THE SCENE LANGUAGE: REPRESENTING SCENES WITH PROGRAMS, WORDS, AND EMBEDDINGS

Authors: Yunzhi Zhang, Zizhang Li, Matt Zhou, Shangzhe Wu, Jiajun Wu

Presenter: Hao Su

Authors

Jiajun Wu's Group

- Yunzhi Zhang
  - Ph.D. student at Stanford
- Zizhang Li
  - Ph.D. student at Stanford
- Matt Zhou
  - Currently at letta, graduated from UC Berkeley
- Shangzhe Wu
  - Postdoc at Stanford
- Jiajun Wu
  - AP at Stanford

Physical Scene Understanding

Build machines that see, reason about, and interact with the physical world.

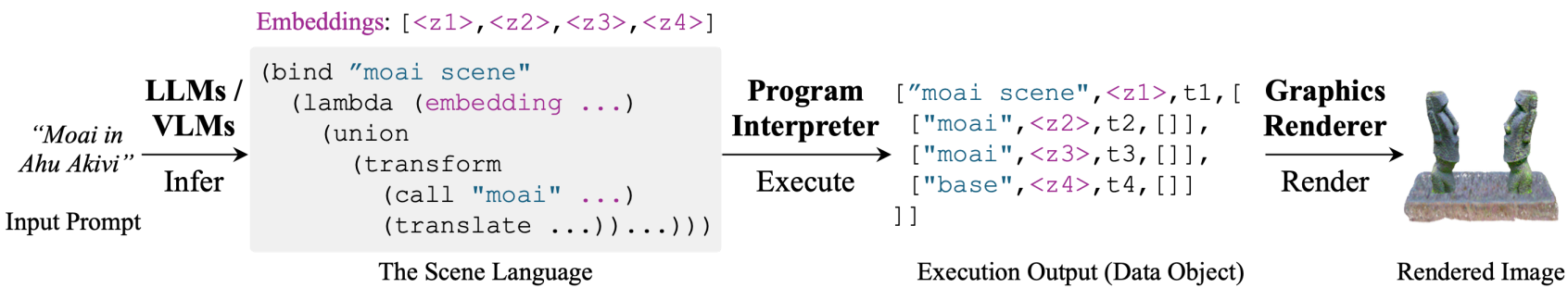
Besides learning algorithms, what are the levels of abstraction needed by AI systems in their representations, and where do they come from?

Inspirations

- nature, i.e., the physical world itself
- human cognition
- Representative projects include Galileo, MarrNet
- the Neuro-Symbolic Concept Learner
- Neural Flow Maps
- 3D-Fauna.

Main Contributions

1. A scene representation, the Scene Language, capturing structure, semantics, and identity of visual scenes using **programs**, **words**, and **embeddings**.
2. A training-free method that infers the representation from texts and/or images using pre- trained language models.
3. A generic rendering module that renders the Scene Language into an image.
4. Empirical results on text- and image-conditioned scene generation and editing tasks.



Usage

Generate a scene of Moai in Ahu Akivi, with slight variations.

Generating the Scene Language…

```
(bind "Ahu Akivi scene"
  (lambda (embedding ...)
    (define create-moai
      (lambda (i) (let ((pose ...)
         (statue (call "moai" ...)))
        (transform statue pose))))
    (define statues
      (union-loop 7 create-moai))
    (union statues ...)))
(bind "moai" ...)
...
```

Embeddings: [(<z1>, <z2>, ..., <z9>]

Generated 3D Scene (Two Views)

Keep 2 statues, facing each other.

```
... pose ...
union-loop 2
...
```

Content Editing

Change <z1> to <z1*>:

<z1*>

Style Transfer

Scene Language Overview

The Scene Language: A Domain Specific Language

Data Types

```
Word                  // Word specifying semantics
Embedding             // Embedding specifying an entity's attributes
Matrix              ::= Array[Array[Float]]  // Transformation in GA(3, ℝ)
Entity              ::= Tuple[Tuple[Word, Embedding], List[Tuple[Entity, Matrix]]]
```

Grammar

```
<START>             ::= <bind-expr>*
<bind-expr>         ::= (bind <word> <entity-func>)
<entity-func>       ::= (lambda (embedding::Embedding embedding-list::List[Embedding])
                           <sub-entities>)
<sub-entities>      ::= (union <entity-transform>*)
                      | (union-loop <loop-count> (lambda (i::Integer) <entity-transform>))
<entity-transform> ::= (transform <entity> <matrix>)
<entity>            ::= (call <word> <embedding>*)
<word>              ::  Word
<entity-func>       ::  Embedding -> List[Embedding] -> Entity
<loop-count>        ::  Integer
<matrix>            ::  Matrix
<embedding>         ::  Embedding
```

Macros

```
call                ::= (lambda (word . embedding-list)  // Return an entity from the semantic class of word
                           (cons (cons word (car embedding-list))
                              ((retrieve word) (car embedding-list) (cdr embedding-list))))
union               ::= list  // Compose transformed entities
union-loop          ::= (lambda (loop-count loop-func)  // Compose transformed entities using a for loop
                           (union (map loop-func (iota loop-count))))
transform           ::= cons  // Transform entity pose
```

Special Forms

```
(bind <word> <entity-func>)  // Defines and binds an entity function
(retrieve <word>)  // Retrieves an entity function bound to word, or (lambda (_) (list)) if such function does not exist
```

Inference via LLMs

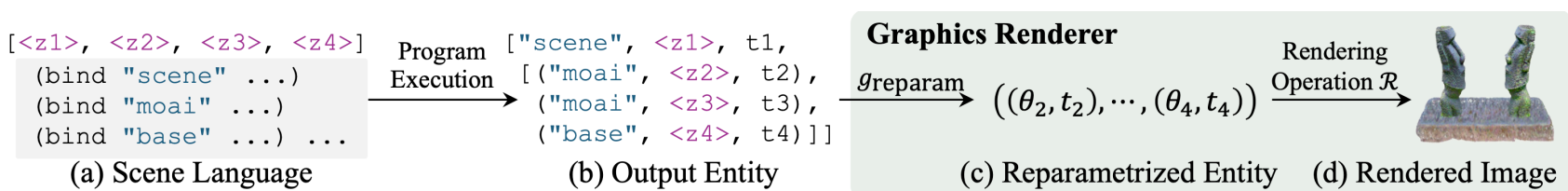
LMs are prompted to generate Python scripts.

Prompt Composition

1. A system prompt that defines the data types and the function headers of macros from the DSL, written in Python. (a Python script of helper functions converted from the DSL)
2. An example script using the helper functions.
3. The task: a scene description in texts or an image.

Rendering

A graphics renderer maps the Entity data object to its rendering parameter space and renders it into a final image.



An LM Response

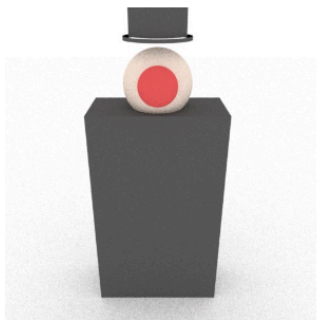
Here's a program to create a chessboard at game start using the provided helper.py:

```
from helper import *
"""
a chessboard at game start
"""

@register("chess_square")
def chess_square(color: tuple[float, float, float])
@register("chessboard")
def chessboard() -> Shape:
@register("chess_pawn")
def chess_pawn(color: tuple[float, float, float]) ->
@register("chess_rook")
def chess_rook(color: tuple[float, float, float]) ->
"""
...
"""
@register("chessboard_at_game_start")
def chessboard_at_game_start() -> Shape:
```

call `library_call('chessboard_at_game_start')` to use

Scene Editing

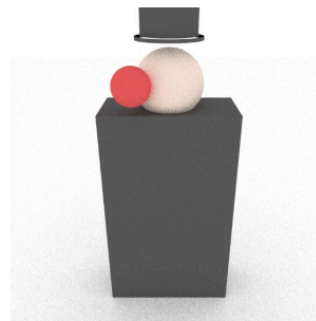



Initial (Mitsuba) Initial (Gaussians)

"Rene Magritte The Son of Man"

```
(transform apple
translate((0,
0.62, 0.15)))
        ↓
(transform apple
translate((-0.15,
0.62, 0.15)))
```

Scene Language




Edited (Mitsuba) Edited (Gaussians)

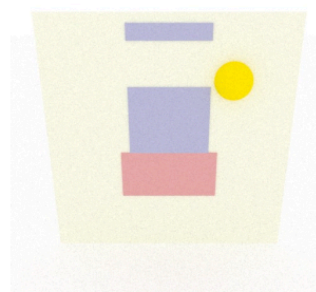
"Move the apple to the left"


Initial (Mitsuba) Initial (Gaussians)

"Paul Klee Castle and Sun"

```
((... left-tower)
(... right-tower)
(... center-tower))
        ↓
((... center-tower)
(... center-tower)
(... center-tower))
```

Scene Language




Edited (Mitsuba) Edited (Gaussians)

"Change all castles to be the middle one"

Failure Analysis

Minor variations in textual scene descriptions can lead to large quality differences in the output.

Octopus sculpture
(Sample 1, accurate)

Octopus sculpture
(Sample 2, accurate)

Octopus
(Inaccurate)

Octopus puppet
(Inaccurate)

Discussion

How does it handle primitives/searching?

1. LMs are really good at Python.
2. Primitives are given/defined by LMs.
3. Relies heavily on the capabilities of LLMs, which is very remarkable right now.

Thanks For Your Attention

Questions are welcome. References will be updated on future versions of the slides.

Powered by Slidev