VoxSim

- https://github.com/VoxML/VoxSim

Architecture

- Built on Unity Game Engine
- NLP may use 3rd-party tools
- Art and VoxML resources loaded locally or from web server
- Input to UI or over network

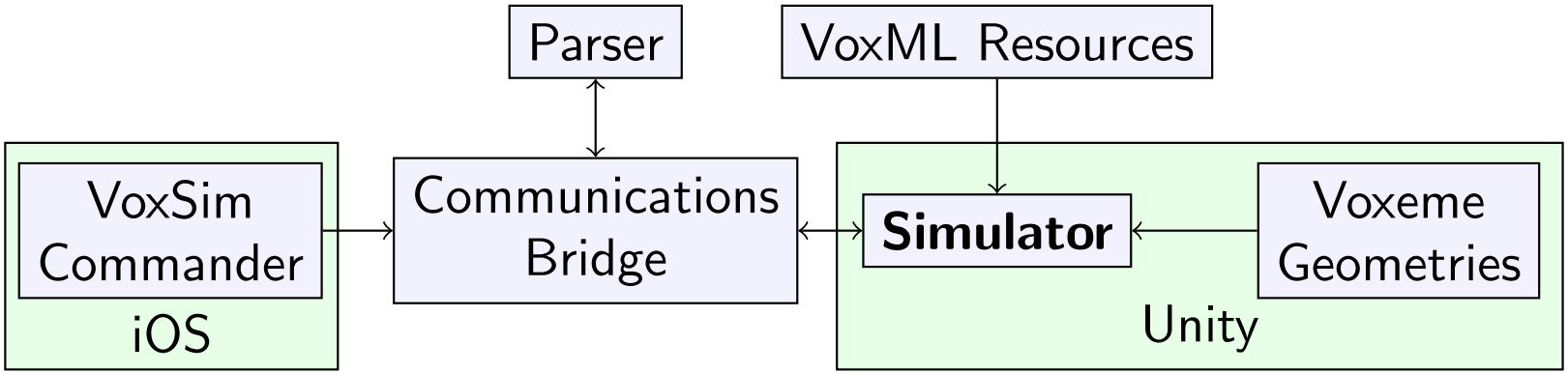
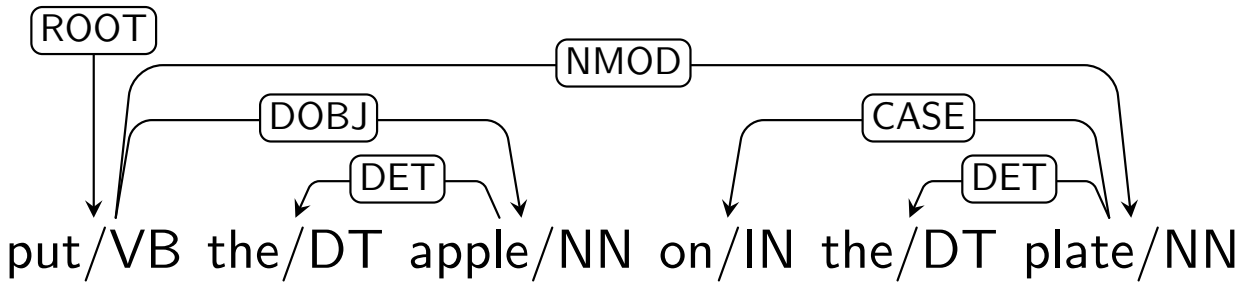


Figure: VoxSim architecture schematic

Processing Pipeline



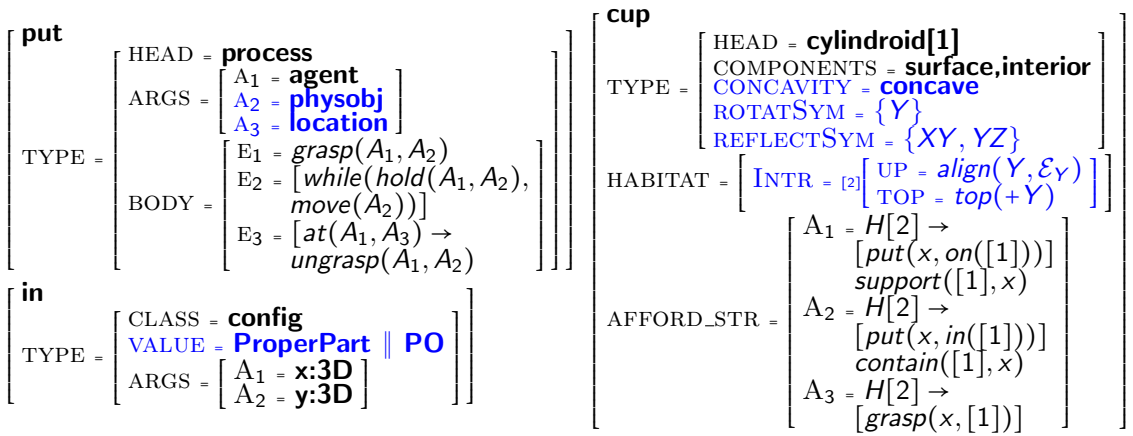
1. $p := put(a[])$	5. $nmod := on(iobj)$
2. $dobj := the(b)$	6. $iobj := the(c)$
3. $b := (apple)$	7. $c := plate$
4. a.push($dobj$)	8. a.push($nmod$)
$put(the(apple),on(the(plate)))$	

Figure: Dependency parse for *Put the apple on the plate* and transformation to predicate-logic form.

Vocabulary is restricted for this tutorial

Flow of Control

1. Input sentence
2. Generate parse
3. Compute satisfaction conditions from voxeme composition



Flow of Control

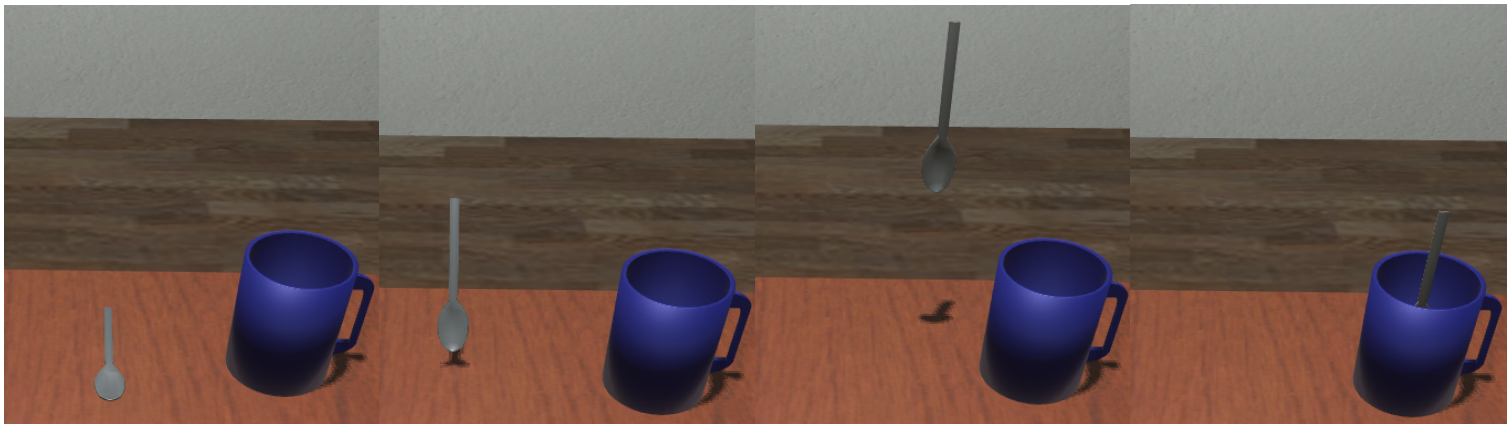


Figure: Object properties impose constraints on motion

4. Move object to target position
5. Update relationships between objects
6. Make or break parent-child rig-attachments
7. Resolve discrepancies between Unity physics bodies and voxemes

Object Model of Motion

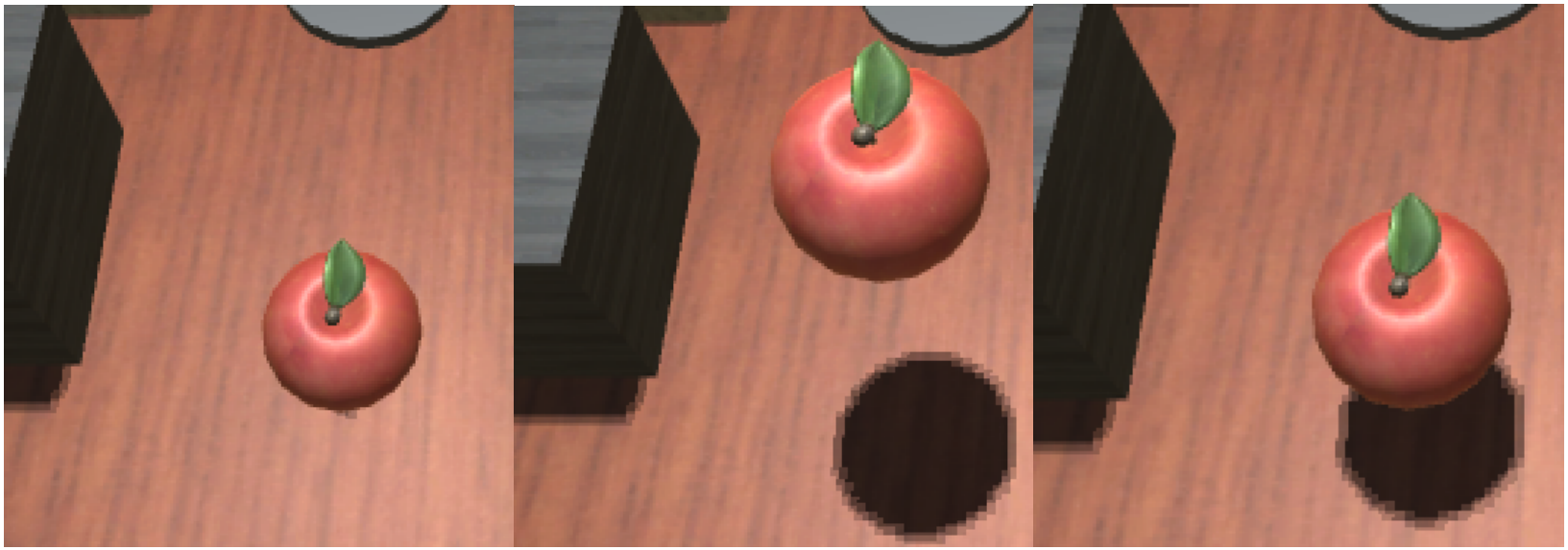


Figure: Object model of lifting and dropping an object

The **object model** of motion enacts the verbal program *only* on the objects involved, independent of any agent.

Object Model of Motion

- Requires reduction of subevent structure, informed by object affordances
  - e.g., $[[\text{PUT}]] = grasp(x, y), [while(hold(x, y), move(x, y)], at(y, z) \rightarrow ungrasp(x, y)$
  - H→[grasp(x, y)]hold(x, y)
  - H, hold(x, y)→[move(x, y)]
- Removing grasper removes "hold" condition and "ungrasp" subevent, $ungrasp(x, y) \rightarrow \neg move(x, y)$
- $at(y, z) \rightarrow \neg move(x, y) \Longrightarrow \neg at(y, z) \vee \neg move(y)$ by CNF conversion
- One condition must be true, so "while" constraint holds
- $grasp(x, y), [while(hold(x, y), move(x, y)], at(y, z) \rightarrow ungrasp(x, y) \Longrightarrow [while(\neg at(y, z), move(y))]$

Action Model of Motion

Figure: Action model of lifting and dropping an object

The **action model** factors out the objects, leaving a "pantomime" version of the event.

Action Model of Motion

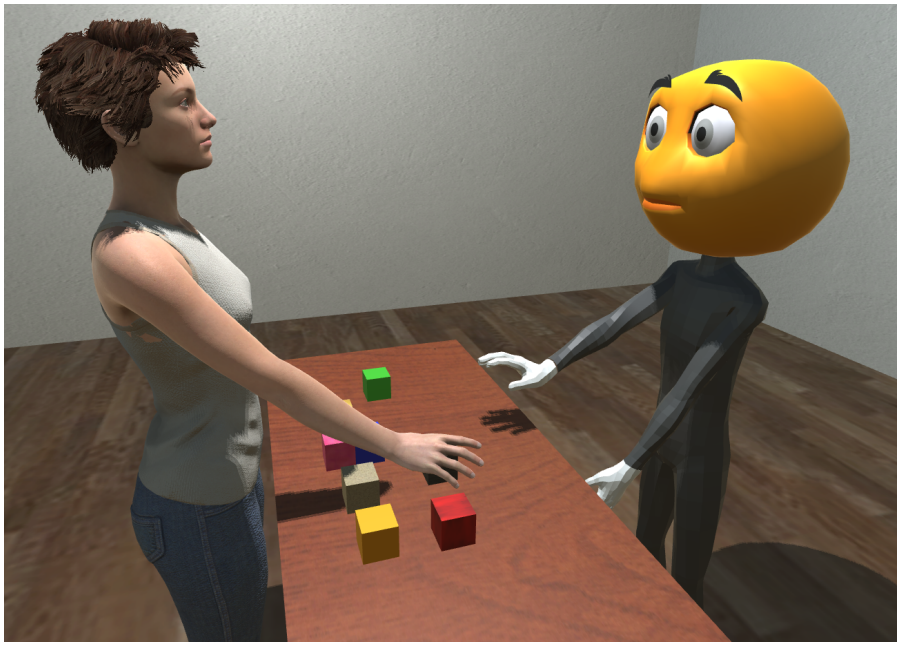


Figure: Sample gesture and object manipulation interaction

Action models can also be used to execute gestures as programs that operate over objects without affecting them.

Event Model of Motion

Figure: Event model of lifting and dropping an object

Objects can be attached to joints of an animated agent to make it move with the agent.

Event Model of Motion

- Programs enacted over the agent also affect the object
- Generalizes: trajectory from source to destination
- Specifies: how to grasp, how to calculate destination given object properties, current habitats, etc
- "Event model" = "object model" + "action model"
- Primitive behaviors (grasp, translocate, turn, etc.) can be composed into complex behaviors
  - roll, slide, flip, lean, etc.

Composition Example: LEAN (Object Model Only)

Theoretical formulation:

- Instruction: "Lean [[THEME]] on [[DEST]]"
- Goal: [[THEME]] is supported by [[DEST]] at an angle θ
  - For this example, assume $\theta = 45°$

1. Turn [[THEME]] such that major axis is θ off from +Y axis
2. Move [[THEME]] so it touches a side of [[DEST]]

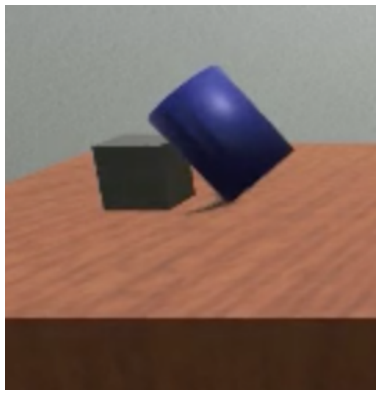


Figure: Desired goal state of "lean *x* on *y*"

Composition Example: LEAN (Object Model Only)

Operationalization:

- Instruction: "Lean [[THEME]] on [[DEST]]"
- Goal: [[THEME]] is supported by [[DEST]] at an angle θ
  - For this example, assume $\theta = 45°$
- Starting position of [[THEME]] is arbitrary
  - Not necessarily lying flat
  - Not necessarily axis-aligned
- 3D transformations take shortest path
  - Single rotation may result in unstable configuration

1. Turn [[THEME]] such that **minor axis** is $90° - \theta$ off from $+$Y axis
2. Turn [[THEME]] **about minor axis** such that major axis is θ off from $+$Y axis
3. Move [[THEME]] so it touches a side of [[DEST]]

Composition Example: LEAN (Object Model Only)

- Three types of primitive motions
  1. TURN-1: turn(x:**obj**,V_1:**axis**,$\mathcal{E}_{V_2}$:**axis**) — turn object x so that object axis V_1 is aligned with world axis V_2
  2. TURN-2: turn(x:**obj**,V_1:**axis**,$\mathcal{E}_{V_2}$:**axis**,$\mathcal{E}_{V_3}$:**axis**) — turn object x so that object axis V_1 is aligned with world axis V_2, constraining motion to around world axis V_3
  3. PUT: put(x:**obj**,y:**loc**) — put object x at location y

$$\left[\begin{array}{l}\textbf{lean} \\ \text{LEX} = \left[\begin{array}{l}\text{PRED} = \textbf{lean} \\ \text{TYPE} = \textbf{transition\_event}\end{array}\right] \\ \text{TYPE} = \left[\begin{array}{l}\text{HEAD} = \textbf{transition} \\ \text{ARGS} = \left[\begin{array}{l}\text{A}_1 = \textbf{x:agent} \\ \text{A}_2 = \textbf{y:physobj} \\ \text{A}_3 = \textbf{z:location}\end{array}\right] \\ \text{BODY} = \left[\begin{array}{l}\text{E}_1 = \mathit{grasp}(x,y) \\ \text{E}_2 = [\mathit{while}(\mathit{hold}(x,y), \mathit{turn}(x,y, \\ \quad \mathit{align}(\mathit{minor}(y), \\ \quad \mathcal{E}_Y \times (90-\theta, \mathit{about}(\mathcal{E}_{\perp Y})))))] \\ \text{E}_3 = [\mathit{while}(\mathit{hold}(x,y), \mathit{turn}(x,y, \\ \quad \mathit{align}(\mathit{major}(y), \\ \quad \mathcal{E}_Y \times (\theta, \mathit{about}(\mathcal{E}_{\perp Y}))), \\ \quad \mathit{about}(\mathit{minor}(y))))] \\ \text{E}_4 = [\mathit{while}(\mathit{hold}(x,y), \mathit{put}(x,y))] \\ \text{E}_5 = [\mathit{at}(y,z) \rightarrow \mathit{ungrasp}(x,y)]\end{array}\right]\end{array}\right]\end{array}\right]$$

Composition Example: LEAN (Object Model Only)

Result: "Lean the cup on the block"

VoxSim Summary

- Provides method for generating 3D visualizations using NL interface
- Provides platform to conduct experiments on observables of motion events
- Provides intuitive way to trace trace spatial cues and entailments through narrative
- Used to generate data on theoretical intuitions
- Enables broader study of event and motion semantics

VoxSim Summary