The Myst Graph, 3: Creating the Graph Using DeMystify

How DeMystify turns HyperCard stacks into the graph

Guillaume Lethuillier
glthr.com

May 9, 2025 · updated November 20, 2026

Series: The Myst Graph — Part 3 of 3
Next: The Riven Graph, 1: The 1997 Game as a Graph (glthr.com/riven-graph-1)
Interactive browser: glthr.com/myst-graph
All 5 Myst games: glthr.com/myst-graphs (Myst, Riven, Exile, Revelation, End of Ages)

Abstract

DeMystify is the open-source Go program that builds the Myst Graph from HyperCard cards and scripts, analyzes its connections and renders PDF and browser views. This third article explains the original XML-based workflow and its later changes: DeMystify now reads the six Age stacks directly, while retaining XML input. Its analysis identifies structural references between cards; it does not execute the complete game logic or encode gameplay requirements into the edges.

> **Revised (November 2026):** I revised this article after agents re-examined my Myst Graph and DeMystify against the original game files, and the corrections are listed on the series page.

Card labels such as Myst:70105 have a thick underline, in the color of their Age, and open that card in the graph browser.

The Myst Graph, a graph of **Myst** (1993), is generated by DeMystify, an open-source Go program: **github.com/glthr/DeMystify**. This third article focuses on the implementation details and shares suggestions for further enhancement.

Version of the game: Myst, original Macintosh CD-ROM release (1993). The card IDs are those of its six HyperCard stacks, all dated July 1993.

Converting the Original HyperCard Files

My initial attempt to implement a parser for HyperCard files proved more challenging than expected. Creating a parser based on the unofficial file format standard (and its helpful addendum) would have been the best approach but would have required a significant effort for a single use—the Myst HyperCard files only need to be converted once, after all.

Fortunately, **stackimport**, a C++ library, demonstrated its ability to reliably convert HyperCard files into XML (and PBM) files. DeMystify first leveraged this ability and therefore required converting the original files beforehand into XML files. The path to these files must be provided as a command-line argument. I do not

(Update November 2026) DeMystify now reads the six HyperCard stacks directly; converted XML directories remain supported. The reusable Go reader proposed here is in the repository: it loads cards,

Cite as: Guillaume Lethuillier, "The Myst Graph, 3: Creating the Graph Using DeMystify", glthr.com, May 9, 2025. https://glthr.com/myst-graph-3

provide these files in the repository as they *may* be copyrighted: converting the original source code is left as an exercise to the reader. (More recently, Andrew Plotkin's mystextract has offered another way to extract these files, and makes Myst's scripts publicly available.)

Parsing the Converted HyperCard Files

Stacks

The DeMystify parser package translates these XML files into a comprehensive "proto graph" representation of HyperCard objects and their relationships. The parser logic begins by processing each stack, resulting in a slice of stack objects that contain essential information such as the stack's name (*i.e.*, its Age), processed HyperTalk scripts, and a mapping of card IDs to their corresponding names (if applicable). Each stack is also associated with a list of cards that are part of it. To illustrate this concept, let's take a look at an example:

```
. . .
<card id="8277"
      file="card_8277.xml"
      marked="false"
      name=""
      owner="2702" />
<card id="8776"
      file="card_8776.xml"
      marked="true"
      name="qbow"
      owner="2702" />
. . .
```

In this example, two cards, 8277 and 8776, are part of the stack. One is marked; the other is not (this information is not used for the Myst Graph). And 8776 was named "qbow" by its creator. The parser gets "qbow" from the stack's ID-to-name map.

(Update November 2026) The name also appears in the XML file for card 8776. Of the 1,355 card files, 159 contain a non-empty card name.

Cards

After the stacks, the cards are processed. The goal is to generate a card object that contains a card ID, a reference to the stack to which the card belongs, its original name (if any) by using—as just explained—the IDs-names map populated during the analysis of the stacks, the parsed scripts, and the backgrounds—corresponding to the name of the image featured by the card. The parser reads the picture name from the background field with ID 1. For instance, the asset embedded in the following card is "Caldera 4-N":

(Update November 2026) Thus field supplies a picture name in 1,345 of the 1,355 card files. The other ten files have no background content entry.

```
. . .
<content>
    <layer>background</layer>
    <id>1</id>
    <text>Caldera 4-N</text>
</content>
```

```
<content>
    <layer>background</layer>
    <id>7</id>
    <text>card id 31046</text>
</content>
```

Furthermore, the card is internally named following the pattern "{stack}:{ID}", where "{stack}" represents the name of the stack containing the card, and "{ID}" is the unique identifier of the card, as explained in the first article.

The preprocessing of HyperTalk scripts is, at this stage, minimal. The scripts, in the `<script></script>` tags, are split by line breaks. As in SQL, scripts are commented out with `--`. When a line is prefixed this way, the command is identified as being `disabled`. The same holds for the body of a handler whose declaration line was commented out while the rest was left in place: it belongs to no handler, so HyperCard never runs it, and its commands are identified as disabled too.

References to Other Cards

Subsequently, the relationships between stacks and cards, and between cards, are inferred from the analysis of the scripts. Currently, this analysis relies on regex pattern matching to identify the links between elements. As explained in the previous articles, this analysis is not granular: it does not consider the gameplay or the scripts as a whole. It suffices for a card to point to another to identify a link between them.

HyperTalk provides a convenient mechanism for determining when a card references another card. When they belong to the same stack, the script contains a `go to card id {ID}` (where "to" is optional). When a card refers to another stack, the pattern is `go to card id {ID} of stack "{stack name}"` or `go to card "{card name}" of stack "{stack name}"`.

(Update November 2026) DeMystify now also resolves a position in the stack's own order (`go next`, `go prev`, `go first`, `go last`) against the order of the cards stored in the stack.

There is also a more subtle push/pop card mechanism. When a card is pushed (*push* command), it is stored in memory as a reference. A *pop* operation is, at least the way it has been used for Myst, equivalent to a `go to card id {ID}`, with the ID of the card that had been stored in memory. This duality characterizes two classes of relationships.

When the first card is pushed (`push card`), the second card just backtracks to the first.

However, when the first card pushes another card (`push card id {ID} of stack "{stack name}"`), a more complex transitive relationship emerges, making the player navigate from the first card to the pushed card through an intermediate card (which "pops" to the target card).

For instance, card 100894 from Channelwood Age contains this script:

```
. . .
push card id 46439 of stack "Myst"
go to card id 44018 of stack "Myst"
. . .
```

It pushes card 46439 from a different stack, then opens card 44018 belonging to the Myst Island.

The latter embeds the following script:

```
on mouseUp
htlock "nobw"
htlock true
vs up
pop card
end mouseUp
```

It just pops a card (line 5). But instead of backtracking to Channelwood Age, it goes to the card pushed last: card 46439. Therefore, the path becomes:

Channelwood Age:100894 (push Myst:46439; go to Myst:44018) → Myst:44018 (intermediary card: library up; pop) → Myst:46439 (facing the bookshelf).

Literal navigation can contain several steps inside one handler. Mechanical Age:34882 first requests card 49328, then continues to card 39710. The intermediate card has no navigation script of its own: the continuation belongs to the handler that opened it. The graph keeps both references on the calling card and also records the step from 49328 to 39710 as an automatic next step, drawn with a hollow arrowhead and kept separate from a saved return. Another caller, 16622, uses 49328 before requesting 43114. Path search does not follow automatic next steps, so one click is never counted as two and neither caller is connected to the other's destination.

The script analyzer also identifies the cards that feature pages using the following patterns:

- `put "{digit},A,0" into ALL_Page` for blue pages *(A is for Achenar)*
- `put "{digit},S,0" into ALL_Page` for red pages *(S is for Sirrus)*
- `put "Atrus" into ALL_Page` for the white page

Last, if a card refers to a card that cannot be found, a new virtual card is created. A virtual node preserves a reference to an absent target; the release files do not establish when it existed or why it is absent.

This initial approach may appear rudimentary but is well suited to the HyperTalk grammar and the way the Myst creators have used it. A welcome evolution would be to make the analysis more comprehensive to detect additional associations (typically capturing the navigation from page to page in a book, handled by image substitution). It could also, perhaps, encode the gameplay constraints into the edges.

First Layer of Abstraction (or: Creation of the Proto Graph)

The last action of the parser is to transform stacks and cards into *node* objects that contain attributes (stack or card; contains colored pages; virtual) and *edge* objects, also containing attributes (intra-Age or cross-Ages; disabled; self-reference; *etc.*) and pointing to the two nodes they link. At this point, the resulting structure is not yet a proper graph but rather a preliminary representation of the data that will eventually be used to generate an actual graph.

Note that the parser ensures that the edges are unique. A card can refer several times to another card (*i.e.*, clicking on different parts of the first card leads to the same target card). I decided, for simplicity, to merge edges having the same attributes and directionality. If DeMystify is updated to encode the gameplay into the edges, then it would make sense to unmerge these edges.

Also, the parser translates the properties of the cards and links into node and edge attributes. For a node, the base attribute is either stack or card. Then, if it contains a page, an additional attribute is the color of the page (can be cumulative: both blue and red). Edges can be `disabled` (they initially existed, but the developers disabled them by commenting them out in the script), `shadowed` (the link is in a button that lies entirely under another button, so that no click reaches it), `notImplemented` (the link connects to a node that refers to a card that does not exist in the stack), `intraAge` or `crossAge` (depending on whether they refer or not to a node from a different Age). An edge can have special attributes, such as a `backtracking` attribute (A to B, then B to A). When the edge is partially transitive (see the figure on the right of the Transitivity section of article 1), the transitivity relationship includes three nodes (D to B to A), and the parser distinguishes between the "tail" of the transitivity and its "head". The *tail* is the edge between the initial node and the intermediary node (D to B); the *head* is the edge between the intermediary node and the target node (B to A). This distinction is helpful for the rendering of the graph (no arrowhead on the tail section of the transitive edge).

(Update November 2026) The `shadowed` attribute was added after publication.

Once this process is done, the parser returns an object containing a "bag" of nodes and edges. The next step involves transforming these components into a graph.

Second Layer of Abstraction: Creation of the Myst Graph

DeMystify leverages the Gonum library to create an instance of a graph. Due to the directionality of the edges, a directed graph is necessary. Moreover, since multiple edges can coexist between two nodes for a specific direction (even when identical edges are merged, they may differ in attributes) and self-loops should also be supported, a multigraph is required. Additionally, I had to consider backtracking edges, which should be excluded during distance calculations. To accomplish this, I utilized weighted edges with regular edges assigned a value of 1 and backtracking edges given an extremely large positive value (positive infinity).

To satisfy these constraints, after evaluating alternative solutions, I determined that `multi.WeightedDirectedGraph` from Gonum was the most suitable choice. This graph type effectively accommodates both directed and multigraph structures, as well as weighted edges with distinct properties.

Creating the weighted directed multigraph is straightforward, thanks to the parser's preprocessing. Nodes are added as-is, while edges default to a weight of 1. On the other hand, backtracking edges are assigned the `math.Inf(1)` value to distinguish them from regular edges. Disabled and shadowed edges are skipped altogether and stored in a map for future rendering.

With the graph now created, I can analyze it.

Three types of analyses are performed: nodes, components, and paths.

- **Node analysis.** The node analyzer identifies key nodes based on their in-degree and out-degree. It, therefore, identifies nodes with the most *incoming* edges (by iterating over the nodes, counting the number of in-degree edges, then comparing with the current maximum), with the most *outgoing* edges, source nodes (no incoming edges from another node), sink nodes (no outgoing edges), and isolated nodes (neither outgoing nor incoming edges; they count as both sources and sinks).

- **Components analysis.** This analysis aims to identify the graph components (that is, subgraphs not connected to others). The general idea is to iterate over all nodes and, for each node, traverse (using BFS) the subgraph to which it belongs. By keeping a "map" (in fact, a slice, for making the outcome of the process deterministic) associating the subgraphs with the nodes that compose them, it is possible to identify the components.
- **Path analysis.** This analysis examines two types of paths: the shortest paths and the longest shortest paths (*i.e.*, most separated nodes). For the computation of the shortest path, Gonum provides a valuable resource: `DijkstraFrom`: "*DijkstraFrom returns a shortest-path tree for a shortest path from u to all nodes in the graph g. If the graph does not implement Weighted, UniformCost is used*". This function uses Dijkstra's algorithm under the hood, taking into consideration the weighted edge values in the graph. As a result, backtracking edges are de facto excluded.

When implementing DeMystify, I experimented with other conceptual tools, such as the identification of bridges. But for the purpose of the initial version of the Myst Graph, I found that focusing on just these three dimensions would provide interesting information without making it too complex. It provides a way to categorize the nodes from a graph theory perspective, identify the isolated subgraphs, and compute varieties of paths. I would be very interested in the readers' findings in addition to these dimensions.

At this point, the Myst Graph is analyzed. It is now time to visually render it.

Rendering the Graph

At the start of this project, I wanted to use the DOT language due to its "natural" affinity for representing graphs. Initially, I used the Gonum DOT library. Then, when things became too complex for this library, I assessed other libraries. Unfortunately, none of them were flexible enough to support the requirements: I had no choice but to develop a custom encoding mechanism using string manipulation techniques. This aspect of DeMystify lacks elegance and was a source of frustration during its implementation.

From the generated DOT object, I wanted to keep things simple for the first graph version, so I decided to generate a static file. At first, I tried generating PNG images, but that quickly became impractical because of the sheer size of the graph. Creating a PDF file proved to be a good solution, also having the benefit of being searchable. At first, I did it manually, using the DOT file as input. After experimenting with different CLI tools, I found that `neato` generated an acceptable output. As with the DOT object, I wanted to use a library to generate the PDF the way `neato` does. Unfortunately, I did not find any: consequently, DeMystify must externally call the `neato` binary.

From a visual perspective, I would like to view alternative renderings from contributors.

(Update November 2026) Rendering has since been refactored to use libraries. DeMystify also generates an offline browser view with zoom, search and Age filters. Following a suggestion on Hacker News, it can show each view's picture inside its node when readers supply images decoded from their own copy of the game; the project distributes no game assets. A

Conclusion

The initial prototyping phase focused on quickly demonstrating a working graph representation. DeMystify has since grown beyond it: it has regression tests and

independent graph and renderer checks, reads the original stacks directly and writes a browser view; external `neato` remains a dependency for the PDF, and the gameplay is not yet encoded into the edges.

Because this is a personal project with limited time commitment, I am actively seeking contributions from the open-source community. Your expertise and passion could dramatically improve DeMystify. I believe that with collaborative effort, DeMystify can truly reach its full potential.

"*Structurally, narrative shares the characteristics of the sentence without ever being reducible to the simple sum of its sentences: a narrative is a long sentence, just as every constative sentence is in a way the rough outline of a short narrative.*" Drawing on Roland Barthes' observation that narrative is more than the sum of its parts, **the Myst Graph project** explores a similar principle. It demonstrates that the relationship between game world views (the HyperCard cards, individually; the "sentences" of the game) transcends their mere aggregation: the whole (the narrative of the game) is greater than the sum of its parts (the collection of views and their own puzzles). There is an emergent narrative property at play, one that can only be captured, understood, and analyzed by considering the network-level properties (clustering, re-arrangement of edges, *etc.*).

My goal with this project has been to reveal a new understanding of Myst by studying it *structurally*, treating its components as elements within a larger system. I believe that structural analysis offers a powerful tool for video game archaeology, though its applicability is limited to specific contexts—particularly games, and, more generally, interactive media, inherently structured as a graph. *The Manhole*, released in 1988 by the Millers, could perhaps be a natural application for this approach, mirroring *Myst*'s foundation in interlinked cards. This approach could be applied to any sufficiently complex application built on HyperCard—a technology created by Bill Atkinson, whose passing in June 2025 was a significant loss. Representing other games as graphs will inevitably prove more complex; the original structure of Myst facilitated a relatively natural graph creation. To apply such an approach to free-movement games, nodes must shift from static views (files) to map locations, introducing challenges in automatically identifying edges based on how game engines encode location reachability.

I am delighted to see that the Myst Graph has generated interest, particularly from Cyan, the company behind Myst, in an enthusiastic discussion on Hacker News, and from Andrew Plotkin, whose mystextract was inspired by this project. In August 2025, I presented the Myst Graph at Mysterium 2025 (recording). Initially, I envisioned a more niche audience for this project. I hope to see further creative works built upon this foundation.

I would be glad to hear from you, to collaborate, to ask about the implementation or to share a graph you made: myst@glthr.com. I am excited to see how the community will shape and evolve this project: feel free to fork the repository and contribute your ideas.

Series and Companion Articles

The Myst Graph (glthr.com/page/myst-graph)

#	Article	Address
1	The Myst Graph, 1: The 1993 Game as a Graph	glthr.com/myst-graph-1
2	The Myst Graph, 2: Unreachable Views and Disabled Links	glthr.com/myst-graph-2
3	**The Myst Graph, 3: Creating the Graph Using DeMystify** (this article)	glthr.com/myst-graph-3

The 5 Myst Studies on glthr.com (glthr.com/myst-graphs)

Game	Graph Study & Browser	Findings
Myst (1993)	The Myst Graph (3) · Browser	The Library Ceiling Shortcut, Behind the Ages, Trapped in Myst
Riven (1997)	The Riven Graph (2) · Browser	The Riven Diffs, The Contradictions of the Fifth Age, Catherine Is Still in Her Cell, The Sixth Knock, What Changed Between Riven's CDs and DVD
Myst III: Exile (2001)	The Exile Graph (2) · Browser	Linking Without a Book, The Valve That Refuses to Turn
Myst IV: Revelation (2004)	The Revelation Graph (2) · Browser	Visions the Amulet Never Showed
Myst V: End of Ages (2005)	The End of Ages Graph (2) · Browser	Commands Beyond the Slate

How to cite this article

> Guillaume Lethuillier, "The Myst Graph, 3: Creating the Graph Using DeMystify", glthr.com, May 9, 2025.
> https://glthr.com/myst-graph-3

For BibTeX:

```
@misc{lethuillier2025myst,
  author = {Lethuillier, Guillaume},
  title  = {The Myst Graph, 3: Creating the Graph Using DeMystify},
  year   = {2025},
  month  = may,
  url    = {https://glthr.com/myst-graph-3},
  note   = {glthr.com}
}
```

Your questions and leads are welcome: myst@glthr.com

This document was typeset with LaTeX from the article published at https://glthr.com/myst-graph-3, which remains the reference version (as updated on November 20, 2026).