

Tool Brick: A procedural pipeline brick tool

Project Breakdown

By James Marland, 24/11/2025

Introduction

Firstly, if you haven't seen it, check out my final piece on my page here:

<https://www.artstation.com/artwork/a0xO2z>

This project was created for my university advanced specialism module, where, over 8 weeks, I would build a pipeline tool that showcased my skills as a Technical Artist (TA).

In my search for inspiration, I thought about LEGO and the amazing games I played as a child, and I was reminded of the LEGO group's recent collab with UEFN, allowing a whole new generation to interact with the iconic bricks. Looking closer, I was partially disappointed. UEFN showed incredible potential but limited development to only UEFN, Users had access to only ~50 preset parts in the brick-by-brick mode, as well as a clean, stylised look that fit Fortnite.



Available parts in Brick-by-brick mode (Epic Games and LEGO Group, 2025).

While very creative for its use, it highlighted a gap: an Unreal brick-building tool focused on realism to create iconic brickfilms just didn't exist.

The LEGO Movie was the first time we saw LEGO really embrace realism in its animation, with scratches, fingerprints, and slightly misplaced pieces that help simulate the feel of a brickfilm virtually. In particular, I was naturally drawn to Emmet's apartment, shown near the beginning of the film; its stunning lighting and small scale make it look like it's pulled from real life. This became a key reference.



Emmet's Apartment (The Lego Movie, 2014).

Animal Logic, the animation team behind The LEGO Movie, famously talked about how they used the publicly available LEGO Digital Designer (LDD), now reborn as Bricklink, to build their sets (Fxguide, 2015).

Being a TA, I had an idea to optimise this workflow and integrate the modelling software directly into the engine to reduce the need for artists to export and import and set up parts by hand, and so was born Tool Brick, a pipeline tool that aims to let artists build environments in-engine out of LEGO for a faster production experience.

Purpose

Toolbrick's development was deeply aligned with my personal goals to show my skills as a TA, and I set myself some personal goals to show competency in several areas:

- Unreal Integration - Using Unreal's Editor Scripting Library for out-of-gameplay tools would help demonstrate my ability to create artist-friendly tools that feel integrated with the engine.
- Houdini - I wanted to solve a unique problem that demonstrated my understanding of the software and to handle a large dataset using Task Operator (TOP) nodes.
- Scripting - I also wanted to use Python to develop scripts for large-scale dataset processing.
- Portfolio Piece - Create a visually appealing portfolio environment using my tool to showcase it.

Overall, the core purpose of Tool Brick was to create a tool that really fused artistic and technical skills. By undertaking this project, I would create a tool that has not been made this way before, engineering a solution and breaking the problem down into manageable

steps. The tool had to streamline an artist's process for creating a LEGO environment and, by making it publicly accessible, could serve a real-world use case rather than just a university project.

Development

Making LEGO parts

Development began by getting a Minimum Viable Product (MVP) of my tool. I followed a simple methodology: “Make it work, then make it good.” I would make a scrappy but working version of each step before going back to connect and polish them into a final product. Let's get into what I did.

Now, as is probably clear, I did not have every single LEGO part pre-made on hand. This is where LDRAW came in, the backbone of the project.

LDRAW is an open-standard, community-driven digital recreation of almost all LEGO parts, founded by the late James Jessiman, that has existed for 30+ years and has enabled this project to access the parts data needed, ensuring accuracy, compatibility, and interoperability with other software versions of parts.

Welcome to the LDraw.org library. Here you will find the Parts Tracker, parts updates, documentation for the LDraw file format and library, and the Official Model Repository

LDraw homepage (LDraw.org, 2002).

First steps revolved around downloading this data set and extracting the metadata needed for Tool Brick, such as the Name, Part ID, Category, Keywords and most importantly the models; however, LDRAW uses its own data format to do this using raw transforms and a reference to a primitive library to reduce the filesize and standardise parts and so it does now need to be converted into a traditional model format.

File Header:

```
0 Brick 2 x 4
0 Name: 3001.dat
0 Author: James Jessiman
0 !LDraw_ORG Part UPDATE 2004-03
0 !LICENSE Licensed under CC BY 4.0 : see CReadme.txt

0 BFC CERTIFY CCW

0 !HISTORY 2002-05-07 {unknown} BFC Certification
0 !HISTORY 2002-06-11 [PTadmin] Official Update 2002-03
0 !HISTORY 2004-02-08 [Steffen] used s\3001s01.dat
0 !HISTORY 2004-09-15 [PTadmin] Official Update 2004-03
0 !HISTORY 2007-05-07 [PTadmin] Header formatted for Contributor Agreement
0 !HISTORY 2008-07-01 [PTadmin] Official Update 2008-01

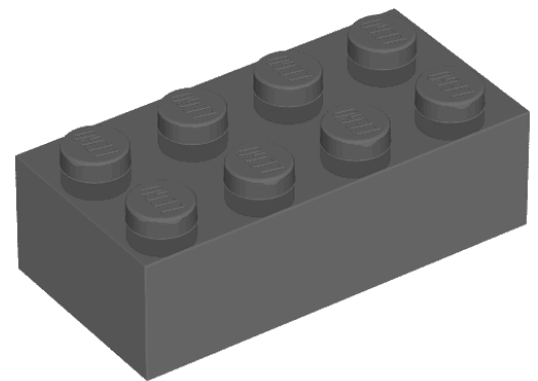
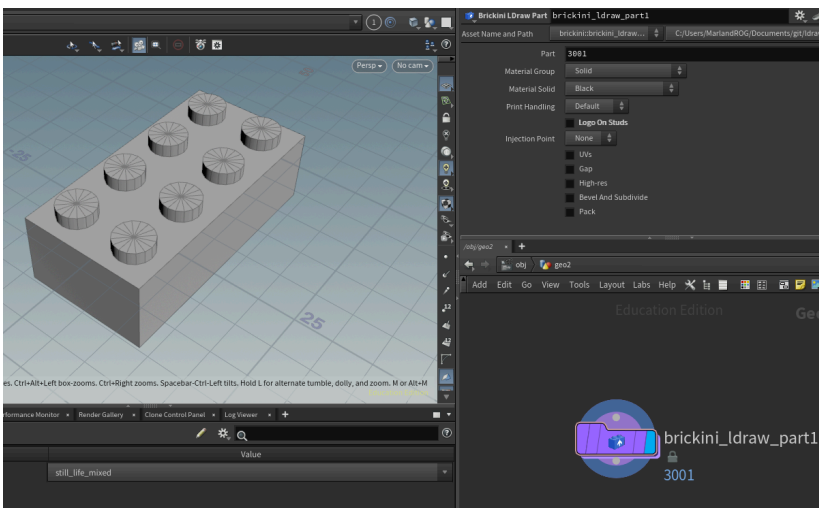
▼ Show contents
1 16 0 0 0 1 0 0 0 1 0 0 0 1 s\3001s01.dat
4 16 -40 0 -20 -40 24 -20 40 24 -20 40 0 -20
4 16 40 0 20 40 24 20 -40 24 20 -40 0 20
0
```

```
{
  "PartID": "29120",
  "Name": "Slope Brick Curved 2 x 1 with Cutout Left",
  "Keywords": "slope;brick;curved;2;x;1;with;cutout;left",
  "Studs": []
},
{
  "PartID": "2921",
  "Name": "Brick 1 x 1 with Handle",
  "Keywords": "brick;1;x;1;with;handle",
  "Studs": []
},
{
  "PartID": "3001",
  "Name": "Brick 2 x 4",
  "Keywords": "brick;2;x;4",
  "Studs": []
},
{
  "PartID": "3001b",
  "Name": "Brick 2 x 4 with Hole in Top",
  "Keywords": "brick;2;x;4;with;hole;in;top;bricklink 3001oldb;set 513-1",
  "Studs": []
},
{
  "PartID": "3002",
  "Name": "Brick 2 x 3",
  "Keywords": "brick;2;x;3",
  "Studs": []
}
```

[Left] LDraw Part Brick 2 x 4 - 3001.dat (Jessiman, 2004) [Right] Generated part list from .dats.

Generating meshes

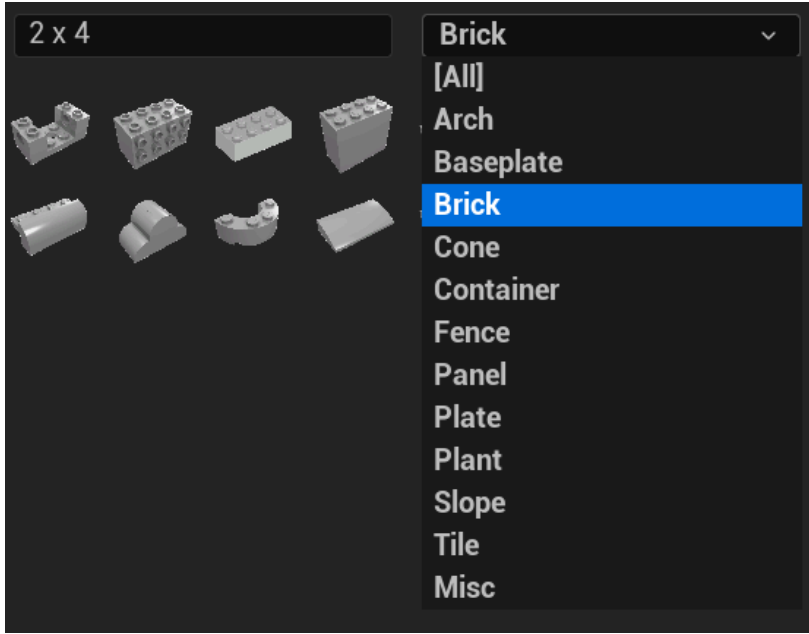
This is where the second backbone of this project comes in. I chose to adopt LDraw2Houdini (LD2H), made by Stefan Müller. As a Houdini tool, a software I planned to use anyway, it enabled fast, in-depth control of model generation using a Houdini Digital Asset (HDA). This HDA saved me the time of parsing LDraw .dat files and generating the parts myself (Müller, 2023), allowing me to focus on higher-level development for my tool. Going inside and learning how LD2H worked helped me understand how to adapt it to my tool. I also gathered images to serve as thumbnails for each part in the UI via Rebrickable's API, which let me search for a Part ID and download an image of it. Using the Python Image Library (PIL), I removed the white background and grayscaled each part to ensure uniform colour.



[Left] LDraw2Houdini generating Brick 3001 [Right] Processed Thumbnail using API (Rebrickable, 2011).

Part selection UI

With the parts ready, I imported them into Unreal as .fbx files and created a Editor Utility Widget (EUW) to let users build with them easily. I created a Data Table that could import the generated part's JSON data and could internally search the content browser for each mesh and thumbnail. The EUW could then create an Unreal Motion Graphic (UMG) element for each, and I created a search and category selection system for this. To futureproof this system, I cache the part objects when first created and only create the UMG slates when they meet the current search criteria. This provides a responsive UX even with 1000s of part tiles in the EUW and allows users to get parts faster than navigating the content browser.

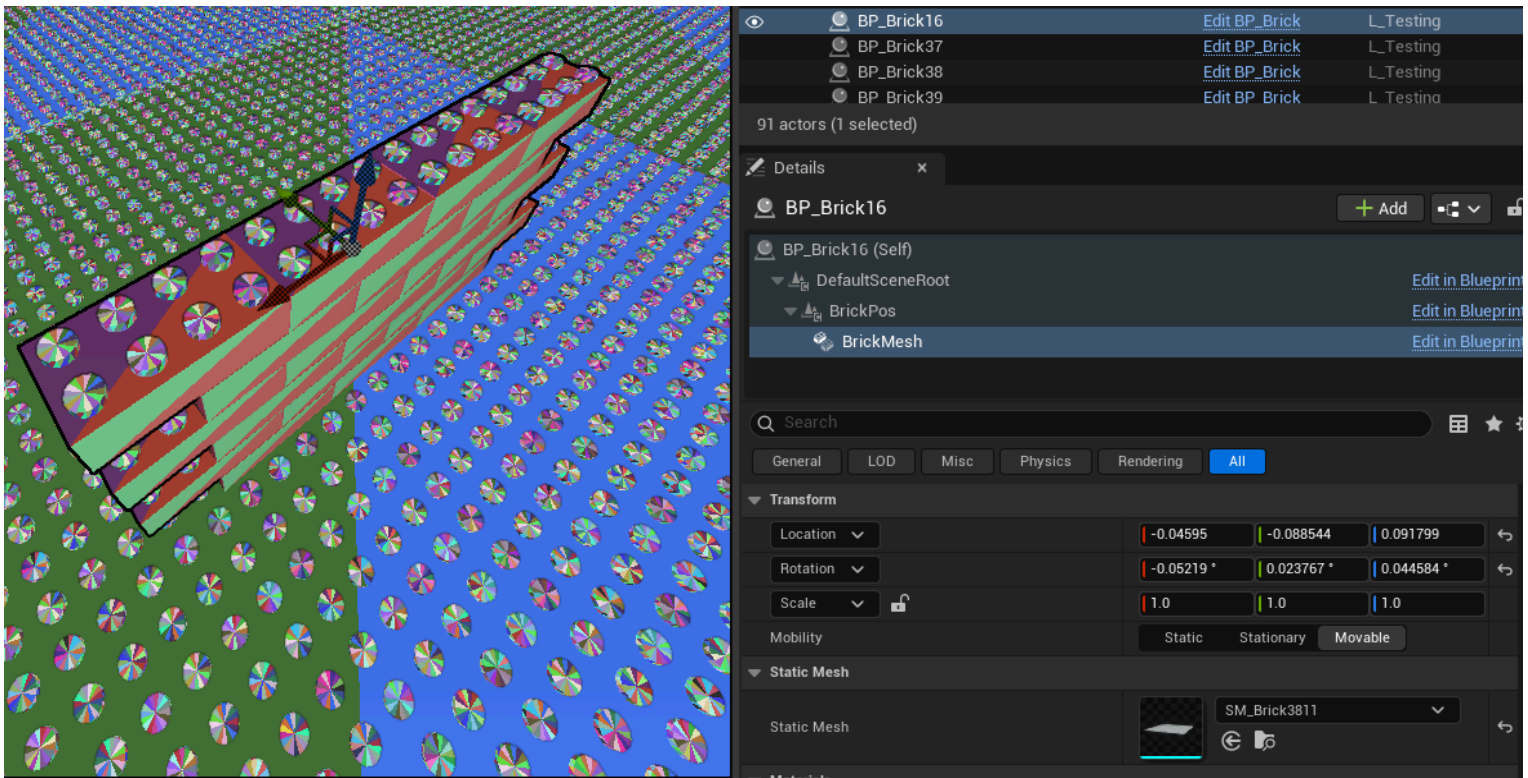


Tool Brick EUW's UI search and category selection in use.

Unfortunately, the placement isn't as UX-friendly as the content browser, since EUW don't have access to the mouse position in the viewport like you do in gameplay. As I can't create a drag-and-drop system like the one in the content browser. I compromised by using the editor camera position to raycast into the scene, placing it directly where the viewport is facing. When it hits a surface, the brick is placed on it.

Optimising bricks

I also optimised the part meshes as planned. All meshes are stored inside a Blueprint (BP) as an Instance Static Mesh (ISM) so that Unreal can efficiently handle multiple of the same part across the scene. I also used Nanite on the meshes to keep their visual fidelity while remaining very performant. Running a performance test showed that Unreal could run this setup very efficiently, reducing the rendering cost footprint of otherwise high-quality meshes. Perfect for a production environment that requires real-time modification.

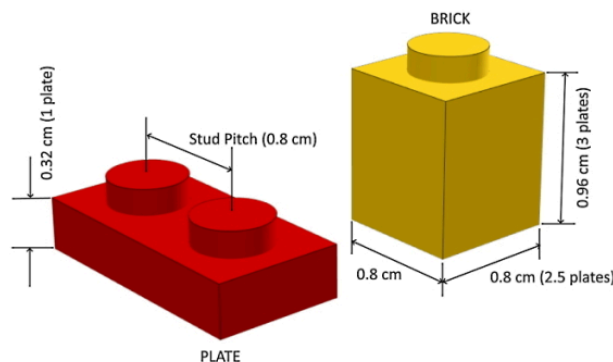


Nanite Visualiser clusters on and a ISM Component on a brick blueprint.

While it was not necessary to I made sure that as much logic was handled by the EUW and not the bricks themselves, for easier modification of the tool into other setup systems where bricks might have behaviour added.

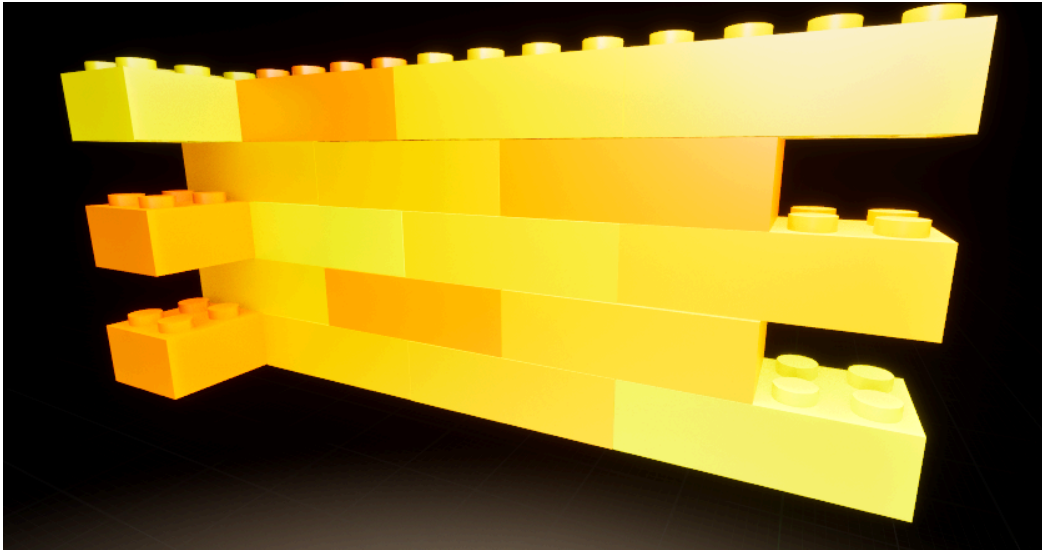
Controlling Bricks

Next, let's talk about how I handle the bricks in the scene. Firstly, LEGO bricks have a non-uniform grid system. At first, this sounds weird. Surely LEGO lines up a grid? Yes, but with a caveat: a LEGO 1 x 1 Brick's real-world dimensions are 8mm x 8mm x 9.6mm, which is not a uniform square. The units have also changed as I've scaled up the grid by 10 to Unreal units (mm to cm), then by 1.25 to fit nicely into Unreal's native grids. Resulting in our original LEGO 1 x 1 Brick being 10cm x 10cm x 12cm (a scale factor of 12.5). I also have to allow for a subgrid in height, as LEGO has plates. 3 plates being the height of 1 brick. To allow for the plates in Unreal, I actually snap our grid to 10cm x 10cm x 4cm.



LEGO size dimensions for a 1 x 1 Brick and a 1 x 2 Plate. (Shenoy, 2022)

Before moving on, I also slightly scale down the bricks by 0.99 and randomly offset each brick to simulate the small gaps between bricks, as shown below. This helps greatly with simulating real life, as bricks are never perfectly placed, and real LEGO bricks are slightly smaller to allow tolerance between each brick. The Bricknerd's Osker Cederwell has a fantastic breakdown of the weirdness you get from these size decisions (Schefcik, 2022).

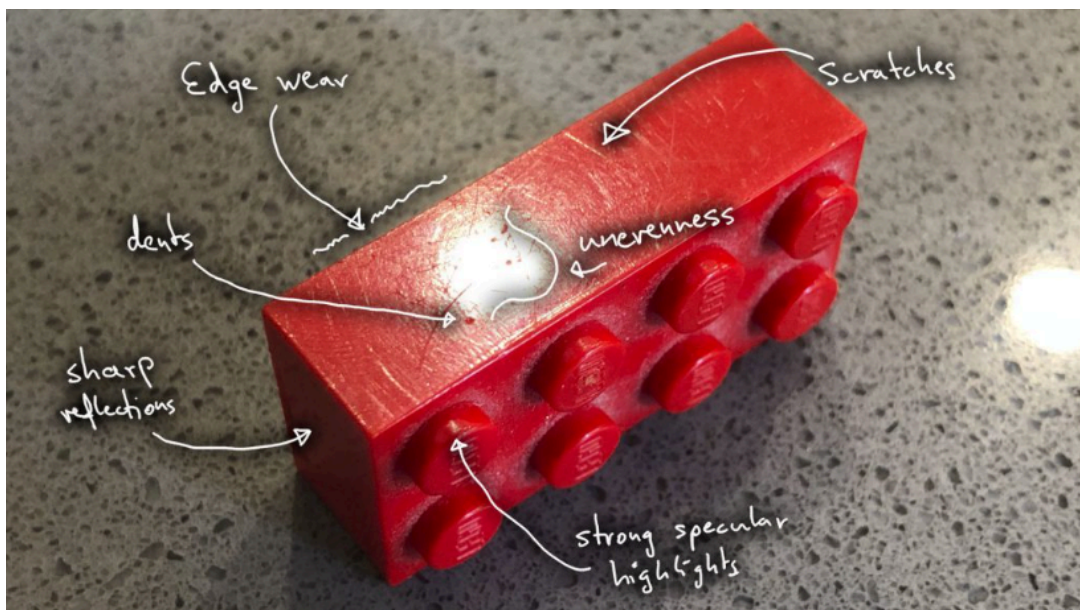


Bricks are slightly scaled down and randomly offset in location and rotations.

Colouring the bricks

Next, I worked on the colours, like I said before, I wanted realism. Luckily, I had the answers in front of me. Stephan Müller has been studying LEGO brick materials since 2016 and has a backlog of blogs breaking down the steps required, though not every aspect was available in real time without generating baked textures for each part, I did include the most important:

Surface roughness, scratches, noise, dust, fingerprints and a soft, bumpy surface are key to keeping the bricks grounded in realism.



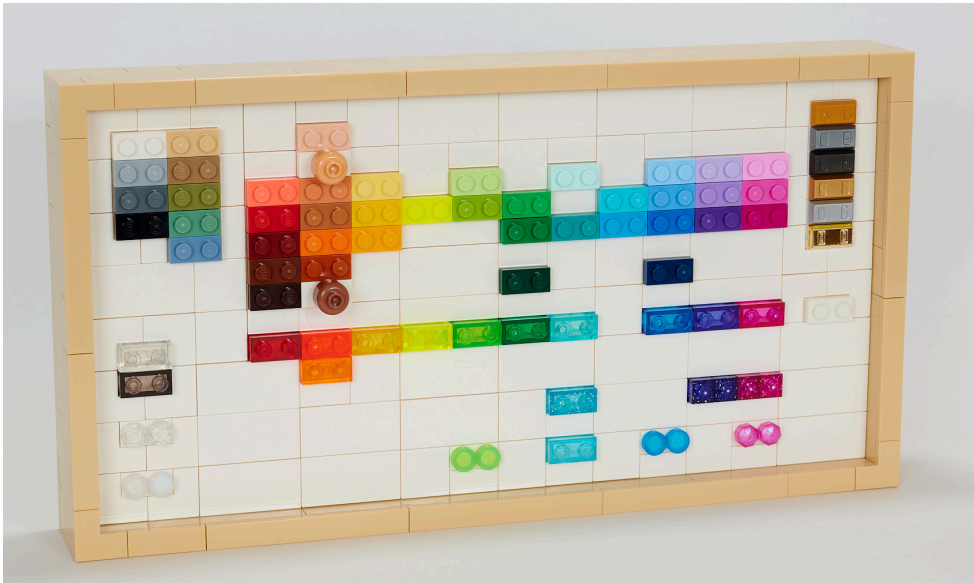
Breakdown of what makes the brick look real (Müller, 2019).

Then came what colours to support. LEGO has released 100s of colours, and it would be ridiculous to include every one off, so I created a requirement to be included using LEGO Bricklink's colour guide: The part has to be from the main LEGO series, actively printed and have more than 100 parts using it, or if it was active and a special colour, I would include it. I also decided to use Bricklink's naming standard for each part, as there's a surprising amount of naming conflict across platforms.

Solid Colors							^
	Color Name ↕	Parts ↕	In Sets ↕	Wanted ↕	For Sale ↕	Timeline	ID
	White LEGO Color: White - 1	16,873	12,391	23,999	17,195	1949–2025	1
	Light Bluish Gray LEGO Color: Medium Stone Grey - 194	5,343	8,575	9,532	6,023	1993–2025	86
	Dark Bluish Gray LEGO Color: Dark Stone Grey - 199	4,541	7,870	8,574	5,062	1991–2025	85
	Black LEGO Color: Black - 26	14,197	14,200	19,440	14,424	1957–2025	11
	Dark Red LEGO Color: New Dark Red - 154	2,071	2,338	4,345	2,171	1957–2025	59
	Red LEGO Color: Bright Red - 21	8,891	11,941	14,003	9,432	1949–2025	5
	Reddish Orange LEGO Color: Reddish Orange - 402	192	191	944	194	2024–2025	167
	Coral LEGO Color: Vibrant Coral - 353	296	541	1,022	304	2019–2025	220
	Dark Brown LEGO Color: Dark Brown - 308	921	2,218	3,063	1,002	2007–2025	120

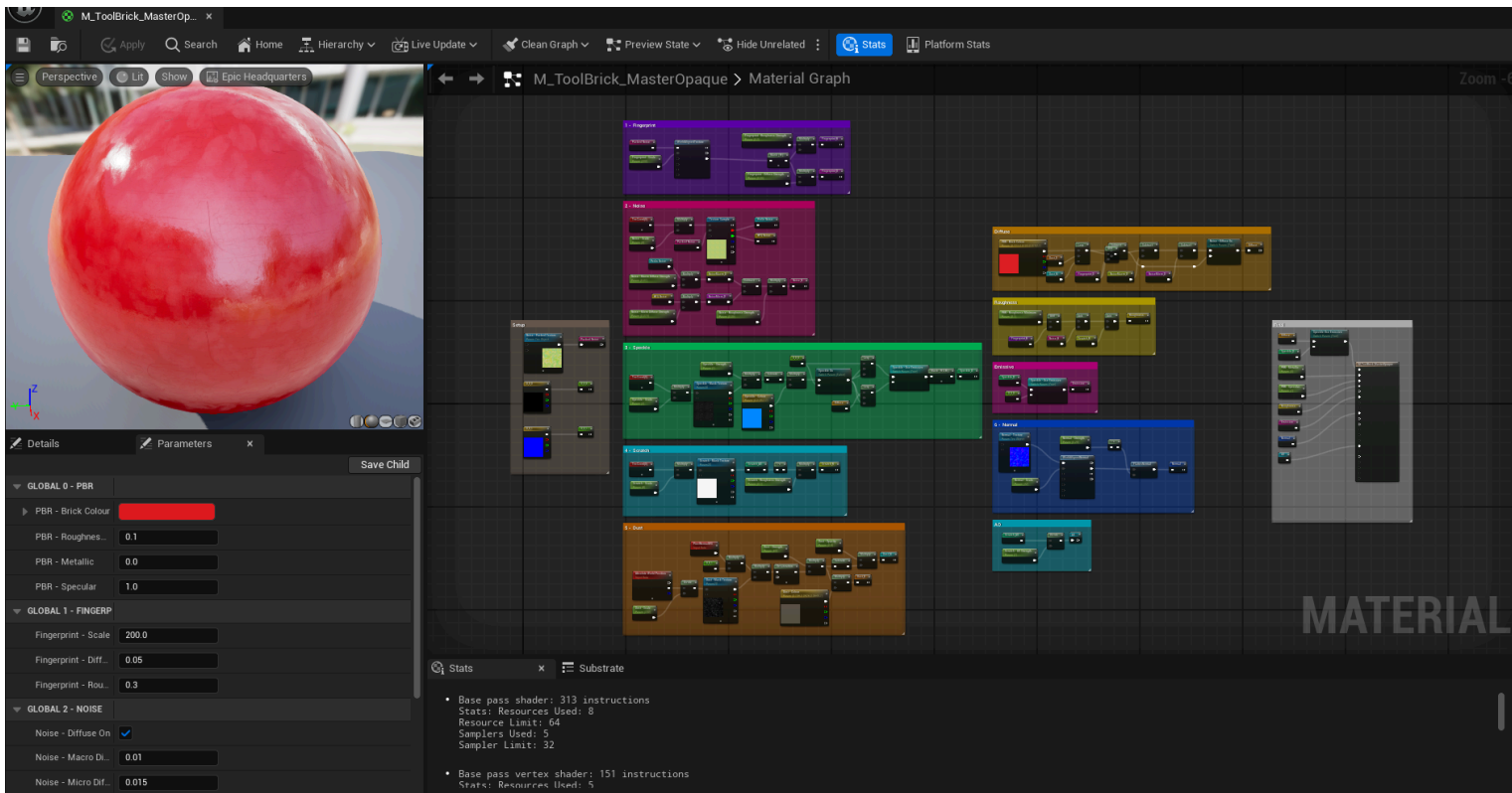
Colour guide (LEGO Group, 2024)

Inspired by Bricknerd's Chris Clarke, I laid out my Colour swatches as top-facing bricks in a similar grid style. Chris's research into the hex code for each colour was also incredibly useful to nail each solid colour's exact look.



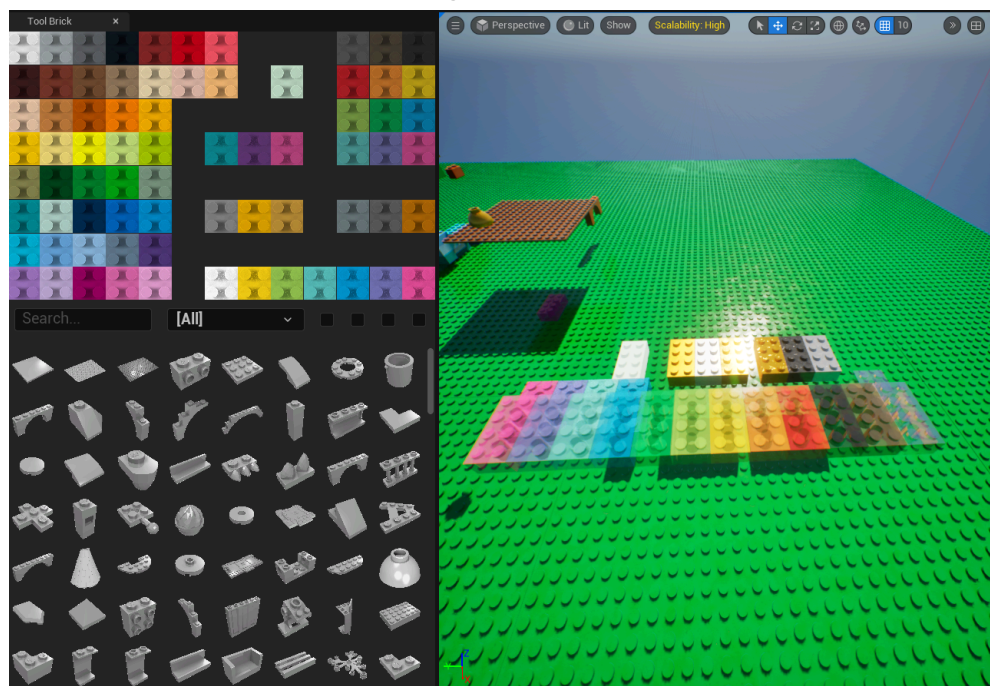
2023 Colour Palette recreated in real life (Clarke, 2023).

And so, with the colours decided, I made this as a master material in Unreal, first for solids, then later for transparents, and allowed complete control over each aspect of the shader. I also used material instances for different colours, which allows Unreal to batch the materials for performance.



M_ToolBrick_MasterOpaque, containing several sections to create a realistic LEGO shader.

The UI with colour swatches, a searchable part list, and the in-scene bricks with their realistic shaders is shown below at this stage in development.



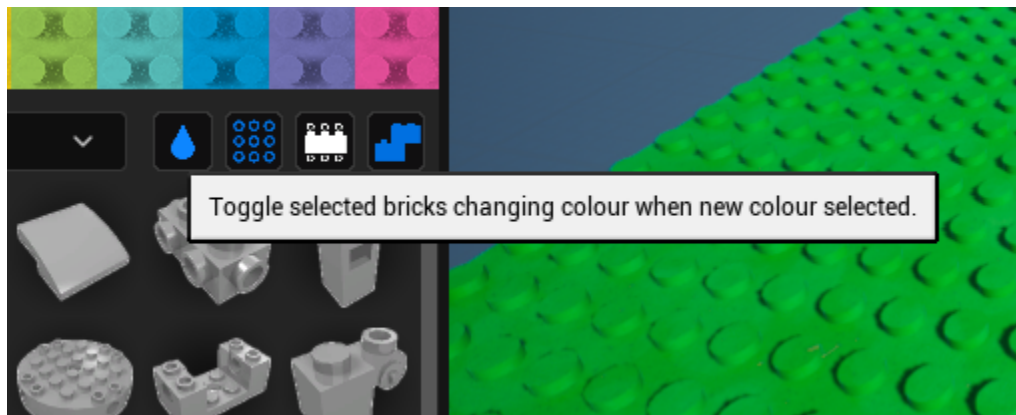
Tool Brick's UI and scene.

Feedback

MVP done! Now that I had the base of my tool down, I conducted playtesting to see how testers interacted with Tool Brick, and I gained invaluable insights:

- User controls - Testers sometimes wanted to place bricks off the grid and control them by their stud points, snapping them together.
- Better UX - Testers didn't know which colour they had selected or how to move the brick properly.
- Saved configuration - Testers wanted the EUW to save its state so that when reopened, it wouldn't be reset.
- More Categories - Testers wanted more categories so they could get more specific brick types quicker.

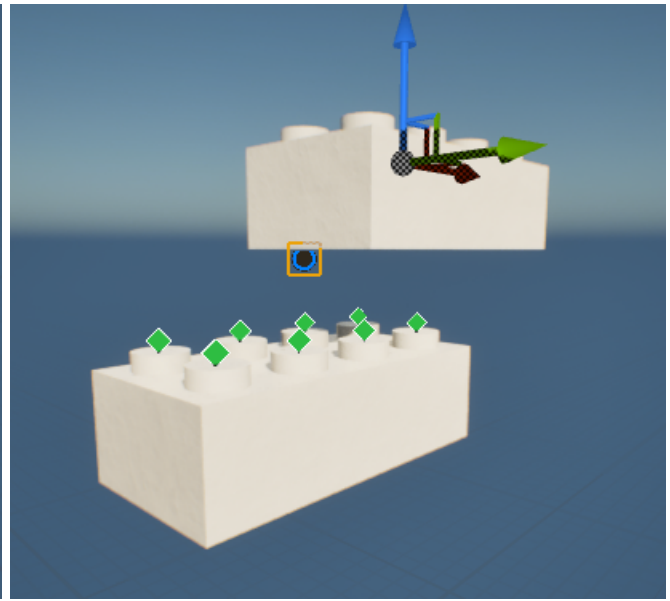
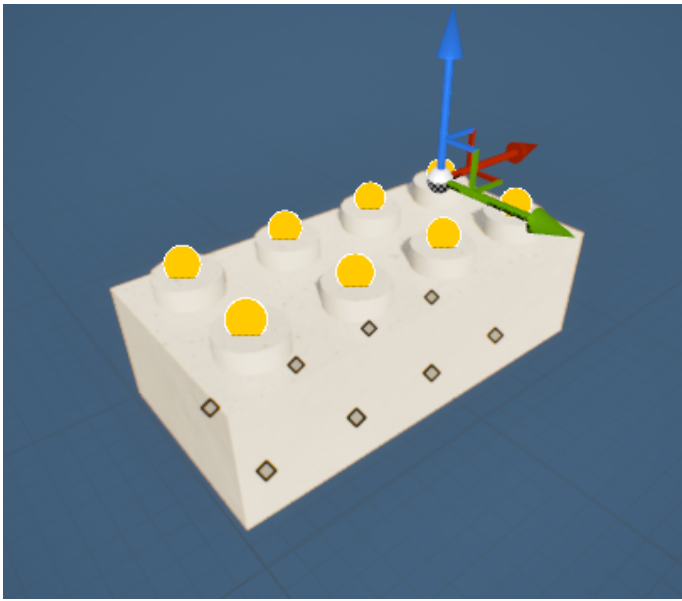
Even with these things to work on, the tools' use was positive, and testers showed a real interest in Tool Brick, so I felt confident that these changes would be beneficial. Adding more categories was simple enough due to the framework I had made. Next, I added icons to a selected brick to indicate its state and tinted the part tiles the selected colour. Then I worked on a save state for the EUW by saving the control variables as JSON and loading them when the tool opens or closes. Finally, I added controls to turn on and off for the following features: Colouring selected bricks, Snapping to the grid, Selecting studs and Snapping studs. These massively helped the user control their experience.



EUW's UI with toggle buttons for each option.

Stud Selection

Now I mentioned stud selection and snapping, but I haven't talked about it yet. Up to this point, it was only on one brick for testing and development. When a brick is selected, the tool creates snap handles at every stud. The user can select these and move the part relative to the stud. It also removes any points that can't connect to the selected stud and, if those are clicked, snaps the bricks together relative to the studs. However, this required setting the studs' positions and types manually; ideally, I could do this procedurally for each part. This is where Houdini becomes useful again.

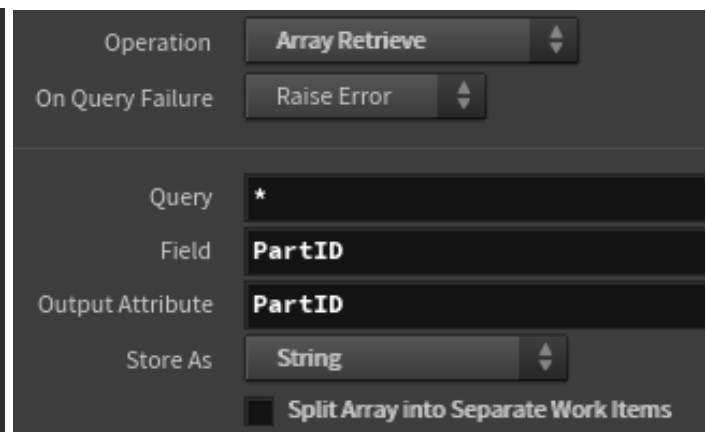
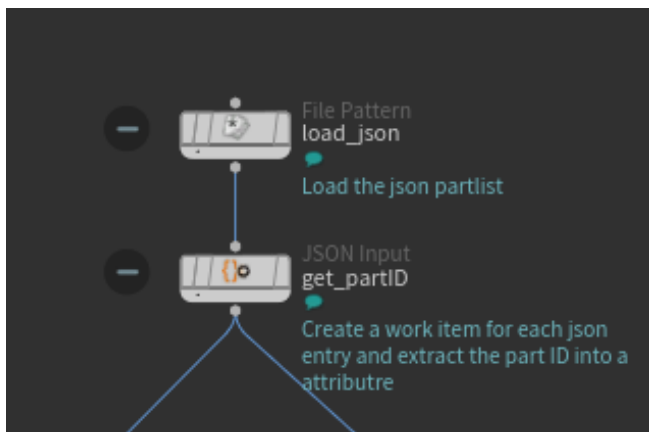


Stud snap handles that, when selected, show the points to which they can be attached.

Stud Position Generation

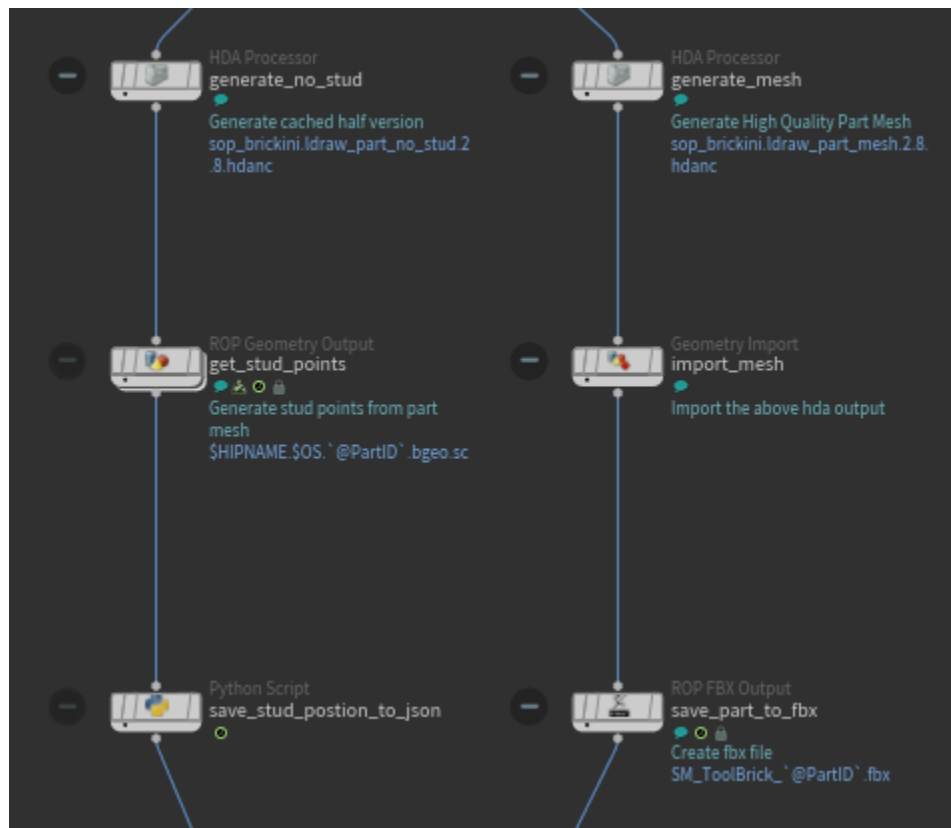
This step was left to last for good reason. It wasn't essential for the tool to work, but it would be a nice addition to get finer control of the bricks. Ultimately, I did have the time to include it, which I was very glad about, as it let me show off my Houdini and problem-solving skills. Here's how it works:

Firstly, I used a Procedural Dependency Graph (PDG) with Task Operators (TOP) network nodes, as it is perfect for handling and creating a large set of parts. I started by upgrading the original mesh generation to use PDG. I create a work item for each part in our previously generated initial part list and extract the part number for later use.



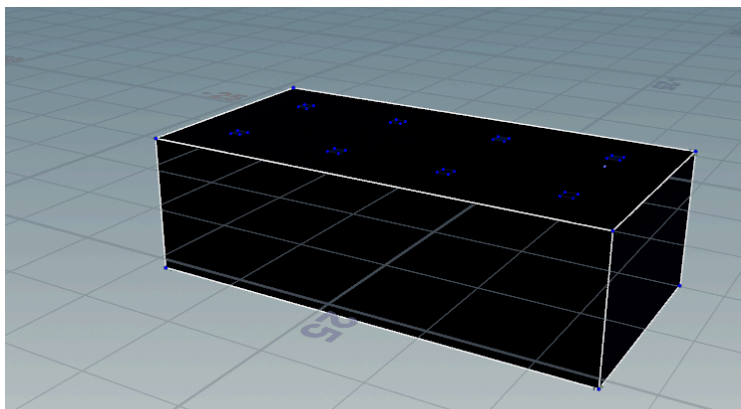
PDG that loads json file and extracts each part as a work item.

This is then split into two branches. The right path is what I had before, converted to PDG. I get the part number and generate a high-quality mesh used in-engine using the LDraw2Houdini HDA. It is primarily the same as the original, with some slight tweaks: scaling it up, setting the origin to the LEGO grid, and UVing. Then it's saved as an .fbx for import into the engine.



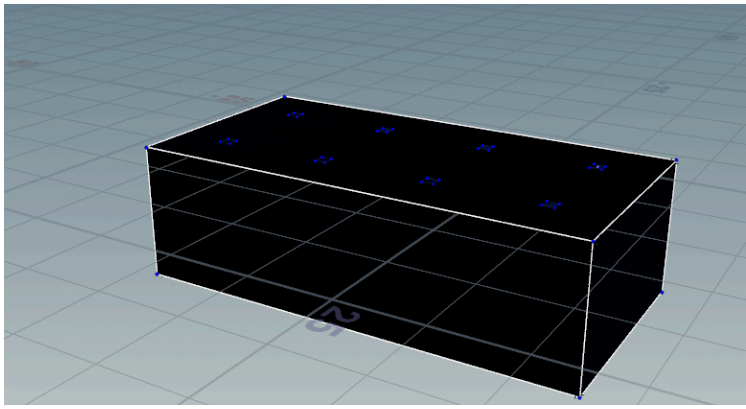
PDG's TOP nodes laid out.

The left is where I generate the stud positions. I start by hijacking the LD2H caching system to generate a halfway version of the part, containing only its base mesh and attributes that provide useful information.



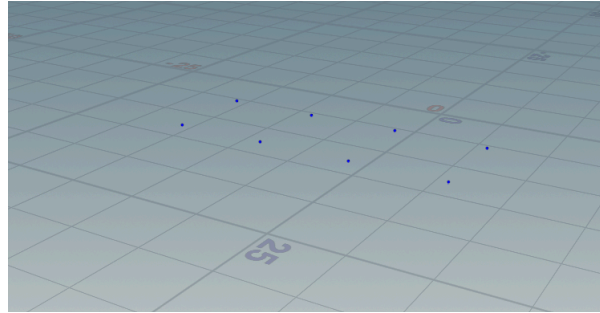
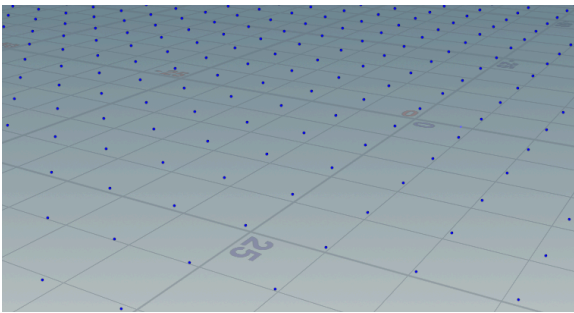
Cached part 3001 geometry.

Working out where a top stud is actually quite easy. Since LDraw references the same primitive for a top stud each time, LDraw2Houdini saves these positions in the cached version and adds the top-stud geometry in the final version. I save these points as our top stud positions, get the first one, and set the mesh origin to it. This means that in-engine, the brick will be aligned to the grid.



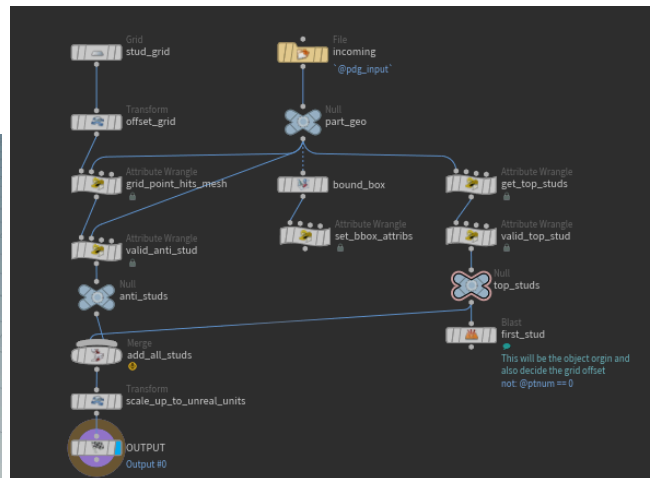
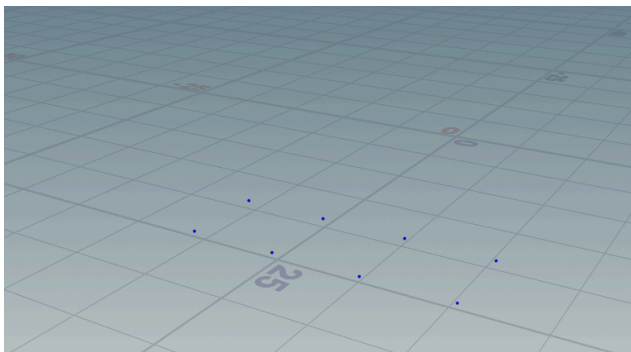
Origin point set to the first stud, and geometry moved so the origin is at 0,0.

Now that the mesh is aligned, I create a grid of evenly spaced 1-brick points below the mesh (using the mesh's bounds and the lowest point) and perform a raycast up into the mesh, deleting any that don't hit it.



[Left] Evenly spaced by 1 brick grid, point 1 is at 0,0 to line up. [Right] The points that hit the mesh.

Then I iterate up from the lowest point to the highest point, where we hit the mesh. Each iteration, it moves up by one plate height and raycasts in 8 directions. If 6 out of the 8 rays hit the mesh, this is a place where the stud could sit, and so it is a valid anti-stud position.



[Left] Final anti-stud positions after iteration. [Right] ROP Output Workflow.

If it completes all iterations without getting a valid point, I delete it. I bring all these points together, then, using a Python script, iterate over each point, store its offset from the origin and type, and write it to a JSON file for each part.

```

1 import json
2 import os
3 import hou
4 import pdg
5
6 # Load Geo
7 geo = hou.Geometry()
8 geo.loadFromFile(str(work_item.inputFiles[0]))
9 valid = True
10 partID = str(work_item.stringAttribValue("PartID", 0))
11
12 if not partID:
13     valid = False
14
15 # Collect each stud
16 studs = []
17 for idx, pt in enumerate(geo.points()):
18     pos = pt.position()
19     stud_type = pt.attribValue("type") or "Unknown"
20
21     # Format stud
22     studs.append({
23         "ID": idx,
24         "Type": stud_type,
25         "PosOffset": { # Convert position Houdini Y-up > Unreal Z-up
26             "X": round(pos[0]),
27             "Y": round(pos[2]),
28             "Z": round(pos[1])
29         },

```

```

30     "RotOffset": { # Expand in future to support side studs
31         "Pitch": 0,
32         "Yaw": 0,
33         "Roll": 0
34     }
35 })
36
37 # Format Output
38 output_data = {
39     "PartID": partID,
40     "Studs": studs}
41
42 # Save JSON
43 hip_dir = os.path.dirname(hou.hipFile.path())
44 output_folder = os.path.join(hip_dir, "StudPos")
45 os.makedirs(output_folder, exist_ok=True)
46 output_path = os.path.join(output_folder, f"StudPos_{partID}.json")
47
48 if valid:
49     with open(output_path, "w") as f:
50         json.dump(output_data, f, indent=4)
51     print(f"SUCCESS: {partID} had {len(studs)} studs")
52 else:
53     print("ERROR: Work item did not have a Part ID!")
54

```

Code for saving JSON files for each part's stud positions.

I'm quite proud of it, and it gets stud positions for ~70% of the parts. Now this isn't perfect. If there aren't any top studs, the grid isn't aligned, and the rays fail to find any valid anti-stud positions. The mesh's origin is also dependent on the top studs, so some parts are misaligned in the current release.

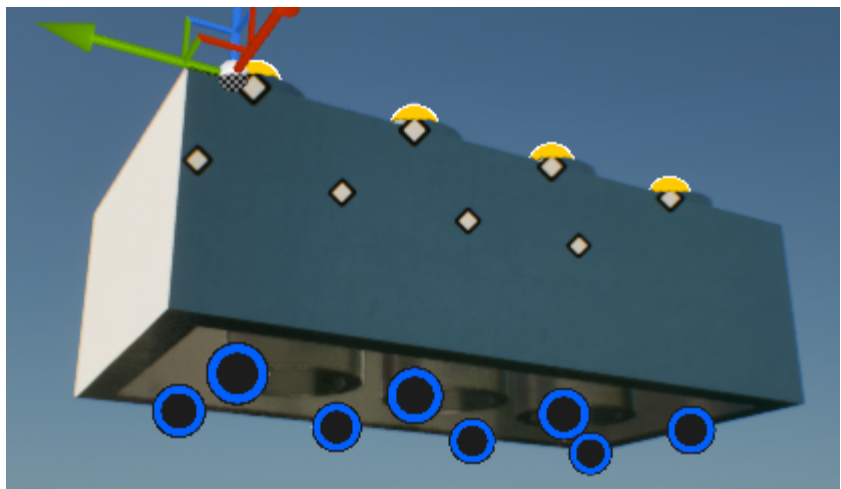
The current time frame meant that working on this in more detail was out of scope, but if I worked on it further, I could check whether the studs' geometry fits closely to the walls and reduce false positives. I'd love to expand what it's capable of, including side studs, non-stud parts, and weird-format stud parts.

Anyway, this is enough for now. With the part stud positions saved, they are merged back into the main part list. This is used by the tool to spawn snap-point handles.

```

{
  "PartID": "3001",
  "Name": "Brick 2 x 4",
  "Keywords": "brick;2;x;4",
  "Studs": [
    {
      "ID": 1,
      "Type": "SolidTopStud",
      "PosOffset": {
        "X": -10,
        "Y": 0,
        "Z": 0
      },
      "RotOffset": {
        "Pitch": 0,
        "Yaw": 0,
        "Roll": 0
      }
    }
  ]
}

```

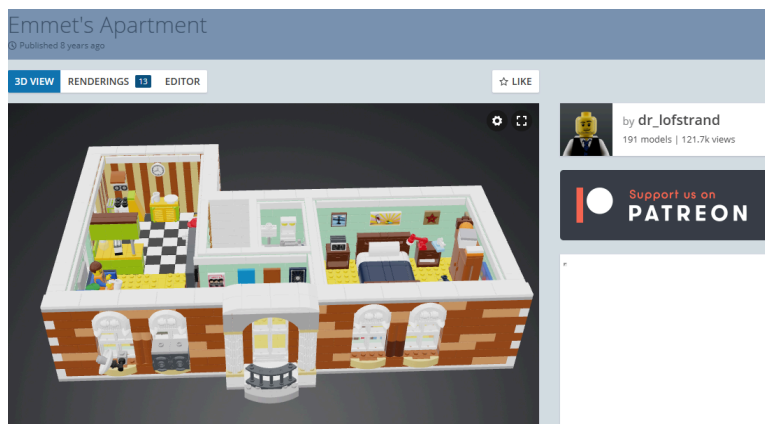


[Left] Part 3001 main list entry with stud positions saved. [Right] Snap handles on a part.

Wrapping up

Finally, I worked on cleaning up the project, organising files and code, adding QoL features, and reducing the number of steps required to set it up. This was because not only was it useful for presentation, but also because I would release it.

With my tool complete, I was able to work on my final goal, creating the environment. This was made into a very simple process, thanks to my pipeline tool, and I was able to recreate Emmet's apartment. With only a few parts and layout changes, I was able to make it quickly, thanks to dr_lofstrand's model recreation of the apartment as my reference.



Emmet's apartment from The LEGO Movie recreated (dr_lofstrand, 2017).

I had the model and movie reference on a second screen and was able to recreate each part from emmet's bedroom. This process took 5 hours, including setting up the lighting and camera, and using my tool was great as I could use it in the showcase video of a practical example. The level is also available in the release and was clearly built for one camera angle, and looking at any other angles shows nothing else, as it wasn't necessary for this piece. Here's an editor screenshot of the level, followed by the final image after retouching in Photoshop.



In-editor screenshot of ToolBrick in use and environment.



Final shot for portfolio.

Finally, I worked on cleaning and improvements to make the experience of setting it up much clearer than in the MVP, reducing steps, adding QoL features, cleaning up code, and organising the project for release. There isn't much to talk about other than going back over old systems and streamlining them. To see how each system works in detail, download and look over the entire project on GitHub here: <https://github.com/jamie-marland/ToolBrick>

Conclusion

Future Improvements

While I was happy with what I've created in the 8 weeks. I definitely still have ambitions for how it could be expanded:

- Known Issues - There are some minor issues that can be easily avoided, but I would really like to remove them:
 - Parts that aren't aligned to the grid because their top stud origin isn't set are much harder to move around nicely, since they are off the grid and require turning off grid snapping. Fixing this would be very beneficial.
 - Small bugs: Deleting a brick with snap handles triggers an unnecessary warning message, and part tiles without thumbnails can unintentionally become tinted when scrolling.
 - Nanite still doesn't support translucent materials, so they can be quite expensive. Handling this better, most likely with traditional LODs, is an aspect to consider.
- Component-based snap handles - The current setup creates a new BP for each stud handle. On parts with a large number of studs, this can become quite laggy to move due to the attached actor component. Moving this system inside the brick itself would be more performant and easier to handle.

- Upgrade LD2H - During development, LD2H upgraded to a more streamlined generation process, which is great, but it would involve reworking my tool to fit this new version or recreating it.
- Set importing - being able to read .moc set files and converting them into Unreal would be useful in cases where the user has existing sets to add, and massively increase the ability to share and collaborate with others.
- In-editor part generation - Generating all the parts beforehand is great for this setup, but if the user needed just one more part, it would require generating them all again from scratch. Making this system more modular to treat each part as a separate package, as well as using Houdini Engine to bring HDA behaviour into the editor, could mean a setup where the artist can request a part, generate new parts, and dynamically load them into the engine without ever leaving.
- More tools - An idea from the original brief I wrote that was quickly abandoned to focus on the core setup in the time frame was having smaller tools using the brick setup, placing fences, rocks, walls, greebling with a simple selection system and configurable settings, rather than hand placing every part would really help speed up the process.
- C++ and deeper engine integration - Without a doubt, one of the biggest pitfalls of this project was the lack of depth blueprint had exposed. Editor Utilities and JSON handling lack sufficient control in Unreal. I consistently have to create workaround behaviour. Writing the tool in C++, while probably being of a smaller scale, would have allowed greater control and engine integration to let the tool interact at a deeper level, such as standalone non-component ISMs, getting mouse position in the editor viewport, reading JSON output files effectively and automatic importing for Unreal assets.

Reflection

I'm incredibly proud of what I have created and the knowledge I've gained throughout this module. In retrospect, now that I have my tool, I feel genuine satisfaction because I achieved every goal I set for myself at the beginning. The opportunity to flex a variety of skills during the module that I had yet to really show in my portfolio was a big part of this tool, and showing off with a great visual piece paired perfectly with it. This module helped me grow professionally and improve my public presence, and provided a great independent experience that helped me get used to creating a tool in a more industry-oriented format. This project would not have been possible without the LDraw Community, Stephen Müller, and the Houdini and Epic Games documentation and forums for their invaluable insight and resources. I'm excited to continue and show my passion and potential for TA going forward, and if you've got this far, thank you for reading all of this. And I appreciate your interest in my work. If you want to talk more about it, I'm always available!

Disclaimers

As part of my university's AI policy, its use requires a mandatory disclaimer:

'I acknowledge my use of OpenAI's ChatGPT 5.1 <https://chatgpt.com/> that aided in researching, concepting, coding, and proofreading. While mainly serving as a consultant and a search system, some code was made with its assistance.'

If external assets were used, they have been fully credited and used in accordance with each respective license. Credit can be found on GitHub:

<https://github.com/jamie-marland/ToolBrick>

The project is released under a CC BY-NC 4.0 License:

<https://creativecommons.org/licenses/by-nc/4.0/>

LEGO® is a registered trademark of the LEGO Group, which does not sponsor, endorse, or authorise this tool. Visit the official LEGO website at <http://www.lego.com>

Bibliography

- LDraw.org (2002). LDraw.org - LDraw.org Homepage. [online] Ldraw.org. Available at: <https://www.ldraw.org/>.
- Clarke, C. (2023). The LEGO Color Palette: 2023 Edition. [online] BrickNerd - All Things LEGO and the LEGO Fan Community. Available at: <https://bricknerd.com/home/the-lego-color-palette-2023-edition-1-24-23>.
- dr_lofstrand (2017). Mecabricks.com | Emmet's Apartment. [online] Mecabricks.com. Available at: <https://www.mecabricks.com/en/models/3Lkj91ZqjAp>.
- Epic Games and LEGO Group (2025). The LEGO® System in Play Expands with Brick-by-Brick Building and More Customisation Options Coming to Unreal Editor for Fortnite. [online] LEGO.com. Available at: <https://www.lego.com/en-us/aboutus/news/2025/june/updates-unreal-editor-for-fortnite> [Accessed 23 Nov. 2025].
- Fxguide (2015). Brick-by-brick: How Animal Logic Crafted the LEGO Movie - Fxguide. [online] Fxguide. Available at: <https://www.fxguide.com/featured/brick-by-brick-how-animal-logic-crafted-the-lego-movie>.
- Jessiman, J. (2004). File Detail parts/3001.dat. [online] Ldraw.org. Available at: <https://library.ldraw.org/parts/3436> [Accessed 23 Nov. 2025].
- LEGO Group (2024). Color Guide. [online] Bricklink.com. Available at: <https://v2.bricklink.com/en-us/catalog/color-guide> [Accessed 23 Nov. 2025].
- Müller, S. (2019). Exploring LEGO Material Part 3 - Stefan's CG Blog. [online] Stefanmuller.com. Available at: <https://stefanmuller.com/exploring-lego-material-part-3/>.

- Müller, S. (2023). GitHub - stefanmuller/ldraw2houdini: Import LDraw Files Directly into Houdini. [online] GitHub. Available at: <https://github.com/stefanmuller/ldraw2houdini>.
- Rebrickable (2011). Rebrickable | Rebrickable - Build with LEGO. [online] rebrickable.com. Available at: <https://rebrickable.com/home/>.
- Schefcik, D. (2022). LEGO SNOT Basics: Geometry, Techniques and Pitfalls. [online] BrickNerd - All Things LEGO and the LEGO Fan Community. Available at: <https://bricknerd.com/home/snot-basics-geometry-techniques-and-pitfalls-3-18-2021>.
- Shenoy, D. (2022). Isn't That Stud Supposed to Be on Top? No, It's SNOT - Building Sideways Using LEGO. [online] Brick Builder's Handbook. Available at: <https://brickbuildershandbook.com/2022/01/20/> [Accessed 23 Nov. 2025].
- The Lego Movie. (2014). Warner Bros. Pictures.