The Emergence Engine:
A Behavior Based Agent Development Environment for Artists

Eitan Mendelowitz

AI Lab - 4532 Boelter Hall
Computer Science Department, University of California, Los Angeles
Los Angeles, California 90095
eitan@cs.ucla.edu

Abstract

Many artists are intrigued by the creative possibilities presented to them by virtual worlds populated with autonomous agents. Artists wishing to explore these possibilities face many obstacles including the need to learn artificial intelligence programming techniques. The Emergence Engine allows artists with no programming experience to create complex virtual worlds. Using behavior based action selection, the Emergence Engine allows artists to populate their worlds with autonomous situated agents. Artists can then direct the agents' behaviors using Emergence's high level scripting language. Artists have used the Emergence Engine successfully since 1998 to create numerous art installations exhibited both in the US and abroad.

Introduction

Computers are becoming increasingly important in the art world. Not only are computers being used as tools for creating traditional art (e.g. drawing, animation, and video) but they are being used as an artistic medium in their own right. Of particular interest is the growing field of interactive computer art. The best of these art pieces often use artificial intelligence technologies to create a meaningful and aesthetically engaging experience for the user.

David Rokeby's piece, *Very Nervous System*, uses vision systems and neural networks to turn a person's movements into music. *A-Volve*, by Christa Sommerer & Laurent Mignonneau, allows users to evolve virtual "creatures" using genetic algorithms. Steve Wilson's *Is Anyone There*, uses digitized speech and voice recognition to call pay phones and engage those who answer in conversation. Each of these installations received critical acclaim in the art world.

While many artists and designers are interested in creating interactive pieces, only a small number have the technical ability to do so. The Emergence Engine addresses the needs of artists who wish to explore the artistic and aesthetic issues of user interaction with autonomous situated agents in real-time 3D virtual environments. The agents created with the Emergence Engine are "believable" as defined by Loyall and Bates (Bates & Loyall 1997). Users are able to suspend belief and accept the agent as genuine. The Emergence Engine allows the artist to create inhabited virtual worlds without first requiring them to master the complexities of programming and artificial intelligence technologies.

The key component in the success of the Emergence Engine is the use of behavior based artificial intelligence for agent control. The Emergence Engine's control system allows artists to create and direct the behaviors of situated agents. The artist's direction of agent behavior can be done interactively in real-time through a graphical front-end or through the Emergence scripting language. Artists have exhibited pieces using the Emergence Engine in numerous art shows including Ars Electronica, and SIGGraph.

Related Applications and Research

A number of research groups have worked on the task of creating and directing autonomous agent interaction in real-time virtual environments. While not geared towards digital artists, their work was helpful in giving Emergence a point from which to start.

The Oz project (Bates, Loyall, & Reilly 1992), allows designers to give agents sets of goals. Each goal contains of sets of behaviors and sub-goals. An agent chooses from the set in order to best satisfy its goals. Behaviors are essentially action scripts heavily influenced by Dyer's work in story understanding (Dyer 1983). Most of the interaction in Oz worlds is linguistic.

Motivate is a commercial product whose main target is game creation. Motivate is a hierarchical finite state machine. Its strength for game companies is that it supports animation blending. For example the "walk" animation can be combined with the "chew gum" animation and the result would be an agent that is both walking and chewing gum. Motivate's "behaviors" are on the level of actions, for example sit and walk. Game designers wanting complex interaction are forced to build their own artificial intelligence libraries.

Copyright © 2000, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

Bruce Blumberg and the ALIVE project (Blumberg & Galyean 1995) use a goal oriented behavior system for situated agents. Behaviors represent both high level goals like "find food" and low level goals such as "move to." Behaviors can execute motor actions, change internal variables, and inhibit other goals. Action is taken on a winner take all basis. Once an action is taken it is performed by the motor system and often involves a sequence of animations. Unlike the Emergence Engine, behaviors are created specifically for an individual agent and require c++ programming. The system runs on an Onyx Reality Engine.

Improv (Perlin & Goldberg 1996) is the system most similar to the Emergence Engine. Like Emergence, Improv also has a behavior based scripting language. In Improv a script is a sequence of actions. Scripts are grouped. When a script in a group is chosen for execution all other scripts in that group are suppressed. Scripts are chosen probibalistically based on decision rules. While the Improv scripting approach seems very different from the Emergence scripting language, Emergence can simulate the Improv approach through probabilistic transition rules. Each character in the Improv system requires a dedicated computer for its behavior system.

Like Motivate and ALIVE (and unlike Emergence) Improv's animation is procedural. Actions taken determine the state of limbs and joints. The Emergence Engine supports key frame animation rather than procedural animation. Digital artists are often skilled animators and desire a high level of control over how their creations move. It was because of Emergence's focus on the artist that the decision was make to use interpolated key frame animation.

Improv was created with virtual theater in mind. As a result, Improv's scripts are performance oriented consisting of behaviors like "turn to camera" and "walk offstage." In contrast, the Emergence Engine was created for virtual environments. There is no on and off stage. More significantly, Emergence agents are situated in the world. Emergence agents sense and react to their environment.

Task Description

The Emergence Engine was created to allow artists and designers to create complex aesthetically pleasing worlds. To that end it was decided that: (1) The Emergence engine must be immersive. (2) The Emergence engine must support the creation of complex agent interactions with relative ease. (3) The Emergence Engine must be usable by artists. And (4), The Emergence Engine must be accessible.

In order to be immersive, the virtual world must engage the users' senses. It was decided that the Emergence Engine must be able to render geometrically complex texture mapped 3D worlds at high resolutions and high frame-rates. Because hearing is an important part of an immersive experience, the Emergence Engine was required to support multiple voices, stereo panning, and distance attenuation. Finally, there was a strong desire on the part of artists to explore other interface modalities such as voice and touch. The design of the Emergence Engine should take these desires into account.

The primary design goal of the Emergence Engine was to support artist creation of intra-agent (i.e. agent/user) and inter-agent (i.e. agent/agent) interactions. Emergence should enable artists to create societies of agents that exhibit such social behaviors as flocking, collective foraging, and construction. Artists should be able to exert as much control over the agents as they please. Emergence should support the continuum of artist direction of agent behavior, ranging from completely autonomous agents to completely scripted agents.

The Emergence design philosophy was to allow artists to use tools with which they were already familiar. To better aid visual artists Emergence would need a graphical interface for behavior control. Finally, most artists are not programmers. The Emergence scripting language would have to be compact and easy to learn.

The last requirement for the Emergence Engine is that it be accessible to artists. Most artists work on tight budgets. While most interactive virtual environments run on high-end computer graphics machines (e.g. SGIs and Onyx reality engines) the Emergence Engine would be designed to run on high-end personal computers. The requirement of running a real-time system on a PC platform affected many choices in the implementation of the Emergence Engine's artificial intelligence components.

Application Description

To appreciate the Emergence Engine, it is important to understand for what Emergence is used. To that end, this paper describes an example Emergence Engine installation. It then describes the design process used by artists to create such an installation. After describing how the system is used, this paper will describe the software architecture of the Emergence Engine. Special attention will be given to artificial intelligence techniques used in its execution.

An example installation

The Emergence Engine has been installed in many art shows all over the world. Its first installation was Rebecca Allen's work entitled "The Bush Soul (2)" in SIGGraph 1998's "Touchware" art show. The installation used three networked personal computers to render a three screen first person panoramic view of the virtual environment. Because the system is interactive, no two user experiences were the same. What follows is a brief narrative description of a typical user experience.

The installation begins with a view of the avatar, the user's embodiment in the virtual world, perched on a hilltop. Below the avatar is a "village" teeming with life-like but abstract creatures.

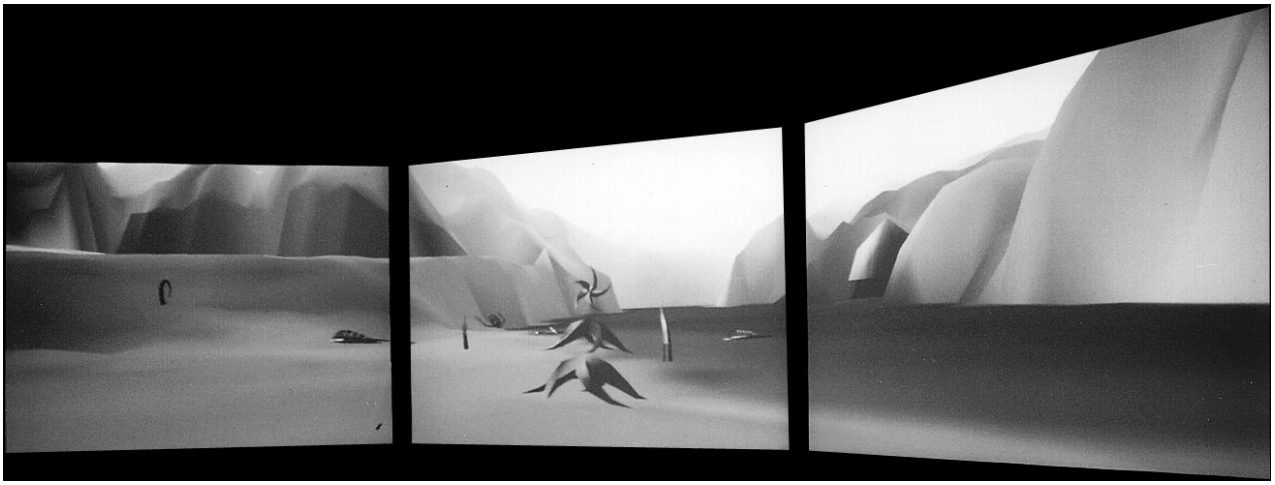

Figure 1: Photograph of the Emergence Engine's three-screen installation at SIGGraph 1998

Through the use of a gamepad, the user is free to explore the world by controlling the avatar's movements. As the Avatar moves down the hill into the Village the user encounters three tall creatures made out of whirling 5 pointed stars. The Whirlers by pass the user uninterested, mumbling among themselves. A small childlike Whirler follows them. Upon seeing the avatar the child playfully follows the user around bouncing happily and giggling. The small Whirler is conflicted; it wants to stay playing with the avatar but also wants to stay with the adult Whirlers. The Whirler plays with the avatar for a time then decides to rejoin its group.

Also inhabiting the Village are two large lumbering three legged creatures, Tripods. Occasionally the Tripods get tired and curl up into a protective rocklike ball. The Tripods feel threatened by strangers; when the avatar approaches, they try to scare off the user by rearing-up on a single leg. If the user continues to approch, the Tripods try to push away the user's avatar with their two free legs.

Overhead soars a flock of birds. Scattered about the Village, swaying in the breeze, are colorful flowers resembling oversized tulips.

Tribal music starts to play and, upon hearing the music, one flower manages to wiggle its way out of the ground. The flower glides across the village to coax a second flower out of the ground. Once together, the flowers begin a coordinated dance.

The above narration only describes a fraction of the interactions and behaviors present in the Village. The village is a small part of the entire virtual world. The world has four other regions of equal complexity. Artists were able to create such complicated intra-agent and inter-agent relations by using the artificial intelligence components built in to the Emergence Engine.

The Design Process

Using behavior based situated agents and a high level behavior scripting language, the Emergence Engine allows artists to create populated worlds without worrying about technical details. Using the Emergence Engine, the virtual world design process does not begin with software design. Rather, artists begin the design process as they usually do with concept drawings, descriptions of scenarios, and character sketches.

Once done with the concept, the artists create "bodies" for their agents. Emergence allows artists to create animated models using the standard animation packages with which they are already familiar (e.g. SoftImage, 3DMax and Maya). These models are imported into Emergence for use as agent bodies. The Emergence Engine uses a simple gray scale image as a topographical map to create world terrain.

Once the bodies of the agents are complete, artists concentrate on agent behavior and interaction. Typically, designers will have a character sketch of an agent (or a set of agents). The sketch may be something like "this character is wary of strangers." Artists can choose from a palette of behaviors which behaviors each agent should exhibit. Using a graphical interface, artists can modify behaviors in real-time allowing them to work visually, as they are accustomed. Once an artist is satisfied with an agent's behavior he/she can export the behavior settings directly into a script.

The final step of the design process is script creation. Scripts support higher level interaction. Using scripts artists can simulate changes in emotional state or goals. In addition, artists can use scripts to directly instruct agents to take a particular action.

The Emergence Architecture

The Emergence Engine has five components: the graphics module, physics module, the networking module, the behavior module, and the scripting module. Each component works independently and communicates with each of the other components through well defined interfaces. All five components will be described below

with special attention given to those of interest to this conference: the behavior module and the scripting module.

The Graphics Module

The graphics module was designed to support real-time 3D environments. The Emergence graphics engine makes the most of available consumer-level OpenGL acceleration to provide performance that rivals high-end graphics workstations. The graphics engine renders an average of 6000 texture mapped or vertex colored polygons. The Emergence Engine supports interpolated key frame model animation, dynamic lighting with multiple light sources, and particle systems.

The graphics module receives model locations from the physics module. The scripting module provides the graphics module with key frame information for animating agent models.

The Physics Module

The Emergence Engine supports reasonably realistic real-world physics. To that end the physics module has two tasks, collision detection and force application.

Every object in the world has a collision model. For computational efficiency, the collision model is usually a simplified version of the agent model. When an agent moves between frames, its collision model sweeps out a collision volume. A frame is the smallest unit of time in the Emergence Engine. If the collision volumes of two objects are interpenetrating they are said to be colliding. Collisions are computed using hierarchical bounding boxes allowing for quick and efficient collision detection. This approach is similar to that used by I-Collide (Cohen, Lin, Manocha, &. Ponamgi 1995)

All agents in the world may be subjected to physical forces. The artists can choose what forces should be applied to a given object in the world. The artists are given a standard set of forces from which to choose (e.g. friction, gravity, and drag). In addition to choosing the forces to which an agent is subjected, artists can set the physical qualities of an agent (e.g. mass, coefficient of friction, elasticity).

The physics engine runs on a separate thread than the graphics engine. Given a fast enough computer or multiple processors, threading allows the physics engine to run at a higher frame rate than the graphics engine. The result is better physical realism through super sampling.

The physics module provides the graphics module with the location and orientation of objects in the world. It also provides the situated agents with many of their senses by sending events to the behavior and scripting modules.

When the physics engine detects a collision is sends the agent a collision event. The collision event tells the agent with what it collided, where the collision took place and with how much force the collision occurred. The collision events constitute the agents' sense of touch.

In addition to having a collision model every agent has a vision model. The vision model represents the agent's field of vision. When the physics engine detects an object within an agent's vision model it sends a vision event to the agent. Agents are given knowledge of visible objects and their position through the vision events.

Finally, the physics module keeps track of sound attenuation. When an agent make a sound, all agents within hearing distance are sent a hearing event along with the sound that was made (in the form of the sound's filename).

The Networking module

The Emergence Engine supports networking between computers over TCIP. While each machine on the Emergence network has a local copy of agents, only one computer on the network, the server, has "ownership" of that agent. The server broadcasts changes in that agent's state over the network to other participating machines. The network is tolerant to lost packets, if information about an agent's position is lost the local machine interpolates the agent's current position until accurate information is received. Any machine on the network can act as a display and render the world.

At SIGGraph '98, the Emergence system displayed a three screen panoramic first person view of the Emergence world using three networking computers.

Another feature of the networking module is that it allows arbitrary string messages to be sent over the network. The network messaging was included to allow for the integration of sensors and other devices with the emergence system.

For example, at SIGGraph '99, a separate PC with a touch sensor was used as a networked input device. The sensor informed agents in the virtual world when a user approached the installation. The networking module allows artists flexibility in examining interface modalities.

The last two modules of the Emergence Engine, the behavior module and the scripting module constitute Emergence's use of artificial intelligence technology.

Uses of Artificial Intelligence Technology

The Behavior Module

Agents are situated in the virtual environment and thus respond to external stimuli or lack thereof. In addition to the senses sight, touch, and hearing, agents can sense string "messages" and can sense the passing of time. At every frame, the agents must arbitrate between different and possibly competing behaviors to arrive at a single action.

The Emergence Engine behavior module resembles Craig Reynolds' steering behaviors (Reynolds 1999). The Emergence Engine uses a system of relative weights to choose between competing behaviors for action selection. This approach is not so different from the inhibition/excitations approach often used by other behavior systems. Raising the weight of one behavior will increase its influence while simultaneously decreasing the

influence of all other behaviors. In effect, excitation of one behavior inhibits all other competing behaviors.

Low-level behaviors do not depend on sensory input. Such behaviors are of the type: move to a specific point, face a particular direction, move around at random, slow down, etc.

Other behaviors are higher level and require vision or other senses. Such behaviors include: agent following, collision avoidance, path following, agent watching, move in formation with other agents (e.g. to an agent's left), etc.

Every active behavior, those with non-zero weights and the required stimuli, chooses a goal point. A single unified goal point is decided upon by computing the weighted average of all the behaviors' goal points. While more complex arbitration schemes exist including decision-theoretic approaches (Pirjanian & Mataric 1999) and constraint-based approach (Bares, Grigoire, & Lester 1998) they are often computationally expensive.

The Emergence Engine is required to arbitrate between dozens of behaviors for scores of agents on a single personal computer. Weighted averaging was chosen for its computational frugality. The choice of a simple arbitration method does not require a sacrifice in behavioral complexity. Many of the benefits gained from more complex arbitration schemes can be achieved using weighted averages if behaviors are allowed to modify their own weights. For example, if a collision is imminent the collision avoidance behavior can temporarily double (or quadruple) its own weight in order to suppress the influence of other competing behaviors.

Once a single goal point is selected, it is passed on to the agent's motor system. Not all agents have the same motor abilities. Some agents can fly while others are restricted to movement on the ground. Some agents can move sideways while others must move in the direction they are facing. Agents move by telling the physics module to apply a force and/or a torque to their bodies.

In addition to its computational efficiency, behavior based control has a representational advantage for the artist. The approach allows artists to be presented with a palette of behaviors. Each behavior has a concrete meaning and the system of relative weights is intuitive. Using the Emergence Engine's graphical front-end artists can change behavior weightings while the system is active. Such interactivity greatly shortens the amount of time required by the artist to arrive at desired agent behaviors.

The Scripting Module

The Emergence scripting language was designed to be small and easy for artists to learn. The scripting module is used for high level control over agent behavior.

Similar to Brooks' subsumption architecture (Brooks 1986), the Emergence scripting module implements multiple augmented finite state machines running in parallel. Communication between different finite state machines is done by passing string messages. Unlike the subsumption architecture there is no hierarchy. All finite state machines have equal influence over an agent.

The scripting module is tightly coupled with the Emergence Engine's behavior module. Any behavior value an artist can set interactively, through Emergence's graphical front end, can be scripted. Usually, this feature is used to simulate changes in an agent's emotional state or mood by changing the relative weights of different steering behaviors.

In addition to changing behavior variables, the Emergence scripting language allows for the creation and destruction of agents, the playing of sound files, the playing of key framed animations, and the sending of messages. Scripts can also call other scripts (and wait for them to terminate), or spawn other scripts (and run them in parallel).

Like a standard finite state machine, every script has a special start state which is entered when an script is executed. Upon entering a state, a list of statements is executed sequentially. Executed statements perform the aforementioned actions.

Every state has a list of transitions. Each transition has a condition and a state. When a transition's condition is satisfied the script enters the transition's state.

Emergence transitions are associated with sensory events. These events correspond to an agent's senses and include vision, collision, hearing, message, timer, immediate, and animation-end. The physics module generates the vision, collision, and hearing events. The timer event is generated at a set time interval specified in the state. The immediate event is generated once, immediately after the last statement in the state executed. The animation-end event is called when the graphics module displays the last key-frame of an animation sequence.

Transition conditions are only checked when their associated event is sent.

Another departure from traditional finite state machines is the use of local variables. Every state can have a list of parameters. Upon entering a state, values are bound to the members of the parameter list. The use of parameters allows scripts to be compact and reusable.

The following is an example of a script. It instructs an agent to follow creatureA. If creatureA bumps into the agent the agent is instructed to avoid creatureA and the script terminates. CreatureA is a parameter and is bound to a value when the script is called.

```
State FollowTheCreature(string CreatureA) {
  Follow.Weight[CreatureA] = 10;
  // follow CreatureA with a weight of 10
  Follow.Distance[CreatureA] = 2;
  // follow A from a distance of 2 meters
} transitions {
  OnCollision {
    if that T== CreatureA then
      AviodTheCreature(CreatureA);
    // if that with which you collided
    // is creatureA then go to state
    // AviodTheCreature
```

```
  }
}

state AviodTheCreature(string CreatureA) {
  Follow.Weight[CreatureA] = 0;
  //stop following creatureA
  Aviod.Weight[CreatureA] = 10;
  //avoid creatureA with a weight of 10
}
//since there are no transition statements
 //the script terminates in state
 //AviodTheCreature
```

Application Use and Payoff

Since the spring of 1998, the Emergence Engine has been used by dozens of artists to create interactive virtual environments. Artists typically require less than two weeks of learning before becoming comfortable with the Emergence Engine. Installed in art shows and conferences including SIGGraph 1998, SIGGraph 1999, and Ars Electronica 1999, Emergence has been experienced by thousands of users. The Emergence Engine has allowed artists to examine issues of human-computer interaction, artificial intelligence, and virtual interactive environment without requiring them to learn how to program.

Application Development and Deployment

The Emergence Engine began in 1997, when Professor Rebecca Allen of UCLA's Department of Design | Media Arts assembled a team of artists and computer scientists to explore issues of human–agent interaction. This original team created a prototype Emergence Engine that supported agents with a few hand-coded behaviors. This prototype system was exhibited at Art Futura in Madrid, Spain in 1997.

Using lessons learned from the prototype, Loren McQuade and I collaborated to create the Emergence Engine architecture. The development process began with months of discussions with Professor Allen and other artists. After determining the needs of said artists we spent a month planning the Emergence software architecture. The Emergence Engine itself was written in C++ and some assembly over the course of seven months. Flex and Bison were used in the creation of the Emergence scripting language.

The Emergence Engine runs on any Intel platform machines using Windows NT as an operating system. The Emergence Engine requires a graphics card that supports OpenGL.

Current efforts involve making the scripting language even easier to use. A script editor was created in the spring of 1999. In the future we hope to develop a graphical interface for the scripting language completely freeing the artist from having to write code.

Conclusion

The Emergence Engine provides a unique development environment for designers and artists to explore virtual world creation. Using behavior based techniques and a high level scripting language, the Emergence Engine allow designers with little or no programming experience to create situated agents that interact with the user in intelligent and meaningful ways. The Emergence Engine has been used to create a number of successful interactive computer art installations enjoyed by thousands of users all over the world.

Acknowledgments. I would like to thank Professor Rebecca Allen, Director of the Emergence Lab, and the other artists who used the Emergence Engine to create works of art I could not have imagined. I would also like to thank Intel for their support of the Emergence Lab.

References

Dyer, M. 1983. *In-Depth Understanding.* , Cambridge, Mass.: The MIT Press.

Loyall, A. and Bates, J. 1997. Personality-Rich Believable Agents That Use Language. *Proceedings of the First International Conference on Autonomous Agents*, 106-113. New York, N.Y.: The Association for Computing Machinery.

Bates, J., Loyall, B., and Reilly, W. 1991. Broad Agents, *Proceedinge of the AAAI Spring Symposium on Intergrated Intelligent Architectures*. Stanford University, Calf.: AAAI Press.

Blumberg, B., and Galyean, T. 1995. Multi-Level Direction of Autonomous Creatures for Real-Timer Virtual Enviroments. *Proceedings of SIGGRAPH95*. New York, N.Y.: The Association for Computing Machinery.

Perlin, K., and Goldberg, A., 1996. Improv: A system for Scripting Interactive Actors in Virtual Worlds. *Proceedings of SIGGRAPH96*. New York, N.Y.: The Association for Computing Machinery.

Brooks, R. 1986a. A Robust Layered Control System for a Mobile Robot. *IEEE Journal of Robotics and Automation*. RA-2:14-23.

Bares, W., Grigoire, J., and Lester, J. 1998. Realtime Constraint-Based Cinematography for Complex Interactive 3D Worlds. *Proceedings of the Tenth Conference on IAAI*. Menlo Park, Calf.: AAAI Press.

Pirjanian, P. and Mataric, M. 1999. A decision-theoretic approach to fuzzy behavior coordination. *Proceedings of the IEEE Conference on Computational Intelligence in Robotics and Automation*, Monterey, Calif:, IEEE.

Reynolds, C. 1999. Steering Behaviors For Autonomous Characters. Computer Game Developers Confernece. San Francisco, Calf: Computer Game Developers Confernece.

Cohen, J., Lin, M., Manocha, D., and Ponamgi. K. 1995. I-COLLIDE: An Interactive and Exact Collision Detection System for Large-Scaled Environments. Proceedings of Association for Computing Machinery International 3D Graphics Conference. New York, N.Y.: The Association for Computing Machinery.