FlowPath

CS 498SL Virtual Reality: Project Proposal

David Young: dcyoung3

March 2016

Visualizing runtime flowpath in the context of interactive circuits formed of finite primitive elements. Creating an interactive VR sandbox for the creation of digital logic circuits and graphical scripting.

Contents

1 Motivation and background:

1.1 Why would this project be interesting from a VR perspective:

Virtual Reality can provide unique ways to visualize data, geometry or systems. As a medium, VR is powerful because it can expose audiences to new things, delivering content in a compelling way that inherently attracts a wide audience of people to content they would normally overlook. This makes VR well suited to empathy induction and educational applications. On a more relevant note, it also means that users will be drawn to an interactive virtual sandbox more than a 2D counterpart. Job Simulator is an excellent example of this. This project aims to create a unique interactive sandbox for digital logic circuits, designed specifically for Virtual Reality. Educated users will be eager to explore a familiar topic in a novel medium, and the sandbox will be a compelling enough interactive/visualization experience to attract users that would otherwise avoid engaging the topic. This is the exact same reason TiltBrush can engage individuals that would never sketch a drawing on paper. Virtual reality will provide a "God's Eye" view of circuit design. Beyond the scope of the project, there is enough depth to explore educational opportunities and game-ification via puzzle driven gameplay or task driven learning.

1.2 Relevant Existing Work:

For visual scripting, the most similar existing applications are Scratch and Unreal Engine's blueprints. Both scratch and blueprints are 2D graphical scripting languages with no 3D or VR elements. Scratch is similar because it uses very few elements and emphasizes a simplistic/intuitive design for forming programs. It's different because users do not form flow paths which can be visualized. On the opposite end of the spectrum, UE4 blueprints emphasize visual flow paths in graphical scripting but provide large quantities of complex elements with which to form functions and object oriented programs.

For the design of digital logic circuits, the most similar existing applications are Logisim or Minecraft's Redstone. Logisim is an educational tool for designing and simulating digital logic circuits. You drag and drop primitives onto a scratch board and form connections with lines that pulse green if they are active connections (ie: currently electrically flowing). The system is powerful in that one can create complex simulations of entire CPUs while viewing the flow of any individual link between elements. Logisim is a phenomenal education tool, but it is 2D and not very engaging or graphically stimulating. Conversely Minecraft is a 3D world designed for entertainment and has proven extremely engaging. Some minecraft users have discovered a way to create digital logic circuits. The techniques involve using a binary switch based on torches and connecting material called "Redstone Dust" (its like gunpowder lines between matches). This is an unintended use of a rule abiding system. And although the functional primitives are cumbersome and extremely limited, users have gone to great lengths to combine them... even forming entire calculators and CPUs.

1.3 What problem does it solve? What does it contribute:

Existing implementations fail to engage the user with creative and immersive interaction, and runtime flow is at best informative. Using interactive motion control and addicting visual feedback, this project would entice users to construct working flowpaths in a way that is intrinsically motivating. This project aims to balance simple elements and connections a user can intuitively master, with creative enough designs that users are interested and engaged. Emphasis would be on interactive placement of primitive elements and creative/artistic visualizations of flow paths in 3D space. Not only is visualizing flow patterns an extremely intuitive way to understand flow paths or program execution, it can also be extremely fun to watch. The visualization of successful flow can itself be a reward in a habit formation process providing incentive to create flow paths.

2 Description:

On a broad level, the theme of the project is an open ended visualization of runtime as it may apply to scripting languages, digital logic or circuitry. Narrowing the scope toward a feasible semester project involves selecting a domain to implement this concept. I am proposing an implementation for digital logic circuit design with the stretch goal of a finite scripting language if time permits.

Unity will be the game engine of choice. Leap Motion pinch control will be used for placing and connecting primitive elements. Leap Motion developer demos will be pilfered for example code relevant to pinch control, as well as menu systems or GUI's to aid in the selection of primitive elements during gameplay. Art assets will be found freely available or generated primarily with Maya.

3 Deliverables:

I will create a sandbox where users can place and connect various objects representing digital logic primitives from a top down "God View" perspective. I am picturing the user sitting in front of a virtual work bench, creating the circuits with hand controls and leaning in to view them close up from different angles. Leap Motion pinch control will be used to place and connect primitives. The user will be able augment inputs or switchable elements in an interactive manner and view the flow of logic through the implemented circuit. The user should be able to create functional "black boxes" by grouping together implemented circuits. While I could limit the primitives to only NAND gates and focus on this black box building, I intend to provide the user with a balanced set of primitive elements (AND, OR, NOT, NAND, Multiplexors.. and perhaps registers or and clocks etc). I want this to remain simple enough that someone with no knowledge of digital logic work through it like a graphical puzzle following a small set of rules.

If time permits I will attempt to extrapolate the interaction system to a limited set of blocks defining a scripting language. The language will be extremely basic with only a small set of conditions/branches/switches and a very limited set of action or query elements. The system will be complex enough to offer a sandbox, but simple enough to present all available commands to a user at once. To avoid variable handling and to help provide meaning to the scripting activity, a specific context will be provided where the user can see some tangible representation of output. For example the user might be tasked to create a program to control a small robot in the virtual world. Primitives corresponding to various but SIMPLE environment queries representing robot sensors/inputs could be included. For example there may be a "is robot stationary?" primitive that could be input to a conditional block that leads to a "rotate robot left 90 degrees" action block.

4 Human Factors:

The user will be seated stationary and the environment will be viewed from this stationary position (per oculus best practices), with the only movement being the hands and head. This will prevent motion sickness as there will be no acceleration mismatches (per oculus best practices). While the experience will rely on hand movements, ideally the hands can be kept low (picture work bench) to avoid complete gorilla arms. Any close ups of the game objects will require the user to lean in physically instead of zooming the camera (per oculus best practices). Code will be written with quaternions and not euler angles (per oculus best practices). The primitive elements will manifest as real world objects (per oculus best practices) rather than text or 2d graphics. Any user interfaces will likely be close to the user by being on or very near the virtual work bench. This may be closer than the 2-3m recommended by oculus best practices.

5 Milestones:

5.1 Milestone #1:

Create a parent class for all digital logic primitives and child classes for each desired type. Give each a 3D representation that indicates its type and input/output ports (simple shapes ok for now, no artwork). Research and create an intelligent circuit solver. At the very least it should be a directed graph of nodes (digital logic primitives) that determines whether or not each edge (connection) is active based on the logic rules of the primitives. Add in logic to create black boxes of circuits such that a node in a circuit solver could be an entire circuit itself. Explore the leap motion example content.

5.2 Milestone #2:

Use what was learned from the leap motion example content to implement pinch motion for selecting and placing a primitive element on the workbench. In the case of a finite set of primitives, a bin containing various primitives with distinctive appearances could be available to grab from. In the case of many primitives, an attached menu system would handle selection of elements (leap motion "arm-hud" or htc vive "tilt-brush" gui). Investigate the options of free placement versus snap grid placement. At the very least the user should be able to pinch select an element from the menu/bin and and drag it into the world such that a new instance of the primitive is placed at the location. Then implement an interactive method to generate connections. The final method will depend on the implemented art (ex: electrical sockets and wires). At this stage the user should be able to pinch select an output port of a primitive and then pinch draw to an input port of another primitive element. Provide visual and auditory feedback for highlighting and selecting any pinchable/interactive locus. Implement a functional placeholder for connections (wire or line between ports that glows/changes color when active). One should be able to move the location of primitive elements without breaking connections to input and output ports. Implement a connection break mechanic, by either pinch selecting a connection or some other recognized and intuitive gesture (scissors?).

5.3 Milestone #3:

Enhance the visualization of primitive elements and connections. This will be limited by an ability to find and generate art assets. Again, the project aim is to balance simple elements and connections a user can intuitively pick up with creative enough designs that users are interested/engaged. At the very least various primitives should have unique shapes and input/output ports should be visually connected by lines or wires which change when active (think 3D version of the slate UI used in Unreal Engine 4 Blueprints). The system should be visually pleasing. Going beyond this, ports could be electrical sockets and connections could be hefty electrical wires. Ports could be pipe fittings or valves and flow could be water in pipes. Primitives could be little mini factories or in-out machines with conveyor belts for connections. Taking the flow visualization to a metaphorical level, conveyor belts could run outputs to subsequent input ports. If I somehow manage the artistic talent, little minions could run the elements back and forth like an assembly line factory floor. This milestone should emphasize the user experience from a "God Mode" view. The primitives and connections should be detailed and active enough that the user is motivated to lean in and view them at runtime.

5.4 Milestone #4:

Enhance interactivity and visualization. Continue to tune the controls and any menu systems. Implement an interactive method for switch inputs or elements to alter flow paths during run time. Play around with physical switches the user can flip, valves they could turn, gaze targets they can fixate

upon to trigger different input values or augment the state of primitive that permit change. Continue to improve the artistic visualization of inactive vs. active (flowing) connections.

5.5 Milestone #5:

Implement the graphical scripting language described in the stretch goals of the deliverables to control a small robot in the virtual world.