

1 **HaptiEditor: Haptics Integrated Virtual Terrain Editing Tool**

2  
3 CÉLESTE MARUEJOL\*, École de technologie supérieure, Canada

4 SEAN BOCIRNEA\*, University of British Columbia, Vancouver, Canada

5 RISHAV BANERJEE\*, University of British Columbia, Okanagan, Canada

6  
7 The contemporary landscape of virtual world design is characterized by the ubiquity of diverse tools and terrain design engines,  
8 which have significantly reduced the barriers to entry in this domain. Despite this progress, the predominant input modalities of  
9 mice and keyboards fail to provide users with haptic feedback during the processes of designing, testing, and experiencing virtual  
10 environments. Addressing this limitation, we introduce HaptiEditor, a virtual terrain editing tool that integrates haptic feedback  
11 through the utilization of force-feedback capabilities offered by the Haply 2Diy device, implemented within the Unity game engine  
12 framework. Our primary objective is to enhance both the design process and the evaluative capacity of designers, as well as to enrich  
13 the immersive engagement of users or players navigating these virtual worlds.

14  
15 Additional Key Words and Phrases: Haptics, Tool Design, Unity, Haply, ForceFeedback, Haptic Texture Rendering



36 Fig. 1. The main screen of the haptic editing tool

37  
38 **1 INTRODUCTION**

39 Haptics interfaces are often used for enhancing sculpting experiences. However, no significant implementation of force-  
40 feedback interfaces in the case of terrain editing and terrain painting has been created. Many existing haptic-augmented  
41 sculpting applications might be applicable to this task, but use haptic devices with three, or even six degrees of freedom,

42  
43 \*All authors contributed equally to this project

44  
45 Authors' addresses: Céleste Maruejol, École de technologie supérieure, 1100 R. Notre Dame O, Montréal, Canada, celeste.maruejol.1@ens.etsmtl.ca; Sean  
46 Bocirnea, University of British Columbia, Vancouver, 2329 West Mall, Vancouver, Canada, seanboc@student.ubc.ca; Rishav Banerjee, University of British  
47 Columbia, Okanagan, 3333 University Way, Kelowna, Canada, rishav.banerjee@ubc.ca.

53 which are often both large and expensive. With HaptiEditor, we intend to use the target domain of terrain editing to  
 54 our advantage, developing a compelling user interface with a relatively inexpensive two degree of freedom device.  
 55

56 HaptiEditor is a Unity application which has for objective to edit maps and terrain by painting textures and objects  
 57 on different scales. By using the Haply 2DIY we hope to create an interesting approach to terrain creation and edition  
 58 through haptic feedback. The goal is to allow real time editing of a virtual space, and simultaneously feel the effect of  
 59 the changes right away.  
 60

61 The development period spaned 3 months during the Winter session of 2024, as part of the cross-institutional  
 62 CanHap501 course. CanHap501 is a program which covers multiple Canadian universities, introducing graduate  
 63 MSc students to haptic interfaces. The course teaches students how to conceptualize, prototype, develop and do user  
 64 evaluation with multimodal human-computer interfaces and haptic experiences. This project is being developed in a  
 65 team of three, with each member in different location (Montreal, Okanagan and Vancouver).  
 66

67 Since the development timeline of this project is very short, we decided to go with a rapid prototyping approach;  
 68 developing features in parallel during the course. Moreover, since we didn't have enough time to create a fully fledged  
 69 terrain editor software, we use Unity to simplify a lot of the designing and implementation to get to experiment with  
 70 the haptic side of the project quicker. This choice proved advantageous, as Unity, a game engine, already implements  
 71 some solid systems for collision, forces and texturing.  
 72

73 HaptiEditor allows exploration of terrain at different scales via a novel zooming system. Using a system of sampling  
 74 existing textures form the surface's material in order to create haptic feedback and Unity's collision system, HaptiEditor  
 75 is able to provide different haptic feedback depending on the scale of the end effector scale in the scene. This allows a  
 76 haptic continuum to enable the user to feel the terrain they are editing at any scale they would potentially need.  
 77

78 This report is a statement of the advancements and findings made during the development of HaptiEditor. We will  
 79 go over the different avenues tried in order to create HaptiEditor, what was prototyped and how it shaped the current  
 80 software. We will then examine the results gathered through semiformal user testing and the overall appreciation the  
 81 project received. The results of the user testing and our implementation will thoroughly be discussed in the Discussions  
 82 section.  
 83

## 84 2 RELATED WORK

85  
 86 *Image-based Haptic Texture Rendering.* Li et al. [3] produce a method for extracting normal forces from 2D images,  
 87 which may then be rendered by a 3D haptic interface. The paper distinguishes between normal forces, acting in the  
 88 vertical axis, and tangential forces, acting in the surface plane. In our work, we render these tangential forces and forego  
 89 normal forces due to our use of a 2 DOF haptic device. We do not introduce a method for *creating* normal maps for our  
 90 application, but instead reference the work of Li et al. [3] as an example of how one may extract normal information  
 91 from image textures. Li et al. [3] display high rates of differentiability between represented textures with their method,  
 92 though textures used in their evaluation are contrived and not based on real-world images.  
 93

94  
 95 *Haptic Perception of Material Properties and Implications for Applications.* Klatzky et al. [1] offer an overview of  
 96 state-of-the-art approaches to haptic rendering of material properties. We bring attention to the discussion on texture  
 97 representation, and note we use a normal mapping approach which allows for both direction and magnitude control  
 98 of surface normals, forming a modified single-point probe model. Klatzky et al. [1] also note roughness and texture  
 99 discrimination as a common evaluation metric for applications targeting textural rendering.  
 100

*Hand Movements: A Window into Haptic Object Recognition.* Lederman and Klatzky [2] catalogs exploratory techniques used when exploring physical objects. Our interface affords the patterns of lateral motion, static contact, pressure and contour folding, mediated through the single-point probe interface offered by our 2 DOF device. We note that these affordances were deemed by Lederman and Klatzky [2] to be sufficient (in that they allow performance better than chance) for texture and exact shape differentiation.

### 3 APPROACH

We attempt to deliver on a haptics focused terrain editing experience first and foremost. We first establish a reliable coupling via a virtual proxy to Unity's physics system, and then create a proof of concept terrain editing tool with force feedback driven by said proxy.

The three fundamental questions we had were as follows:

- (1) How should we design a generic virtual coupling between Unity's physics engine and the Haply's force feedback mechanisms?
- (2) How should we detect and render textures in real-time?
- (3) How should we design the tool itself, with the main mode of interaction being through the Haply?

#### 3.1 Designing a generic virtual coupling between the Unity physics engine and Haply device

We built off the Unity template obtained from the Haply GitLab repository as the foundation for our implementation. Upon closer examination, it became evident that the forces applied were hard-coded, prompting us to adopt a PD controller model [4] facilitated by a virtual proxy (*See 2*).

In our implementation, we utilize a Unity game object, "**End Effector Actual**" to track the ideal positional data of the Haply in the absence of obstacles in our terrain, proxied by another game object "**End Effector Representation**" which respects collisions with scene objects thanks to its built-in sphere collider, allowing it to interact with Unity's physics engine. Subsequently, we establish a PD controller relationship between these two entities. The underlying operational logic mandates the "**End Effector Representation**" to consistently attempt to minimize the euclidean distance between itself and the "**End Effector Actual**". This behavior is governed primarily by the proportional component of the PD controller, supplemented by the derivative component to offer additional smoothing (*See 2a*). In instances where the "**Representation**" detects any physical collisions, it directs the Haply to exert a force in the direction of the "**Representation**" from the "**Actual**" (*See 2b*). This happens in parallel to the distance minimization attempts of the "**Representation**". Consequently, this establishment facilitates an adaptable virtual coupling mechanism, subject to the influence of Unity's physics engine via the intermediary proxy.

Notably, while the direction vector is three-dimensional, only the X and Z components of this vector are translated to the Haply to render force feedback. The selected axes are simply an artifact of our design decision for editing on a terrain lying in the X-Z plane.

#### 3.2 Generating and rendering textures in real-time

Tangentially influenced by the research conducted by Li et al. (2010) [3], our initial approach to texture rendering was by sampling a three by three pixel window beneath the end effector representation, subsequently extracting the brightness values of each pixel. Each pixel then exerted a force on the end effector away from itself proportional to its

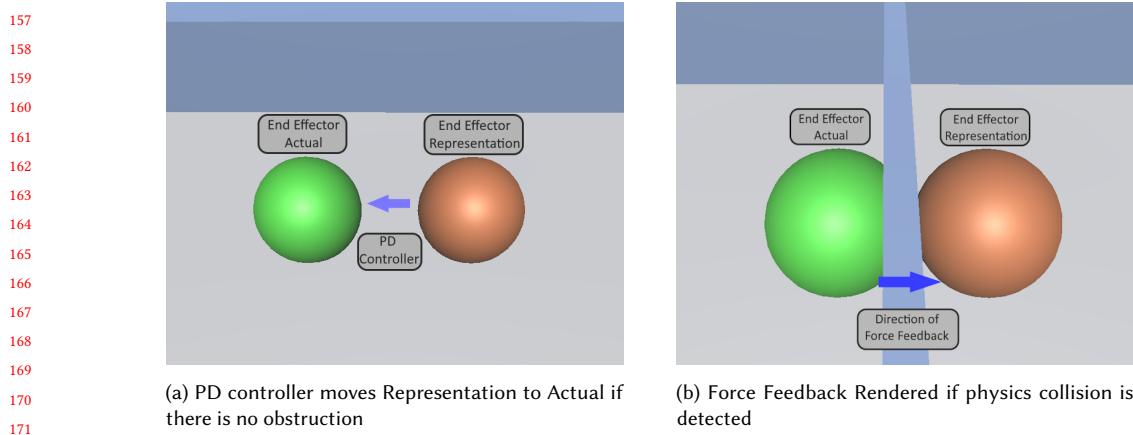


Fig. 2. Virtual Coupling between Representation and Actual End Effector. Note that the "End Effector Actual" is never visualized in the tool itself.

brightness. This mechanism imparts the perceptual impression of being coerced towards regions of lower luminosity, which can intuitively be mapped to "lower points" in the texture.

However, our development environment affords us a different option. Unity supports normal-mapping for textures, used in game development for more realistic rendering of surface reflections. Normal maps contain for each pixel a normal vector representing the direction of the surface of the material, allowing us to compute from a single pixel the direction in which a probe (our end effector) should be pushed by a surface interaction from a single pixel sampled from the normal map. We thus save both memory accesses and computation time, improving simulation speed. Normal maps for in-game materials not only commonly available, but are usually generated programmatically from a sculpted surface texture, and thus also offer improved accuracy without incurring significant overhead to potential users.

Critically, texture-driven force modulation only manifests during end effector movement, thereby avoiding any undesirable tremors during static user positioning. By recalculating this force on a per-frame basis, an appreciable frequency modulation is introduced to the end effector, directly mirroring the texture's visual attributes (See 3).

### 3.3 Tool design and Haply interaction

Game engines typically offer a range of tools for terrain editing, which are primarily oriented towards editing within the editor environment rather than functioning during runtime. Consequently, it became necessary for us to reconstruct the fundamental components of a terrain editing tool within Unity, essentially embedding them as "game mechanics". This endeavor primarily involved leveraging Unity's Terrain game object [5], which is specially built to optimally encode localized texture and static object placement data. A straightforward user interface facilitates the selection among textures, objects, and an eraser tool, further categorized into sub-menus delineating texture types (e.g., grass, sand) and object variants (e.g., trees, rocks) (refer to Figure 4). The user can then designate their preferred object or texture through mouse interaction. Upon selection, the Haply device acts akin to a virtual paintbrush, allowing object placement or texture painting at the location of the end effector. Activation of the painting action is achieved either through the stylus button integrated with Gen3 DIY devices or, alternatively, by utilizing the space-bar in cases where the stylus button is unavailable.

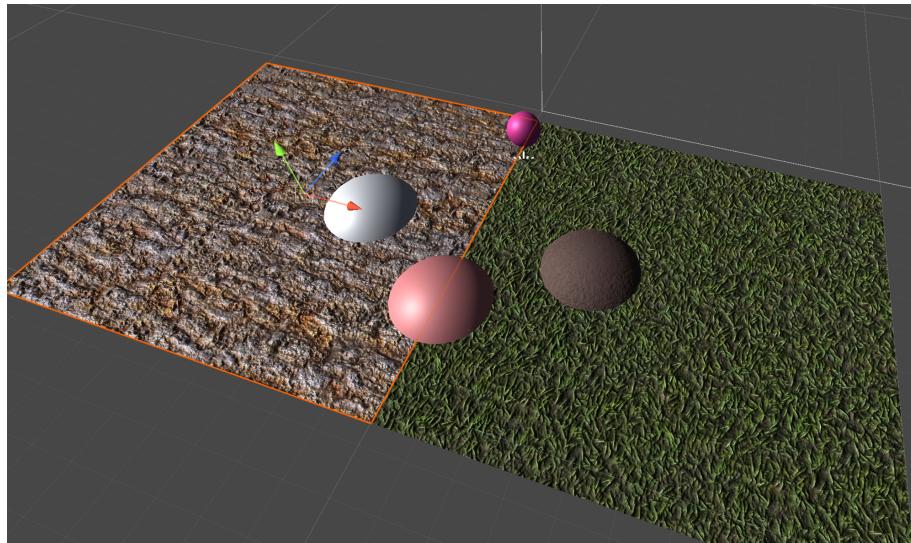


Fig. 3. Two different textures providing different jittery sensations



Fig. 4. The final terrain painting interface

250  
251  
252  
253  
254  
255 The central utility of this approach lies in its capacity to provide users with tactile feedback with their interactions  
256 with the terrain in real-time. Objects impart a rigid collider-based resistance as a consequence of the virtual coupling  
257 (see Subsection 3.1), while textures provide haptic modulation based on their visual characteristics and behavior in  
258 accordance with haptic texture rendering principles (see Subsection 3.2).  
259

260

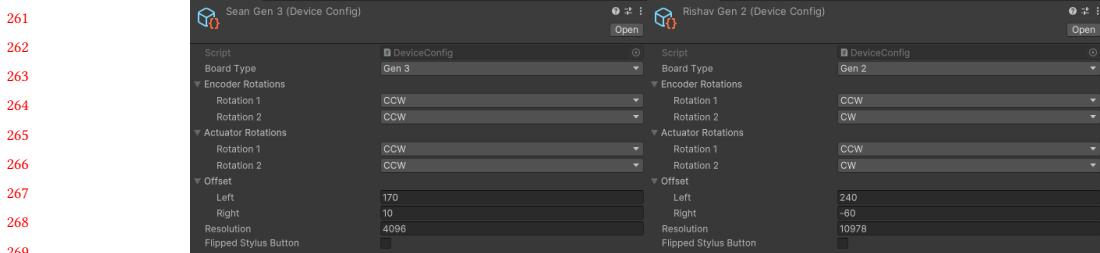


Fig. 5. Different board configurations for the Haply

## 4 PROTOTYPING

### 4.1 Developing for Unity

We re-haul two key pieces of functionality from the sample Unity Haply Gitlab repository:

*Board Configurations.* Since 2DIY Gen 3 Haply boards (and in some cases, Gen 2 boards) may be assembled in different ways, we need a way to switch easily between board configurations without spending time changing code whenever we push code to each other. Given team members on this project worked remotely and in different time zones, this was imperative to achieving a smoother development experience.

To facilitate this, we change the flow of logic for initializing the board to use configuration files, referred to as scriptable objects in Unity. By storing our different configurations in these files, we can easily swap in the appropriate configuration during development time without affecting any of the actual code.

*4.1.1 Utilizing Unity's Physics.* The main benefit to utilizing Unity is its physics engine, which we leverage for automated haptic experiences. The process of connecting the Haply to Unity physics has been described in detail in subsection 3.1. There are minor nuances in the implementation itself, specific to Unity, including making sure the physics engine is running at a higher frame-rate than normal (since physics is separated from visual output), enabling continuous collisions for a smoother experience, and interpolating all collisions. While these changes require higher compute, our finished application runs at 500+ frames per second on mid-range hardware, and these changes generally improve the experience despite the small cost to the performance of the program.

### 4.2 Zooming

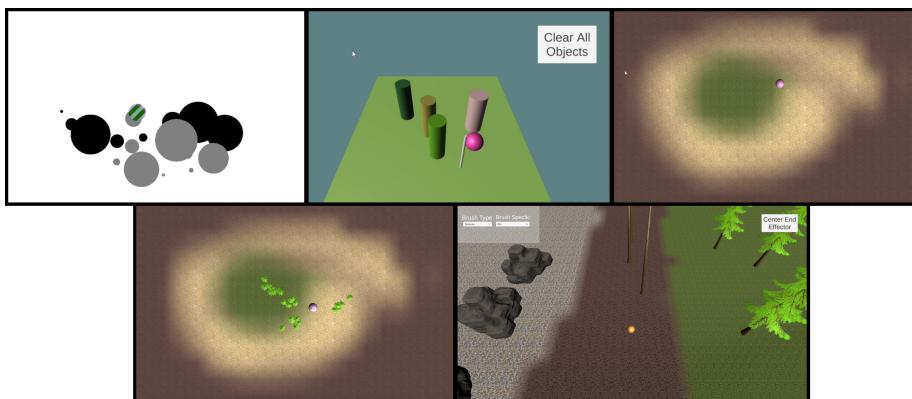
To allow the user to explore the world at different scales, we incorporate the ability to zoom in and out of the world. From a design perspective, we wished to convey the scale of interaction through haptic feedback. This was achieved by letting the user explore the ground and feel relatively strong impedance from sides of objects in the world when small, but allowing the end effector to "climb" on top of clumps of objects when large, thereby providing force feedback on the basis of the top surface geometry of the objects.

Implementing this practically was rather trivial, since we had worked extensively on abstracting the texture and object force feedback information. By changing the scale of the representation and moving the camera a proportional amount to said scale, the correct forces and texture data was automatically conveyed. Note that the ratio between the camera's movement and the scaling of the end effector is cubic. This maintains an approximately consistent size of the

313 end effector on screen, such that it gives the appearance of the world growing and shrinking around the designer or the  
 314 user.  
 315

### 316 4.3 Painting 317

318 The incorporation of painting capabilities for objects and textures within a terrain editor is imperative. Such functionality  
 319 facilitates the creation of intricate and visually captivating landscapes by allowing users to precisely apply diverse  
 320 textures and objects onto terrain surfaces, thereby enhancing realism and aesthetic appeal. More specifically, with the  
 321 ability to add their own textures and objects, this feature enables users to customize their environments with precision,  
 322 enabling the realization their artistic visions.  
 323



332 Fig. 6. Evolution of prototypes throughout the development of the project. (1) shows screenshot of the first prototype made to validate  
 333 our first iteration. (2) represents a screenshot of the first prototype after some code clean-up, transferred to 3D space.  
 334 (3) displays a screenshot of the texture painting working on Unity's terrain game object.  
 335 (4) illustrates painting objects on terrain previously  
 336 textured. (5) is a capture of the final editor, which can be seen in Figure 4

337 In line with the comprehensive nature of this project, the process of painting underwent multiple iterations of  
 338 prototypes throughout the development phase. The creation of our initial prototype, referred to as "the sprinkler,"  
 339 came towards the end of the first iteration. Its primary objective was to validate the efficacy of the PD controller  
 340 implementation and the integration of Haply's API for the third iteration of Haply's 2DIY. This prototype functioned  
 341 as a testament to the feasibility of dynamically generating objects during runtime and eliciting force-feedback from  
 342 interactions between the end-effector and these aforementioned objects. As illustrated in Figure 3.(1), the sprinkler  
 343 places gray circles as long as we press the stylus button or spacebar<sup>1</sup>. These gray circles serve as visual indicators  
 344 devoid of collision detection, providing a prelude to the subsequent placement of black circles, complete with colliders,  
 345 upon the eventual release of the stylus button or the space-bar key.

346 Just after finishing the transition from 2D to 3D, we repurposed the code of "the sprinkler" to be usable in 3D space  
 347 as a quick way to verify functionality of our scene and Haply interface. We reuse the same logic behind the placement  
 348 of object, as it had proven effective for the first prototype. In the second image of Figure 3, we see that the scene is now  
 349 in 3D, and what were previously circles are now cylinders with random colors. Force-feedback is kept the same; if we  
 350 move the End-Effector into a cylinder, we will be pushed out as if it is a wall. We concluded from this prototype that

351  
 352  
 353  
 354  
 355  
 356  
 357  
 358  
 359  
 360  
 361  
 362  
 363  
 364<sup>1</sup>The spacebar is used as backup throughout the project if the stylus button is for whatever reason unavailable

365 handling the object placement using a similar script is aligned with our goals for object placement, especially since it is  
 366 easy to use and learn how to use it, thanks to the stylus affordance.  
 367

368 The subsequent prototype aims to enhance information retention across multiple executions. Our approach pivots  
 369 towards leveraging Unity's terrain game object, which inherently retains terrain deformation, applied textures, and  
 370 placed objects. This strategic choice enables seamless integration with Unity's terrain editing system, obviating the  
 371 need for extensive bespoke implementation. This not only improves temporal efficiencies but also enhances usability,  
 372 capitalizing on user familiarity with the platform. Our focus lies on object and texture painting, and we thereby omit  
 373 tools for manipulating terrain elevation.  
 374

375 In the prototype corresponding to Figure 3.(3), we introduced the capability to paint textures via mouse input,  
 376 expediting debugging. This prototype facilitated rapid parameter experimentation, refining aspects such as brush radius,  
 377 fall-off characteristics, and curve adjustments to ensure smoother edge rendering during texture painting. From this  
 378 prototype as a base, we implemented object creation for terrain and object deletion, still utilizing the mouse as an input.  
 379 Similarly, it permitted us experiment with different parameters to find what feels best, user experience-wise. The results  
 380 are displayed in Figure 3.(4).  
 381

382 Subsequently, the amalgamation of prototypes culminated in a singular definitive outcome, depicted in Figure 3.(5).  
 383 Firstly, we transitioned the painting process to be contingent upon the positional data of the end-effector representation.  
 384 Secondly, we repurposed the co-routine mechanism utilized for object painting from the second prototype, integrating  
 385 it seamlessly with newly devised functionalities for object and texture painting, as well as object erasure. Lastly, we  
 386 devised a concise user interface, affording runtime modifications of tools and their respective painting attributes.  
 387  
 388

#### 389 4.4 Texture

390 Our initial approach to texture rendering involved firing a ray-cast to detect the material underneath the end effector.  
 391 This would then return the corresponding texture information, allowing us to sample a 3x3 window of pixels underneath  
 392 the end effector representation. Using this we could extrapolate the ideal direction the end effector should be moving.  
 393 However, using ray-casts was computationally expensive, requiring us to rethink our approach. The texture rendering  
 394 process in its current state has been described in detail in subsection 3.2 and performs better while retaining accuracy  
 395 in its representation.  
 396

## 400 5 EVALUATION AND RESULTS

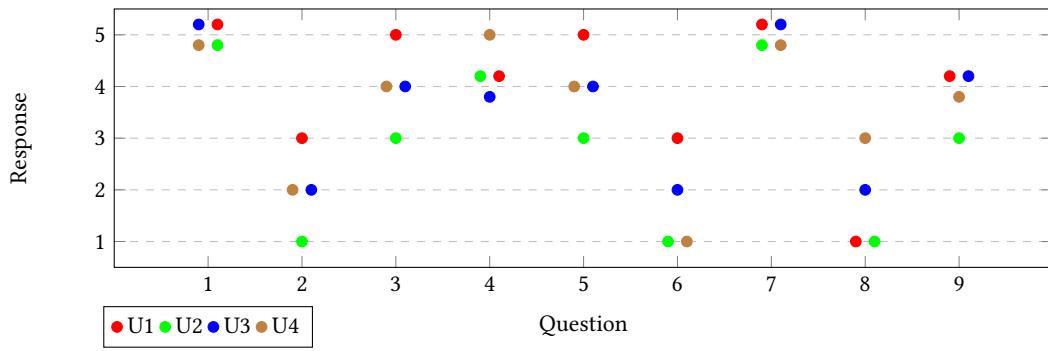
401 Evaluation took the form of a user study, in which we provided a directed walkthrough of our work to novice users,  
 402 after which we asked a nine linear scale questions about their experience. We walked users through as follows:  
 403

- 404 (1) The user was briefed on the purpose of the project, and that the end effector would be their point of interaction  
 405 with the terrain, both for painting and for feedback.
- 406 (2) The user was directed through the menu and setup screens.
- 407 (3) The user was shown the texture selection menu and told to paint different textures.
- 408 (4) The user was allowed to explore this functionality to their satisfaction.
- 409 (5) The user was shown the object selection menu and told to paint different objects.
- 410 (6) The user was allowed to explore this functionality to their satisfaction.
- 411 (7) The user was shown the object deletion menu and told to delete existing objects.
- 412 (8) The user was allowed to explore this functionality to their satisfaction.

413  
 414  
 415  
 416

417 After this experience, the user was asked the following set of nine 5-point linear scale questions, and told to answer  
 418 between 5 meaning "most" and 1 meaning "least":  
 419

- 420 (1) How easy was it to tell the difference between feedback from objects and feedback from surface textures?
- 421 (2) How easy was it to tell the difference between feedback from different surface textures?
- 422 (3) How easy was it to tell the difference between feedback from different objects?
- 423 (4) How easy was it to tell the difference between feedback from all sources at different zoom levels?
- 424
- 425 (5) How helpful was feedback from objects in understanding and navigating the current state of the terrain?
- 426 (6) How helpful was feedback from texture in understanding and navigating the current state of the terrain?
- 427
- 428 (7) How synchronized was haptic feedback with the terrain you saw on screen?
- 429 (8) How physically fatiguing did you find the haptic feedback?
- 430 (9) How would you rate your overall enjoyment of the experience?
- 431
- 432



444 Fig. 7. User Study Responses  
 445

446 The responses to these questions is charted in Figure 7; we provide data on four respondents, grouped by color. Overall  
 447 sentiment was positive with respect to force feedback, with all respondents rating both the utility and differentiability  
 448 of shape-driven feedback highly. Users generally found it difficult to differentiate between textures, and did not find  
 449 them helpful for terrain navigation. We conjecture improving texture differentiability would improve the usefulness of  
 450 this feedback, though in its current state textural feedback still provides users feedback on scale and rate of motion.  
 451 Users did however find it easy to discriminate scale on the basis of haptic feedback. Users generally did not find the  
 452 experience fatiguing, and found it both well-synchronized with the visual terrain rendering and enjoyable overall.  
 453

454 We conclude that users generally felt that feedback from scene objects was both salient and useful, but that textures  
 455 fell short due to difficulty in differentiation. However, the positive responses for both sentiment, fatigue, and usefulness  
 456 of shape-driven feedback supports the viability of the project.  
 457

## 462 6 DISCUSSIONS AND LIMITATIONS

### 463 6.1 Discussions

464 *The Haptic-Editing process.* Through our evaluation, we discovered that users generally appreciated shape-driven  
 465 haptic feedback, as it is salient, easily discriminated, offers practical uses like density estimation, and prevents users  
 466

469 from overpopulating terrain regions. We also learned that texture feedback as we implement it in this work is of limited  
 470 usefulness, as many users complained of poor differentiability. Use of a 2Diy interface to populate a 2D terrain object  
 471 was grasped intuitively, and none of our participants had difficulty understanding the connection between the physical  
 472 end effector and the representation of the end effector in our editor. We claim our interface would be well-suited as a  
 473 plugin to Unity, allowing game developers to continue using tools they're familiar with, but with the added benefit of  
 474 haptic feedback for terrain editing and similar applications. Indeed, force and texture feedback are both driven by parts  
 475 of the engine, colliders and normal maps respectively, that will necessarily be used by a game developer in this context,  
 476 and thus our interface requires no additional burden to use.  
 477

478     *Unity as a tool for haptics design.* Throughout history, various art forms, such as music, drawing, photography,  
 479 film, video editing, game development, and XR development, have experienced significant advancements whenever  
 480 developers have embraced user-friendly tools. These advancements have been facilitated by the availability of accessible  
 481 cameras, freely available software for music composition and video editing, as well as widely supported game engines  
 482 like Unity and Unreal. However, when the primary means of entry into these fields is limited to Java and Processing code,  
 483 the potential for designing experiences becomes constrained by the technical skills of developers, creating a bottleneck  
 484 effect. It is imperative to encourage the haptics community to explore beyond mere code frameworks and instead adopt  
 485 a singular, user-friendly engine (such as Unity) that can empower designers to focus on the user experience aspect of  
 486 haptics, rather than being bogged down by technical jargon. Such an approach would enable individuals to specialize in  
 487 various aspects of haptics, be it hardware, software, or design, akin to the specialization seen in the game development  
 488 industry, thereby enhancing the overall quality of output.  
 489

## 490     6.2 Limitations

491     *Movement in an infinite space.* With our current implementation, the end effector can only move in the virtual space  
 492 a distance proportional to the Haply device's arm length. While we can change the movement scale in the engine, the  
 493 end effector will eventually get stuck due to the haply 2DIY's limited range of motion. The primary difficulty lies in  
 494 changing the relative positioning of the proxy with respect to the world, and what the underlying user experience  
 495 design philosophy should be for the same without disorienting the user.  
 496

497     *Fine grain control of placed objects.* While we have the ability to place objects around a specific space, we do not have  
 498 the ability to move or rotate a placed object in any degree of freedom, or scale the object up and down. This was an  
 499 initial consideration our project had but had to be discarded in the interest of time and producing a working prototype.  
 500 The main issue with this comes in tackling the user interaction model to edit the transform data of an object. Since the  
 501 haply is the main mode of interaction, we could consider using it similar to a mouse, and designing movement, rotation  
 502 and scale gizmos that can subsequently be used for the editing process.  
 503

## 504     7 ACKNOWLEDGMENTS

505     We would like to thank all the professors of the CH501 course for providing us this unique opportunity of learning  
 506 and developing haptic experiences from scratch. This includes Dr. Oliver Schneider, Dr. Karon MacLean, Dr. Jeremy  
 507 Cooperstock, Dr. Pascal Fortin, Dr. Vincent Levesque and Dr. Pourang Irani. We would also like to thank Dr. Antoine  
 508 Weill-Duflos from the R&D department of Haply Inc. for providing technical assistance during the developing for the  
 509 Haply. Finally we would like to thank the Teaching Assistants Bereket Guta, Anuradha Herath, Sabrina Knappe and  
 510 Juliette Regimbal for guiding us through the various challenges in the course and our project.  
 511

## 521 8 CONCLUSION

522 Throughout the development of this project, our concept has undergone iterative refinement, integrating insights  
523 gleaned from collaborative discussions and the outcomes of prototyping endeavors. Initially, we targeted terrain editing  
524 within three-dimensional space. However, subsequent deliberations with our instructors and internal team discussions  
525 led to the realization that the Haply 2DIY platform would fail to accurately perceive depth within a three-dimensional  
526 environment, yielding poor outcomes. Consequently, we resolved to focus primarily on surface-level terrain features,  
527 directing our attention towards the painting of objects and textures on a flat surface while retaining the innovative  
528 capability to manipulate scale.

529 As a result of these findings, we positioned our project as a complementary plugin within the Unity ecosystem,  
530 specifically tailored to augment the functionality of Unity's terrain game object. Our endeavor reframes the editor as an  
531 extension of Unity's game engine, enhancing the user experience by facilitating the intuitive painting of objects and  
532 textures via the Haply 2DIY.

533 In our prototyping endeavors, we encountered the unexpected ease of transitioning from haptic texture to haptic  
534 force feedback. Leveraging Unity's foundational concepts of textures and colliders, we observed that altering the scale  
535 of the End-Effectuator representation sphere significantly influenced its collision behavior with surrounding objects. At  
536 certain scales, the End-Effectuator exhibited the ability to navigate on top of obstacles that previously blocked its progress.

537 As a proof-of-concept, we believe our work effectively showcases the future possibilities granted by haptic integration  
538 in a game development context; a sentiment supported by the encouraging results we received from our user evaluation  
539 and testing. Moreover, we present our project as proof of the extensibility offered by using Unity's robust scripting  
540 systems, which lays a solid foundation for future expansion. For instance, future work may bring improvements and  
541 additional functionalities to the painting UI such as being able to change the brush radius. Experimentation is required  
542 to counteract our current limitations and improve the utility of texture-driven haptic feedback to users of the project.

## 543 REFERENCES

- 544 [1] Roberta L. Klatzky, Dianne Pawluk, and Angelika Peer. 2013. Haptic Perception of Material Properties and Implications for Applications. *Proc. IEEE* 101, 9 (2013), 2081–2092. <https://doi.org/10.1109/JPROC.2013.2248691>
- 545 [2] Susan J Lederman and Roberta L Klatzky. 1987. Hand movements: A window into haptic object recognition. *Cognitive Psychology* 19, 3 (1987), 342–368. [https://doi.org/10.1016/0010-0285\(87\)90008-9](https://doi.org/10.1016/0010-0285(87)90008-9)
- 546 [3] Jialu Li, Aiguo Song, and Xiaorui Zhang. 2010. Image-based haptic texture rendering. In *Proceedings of the 9th ACM SIGGRAPH Conference on*  
547 *Virtual-Reality Continuum and Its Applications in Industry* (Seoul, South Korea) (VRCAI '10). Association for Computing Machinery, New York, NY,  
548 USA, 237–242. <https://doi.org/10.1145/1900179.1900230>
- 549 [4] MathWorks. 2011. What is PID Control? – mathworks.com. <https://www.mathworks.com/discovery/pid-control.html>. [Accessed 18-04-2024].
- 550 [5] Unity Technologies. 2014. Unity - Manual: Terrain – docs.unity3d.com. <https://docs.unity3d.com/Manual/script-Terrain.html>. [Accessed 18-04-2024].

## 560 A VIDEO FIGURE

561 Please find a 2-minute overview of the functionality of our project here:

562 <https://www.youtube.com/watch?v=PPx2JuhQ9Us>

## 563 B INDIVIDUAL CONTRIBUTIONS

### 564 B.1 Rishav's Contributions

565 For the final iteration, we wanted to build an executable that anyone with a haply 2diy board can plug and play.  
566 Expecting everyone to install Unity would be quite an ask, so we needed to build a live board configurator.

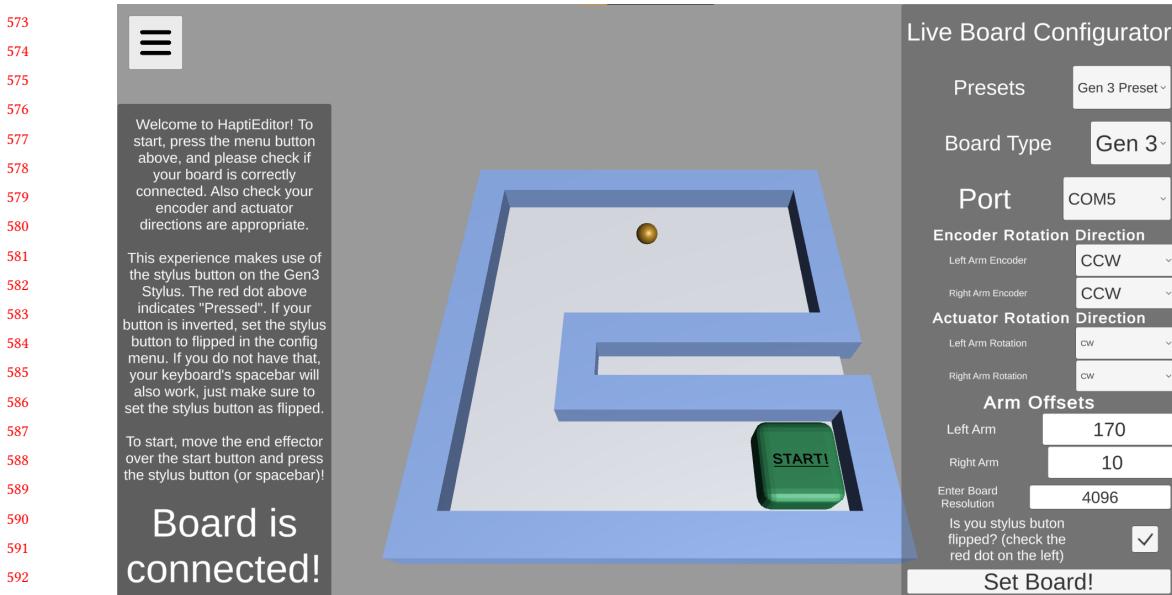


Fig. 8. HaptiEditor configuration menu

The live board configurator would need to modify the following parameters:

- Board Presets
- Gen2 or Gen3 (for arm distance offset)
- Encoder rotation direction
- Actuator rotation direction
- Arm offsets for base position
- Encoder resolution
- Flipped Stylus Button (Some Gen3 boards have flipped sensor data for the stylus port)

Actually passing the appropriate data and reloading the board was significantly more difficult. In a nutshell, I had to do the following:

- Cancel the worker thread simulation task gracefully
- Flush all forces
- Delete the existing instance of the board (along with the encoder, actuator and sensor parameters)
- Create a new board instance with the new parameters
- Attempt a connection to the new specified port based on the user's selection from current active ports
- Launch a new worker thread.
- Potentially connect to a Button Handler if the scene had one present.

This required a significant amount of refactoring of the core hAPI, but in the end I was successfully able to load and reload new user specified board configurations. The main chunk of this was happening in the `EndEffectorManager.cs` as follows:

624

```

625 1 public void ReloadBoard(DeviceConfig customConfig, string targetPort)
626 2 {
627 3     // Destroying existing setup
628 4     haplyBoard.DestroyBoard(); // added function to close port and destroy this board
629 5     CancelSimulation();
630 6     SetForces(0f, 0f);
631 7     Destroy(pantograph.gameObject.GetComponent<Device>());
632 8     // New Setup
633 9     device = pantograph.gameObject.AddComponent<Device>();
634 10    device.Init(); // re-establishes basic connections with pantograph and board
635 11    LoadBoard(customConfig, targetPort);
636 12    // Checking for button handler
637 13    ButtonHandler buttonHandler = gameObject.GetComponent<ButtonHandler>();
638 14    if (buttonHandler == null) return;
639 15    buttonHandler.SetButtonState(customConfig.FlippedStylusButton);
640 16 }
641 17
642 18 private void LoadBoard(DeviceConfig customConfig = null, string targetPort = null)
643 19 {
644 20     device.LoadConfig(customConfig); // loads new config if custom config is not null
645 21     haplyBoard.Initialize(targetPort); // attempts connection with new port
646 22     device.DeviceSetParameters();
647 23     // ...
648 24     simulationLoopTask = new Task( SimulationLoop );
649 25     simulationLoopTask.Start();
650 26 }

```

After this, I decided to improve the visual experience of the project. Users will be expected to understand that they have to configure their board, and that their board might be set up different to our dev setups, so the menu scene should ideally be different from the actual terrain editing scene. Additionally, the stylus button (or space bar) is critical to the experience, so the users should be aware of how to do that as well.

To let the user learn this separately, I worked on a simple menu scene with a bit of flair, and added scene transitions. Users will now be expected to first configure their board, be able to move over a physical button in the world, and then click on the stylus button to enter the terrain painter tool.

I fished out a basic scene manager and transition handler from an older project, and after some tweaking, blender modelling and building an executable for Windows, I produced the menu scene in [Figure 8](#).

## 661 B.2 Sean's Contributions

663 This iteration, I finalized work on the texture pipeline and integrated it with Celeste's work from iteration 2 on texture  
 664 and object painting on the terrain object. Celeste wrote some logic to get world position into a corresponding position  
 665 on the surface of the terrain object, removing the need for our expensive physics ray-cast. The rest of the process  
 666 involved the following changes:  
 667

- 669 • Detecting which terrain texture was currently painted onto the surface of the terrain underneath the end  
 670 effector. This required fetching the blend ratios of each material at the EE location and determining which was  
 671 present:  
 672 1 float[,] swatch = terrain.terrainData.GetAlphamaps((int)pixelUV.x, (int)pixelUV.y, 1, 1);
- 674 • Once I have the ID of the texture to sample, we fetch color from its normal texture, giving us both a direction  
 675 and intensity of force we immediately apply to our end effector. This greatly simplified the sampling code:

676

```

677     1 Color normalPixel = prot.normalMap.GetPixel((int)normalUV.x, (int)normalUV.y);
678     2
679     3 forces.x += normalPixel.r - .5f
680     4 forces.y += normalPixel.g - .5f;
681     5
682     6 forces *= intensity;
683     7 previousPosition = eeTransform.position;

```

- I introduced a variable `swatchscale` to allow us to control the ratio of world movement relative to movement on the normal map, controlling the linear density of our texture. Normal sampling also greatly improved the accuracy of forces, as we're no longer estimating heightmaps from surface color but now have geometry-accurate normal maps.
- Selected texturally distinct normal maps for our materials for a clearer distinction between painted materials.
- Added back space-bar support for painting, so a user has an alternative to the stylus button.

691 I then tuned the scaling code Rishav worked on in iteration 2 and added it to our scene. The following changed:

- I added a scale factor for the movement range of the end effector, so it expanded as the scene zoomed out, covering the new area.
- I then tuned the ratios for movement range, end effector size and camera zooming so that they scaled in tandem.
- I changed the painter code so that the painted objects had their appropriate colliders, allowing the large end effector to skate over rocks as we'd intended it to, rather than getting stuck like before.

699 Lastly, I tuned the texture sampling code again so that it would play nice with the zooming feature we'd introduced.  
700 I chose a `swatchscale` such that at high zoom levels, individual surface features like pebbles were quite large and  
701 discernible, but with a wider camera, surface features became higher frequency noise and force feedback from geometry  
702 became the primary force on the end effector. Textures representations were different depending on the painted texture  
703 on the terrain and could be rendered in tandem with terrain objects placed during use. We learned that these pipelines  
704 could all coexist in a cohesive way and were pleased to see our parallel approach to developing them paid off.  
705

706 Post iteration 3, I fixed some outstanding bugs, worked on my section of the report, and prepared the video figure.

### 709 B.3 Celeste's Contributions

711 As stated in our blog posts for iteration 2, the goal for the third was to merge every concepts and prototypes into  
712 one single, preferably enjoyable, experience. In this iteration we ended up working together way more and in closer  
713 collaboration by helping each others a lot more. This can be easily explained because we didn't prototype on a different  
714 aspect of the experience and were actually making something unique together. This is why sometimes the lines of  
715 contributions of what I did and what a team member did will blur into what we did this feature together.  
716

717 While we knew that we would merge all of our progress into the Terrain scene because the end goal is to use Unity's  
718 Terrain game object has our editor, the `TerrainScript.cs` from iteration 2 wasn't usable as is. Firstly, we need to fix  
719 the lack optimizations. Secondly, the user should have a way to change the brush type and their specifics without using  
720 Unity editor menus (it is necessary if the user interacts with our software through an executable instead of the Unity  
721 Project). Thirdly, Lastly, it is unlikely the code we used for texture sampling for haptic feedback would work on the  
722 Terrain game object, because it has the particularity to not have a `MeshRenderer` (a component that allows a game  
723 object to render a mesh). Finally, there needs to be a way paint using the stylus button instead of the mouse.  
724

725 During this iteration Rishav did an amazing work at going over the code that has been made and telling us what  
726 should be avoided in the future. Following those practices we started a refactoring on `TerrainScript.cs`. During  
727

728

729 this time, I found a way to optimize the management of the TreeInstances what were giving us issues during the  
 730 second iteration, basically, deleted TreeInstances would never really be deleted but replaced with empty version of  
 731 themselves.  
 732

733 This is problematic because with the core logic of the problem they would be given another collider the next time  
 734 the software is running even though there is nothing to delete.

```
735 1 private void OnApplicationQuit()
736 2 {
737 3     //test
738 4     List<TreeInstance> trees = new List<TreeInstance>(terrain.terrainData.treeInstances);
739 5     List<TreeInstance> trees_cleaned = new List<TreeInstance>();
740 6     TreeInstance empty_tree = new TreeInstance();
741 7     for (int i = 0; i < trees.Count; i++)
742 8     {
743 9         if (!trees[i].Equals(empty_tree))
744 10             trees_cleaned.Add(trees[i]);
745 11     }
746 12     terrain.terrainData.SetTreeInstances(trees_cleaned.ToArray(), true);
747 13 }
```

747 Using this code when the application is closed we remove all of the TreeInstances that we could consider empty.  
 748 The way it works is by going through all the objects in the Terrain TreeInstances and comparing it with an empty  
 749 object. If they are different we add them to the new list that will contain all the non-empty TreeInstance.  
 750

751 Then using the ObjectPlacer.cs as a reference I created two co-routines one for painting object on the terrain and  
 752 the other for painting texture. We implemented an enumerator containing all the brushes types (i.e. Texture, Object and  
 753 Object Eraser) and depending on this brush type one of the co-routine or the object deletion will be enabled.  
 754

755 From then on Rishav worked with me on a basic UI so we can change the brush types and which texture/object is  
 756 being painted.  
 757

## 758 C PREVIOUS ITERATIONS

759 For further reading into the development of this project, links to the previous iterations can be found on the Project  
 760 Winnipeg website.  
 761

762 (1) Iteration 1:

- 764 • Celeste's Blog: Iteration 1
- 765 • Sean's Blog: Iteration 1
- 766 • Rishav's Blog: Iteration 1

767 (2) Iteration 2:

- 769 • Celeste's Blog: Iteration 2
- 770 • Sean's Blog: Iteration 2
- 771 • Rishav's Blog: Iteration 2

## 773 D ARTIFACT

774 Source code can be found at <https://github.com/780nm/HaptiEditor/>