Braitenberg and the Browser

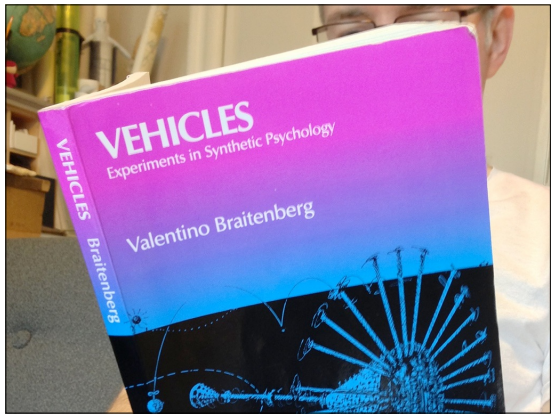
The title of this talk is Braitenberg and the Browser. Valentino Braitenberg invented Braitenberg Vehicles. We'll talk about Braitenberg Vehicles and how to recreate them in a web browser using JavaScript. We'll also talk about some of the broader concepts Braitenberg illuminated when he created the vehicles.

Vince Allen

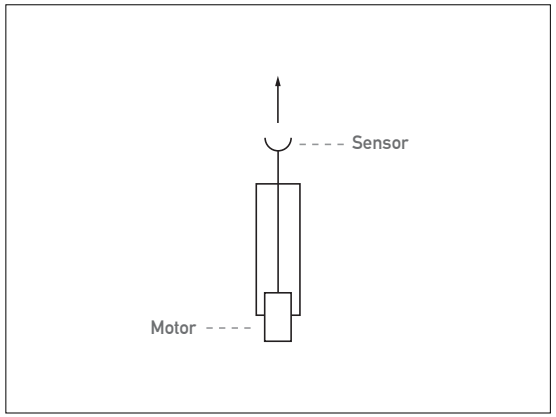
I'm Vince Allen, software engineering manager at Spotify. You can reach me @vinceallenvince on Twitter and find code for all the demos at github.com/vinceallenvince.

Valentino Braitenberg was an Italian neuroscientist interested in studying how human beings developed reasoning and intelligence.

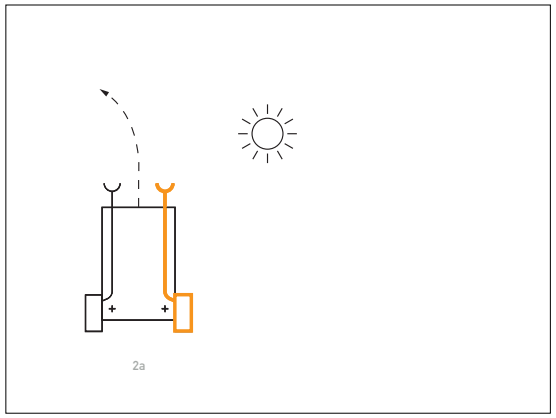
Instead of looking at humans, he started with insects. By examining simple brains, he hoped he would find the building blocks to our cognitive evolution. He also carefully considered his methodology. He focused on the mechanics and inner workings of simple systems as a path toward answering larger questions. By understanding how our brains evolved, he hoped to better understand how our brains work.



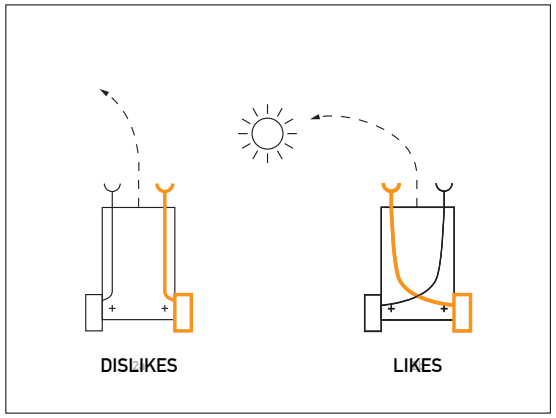
In the late 1970's he published a book called "Vehicles: Experiments in Synthetic Psychology". The book describes a thought experiment where he imagines 10 simple vehicles. These vehicles became known as Braitenberg Vehicles.



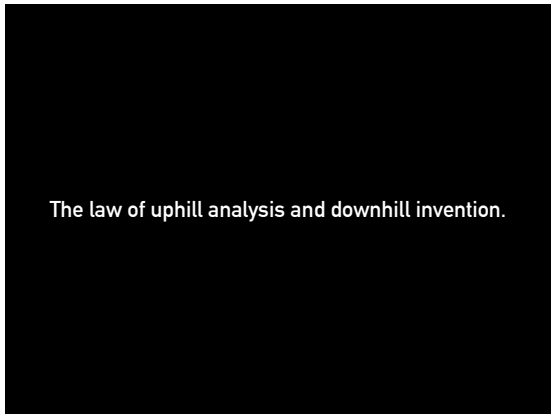
Each vehicle is equipped with sensors tuned to specific input. For example, the first vehicle has one sensor wired to a motor. The sensor is tuned to temperature and activates the motor in proportion to the temperature it senses. The effect… the vehicle slows down around cold things and speeds up around hot things.



The vehicles progress with increasingly sophisticated sensors and wiring. For example, Vehicle 2a has two sensors wired to two motors. The sensors are activated by light. The right-side sensor is closer to the light source and delivers a stronger input to the right-side motor. The result, the vehicle steers away from light.



Vehicle 2b has two sensors wired to two motors on opposite sides. When exposed to light, the left-side sensor drives the right-side motor more than the left-side motor. The result, the vehicle steers toward the light. As observers, we would say vehicle 2a DISLIKES light while vehicle 2b LIKES it. At a distance, without any knowledge of how these vehicles were designed, we project a certain degree of intelligence on to the vehicles.

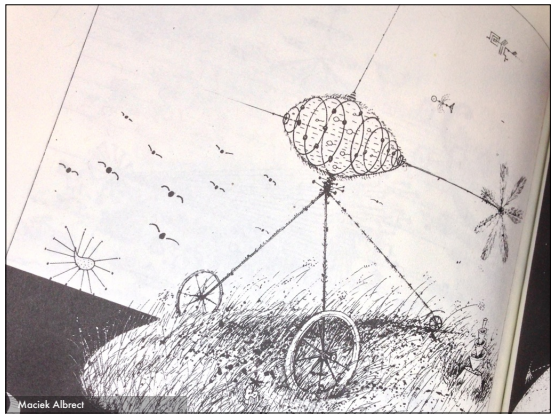


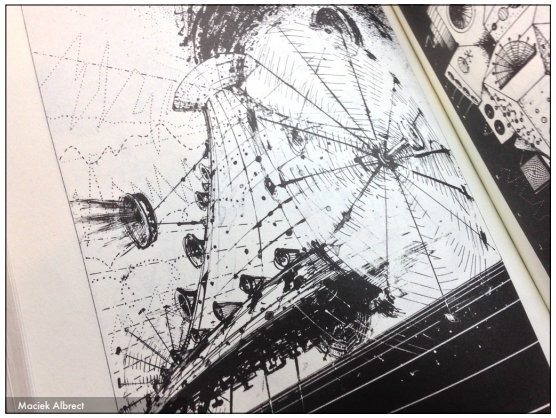
But as observers, we're wrong due to a theory Braitenberg calls "The law of uphill analysis and downhill invention". When observing a system, we typically overestimate its complexity. As designers, we know our vehicles behave according to rules dictated by a simple internal mechanism.

The law of uphill analysis and downhill invention.

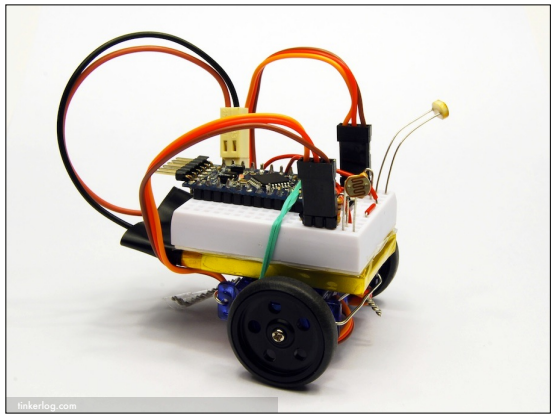
However, as observers we can only guess at that mechanism. As a result, we interpret the vehicles in human terms and attribute qualities like fear, aggression, and desire to their behavior.

The interpretive space is the magic and appeal of Braitenberg Vehicles. As observers, the vehicles' inner workings are completely open to our imagination. Artist Maciek Albrect provided some beautiful illustrations for Braitenberg's book that demonstrate how far our interpretations can go.

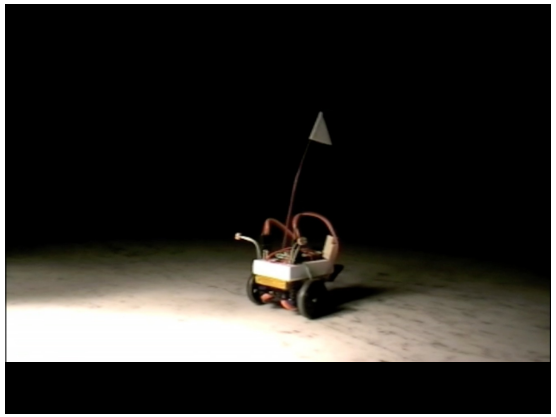




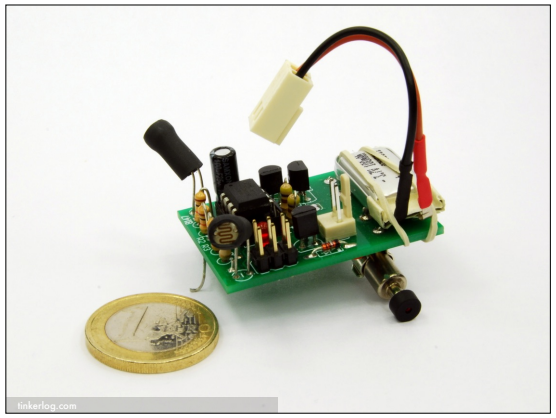
It's really interesting Braitenberg included Albrect's illustrations… by doing so, he vividly juxtaposed the contrasting perspective of the inventor and observer.



Alex Weber is an inventor who created Braitenberg Vehicles using Arduinos, servos and photo-resistors. You can read more on his blog at tinkerlog.com. You can see the photo resistors here positioned as antennae. They are wired to an Arduino which controls the servo motors. Here it is in action.

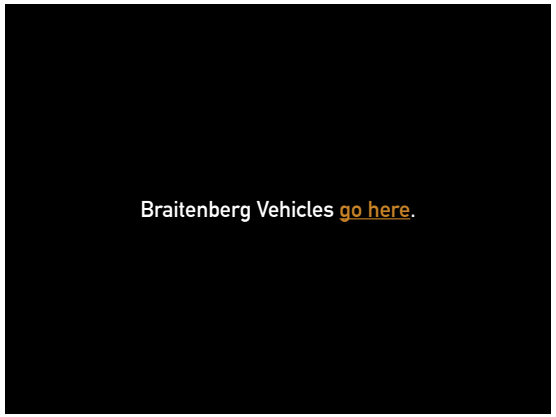


http://vimeo.com/5029632



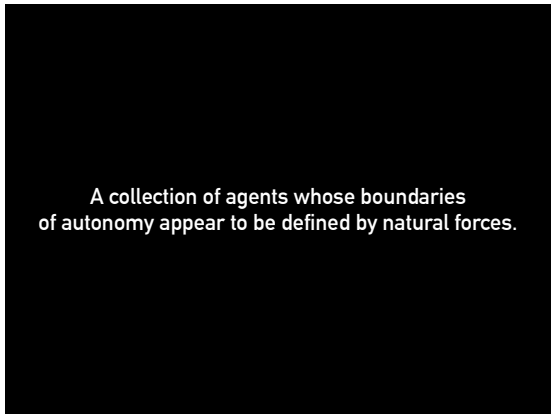
This one is even smaller and uses a custom circuit board.

http://vimeo.com/5664333

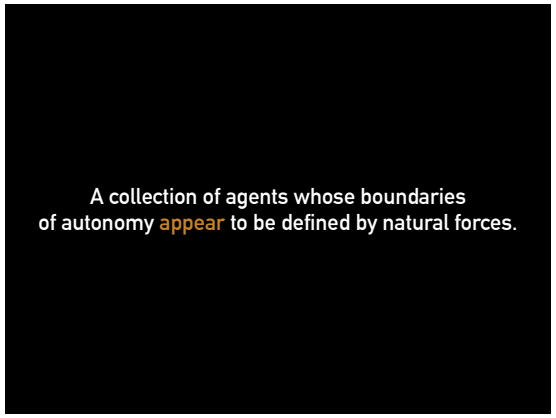


As a JavaScript developer, my first thought after seeing these was, how can I recreate them in a web browser?

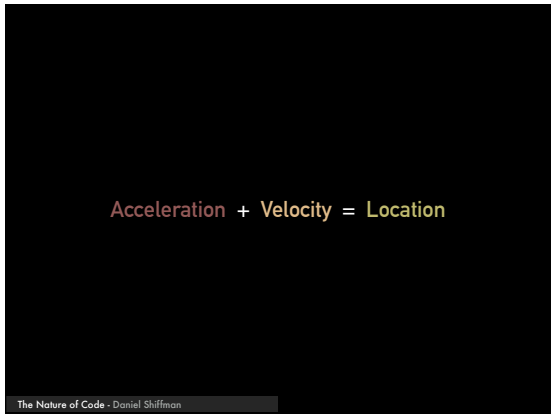
For examples visit:
http://vinceallenvince.github.io/JSCampRO2014/



How do we go about building something like this in a browser? Let's talk about creating a natural simulation. What is a natural simulation? I like this definition… a collection of agents whose boundaries of autonomy appear to be defined by natural forces.



"Appearance" here is important. Remember the law of uphill analysis and downhill invention? Creating a convincing natural simulation only requires enough rules to convince the observer… and from what we've seen with Braitenberg Vehicles, observers are easy to convince.



We're not going to create a library with hyper-accurate physics and collisions, etc. Again, think about Braitenberg Vehicles… a combination of very simple rules can be very persuasive. Instead of creating a physics engine, we'll create a system that simply outputs the sum of forces in our simulation. We'll then apply that sum to agents on the screen and move them around. I'm crediting Daniel Shiffman here because I'm basing this approach on principles he outlines in his book, The Nature of Code.

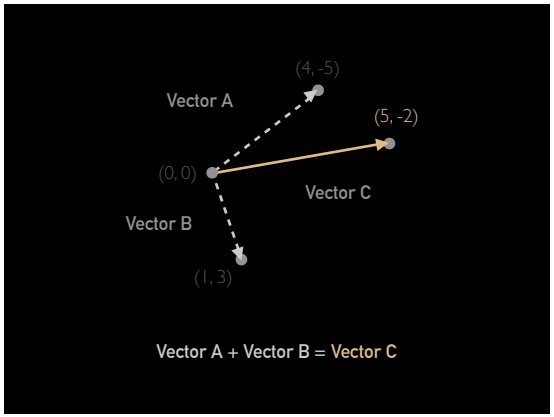
SuperSimpleSim classes + responsibilities

System
- setup World conditions (document.body)
- define loop (request animation frame)
- draw items on screen (style.cssText)

Item
- create a DOM object ( )
- assign visible (ex: color) and non-visible (ex: mass) properties
- define a step() function

Vector
- provide properties and method to help position items

Our simulation framework is called SuperSimpleSim and contains the following Classes… System, Item and Vector.

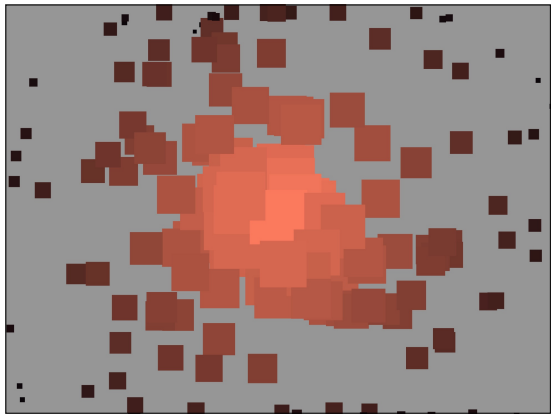


A vector is an offset. It's the difference between two points. It also implies a magnitude. Vector A has a greater magnitude than Vector B. We'll use vectors to represent forces in our system. And we'll use the sum of those forces to determine the location of our items at any point in time. For example, if Vector A is steam coming from the ground and Vector B is wind, what would it feel like to stand in this system? We would feel Vector C… a force point up and to the right.

Super Simple Simulator demos go here.

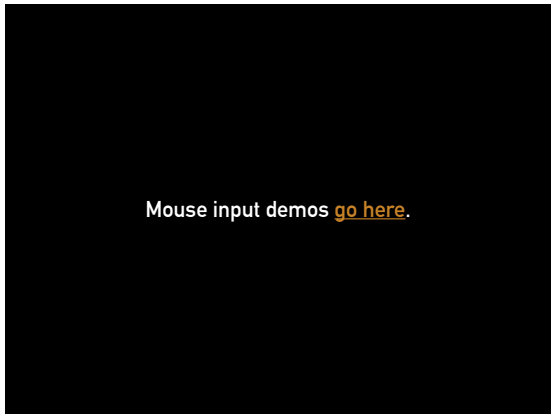
For examples visit:

http://vinceallenvince.github.io/JSCampRO2014/



Now that we have a basic simulator running in the browser, we could extend it to handle the type of interactions you saw in the Braitenberg simulations. However, most Braitenberg experiments leave the inventor out of the system. What if we want to influence it?

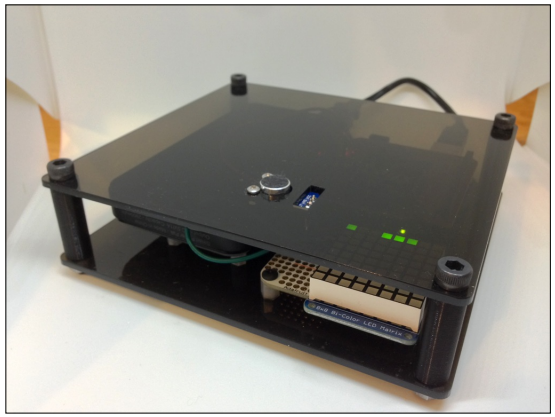
Let's talk about three ways we can assert ourselves. First, the mouse. We want our interaction with a natural system to be as natural as possible. A mouse is limited in this respect. It's like a weird little hand that taps on things. But we can express acceleration with a mouse.



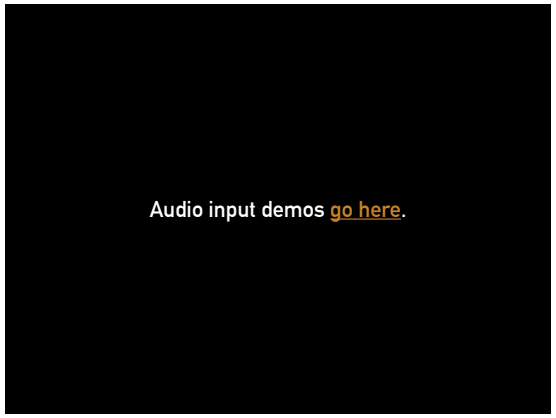
For examples visit:

http://vinceallenvince.github.io/JSCampRO2014/

Braitenberg simulations in the real world typically involve setting up stimulants around a room and setting loose some robots on the floor. We saw Alex Weber use light as a stimulant…

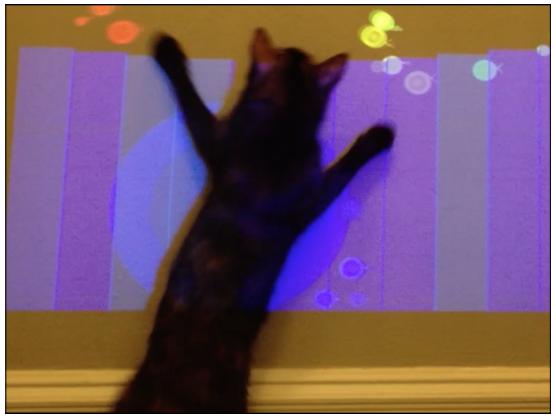


But what about using sound? Imagine a thirsty deer picking up the sound of a creek. The sound of a stream would probably inspire specific behavior. To experiment with sound as a stimulant, I built two of these… they are wireless microphones connected to Arduinos. When a mic detects a specific frequency, it wirelessly sends the magnitude of that frequency to a local node server running Node serial port. The node app then forwards that value to our simulation in the browser via web sockets.

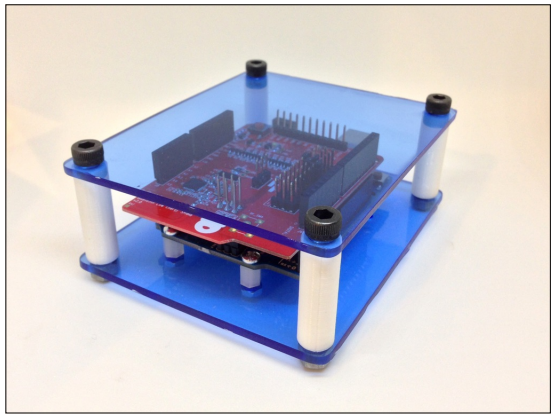


For examples visit:

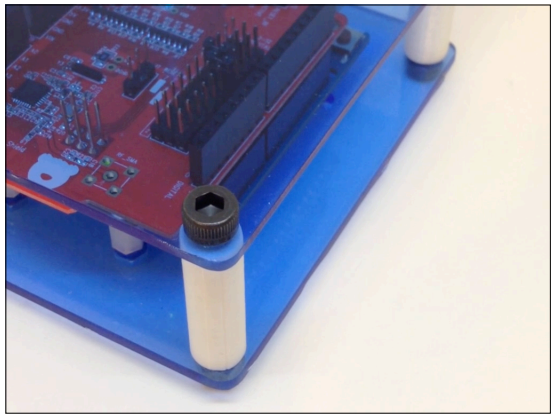
http://vinceallenvince.github.io/JSCampRO2014/



How could we involve our physical bodies directly in a Braitenberg simulation? We probably all carry a mobile device with several onboard sensors. If a Braitenberg Vehicle is essentially a stack of sensors with wheels, why not turn our phones into Braitenberg Vehicles.



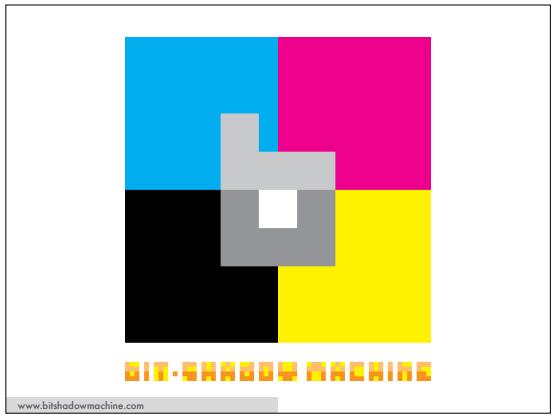
To do this, I've coded up an iOS app that sends motion data to an Arduino over Bluetooth. I'm again using Node serial port to forward that information to a node app which sends it to the client via web sockets.



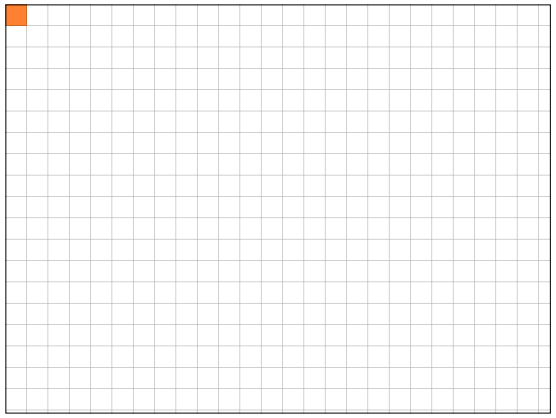
For examples visit:

http://vinceallenvince.github.io/JSCampRO2014/

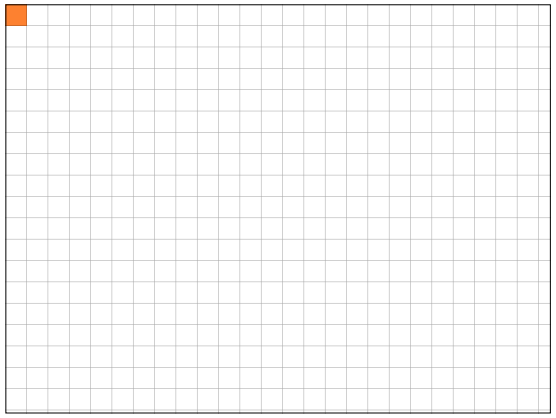
Remember Braitenberg's law about uphill analysis and downhill invention. At this point we've crossed over from inventor to active participant. But what about observer? How do we create simulations that are meant for viewing only. In other words, using JavaScript, how could we output a natural simulation to video.



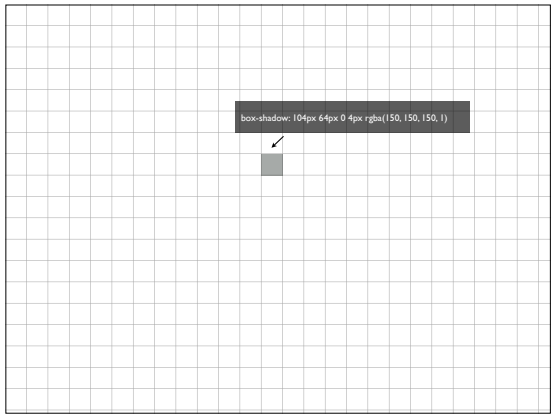
My current approach is called Bit-Shadow Machine and depends entirely on CSS box-shadows. Let me explain how it works.



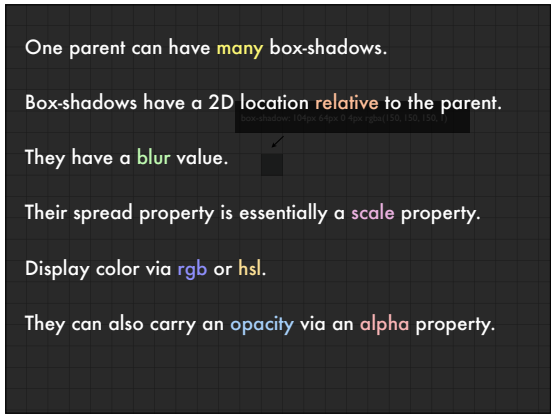
We divide the visible browser into a grid and place a single div in the top left corner.



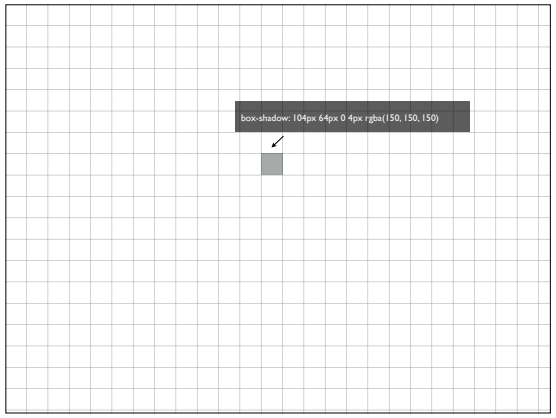
Next, we hide the div by assigning a height and width of 0.



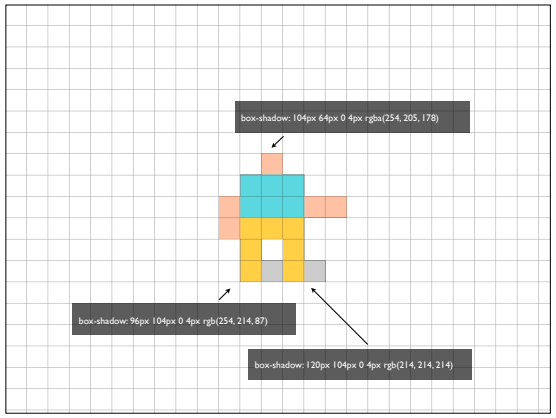
An interesting thing about box-shadows, if we hide the parent like this, we can still see its box-shadows.



One parent can have many box-shadows. Box-shadows have a 2D location relative to the parent. They have a blur value. Their spread property is essentially a scale property. They display color via rgb or hsl. They can also carry an opacity via an alpha property.



To render an item in simulation, I only need location, scale and color. Box-shadows give us those properties and more.

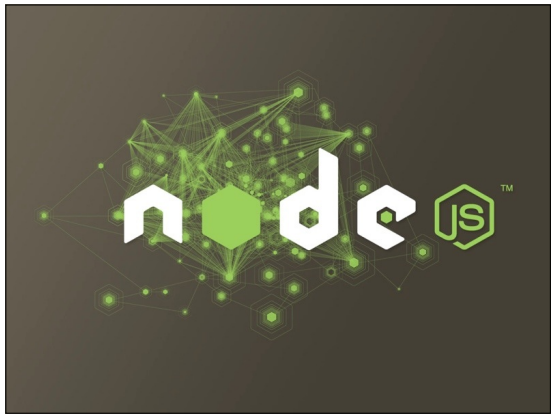


This means it's only a matter of adding more box-shadows and adjusting their properties to get what we want.

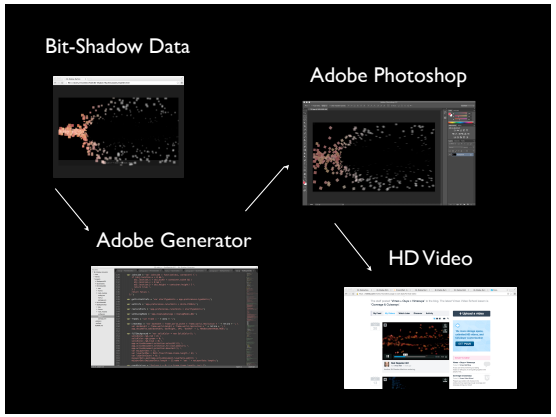
For examples visit:

http://vinceallenvince.github.io/JSCampRO2014/

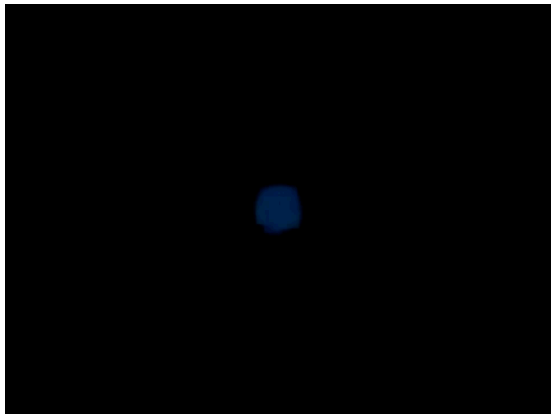
How could we output these renderings to video? We'll use two things… Photoshop…



… and NodeJS.

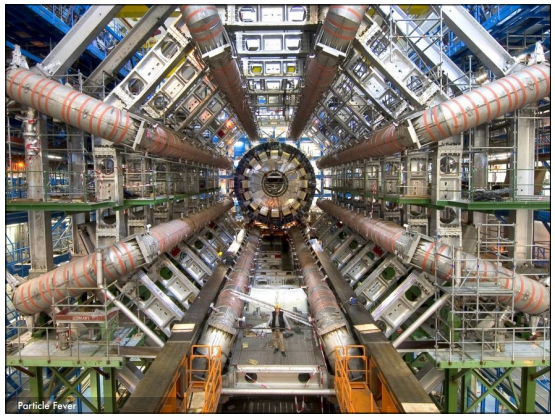


With the 14.1 release of Photoshop, you get Adobe Generator which allows you to automate tasks in Photoshop using Node. Generator allows you to create Photoshop plugins which means we can write Photoshop plug-ins using JavaScript. I created a plug-in that renders each frame of these simulations and outputs still frames I eventually compiled into video.

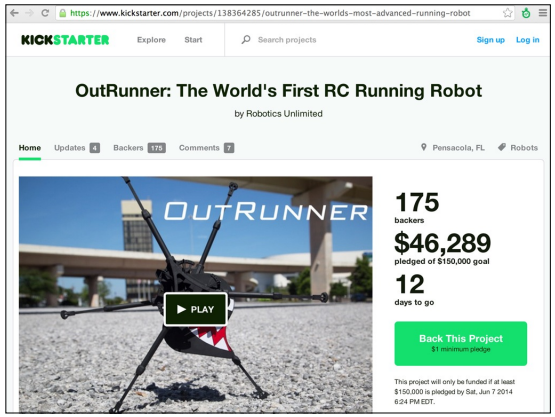


http://www.bitshadowmachine.com/video/blueagents001/

http://www.bitshadowmachine.com/video/redrepeller002/

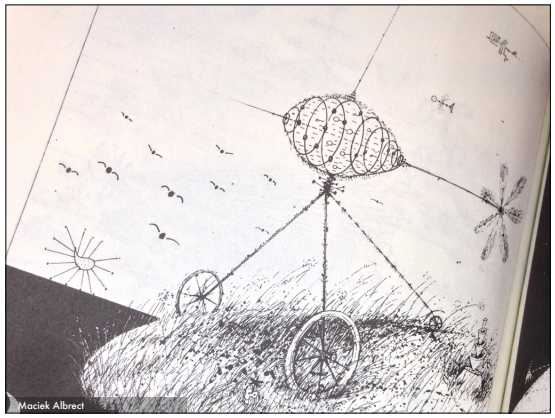


This photograph is from a documentary called Particle Fever that follows several physicists as they conduct the first experiments in CERN's Hadron collider. The documentary introduced me to the contrasting relationship between theoretical and experimental physicists. And it reminded me of relationship Braitenberg describes between the inventor and the observer. Both want to better understand the natural world. But by reading Braitenberg's "Vehicles", you may think the inventors have the advantage… like these guys…

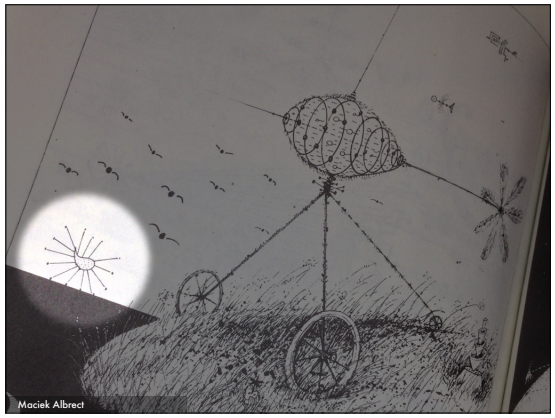


Robotics Unlimited is currently running a Kickstarter campaign to create a robot they claim is the first commercially available legged robot. The design is "biologically inspired" and meant to emulate running motion seen in legged animals.

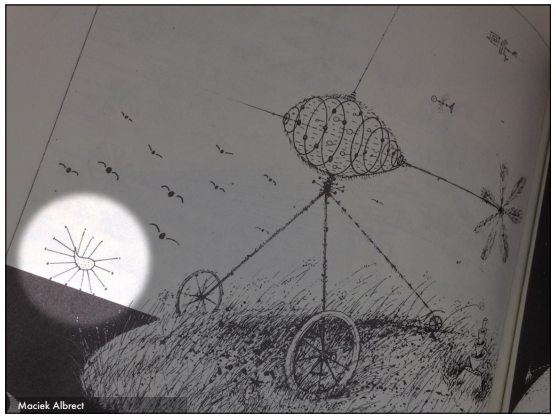
http://www.youtube.com/watch?v=NcO_hKkFxlE



Remember Albrect? He's the artist observer who provided illustrations in Braitenberg's book…



That's an Out Runner! "Vehicles" was written in the 70's… likely before the founders of Robotics Unlimited were even born. So between the inventor and observer, who is really leading and challenging our understanding of the natural world?



As JavaScript developers, I believe we can occupy a place in between the two. I hope I showed you today how we can be participants. We can code simulations and open the door for natural interaction. Or we can sit back and simply enjoy what we've created. Either way, I hope you've enjoyed this talk and had a great JSCamp.

FloraJS is a JavaScript framework for creating natural simulations in a web browser. I used it to create most of the demos for this talk including the Braitenberg Vehicle demos. SuperSimpleSim is a much simpler version of FloraJS that we used to demonstrate the basic principles of rendering simulations in a web browser.
