



logram Documentation

MESH Consultants Inc
December 12, 2016

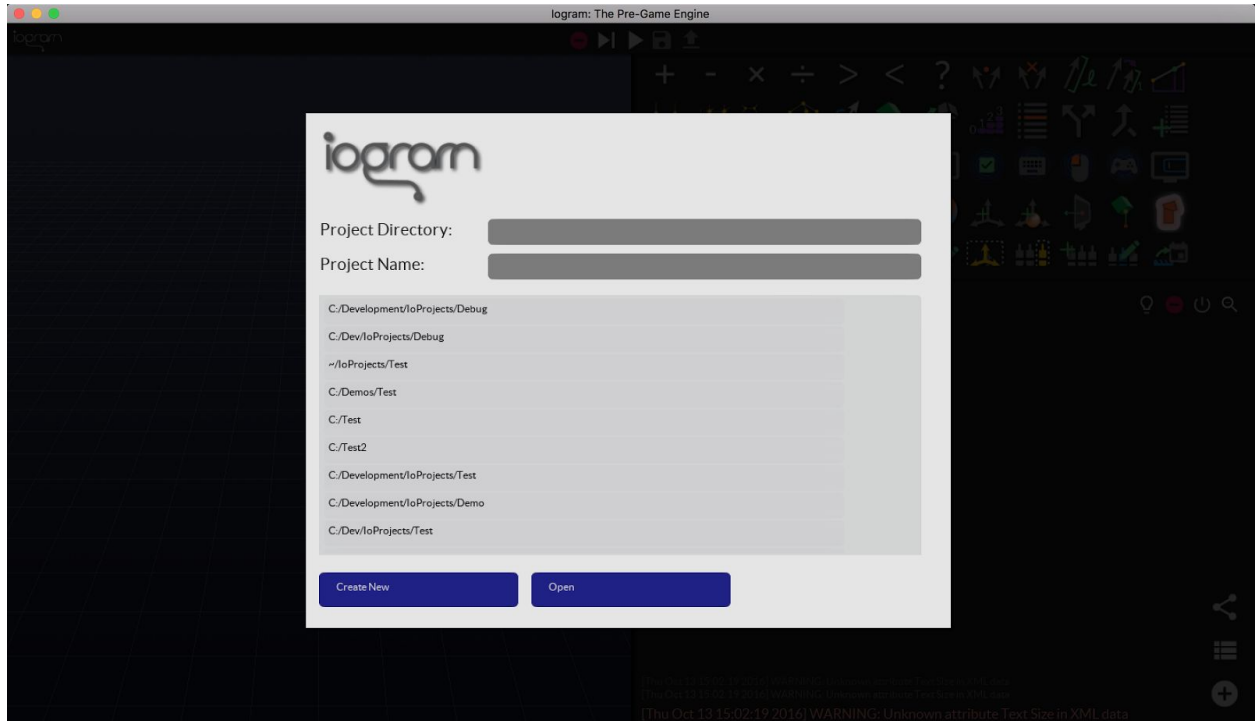
Introduction

logram is a software product that accelerates and unifies the development of 3D models, apps, and games. logram lets you easily create and explore thousands of prototypes during the early phases of your digital design project. It enables the tools that create these prototypes to evolve naturally with the project, potentially even maturing into robust products themselves!

Getting started

Launching logram

- 1) Launch logram by double clicking on the icon in your OS. You are prompted to create a new project or load from a previous project.

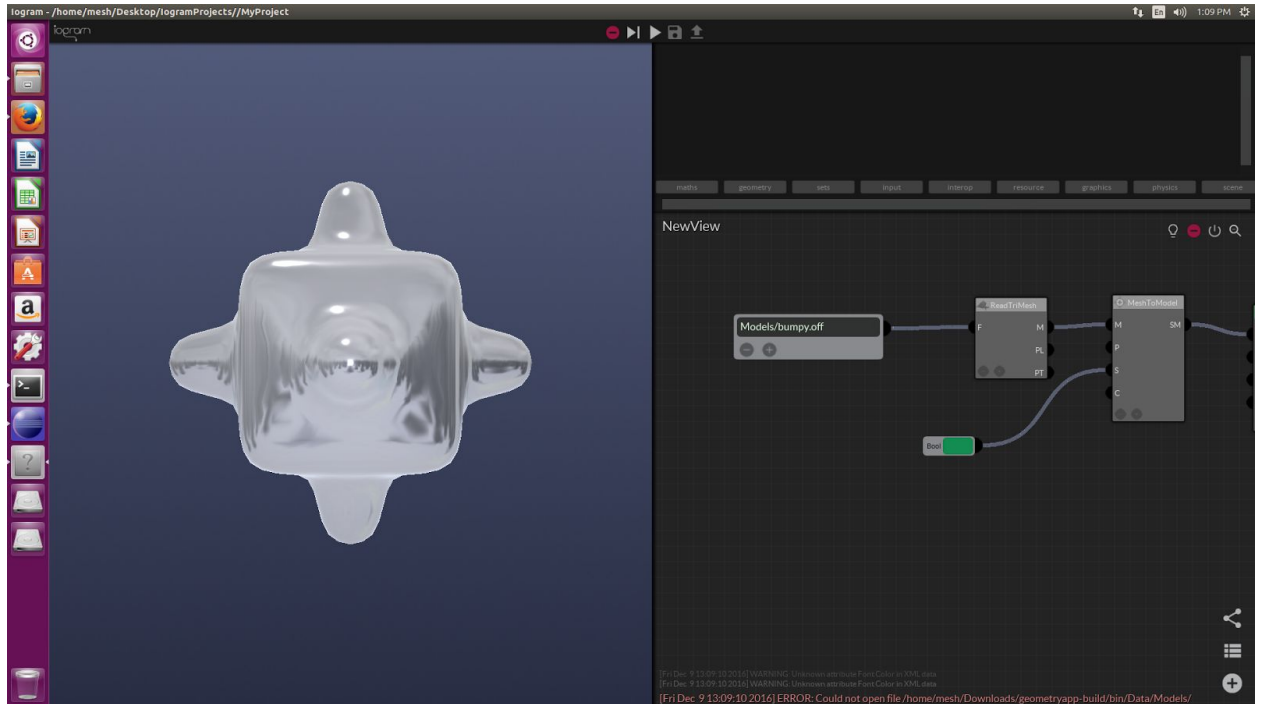


Create New Project: Enter your desired Project Directory and Project Name.

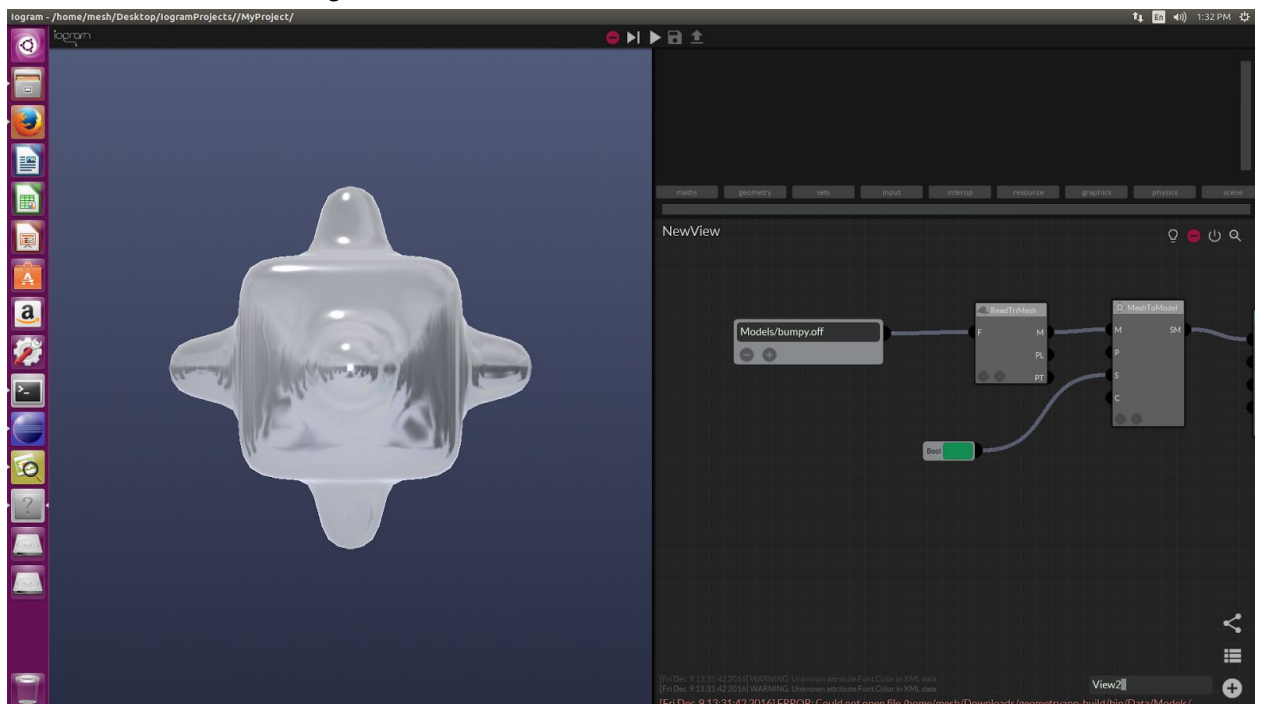
Please specify a full path to where you'd like the new project directory to be created. For example "C:/logram_Projects/First_Project".

Open Existing Project: Click on the relevant project from the previous projects list, or enter the project directory.

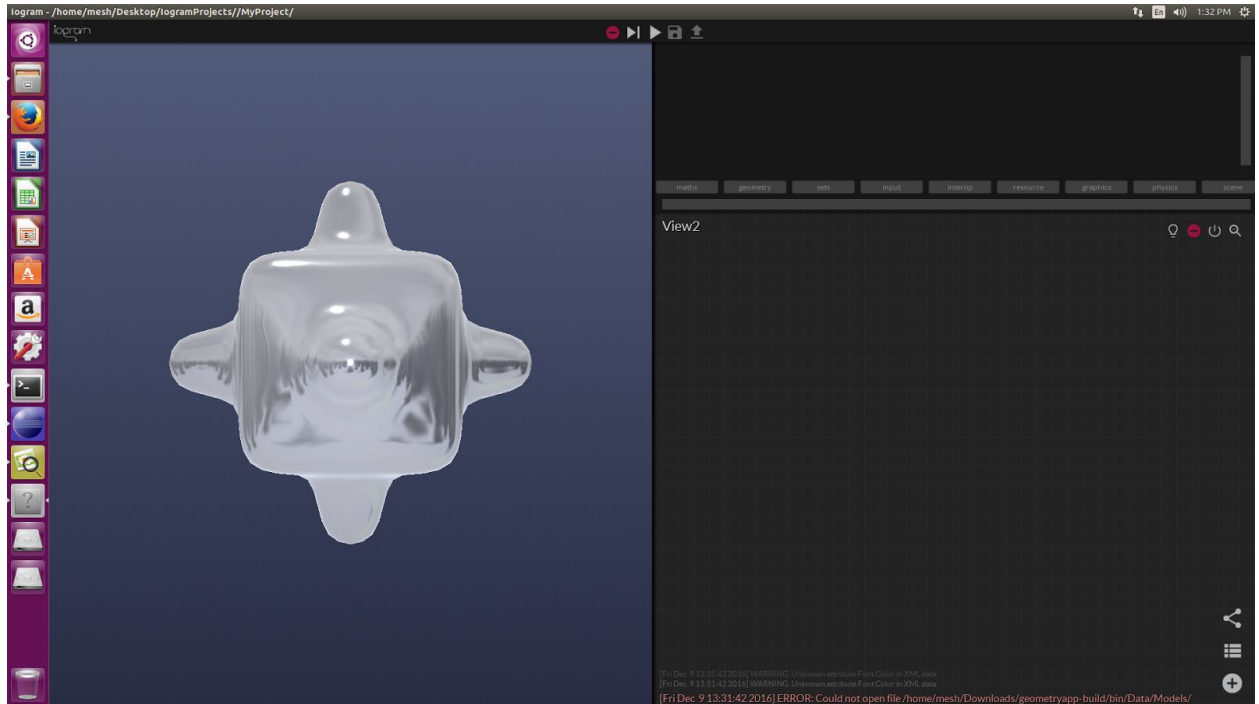
- 2) You will now see the main iogram editor. If you are starting a new project, you will see the default project, which causes a bumpy cube to appear in the scene



Components in iogram live on a workspace called a "View". In our project, the default view is called "NewView". We will add a second view called "View2". To do this, find the '+' button at the bottom right of the screen, and enter the text "View2".



Hit enter and a new view named "View2" will appear.

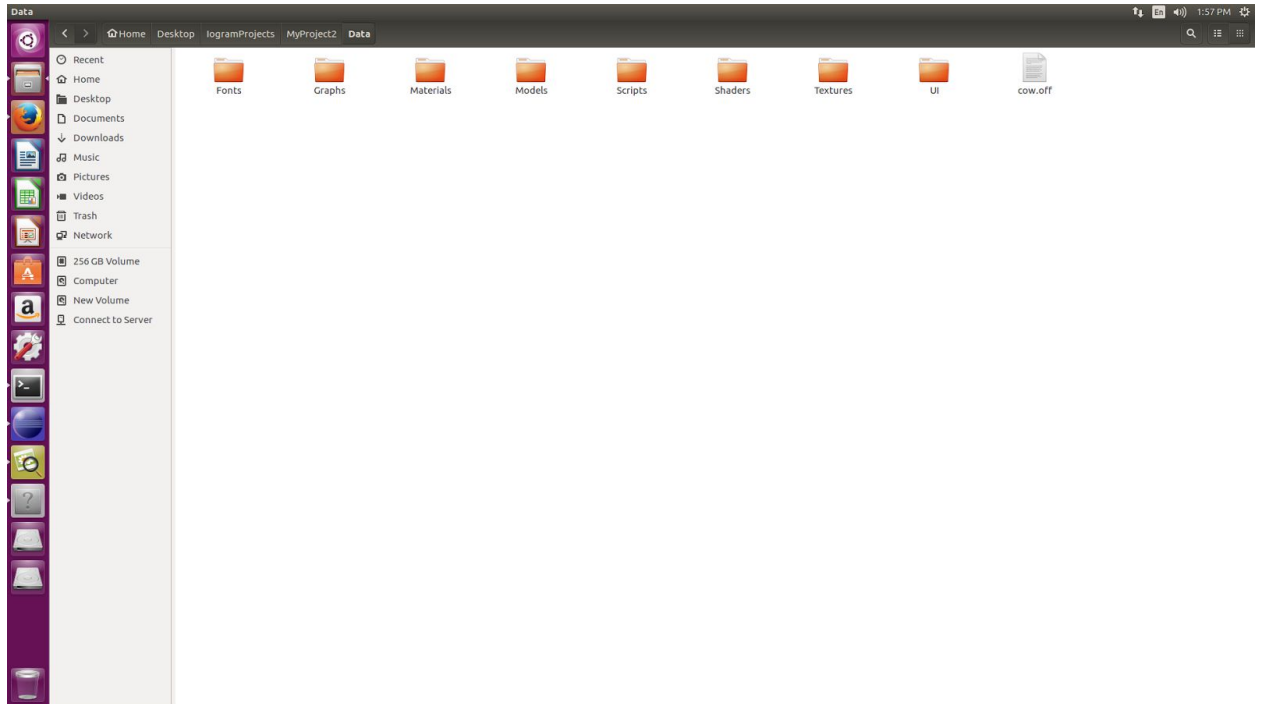


View2 is now the active view, but notice that the bumpy cube is still present in the scene! That is because the components in the original view are still part of the total component graph of the project. Try switching back and forth between NewView and View2 a few times using the "View Manager" button immediately above the '+'.

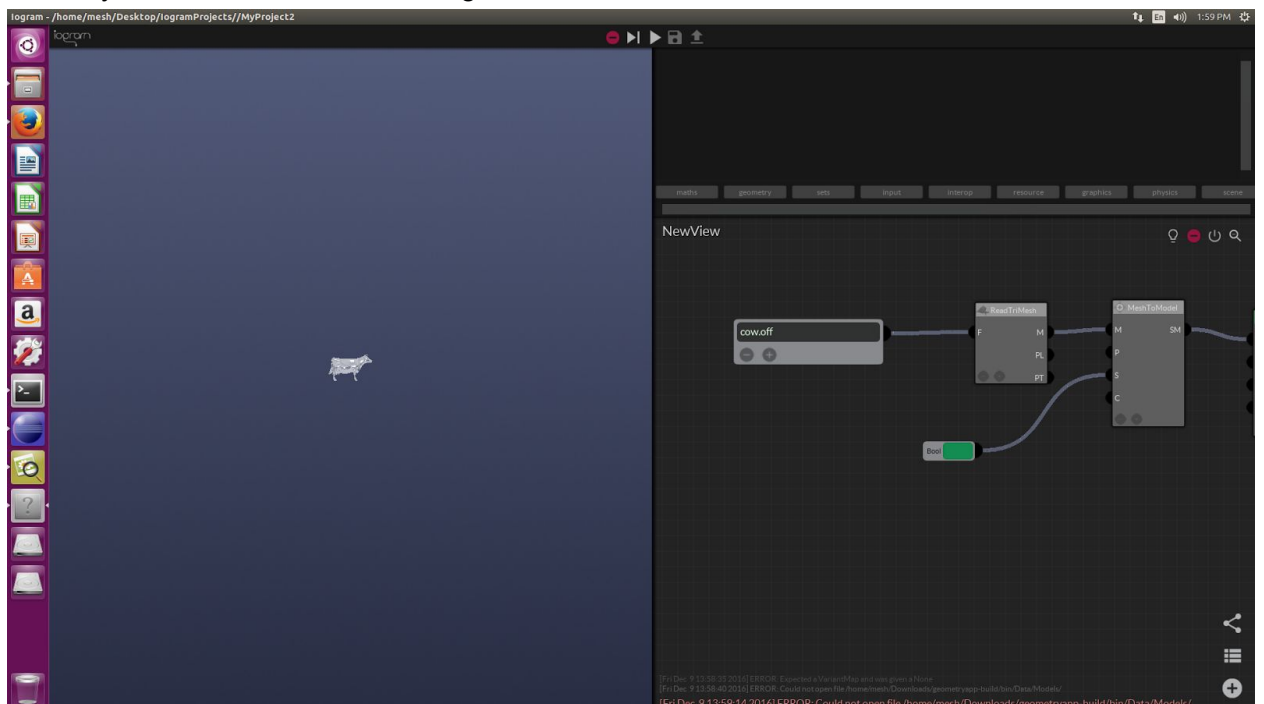
- 3) You are ready to start creating, loading, modifying, parameterizing, and exporting! Drag a component onto the canvas to get started. Perhaps start with a mesh cube primitive. You can also use the "ReadTriangleMesh" component to import your own geometry from obj, off, dxf, stl and others (see details below).

Example: Create a Basic App

- 1) Start by creating a new project, following the directions in "Getting Started" above. If you do not have another model lying around, keep using the default model "bumpy.off". We happen to have a model "cow.off", and it is instructive to see how to import resources into your project.
- 2) Copy the file "cow.off" into the Data/ folder that resides in your project's top level directory.

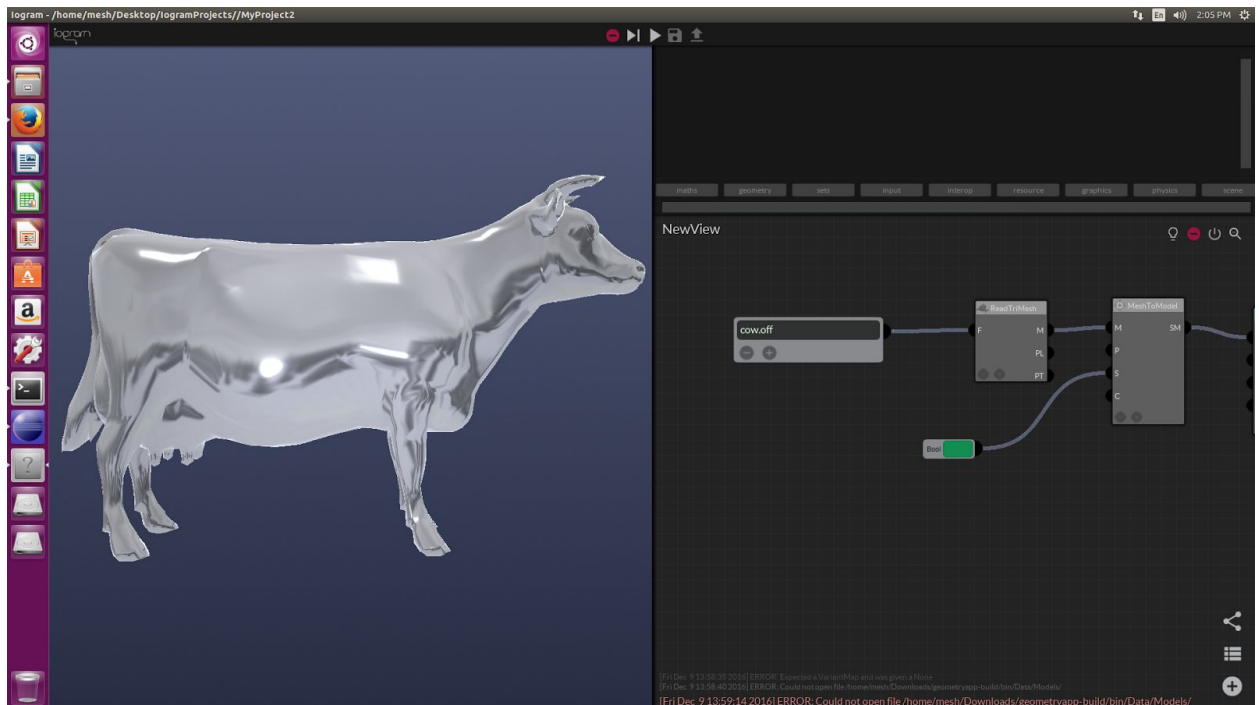


Now find the text "Models/bumpy.off" in the Input_Panel and replace it with simply "cow.off". logram will find the new model because you have copied it into the Data/ directory. You should see something like this on screen:

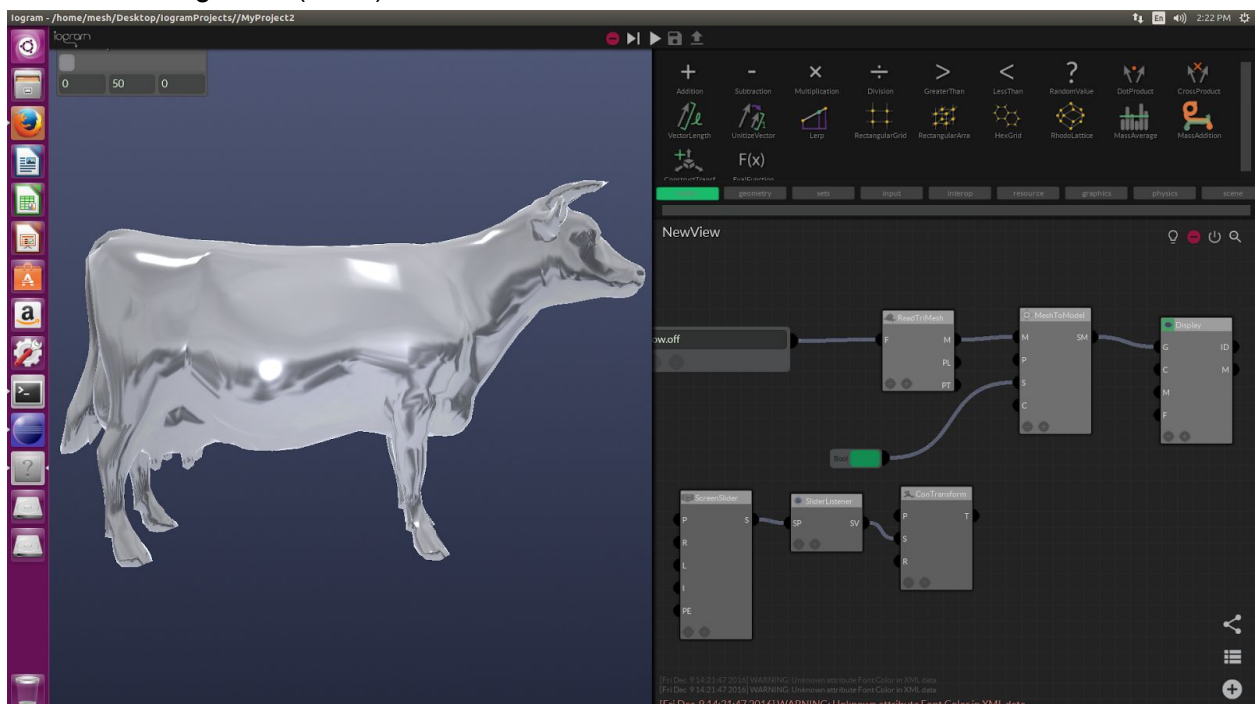


The cow is pretty small! Try zooming in and out using Ctrl + "Right mouse click and

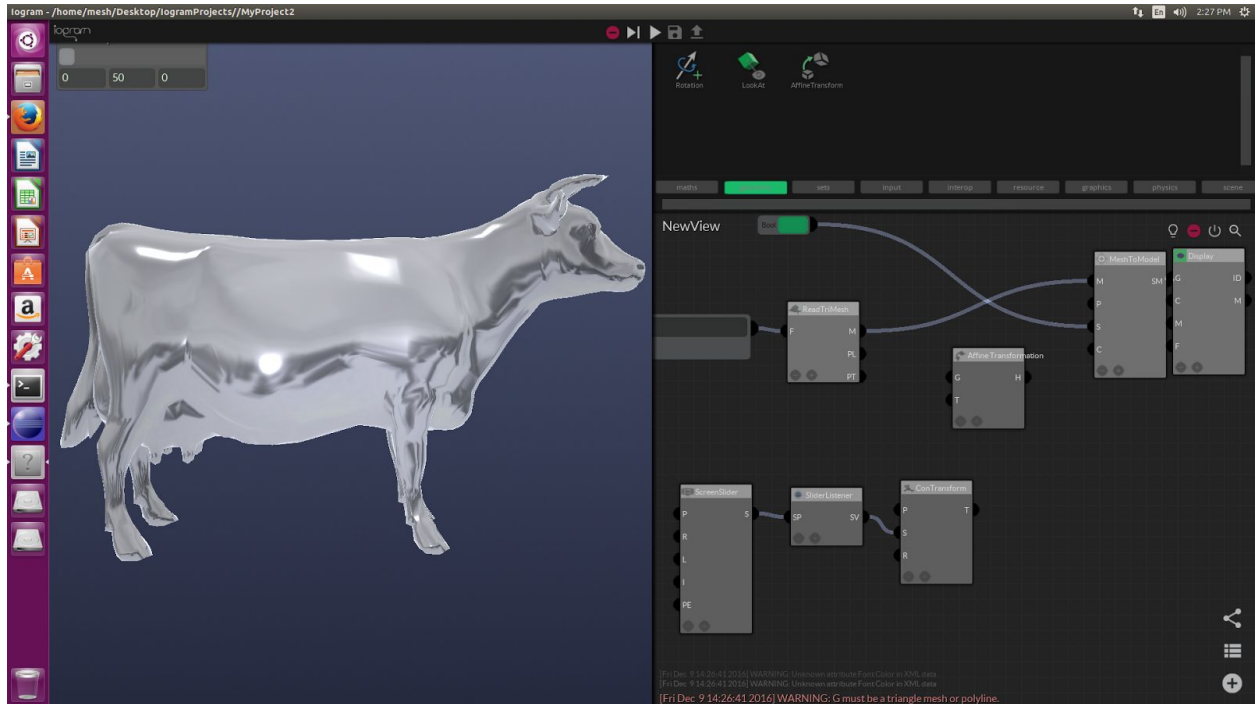
drag" until you are happy with the location of your model.



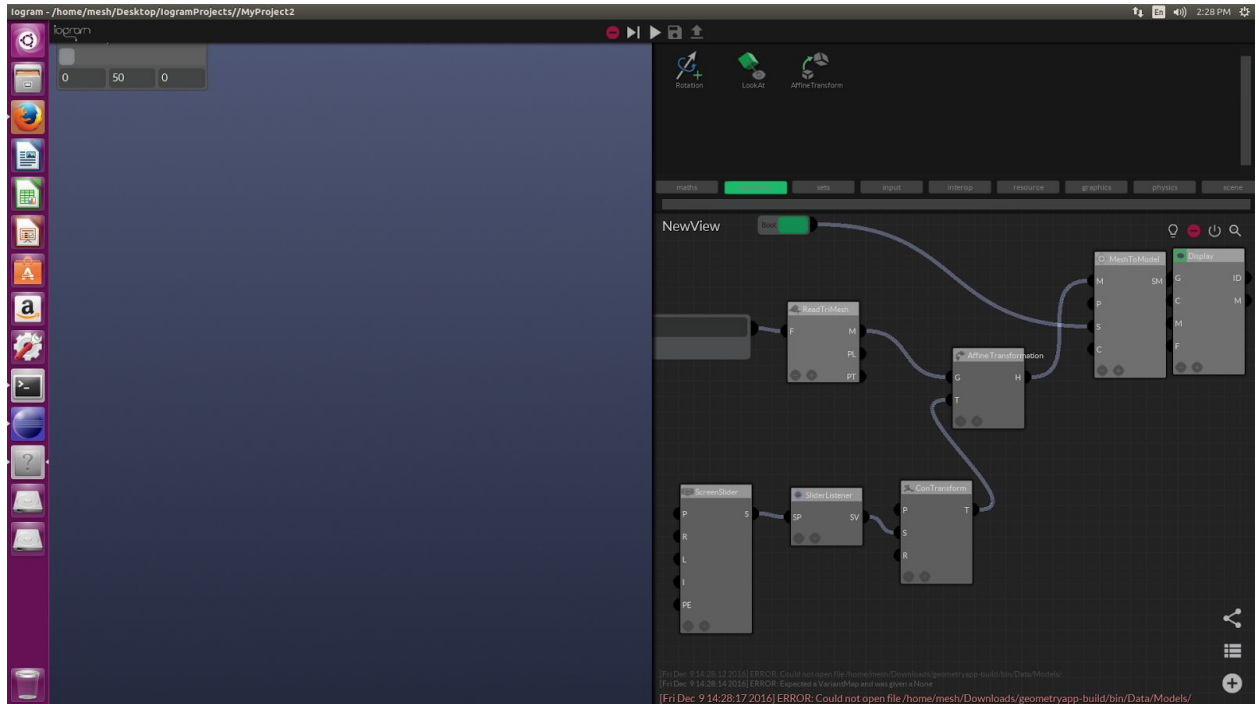
- 3) Zooming in and out has not changed the size of the model, but only our view of it. In this step we'll add a basic UI element to control the scale of the model. First, we need three components. From the *input* tab, drag the **ScreenSlider** and **SliderListener** components onto the canvas. From the *math* tab, find the **ConstructTransform** Component. Hook up the components as shown, with the output from the screen slider listener entering the **S** (scale) in construct transform.



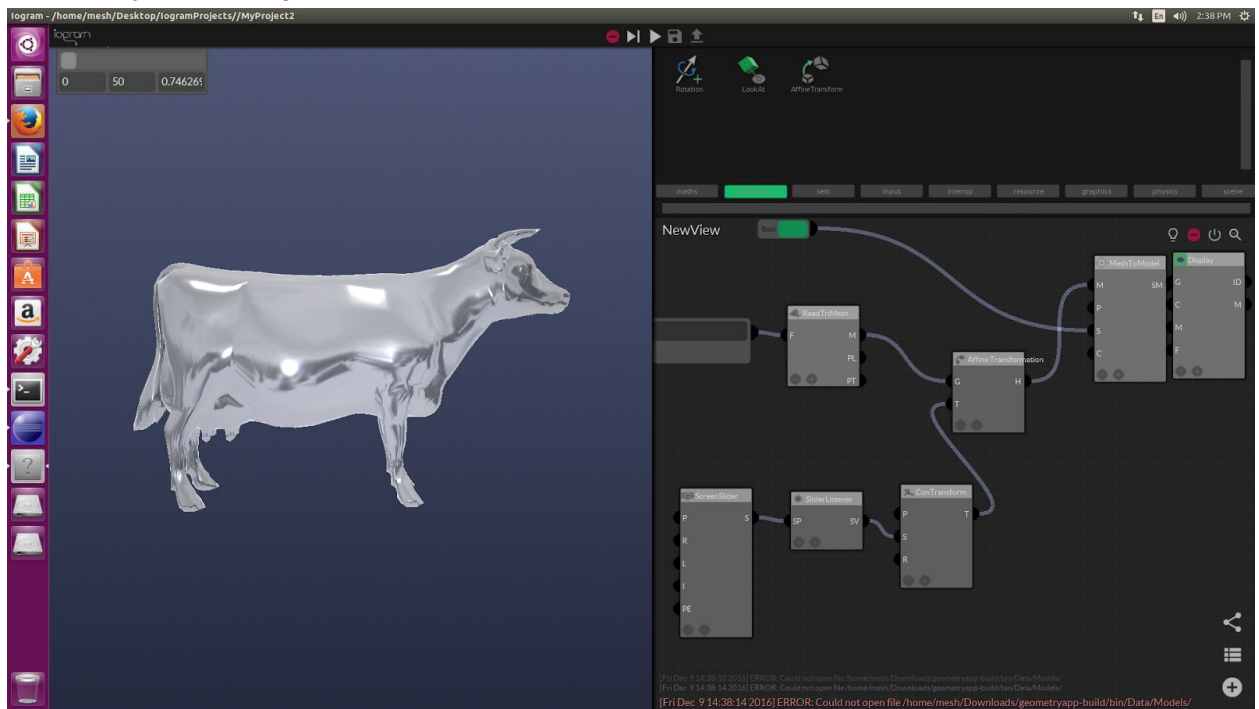
- 4) Now we have a geometric transform we can use to scale the model, but we have to hook it up in such a way that it will act on the model. We will have the transform act directly on the mesh geometry. From the *geometry* tab drag the **AffineTransform** component onto the view. See how we have rearranged the surrounding components to make room for the new one?



Now unhook the **ReadTriMesh** output from the **MeshToModel** input. Your model will disappear. Hook **ReadTriMesh** output M into **AffineTransformation** input G, and hook **ConTransform** output T into **AffineTransformation** input T, as shown.

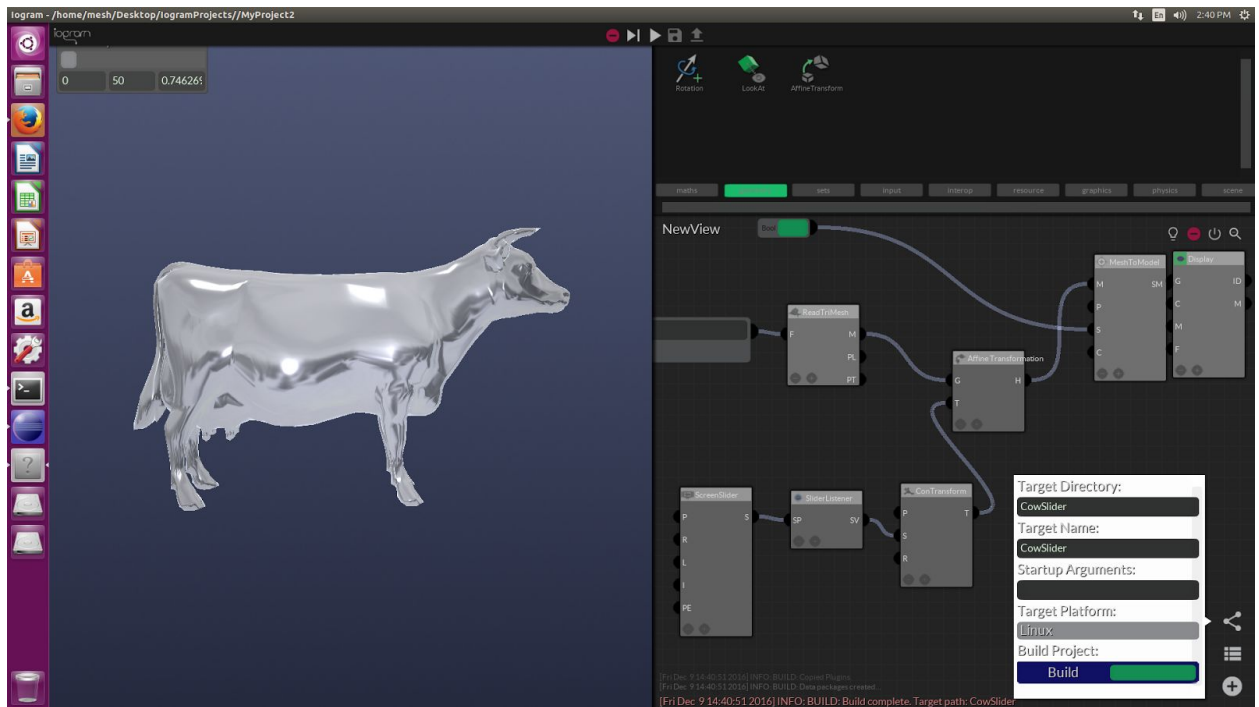


Your model vanished when you unhooked the input at **MeshToModel**, but shouldn't your model be visible again now that it's hooked back up? The model is invisible because the screen slider defaults to a value of zero. Move the slider to see how the model is affected by the scaling.



- 5) Build the app. First save your work by clicking on the save icon at the top of the screen. Then find the *share* icon at the bottom right. First select your target platform. We'll build for Linux by selecting "Linux" from the drop down menu. We've chosen "CowSlider" for

both the TargetDir and TargetName. When you're ready, click Build. The button will turn green when the build is complete.

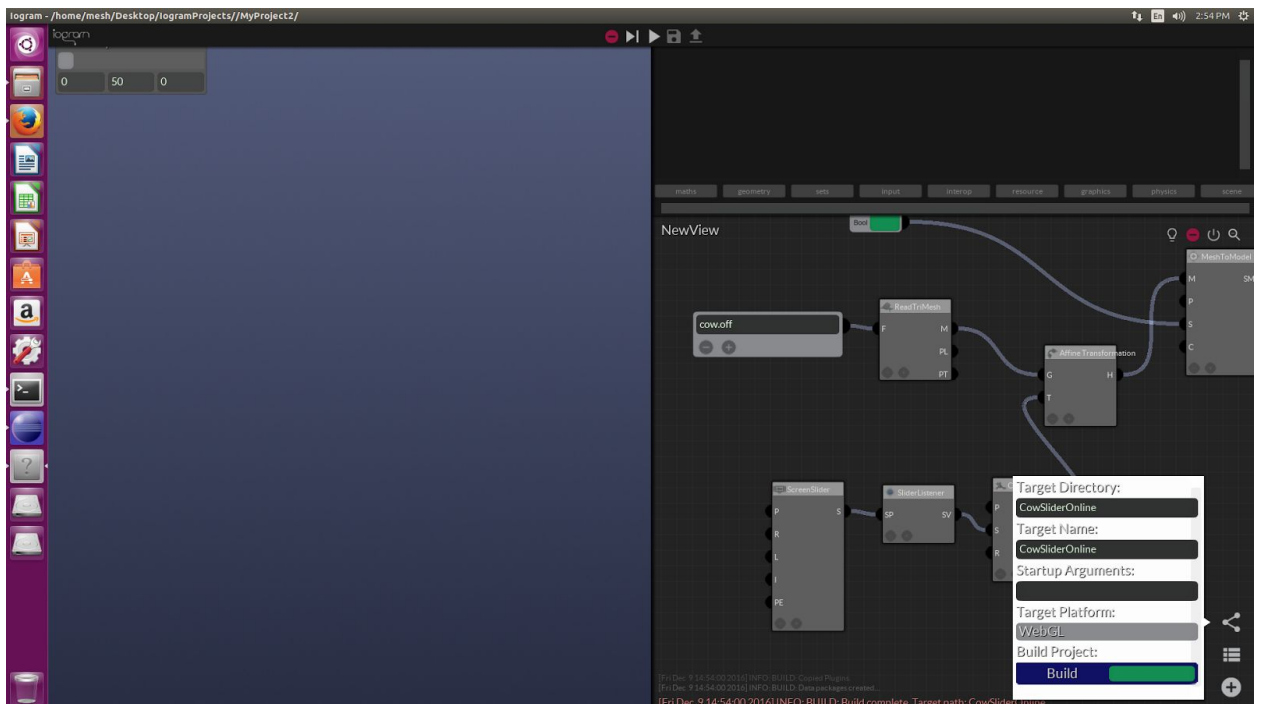


- 6) To try out the application, navigate to your project's directory. The entire CowSlider application is contained in the CowSlider/ folder. This folder can be exported as a standalone application. Find the CowSlider executable in the CowSlider/ folder and launch the application using your preferred method (e.g., double click). *Warning:* You will likely have to change permissions to make the file executable.
- 7) When the application launches, you will not see anything because the slider defaults to zero. Go ahead and drag the slider in the application and watch what happens.



Observe how in the exported application the iogram graph views and component menus are gone. iogram exports the scene and the functionality implemented in the components to a separate standalone application.

- 8) If you closed iogramEditor, launch it again and load the project once more. This time, we will do a WebGL build.



- 9) There are many ways to view the WebGL app. We offer two suggestions:

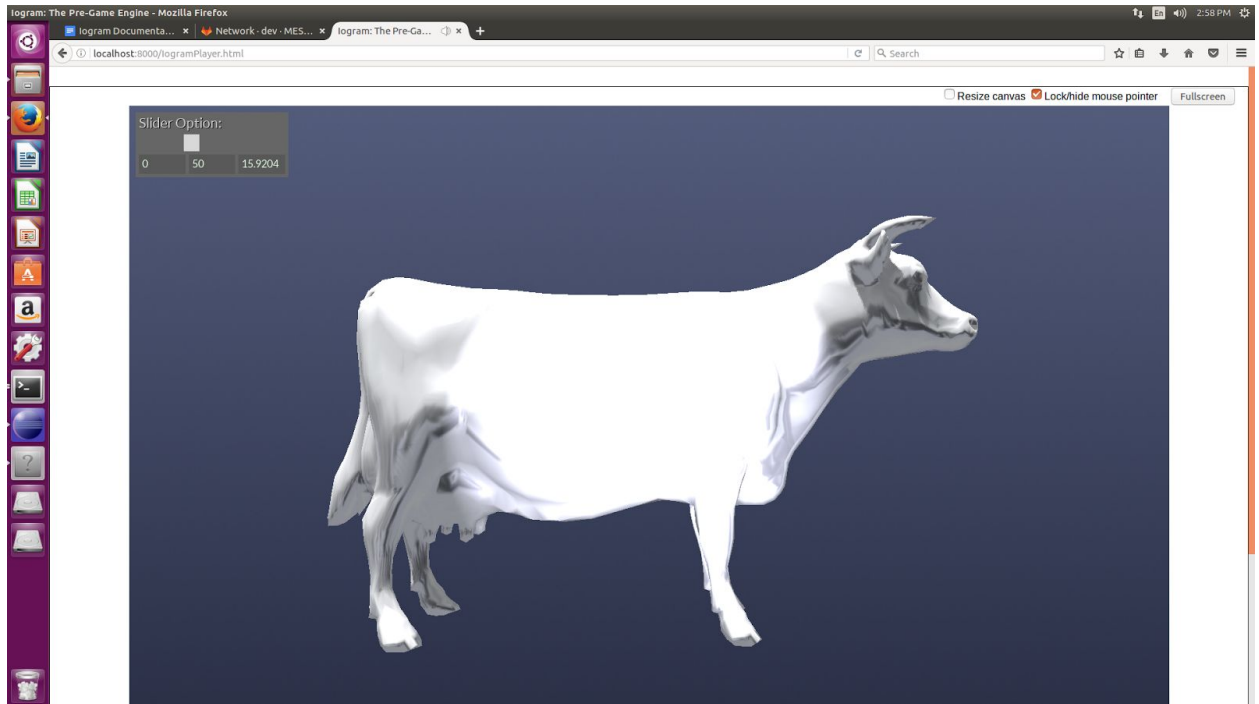
- a) Open a command prompt in the folder where you built the application; for us, it is the CowSliderOnline/ folder. Enter the following python Command (depending on your python installation):

```
'python -m SimpleHTTPServer' OR: 'python -m http.server'
```

Then in a web browser, go to localhost:8000. You should see your app! If it doesn't launch automatically, choose logramPlayer.html from the file list.

- b) Alternatively, copy your TargetDir/ folder into a public dropbox folder. View the app using the public link to the 'logramPlayer.html' file in the TargetDir/ folder.

When the application loads in your browser, you should see something like this:

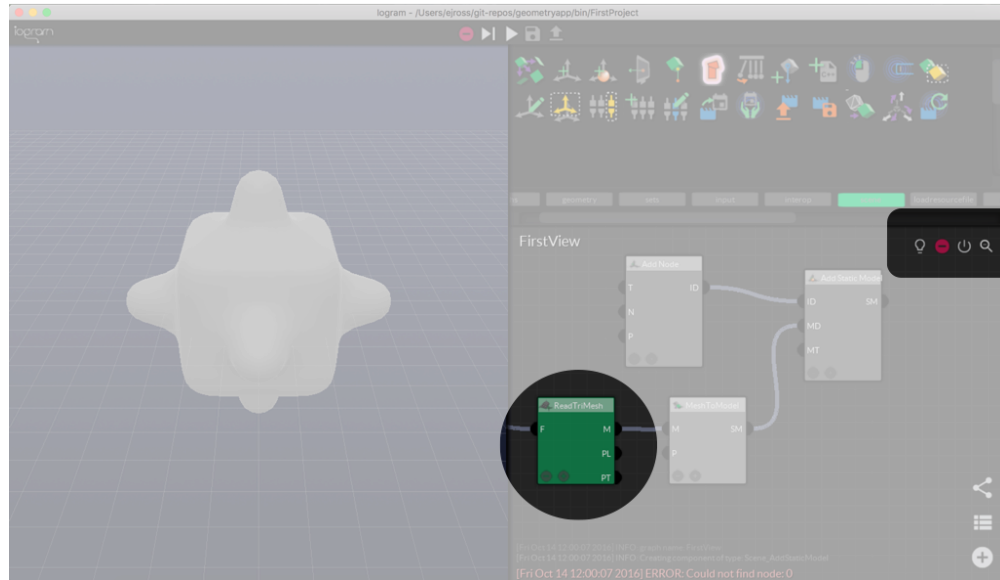


Elements of logram

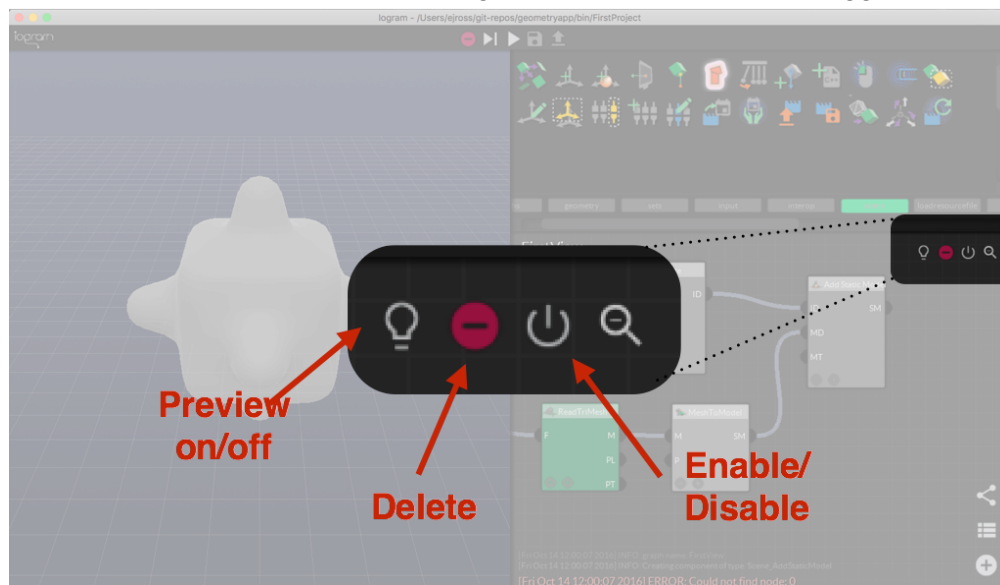
The Editor View

- Components
 - Components are the basis of logram. All of the inputs, geometric primitives, geometric operations, and scene logic are stored as components. See “Component Index and Specifications” section for details on some of the components.
 - Components are loosely grouped into tabs

- Graph navigation (group ops on selected components)
 - The components can be managed using the Component Manager tools.

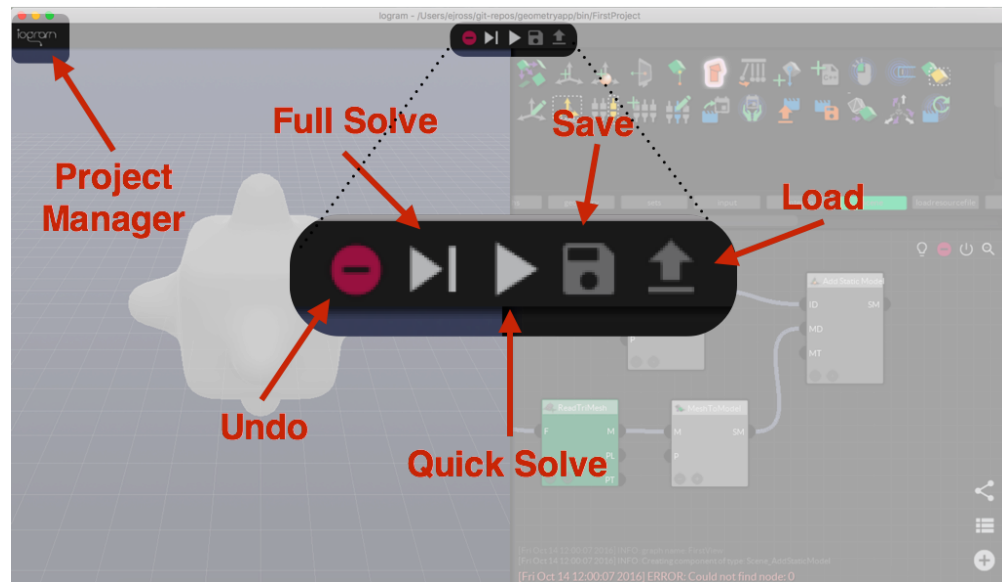


- These tools operate on selected components, which are highlighted green.
- To **delete** a component, select the component and hit delete.
- To **disable** a component from solving, use the Enable/Disable toggle button.

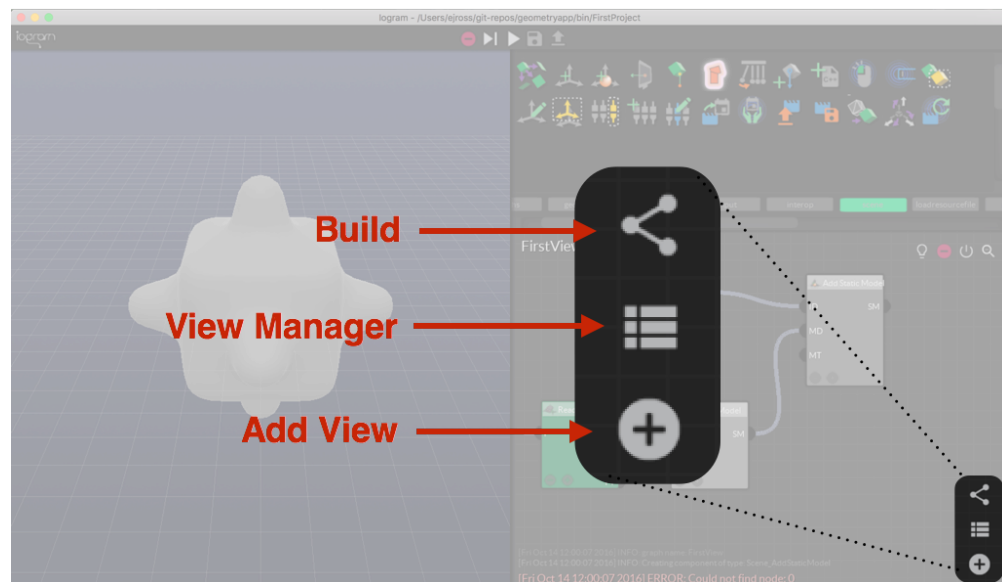


- Saving, loading, solving and project management.

- These tools are located at the top of the display.



- View and Build manager.
 - These tools are at the lower left.



The Scene View

Navigation:

- *3D rotate*: Right mouse click and drag
- *3D translate*: Shift + Right mouse click and drag
- *Zoom*: Ctrl + Right mouse click and drag

Building a project

iogram lets you take the functionality you've built into the Scene and export it as a standalone application. The build process constructs an executable for the platform of your choice, including Win64, OSX, Linux, and WebGL.

Use the Build manager to configure options for the targeted application:

- *Target Dir*: A folder with this name will be created inside the project directory
- *Target Name*: An executable with this name will be created inside the target directory
- *Platform*: choose from Win64, OSX, Linux, and WebGL (note OSX and Linux are disabled for this release).
- *Startup Args*: Leave startup arguments empty for now

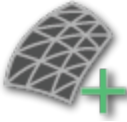

Once all options are configured:







- *Build Project*: Builds and writes the executable for the appropriate platform to the target directory

The relevant build files (e.g. Windows executable) can now be found in the *Target Directory*. (The *Target Directory* is given the name "MyBuild" by default, but you can use any name.)

Component Index and specifications

Below we summarize only a small selection of key components. Basic descriptions of the components and their inputs and outputs are accessible in the app by hovering on the components and their input and output slots.

Name	Description	Inputs	Outputs
ReadTriangleMesh (mesh tab) 	Reads a triangle mesh from disk. <i>Preview Geometry</i> : Meshes, Polylines, Points	F : Path to file. Accepted inputs: .3ds .blend .dae .dxf .lwo .obj .ply .raw .stl .off	M : Meshes PL : Polylines PT : Points
Panel (input tab) 	Flexible input to components. This will default to a string, but may accept other arguments. The input text is parsed into two section, separated by a comma. The content should precede the comma, followed by the type. Examples: <ul style="list-style-type: none"> • 1, int • 1.0, float • 2.1 3.5 4.0, Vector3 • Hello, string <i>Preview Geometry</i> : none.	N/A	String, int, float, quaternion, path

Inspector (input tab) 	Allows inspection of component output. Note that meshes and polylines are stored as "VariantMap"s. <i>Preview Geometry:</i> none.	Anything.	Will output exactly what it reads as input.
AddNode (scene tab) 	Adds a node to the scene. This node is associated with a transform (position, scale, rotation) and an ID. We can associate models and behaviours with this ID. <i>Preview Geometry:</i> axes.	T: transform. Defaults to the origin. Will also accept partial information: Vector3 position, Float (uniform) scale, Quaternion rotation. Or hook up the output of the ConstructTransform (math tab) component. N: (optional) name. P: (optional) parent node.	ID: node ID.
ScreenToggle (input tab) 	Places a button in the scene. Use in combination with ButtonListener	P: Button position in screen coordinates L: Button label (e.g. from the Panel component)	Ptr: pointer to the UI element
ButtonListener (input tab) 	Listens for button input from the scene button. Translates the values back to the graph. Use in combination with ScreenToggle	BP: Pointer to screen button (output from ScreenToggle) M: Boolean to mute the listening. Defaults to true.	V: (Boolean) values from the button.
RigidBody (scene tab) 	Adds rigid body behaviour to a node.	ID: node ID PW: Pointer to the physics world. Connect to the output of the PhysicsWorld component. M: Mass. Set to zero for fixed objects S: Collision shape type. Accepted types (as strings) are: "Box", "Sphere", "Convex"	PB: pointer to rigid body CS: pointer to collision shape.
PhysicsWorld (scene tab) 	Initializes a physics world (simulation).	Go: (Boolean) toggle physics on or off. G: Gravity vector. Defaults to the usual Earth gravity vector.	PW: Pointer to static physics world. Connect this to any rigid bodies you wish to include in the simulation.