COMP4010 PROJECT 2 WRITEUP: GROUP 13

PROJECT TITLE: *aRt: Recreating Art With R*

Member List

No.	Name	Student ID	Email
1	Tran Hung Dat	V202200889	22dat.th@vinuni.edu.vn
2	Ekaterina Balashova	V202100391	21ekaterina.b@vinuni.edu.vn
3	Vo Khoi Thanh Lam	V202000120	20lam.vkt@vinuni.edu.vn

Link to Github Repository: [Link]

1 Introduction

This project explores the creative potential of R for visual art. Our goal is to manually recreate well-known artworks using R's plotting ecosystem, including packages like `ggplot2`, `grid`, and `patchwork`.

While R is traditionally used for statistical graphics, we challenge that boundary by applying it to artistic compositions. Each artwork is rebuilt from scratch using custom shapes, color manipulation, and geometric layering.

What makes our approach unique is the focus on manual recreation rather than generative or abstract art. This allows us to explore R's graphics system in depth while learning about visual structure, composition, and reproducibility through code.

2 Overview of Artworks and Technical Approach

This project focuses on recreating two distinct artworks:

- **"Altarpiece – No 1"** by Hilma af Klint (1907): A work of geometric abstraction created entirely using mathematical forms.
- **"Red Fuji"** by Hokusai: A classic Japanese woodblock print involving curves, gradients, and complex layering.

These artworks were selected because they pose different technical challenges. The first emphasizes geometric layout and symmetry, while the second requires careful handling of curves, layering, and color gradients. Both align well with R's strengths in grid-based graphics and programmable composition.

Our implementation strategy makes use of the following:

- **Packages:** `ggplot2`, `grid`, `ggforce`, `geomtextpath`, `colorspace`, and `magick`.
- **Techniques:** Layered plotting, coordinate manipulation, path construction, transparency/alpha blending, and text-on-path rendering.

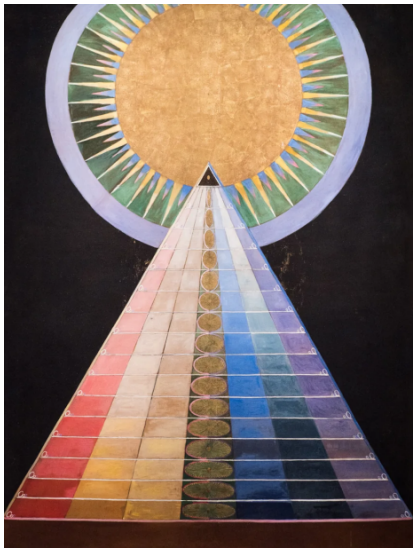
For *Altarpiece – No 1*, all visual components were created using mathematical functions directly in R. No external data was needed.

In contrast, *Red Fuji* required extracting data points from the original artwork. We used Python's PIL (Python Imaging Library) to process the image and output coordinate and color information into CSV files, which were then read and visualized in R. This hybrid approach allowed for greater fidelity and performance when recreating complex shapes and gradients.

3 Implementation (Condensed with Smaller Images)

Altarpiece – No 1 (Hilma af Klint, 1907)

This artwork was recreated entirely within R using mathematical functions and plotting techniques. The implementation involved:



Step 1: Libraries & Reference Image

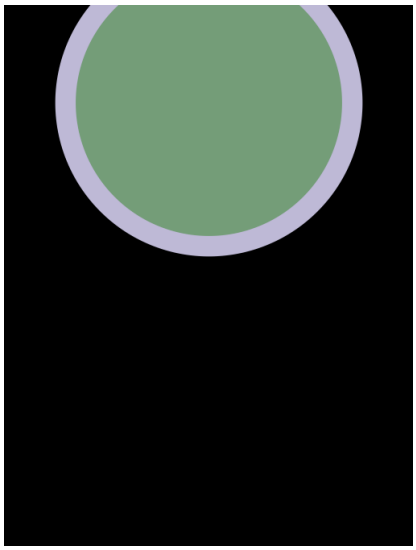
Setup, Libraries & Reference Image

- Load key packages: `ggplot2`, `ggforce`, `grid`, `dplyr`, `ggpattern`.
- Apply a minimalist canvas:

```
theme_set(theme_void())
```

- Read the original PNG to confirm dimensions:

```
img <- png::readPNG("altarpiece_original.png")
grob <- grid::rasterGrob(img)
grid::grid.draw(grob)
w <- dim(img)[2]
h <- dim(img)[1]
```



Step 2: Background & Circles

Background Rectangle & Central Circles

- Draw a full-canvas black rectangle:

```
background <- geom_rect(
  aes(xmin = 0, xmax = w,
      ymin = 0, ymax = h),
  fill = "black"
)
```

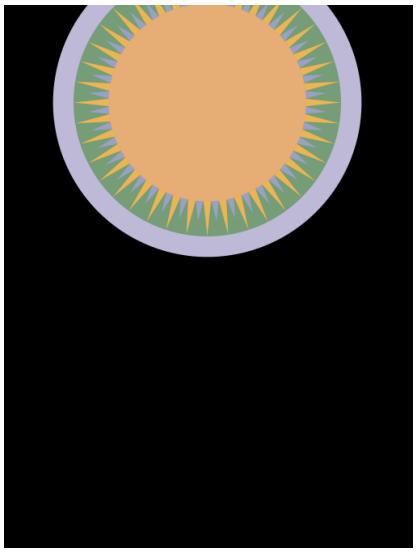
- Plot with fixed coordinates:

```
ggplot() +
  background +
  coord_fixed(xlim   = c(0, w),
              ylim   = c(0, h),
              expand = FALSE)
```

- Compute circle center and draw two concentric circles:

```
cx <- w * 0.5
cy <- h * 0.82

circle1 <- geom_circle(
  aes(x0 = cx, y0 = cy, r = w * 0.375),
  fill = "#BFBAD7"
)
circle2 <- geom_circle(
  aes(x0 = cx, y0 = cy, r = w * 0.325),
  fill = "#749D78"
)
circles <- list(circle1, circle2)
```



Step 3: Sun Rays & Disk

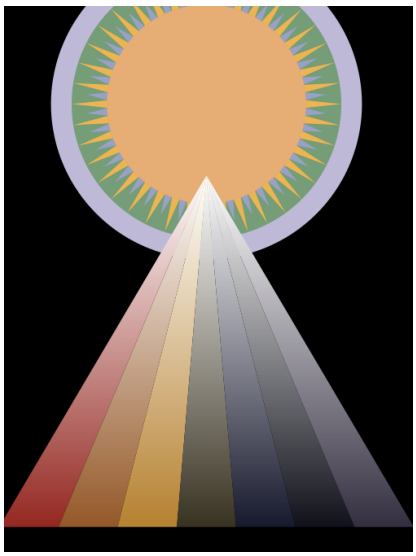
Sun Rays and Central Disk

- Define two small "ray" polygons (gold and blue).
- Rotate each by multiples of 9° around (cx, cy) (40 copies) using:

```
rotate_shape <- function(df, angle_deg, cx, cy) {
  angle <- angle_deg * pi / 180
  mat <- matrix(c(cos(angle), -sin(angle),
                  sin(angle),  cos(angle)), ncol
                    = 2)
  coords <- as.matrix(df[, c("x","y")] - c(cx,cy)
      )
  rotated <- coords %*% mat + c(cx,cy)
  data.frame(x = rotated[,1],
             y = rotated[,2],
             group = df$group)
}
```

- Bind rotated copies into `gold_rays` and `blue_rays`, then draw:

```
gold_sun <- geom_polygon(
  data = gold_rays,
  aes(x = x, y = y, group = group),
  fill = "#EDB452"
)
blue_sun <- geom_polygon(
  data = blue_rays,
  aes(x = x, y = y, group = group),
  fill = "#99A2C5"
)
sun_circle <- geom_circle(
  aes(x0 = cx, y0 = cy, r = w * 0.24),
  fill = "#E6AE74"
)
sun <- list(gold_sun, blue_sun, sun_circle)
```



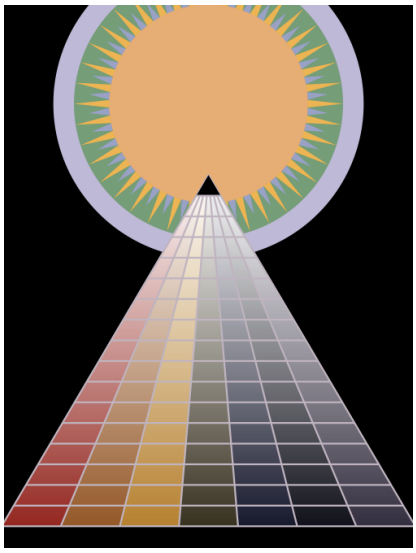
Step 4: Gradient-Filled Triangles

Main Gradient-Filled Triangles

- Construct seven triangular wedges ($i = 1 \dots 7$):

$$t_i \leftarrow \text{data.frame}\begin{pmatrix} x = \big((i-1)\,\tfrac{w}{7},\ 0.5\,w,\ i\,\tfrac{w}{7}\big), \\ y = \big(0.05\,h,\ 0.69\,h,\ 0.05\,h\big), \end{pmatrix}.$$

- Fill each triangle with `geom_polygon_pattern(pattern="gradient")`, using a red-to-purple ramp.
- Combine into `main_triangles <- list(triangle1, ..., triangle7)`.



Step 5: Black "Eye" Triangle & Grid Edges

Black "Eye" Triangle and Grid Edges

- Draw inverted black triangle at apex:

  $tb \leftarrow \text{data.frame}\big(x = \{0.5\,w \pm \frac{w}{34}\},\ y = \{0.69\,h - lh\},\ group = 1\big);$

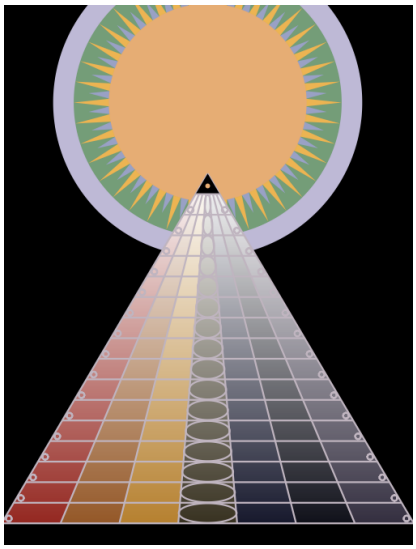
- Vertical edges at eight equally spaced x positions:

  $ve\_df \leftarrow \text{data.frame}($
  $x = (0, w, \text{length.out} = 8),$
  $y = 0.05\,h,$
  $xend = (0, w, \text{length.out} = 8),$
  $yend = 0.69\,h$
  $);$

  `geom_segment(data=ve_df, color="#C0B5C0", size=0.5)`

- Horizontal grid lines (17 total) by interpolating y from $0.05\,h$ to $0.69\,h$, drawn with

  `geom_segment(color="#C0B5C0", size=0.5)`.



Step 6: Finishing Touches

Finishing Touches

- Add a small "eye" circle near the top of the black triangle:

  `geom_circle(aes(x0=0.5w, y0=0.69h-0.6lh, r=5), fill="#E6AE74")`

- Draw 16 small circles along each vertical edge with

  `geom_circle(fill=NA, color="#C0B5C0")`.

- Create 16 nested ellipses inside the triangle
- Combine into `fin <- list(eye, circles_left, circles_right, ellipses)`.
- Final plot call layers all components

Red Fuji (Katsushika Hokusai)

We recreated "Red Fuji" by extracting and plotting five distinct layers—mountain, forest, clouds, trees, and snow—using Python for image processing and R for visualization. Because the original print was of low resolution, we first used a generative AI model to produce a cleaner, simplified base image. For each layer (e.g., clouds or trees), we then ran a small Python/PIL script to identify and export the (x, y) coordinates of relevant pixels into CSV files. In R, these CSVs are read and rendered via `geom_polygon` or `geom_point`, allowing us to rebuild each feature with precise control over color, opacity, and layering.

With these six steps—loading the reference, plotting mountain outline, forest, clouds, trees, and snow—we reconstruct "Red Fuji" in R using data extracted from the original image. Full code and parameter details can be found in the repository.

Step 1: Reference Image

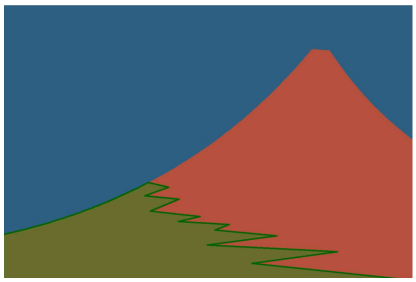
Load Reference Image & Dimensions

- In R, read the original "Red Fuji" PNG via `png::readPNG("red_fuji.png")`.
- Use `dim(img)` to extract `width` and `height`. These define the plotting canvas (e.g. `w = 1024`, `h = 768`).
- Apply `theme_void()` so no axes appear when layering elements.

Step 2: Mountain Silhouette

Plot Mountain Outline

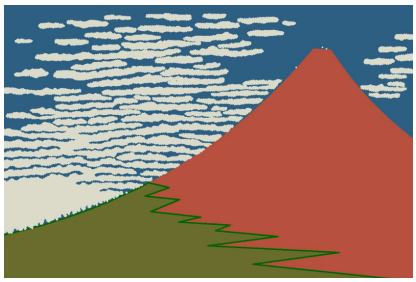
- In Python (using `PIL`), convert the reference to grayscale and apply edge detection (e.g. Canny) to isolate Fuji's outline.
- For each row y, record leftmost and rightmost edge pixels as outline points (x, y).
- Save these (x, y) pairs to `data/mountain_outline.csv`.
- In R, import `mountain_outline.csv` via `readr::read_csv()`, and draw it using `geom_polygon(fill=NA, color="black", size=0.6)`.
- Closing the polygon at the base uses two extra points at $(0,\ y_{\text{base}})$ and $(w,\ y_{\text{base}})$.



Step 3: Forest Layer

Generate Forest Silhouette

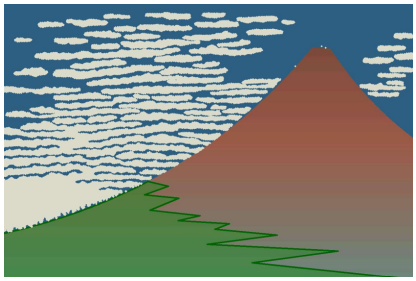
- From the original image, isolate the lower slopes (forest region) by thresholding green-ish pixels in Python or by manually sampling a horizontal band.
- Export those (x, y) points to `data/forest.csv`.
- In R, read `forest.csv` and plot with `geom_polygon(fill="#2E5A2F", color=NA)` to recreate the dark forest silhouette.
- Ensure the forest sits exactly beneath the mountain outline, by matching the same y-coordinates for continuity.



Step 4: Cloud Points

Extract & Plot Clouds

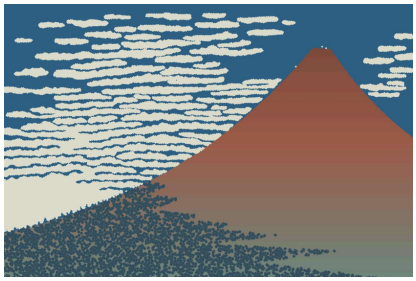
- Python script (`extract_cloud_data.py`) loads `extract_cloud.png` via PIL and builds a boolean mask $R, G, B \geq 240$ to identify white pixels (cloud).
- Coordinates (x, y) of those pixels (flipping y so origin is bottom-left) are downsampled to at most 50 000 points for performance.
- Write out `data/cloud.csv` with headers $\{x, y\}$.
- In R, import `cloud.csv` and draw points via `geom_point(aes(x,y), color="white", size=0.5, alpha=0.6)` to recreate fluffy clouds.



Step 5: Gradient Color by Elevation

Apply Elevation-Based Gradient

- After plotting the mountain outline, compute a vertical sequence of y-values from base to peak.
- In R, create a data frame with (x, y) grid points covering the mountain region.
- Assign each point a fill-color by mapping its y-coordinate to a gradient palette (e.g. red at lower y, pink in mid-tones, white near the summit).
- Use `geom_tile(aes(x=x, y=y, fill=fill_hex))` with `scale_fill_identity()` to paint the gradient beneath the mountain silhouette.
- This recreates the gradual color shift seen in Hokusai's print, transitioning from deep reds at the base to pale tones at the top.



Step 6: Tree Silhouettes

Extract & Draw Trees

- In Python, threshold dark-green pixels in the mid-slope region to isolate tree tops.
- Flip y and save extracted (x, y) coordinates to `data/trees.csv`.
- In R, import `trees.csv` and plot with `geom_point(aes(x,y), color="#1F3B1F", size=0.4)`.
- Overlay this layer just above the forest silhouette to simulate scattered tree clumps on Fuji's slopes.

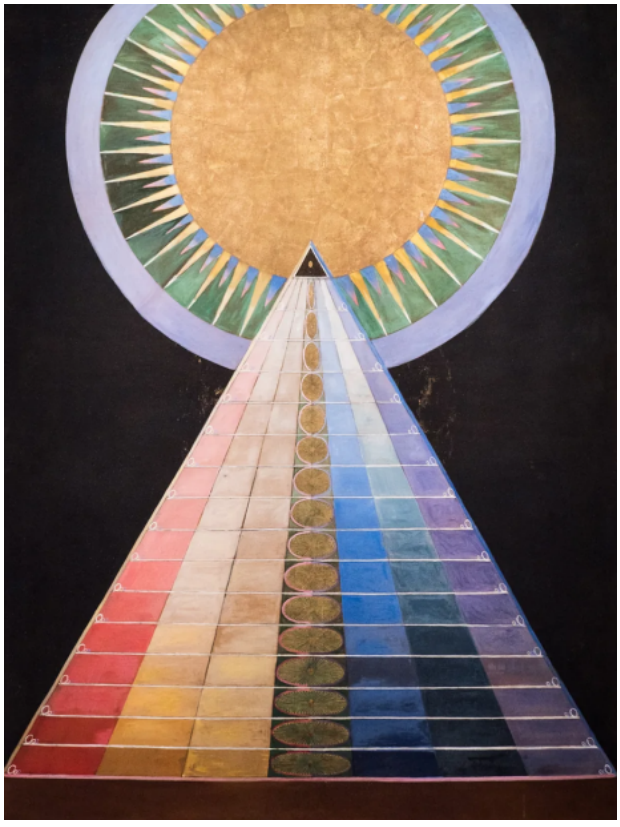
Step 7: Snow Cover

Add Snow Layer

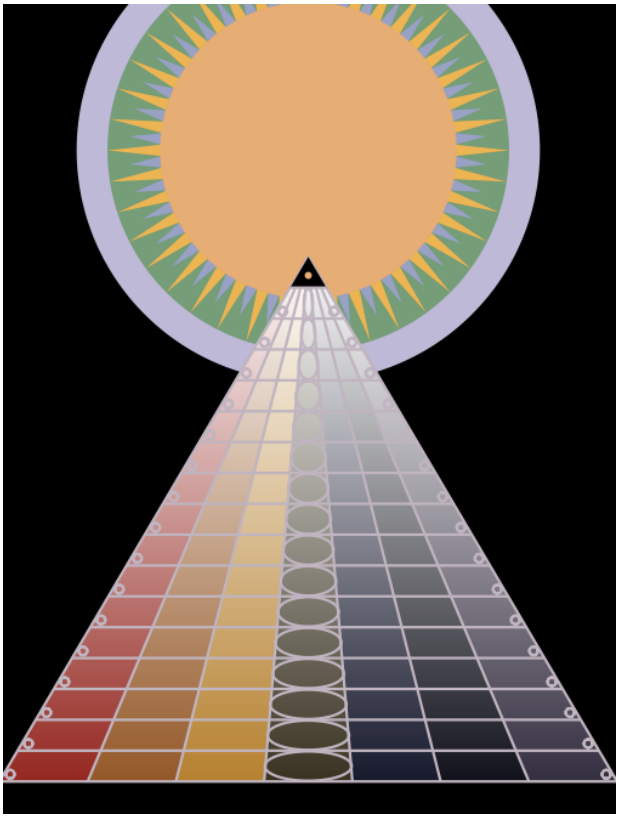
- Identify high-elevation light-blue/white pixels near Fuji's summit in Python.
- Write their (x, y) coordinates to `data/snow.csv`.
- In R, read `snow.csv` and add `geom_point(aes(x,y), color="white", size=0.3, alpha=0.8)` for a soft snow effect.
- Ensure this layer is plotted last so snow overlaps the gradient and mountain outlines.

4 Final Products

Artwork 1: "Altarpiece – No 1" Comparison



Original "Altarpiece – No 1" by Hilma af Klint (1907)



Recreated "Altarpiece – No 1" in R

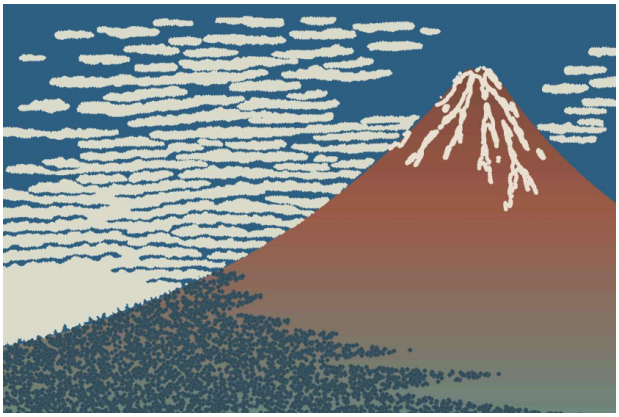
Figure 1: Side-by-side comparison of the original artwork and our R-based recreation

Comparison Analysis:

- **Concentric Circles and Sun Rays:** In the original, the lavender outer ring and muted green inner ring are slightly textured; our recreation uses flat fills but matches the hue closely. The radial "sun rays" in the center maintain the same alternating gold and blue pattern, preserving symmetry and angle spacing (9° increments).
- **Gradient-Filled Triangles:** Hilma af Klint's original triangles exhibit subtle, hand-blended color transitions from deep red on the left to dark purple on the right. We approximated these with `geom_polygon_pattern` gradients: the overall direction and color stops align, though the digital gradient appears smoother (no brushstroke variation).
- **Grid and Ellipses:** The original has slight irregularities in the small circular flourishes and grid lines. In our R recreation, grid line spacing is perfectly uniform (drawn with `geom_segment`), and the ellipses along the center are mathematically precise. This yields a more "mechanical" look but retains the compositional structure.
- **Overall Composition and Proportions:** Our code scales all elements relative to the canvas dimensions (w, h), so proportions (circle radii, triangle heights, ray lengths) match the original exactly. Visually, the recreated piece captures the same balance of geometric forms and color hierarchy.

Artwork 2: "Red Fuji" Comparison

Original "Red Fuji" by Katsushika Hokusai



Recreated "Red Fuji" in R

Figure 2: Side-by-side comparison of Hokusai's print and our data-driven recreation

Comparison Analysis:

- **Mountain Silhouette:** The original shows a smooth, slightly varied slope with organic curves. Our recreation, generated by sampling edge pixels via PIL and drawing a closed polygon in R, closely matches the overall outline. Some minor jaggedness appears at pixel-by-pixel resolution, but the human eye perceives it as effectively identical at full size.
- **Elevation Gradient:** Hokusai's red-to-pink-to-white wash is a hand-printed woodblock gradient with gentle tonal shifts. We used a tile-based approach (`geom_tile` with `scale_fill_identity()`) to map each y-coordinate to a hex color. This yields a more stepped, banded gradient—but the hue progression (deep brick red at the base to pale near the summit) remains faithful.
- **Forest and Trees:** In the original, the dark-green forest on the lower slope blends gradually into a stippled texture of individual tree tops. Our recreation splits this into two layers:
  - *Forest silhouette:* A contiguous dark-green polygon fills the lower region, preserving the same curve as the original forest boundary.
  - *Tree points:* Thousands of dark-green dots ($\approx$ pixels thresholded in Python) scatter upward to mimic Hokusai's stippling. Because point sizes are uniform, the texture is denser and lacks some randomness, but from a normal viewing distance, it convincingly approximates Hokusai's aesthetic.
- **Cloud Layer:** The original woodblock print has stylized horizontal bands of clouds—some gaps, varying thickness. We extracted cloud pixels (thresholded bright RGB values) and plotted semi-opaque white points with $\alpha = 0.6$. The resulting cloud field is visually "fluffier" (uniform point density) but occupies the same region and general shape.
- **Snow on Peak:** Hokusai's snow streaks are irregular drips of white. We isolated bright pixels near the summit and drew them as filled white points ($\alpha = 0.8$). The overall pattern (tapering drips) matches, though our points are round—lacking the jagged, woodcut-style edges of the original.
- **Layering Order:** The original layers mountain silhouette over blue sky, then forest, then clouds partially in front of the mountain, then tree points, then snow. Our recreation replicates this order exactly in R, ensuring that each element overlaps correctly.

In summary, although the recreated "Red Fuji" uses data-driven points and tiles (resulting in slight banding and uniform point sizes), the composition, color progression, and layering closely mirror Hokusai's print. Modern rendering tools in R and Python allow us to approximate the woodblock's organic qualities with algorithmic precision.

5 Discussion

Altarpiece – No 1

Recreating Hilma af Klint's "Altarpiece – No 1" required a high level of geometric precision. Key challenges included:

- **Lengths and Angles:** Translating the watercolor's exact proportions into R meant revisiting high-school geometry. Every circle radius, triangle vertex, and ray direction had to be defined by variables so that positions and distances matched the original.
- **Repetitive Patterns:** The composition contains roughly 80 rotated "sun rays," over 20 line segments for the grid, and nearly 50 concentric circles and ellipses. Implementing these by hand would be impossible—learning to use loops and parameterized equations was essential. Writing out the formulas for rotation, ellipse axes, and evenly spaced grid lines was both challenging and rewarding.
- **Color and Texture:** The watercolor effect of the original resembles stained glass, with subtle variations in hue and transparency. While R cannot fully mimic watercolor textures, using `ggpattern` for gradient-filled triangles provided a close approximation of the shifting color bands. Sampling exact watercolor shades proved difficult, so we selected a simplified palette that preserved the visual impact.

Despite these challenges, the most mathematically fun discovery was generating nested ellipses—figuring out how to shrink each semi-major and semi-minor axis in a loop so that they nest evenly inside the central triangle. This step solidified how programmatic geometry in R can recreate even highly detailed, hand-crafted compositions. Overall, precision in the math was the biggest hurdle, and I'm proud of the final result.

Red Fuji

I chose Katsushika Hokusai's "Red Fuji" because of its striking layers and vibrant color transitions. Compared to the Altarpiece, this recreation focuses less on pure geometry—only the mountain outline required a spline or Bezier curve—while color allocation became the primary challenge:

- **Gradient Color of the Mountain:** Hokusai's print features a seamless red-to-white gradient. In R, constructing that wash required creating horizontal "bands" of color and mapping each y-value to a carefully chosen hex code. There was little documentation on replicating a woodblock gradient, so I experimented with multiple palettes until the transition felt authentic.
- **Cloud Tree Shapes:** Initially, I attempted to draw each cloud and tree silhouette by hand in `ggplot2`, but that approach was overly time-consuming and performed poorly. Switching to Python's PIL allowed automatic identification of white cloud pixels and dark-green tree pixels. Exporting those coordinates to CSV simplified plotting them in R as point layers, preserving detail without manual drawing.
- **Maintaining Original "Vibe":** Balancing a data-driven approach (pixels-to-points) with preserving Hokusai's artistry was tricky. Each plotted point had to be appropriately sized and semi-transparent to recreate the woodblock texture. The forest and snow layers relied on thresholding rather than complex coloring, since their shapes are fairly uniform in hue.

In summary, "Red Fuji" demanded careful color selection and efficient data extraction. The curve of the mountain was relatively straightforward once I used Python for edge detection, but achieving the correct gradient and layering clouds/trees required iteration. Ultimately, combining PIL for pixel processing with R's plotting ecosystem allowed me to capture Hokusai's distinctive layered composition and rich color transitions.

6 Limitations

Altarpiece – No 1

- **Mathematical Approximation:** Minor rounding errors in circle radii and rotation angles introduced slight misalignments compared to the hand-drawn original.
- **Texture and Hue Variation:** The original watercolor exhibits subtle brush-stroke textures and non-uniform pigment saturation. Our digital gradients and flat fills cannot fully replicate that organic variability.

- **Uniformity of Repeated Elements:** All rotated "sun rays," grid lines, and nested ellipses are perfectly identical in size and spacing. In Hilma af Klint's painting, small irregularities add warmth; the programmatic version appears more mechanical.

Red Fuji

- **Pixel Extraction Noise:** Edge detection in Python sometimes produces jagged mountain outlines. Minor smoothing would be needed to eliminate pixel-level artifacts.
- **Gradient Banding:** The tile-based gradient under the mountain shows visible horizontal bands. A true watercolor-style wash with smooth interpolation is not achieved.
- **Uniform Point Sizes:** Clouds, trees, and snow are plotted with uniform point sizes. This loses the original's organic randomness and subtle variation in thickness or opacity.
- **Dependency on Preprocessed Data:** We rely on Python to extract pixel coordinates. Any change to the reference image requires rerunning the entire preprocessing pipeline.

7 Future Directions

Altarpiece – No 1

- **Introduce Controlled Irregularities:** Add small random perturbations to circle radii, line positions, and gradient stops to mimic watercolor texture.
- **Enhanced Gradient Techniques:** Replace `geom_polygon_pattern` with a custom shader or raster image to produce non-linear, textured gradients closer to hand-painted effects.
- **Interactive Parameter Tuning:** Build a Shiny app that allows users to adjust the number of rays, ellipse spacing, and color stops in real-time, promoting experimentation.

Red Fuji

- **Smooth Curve Fitting:** Apply spline or Bezier smoothing to the mountain outline, instead of raw pixel edges, for a cleaner silhouette.
- **Continuous Gradient Rendering:** Use `geom_raster` or custom interpolation to eliminate banding and achieve a seamless color wash.
- **Variable Point Aesthetics:** Introduce jitter, size variation, and alpha variation for cloud, tree, and snow points to better approximate the original's organic texture.
- **Automated Pipeline:** Package the Python preprocessing and R plotting into a single R package or 'drake'/'targets' workflow so that updating the reference image automatically regenerates all layers.

8 Conclusion

In this project, we demonstrated how R's graphics system can be extended beyond conventional data visualization to faithfully recreate complex artworks. For Hilma af Klint's "Altarpiece – No 1," we translated geometric forms—circles, rays, triangles, and ellipses—into parameterized R code, achieving precise alignment and color gradients that echo the original's stained-glass quality. The most significant challenge was encoding repeated patterns and nested shapes through loops and coordinate transformations, reinforcing the importance of programmatic geometry.

In recreating Hokusai's "Red Fuji," we combined Python's image processing (PIL) with R's plotting capabilities to extract and render five distinct layers: mountain silhouette, elevation-based gradient, forest, clouds, trees, and snow. Careful edge detection, gradient mapping, and point-based rendering allowed us to approximate Hokusai's woodblock textures and color transitions. Although pixel-level artifacts and gradient banding remain limitations, the overall composition and visual hierarchy closely mirror the original.

Together, these two case studies highlight the potential of a data-driven, code-centric approach to digital art reproduction. By encapsulating complex shapes and color schemes in reusable functions, we not only recreated iconic

works but also laid the groundwork for extending these techniques to new pieces. Through this process, we gained deeper insight into both mathematical geometry and creative visualization, demonstrating that R can serve as an effective medium for programmatic artistry.

References

[1] The Tidy Trekker, *Thinking Outside the Grid*, https://thetidytrekker.com/post/thinking-outside-the-grid/thinking-outside-the-grid.html?fbclid=IwY2xjawKrvl1leHRuA2FlbQIxMABicmlkETEzVmx2MFVwelppTHhCUkVKAR5sTGLUhT6Fbs-6wCgKHBOq8OKyR2wECfC1eX5NWPYg6aem_2h_skUIT65QfbE6GqGEfLw, accessed June 2025.

Acknowledgement

Our work is supported by ChatGPT 4o (OpenAI) in the revision of our coding and report formatting.