

L-System Visualizer

Binyamin Brion

May 19, 2020

Contents

Introduction	3
What is an L-System?	3
Why use an L-System?	3
What is L-System Visualizer?	4
Features:	4
How Does It Works?	4
Using the L-System Visualizer	6
Program Layout	6
Creating a Project	7
Creating a Script	7
Adding a Variable	7
Adding a Constant	8
Adding Rules	9
Running a Script	10
Saving Execution Results	11
Interacting With the Rendered Result	11
Adding to the Rendered Result	12
Open / Saving the Project	13

Introduction

What is an L-System?

An L-System is a way of specifying geometrical structures through the use of a formal grammar. This system consists of the following components:

- Variables: a set of symbols that can be replaced with some other tokens
- Constants: a set of symbols which cannot be replaced with other tokens
- Axiom: The starting token of the system, which is either a variable or a constant
- Rules: define ways to replace variables with other variables and constants

Together, these components can be used to generate geometrical structures in a recursive manner.

Why use an L-System?

Using an L-System enables the generation of many different variations of geometrical structures without having to manually come up with all of those variations, nor actually producing the rendering data to render those structures.

L-Systems provide a way of automating this potentially labour-intensive process. The rules for generating a structure only have to be specified once, and from that, many different variations of geometrical structures can be created. Combined with being able to view the rendered result in a 3D environment further speeds up the process of viewing different structures by not having to not only think of a potential structure, but also model it as well.

Additionally, L-Systems can model many different pieces of nature commonly seen. Many things that immediately come to mind when one thinks of nature- trees, leaves, flowers, etc. - can be specified and created through the use of L-Systems. Combined with the randomizing of the results of an L-System additionally serves to produce more realistic results. Not only can patterns be imitated, such as the patterns found on certain flowers, it can be insured that not all geometrical structures are exactly the same as each other, which immediately ruins any sense of models being realistic.

These above advantages make L-Systems a useful tool when it comes to creating 3D environments to ensure that the world is realistic, while cutting down on the labour involved in generating the assets in that environment. Care must be placed to generate a desired geometrical model, but once that is done, the results can be many, and fit the appearance required for the program at hand.

What is L-System Visualizer?

An aid to using L-System. It provides the means to specify an L-System components in a GUI, easing the complexity of specifying a script for an L-System. Additionally, the ability to view the result of the L-System- both the tokens generated as well as the rendered result- further simplifies the use of L-System by giving visual results of the system. Lastly, being able to save and load the scripts created means that once an ideal result is generated, it can be kept for future reference.

Features:

- Specify required components of an L-System through a GUI interface
- Execute the L-Script up to a specified recursion depth
- Get error messages related to the execution of the L-Script
- View rendered result of the L-System (with textures) by specifying models to use for rendering
- Execute the L-Script as a stochastic grammar, with the ability to save a particular execution result
- Add specific instances of models used in the render in addition to what was generated by the L-Script
- Save a project, and reopen it later to view it

How Does It Works?

A brief overview of how the rendered result is generated is required to know how to use the program. Thus a summary is given below.

In a program, there are variables. With each variable is an associated model- this a model generated in a modelling program, such as a cube, a tree, or a sphere. Anything works, as long as it is in the WaveFront format.

There are also constants specifying how to manipulate the generated models, and rules stating the order of applying variables and transformations.

For example, consider the following below:

Variables: Name- Cube, Associated Model- Cube.obj

Constants:

- Left (Move to the negative one world unit). Push the stack.
- Right (Move to the positive one world unit). Pop the stack

Rules:

- Axiom: Cube
- [Rule 1] Start: Cube, Predecessor: Cube Left, Probability of occurring: 50%, Stack Operation: Pop
- [Rule 2] Start: Cube, Predecessor: Cube Right Cube, Probability of occurring 50%, Stack Operation: Push

Suppose the resulting scripts were executed up to a recursion depth of three; the resulting result might occur.

Rule Taken	Stored Tokens
Rule 1	Cube Left
Rule 1	Cube Left Left
Rule 2	Cube Right Cube Left Left

To interpret the result, the last recursion depth will be read left to right and the appropriate transformations done to the next drawn model when encountering a constant; if a variable is encountered, the associated model will be drawn with the sum of the transformations up to that point in time. Continuing with the above example, the following will be done on the last recursion depth result:

Current Token	Stack	Resulting Action
Cube Right Cube Left Left		Draw Cube at origin
Cube Right Cube Left Left	Right	Draw next model translated one world unit in the positive direction
Cube Right Cube Left Left	Right	Draw Cube translated one world unit to the right
Cube Right Cube Left Left	<i>Left is a popping action</i>	Drawing next model at origin
Cube Right Cube Left Left	<i>Stack is already empty</i>	Drawing next model at origin

As seen above, if the stack is already empty, then a popping operation has no effect. If the stack is not empty by the end of the last token, then the result executed was not balanced- this not necessarily an error however, if the resulting model is as desired.

It should be noted that the use of the stack is not mandatory; one could get have imitate the stack by specifying a constant with a pushing action that is the opposite of the most recent element in the stack. This may be less complicated and worth considering.

Using the L-System Visualizer

Program Layout

Upon launching the program, the following is seen:

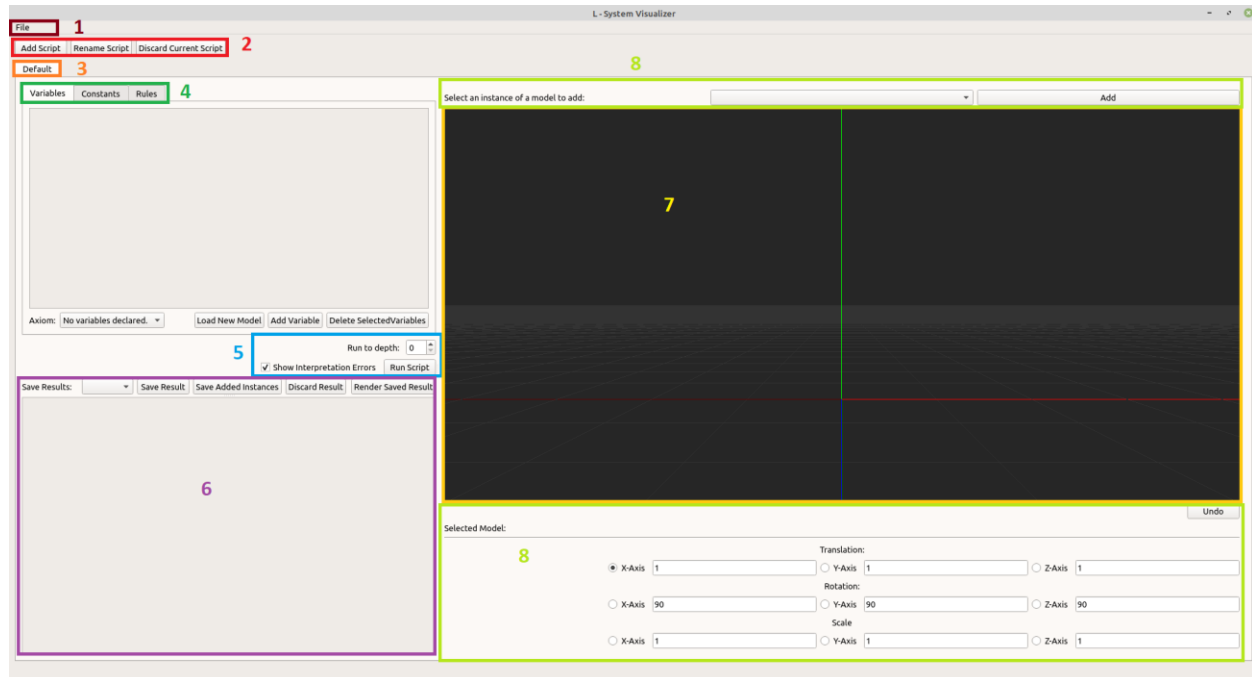


Figure 1: An overview of the program upon launching of the program.

The areas within the program are as follows:

1. Menu bar: Contains the actions to create, save and open projects.
2. Script action bar: holds the actions to modify scripts
3. Script tabs: holds the different scripts that have been created
4. Script information tabs and respective information area: allows viewing different components of the current script
5. Execution Information: provides information on how far recursively to run the result, as well as to actually execute the script
6. Output area: shows the token output of running the script, with the actions to save and discard specific results
7. Render area: shows the rendered result of the execution
8. User Transformation area: Provides the option of manually adding instances of models loaded in the program as well as to apply transformations to those models.

The rest of this document provides a walkthrough of the above mentioned areas of the program.

Creating a Project

By default, when the program is opened it will have an empty project created. Work can be started from this empty project. Otherwise, the following steps can be used to create a new project:

- Click on “File” in the menu bar *
- Click on “New Project” in the drop-down menu. A warning will appear asking if any unsaved changes should be saved. **This always appears- even if no modifications have been made since the last change.**
- A new empty project will be created

* The menu bar may be hidden to conserve vertical space- pressing the key “Alt” will toggle the menu bar’s visibility.

Creating a Script

To create a script, click the “Add Script” button in the Script Action bar (Area 2 in Figure...). When this button is clicked, a pop-up asking for the name of the script will appear. The name of the script must be unique- if it is not, the line edit where the script name is given will appear red to show an error, and the “Ok” button will be disabled.

The other buttons in the area do as one would expect- the Rename Script button renames a script, using the same dialog as when creating a script (note: renaming a script will free the old name for later use). The Discard Current Script deletes the currently opened script. A warning will be given before deleting the script, asking if this action should really be performed.

Adding a Variable

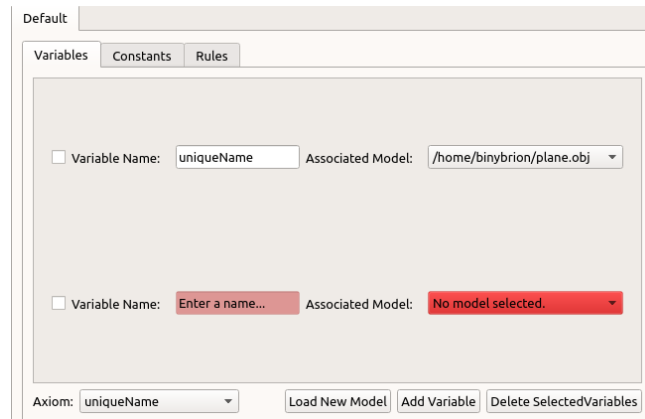
To add a variable, ensure that in the opened script, the “Variables” tab is opened. Add a variable by clicking the Add Variable button. A new variable entry will appear. The name field will red, as will the associated model combo box. This indicates an error. To correct these, enter a unique name variable name (not used by any other variable or constant), and a valid associated model.

To load an associated model, click the Load New Model button. This will open a dialog from which a model can be loaded into the program. Select an OBJ file containing a WaveFront file. Afterwards, that model can be selected in the associated combo box.

Make sure to select an axiom for the script- it is the starting variable for the script.

The check box beside the variable name indicates whether the variable entry is selected. When the Delete Selected Variables Button is clicked, all selected variable entry will be deleted.

An example of creating variable entries can be seen below:



The screenshot shows a software interface with a 'Variables' tab. It contains two variable entries. The top entry is selected (checkbox checked) and has the name 'uniqueName' and the associated model '/home/binybrion/plane.obj'. The bottom entry is not selected (checkbox unchecked) and has the name 'Enter a name...' (highlighted in red) and the associated model 'No model selected.' (highlighted in red). At the bottom of the interface, there is an 'Axiom' dropdown menu set to 'uniqueName', and three buttons: 'Load New Model', 'Add Variable', and 'Delete SelectedVariables'.

Figure 2: Two variable entries- one with valid information (top), And another one with erroneous information (bottom).

Adding a Constant

To add a constant, ensure that in the opened script, the “Constants” tab is opened. Add a constant by clicking the Add Constant button. A new constant entry will appear. Enter a unique name, not used by any other variable or constant.

A constant can be one of the following:

- Translation Constant: applies a translation to a model
- Rotation Constant: applies a rotation to a model

The type of transformation the constant applies can be toggled through the appropriate radio buttons. Make sure to specify the appropriate information depending on the type of transformation the constant is. Any errors will be highlighted in red in the appropriate data fields.

Each model can either push or pop the stack. This can be specified through the appropriate combo box.

Beside the constant name is a check box. Toggling this check box will hide the information part of the constant if it is unchecked, and will show the information part if it is checked.

To delete a constant, mark it for deletion by checking the check box at the bottom of the information part, and click the Delete Constant button.

An example of adding constants can be seen below:

The screenshot shows a software interface with three tabs: 'Variables', 'Constants', and 'Rules'. The 'Constants' tab is active. At the top, there is a 'Default' label and a text input field. Below the tabs, there are three radio buttons: 'constantName' (checked), 'Rotation Constant', and 'Translation Constant'. Under 'Translation Constant', there is a 'Translation: Stack Operation' dropdown menu set to 'Pop'. Below this are three input fields for 'X', 'Y', and 'Z', each containing the value '1'. Under 'Rotation Constant', there is a 'Rotation: Stack Operation' dropdown menu set to 'Pop'. Below this is an 'Angle (Degrees)' input field, which is empty. At the bottom, there is a 'Mark for deletion' checkbox, which is unchecked. At the bottom right, there are two buttons: 'Add Constant' and 'Delete Constant(s)'.

Figure 3: Creating a constant with a valid name that represents a Translation with valid information and a Push stack operation (Rotation data is invalid, but since the constant is a Translation, it does not matter)

Adding Rules

To add a rule, ensure that in the opened script, the “Rule” tab is selected. Add a rule by clicking the “Add Rule” button. This will create a new Rule Entry. Going from left to right in the new entry, the items are as follows:

- **Check Box:** Toggling this will either hide or show parts of the Rule, depending on the state of the check box after toggling it.
- **Name field:** A name for the rule entry can be specified. This is not required and is for personal preference only.
- **Mark for delete check box:** if this check box is checked, then the rule will be deleted if the Delete Rule button is clicked.
- **Starting Rule:** Specifies the starting variable for the script. It will only contain valid variables (those with unique names and an associated model).

- Click to add production check box. Allows a token to be added to the resulting string of tokens that the starting token is replaced with. It will contain only valid variables and constants (meaning all names are unique and data fields contain no errors).
- Probability spin box: specify the probability that this rule will be executed; this 100% probability is shared between all of the rules with the same starting token. As this value goes up, the available probability for other rules with the same starting token goes down. The amount of available probability is stated beside the Add Rule button.

To undo an added production, click the Undo button. If a rule has a probability of 0%, then it is not considered for execution of the script.

A view of adding rule entries is seen below:

Figure 4: Adding a rule with a starting rule and an ending productions, with a 100% probability of occurring

Running a Script

With rules specified, the script can now be executed. To do so, specify the recursion depth level in the appropriate spin box. Afterwards, click the Run Script button. The checkbox beside the Run Script button can be toggled to display any execution errors or to hide those errors in the output area.

After executing a script, the rendered result will appear in the render area of the program. Additionally, the resulting tokens generated will be displayed in the output errors.

Figure 5: Executing the script to a recursion depth level of three, and showing all errors that occurred during execution

Saving Execution Results

Since rules may be executed with a probability of occurring, there is a possibility of no two execution runs giving the same execution result. Thus, if there is a result that should be saved for future use, click the Save Result button. A pop-up dialog to specify the name of the result will be given; this name must be unique to all other saved results.

To re-render a saved result, open it by selecting the saved result from the Save Results combo box. Afterwards, click the Render Saved Result button.

To delete a saved result, open it using the steps outlined above, and click the Discard Result button.

If a saved result was re-rendered and the user has added instances of models, click the Save Added Instances button to ensure those instances are kept in the saved result; otherwise those instances will be lost when the script is closed (such as when closing the program).

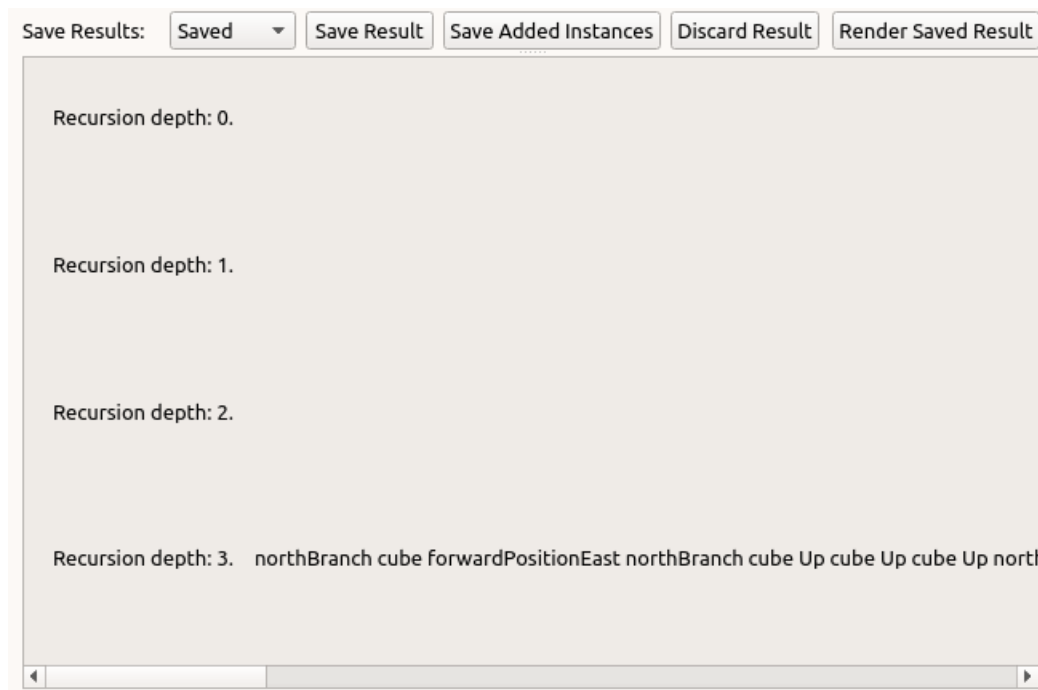


Figure 6: Viewing the execution result tokens, as well as displaying the various buttons mentioned earlier

Interacting With the Rendered Result

When a script is executed, the resulting model will be rendered in the render area of the program. The model can be viewed by moving the camera around much like one would in a game:

- To move around, use the WASD keys:
 - W: move forward
 - A: move left
 - S: move backwards
 - D: move right
- To rotate: hold the down the scroll wheel button and move the mouse around.

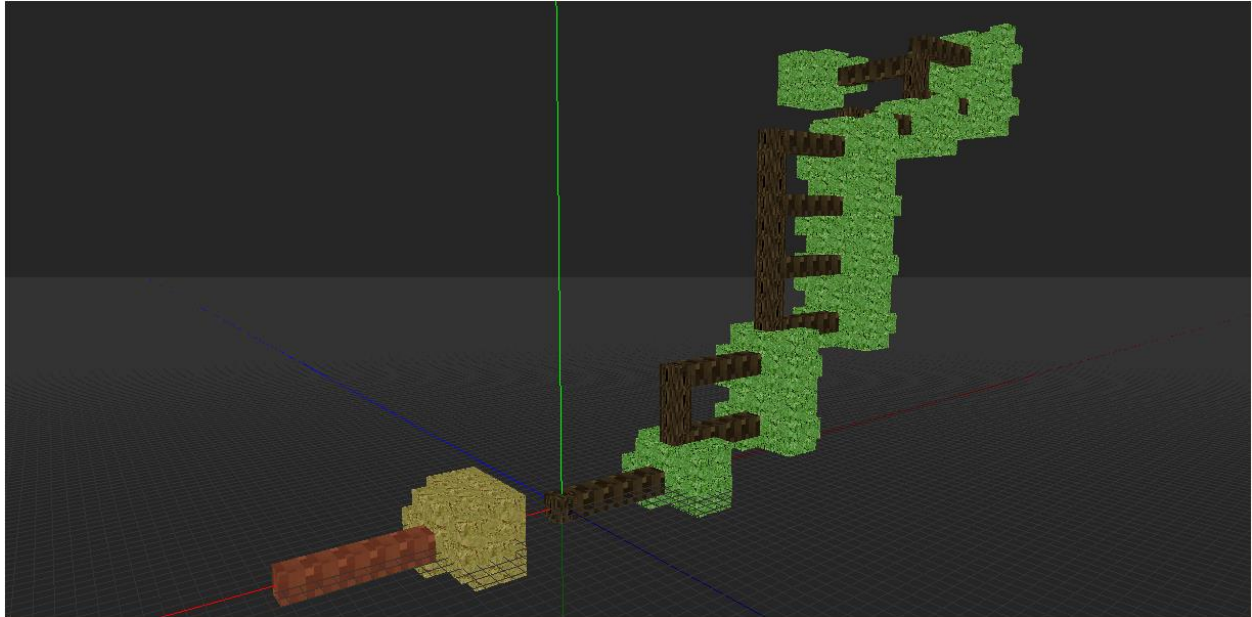


Figure 7: A rendered result with a user added instance that is selected, highlighted orange.

Adding to the Rendered Result

Once a script is executed and the result is saved, instances of loaded associated models can be added to the render scene. To do so, click the associated model name in the combo box associated with the text “Select an instance of a model to add” and click the Add button. An instance of that model will appear in the origin of the world.

To modify an added instance, it must first be selected. To select it, click on it in the rendered scene. To unselect it, click it again. The shift key can be held down to select several instances at once, regardless of what model the instance is. **By default a newly added instance is selected.** This makes sure that even if the instance is not visible when added by being obscured by a different object, it can still be interacted with. Two things to note: when an instance is selected, it changes colour, and the instances of models added by the execution of the script cannot be selected.

Operations that can be done on selected objects:

- Translation
- Rotation
- Scale

To perform one of these actions, click the appropriate radio button in the selected model area. Ensure that a valid number is in the respective line field as well. To perform the positive version of the selected action, click the Right arrow key; the left arrow key performs the negative action. For example, if the X-Axis radio button is clicked under the translation section, and the line edit value is one, then clicking the Right arrow key will move the selected models one positive world unit, while clicking the Left arrow key will move the selected models one negative world unit.

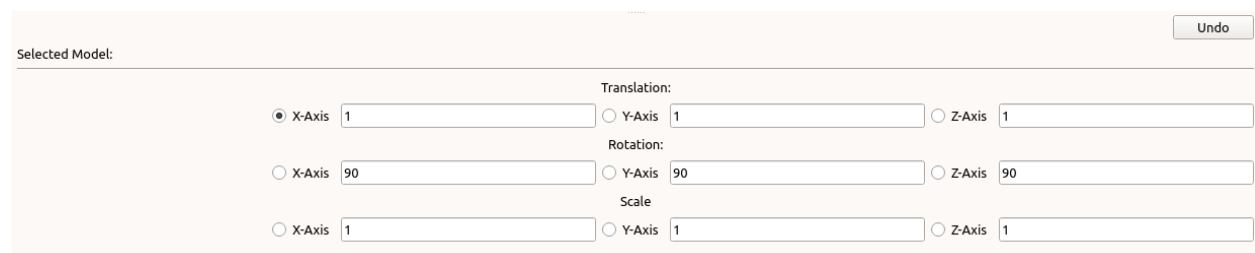
Only one transformation can be done at a time.

To delete selected models, press the delete key.

To undo actions, press the 'Z' key, or click the Undo button right underneath the render area.

Note: the render area must be in focus when pressing the arrow keys. To ensure that this is the case, move the cursor into the render area.

To ensure that all instances that have been added, as well as any that were changed are saved, click the Save Added Instances button located right above the output section of the program.



The screenshot shows a user interface for transforming selected model instances. At the top right is an "Undo" button. Below it, the text "Selected Model:" is followed by a horizontal line. Under this line, there are three sections: "Translation:", "Rotation:", and "Scale:". Each section contains three radio buttons labeled "X-Axis", "Y-Axis", and "Z-Axis", each followed by a text input field. In the "Translation" section, the "X-Axis" radio button is selected, and all three input fields contain the value "1". In the "Rotation" section, all three input fields contain the value "90". In the "Scale" section, all three input fields contain the value "1".

Figure 8: A view of the options available to transform selected user added model instances.

Open / Saving the Project

To save a new copy of the project, go to the menu bar, and click "Save As". A file dialog will appear asking where to save the project. To save an existing version, click "Save".

To open a project, go to the menu bar, and click "Open". A file dialog will appear asking what project file to open. The file extension of project files is ".L-Vis".

