Mathcraft: Topos theory in videogames like minecraft

Luca Leon Happel

from Mönchengladbach

Acknowledgements

Contents

1 Introduction **1**

2 Rendering a Mathcraft Sheaf **4**

3 Conclusion **6**

A Appendix **7**

References **7**

1 Introduction

As established in [1], information that depends on location can be modelled using *sheaves*. We wish to expand on this idea in the context of the popular video game *Minecraft*. Its world is a subset of $\mathbb{R}^3$, inhabited by two kinds of objects:

(1) *blocks*, which are unit cubes placed on the lattice $\mathbb{Z}^3 \subset \mathbb{R}^3$, and

(2) *entities*, such as dropped items, players and enemies. Several entities may occupy the same location at the same time.

To render a (more or less) infinite world in finite time, Minecraft approximates it by finitely many pieces, a technique known as *chunking*. A *chunk* is a bounded region of $\mathbb{R}^3$, together with the information about which blocks and which entities are placed inside of it. In [1] we saw that chunking can in fact be modelled using (pre)sheaves. To this end, fix a width w, a depth d and a height h, all positive integers, and equip $\mathbb{R}^3$ with the topology τ generated by the closed boxes

$$C_{i,j,k} = [i, i+w] \times [j, j+d] \times [k, k+h], \qquad (i,j,k) \in w\mathbb{Z} \times d\mathbb{Z} \times h\mathbb{Z}.$$

These boxes tile the space $\mathbb{R}^3$, as seen in Fig. 1. Since the boxes are closed, two neighbouring chunks share a common face, so τ contains not only (unions of) chunks but also their faces, edges and vertices as open sets.

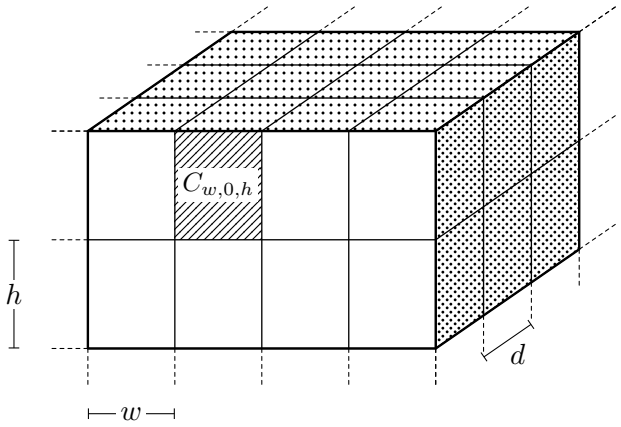


Figure 1: The boxes $C_{i,j,k}$ tile the space $\mathbb{R}^3$. Only finitely many chunks are drawn. The dashed lines indicate that the tiling continues in every direction.

A sheaf over chunks is then given by a contravariant functor $P\colon \tau^{\mathrm{op}} \to \mathbf{Set}$, which assigns to every chunk the set

$$P(C_{i,j,k}) = \mathrm{Blocks}_{w,d,h} \times \mathrm{Entities}_{w,d,h}$$

of its possible contents, and which satisfies the *sheaf conditions*: for every open set U and every open cover $U = \bigcup_{\iota \in I} U_\iota$,

(i) *locality*: if $s, t \in P(U)$ satisfy $s|_{U_\iota} = t|_{U_\iota}$ for all $\iota \in I$, then $s = t$, and

(ii) *gluing*: if $s_\iota \in P(U_\iota)$ is a family with $s_\iota|_{U_\iota \cap U_\kappa} = s_\kappa|_{U_\iota \cap U_\kappa}$ for all $\iota, \kappa \in I$, then there is some $s \in P(U)$ with $s|_{U_\iota} = s_\iota$ for all $\iota \in I$.

These conditions are important for the boundaries between chunks. Two neighbouring chunks share a common face, and the sheaf conditions ensure that the contents of two chunks which agree on this face fit together into exactly one content of their union. Here we may assume

$$\text{Blocks}_{w,d,h} = \text{Hom}_{\mathbf{Set}}(\mathbb{N}_{<w} \times \mathbb{N}_{<d} \times \mathbb{N}_{<h},\ \text{BlockID}),$$

where BlockID $\cong \mathbb{N}$ stores which block is placed at a given position (say 0 is an empty or air block, 1 is a dirt block, 2 is a water block, and so on), and

$$\text{Entities}_{w,d,h} = \text{Hom}_{\mathbf{Set}}([0,w) \times [0,d) \times [0,h),\ \text{List}(\text{EntityID})),$$

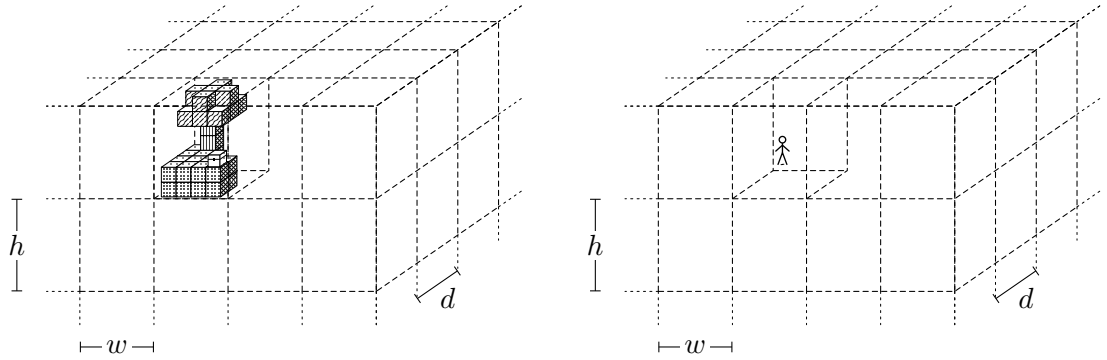
where $\text{List}(A)$ is the inductive set with $\text{List}(A) \cong A \times \text{List}(A) + 1$, and EntityID $\cong \mathbb{N}$ stores which entity is placed at a given point of a chunk. Since the value at a point is a list, multiple entities may share a position.

We may also obtain P as the product $P = E \times B$ of an *entity sheaf* E and a *block sheaf* B, with $E(C_{i,j,k}) = \text{Entities}_{w,d,h}$ and $B(C_{i,j,k}) = \text{Blocks}_{w,d,h}$. This works because a functor into a product of categories is the same as a pair of functors,

$$[\tau^{\text{op}}, \mathbf{Set} \times \mathbf{Set}] \cong [\tau^{\text{op}}, \mathbf{Set}] \times [\tau^{\text{op}}, \mathbf{Set}],$$

Is this really true for sheaves?

and P is the image of the pair (B, E) under this isomorphism, composed with the product functor $\times\colon \mathbf{Set} \times \mathbf{Set} \to \mathbf{Set}$.



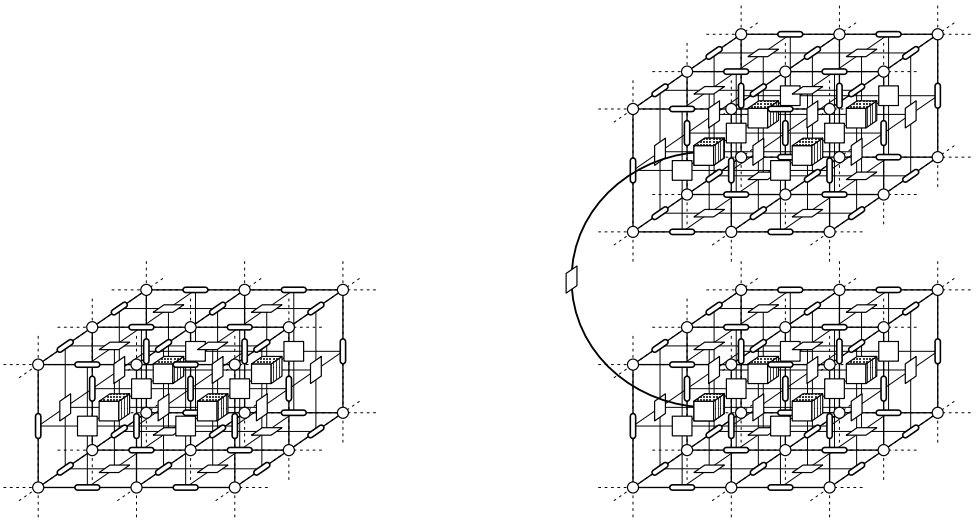
(a) The block sheaf B: blocks placed in the chunk $C_{w,0,h}$.

(b) The entity sheaf E: a player standing in the chunk $C_{w,0,h}$.

Figure 2: The two components of $P = E \times B$ over the same chunk $C_{w,0,h}$. Blocks (left) sit on the lattice $\mathbb{Z}^3$, entities (right) at arbitrary points.

We call a sheaf $P\colon \tau^{\text{op}} \to \mathbf{Set}$ with $P(C_{i,j,k}) = \text{Blocks}_{w,d,h} \times \text{Entities}_{w,d,h}$ a *Euclidean Mathcraft sheaf*. We use the word *Euclidean* because instead of τ we could also consider graphs

γ and sheaves over them with values in $\text{Blocks}_{w,d,h} \times \text{Entities}_{w,d,h}$ for appropriate values of w, d and h (for example, a node that corresponds to a face between two chunks might have $d = 1$). The topology τ corresponds to the lattice X that tiles $\mathbb{R}^3$, as shown in Fig. 3a, but we may also add a "wormhole", which is then given by an additional face shared by two chunks. This yields the graph Y, as shown in Fig. 3b.



(a) The graph X.

(b) The graph Y, with a wormhole face.

Figure 3: Graphs underlying Mathcraft sheaves, drawn for $2 \times 2 \times 1$ chunks. Each node stands for a cell of τ: vertices are drawn as circles, edges as bars, faces as squares and chunks as cubes. Lines join every cell to the cells on its boundary.

In fact, in Minecraft this is exactly what is used for portals, such as Nether portals or End portals. This becomes apparent when considering a block sheaf over Y, as shown in Fig. 4.

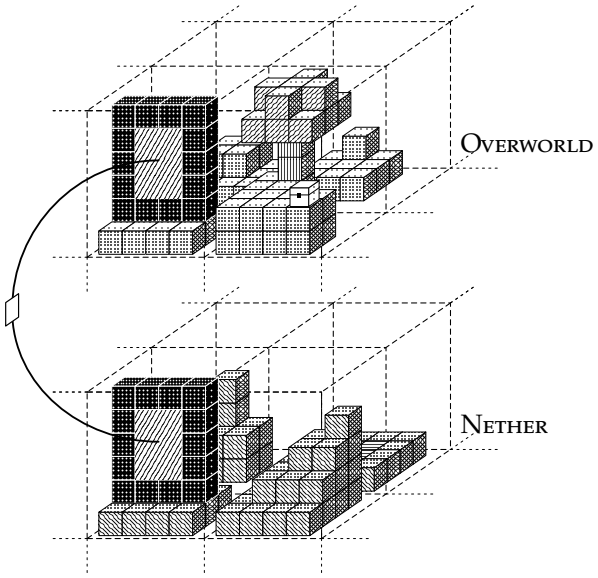


Figure 4: A block sheaf over Y. The chunk at the front left of the Overworld and the chunk at the front left of the Nether each contain a Nether portal, and the wormhole face of Y glues the two portals together.

2 Rendering a Mathcraft Sheaf

As we have established in Section 1, Minecraft worlds with their blocks and entities can be represented as sheaves W. The most natural next question is: how do we *render* such a sheaf?

Rendering here means that we want to draw an image onto a (computer) screen that displays the information contained in W. During gameplay, a user is typically assigned a dedicated entity called the *player*, and the user has a *rendering distance*. The player is an entity e, which naturally encodes its position $e_p \in \mathbb{R}^3$. By its EntityID, we wish to associate with it a velocity $e_v = \partial e_p/\partial t$, an acceleration $e_a = \partial^2 e_p/\partial t^2$ and a *heading direction* $e_d \in S^2$, which encodes where the entity looks.

The rendering distance in classical Minecraft is given by some natural number $r \in \mathbb{N}$. Given the chunk C_i the player is in, it considers the open set

$$U = \bigcup_{\mathrm{dist}(i,j)\leq r} C_j,$$

where dist: $\gamma \times \gamma \to \mathbb{N}$ counts how far apart two nodes of the graph are, that is, the length of a shortest path between them, disregarding any nodes that do not correspond to chunks, such as those for faces. Thus two chunks are one step apart exactly when they share a common face. For the lattice X from Fig. 3a, dist is then just the taxicab metric on the chunk indices $(i/w, j/d, k/h) \in \mathbb{Z}^3$.

We may however choose any open set, or subgraph, U which contains the player for rendering, and we will call it the *rendering set*.

To model rendering, we will use the topos of trees $\mathcal{S} \coloneqq [\omega^{\mathrm{op}}, \mathbf{Set}]$, which we have intro-

duced and studied in [2]. In particular, this topos allows us to model ray tracing naturally and in a safe manner.

Possibly also add the topos for synthetic differential geometry here?

The general idea is that we construct a term ρ of the coinductive stream type

$$\mathtt{Str}(U) \cong U \times \blacktriangleright \mathtt{Str}(U)$$

in $\mathcal{S}$, which starts at the player e in $W(U)$ by setting

$$\rho|_0 = [\rho_0] \coloneqq [e_p],$$

and which follows the geodesic spanned by e_v until it collides with a block or an entity. We use here that each chunk $C_j \subseteq U$ is trivially diffeomorphic to $[0,w] \times [0,d] \times [0,h]$ via a chart $\varphi_j \colon C_j \to [0,w] \times [0,d] \times [0,h]$ (for the lattice X, simply the translation that moves the corner of C_j to the origin), and that for each $v \in U$ we can find an appropriate subset of $\mathbb{R}^3$ which it lies in, such that addition, differentiation and a canonical dot product are well defined. Write $v_0 \coloneqq e_v$ for the initial direction of the ray, and let

$$\lambda_{n+1} \coloneqq \inf\{\lambda > 0 \mid \rho_n + \lambda v_n \text{ lies in a block or an entity}\} \in \mathbb{R}_{\geq 0} \cup \{\infty\}$$

be the distance (in units of v_n) to the next collision, so that $\rho_{n+1} = \rho_n + \lambda_{n+1} v_n$ is the point of collision. There, the ray is reflected and continues in the direction

$$v_{n+1} \coloneqq v_n - 2\langle v_n, \nu_{n+1}\rangle \nu_{n+1},$$

where ν_{n+1} is the unit normal of the surface that was hit, which follows from the law of reflection with respect to the tangent plane $t = \nu_{n+1}^{\perp}$.

For blocks, colliding is easy, as the tangent plane t is clear. Every face of a block is axis-aligned, so $\nu_{n+1} \in \{\pm e_1, \pm e_2, \pm e_3\}$, and the reflection simply flips the sign of one coordinate of v_n. Thus, when colliding with a block,

$$\rho|_{n+1} = [\rho_0, \dots, \rho_n, \rho_n + \lambda_{n+1} v_n], \qquad v_{n+1} = v_n - 2\langle v_n, \nu_{n+1}\rangle \nu_{n+1}.$$

For entities, we would indeed need to know some additional information, such as the shape of the entity. For simplicity, we will just assume that every entity f is a sphere of radius s around its position f_p. The unit normal at the point of collision is then $\nu_{n+1} = (\rho_{n+1} - f_p)/s$, which gives, when colliding with an entity f,

$$\rho|_{n+1} = [\rho_0, \dots, \rho_n, \rho_n + \lambda_{n+1} v_n], \qquad v_{n+1} = v_n - \frac{2}{s^2}\langle v_n, \rho_{n+1} - f_p\rangle(\rho_{n+1} - f_p).$$

Finally, assume that the ray starting at ρ_n in direction v_n never collides with a block or an entity, or that it reaches the boundary ∂U of the rendering set before doing so. In both cases we set $\lambda_{n+1} = \infty$, and the ray ends there, so we let the stream stay constant from then on,

$$\rho|_{n+1} = [\rho_0, \dots, \rho_n, \rho_n], \qquad v_{n+1} = v_n.$$

We may therefore generalize this procedure of ray tracing to a family of global elements in the topos of trees. Given a starting position $x \in U$ and a heading vector $v \in T_x U$, this yields

$$\begin{aligned} &\mathsf{ray}_{x,v} \colon 1 \to \mathtt{Str}(U), \\ &\mathsf{ray}_{x,v} \coloneqq [\rho_0, \rho_1, \dots], \end{aligned}$$

where ρ is constructed as above, starting with $\rho_0 = x$ and $v_0 = v$.

We can now consider the oriented plane of width plane_w and height plane_h, spanned by orthogonal unit vectors $u, v \in T_{e_p} U$, given by

$$P \coloneqq \left\{ e_p + \exp(au + bv) \in U \;\middle|\; a \in \left[-\frac{\mathsf{plane}_w}{2}, \frac{\mathsf{plane}_w}{2} \right],\ b \in \left[-\frac{\mathsf{plane}_h}{2}, \frac{\mathsf{plane}_h}{2} \right] \right\},$$

where $\exp \colon (T_{e_p} U, +) \to (\mathbb{R}^3, +)$ is the exponential map of Lie groups [3], using the identification of U with subsets of $\mathbb{R}^3$ from above.

3 Conclusion

A Appendix

References

[1] Luca Leon Happel. *Sheaves in Minecraft*. Oct. 29, 2021. URL: https://happel.ai/posts/sheaves_in_minecraft/ (visited on 11/18/2026).

[2] Luca Leon Happel. "The Topos of Trees". Bachelor's thesis. Heinrich Heine University Düsseldorf, Institute of Computer Science, Dec. 17, 2026. URL: https://github.com/Quoteme/bachelor2.

[3] Luca Leon Happel. *The Exponential Map of Lie Groups*. Feb. 27, 2026. URL: https://happel.ai/posts/lie-groups-and-algebras/ (visited on 11/18/2026).