Blocks-to-CAD: A Cross-Application Bridge from Minecraft to 3D Modeling

Ben Lafreniere*, Tovi Grossman*†

*Autodesk Research, Toronto, ON, Canada
{*firstname.lastname*}@autodesk.com

†University of Toronto, ON, Canada
tovi@dgp.toronto.edu

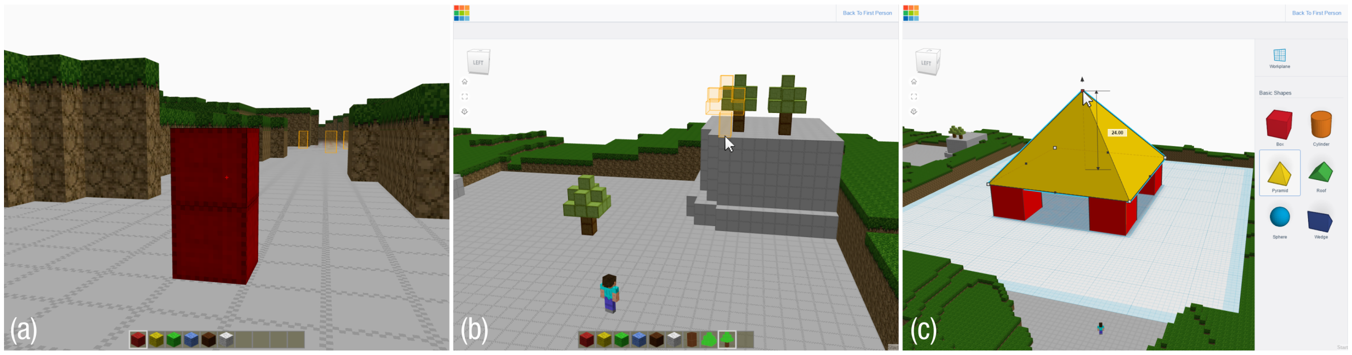


Figure 1. A cross-application bridge from Minecraft-style games to Tinkercad-style 3D solid modeling. (a) The player starts out in a Minecraft-like voxel world; (b) over time, tools are introduced which alter the interaction model and introduce 3D-modeling concepts; (c) eventually, the player transitions to Tinkercad-style 3D solid modeling.

ABSTRACT

Learning a new software application can be a challenge, requiring the user to enter a new environment where their existing knowledge and skills do not apply, or worse, work against them. To ease this transition, we propose the idea of *cross-application bridges* that start with the interface of a familiar application, and gradually change their interaction model, tools, conventions, and appearance to resemble that of an application to be learned. To investigate this idea, we developed Blocks-to-CAD, a cross-application bridge from Minecraft-style games to 3D solid modeling. A user study of our system demonstrated that our modifications to the game did not hurt enjoyment or increase cognitive load, and that players could successfully apply knowledge and skills learned in the game to tasks in a popular 3D solid modeling application. The process of developing Blocks-to-CAD also revealed eight design strategies that can be applied to design cross-application bridges for other applications and domains.

Author Keywords

Feature-rich software; software learning; skill transfer.

ACM Classification Keywords

H.5.m. Information interfaces and presentation: Misc.

UIST '18, October 14–17, 2018, Berlin, Germany
© 2018 Copyright is held by the authors. Publication rights licensed to ACM.
ACM ISBN 978-1-4503-5948-1/18/10…$15.00
https://doi.org/10.1145/3242587.3242602

INTRODUCTION

Learning a new software application can be difficult, presenting users with a range of challenges [18]. When faced with a new application, users are known to have a *production bias* – progress toward achieving goals is the paramount concern, and they have little motivation to spend dedicated time on learning – and an *assimilation bias* – they apply what they already know to interpret new situations [7, 15]. In early software learning research, these phenomena were used to argue for techniques that enable users to get started quickly, and relate knowledge from the non-software world to the concepts being learned [5]. Thirty years later, we live in a world where people's expectations about software are shaped heavily by their experiences with other software – a child may grow up playing video games and using simple apps before moving on to more sophisticated software as required by their changing interests, careers, and creative endeavors.

Motivated by the above, we are interested in how a user's experience with an existing software application can be used as a foundation for expanding their knowledge into new and unfamiliar software and domains. We propose a model of *cross-application bridges* that start with the interface of a known application, and gradually change their interaction model, tools, and conventions to resemble that of an application being learned. Cross-application bridges are analogous to training wheels [6] and multi-layered interfaces [36], but while such techniques progressively disclose UI components to aid learning within a *single* interface, cross-applications bridges aid in learning across multiple user interfaces. Changes are triggered by user behavior, to provide new capabilities in motivated learning scenarios where they are most likely to be understood and appreciated. The overall idea is to help users stay motivated by embracing their production and assimilation biases, allowing them to gain new skills and capabilities while completing personally-relevant tasks in a familiar interface.

To investigate this concept, we designed and developed *Blocks-to-CAD*, a cross-application bridge that starts in a Minecraft-style voxel building game and gradually introduces 3D solid modeling in the model of Tinkercad (Figure 1a-c). This application of the cross-application bridge approach is appealing for several reasons. First, Minecraft is extremely popular, having sold over 144 million copies since its release in 2011, and with more than 74 million monthly players. This suggests that a huge base of users possess Minecraft skills, and could benefit from a technique that can leverage their experience to develop skills in other applications. Second, Minecraft is fundamentally about building in 3D, albeit using a block-by-block approach, which suggests that some of its players may be interested in learning more sophisticated 3D modeling skills.

The specific contributions of this work are as follows:

- We introduce the concept of *cross-application bridges* – interfaces that gradually transform from a known application's interface into a target application's interface, introducing new features and capabilities in motivated learning scenarios triggered by user behavior;
- We present a prototype system that implements this concept, to transition users from a Minecraft-style building game to 3D solid modeling;
- We present an evaluation of our system with children ages 10–14, which shows that it can effectively teach 3D modeling concepts without hurting enjoyment of the game;
- Based on our experience, we present eight strategies that designers can apply to develop cross-application bridges for other applications and domains.

RELATED WORK

This work builds on existing research on software learning, reduced functionality interfaces, techniques for transferring knowledge between applications, and theories of learning and education. Each of these areas is reviewed below.

Software Learning

Early HCI research on software learning pointed out the failures of manuals and documentation [5, 7, 33], and identified a *task-focus* in users – progress toward goals is paramount, and time spent on other concerns, such as learning how to use the software, is minimized to the greatest extent possible [7, 34]. At a lower level, Grossman et al. identified five common classes of problems that users face when learning modern feature-rich software applications: understanding the sequence of operations for a task, awareness of functionality, locating functionality, understanding how to use specific tools, and transitioning to efficient behaviors [18].

Building on the above work, a variety of techniques have been developed to promote software learning, with many focused on providing minimal instruction and helping users to learn in the context of realistic tasks. Examples include in-context help and tutorials [12, 17, 21], animated and video instruction [1, 30], game-based training [11, 25], tutorials that react to a user's progress or skill level [12, 30], and methods for linking web-based help content with an application's interface [14]. However, little research in the software learning literature has explored how knowledge can be transferred *between* applications. A notable exception is ShowMeHow [31], which builds translation maps between the interface languages of similar applications (e.g., GIMP and Photoshop) to help users locate commands in one application using the vocabulary of another. The authors demonstrate that this allows users to more quickly find corresponding functionality between applications, and to apply tutorials intended for one application to a similar application.

In contrast to ShowMeHow, we are interested in gradually transitioning a user between applications, by leveraging their existing knowledge and the constraints and similarities in the conceptual models of the source and target applications. The idea of using gradual UI transitions to aid software learning has been explored within a single application in the form of "training-wheels" techniques that progressively reveal functionality, as described below.

Training Wheels and Multi-Layered User Interfaces

For users in the early stages of learning an application, the complexity of the interface can be a hindrance. Recognizing this, researchers have proposed *training-wheels* for interfaces [6], which limit functionality to prevent common or particularly troublesome error states, and *multi-layered interfaces* [36], which enable progressive revelation of an system's full functionality by organizing the interface into multiple conceptual layers that the user can switch between.

Reduced-functionality approaches have been shown to enable faster learning [2, 13, 23], and to reduce errors and time spent on error recovery while learning [6]. When it comes to transferring knowledge, some studies have indicated that this approach improves a user's ability to perform in the full system [8], while others show effects only for certain types of interfaces, such as those with deeply nested menus [24], or no change in performance on new tasks learned in the full system [23]. Overall, however, work in this area suggests that it is valuable to make a user's first steps in working with a new application easy by reducing the system's complexity, and by providing learning materials and guidance [24].

This form of learning is often successfully employed in games, which use progressive disclosure to match the level of challenge to a player's developing skills [9, 37]. Gamification mechanics have now been introduced into many non-game contexts [10], including software learning systems [11, 25]. In particular, our work was inspired and informed by Dong et al.'s discovery-based puzzle game to help people learn Adobe Photoshop features [11], and Bruckman's design recommendations for educational games [3].

Cross-application bridges can be viewed as a type of adaptive multi-layered interface, in that the system initiates changes to the interface that progressively reveal functionality over time. However, unlike existing work in this area, functionality for a new application is revealed in the interface of a known application. This enables the user to learn in a familiar environment, where they can fall back on their existing skills, and enables the system to use the familiar environment as a context for motivated learning scenarios that reveal new functionality.

Theories of Learning and Education

Several prominent theories of learning and education suggest that a task-centric focus toward learning, as well as being favored by users, is beneficial for learning. The theory of *situated learning* emphasizes that learning should take place in authentic settings [32]. *Constructivism* [29] positions learners as active sense makers who seek to build coherent and organized knowledge, and can thus choose situations to manipulate and discover where their current conceptions conflict with observations. Likewise, *discovery learning* [4, 11] advocates for active participation in the learning process, and *constructionism* [27] posits that learning occurs "most felicitously" when constructing a public artifact "whether a sand castle on the beach or a theory of the universe." [28] The overall concept of cross-application bridges embraces learning in the context of active participation, and Blocks-to-CAD is specifically built around the idea of construction.

Though theories such as constructionism and discovery learning are sometimes interpreted as prescribing a fully hands-off approach, it has been argued that "…students need enough freedom to become cognitively active in the process of sense making, and […] enough guidance so that their cognitive activity results in the construction of useful knowledge." [26] Methods that allow active exploration but also guide the construction of useful knowledge (*guided discovery methods*) are advocated to satisfy these criteria.

The cross-application bridges approach naturally fits with the model of guided discovery. Our prototype system is designed to transition users between interactive systems using a series of discoveries, supplemented with guidance in the form of short videos designed to motivate and communicate to the user how the interaction model has changed.

CROSS-APPLICATION BRIDGES

Cross-application bridges incrementally introduce the interaction model of an unfamiliar target application within the interface of a known application that is familiar to the user. Instead of forcing the user to switch to an unfamiliar application and engage in a *performance dip* [35] – a frustrating period of orientation and re-learning of basic skills (Figure 2a) – the user can learn through a series of motivated learning scenarios that introduce new knowledge, capabilities, and skills, all within a familiar environment (Figure 2b). Expanding on the above, our model for cross-application bridges is built on three main design principles:

Build on the known application. First, we wish to inject learning of the target application into the known application, which the user has experience using to perform personally-relevant tasks. This gives the user a familiar environment to work in as they learn, and enables them to fall back on their existing skills and abilities during the process. It also shifts the framing of the experience – instead of being dropped into an unfamiliar interface where their existing skills may not apply, the user learns in a familiar domain where new capabilities are provided that augment their existing skills.

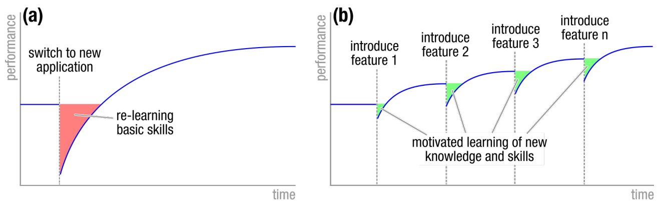


Figure 2. Two models for switching between a known and target application. (a) Switch outright, which imposes a cost in re-learning basic skills. (b) Transition from the known application, through a series of motivated transition tasks.

Present new capabilities in motivated learning scenarios. Second, the system should introduce capabilities from the target application in scenarios where they provide an obvious advantage, or illustrate a key difference between the known and target applications. By doing so, the system should help the learner to discover how the interaction models of the two applications differ, and foster an appreciation for the advantages of the target application's interaction model.

Gradually transition between interaction models. Finally, the system should create a gradual transition toward the interaction model of the target application. To keep the transition gradual, it may be necessary to develop intermediate interaction models between those of the two applications.

The above design principles informed the design of our prototype cross-application bridge between Minecraft-style building games and Tinkercad-style 3D solid modeling. Before describing our prototype system in detail, we briefly introduce these two applications.

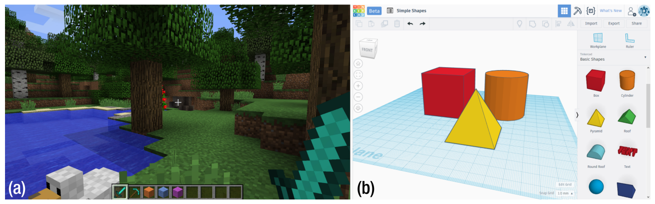


Figure 3. The user interfaces of (a) Minecraft, (b) Tinkercad.

FROM MINECRAFT TO TINKERCAD

Minecraft is a sandbox game in which players explore, interact with, and build in a procedurally-generated voxel world. Players start the game with no instructions, and engage in a simple form of 3D building in which the world is manipulated one block at a time. The game is played from a first-person perspective (Figure 3a) in which the player can walk around and create or remove blocks up to a set distance in front of them.

Tinkercad (Figure 3b) is a 3D solid modeling application that enables users to create, manipulate, and combine 3D primitives. Primitive shapes are dragged from a drawer on the right side of the screen onto a workplane. Shapes can be manipulated (scaled, resized), repositioned, and combined with one another to form more complex models.

While Tinkercad's interface is designed to be simple, past work has shown that it can still be difficult to learn and use by novice users [20]. To use Tinkercad effectively, a user must understand a range of skills, including object-centric 3D navigation to move the camera around the scene; interaction mechanisms to reposition, resize, and rotate objects; and additive and subtractive Boolean operations to combine primitives into more-complex shapes. A particularly important concept in Tinkercad is the *workplane*, which defines ground plane relative to which other operations occur (the blue grid in Figure 3b). Primitives are placed on the workplane when created, and dragging a shape moves it across the workplane. To place shapes on the faces of other shapes, a *workplane tool* can be used to set the workplane to be parallel to a face of an object in the scene. Past work has shown that understanding the workplane tool is a particular source of difficulty for new users of Tinkercad [20].

In terms of skills that overlap between Minecraft and Tinkercad, both require the user to conceptualize and plan how to build in 3D spaces, though they differ significantly in how these building activities are carried out. As a starting point for the design of our bridge from Minecraft to Tinkercad, we now consider how these applications differ in terms of application domain, data representation, and interaction paradigms.

Application Domains

Both Minecraft and Tinkercad enable the user to build objects in a 3D environment, but they differ in how this activity is framed. Tinkercad provides an environment for designing models for 3D printing, whereas Minecraft provides a virtual world with minimal restrictions placed on the user, and allows them to define their own tasks. That is, Tinkercad, like most CAD software, is used with a mainly extrinsic motivation, whereas activities in Minecraft are intrinsically motivated. This suggests a potential benefit for users learning Tinkercad through a cross-application bridge from Minecraft – it can provide an intrinsic motivation for their learning activities that would not exist in Tinkercad alone.

In addition to differences of motivation, we hypothesize that a key feature that makes building in Minecraft compelling is a flat difficulty curve. The difficulty of creating a given shape in Minecraft is proportional to the number of blocks that make it up, with the complexity of the object outweighed by the time it takes to place the individual blocks. In contrast, in Tinkercad, the complexity of an object plays a much greater role in determining how successful a user will be in building something, because creating geometrically complex objects requires more sophisticated operations and manipulations than simpler objects. The result of this difference may be that beginners with Minecraft are less limited in the classes of objects they can create – complex objects are as easy to create as simple ones, with the main constraining factor being the number of blocks a design is made up of. In contrast, beginner users of Tinkercad may be able to easily conceive of designs that are unreachable with their current skill set. This also highlights a disadvantage of the Minecraft model of building – the time to build an object is proportional to the number of blocks that make it up, regardless of its complexity, making it difficult to build large structures, even if they are not complex (e.g., a tall cylindrical tower).

The above discussion suggests a specific strategy that can be used by a cross-application bridge between Minecraft and Tinkercad – tools and capabilities from Tinkercad can be presented as saving time when many blocks need to be created or removed.

Data Representations

Minecraft represents the world as a three-dimensional grid of voxels, whereas Tinkercad represents the scene as a collection of solid shapes. As described earlier, this has implications for the difficulty of creating different objects in the two applications. This difference also makes it much more difficult for a user to modify an object built in Minecraft, as any alterations must be done block-by-block, and there are no easy methods to move, resize, or duplicate an object once it has been created. In contrast, objects created in Tinkercad can be easily manipulated after they have been created.

To address this discrepancy, our system adopts a two-way mapping between the voxel and 3D solid model representations, so that users can work with either representation and see the advantages of each.

Interaction Paradigms

In terms of interaction paradigm, the controls to Minecraft are like those of most first-person games – the mouse cursor is "locked" to the center of the screen, and mouse movement maps to looking up and down or rotating the player left and right, with keyboard controls to walk forward, backward, or laterally left and right. The currently-equipped tool or block type can be changed using keyboard commands to open an inventory screen, or through use of the mouse wheel to select items in a toolbar visible at the bottom of the screen.

Tinkercad, in contrast, follows standard WIMP interaction conventions. The mouse cursor is not locked, and is used to interact with toolbar buttons, to drag primitives into the scene from the shape drawer, to select objects in the scene, and to directly manipulate selected objects using resizing, repositioning, and rotation handles. The 3D scene is navigated using object-centric controls centered on a pivot point, and allows panning, zooming, and orbiting with a ViewCube widget [22] or by clicking and dragging the mouse (right mouse button + drag to orbit, middle mouse button + drag to pan, scroll wheel to zoom in/out).

Given the extensive differences between the interaction paradigms of the two applications, we adopted a multi-step approach to bridge the two interaction models, where new UI concepts are introduced gradually.

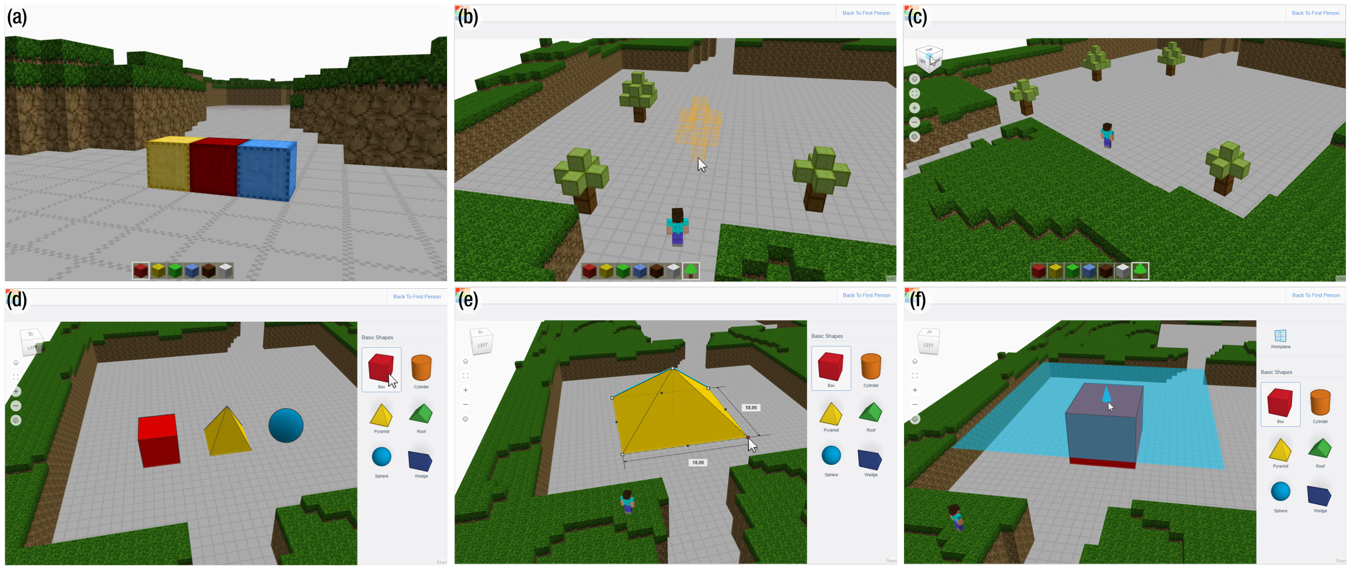


Figure 4. The six stages of functionality in the Blocks-to-CAD system. (a) Minecraft-style block tools; (b) Tree-stamp tool; (c) 3D navigation widgets; (d) Shapes tool; (e) shape-resizing handles; (f) Workplane tool.

SYSTEM DESCRIPTION

When Blocks-to-CAD is started, the user begins in a traditional Minecraft-style interface (Figure 4a). They can walk around the world in a first-person perspective, create blocks from a set of six colors, and remove blocks one at a time.

As they interact with the world, the system monitors their activity and new tools and functionality are gradually unlocked. We start by describing the progression of tools and capabilities that are unlocked, followed by the mechanisms and criteria used to determine when tools are unlocked.

Tree-stamp tool. The first tool to be unlocked enables the user to stamp a multi-block tree shape in one click (Figure 4b). The tool is added to the player's toolbar, and can be selected as if it was another block type. When the tool is used, the point of view transitions to a third-person view of the scene, centered on a pivot point where the player was looking when the tool was activated. The mouse cursor is unlocked, and the user can stamp multiple tree shapes by clicking the corresponding point in the scene. Clicking a 'Back to First Person' button re-locks the mouse cursor and returns the user to the first-person view.

3D navigation widgets. The next tool to be unlocked is Tinkercad's set of 3D navigation widgets, including the ViewCube and a set of additional controls (Figure 4c). These widgets are displayed only when the user is in third-person mode (i.e., when using the tree-stamp tool or other third-person tools described in the rest of this section), and enable the user to rotate the camera around the scene's pivot point, or zoom in/out. In addition to 3D navigation with these widgets, Tinkercad-style 3D navigation using the mouse can be used in the third-person view.

Shapes tool. The Shapes tool operates similarly to the tree-stamp tool when invoked, changing the player to the third-person perspective. For this tool, however, a shape drawer is displayed on the right side of the screen (Figure 4d). The user can drag 3D solids from this drawer into the world. Once in the world, solids can be clicked and dragged to move them. Upon returning to the first-person view, shapes created in this way are 'voxelized', converting them into blocks (Figure 5). However, the shapes retain their dual representation – if the player returns to the third-person view, they are displayed as 3D solids again, and can be moved or manipulated. Maintaining this dual representation was intentional, to emphasize one of the advantages of 3D solid modeling over building in Minecraft – namely that shapes can be easily repositioned or manipulated after being created. As mentioned previously, achieving the same goal in Minecraft alone would be challenging, requiring the user to re-create each constituent block of a large shape in a new location.

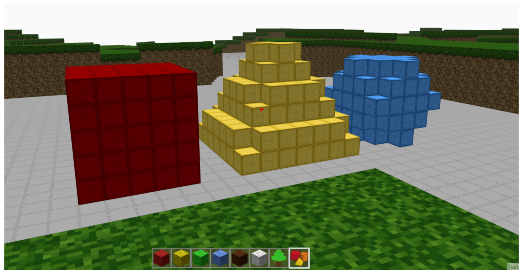


Figure 5. The shapes from Figure 4d, converted into blocks upon the user returning to first-person mode.

Shape resizing. Next, players gain the capability to resize solids in the third-person perspective. Resizing handles, matching those provided in Tinkercad, are provided at the corners and top of solids (Figure 4e). Clicking and dragging these handles alters the dimensions of the solid.

Workplane tool. The final tool to be unlocked is the Workplane tool – a button for this tool is added to the shapes drawer, just above the list of solids (Figure 4f). As described earlier, the Workplane tool in Tinkercad allows the user to set the 3D plane relative to which other operations are performed (e.g., when the user clicks and drags an object, it moves across the current workplane). In practice, this enables the user to easily place solids on top of one another (e.g., by setting the workplane on top of an existing solid, before creating the new solid), or on the faces of other solids.

While the above tools do not represent all functionality available in Tinkercad, we believe they collectively provide a good representation of 3D solid modeling. Moreover, our approach could easily be extended with additional tools and capabilities, such as functions for grouping objects or performing additive and subtractive Boolean operations.

In the next section, we discuss how our prototype system determines when to unlock new tools and capabilities.

Unlocking Features

The progression of tools described in the previous section fulfills two of our design goals – building on the known application, and creating a gradual transition between interaction models. To fulfill our final design goal of creating motivated learning scenarios, we adopted a behavior-driven approach to determine when features should be unlocked. Our intention was for tools to be unlocked in situations where the user would be able to appreciate the advantages that the new tool provides. To this end, we used two strategies for determining when to unlock a feature.

Content-based unlocking. The first approach we employed was to detect when the user was building a particular kind of object with blocks, or engaging in a particular type of building activity, and to unlock tools that could help with that activity. We developed heuristics to detect when the user was building trees, basic shapes, and 'stacked' structures consisting of a pattern of blocks repeated on top of one another (as would be used to build a large structure from the ground up).

To implement this approach, the system tracks all blocks created by the user. When a block is created, the region of user-created blocks connected to that block are determined by performing a 3D flood fill algorithm from the new block's position. The following heuristics are then applied to the resulting connected region of blocks:

- *Trees* – Region is at least 4 blocks high, and contains at least twice as many green blocks above the midpoint as brown blocks below the midpoint.
- *Shapes* – Region is matched by a sliding window of templates for cubes, pyramids, and spheres of up to 4×4×4 blocks.
- *Stacked structures* – Region consists of at least 4 layers of an identical 2D pattern of at least 4 blocks, stacked vertically.

While we used manually-created heuristics for content-based behavior detection, we believe that more sophisticated approaches based on machine learning could learn heuristics over time, or detect the type of object a user is building.

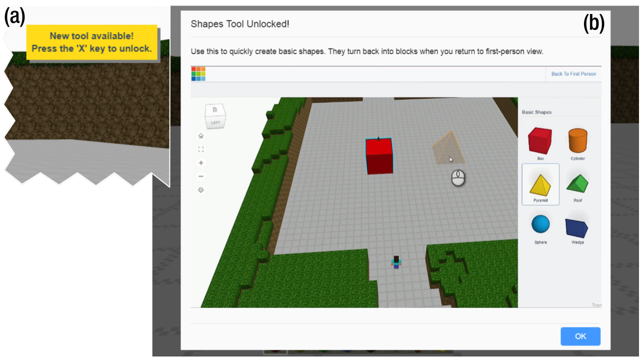
Operation-based unlocking. The second approach we used was to detect the number of operations of various types the user has performed (e.g., the number of blocks created, or instances of moving/resizing a shape). Our rationale is that it is valuable to wait until the user becomes familiar with each new tool or capability before unlocking the next.

Drawing together the above approaches, the unlocking criteria we used in our system are set out in Table 1. Counts are reset after each tool is unlocked. These criteria were tuned to enable participants to experience all tools within the timeframe of the user study described later in the paper. For other settings, such as users playing Minecraft over hours, days, or weeks, a different tuning may be more appropriate.

Tool	*Unlocking criteria*
Tree-stamp tool	2 heuristic trees *or* 40 blocks created
3D nav. widgets	2 trees stamped *or* 80 blocks created
Shapes tool	2 heuristic shapes *or* 80 blocks created
Shape Resizing	1 heuristic stacked structure *or* 2 shape moves *or* 80 blocks created
Workplane tool	2 shape resizes *or* 6 shape moves *or* 80 blocks created

Table 1. Tool unlocking criteria used in our system.

In terms of the mechanism for unlocking new tools and functionality, a notification is displayed in the corner of the user's screen (Figure 6a). When the user presses a key to unlock the tool, a modal dialog is displayed with a short 15–60 second video loop demonstrating the newly-unlocked capability. This mechanism was designed to feel like unlockable "achievements" that are often used in video games, rather than formal training materials.



Shapes Tool Unlocked!

Use this to quickly create basic shapes. They turn back into blocks when you return to first-person view.

Figure 6. When the threshold for unlocking a tool is met, a notification is displayed in the corner of the screen (a). When the user unlocks the tool, a modal dialog with a short video demonstrating the tool is played (b).

System Implementation

The system was implemented in JavaScript using three.js and a range of other JavaScript libraries. The voxel game component was based on the open-source voxel-engine[1] project, extensively modified and customized for the project. The embedded CAD functionality was built on a closed-source 3D editor library on which Tinkercad is built, used with permission and substantially modified for the project.

Though both voxel-engine and the Tinkercad editor are built using JavaScript and the three.js library, it was challenging to combine them. In particular, it was tricky to manage user input, as both applications were developed assuming raw access to keyboard and mouse events. To address this, our

[1] https://github.com/maxogden/voxel-engine

system uses a state-based representation to manage which application receives input events at any given time, with states for the first-person and third-person interaction modes described previously. We believe that this state-based approach could be applied in creating cross-application bridges between other applications as well.

While our project incorporates closed-source code, we believe that the cross-application bridge approach is well-suited to the open-source software ecosystem, in which the full source code of multiple applications is available to modify and extend.

EVALUATION

We conducted a user study with two main goals. First, we wanted to understand reactions to the unlockable features, including whether users would incorporate them into their building activities in the game, and how they would impact the subjective experience of the game. Second, we wanted to test whether use of the unlockable tools in the game translated into skills that could be transferred to the Tinkercad application.

Study Design

Our study followed a between-subjects design. In the experimental condition, participants used Blocks-to-CAD with unlocking of features enabled. In the control condition, participants also used Blocks-to-CAD, but unlocking was disabled (i.e., participants had access to the basic block tools only, for the duration of the study). Comparing these two conditions enables us to understand the impact our system has on learning Tinkercad, in comparison to someone who only has experience playing Minecraft-style games.

Study Procedure

The study began with a *game phase* in which participants spent 25 minutes playing Blocks-to-CAD. For this phase, they were given the task of creating "parks" in each of a series of square-shaped open areas in the game world. Participants were instructed to create a park in as many of the squares in the world as they could in the time provided. Each park had to meet four requirements: (1) four trees, around the outside of the park; (2) two shapes – spheres, boxes, triangles, etc. of dimensions at least 2×2×2; (3) a tower at least 2×2 blocks wide and 10 blocks tall; and (4) a simple house consisting of a box of size 3×3×3 or greater and a triangular roof. Participants were given a handout explaining these requirements with example screenshots (see supplementary materials), but were told that they did not have to exactly reproduce these examples, so long as all the components were present. The experimenter checked each park before allowing the participant to begin building the next.

In the experimental condition, participants were told that the game had unlockable features, and that a notification would be displayed in the corner of the screen if one was unlocked. If a participant had an unlocked tool pending when they completed a park (i.e., the unlock notification displayed), the experimenter reminded them that they could open it with the 'X' key before they went on to start the next park. In practice, this was a rare, occurring only twice across all participants.

The game phase was followed by a *transfer phase* in which participants attempted a series of 3D modeling tasks in Tinkercad. Participants were given 25 minutes in total to complete three tasks: (1) creating three primitive shapes in a specified arrangement (to test basic shape placement); (2) creating a box and three pyramids of specified sizes (to test shape positioning and resizing); and (3) reproducing a simple car model (to test more-advanced orientation and placement of shapes). The handouts for each task are provided as supplementary materials with this paper. Tasks were presented serially in the order above.

Participants were not given instruction on how to use the software, apart from being told that they could delete shapes by selecting them and using the 'backspace' key. The experimenter stated that they could not provide help with performing the tasks, but that they could confirm if a model was acceptably close to the reference model.

A short questionnaire was administered after each of the game phase and the transfer phase, to measure how fun participants found each part of the study, and their cognitive load (using an adapted version of the NASA-TLX questionnaire [19]). For the game phase, we asked participants in the experimental condition to rate how useful, fun, and annoying/disruptive they found the unlockable features. For both conditions, we asked participants to tell us what they liked about the game, and how they felt it could be improved or made more fun.

Participants

We recruited 12 participants (6 male, 6 female, ages 10-14, mean 11.7, SD 1.2) from two pools: children of employees of a large software company, and members of a local children's soccer team. Participants were screened to ensure that they had experience playing Minecraft, but no experience using Tinkercad. Participants were given a $25 gift card for participating. We balanced the ages of participants across the two conditions, to control for learning ability.

Best practices for working with young participants were followed, including requiring parental consent to participate in the study, and gaining assent to participate from the participants themselves. Based on our observations, the study was a fun experience for participants.

Results

Game phase – Use of unlockable features

To analyze reactions to the unlockable features, we look at whether the features were used, how the features were used, and participants' subjective reactions to them.

All but one participant in the experimental condition unlocked all five of the unlockable features. Figure 7 shows a timeline indicating participants' use of the third-person mode over time. We can see that use of the unlockable features increases over time, to a point where near the end of the session many participants were heavily using the unlockable tools. Overall, participants in the experimental condition spent an average of 26% of their time in the third-person mode, with individual participants ranging from 5% (P3) to 50% (P2).

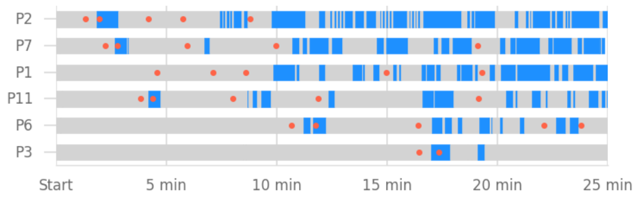


Figure 7. Timelines for participants in the experimental condition, indicating use of Tinkercad mode (blue) and points at which tools were unlocked (red dots). Participants are sorted by total use of the Tinkercad mode.

P3, the one participant who did not unlock all the tools, appeared to be somewhat overwhelmed with figuring out the game's control scheme and how to perform the building tasks. However, she did not express a negative view of the unlockable features, and seemed interested in the tree-stamp tool when she did try it.

In summary, nearly all participants appeared to immediately recognize the value in unlockable features, and incorporated them into their building activities.

Game phase – Subjective ratings and impressions

In terms of subjective ratings, participants rated the unlockable features highly when asked about their usefulness and fun, and low for annoyance or disruption (Figure 8).

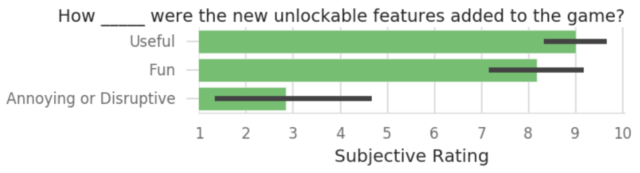


Figure 8. Subjective ratings of the unlockable features.

When asked about what they liked about the overall system, several participants mentioned the unlockable features:

> *I liked that you could unlock stuff, because it helped build and it was really nice. – P6*

> [P7, when asked what she liked about the system]: *How you could unlock the features, and it made it easier, to do it. Because the first time I was, like, making, I had to make them myself. Next time after I did that [unlocked the features] it was a lot easier, I just had to place them down. – P7*

When we asked participants what they felt could be improved about the system, or how it could be made more fun, none of the participants mentioned the unlockable features as a negative. Instead, suggestions focused on features of the full Minecraft game that could be added to our system, such as additional block types; enemies and other survival aspects; and a more-realistic sky texture. None of the participants who experienced the unlockable features suggested that they made the system feel less game-like, and participants talked about the features as though they were part of a game.

One participant, P11, wanted to be able to modify the blocks created using the shapes tool after returning to the first-person mode, and have these changes persist (in our prototype system, blocks modified in voxelized Tinkercad shapes do not persist in this way). This highlights how including multiple data representations and interaction models can create the expectation that they will work together seamlessly.

Comparing participants' subjective ratings of the game between conditions, we did not find a significant difference between cognitive load (Welch two-sample t-test, t(9.9)=0.4, p = .67) or subjective ratings of fun (t(6.2)= -0.2, p = .87), with similar means (Figure 9). This provides further evidence that the modifications did not make the game seem more challenging or less enjoyable.

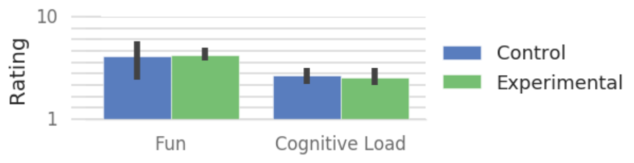


Figure 9. Subjective ratings of the overall game experience, and the cognitive load (based on 5 NASA-TLX questions)

In summary, participants' subjective ratings and comments indicate that the unlockable features enhanced the game experience, were welcomed and utilized by participants, and did not have adverse effects on enjoyment or cognitive load.

Transfer-phase

To analyze whether the unlockable features in the game resulted in skills transfer to Tinkercad, we examine task completion and task time, and present observations of how participants worked on the transfer tasks in the experimental versus control conditions.

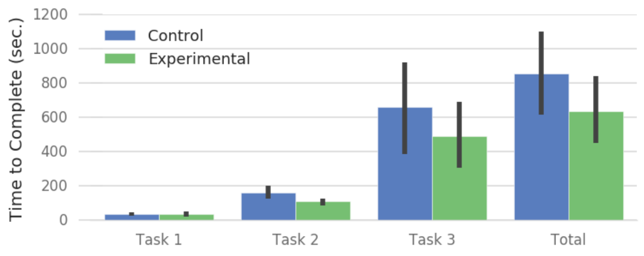


Figure 10. Timing for transfer tasks.

All participants, across both conditions, completed all three transfer tasks. Task completion times for Task 1 were nearly identical between conditions (mean of 35s for both). For Tasks 2 and 3, the mean completion times were lower for the experimental condition (109s vs. 161s, and 637s vs. 855s respectively). However, individual times varied widely, and Welch two-sample t-tests failed to show a significant difference (*Task 2*: t(7.4)=2.1, p = .07; *Task 3*: t(9.1)=0.9, p = .39).

Qualitatively, we observed that participants in the two conditions worked differently on the transfer tasks. None of the participants in the control condition used the workplane tool, while all but one participant used it in the experimental condition. The exception to this was P3, the participant who did not unlock this tool in the game phase, as discussed previously. Moreover, these participants demonstrated through their actions that they understood how the workplane tool was meant to be used, and how it could be applied the transfer task. Participants in the experimental condition also appeared more confident in moving and resizing shapes.

Given these qualitative differences, the lack of an observed significant difference warrants discussion. We believe there are two reasons a statistical difference was not observed. First, our sample size is small for a between-subjects study. Second, 3D modeling tasks naturally lead to a high variance in task times. Even if a user has determined a fundamentally correct strategy to use, it can be easy to run into difficulties while carrying out that strategy. For example, accidentally manipulating the wrong resize/rotate handle, or clicking and dragging the wrong shape, can result in modifications that are time-consuming to recover from. We observed this frequently during the study, and in-practice this greatly increases the variances of task times for novice users.

In summary, our qualitative observations provide evidence that participants did learn skills in the game that they could apply in the full Tinkercad application. This may be reflected in a decreased mean task time for the transfer tasks, but additional timed studies are warranted to confirm this.

DISCUSSION

Overall, our study findings indicate that less than 30 minutes of playing a familiar game with cross-application bridge enhancements can enable users to learn tools and skills they can successfully apply in an unfamiliar application and domain. Moreover, the enhancements achieved this without decreasing enjoyment of the game for players, or significantly increasing their cognitive load while playing.

In this section, we consider the question of how cross-application bridges could be built for other applications. To this end, we present a set of strategies for applying this approach that emerged out of this project. We also consider next steps for extending the research presented in this paper.

Strategies for Designing Cross-Application Bridges

Developing Blocks-to-CAD revealed several insights into how to design cross-application bridges. Based on our experience, one of the most important design considerations is how the conceptual models of the known and target applications differ. Six areas stand out as particularly important:

Application Domain. The known and target applications may exist in different domains. An application's domain defines the *purpose* of the application from the user's perspective, and thus can be useful for framing motivated learning scenarios. For example, Blocks-to-CAD uses the common theme of 'building' to frame new capabilities, and to motivate and advance the progression through the unlockable features.

Conceptual Model Differences. Understanding the differences between the conceptual models of the known and target applications is important for developing a set of steps that illustrate the conceptual model of the target application. It is also important to understand that users will have a strong mental model for the known application, and care must be taken when the target application's model contradicts it or differs significantly.

Data Representation. A particularly important difference between applications is how each represents data (e.g., a matrix of voxels versus a collection of 3D solids). To bridge differences in data representation, commonalities or mappings between the representations can be exploited. Depending on how extensively they differ, an artificial mapping may need to be developed to relate the two. In Blocks-to-CAD, we created a mapping between 3D solids and blocks in which the solids are converted to blocks when the player returns to the first-person mode, but the 3D solid representation is maintained in parallel and can be returned to in the third-person mode.

Tools and Capabilities. The known and target applications may provide different sets of tools and capabilities for modifying data, or tools with similar names that act differently in the two applications (e.g., the ability to remove blocks in Minecraft, and the command for deleting objects in Tinkercad). Unlike differences in application domain, or cross-cutting concerns such as data representation, tools are easily thought of individually. This makes them a good unit for introducing new capabilities to the user. In Blocks-to-CAD, we took advantage of this and used a series of tools to introduce new functionality. We also used the invocation of the tree-stamp and shapes tools as a mechanism for entering the third-person mode, in which the interaction model of Tinkercad is active. We believe this works well because it gives the user explicit control over switching the interaction model, and because it is not uncommon for tools in applications to impose a different interaction model for the duration that the tool is being used.

Interaction Techniques. The interaction techniques for applying tools and commands, and otherwise interacting with the two applications, may differ as well. In Blocks-to-CAD we had to contend with an extreme example – a 'locked' mouse cursor for the game, and an unlocked cursor for 3D modeling. To bridge this difference, we broke the transition into multiple steps – the tree-stamp tool releases the pointer lock for the duration that the tool is used, then re-locks it when the user returns to first-person mode. Next, the 3D navigation widgets introduce a new way to navigate the scene. Over time, participants spend more and more of their time working using the mouse cursor in the unlocked state.

Interface Conventions and Aesthetics. Finally, applications may differ in how action possibilities are communicated (i.e., affordances), how they provide feedback or feedforward to the user, and their overall aesthetics. We bridged these differences by tying conventions and visual style to the capabilities of the application being introduced (i.e., the game features and 3D modeling features each match the visual style of their respective applications). An advantage of this approach is that the conventions provide cues as to what conceptual model each tool will operate in. An alternate approach would be to gradually alter the visual style of the application as a whole – an approach that has been used in previous work as a means to reveal keyboard shortcuts [16].

Summarizing the above points, we recommend the following strategies for designing cross-application bridges:

- *Use a common theme or purpose to bridge application domains, and to tie together activities in the two applications.*
- *Carefully consider how the conceptual models of the known and target application differ, to identify areas of potential confusion, and to plan a series of steps between models.*
- *Use commonalities in data representation, or a mapping between representations, to motivate and enable users to discover key differences between the two conceptual models.*
- *Maintain parallel data representations, and use switching to demarcate changes in the interaction model.*
- *Use tools and commands as a unit for introducing new functionality or changes in the interaction model.*
- *Tie interface conventions to tools or data representations.*
- *Use intermediate interaction models to incrementally bridge larger differences in interaction techniques or modalities.*
- *Gradually transition toward the visual style of the target application.*

Generalizing to Other Applications

The strategies above could be immediately applied to develop a bridge from simple 3D solid modeling in Tinkercad to more complex *parametric* 3D modeling – a paradigm used in sophisticated CAD software that starts with creating 2D "sketches" of geometrically-constrained points, lines, and shapes, which are then transformed into 3D geometry. As a first step, an unlockable "2D-sketch" tool could be added to Tinkercad for drawing geometric shapes on the workplane, with simple presets for how these would be converted to 3D solids. Next, successive classes of geometric constraints from the target application could be unlocked and added to the 2D-sketch mode. The progression through unlockable functions could be motivated by the key advantages of parametric modeling (e.g., the ability to create models that are easily customizable, being defined by a few user-facing parameter values and a set of geometric constraints), and the 2D-sketch mode's visual style could match that of the target application (e.g., DSS SolidWorks, or Autodesk Fusion 360).

As another example, these design strategies could be applied to create a bridge from a Scratch-style "block" programming interface to a more traditional text-based programming IDE. Similar to the design of our prototype system, a two-way mapping between data representations could be maintained for the visual- and text-based code, and features could be gradually unlocked to reveal the advantages of text-based programming (such as the ability to quickly modify code without dragging visual blocks).

Beyond the examples above, we believe that opportunities arise for creating cross-application bridges whenever a significant number of users of one application may be interested in learning another, and the two applications have sufficient overlap in skills that motivated learning scenarios can be developed.

Toward a Toolkit for Cross-Application Bridges

Our design strategies provide high-level guidance for developing cross-application bridges, but they do not directly address the many lower-level design and technical challenges that arise when trying to bridge two applications. Thus, a key area for future work is to develop more specific approaches for user-skill modeling, interface adaptation, and creating motivated learning scenarios, as well as technical approaches for combining the interfaces and features of different applications. Ideally, work on these research problems could be collected in a toolkit for applying the cross-application bridges approach to new applications.

Limitations and Future Work

This work has several limitations that should be addressed in future research. First, our study was conducted with only 12 participants from a narrow age range (children ages 10-14). While this was sufficient to provide initial insights into our prototype system, and children are an ecologically-valid user group for the applications we were working with, it would be valuable to study a larger and more diverse group of users (e.g., adults or professional users).

Second, our evaluation was focused on understanding whether the cross-application bridges approach had learning value, and how it might impact learner motivation. Future studies are needed to investigate how the approach compares to standard tutorials, in terms of impact on learner motivation and depth of understanding. It would also be valuable to evaluate specific components of our system, such as the heuristics for determining when new features are unlocked.

Third, Blocks-to-CAD could be extended to include other fundamental 3D modeling concepts, such as additive/subtractive Boolean operations, or understanding what makes a model 3D printable or not.

Finally, working with children can be unpredictable, and the experimental setting may have introduced expectations that are different from a real-world setting.

CONCLUSION

This work has demonstrated that a cross-application bridge can effectively teach skills that transfer to a target application, without hurting enjoyment or increasing cognitive load. The design process for our prototype also provided deeper insights into the cross-application bridges approach, including design strategies for applying this approach to other applications. Overall, we see this work as a first step toward a future in which users seamlessly transition from the games they play in their youth, to the ever more sophisticated software applications demanded by their changing interests, careers, and creative endeavors.

REFERENCES

1. Ron Baecker. 2002. Showing instead of telling. In *Proceedings of the 20th annual international conference on Computer documentation* (SIGDOC '02), 10–16. https://doi.org/10.1145/584955.584957
2. M Bannert. 2000. The effects of training wheels and self-learning materials in software training. *Journal of Computer Assisted Learning* 16, 4: 336–346. https://doi.org/10.1046/j.1365-2729.2000.00146.x
3. Amy Bruckman. 1999. Can Educational be Fun? In *Game Developers Conference*, 5 pages.
4. Jerome S. Bruner. 1961. The act of discovery. *Harvard Educational Review* 31: 21–32.
5. John M. Carroll. 1990. *The Nurnberg funnel: designing minimalist instruction for practical computer skill*. MIT Press.
6. John M Carroll and Caroline Carrithers. 1984. Training wheels in a user interface. *Commun. ACM* 27, 8: 800–806. http://doi.acm.org/10.1145/358198.358218
7. John M. Carroll and Mary Beth Rosson. 1987. Paradox of the active user. In *Interfacing Thought: Cognitive Aspects of Human-Computer Interaction*. MIT Press, 80–111. Retrieved March 5, 2012 from http://dl.acm.org/citation.cfm?id=28446.28451
8. Richard Catrambone and John M. Carroll. 1987. Learning a Word Processing System with Training Wheels and Guided Exploration. In *Proceedings of the SIGCHI/GI Conference on Human Factors in Computing Systems and Graphics Interface* (CHI '87), 169–174. https://doi.org/10.1145/29933.275625
9. Mihaly Csikszentmihalyi. 2008. *Flow: The Psychology of Optimal Experience*. Harper Perennial Modern Classics.
10. Sebastian Deterding, Dan Dixon, Rilla Khaled, and Lennart Nacke. 2011. From Game Design Elements to Gamefulness: Defining "Gamification." In *Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments* (MindTrek '11), 9–15. https://doi.org/10.1145/2181037.2181040
11. Tao Dong, Mira Dontcheva, Diana Joseph, Karrie Karahalios, Mark Newman, and Mark Ackerman. 2012. Discovery-based Games for Learning Software. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '12), 2083–2086. https://doi.org/10.1145/2207676.2208358
12. Jennifer Fernquist, Tovi Grossman, and George Fitzmaurice. 2011. Sketch-sketch revolution: an engaging tutorial system for guided sketching and application learning. In *Proceedings of the 24th annual ACM symposium on User interface software and technology* (UIST '11), 373–382. https://doi.org/10.1145/2047196.2047245
13. Leah Findlater and Joanna McGrenere. 2007. Evaluating Reduced-functionality Interfaces According to Feature Findability and Awareness. In *Proceedings of the 11th IFIP TC 13 International Conference on Human-computer Interaction* (INTERACT'07), 592–605. Retrieved March 29, 2017 from http://dl.acm.org/citation.cfm?id=1776994.1777071
14. Adam Fourney, Ben Lafreniere, Parmit K. Chilana, and Michael Terry. 2014. InterTwine: Creating interapplication information scent to support coordinated use of software. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology* (UIST '14), 10 pages.
15. Wai-Tat Fu and Wayne D. Gray. 2004. Resolving the paradox of the active user: stable suboptimal performance in interactive tasks. *Cognitive Science* 28, 6: 901–935. https://doi.org/10.1016/j.cogsci.2004.03.005
16. Emmanouil Giannisakis, Gilles Bailly, Sylvain Malacria, and Fanny Chevalier. 2017. IconHK: Using Toolbar Button Icons to Communicate Keyboard Shortcuts. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 10 pages.
17. Tovi Grossman and George Fitzmaurice. 2010. ToolClips: An investigation of contextual video assistance for functionality understanding. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '10), 1515–1524. https://doi.org/10.1145/1753326.1753552
18. Tovi Grossman, George Fitzmaurice, and Ramtin Attar. 2009. A survey of software learnability: metrics, methodologies and guidelines. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '09), 649–658. https://doi.org/10.1145/1518701.1518803
19. SG Hart and LE Stavenland. 1988. Development of NASA-TLX (Task Load Index): Results of empirical and theoretical research. In *Human Mental Workload*, PA Hancock and N Meshkati (eds.). Elsevier, 139–183. Retrieved February 5, 2013 from http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20000004342_1999205624.pdf
20. Nathaniel Hudson, Benjamin Lafreniere, Parmit K. Chilana, and Tovi Grossman. 2018. Investigating how online help and learning resources support children's use of 3D design software. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '18), 10 pages.
21. Caitlin Kelleher and Randy Pausch. 2005. Stencils-based tutorials: Design and evaluation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '05), 541–550. https://doi.org/10.1145/1054972.1055047
22. Azam Khan, Igor Mordatch, George Fitzmaurice, Justin Matejka, and Gordon Kurtenbach. 2008. ViewCube: A 3D Orientation Indicator and Controller. In *Proceedings of the 2008 Symposium on Interactive*

3D Graphics and Games (I3D ’08), 17–25. https://doi.org/10.1145/1342250.1342253

23. Rock Leung, Leah Findlater, Joanna McGrenere, Peter Graf, and Justine Yang. 2010. Multi-Layered Interfaces to Improve Older Adults’ Initial Learnability of Mobile Applications. *ACM Trans. Access. Comput.* 3, 1: 1:1–1:30. https://doi.org/10.1145/1838562.1838563
24. D. Leutner. 2000. Double-fading support — a training approach to complex software systems. *Journal of Computer Assisted Learning* 16, 4: 347–357. https://doi.org/10.1046/j.1365-2729.2000.00147.x
25. Wei Li, Tovi Grossman, and George Fitzmaurice. 2012. GamiCAD: A Gamified Tutorial System for First Time Autocad Users. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology* (UIST ’12), 103–112. https://doi.org/10.1145/2380116.2380131
26. Richard E. Mayer. 2004. Should there be a three-strikes rule against pure discovery learning? The case for guided methods of instruction. *The American Psychologist* 59, 1: 14–19. https://doi.org/10.1037/0003-066X.59.1.14
27. Seymour Papert. 1980. *Mindstorms: Children, Computers, and Powerful Ideas*. Basic Books, Inc., New York, NY, USA.
28. Seymour Papert and Idit Harel. 1991. Situating Constructionism. In *Constructionism*. Ablex Publishing Coroporation.
29. Jean Piaget. 1971. *Science of Education and the Psychology of the Child*. Penguin Books.
30. Suporn Pongnumkul, Mira Dontcheva, Wilmot Li, Jue Wang, Lubomir Bourdev, Shai Avidan, and Michael F. Cohen. 2011. Pause-and-play: automatically linking screencast video tutorials with applications. In *Proceedings of the 24th annual ACM symposium on User interface software and technology* (UIST ’11), 135–144. https://doi.org/10.1145/2047196.2047213
31. Vidya Ramesh, Charlie Hsu, Maneesh Agrawala, and Björn Hartmann. 2011. ShowMeHow: translating user interface instructions between applications. In *Proceedings of the 24th annual ACM symposium on User interface software and technology* (UIST ’11), 127–134. https://doi.org/10.1145/2047196.2047212
32. L. B. Resnick. 1987. Learning in school and out. *Educational Researcher* 16, 9: 13–20.
33. Marc Rettig. 1991. Nobody reads documentation. *Commun. ACM* 34, 7: 19–24. https://doi.org/10.1145/105783.105788
34. John Rieman. 1996. A field study of exploratory learning strategies. *ACM Transactions on Computer-Human Interaction (TOCHI)* 3, 3: 189–218.
35. Joey Scarr, Andy Cockburn, Carl Gutwin, and Philip Quinn. 2011. Dips and Ceilings: Understanding and Supporting Transitions to Expertise in User Interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI ’11), 2741–2750. https://doi.org/10.1145/1978942.1979348
36. Ben Shneiderman. 2003. Promoting Universal Usability with Multi-layer Interface Design. In *Proceedings of the 2003 Conference on Universal Usability* (CUU ’03), 1–8. https://doi.org/10.1145/957205.957206
37. Penelope Sweetser and Peta Wyeth. 2005. GameFlow: A Model for Evaluating Player Enjoyment in Games. *Comput. Entertain.* 3, 3: 3–3. https://doi.org/10.1145/1077246.1077253