AgentCubes: Raising the Ceiling of End-User Development in Education through Incremental 3D

Alexander Repenning[1,2] & Andri Ioannidou[2]
[1]University of Colorado, Computer Science Department, Campus Box 430
[2]AgentSheets Inc, 6560 Gunpark Dr.
Boulder, Colorado 80301
ralex@cs.colorado.edu, andri@agentsheets.com

Abstract

Now that we have end-user programming environments capable of empowering kids with no programming background to build games in a matter of hours, a new quest for raising the ceiling of end-user development is emerging. Environments not only focusing on programming, but also including rich media such as 3D, could work as compelling tools for introducing information technology at the K-12 level, addressing even the problem of dwindling numbers of computer science student enrollments at universities. The new challenge is raising the ceiling without raising the threshold. Based on our experience with AgentSheets, which has been used worldwide for computational science and game design applications, we created a new authoring tool called AgentCubes. This article discusses the notion of Incremental 3D as a design approach for media-rich end-user development with low threshold and high ceiling in education.

1. End-User Programming in Education

One of the most important goals of end-user programming in education has been to employ the notion of programming as means of interactive expression. This kind of literacy [1] could allow kids to express complex ideas through creating simulations and models. However, programming turned out to be a daunting challenge. Early text-based programming languages, such as Logo, had limited success because of insufficient scaffolding, making it all too simple to create programs that did not work. It became clear that the notion of programming would have to be reconceptualized for end-users. To make this vision work, it was instrumental to devise new computational ideas that would lower the threshold of programming in order to make it work in educational contexts. Hence, end-user programming in education was born. This quest for new programming paradigms explored various mechanisms to scaffold the programming process.

1.1 Trapped by Affordances

An effect that we have called "Trapped by Affordances" [2] has turned out to be common to many end-user programming approaches aiming for a low programming threshold. An affordance is a property of an object that strongly suggests how it could be used. The idea of being trapped by an affordance means that, while initially the affordance makes a task extraordinarily simple, later that same affordance gets in the way. It either fails to help with more complex problems, or even worse, it actually makes a task potentially harder by forcing a user to think about a problem in a way that is not compatible with the problem. Affordances can trap users at the syntactic as well as the semantic level, hence preventing the programming ceiling from being raised.

Syntactic level: Program syntax was commonly perceived to be one of the largest end-user programming challenges. Traditional programming languages included at the time, and still do, syntactic intricacies that are part of the language mostly to allow machine interpretation as opposed to human interpretation. One little semicolon at the wrong place and the program would no longer do what the programmer intended it to do. Programming languages such as the early BLOX Pascal [3] and the more recent LEGO Mindstorms strived to eliminate syntactic problems altogether by conceptualizing programming languages as puzzle pieces. These pieces are shaped in a way so that only syntactically correct programs can be written. The results of numerous variants of this visual programming idea are mixed at best. Many users and researches exploring these programming languages concluded that while they were a great starting point, they did not scale well. Complex programs quickly became unwieldy [4]. To a large degree, the very affordance of the puzzle piece idea that initially helped in building simple programs would trap users and for more complex programs would result in convoluted arrangements dictated by the visual metaphor.

Semantic level: Graphical rewrite rules for programming agents, initially introduced by AgentSheets [5] and later adopted by KidSim[1] [6], had users trapped by semantic affordances. Rewrite rules have employed notions of programming by example. For instance, an agent could be instructed to move to the right simply by using the mouse and dragging it to the right while telling the system to observe, record and potentially generalize the user actions and turn them into rewrite rules. Early usability testing indicated that even young kids quickly, and with almost no instructions, were able to use this programming paradigm. However, similar to the syntactic trap, graphical rewrite rules clearly had a strong affordance; one with even more severe limitations. As soon as users had to create more complex behaviors that either extended or were completely orthogonal to the graphical rewrite rule paradigm, the affordance turned into a trap. One such trap was the need to create a very large set of rules for generalized behavior such as a car agent trying to follow a road system including intersections [7]. Another semantic trap was the lack of procedural abstraction. Rewrite rules did not have a means to name a behavior and be able to invoke the behavior through actions.

1.2 Tactile Programming: AgentSheets

After creating the graphical rewrite rule version of AgentSheets, we realized that because of the Trapped By Affordances effect we would not be able to have users build sophisticated applications. At the same time we had a developer version of AgentSheets that had to be programmed in AgenTalk (an agent-based version of Common Lisp). This allowed a number of power users to make sophisticated applications including games, scientific simulations, and even new authoring environments [5, 8] that would have never been possible with the graphical rewrite rules. Analyzing these Lisp-based applications and synthesizing ideas found in spreadsheets, rule-based programming and bottom up programming we devised the idea of Tactile Programming [9-11] exhibiting the following principles:

- ***Composition***: In AgentSheets' Visual AgenTalk language components such as conditions and actions are elevated to the level of highly manipulatable objects. Users program by composing these objects via drag and drop operations into complete programs. Interactive feedback guides users to create syntactically correct programs.
- ***Comprehension***: Tactile programming languages enable programs to be composed incrementally. The ability to perceive the consequences of incremental programming supports an exploratory style of programming, where users are allowed to play with the language and explore its functionality. Perception by manipulation afforded by tactile programming allows end-users to efficiently examine functionality in a direct exploration fashion in the same spirit as bricolage in Logo [12, 13], but with more support mechanisms. Tactile programming with decomposable test units at different levels of granularity of the programming language (individual commands, rules, methods) provides easy debugging for end-user programmers who do not posses the skills of professional programmers in debugging.
- ***Decomposability***: Decomposable units of behavior in tactile programming enable both testing, as mentioned above, and sharing. We found rule-based languages especially decomposable. Individual conditions and actions can be taken much more easily out of context and tested compared to more traditional imperative programming approaches. Additionally, decomposability promotes sharing. Users are able to share simulation components locally or over a web-based repository [14, 15] making agents a form of currency in a community of simulation and game developers.

1.3 Empirical Evidence of Low-Threshold

Given that AgentSheets has been used for over a decade, perhaps just as important as theoretical perspectives are empirical results based on experiences with AgentSheets in game design and computational science applications. Over the years, the applications that users have produced have exceeded our own expectation. AgentSheets has been employed by a broad spectrum of users ranging from elementary school kids [16] simulating ecosystems to NASA scientists simulating experiments aboard the Discovery space shuttle [17]. AgentSheets has been used in different types of educational settings including elementary school science, introductions to information technology, middle school computer clubs, high school social studies curriculum and programming courses, science discovery after school programs, undergraduate game design courses, and graduate learning technology design courses.

[1] Later known as Stagecast Creator

Initially, we mostly focused on the end-user programming mechanisms themselves, assuming that this would be the most important aspect of enabling end-users. However, once we started using the AgentSheets tool in traditional educational settings, we quickly realized that the end-user programming process in educational settings needs to be scaffolded with new instructional approaches in order to work. In the Trails project (http://www.trails-project.org), a consortium involving multiple universities, we started to develop game design curriculum for undergraduates. After teaching game courses and workshops in the USA, Europe and Japan for a number of years we were able to create perhaps the world's shortest game design and development workshop called "Trails mini". In this kind of workshop we can teach kids – and adults – how to make a video game in about three hours.

2. AgentCubes

Our new goal was to make a new kind of game and simulation authoring tool that would be as simple to use as AgentSheets, but would also allow the creation of sophisticated 3D applications. AgentCubes inherits much from its parent AgentSheets [5, 10, 11, 18, 19]. A cube is a four-dimensional structure consisting of a three-dimensional <row, column, layer> indexed matrix containing stacks of agents. Cubes can be recursive. Because cubes are agents themselves a stack of agents may contain nested cubes.

Figure 1: An AgentCubes Soccer Game

Agents have shapes that can be simple textured tiles, spheres, boxes, inflatable icons [20] or imported 3D models. Additionally, agents have built-in properties controlling their color, transparency, orientation, and size.

A world contains at least one cube. The world in Figure 1 shows a soccer simulation based on a single layer cube representing a soccer field. A pen and eraser based tool interface allows users to quickly create large and complex arrangement of agents in a world.

Agents are programmed in VisualAgenTalk II. An extended VisualAgentTalk I [9-11] set of actions and conditions is used to create IF/THEN rules. Actions include basic movement, rotation control, message sending, color and transparency control, sound output, speech synthesis, spreadsheet-like formula evaluation, 3D surface plotting, and chat interface control. Conditions include scene parsing, timers, probability, keyboard/gamepad input, and web page screen scraping.

To enable procedural abstraction, rules can be grouped into methods that are named by the user. Methods are invoked in the behavior through message actions, or, indirectly through triggers invoked by events such as mouse clicks.

3. Incremental 3D

Based on their daily exposure to information technology including game consoles, children today have higher expectations in terms of media. They expect rich media including MP3 sounds, movies, 3D models and more. This poses an interesting design challenge. To some degree, these new media can be added to authoring tools incrementally. Over time AgentSheets got extended with advanced media features including spatial sound, speech synthesis, speech recognition, Macromedia Flash output, QuickTime movie play and 3D visualizations. But even with these extensions, we recognized the need to raise the ceiling of end-user development [21, 22] for a new level of game design and computational science applications that could only be addressed with a new conceptual framework enabling what we call *Incremental 3D*. This conceptual shift hinges on the transition from end-user programming to end-user development. That is, we no longer limit the scope of authoring to programming but, instead, include all aspects of development necessary to create 3D applications.

AgentCubes is an Incremental 3D end-user development tool to create 2D/3D games and simulations. The fundamental idea of Incremental 3D is that a user should be able to suspend important design decisions to the point in time of the design and development process when the decision really needs to be made. For instance, many game and simulation applications can start as simple 2D applications that may or may not be turned into 3D applications. Initially, the user should not have to worry about the precise look, size, orientation and locations of objects in 3D space or how objects need to be animated when they move. For instance, by utilizing grids, we

transition from the need to deal with Euclidian information (e.g., move my object 1.5 meters to the right), to topological information, (e.g., move my object right to the next space).

In the context of Incremental 3D, we intentionally use the term *end-user development* and not *end-user programming* to indicate the inclusion of non-programming related design activities. For instance, we believe that it is crucial to include incremental mechanisms to rapidly sketch 3D models that may start out as simple 2D sketches. Similar to fat pen approaches used in architectural drawing design, Incremental 3D includes a set of "rough and ready" tools enabling the designer to explore design variations with great speed and low commitment [23].

As a design process, Incremental 3D can address the low threshold, high ceiling [13] challenge. Many games and simulations can initially be conceptualized as applications that may have 3D manifestations, but at a logical level are essentially just 2D systems. If spatial relationships between objects are simplified through a strong spatial organization scheme such as a grid, then programming can be further simplified. From our AgentSheets experience we know that the grid significantly contributes towards lowering the programming threshold because it makes spatial relationships much more transparent.

Through the Incremental 3D design process, users move along well-defined stepping-stones from 2D to 3D applications. This process is raising the ceiling and keeping the threshold low by employing tools described in the next sections.

3.1 Incremental Animation

Animations in games and simulations serve multiple roles. The most trivial role animation can play is to make applications look nicer. However, much more important is the role of animation to communicate complex relationships between objects. We have devised a novel animation approach that can be employed incrementally.

Facilitating the perception of causality through animation. With his work on the perception of causality, Michotte [24] showed that humans perceive causality between objects depending on the exact timing of movement. At the time, his experiments were based on an elaborate mechanical apparatus allowing him to manipulate the animation of two seemingly interacting objects. His experiments showed that even small timing variations in the neighborhood of 100 milliseconds would have people come up with completely different explanations of the causal connection between objects. He reasoned that the causality inferred was directly perceived without the involvement of higher order cognitive processes. In our daily experience, perception of causality is grounded in a world containing objects adhering to physical processes. In the world of computers, this means that animation plays a much more important role that to make, say, games look nicer. It is a crucial spatio-temporal information channel that allows us to perceive and consequently to understand the world that is being simulated more effectively.

It is not hard to argue that animations play an important role in the simulation of physical systems. A simulation of a bouncing rubber ball would not be much of a simulation without actually showing a rubber ball exhibiting the physically grounded behavior. However, Michotte's work can be applied to non-physical systems as well. As Michotte points out, humans will not only perceive causality in the physical sense, but will even attribute motivational and emotional aspects to interacting objects. This kind of interpretation including descriptions such as launching, chasing, and following can also be used in non-physical contexts such as social science simulations.

To be able to achieve the desired effect in the Michottian sense, AgentCubes includes a number of mechanisms to enable and control animations. Users can adjust the time, the trajectory and acceleration of an animation. Using a simple slider, users can adjust the animation time anywhere between 0 (no animation) and a few seconds. We have been surprised how differently we perceive some of our classic simulations – AgentCubes is capable of importing AgentSheets projects – when animation is enabled. Especially simulations featuring large quantities of agents can sometimes be perceived qualitatively differently.

Separation of Logic and Animation. An important aspect of Incremental 3D is that the logic part of end-user programming and the animation part are kept separate. The logic part describes what the agent will do. For instance, in a Space Invaders game, the cursor controlled defender agent will move one grid space to the right. The user will simply use the Move <right> action to achieve this (Figure 2, left).

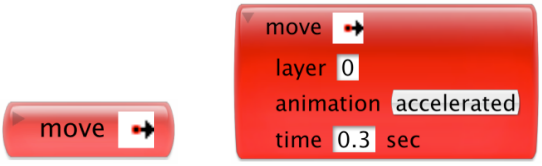

Figure 2: Separate logic from animation. Left: move right action; Right: disclosed version showing additional parameters relevant to animation.

Later in the development process, the user may want to add animation information. The user may want to use an accelerated animation in which the agent continuously accelerates and at the mid point starts to

decelerate until it comes to a complete stop. The time is takes to run this animation can either be controlled by the end-user via a slider or through a computed or fixed value (Figure 2, right).

Scene awareness. Animations quickly become complex for a user to operate if animations have no physical awareness of a scene. The move action hides an enormous amount of complexity because it includes automatic physical interpretations of a world. In 2D environments such as AgentSheets and KidSim, a move will simply remove an agent from one location in the grid and move it to a new location. It should not be any harder for a user to do this in 3D, but the system will have to fill in some blanks with respect to how a move should be interpreted in a three dimensional space. Consistent with the notion of stacks, an agent moving from one stack to another will automatically move on top of the new stack. The animation trajectory consisting of automatically generated x, y, z animation components will be computed to minimize the chance of object intersections. If an agent moves out of a stack but was not on top of the stack, then the stack will be compacted again. Consistent with gravity, all the agents above the agent moving out will drop down. Of course, some applications may not be consistent with stacks and gravity. In this case the user can use layers instead of stacks.

Without physical interpretation assisted by the notions of grids and stacks, a move would become tedious. For instance, in Alice [25] users would have to first write some kind of grid manager with gravity to spatially parse a scene. This would be used to either create a move based on three parallel x, y, z moves or to use the move_to method to move the object on top of the stack to be moved to.

Parallel Time-Jump. AgentCubes uses the novel Parallel Time-Jump animation approach to allow any number of agents to animate in parallel. Animating large number of agents is a hard problem. Imagine even a simple simulation in which agents are moving around randomly. Agents moving to the same stack in the same layer will have to pile up. This would not be a problem if animation could be handled sequentially. The first agent moves to the stack and then the second agent moves on top of that agent in the same stack. This will work nicely with a small number of agents, but the total time it takes to transition an agent world from one step to the next will be the product of the animation time and the number of agents. In some of our science applications, e.g., eColi bacterial in Microgravity [17] we had 10000 agents. Animating only 1000 agents with 0.3 seconds per animation would total in a seemingly never-ending 5-minute animation. In such a case, animation would take too long time to execute, unless done in parallel.

But if animations need to be done in parallel, how can we know where the agents are moving? We can only make this decision once all the agents got dispatched and have been moved to their final destinations. Otherwise, individual agents cannot start their animation trajectory because they do not know where they will end up. This appears to be a contradiction. The Parallel Time-Jump avoids this contradiction by moving forward and backward in time. Conceptually speaking, the Parallel Time-Jump will first dispatch, move and rotate all agents without animation. Then it leaps back in time and generates all the transitional animations from were the agents currently are to where they will be. This way, the 1000 agents will only take 0.3 seconds to be animated.

From the viewpoint of the user, the Parallel Time-Jump is completely transparent. The time-jumps are all done without updating the screen. Users will not have to worry about what currently *is* in the scene and what *will be* in the scene. They can simply run or step the simulation. The animation will appear to the user as if all the agents already know where they need to move. An example may help to illustrate this point. Say there are two crates and one wall agents (Figure 3a). Both crate agents want to move on top of the wall agent. Agents are dispatched in random order. Figure 3b shows the animation resulting when the right crate agent gets dispatched first. Animations are intrinsically hard to capture in paper media. The motion blur provides a very limited sense of the complex trajectory of the two crates. Note, for instance, that the left crate is significantly overshooting vertically to avoid "unnatural" intersection with the right crate as they both move towards their destination in parallel.

The essence of Parallel Time-Jump is that users can effectively employ the animation of a large number of objects without the need to track object locations and without the overhead of sequential animation. This is important in all kinds of 3D applications.

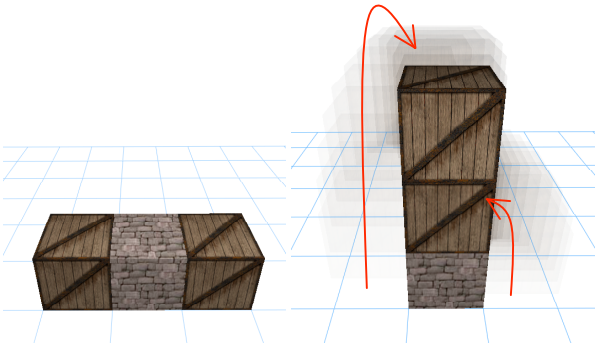

Figure 3: (a) two crate agents (left, right) both want to move on top of brick agent. (b) Right crate gets dispatched first but both crates know where they need to move to. Both crates move in parallel to their respective destinations. The animation makes the left crate overshoot vertically to avoid intersection.

3.2 Incremental 3D Model Development

A big question is where do 3D models come from? From our experience with AgentSheets, we know that allowing user to create their own 2D agent depictions in many cases serves an important role. The icons created may not be professional, but they are an important part of communicating personal ideas. Creating a good-looking 2D icon is by no means trivial, but creating a 3D model is quite often a daunting process. Sophisticated 3D modeling tools such as Maya have extremely steep learning curves. This is not compatible with the notion fat pen design approaches [9] or Incremental 3D. The following steps help users to develop 3D models incrementally.

Tiles. The first step towards a 3D application may be to create a project or import an existing 2D project. Icons are represented as textured flat tiles (Figure 4).

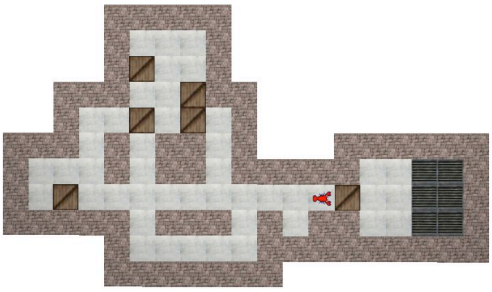

Figure 4: The 2D Sokoban game imported into AgentCubes.

The tile-based 3D world is not completely flat. Each tile has a minimal height. Stacks of agents are still interpreted vertically. When importing existing AgentSheets games, we sometimes found problems that were not visible in AgentSheets. For instance, in a game such as Space Invaders it would be simple to forget to write rules about deleting dropped bombs that missed their targets. This could result in thousands of agent piling up in a worksheet. Over time, this kind of agent "leak" could use up all the memory. Without a 3D interpretation there will be no visible clue if there is one or thousands of agents piled on top of each other. Agent leaks became immediately apparent when opening up worlds in 3D because they visibly render as tall stacks of agents.

Basic 3D Shapes. To give a world a real 3D look and feel, the user maps agents to shapes with real 3D extend. For instance, in Sokoban the shape representing a crate will be turned into a 3D box by changing its shape type from tile to box. The original crate icon will be used as a texture (Figure 6).

High Resolution Texture 3D Shapes. Most hand drawn 2D images are low resolution, e.g., 32 x 32 pixel, icons. Especially when zoomed in close, this kind of texture will look too grainy. The user can now provide higher resolution image textures. Instead of a 32 x 32 x 8 bit color pixel icon the user may want to use a 1024 x 1024 x 32 bit color texture including a 8 bit alpha channel. With this, our shapes begin to look considerably more sophisticated.

Inflatable Icons. Inflatable Icons[2], is a new technique that interactively extrudes 2D pixel-based images into polygon-based 3D models, adding a third dimension to a two dimensional image. The general idea is that through the use of a diffusion-based inflation process and with minimal input from users suitable 2D icon artwork can serve as input for an interactive 2D to 3D transformation process [22]. The user can control inflation pressure, symmetry, and can add noise.

Figure 5: Inflatable Icons turn simple 2D images into 3D models with thousands of polygons.

Inflatable Icons are not meant to replace dedicated tools for creating 3D models, but they do allow creating multi thousand-polygon 3D objects in just seconds or minutes.

Imported 3D models. A user can simply import 3D model files produced by third-party modeling tools or found in 3D model clipart collections.

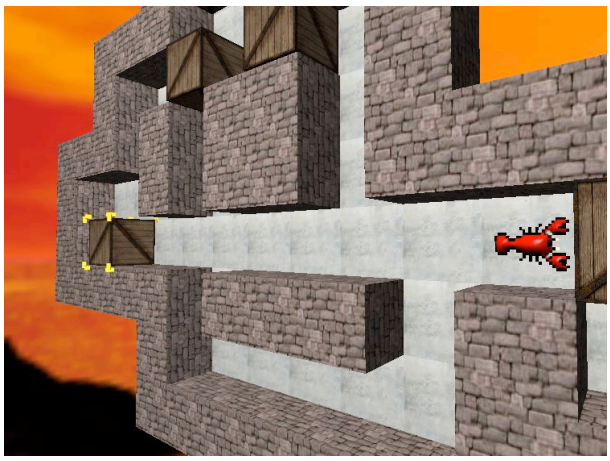

Figure 6: The 3D version of Sokoban turned bricks and crates into boxes and the lobster into an inflated icon.

[2] Patent pending

3.3 Incremental 3D Behavior Development

Programming behaviors in a 3D environment can be a daunting task especially for end-users. More involved than 2D, the 3D environment can present additional challenges to end-user programmers. 3D authoring tools should provide scaffolding of the behavior development process to make the transition from 2D to 3D more gradual. The following features help users to program 3D behaviors incrementally.

Flat Tiles have 3D behaviors. A 2D project based on flat tiles already includes 3D behavior. A number of agents with flat tile shapes piling up will result in a stack that the user can visibly discern as a 3D stack. At this level, the user does not have to provide any 3D programming. A 2D project may have been created from scratch or by importing an existing 2D project from AgentSheets. Creating a 2D world is much less challenging than starting directly in 3D.

3D Shapes have 3D behaviors. Once the user replaced flat tiles with 3D shapes (basic 3D shapes, Inflatable Icons, or imported 3D models) the world will exhibit more pronounced 3D behavior. At this stage, most of the imported objects require their behaviors not to be changed at all. We find, however, that many users do feel compelled, perhaps because of the increase in visual realism, to add and tweak animations.

Layer-Based Behavior. More sophisticated 3D applications tend to use layers to include behaviors that could not be captured with stacks. Say we wanted to create a 3D version of the game Cubes[3]. Instead of just using the concept of two-dimensional neighborhood one could easily generalize the game and include the third dimension. Stacks would not be very useful for this application. It should be possible for a cube agent to float in free space without gravity pulling it down. Layers allow this. In the 3D Cube game if the user clicks a cube agent, it will check all its neighbors in three dimensions. This requires minimal additional programming. For instance, when a cube is checking for red neighbors, it also needs to look across layers for agents of the same color, in addition to checking left, right, up and down in the same layer.

[3] In the Cubes game (http://www.sporecubes.com/), the player clicks groups of two or more of the same-color cubes to make them explode. Any cubes above the gap created will then fall down. If any gaps are created between columns, the cubes to the right of the gap will slide over to the left. The goal is to clear all cubes.

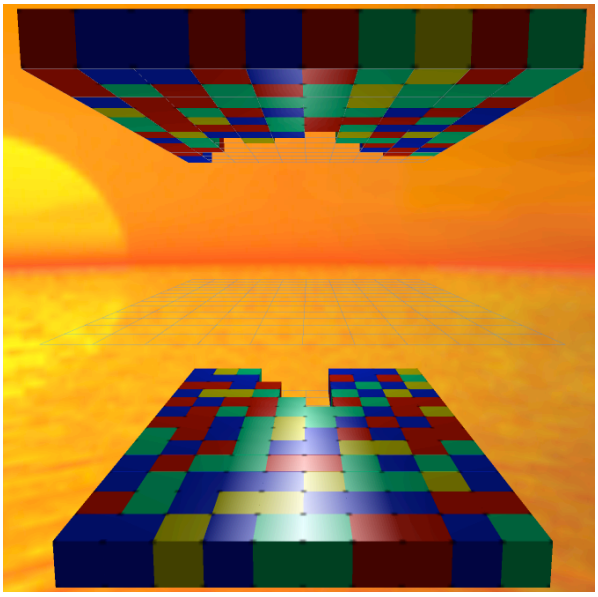

Figure 7: Extending the 2D game of Cubes to a 3D game only requires minor additions to the 2D behavior to account for layer information.

Conclusion

Incremental 3D is a powerful end-user development process to scaffold the design and implementation of 3D simulations and games. End-users start with 2D applications and gradually add 3D animations, models and behaviors. As a gradual process supported by well-defined stepping-stones, Incremental 3D raises the ceiling of end-user development. When combined with a low-threshold end-user programming authoring system such as AgentSheets, Incremental 3D results in low-threshold, high-ceiling end-user development system. Our incarnation of such as system is called AgentCubes, a system that allows end-users with no programming background to quickly create sophisticated 3D games and simulations.

Acknowledgements

This material is based in part upon work supported by the National Science Foundation under Grant Numbers No. 0205625 and DMI-0349663. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

1. diSessa A: Changing Minds: Computers, Learning, and Literacy. Cambridge, MA: The MIT Press, 2000.

2. Schneider K, Repenning A: Deceived by Ease of Use: Using Paradigmatic Applications to Build Visual Design. Proceedings of the 1995 Symposium on Designing Interactive Systems, Ann Arbor, MI, 1995.

3. Glinert EP: Out of flatland: Towards 3-d visual programming. IEEE 2nd Fall Joint Computer Conference, 1987.

4. Gage A, Murphy RR: Principles and Experiences in Using LEGOs to Teach Behavioral Robotics. 33rd ASEE/IEEE Frontiers in Education Conference, Boulder, CO, November 5-8, 2003, 2003.

5. Repenning A: Agentsheets: A Tool for Building Domain-Oriented Dynamic, Visual Environments [Department of Computer Science]. University of Colorado at Boulder; 1993.

6. Smith DC, Cypher A, Spohrer J: KidSim: Programming Agents Without a Programming Language. Communications of the ACM 1994; 37(7): 54-68.

7. Repenning A: Bending the Rules: Steps toward Semantically Enriched Graphical Rewrite Rules. Proceedings of Visual Languages, Darmstadt, Germany, 1995.

8. Gindling J, Ioannidou A, Loh J, Lokkebo O, Repenning A: LEGOsheets: A Rule-Based Programming, Simulation and Manipulation Environment for the LEGO Programmable Brick. Proceeding of Visual Languages, Darmstadt, Germany, 1995.

9. Repenning A, Ambach J: Tactile Programming: A Unified Manipulation Paradigm Supporting Program Comprehension, Composition and Sharing. Proceedings of the 1996 IEEE Symposium of Visual Languages, Boulder, CO, 1996.

10. Repenning A, Ioannidou A: Behavior Processors: Layers between End-Users and Java Virtual Machines. Proceedings of the 1997 IEEE Symposium of Visual Languages, Capri, Italy, 1997.

11. Repenning A, Ambach J: Visual AgenTalk: Anatomy of a Low Threshold, High Ceiling End User Programming Environment. submitted to Proceedings of UIST, 1996.

12. Papert S: The Children's Machine. New York: Basic Books, 1993.

13. Papert S: Mindstorms: Children, Computers and Powerful Ideas. New York: Basic Books, 1980.

14. Repenning A, Ambach J: The Agentsheets Behavior Exchange: Supporting Social Behavior Processing. CHI 97, Conference on Human Factors in Computing Systems, Extended Abstracts, Atlanta, Georgia, 1997.

15. Repenning A, Ioannidou A, Rausch M, Phillips J: Using Agents as a Currency of Exchange between End-Users. Proceedings of the WebNET 98 World Conference of the WW, Internet, and Intranet, Orlando, FL, 1998.

16. Ioannidou A, Rader C, Repenning A, Lewis C, Cherry G: Making Constructionism Work in the Classroom. International Journal of Computers for Mathematical Learning 2003; 8: 63-108.

17. Klaus DM: Microgravity and its Implication for Fermentation Technology. Trends in Biotechnology 1998; 16(9): 369-373.

18. Repenning A, Ioannidou A: Agent-Based End-User Development. Communications of the ACM 2004; 47(9): 43-46.

19. Ioannidou A, Repenning A: End-User Programmable Simulations. Dr. Dobb's 1999(302 August): 40-48.

20. Repenning A: Inflatable Icons: Diffusion-based Interactive Extrusion of 2D Images into 3D Models. The Journal of Graphical Tools 2005; 10(1): 1-15.

21. Klann M: D1.1 Roadmap: End-User Development: Empowering people to flexibly employ advanced information and communication technology. EUD-Net: End-User Development Network of Excellence, 2003; 17.

22. Paternò F: D1.2 Research Agenda: End-User Development: Empowering people to flexibly employ advanced information and communication technology. EUD-Net: End-User Development Network of Excellence, 2003; 17.

23. Gross MD: The Fat Pencil, the Cocktail Napkin, and the Slide Library. Proceedings of Association for Computer Aided Design in Architecture (ACADIA '94) National Conference, St Louis, 1994.

24. Michotte A: The perception of causality. Andover, MA: Methuen, 1962.

25. Cooper S, Dann W, Pausch R: Teaching Objects-first In Introductory Computer Science. Proceedings of the 34th SIGCSE Technical Symposium on Computer science education, Reno, Nevada, 2003.