Editors: Gitta Domik
and Scott Owen

Beyond *Minecraft*

Facilitating Computational Thinking through Modeling and Programming in 3D

Alexander Repenning, David C. Webb, Catharine Brand, Fred Gluck, Ryan Grover, Susan Miller, Hilarie Nickerson, and Muyang Song
University of Colorado Boulder

The popularity with children of 3D design environments such as *Minecraft* has revealed previously untapped interests in modeling worlds, regardless of whether they're real or fantasy. As processors and graphics engines have improved, so have opportunities to consider how 3D programming environments might engage a significant proportion of the next generation of students in computer programming. We believe that when students have access to programming environments that can model interesting social, natural, and imagined phenomena, they'll more likely see programming as an essential skill for rendering their worlds.

Programming in three dimensions is engaging to young people who have regular experiences with 3D virtual worlds. Their experiences with high-quality 3D media—in movies, television, and gaming—have raised their expectations for computer-based design experiences. It's also important to consider design as an entrée to computer programming because representing agent behavior in 3D is an activity in which design and programming are seamless. When children focus primarily on modeling the world and related agent interactions, they find that programming is just part of the design process rather than an arbitrary skill set requiring the completion of a class.

As part of a large-scale study of computer science education based on the Scalable Game Design curriculum, we've introduced more than 12,000 students and more than 200 teachers to 2D and 3D end-user programming environments to create games and STEM (science, technology, engineering, and mathematics) simulations. AgentSheets, our 2D environment, has been used for years in computer science education and computational-science simulations. AgentCubes, a much more recent 3D environment, can turn AgentSheets projects into 3D projects through a process we call incremental 3D.[1]

On the surface, 3D sounds much more appealing than 2D, but what are the benefits of creating and using 3D? Here, we discuss our experiences with the differences between 2D and 3D as they relate to three concepts connecting computer graphics to computer science education: ownership, spatial thinking, and syntonicity.

Ownership

We've found ownership, and the related notion of creativity, important to motivation. Motivation, in turn, is a key to computer science education. Students who believe programming is difficult and boring won't likely engage in computing careers.

In spite of its simplicity, AgentSheets' 2D depiction editor was a key factor in getting students excited about computing by first having them draw objects they can use as characters in their worlds. Although middle-school students have developed higher expectations for computer graphics by playing sophistically rendered 3D games such as *Halo* 3, to our surprise they still enjoyed AgentSheets' simple 2D depictions. As one student put it, "I like it because I made it." More generally, we found such ownership essential to broaden student participation.[2]

The big appeal of a simple 2D depiction editor is that it has a low threshold for engaging in creative endeavors. Typically, the same thing can't be said about 3D creativity, which is often perceived as far more arduous. Tools to create 3D objects from scratch, such as Maya 3D or Blender, have intricate interfaces with steep learning curves and are well suited for professional 3D modelers but much less so for end-user designers. Most educational

 Published by the IEEE Computer Society

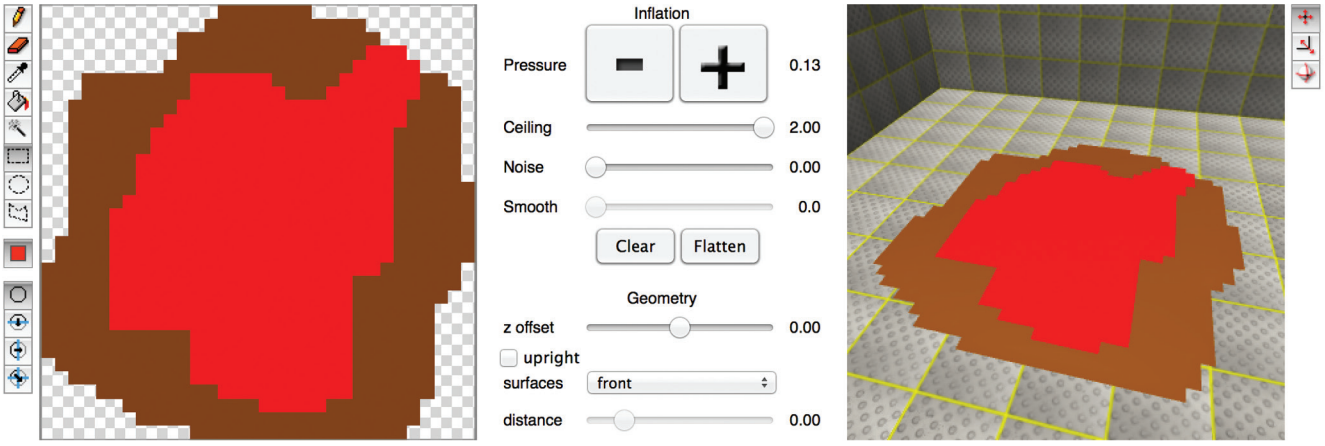


Figure 1. The Inflatable Icons editor. A user drew a top-down view of a volcano as an irregular brown shape with red lava inside (see the left image). After placing the volcano in the 3D editing environment (see the right image), the user applied inflation and other operations to create the 3D version in Figure 2.

3D programing environments minimize 3D modeling's obscurities by offering limited 3D mechanisms. With Alice, for instance, most users don't create their own 3D models but select 3D objects from a palette. In *Minecraft*, users assemble objects consisting of large numbers of boxes.

In contrast, we aimed to preserve the rapid-sketching spirit of 2D icon editing. So, we developed the Inflatable Icons approach. To create simple 3D shapes incrementally from scratch, users draw a 2D image and inflate it into a 3D image (see Figures 1 and 2). This paint-then-model approach differs radically from most other model-then-paint end-user 3D-modeling approaches.

In AgentCubes, with little more than a minute of instruction, students can build 3D models (see Figure 3). Of course, these shapes are aimed not at Pixar animators but at casual 3D modelers. Teacher and student feedback confirms that creating these 3D shapes is highly motivational. In some cases, teachers "complained" that students were so excited to create 3D shapes that they spent too much time with 3D modeling.

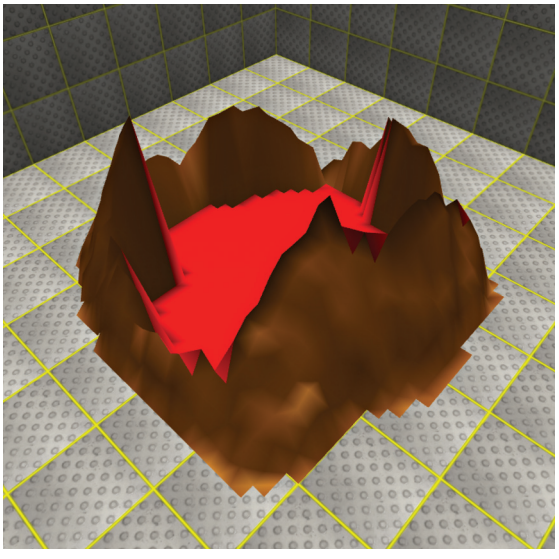


Figure 2. With Inflatable Icons, the user inflated the shape of the volcano in Figure 1, added noise to simulate a rocky surface, selected the lava, removed the noise from the lava, and pushed down the ceiling to flatten the lava's surface.

Spatial Thinking

If the main objective is for students to learn about computational thinking, educators must carefully consider the benefits and costs of employing 3D technology. Inflatable Icons can make creation of 3D models accessible even to young children, but creating a 3D game or simulation involves more than just creating individual shapes. Compared to 2D authoring, assembling 3D shapes in 3D worlds places heavier demands on student reasoning. Students must be able to understand how to control cameras, how to think in three dimensions by stacking objects to build composite structures, or even how to use layers to create more sophisticated worlds.

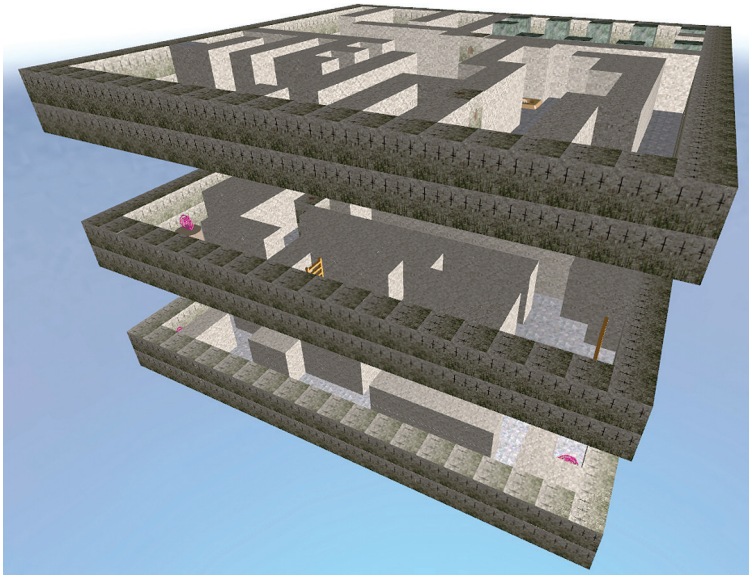
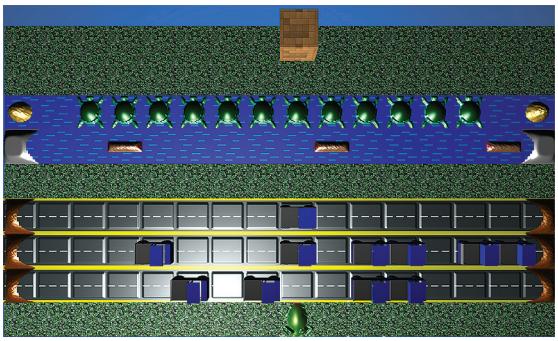
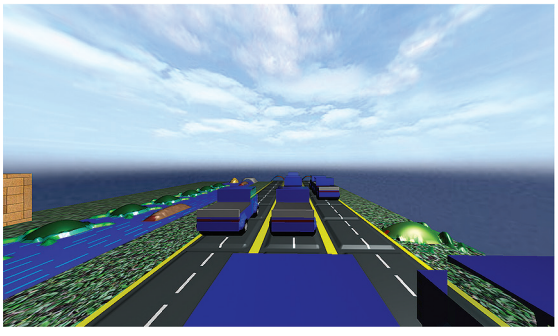


Figure 3. Using AgentCubes, a middle-school student created and programmed a complex multilayered game.



(a)

(b)



(c)

Figure 4. In a syntonic perspective, the user assumes the viewpoint of an object in the virtual world. (a) A (nonsyntonic) bird's-eye perspective. (b) A syntonic frog perspective. (c) A syntonic car perspective.

However, we found little evidence that students struggle with 3D composition and camera control. It's certainly possible to flip worlds around and to get the camera in an awkward position in ways that wouldn't have occurred in a 2D world. However, this rarely happened. In many cases, these apparent 3D-thinking skills could have resulted from students having played advanced 3D games or 3D construction-oriented games such as *Minecraft*.

Sometimes, 3D design is actually simpler than 2D design. One such case is debugging using AgentCubes. For instance, when creating a *Frogger*-like game, students generate trucks that move on a road and could collide with the frog. They frequently forget to program the trucks to disappear at the end of the road, often resulting in trucks piling on top of each other.

In the 2D environment, this stacking of trucks goes unnoticed; from a bird's-eye perspective, there's only one truck left at the end of the road. When the game runs for a longer period, thousands of trucks might be stacked at the end of the road. This results in the game slowing significantly and possibly crashing.

In the 3D version, just two stacked trucks will get the student's attention, prompting the realization that the rule to remove the trucks must be missing. Design in 3D requires students to think of their worlds in ways that integrate troubleshooting and spatial visualization, which are different aspects of computational thinking.

Syntonicity

The final concept comes from the psychology of programming. Early on, Seymour Papert speculated that humans' ability to project themselves onto objects—essentially becoming the objects—would help them overcome otherwise difficult programming challenges.[3] The ability to comprehend and even predict behaviors is called *body syntonicity*.

Papert and his colleagues developed the famous Logo robotic turtle, which children could program through simple instructions that made it move forward and turn. Later, Papert and his colleagues replaced the physical turtle with a simulated version. The Logo programming language eventually incorporated syntonicity, which added a psychological aspect to the otherwise technical framework of computation. Through syntonicity, children could explore interactions that were difficult to understand by formulating first-person questions, such as "If I'm this turtle and I turned 90 degrees left, what would I see?"

Ample evidence exists that programmers naturally employ syntonic notions. For instance, programmers say something like, "How can the compiler do this to me?" when they're actually referring to the program they wrote. However, it's less clear what syntonicity's educational benefits are.

To explore this, we created a syntonic version of the Visual AgentTalk programming language, which is part of AgentSheets and AgentCubes. This syntonic version originally had elaborate commands using language suggesting syntonic interpretations. For instance, instead of a "move (direction)" action, there was an "I move to the left" action. That is, instead of the programmer just thinking that some object is moving, the language suggested that it's "I" who is moving to the left.

Although children with no programming background liked the language, more experienced users rejected the general increase in verbosity. So, we

compromised. We kept the short command names for the actual visual-programming language. In addition, we offered mechanisms such as explanation buttons and tool tips to provide syntonic versions of the commands, in case users needed help interpreting the commands' functions.

With 3D worlds, you can go a significant step further. You can map the psychological perspective of becoming an object onto a user interface that lets programmers select any object in a complex world and put that object into a first-person perspective. If the object moves or turns, the camera adjusts accordingly.

In this way, users can experience any game or simulation in AgentCubes from the perspective of any of its objects (see Figure 4). Many students have employed and enjoyed this feature, but we have no evidence whether students are actually assuming syntonic perspectives. We also don't know whether this feature really helps overcome challenges more easily than would be possible in 2D.

However, we've connected this syntonic perspective explicitly with our programming environment through conversational programming.[4] This approach has proven useful for semantic-programming support. When a user selects an object in AgentCubes, the programing environment immediately shows that object's code. It also runs that code one step into the future and annotates the program to show what the object in its current state will do.

As you can see, 3D computer graphics are highly relevant to computer science education for reasons beyond motivation. In our experience, to receive 3D graphics' full benefits for computer science education, novel programming environments should support authoring approaches that incorporate 3D rapid sketching and should explore ways to connect 3D navigation with semantic support for programming.

AgentCubes Online, which employs HTML5 and WebGL (Web Graphics Library), is the first browser-based 3D-modeling and visual-programming tool of its kind. Over 250,000 students worldwide have used it as one of the Hour of Code tutorials; you can play with it at http://hourofcode.com/ac.

Acknowledgments

The US National Science Foundation supported this research under grants CNS-1138526 and DRL-1312129. Any opinions, findings, and conclusions or recommendations expressed in this article are the authors' and don't necessarily reflect the NSF's views.

References

1. A. Ioannidou, A. Repenning, and D.C. Webb, "AgentCubes: Incremental 3D End-User Development," *J. Visual Languages and Computing*, vol. 20, no. 4, 2009, pp. 236–251.
2. D.C. Webb, A. Repenning, and K.H. Koh, "Toward an Emergent Theory of Broadening Participation in Computer Science Education," *Proc. 43rd ACM Tech. Symp. Computer Science Education*, 2012, pp. 173–178.
3. S. Papert, *Mindstorms: Children, Computers, and Powerful Ideas*, 2nd ed., Basic Books. 1993.
4. A. Repenning, "Conversational Programming: Exploring Interactive Program Analysis," *Proc. 2013 ACM Int'l Symp. New Ideas, New Paradigms, and Reflections on Programming & Software*, 2013, pp. 63–74.

Alexander Repenning *is a computer science professor at the University of Colorado Boulder. Contact him at alexander.repenning@colorado.edu.*

David C. Webb *is an associate professor of curriculum and instruction at the University of Colorado Boulder. Contact him at dcwebb@colorado.edu.*

Catharine Brand *is a community volunteer for the Scalable Game Design project. Contact her at catharine.brand@gmail.com.*

Fred Gluck *is an instructor at the University of Colorado Boulder Science Discovery program. Contact him at fred.gluck@comcast.net.*

Ryan Grover *is a PhD candidate in curriculum and instruction—math at the University of Colorado Boulder. Contact him at ryan.grover@colorado.edu.*

Susan Miller *is a PhD student in curriculum and instruction—math at the University of Colorado Boulder. Contact her at susan.miller@colorado.edu.*

Hilarie Nickerson *is a PhD student in computer science and cognitive science at the University of Colorado Boulder. Contact her at hnickerson@colorado.edu.*

Muyang Song *is a master's student in computer science at the University of Colorado Boulder. Contact him at muyang.song@colorado.edu.*

Contact department editors Gitta Domik at domik@uni-paderborn.de and Scott Owen at sowen@gsu.edu.