leaf defining a convex polygon (with at most Δ sides, the intersection of the halfspaces given by the Δ splits along its root-leaf path) of constant color—a hierarchical piecewise-constant function. This is our *hierarchical poxels image model*. The forest defines a partition that is the intersection of the individual tree partitions, hence it has up to $2^{T\Delta}$ convex polygons (although this is a very coarse upper bound).

This is a good model for images for several reasons. First, a piecewise-constant function is a reasonable model for images because natural images typically do consist of objects that can be approximated by (sufficiently many) constant-color regions. Second, in a hierarchical model the polygons share sides, so it uses fewer parameters than if we had to represent each polygon on its own. This is not a very limiting restriction, and it also makes it easier and faster to draw the partition (by drawing the splits recursively). The overall quality of the model will depend on its size (Δ, T). With a good optimization (shown below), the polygons adapt to represent approximate level sets in the image, and their sides approximately match image contours, such as object boundaries.

3.3. Single tree optimization with TAO

Due to space limitations, we give a brief description of the training algorithms; more details can be found in [1, 2]. The *Tree Alternating Optimization (TAO)* algorithm optimizes an arbitrary objective function, such as eq. (1), over the parameters $\boldsymbol{\Theta}$ of a tree of fixed structure but arbitrary node models, such as axis-aligned or (our case) oblique. Each iteration monotonically decreases the objective until convergence, and consists of updating each node on its own given the rest are fixed. There are two reasons why this is effective. First, assume we want to optimize jointly over all nodes at the same depth (given all other nodes are fixed). This is mathematically equivalent to optimizing each node at that depth independently (*separability condition*). It simplifies the optimization, makes the steps down the objective function larger and introduces significant parallelism. Second, the problem of optimizing a single node given the rest are fixed takes a special form (the *reduced problem*) that is conveniently solvable, at least approximately. For a leaf node j, the optimal $\boldsymbol{\mu}_j$ is the mean of the $\mathbf{y}_n$ vectors for points (pixels) reaching that leaf (its *reduced set*); this means a poxel's optimal color is the average color of the pixels it contains. For a decision node i, the optimal $\mathbf{w}_i, w_{i0}$ are the solution of a weighted 0/1 loss classification problem defined over the reduced set points, each labeled with whatever child pro-

duces the lower objective function value downstream, and weighted with that value. That is, the split tries to send each point down so it is best predicted by the rest of the tree. This is an NP-hard problem, but it can be well approximated by a surrogate loss; we use a logistic regression. Occasionally, the surrogate classifier may have a higher weighted 0/1 loss than the current parameters, in which case we skip the update for this node, to ensure monotonic decrease.

While this seems complex, the final algorithm operates in a simple way. At each iteration, we visit each node from the leaves upwards to the root and update its parameters, by either computing a mean vector (if a leaf) or fitting a binary classifier (if a decision node). Its computational complexity is $\mathcal{O}(N\Delta^2)$ [1], i.e., linear on the image size N, and thus scalable.

Convergence is to a local optimum that depends on the initial tree parameters, which we take as a random median tree: starting from the root, we pick its weight vector orientation at random and set its bias to the median (so it partitions the points 50/50). We repeat this recursively down the tree, so each leaf receives $N2^{-\Delta}$ points. This already produces a nice effect, but further iterations better adapt it to the source image.

Our current implementation is for a multicore CPU (not GPU). It uses C and OpenMP for parallel processing, and LIBLINEAR [28] for logistic regression. This is already fast enough to process an image in a few seconds in a laptop. However, this can be improved substantially by noting that the uniform, 2D nature of the grid makes some optimizations possible (such as storing parameters with few bits and using a scan over orientations rather than LIBLINEAR).

3.4. Forest optimization with FAO

To optimize eq. (1) over a forest $\mathbf{F}$ rather than a tree $\boldsymbol{\tau}$, we use the *Forest Alternating Optimization (FAO)* algorithm [2]. Each FAO iteration over the whole forest consists of two steps. One consists of applying alternating optimization one tree at a time (given the rest of the trees are fixed). This can be done with TAO because, in (1), $\mathbf{y}_n$ minus the fixed trees' prediction can be taken as a fixed label (effectively, the residual error) that the tree we optimize over needs to predict. The other step consists of optimizing over the output vectors of all leaves of all T trees jointly given the decision nodes are fixed. This is a quadratic problem whose solution is given by a linear system. As with TAO, all these steps guarantee monotonic decrease of the objective function until convergence.

The final algorithm is again simple. At each iteration, we update each tree with TAO, and we solve a linear system over all the leaves. Its computational complexity is $\mathcal{O}(NT\Delta^2 + (T2^{\Delta})^3)$ [2], i.e., also linear on the image size.

3.5. Postprocessing of the tree or forest

Having trained a tree or forest with TAO or FAO, respectively, we have our image model, which contains implicitly the poxels. In order to make them explicit, for plotting or for manipulation with software such as PhotoShop, we construct all the polygons and save them as an EPS file, using a Python implementation. This is a vector graphics object and can be rendered at any resolution.

4. USER PARAMETERS AND CONTROL: EFFECTS

Although, at heart, we use a ML algorithm, the goals for achieving aesthetic effcts are different from those of a ML (or image processing) task. For the latter, usually one wishes a high degree of automatization. For example, the final ML model would be the one that achives best generalization in a validation set. But for an artist or special effects designer, some amount of control and interactivity of the result is critical. We can achieve this by tuning various user parameters (and by combining our image model with traditional image manipulation software). Given this, the automatic part is achieved by the optimization algorithm, producing a best fit to the source image based on those parameters—that is, making the image as realistic as possible but within the design rules we impose.

The "cubist" appearance of our generated images is due to partitioning the source image into poxels, each of which optimally seeks to cover a homogeneous region of the image, often corresponding to part of an object, and to align with its contours or object boundaries. Having more poxels will produce a more faithful approximation to the source image, spanning a spectrum between a constant-color image (a tree of depth 0) and a photorealistic one (a deep tree); the more interesting effects occur away from those extremes.

An oblique tree includes as special case an axis-aligned tree, where the split uses a single feature, thus being either horizontal or vertical, and generating rectangular poxels. However, this is a much more limited model for images (whose contours can have arbitrary orientations). While we do not show them here, axis-aligned trees do produce interesting, blocky Mondrian-like effects, but they require much larger trees and hence many more poxels to approximate the image, and the produce jagging artifacts[1].

The user parameters and their effect is as follows:

Tree depth Δ and number of trees T These control the size of the model: 2^{Δ} poxels for a single tree and up to $2^{T\Delta}$ for $T > 1$ (empirically, this is closer to a factor times 2^{Δ}). With few poxels, the result is cartoon-like. The bigger the model, the more photorealistic the result. Using multiple trees gives an appearance of "broken glass" compared to using one tree.

Seed This sets the state of the pseudorandom number generator and thus the initial, random median tree. This can be used to add a small amount of randomness to the result, akin to the fact that a human artist would never draw the same identical picture twice. With video, it could be used to create a form of rotoscopic animation, as in the "Waking Life" 2001 film. On the other hand, for video or animation we can achieve frame-to-frame coherence by initializing the model for each frame from the trained model for the previous frame (warm-start). This also speeds up the optimization (≈1-2 iterations/frame).

Number of iterations As the TAO/FAO algorithm iterates from the initial tree, it approaches the optimal result. Like the seed, this provides a form of randomness.

Line width of the poxel boundaries This is useful to show more clearly the poxels, but also to produce an appealing effect reminiscent of stained-glass windows, such as those in European gothic cathedrals.

We rescale the source image down to a maximum of 100 pixels per dimension (width or height) and run TAO/FAO on that. This speeds up the algorithm and does not change much the results, because useful values of Δ and T (producing the more interesting effects) mean that the number of poxels is quite smaller than the number of pixels.

There are additional effects we can achieve, but they are out of the scope of this paper. Many of them rely on the use of a mask (automatic or user-selected) so that the tree or forest is learned on only the pixels within the mask. This makes it possible to combine multiple cubist models with original pixels of the source image, for example a figure vs a background, or to use them in other ways.

[1] The visual aspect is similar to Treemaps [29], a visualization technique for hierarchical data, whose aesthetic beauty has been recognized before (https://treemapart.wordpress.com).

Fig. 2. *Mona Lisa* (Leonardo da Vinci), *Starry Night* (Van Gogh) and a cubist painting, *Les Demoiselles d'Avignon* (Picasso).

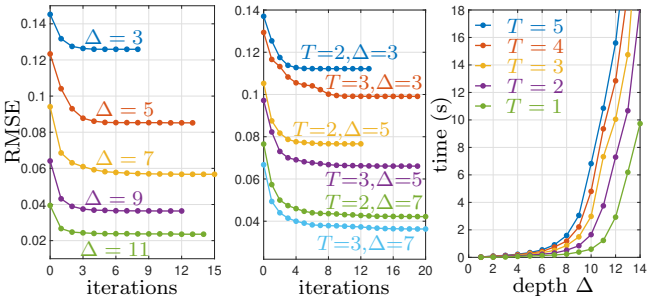

Fig. 3. RMSE over TAO/FAO iterations and training time per iteration for *Mona Lisa*, for different depth Δ and number of trees T.

5. EXPERIMENTS

We show a selection of effects on paintings, photographs and cartoons. **Zoom into the images to see individual polygons.**

Our experiments were ran in a Linux laptop with processor Intel Core i9 2.2 GHz with 24 cores and 32 GB of RAM (using only CPUs, not GPU). In all cases, we use RGB space and reduce the source image to a maximum size of 100×100 pixels before training. This results in at most around 10^4 poxels, for $\Delta \leq 14$ and $T \leq 3$.

Fig. 2 shows several images we use. In terms of quantitative evaluation, fig. 3 shows the RMSE and training time. The RMSE decreases monotonically over iterations of TAO and FAO. A near-optimum result occurs by around 5 iterations, typically. The training time in a laptop is just a few seconds unless the tree is very deep. Fig. 4 shows how the poxels change shape significantly upon training, as they adapt to the source image.

Fig. 5 shows the effect of the initial random tree (by changing the seed), which injects a bit of randomness in the result. Fig. 6 shows the effect of varying the depth Δ and number of trees T for a photograph. As the number of poxels increases, the cubist image moves from cartoonish, to a "cubist" appearance, to increasingly more realistic. Using multiple trees produces an effect of glassy reflections. If a tree is deep enough to have more leaves than pixels ($\Delta \geq \log_2 N$), then it can fit the source image exactly (in fact, the initial random median tree already does so). However, the corresponding poxels are not necessarily squares centered at the pixels. Thus, when plotting the cubist tree image on the entire 2D plane, it looks like a distorted version of the source image, which achieves an interesting effect. This starts being noticeable as an increasingly jagged, noisy aspect as Δ or T become large, as in the example for $\Delta = 12$, $T = 1$ in fig. 6 or the *Starry Night* example for $\Delta = 14$, $T = 1$ in fig. 12.

Our image model produces a vector graphics object, consisting of geometric objects defined by mathematical formulas (color-filled polygons in our case, mostly). This gives shapes that can be scaled to any size without loss of quality. This is unlike bitmap graphics (such as those produced by neural style transfer and diffusion models), which have a finite resolution, and show pixelation and jagging effects if zooming in sufficiently. Fig. 7 illustrates this by showing a highly zoomed-in region for both the source and cubist image. The resulting geometric objects can also be manipulated (with a vector graphics editor such as Adobe Illustrator, CorelDRAW, Inkscape, etc.) in various ways, e.g. to rearrange or merge the polygons, change their color or texture, transform them, etc. Fig. 7 (right plot) illustrates this. Consequently, our approach (possibly in conjunction with other design tools) could be used for vector art for illustrations ranging from small size, such as company logos on business cards, to very large size, such as posters and billboards; as well as for printing on clothing, bags, stickers or even tattoos.

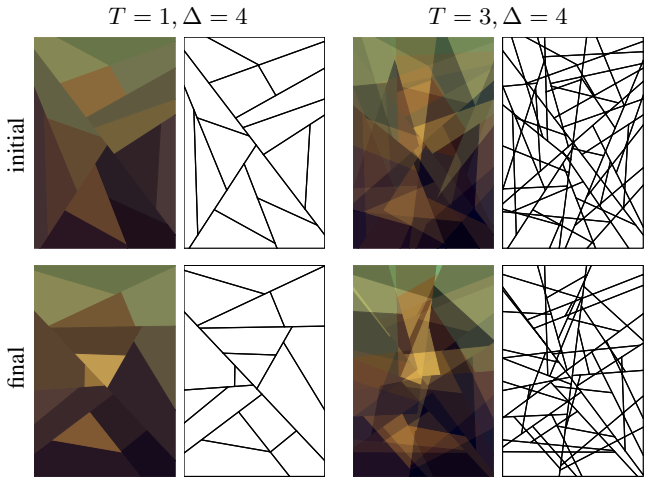

Fig. 4. Initial (random median trees) and final result (after TAO/FAO converge) for *Mona Lisa*, as poxels and boundaries, for two models.

Fig. 5. *Left*: source image. *Rest*: cubist tree images using different seeds (for $\Delta = 6$, $T = 3$). Combining these images into a video produces an jittery effect reminiscent of rotoscopic animation; see https://youtu.be/TXPmOmw4a_A.

Fig. 8 shows how drawing the poxel boundaries with thick lines achieves a stained-glass window effect (compare with a real window shown on the right). Fig. 11 shows how individual trees combine into a forest. Fig. 12 shows a selection of cubist tree and forest images obtained from various photographs, paintings and cartoons.

Finally, for comparison purposes, figs. 9 and 10 show an attempt to generate images with cubist or stained-glass style using Neural Style Transfer and Stable Diffusion. The results are quite odd, although interesting in their own way.

6. CONCLUSION

We have repurposed oblique decision trees and forests as a model that best approximates a source image by a hierarchical piecewise-constant function. They can be trained in seconds in a laptop using the TAO and FAO algorithms. Their visual aspect can be controlled with several user parameters to achieve various image effects, as in cubist and stained-glass artwork. This opens the door to other effects based on manipulating the set of poxels. We are also working on applying this model to image compression and image segmentation.

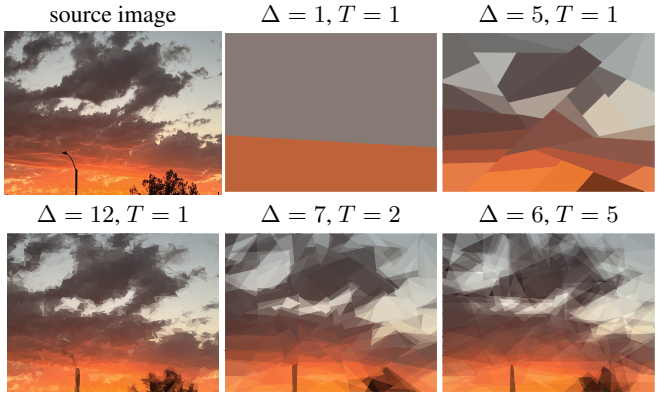

Fig. 6. Cubist tree and forest results using different depth Δ and number of trees T for a source photograph (©Edric Chan).

Fig. 7. *Plots 1–2*: source image and zoomed-in region (mouth and nose). The original pixels are clearly visible. *Plots 3–4*: cubist tree output and zoomed-in region in the same area. The (infinite-resolution) poxels are visible. *Plot 5*: cubist tree output edited with Inkscape to color some of the polygons in the background in green, and to move away some of the polygons in one eye to the right side.

Fig. 8. Stained-glass window effect (left, using $\Delta = 6$ and $T = 1$) for a source photograph (middle, ©Edric Chan). For reference, on the right we show a photograph of an actual stained-glass window.

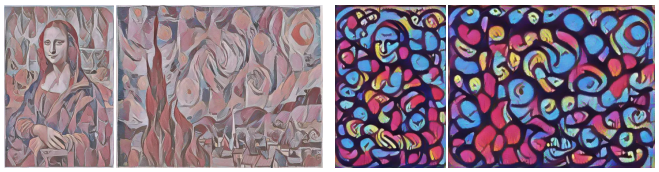

Fig. 9. Using Neural Style Transfer [7] to restyle two content images (*Mona Lisa* and *Starry Night*) with the style of the cubist painting in fig. 2 (left) and the stained-glass window in fig. 8 (right). We use the implementation in https://huggingface.co/spaces/Hexii/Neural-Style-Transfer.

Fig. 10. Generating images with Stable Diffusion with a text prompt where X is *Mona Lisa* or *Starry Night* (one sample for each prompt). We used Stable Diffusion 2.1 (the current text-to-image model from StabilityAI) at https://huggingface.co/spaces/stabilityai/stable-diffusion.

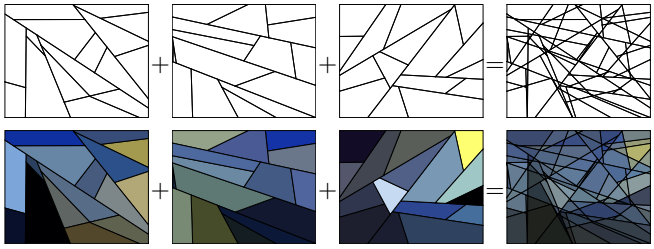

Fig. 11. *Starry Night* using a forest of $T = 3$ trees of depth $\Delta = 4$. We show how the individual trees, each having $2^{\Delta} = 16$ poxels, add up to the final forest, having 158 poxels (above: boundaries, below: poxels). Unlike the forest, the trees have little resemblance to the source image and differ considerably from each other.

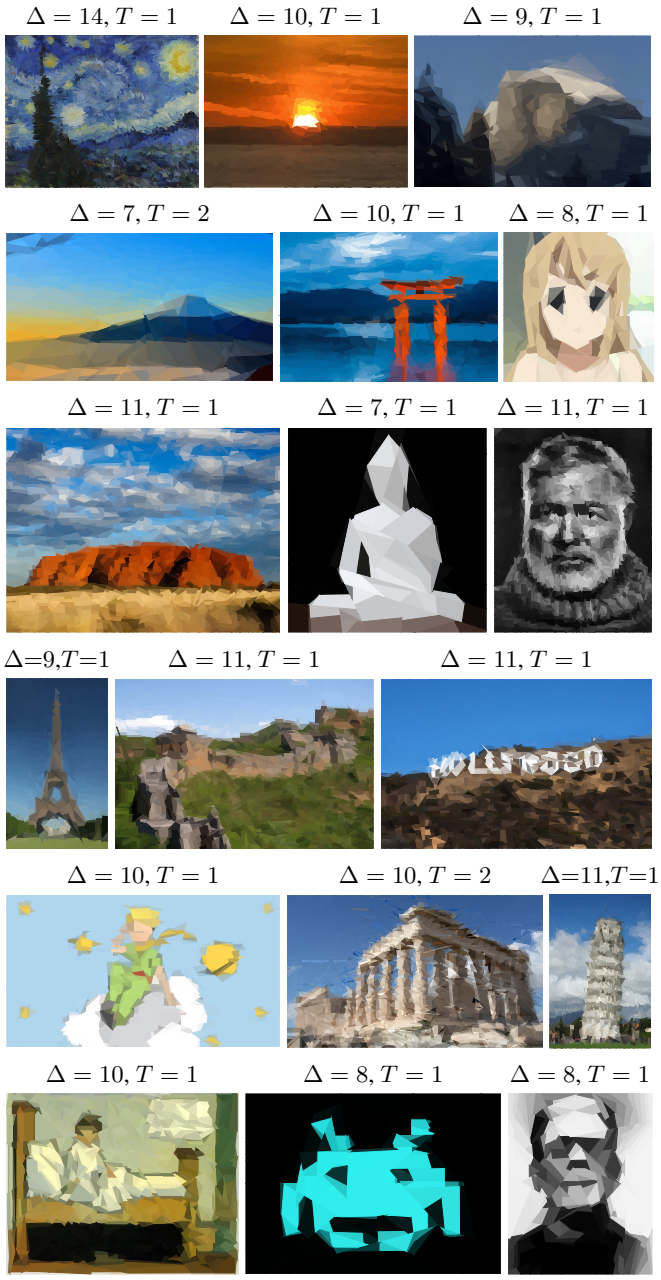

Fig. 12. Art gallery. Can you guess the original paintings, drawings or photographs?

7. REFERENCES

[1] Miguel Á. Carreira-Perpiñán and Pooya Tavallali, "Alternating optimization of decision trees, with application to learning sparse oblique trees," in *Advances in Neural Information Processing Systems (NeurIPS)*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds. 2018, vol. 31, pp. 1211–1221, MIT Press, Cambridge, MA.

[2] Miguel Á. Carreira-Perpiñán, Magzhan Gabidolla, and Arman Zharmagambetov, "Towards better decision forests: Forest Alternating Optimization," in *Proc. of the 2023 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'23)*, Vancouver, Canada, June 18–22 2023, pp. 7589–7598.

[3] Leo J. Breiman, Jerome H. Friedman, R. A. Olshen, and Charles J. Stone, *Classification and Regression Trees*, Wadsworth, Belmont, Calif., 1984.

[4] J. Ross Quinlan, *C4.5: Programs for Machine Learning*, Morgan Kaufmann, 1993.

[5] Aaron Hertzmann, Charles E. Jacobs, Nuria Oliver, Brian Curless, and David H. Salesin, "Image analogies," in *Proc. of the 28th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH 2010)*, Lynn Pocock, Ed., Los Angeles, CA, Aug. 12–17 2010, pp. 327–340.

[6] Alexei A. Efros and William T. Freeman, "Image quilting for texture synthesis and transfer," in *Proc. of the 28th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH 2010)*, Lynn Pocock, Ed., Los Angeles, CA, Aug. 12–17 2010, pp. 341–346.

[7] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge, "Image style transfer using convolutional neural networks," in *Proc. of the 2016 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'16)*, Las Vegas, NV, June 26 – July 1 2016, pp. 2414–2423.

[8] Kristin J. Dana, *Computational Texture and Patterns: From Textons to Deep Learning*, Synthesis Lectures on Computer Vision. Morgan & Claypool Publishers, 2018.

[9] Yongcheng Jing, Yezhou Yang, Zunlei Feng, Jingwen Ye, Yizhou Yu, and Mingli Song, "Neural style transfer: A review," *IEEE Trans. Visualization and Computer Graphics*, vol. 26, no. 11, pp. 3365–3385, Nov. 2020.

[10] Florinel-Alin Croitoru, Vlad Hondru, Radu Tudor Ionescu, and Mubarak Shah, "Diffusion models in vision: A survey," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 45, no. 9, pp. 10850–10869, Sept. 2023.

[11] Ling Yang, Zhilong Zhang, Yang Song, Shenda Hong, Runsheng Xu, Yue Zhao, Wentao Zhang, Bin Cui, and Ming-Hsuan Yang, "Diffusion models: A comprehensive survey of methods and applications," *ACM Computing Surveys*, vol. 56, no. 4, pp. 105:1–39, Apr. 2024.

[12] Rafael C. Gonzalez and Richard E. Woods, *Digital Image Processing*, Prentice-Hall, second edition, 2002.

[13] William K. Pratt, *Introduction to Digital Image Processing*, CRC Publishers, 2013.

[14] Tony F. Chan and Jianhong (Jackie) Shen, *Image Processing and Analysis: Variational, PDE, Wavelet, and Stochastic Methods*, Other Titles in Applied Mathematics. SIAM Publ., 2005.

[15] Rahul Shukla, Pier Luigi Dragotti, Minh N. Do, and Martin Vetterli, "Rate-distortion optimized tree-structured compression algorithms for piecewise polynomial images," *IEEE Trans. Image Processing*, vol. 14, no. 3, pp. 343–359, Mar. 2005.

[16] Gabriel Peyré, "A review of adaptive image representations," *IEEE J. Selected Topics in Signal Processing*, vol. 5, no. 5, pp. 896–911, Sept. 2011.

[17] Adam Scholefield, , and Pier Luigi Dragotti, "Quadtree structured image approximation for denoising and interpolation," *IEEE Trans. Image Processing*, vol. 23, no. 3, pp. 1226–1239, Mar. 2016.

[18] Bruce Gooch and Amy Gooch, *Non-Photorealistic Rendering*, A. K. Peters, Ltd., 2001.

[19] Jan Eric Kyprianidis, John Collomosse, Tinghuai Wang, and Tobias Isenberg, "State of the "art": A taxonomy of artistic stylization techniques for images and video," *IEEE Trans. Visualization and Computer Graphics*, vol. 19, no. 5, pp. 866–885, May 2013.

[20] Florian Nolte, Andrew Melnik, and Helge Ritter, "Stroke-based rendering: From heuristics to deep learning," arXiv:2302.00595, Dec. 30 2022.

[21] Paul Rosin and John Collomosse, Eds., *Image and Video-Based Artistic Stylisation*, Number 42 in Computational Imaging and Vision. Springer-Verlag, 2013.

[22] Thomas Strothotte and Stefan Schlechtweg, *Non-Photorealistic Computer Graphics: Modeling, Rendering, and Animation*, Morgan Kaufmann, 2002.

[23] Aaron Hertzmann, "A survey of stroke-based rendering," *IEEE Computer Graphics & Applications*, vol. 23, no. 4, pp. 70–81, July – Aug. 2003.

[24] Adrian Secord, "Weighted Voronoi stippling," in *Proc. Workshop on Non-Photorealistic Animation and Rendering (NPAR 2002)*, Annecy, France, June 3–5 2002, pp. 37–43.

[25] III Kenneth E. Hoff, John Keyser, Ming Lin, Dinesh Manocha, and Tim Culver, "Fast computation of generalized Voronoi diagrams using graphics hardware," in *Proc. of the 26th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH 1999)*, Warren Waggenspack, Ed., Los Angeles, CA, Aug. 8–13 1999, pp. 277–286.

[26] Alejo Hausner, "Simulating decorative mosaics," in *Proc. of the 28th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH 2010)*, Lynn Pocock, Ed., Los Angeles, CA, Aug. 12–17 2010, pp. 573–580.

[27] Allison W. Klein, Peter-Pike J. Sloan, Adam Finkelstein, and Michael F. Cohen, "Stylized video cubes," in *Proc. of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA 2002)*, San Antonio, TX, July 21–22 2002, pp. 15–22.

[28] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin, "LIBLINEAR: A library for large linear classification," *J. Machine Learning Research*, vol. 9, pp. 1871–1874, Aug. 2008.

[29] Ben Shneiderman, "Tree visualization with tree-maps: 2-d space-filling approach," *ACM Trans. Graphics*, vol. 11, no. 1, pp. 92–99, Jan. 1992.