Project Proposal

Goal: Develop an interior design simulator

What project will demonstrate: Our project will demonstrate the ability to select 3D models of furniture from a GUI and place them in a rendered scene, select textures from a GUI to display on selected models in the scene, and change room layout with snap-to-grid user transformations in a simulated, graphical environment. We discussed our plans with the professor and decided to add a collision detection element as well.

Resources: ThreeJS, Blender, krita

Outline:

Week 1:
- Create models and UV maps of textures for them
- Create a GUI with a menu of available meshes
- Create a GUI section for available textures

Week 2:
- Connect GUI selection events with mesh loading
- Make sure meshes are selectable in the scene
- Implement snap-to-grid user transforms

Week 3:
- Connect GUI events to texture loading for selected meshes
- Finish user transform functionality

Week 4:
- Testing/Debugging, work on presentation

Group Members: Jordan Rowland, Faye Strawn, Zach Newsom

Methodology

Furniture Models
We decided to model our 3D meshes for the furniture objects in Blender because we wanted to present a finished product with a cohesive, aesthetically pleasing theme. Blender provides a handy UI for UV Mapping image texture onto a model, so we were able to begin creating a system of swappable textures from there.

Image Textures
A key goal for this project was to enable the user to customize the furniture in the scene by changing its material. We used free images of linen, woodgrains, and carpeting to demonstrate

this effect. For complex models with multiple parts (i.e. a couch has an upholstered body and wooden feet), we used the image editing software krita to combine multiple textures into a single image. This allowed us to use a single UV map to map the correct sections of a model to the appropriate textures.

GUI Menu

We initially intended to use dat.gui module we learned about in previous assignments to create an interface for the user to select furniture models and textures to display in the scene. After some experimenting, it became apparent that we would be unable to display thumbnail icons for available options if we continued with this approach. Instead, we created our own GUI menu using simple HTML and CSS. By setting a container's position to 'fixed', anything within that container will appear on top of the canvas. From there, we were able to display images of the available models and textures and connect them to an (onclick) callback function. When the (onclick) callback is invoked from a furniture piece thumbnail, the associated furniture object is loaded using the OBJLoader and added to the scene. We were intentional with naming conventions between our asset files so we could use dynamic functions for loading objects and materials by passing strings containing the designated object name (i.e. "chair") into functions. When the (onclick) callback is invoked from a material thumbnail, if there is an object selected in the scene, a new material is created from the texture file associated with the selected thumbnail and that material is assigned to the selected object.

Transform Controller

We were able to leverage JavaScript's 'DragControls' module to handle the majority of interactions with furniture models in the scene. When a furniture object is created, it is added to an array of draggable objects. We create an event listener to listen for a 'dragstart' event for any of the draggable objects. When a 'dragstart' event fires, this callback function disables the camera controls, saves that object as the world's current "active" object, and sets an emissive on the dragging object's material to indicate that it's active. Setting the active object this way allows for easy click-to-select controls for an object. We also create an event listener for a 'dragend' event. We implement the snap-to-grid process in this callback. To rotate the furniture objects, we create an event listener for a 'keydown' event and watch for the letter 'R' to be pressed. When the letter 'R' is pressed and there is an active object, that object will be rotated on its y-axis by 90 degrees.

Snap-to-Grid

Fairly simple to implement, snap-to-grid is built into the 'dragend' event of DragControls. A helper function round_to_precision, which was found from Mozilla's JavaScript docs. The grid is 4x4 squares, and each square is 4x4 of world space units. The furniture snaps into place by rounding to the nearest multiple of 4. Since the room is centered around the origin the snap to the nearest multiple of 4 works fine for our grid. If another developer wanted to scale the grid, they need only change the SNAP_PRECISION constant to change the step in snap.

Collision Detection

We followed many rabbit holes deeply to figure out how to implement this, but in the end we came up short. The idea was that when a furniture object is moved by the user, if its new placement intersects the placement of another furniture item, the moved item will revert back to its previous position. Unfortunately, due to the complexity of our models, we were unable to sufficiently create and update bounding boxes to check for intersection. We then tried to use a mathy grid system to map the objects' locations to a 2D array, but due to the weirdness of loading complex objects from Blender, we were unable to figure out in time how to properly calculate world positions. In our source code you will find a vast ocean of commented-out code that almost worked, but not quite.

How To Use (7 Steps to Success)

1. Launch application from **project.html**
2. Click furniture icons in lower-left-corner GUI to spawn furniture
3. Click and drag spawned furniture to transform its position
4. Tapping the 'R' key will rotate the last selected object 90 degrees counter-clockwise
5. Select upholstery/wood finishes to change the texture of the most recently dragged object
6. Select rug icons to change rugs
7. Design a marvellous interior to dazzle and entertain house guests