VISUALIZING MODULAR CURVES

DAVID LOWRY-DUDA

Abstract. This is a short note on how I produced the initial visualizations for modular curves in the LMFDB. At the second modular curves workshop at MIT last month, JV and AS suggested that I make *badges* for curves, similar to how each modular form has a *badge* or *portrait*.

1. Translations of the Fundamental Domain

The idea is simple. Each modular curve comes from a subgroup H of $\mathrm{GL}(2, \mathbb{Z}/N\mathbb{Z})$ for some N that we call the *level*. In short, we choose a set of coset generators for $H \cap \mathrm{SL}(2, \mathbb{Z}/N\mathbb{Z})$ in $\mathrm{SL}(2, \mathbb{Z}/N\mathbb{Z})$ and form a portrait from the translations of the standard fundamental domain of $\mathrm{SL}(2, \mathbb{Z})\backslash\mathcal{H}$.

In a bit more detail, we consider the intersection $H \cap \mathrm{SL}(2, \mathbb{Z}/N\mathbb{Z})$. Let $\{\overline{\gamma}\}$ denote a set of coset representatives for $H \cap \mathrm{SL}(2, \mathbb{Z}/N\mathbb{Z})$ inside $\mathrm{SL}(2, \mathbb{Z}/N\mathbb{Z})$. We lift each of these $\overline{\gamma}$ to a $\gamma \in \mathrm{SL}(2, \mathbb{Z})$. Thus we obtain a set of lifts of coset representatives $\{\gamma\}$, where each $\gamma \in \mathrm{SL}(2, \mathbb{Z})$.

Let $\mathcal{F} := \{z = x + iy \in \mathcal{H} : -\frac{1}{2} \leq x \leq \frac{1}{2}, |z| \geq 1\}$ denote the standard fundamental domain of $\mathcal{H}$ under the action of $\mathrm{SL}(1, \mathbb{Z})$. We show $\mathcal{F}$ on the Poincaré disk in Figure 1.

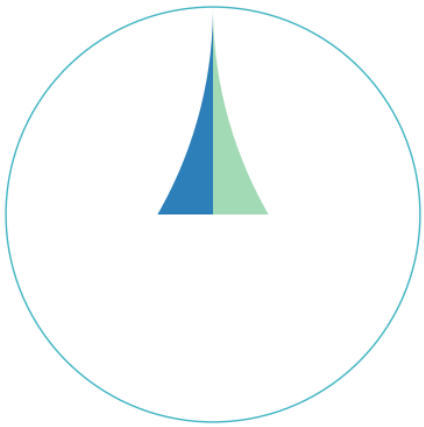

Figure 1. The standard fundamental domain $\mathcal{F}$ on the Poincaré disk.

Date: December 7, 2022.

This work was supported by the Simons Collaboration in Arithmetic Geometry, Number Theory, and Computation via the Simons Foundation grant 546235.

To the subgroup H, we associate the image from

$$\bigcup_{\gamma} \gamma(\mathcal{F}),$$

where γ ranges over the lifts of the coset representatives $\{\overline{\gamma}\}$.

Example 1. Consider the modular curve $X_{\mathrm{ns}}(2)$, which has label `2.2.0.1` in the LMFDB[1]. This corresponds to the subgroup $H = \left\langle \left(\begin{smallmatrix} 1 & 1 \\ 1 & 0 \end{smallmatrix}\right) \right\rangle \subset \mathrm{GL}(2, \mathbb{Z}/2\mathbb{Z})$. One can compute that

$$H \cap \mathrm{SL}(2, \mathbb{Z}/2\mathbb{Z}) = \left\{ \left(\begin{smallmatrix} 1 & 1 \\ 1 & 0 \end{smallmatrix}\right), \left(\begin{smallmatrix} 0 & 1 \\ 1 & 1 \end{smallmatrix}\right), \left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right) \right\}.$$

A set of coset representatives for $H \cap \mathrm{SL}(2, \mathbb{Z}/2/\mathbb{Z})$ is given by the two matrices $\left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right), \left(\begin{smallmatrix} 1 & 1 \\ 0 & 1 \end{smallmatrix}\right)$, which we can think of as being in $\mathrm{SL}(2, \mathbb{Z})$ by taking the most obvious lifts. These matrices have the typical names I (the identity) and T (translation). The identity matrix fixes $\mathcal{F}$, while T shifts $\mathcal{F}$ to the right by one. On the Poincaré disk model, combining these two translations of $\mathcal{F}$ to give Figure 2.

FIGURE 2. Visualization for `2.2.0.1`. We can plainly see $\mathcal{F}$ and a single translation to the right in this image.

While this basic strategy works for all sufficiently simple modular curves, there are challenges that require slightly different approaches. For example, for some modular curves the index of $H \cap \mathrm{SL}(2, \mathbb{Z}/N\mathbb{Z})$ inside $\mathrm{SL}(2, \mathbb{Z}/N\mathbb{Z})$ is *extremely* large, greater than 10000. In practice it is not feasible or worth the effort to include all of these cosets in the image badges.

In the rest of this document, we describe the implementation and some of the design choices that went into the initial generation of images.

[1] https://blue.lmfdb.xyz/ModularCurve/Q/2.2.0.1/

2. Finding coset representatives

We construct a list $\mathcal{L} \subset \mathrm{SL}(2,\mathbb{Z})$ that descends to $\overline{\mathcal{L}} \subset \mathrm{SL}(2,\mathbb{Z}/N\mathbb{Z})$, a set of coset representatives for $H \cap \mathrm{SL}(2,\mathbb{Z}/N\mathbb{Z})$ inside $\mathrm{SL}(2,\mathbb{Z}/N\mathbb{Z})$. A random set of coset representatives $\{\overline{\gamma}\} \subset \mathrm{SL}(2,\mathbb{Z}/N\mathbb{Z})$ would naturally lift to elements $\{\gamma\} \subset \mathrm{SL}(2,\mathbb{Z})$ whose natural translations $\gamma(\mathcal{F})$ would have an irregular, disconnected, malformed appearance. We want to construct $\mathcal{L}$ such that the corresponding domain $\bigcup_{\gamma\in\mathcal{L}} \gamma(\mathcal{F})$ is connected, and more generally "as simple as possible."

To do this, we adapt a well-known algorithm to compute a fundamental domain for a congruence subgroup $\Gamma \subset \mathrm{SL}(2,\mathbb{Z})$, or equivalently to compute a set of coset representatives for Γ in $\mathrm{SL}(2,\mathbb{Z})$. Recall that $\mathrm{SL}(2,\mathbb{Z})$ is generated by two matrices,

$$\mathrm{SL}(2,\mathbb{Z}) = \langle T, S\rangle, \qquad T = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix},\ S = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}.$$

Recognizing a translation $\gamma\mathcal{F}$ of the standard fundamental domain $\mathcal{F}$ as a hyperbolic triangle, the three adjacent hyperbolic triangles are precisely $T(\gamma\mathcal{F}), T^{-1}(\gamma\mathcal{F})$, and $S(\gamma\mathcal{F})$.

We can guarantee that $\bigcup \gamma(\mathcal{F})$ will be connected if we recursively add elements corresponding to hyperbolic triangles at the boundary of the current fundamental domain. The idea is to construct these matrices in $\mathrm{SL}(2,\mathbb{Z})$ recursively, while verifying that the projections to $\mathrm{SL}(2,\mathbb{Z}/N\mathbb{Z})$ are independent.

In the following algorithm, we use I, S, T to mean the identity, inversion, and translation matrices in $\mathrm{SL}(2,\mathbb{Z})$ as above.

Algorithm 1 Complete coset representative construction.

Initialize a queue `queue` with the identity matrix $\{I\}$.
Initialize empty list `cosetreps`. ▷ representatives in $\mathrm{SL}(2,\mathbb{Z}/N\mathbb{Z})$.
Initialize empty list `cosetlifts`. ▷ lifted representatives in $\mathrm{SL}(2,\mathbb{Z})$.
while `queue` is nonempty **do**
    Remove the next element of `queue`. Call it γ.
    Compute $\overline{\gamma}$, the projection of γ in $\mathrm{SL}(2,\mathbb{Z}/N\mathbb{Z})$.
    if $\overline{\gamma}$ does not generate the same coset as any $\overline{\tau} \in$ `cosetreps` **then**
        Add $\overline{\gamma}$ to `cosetreps`. ▷ Found a new coset representative.
        Add γ to `cosetlifts`.
        Add $\gamma \cdot T$, $\gamma \cdot T^{-1}$, and $\gamma \cdot S$ to `queue`. ▷ Consider all new adjacent triangles.
    end if
end while

Lemma 2. *At the end of Algorithm 1,* `cosetreps` *is a complete list* $\overline{\mathcal{L}}$ *of coset representatives of* $H \cap \mathrm{SL}(2,\mathbb{Z}/N\mathbb{Z})$ *in* $\mathrm{SL}(2,\mathbb{Z}/N\mathbb{Z})$*, and* `cosetlifts`

is a list $\mathcal{L}$ of lifts of these coset representatives such that $\bigcup_{\gamma\in\mathcal{L}}\gamma(\mathcal{F})$ is connected.

Proof. It is clear that each $\gamma \in \mathcal{L}$ is a lift of a corresponding $\overline{\gamma} \in \overline{\mathcal{L}}$, as each $\overline{\gamma}$ is constructed from γ in the algorithm. The fact that $\mathcal{L}$ $\bigcup_{\gamma\in\mathcal{L}}\gamma(\mathcal{F})$ is connected the fact that each γ is built recursively from triangles adjacent to earlier triangles in line 10 of Algorithm 1.

It only remains to show that $\overline{\mathcal{L}}$ is a complete list of coset representatives of $H \cap \mathrm{SL}(2,\mathbb{Z}/N\mathbb{Z})$ in $\mathrm{SL}(2,\mathbb{Z}/N\mathbb{Z})$.

To prove this, suppose that the algorithm has terminated. Note that line 7 guarantees that each $\overline{\gamma} \in \overline{\mathcal{L}}$ corresponds to different cosets. Consider a coset $\overline{\gamma'}(H \cap \mathrm{SL}(2,\mathbb{Z}/N\mathbb{Z}))$. We will show that this coset is represented in $\overline{\mathcal{L}}$. Among the representatives for this coset, we (lightly abuse notation and) fix $\overline{\gamma'}$ to be one having a lift $\gamma' \in \mathrm{SL}(2,\mathbb{Z})$ with minimum word length in $\{S, T, T^{-1}\}$, i.e. such that

$$\gamma' = \prod_{1\leq i\leq n} \tau_i \qquad (\text{where } \tau_i \in \{S, T, T^{-1}\})$$

and where no other representative has a lift that can be written as a product of less than n terms in $\{S, T, T^{-1}\}$. We also fix a representation for γ' as above. For each $0 \leq j \leq n$, define

$$\gamma_i := \prod_{1\leq i\leq j} \tau_i$$

to be the jth partial product (and let $\gamma_0 = I$ be the identity matrix). And for each γ_i, let $\overline{\gamma_i}$ denote the image in $\mathrm{SL}(2,\mathbb{Z}/N\mathbb{Z})$.

Let m denote the largest index such that $\overline{\gamma_m}$ is equivalent to a found coset representative $\overline{\gamma} \in \overline{\mathcal{L}}$. Note that m exists and $m \geq 0$, as I is always the first found representative. If $m < n$, then $\overline{\gamma_m}$ is equivalent to some $\overline{g}$, a coset representative in $\overline{\mathcal{L}}$, but $\overline{\gamma_{m+1}} = \overline{\gamma_m\tau_{m+1}}$ is not. But as $\overline{g} \in \overline{\mathcal{L}}$, lifts gT, gT^{-1}, gS were added to `queue` in line 10. Note that $\overline{\gamma_{m+1}}$ necessarily generates the same coset as one of $\overline{gT}, \overline{gT^{-1}}$, or $\overline{gS}$. Hence when this representative in the queue is processed, line 7 guarantees that a coset representative equivalent to $\overline{\gamma_{m+1}}$ was either added to $\overline{L}$, or that it was already equivalent. Thus we conclude that $m = n$, and every coset is represented in $\overline{\mathcal{L}}$. □

Remark 3. Small variations of this algorithm allow one to track all sorts of data on the graph. For example, maintaining a record of when $\overline{\gamma'\tau_i}$ is equivalent to some other $\overline{\gamma}$ would allow one to track which edges of the plotted domain are equivalent, precisely as with standard plots of fundamental domains.

In practice, the final plots we produce will be under 400×400 pixels. This is a reasonable size that fits in a sidebar, but is large enough to see some details. But this also means that there isn't enough space to see an enormous number of translates of the fundamental domain.

For the purpose of making nice visualizations, it suffices to identify the "first" several coset representatives. Ideally, we would identify all cosets whose hyperbolic triangle translates of $\mathcal{F}$ are sufficiently large (in the Euclidean sense). But this is challenging without generating all the cosets. Instead, we follow the heuristic that cosets formed from long words in T, T^{-1}, S typically lead to smaller translates of $\mathcal{F}$ than cosets formed from short words.

This is why in practice we use a FIFO (first-in-first-out) queue in line 5, and when we enqueue new elements in line 10 we append these elements (in the order given) at the end of the queue. The effect is that we find a set of coset lifts $\mathcal{L}$ consisting of elements γ formed from relatively short products of T, T^{-1}, and S. In the LMFDB, we compute the "first" 384 coset representatives in this way.

3. Making visualizations

We now assume that we have a set of coset representatives $\overline{\mathcal{L}} \subset \mathrm{SL}(2, \mathbb{Z}/N\mathbb{Z})$ and a nice set of lifts $\mathcal{L} \subset \mathrm{SL}(2, \mathbb{Z})$, as given in §2.

3.1. **The disk model.** We use the disk model instead of the more common upper half-plane model because it is a finite, bounded representation. All images fit naturally into the same space to facilitate comparison. This is the same reasoning behind the decision to use the disk model for portraits for modular forms in the LMFDB [BBB+21, LD21].

We use the same Cayley transformation taking $\mathcal{H}$ to the disk $\mathbb{D}$ as with modular form portraits:

$$\mu(z) = \frac{z - i}{-iz + 1}.$$

This preserves apparent vertical orientation of the imaginary axis in both models: the points $\{0, i, i\infty\} \in \mathcal{H}$ are sent to $\{-i, 0, i\} \in \mathbb{D}$, vertical and centered. This choice of center point in the disk means that the standard fundamental domain $\mathcal{F}$ has obvious placement in the top half of $\mathbb{D}$.

3.2. **Making plots.** In the Poincaré disk model, geodesics are either portions of straight lines or arcs of circles that intersect the unit circle orthogonally. A hyperbolic triangle has three geodesic edges. The fundamental domain $\mathcal{F}$ for $\mathrm{SL}(2, \mathbb{Z})\backslash\mathcal{H}$ has corners $i, \mu(0.5 + i\sqrt{3}/2) \approx 0.2679$, and $\mu(-0.5 + i\sqrt{3}/2) \approx -0.2679$. As in Figure 1, we split the fundamental domain into two halves. This makes each translation $\gamma(\mathcal{F})$ easier to see.

To make the plots, I assume that we have a procedure `HyperbolicPolygon` that takes three points A, B, C, a color c, and an α for transparency, and plots the hyperbolic triangle with vertices A, B, C in color c with alpha value α. I also assume we have the Cayley map μ as above, and a way to compute the action of a matrix γ on an element z. These are each straightforward to implement. This leads to Algorithm 2.

Algorithm 2 Plotting routine.

Let `cosetlifts` denote the nice lifts in $\mathrm{SL}(2,\mathbb{Z})$.
Set `total` to be the number of elements in `cosetlifts`.
Fix `color1` and `color2`.
$A \leftarrow i$.
$B \leftarrow \frac{1}{2} + i\frac{\sqrt{3}}{2}$.
$C \leftarrow -\frac{1}{2} + i\frac{\sqrt{3}}{2}$.
$D \leftarrow i\infty$.
for `idx` $:= 0$ up to `total` **do**
    $\gamma \leftarrow$ `cosetlifts[idx]`.
    $a \leftarrow \mu(\gamma A)$.
    $b \leftarrow \mu(\gamma B)$.
    $c \leftarrow \mu(\gamma C)$.
    $d \leftarrow \mu(\gamma D)$.
    $\alpha \leftarrow$ `setalpha(idx, total)`.
    `HyperbolicPolygon(a, b, d, color1,` α`)`.
    `HyperbolicPolygon(a, c, d, color1,` α`)`.
end for

Where `setalpha` is the following short function

function SETALPHA(`idx, total`)
    if `idx` $<= 128$ **then** ▷ $\alpha = 1$ for the first 128 cosets.
        $\alpha \leftarrow 1$
    else if `idx` $<= 384$ **then** ▷ α tapers towards 0 as `idx` nears 384.
        $\alpha \leftarrow 1 - (\texttt{idx} - 128)/256$
    else
        $\alpha \leftarrow 0$
    end if
    return α
end function

We show a couple of sample visualizations for `8.24.1.13`[2] and `10.72.1.1`[3] created using this algorithm in Figures 3 and 4.

3.3. **Additional practical considerations.** We comment on a few of the other practical decisions.

Intersections in $\mathrm{PSL}(2,\mathbb{Z})$, and consequences. It is worth noting that the matrices γ and $-\gamma$ have the same action on z. Thus it suffices to work over $\mathrm{PSL}(2,\mathbb{Z})$ when producing plots.

These visualizations capture information about the projection/intersection of modular curves with $\mathrm{PSL}(2)$. This loosely represent some aspects of the

[2] https://blue.lmfdb.xyz/ModularCurve/Q/8.24.1.13/
[3] https://blue.lmfdb.xyz/ModularCurve/Q/10.72.1.1/

FIGURE 3. Visualization for modular curve `8.24.1.13`.

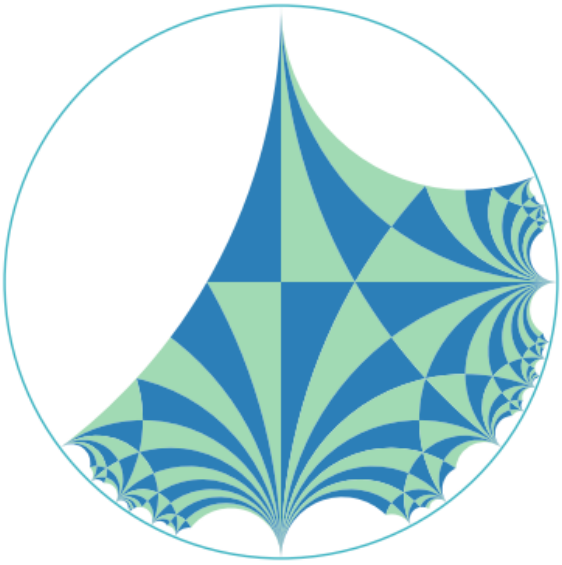

FIGURE 4. Visualization for modular curve `10.72.1.1`.

curve, but omits much information. When two curves have the same projection/intersection, they will have the same picture. We identify these classes of curves and generate one image for each intersection class. We do this using stored `CPlabels` in the LMFDB initially.

This *drastically* cuts down the number of necessary images to tens of thousands, as opposed to millions and millions. This has far-reaching effects: we can afford to make the pictures slightly larger if desired, and it's not necessary to make the visualizations particularly efficiently.

To make the first 3000 pictures took a combined total of approximately 40 hours of computation time, which is on the order of one minute per picture. Essentially all the computation time goes into finding cosets, and essentially all the time in finding cosets goes into determining whether two given elements generate the same coset. Determining subgroup membership in high index subgroups of $\mathrm{SL}(2, \mathbb{Z}/N\mathbb{Z})$ can be time consuming.

Color choices. To make the plots, we use the two colors `#A1DAB4` (light green) and `#2C7FB8` (dark blue). There are blue-green colors near the current two dominant blues of the LMFDB, `#90CAF9` (blue) and `#E3F2FD` (light blue). This pair of colors has good contrast in hue, saturation, and brightness — and thus retain most of their structure even to those with many different types of vision.

Appendix A. Space efficient SVGs - the path not traveled

Finally, we note an alternative, space efficient visualization.

The visualizations produced consist of a finite number of circular arcs and line segments. These are both primitive elements in SVG (scalable vector graphics) files. To produce a high fidelity image, one could then create SVG files.[4]

In addtition, a carefully made SVG is smaller than a raster, even at the 400×400 pixel scale.

A particularly efficient, simple badge comes from visualizing the silhouette of the plots described above. In this way, the visualization ultimately consists of two paths: a path consisting of arcs and line segments, and the bounding unit circle. To make this visualization, it suffices to compute the boundary edges, orient them, and connect them in an SVG file.

In principle this can be done very similarly as in Remark 3, a small modification of Algorithm 1. This amounts to tracking for each coset what it's neighbors are, and whether the neighbors are in the list of cosets (i.e. plotted) or equivalent to those in the list of cosets (i.e. at the boundary). In cases where we truncate the number of cosets, it is also necessary to track neighbors that haven't been examined (i.e. also at the boundary, possibly for truncation reasons).

But in practice it's not hard to generate the boundary from the generated list of cosets given by Algorithm 1. The idea is to study the translations of the three edges of the base fundamental domain $\mathcal{F}$. Collect all translations obtained from act on all of these edges by all of the cosets — any edge that

[4]One reason we didn't do this initially is because we already had the infrastructure for storing and displaying PNG rasters, and these visualizations act more like *badges* and less like *ids*.

appears twice is not a boundary edge, and any edge that appears exactly once is on the boundary.

This gives an (unordered) list of boundary edges. To order the list, it suffices to start from any edge and follow tail to head. We note that the order of coset generation guarantees that each vertex will never occur in more than 2 boundary edges, and thus the naive algorithm suffices.

Require: $\mu(z)$, the Cayley map. described above.
function ComputeBoundaryEdges(`cosets`)
    $B \leftarrow \frac{1}{2} + i\frac{\sqrt{3}}{2}$.
    $C \leftarrow -\frac{1}{2} + i\frac{\sqrt{3}}{2}$.
    $D \leftarrow i\infty$.
    Initialize empty list `edges`.
    for $\gamma \in$ `cosets` **do**
        $e_1 \leftarrow (\mu(\gamma B), \mu(\gamma C))$.
        $e_2 \leftarrow (\mu(\gamma B), \mu(\gamma D))$.
        $e_3 \leftarrow (\mu(\gamma C), \mu(\gamma D))$.
        for $e_i \in (e_1, e_2, e_3)$ **do**
            if $e_i \in$ `cosets` **then** Remove e_i from `edges`.
            else Add e_i to `edges`.
            end if
        end for
    end for
    return `edges`.
end function

function OrderBoundaryEdges(`edges`)
    Remove first element of `edges` and call it `edge`.
    Initialize list `orderedEdges` to be `[edge]`.
    while `edges` is nonempty **do**
        `lastVertex` $\leftarrow$ the second vertex in last edge in `orderedEdges`.
        Find `newEdge` in `edges` that contains `lastVertex`.
        Remove `newEdge` from `edges` and add it to `orderedEdges`, reversing its orientation if necessary so that `lastVertex` is its first vertex.
    end while
    return `orderedEdges`.
end function

Finally, one produces the SVG. A minor complicating factor with SVG arcs is that they take the two endpoints, the radius of the circle (or rather the radii of the ellipse and a rotation for the ellipse), and two flags indicating which of the four potential arcs to draw that connect these two points. These four potential arcs come from the two circles with the given radius that connect the two endpoints, and each circle has a long arc and a short

arc. For these visualizations, we always want the short arc. But the other flag, called the `sweep flag`, is slightly more complicated.

To compute the `sweep flag`, it is helpful to introduce an auxiliary point `via` that represents a point approximately on the intended arc. The key observation here is that for all geodesic arcs on the Poincaré disk, we can take `via` to be a point a bit closer to the center of the disk, and this will always give the correct sweep direction.

Require: `real(z)` and `imag(z)`, giving the real and imaginary parts of z.
Require: `atan2(y, x)`, as in the C standard library.

function ComputeSweep(`start, end`)
    `via` $\leftarrow$ `(start + end) / 3`.
    `SE` $\leftarrow$ `end - start`.
    `SV` $\leftarrow$ `via - start`.
    $\theta \leftarrow$ `atan2(imag(SE), real(SE)) - atan2(imag(SV), real(SV))`.
    Normalize θ to be between $-\pi$ and π.
    if $\theta < 0$ **then**
        return 1.
    else
        return 0.
    end if
end function

Taken together, these steps form the SVG plotting algorithm in Algorithm 3. The visualizations in Figures 5 and 6 depict the same modular curves as in Figures 3 and 4, but using this simplified SVG silhouette presentation.

Although these SVGs very clearly show less information, the primary advantage is that each take less than 1KB of space! That's *extremely* space efficient.

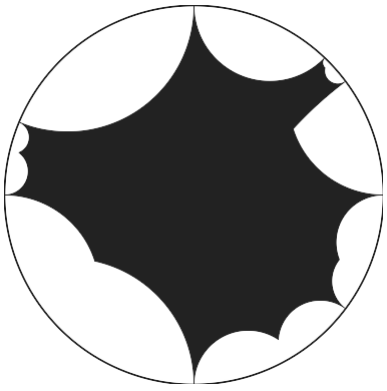

FIGURE 5. Visualization for modular curve `8.24.1.13`.

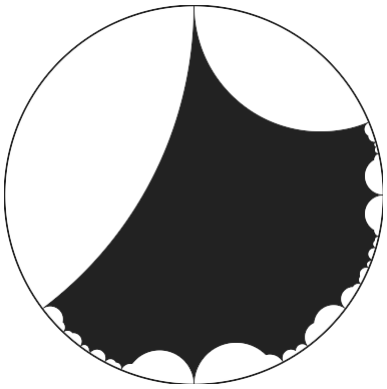

FIGURE 6. Visualization for modular curve `10.72.1.1`.

Algorithm 3 SVG creation

Require: `orderedEdges`, the ordered list of boundary edges from above.
Require: `svgline(start, end)`, a routine to add an svg line between the two given points.
Require: `svgarc(start, end, radius, sweep)`, a routine to add an svg arc as
Require: `ComputeRadius(start, end)`, giving the (Euclidean) radius of the geodesic between `start` and `end`.
Require: `ComputeSweep(start, end)` as above.

```
Initialize empty list svgPath.
for edge in orderedEdges do
    Write edge as (start, end)
    if edge is a line segment then
        Append svgline(start, end) to svgPath.
    else
        radius ← ComputeRadius(start, end).
        sweep ← ComputeSweep(start, end).
        Append svgarc(start, end, radius, sweep) to svgPath.
    end if
end for
```

REFERENCES

[BBB+21] Alex J. Best, Jonathan Bober, Andrew R. Booker, Edgar Costa, John E. Cremona, Maarten Derickx, Min Lee, David Lowry-Duda, David Roe, Andrew V. Sutherland, and John Voight.

Computing classical modular forms. In *Arithmetic geometry, number theory, and computation*, Simons Symp., pages 131–213. Springer, Cham, [2021] ©2021.

[LD21] David Lowry-Duda. Visualizing modular forms. In *Arithmetic geometry, number theory, and computation*, Simons Symp., pages 537–557. Springer, Cham, [2021] ©2021.