Custom Babylon Importer

Shamanth Thenkan

Github repository

The code and the example files required to test the custom Babylon importer is documented in the repository link mentioned below.

Link - https://github.com/shamanth-thenkan/snappy.git

Pseudo code

> read babylon.js file
    >> load as a json file

> Extract required data from the json file
    >> Mesh data
    >> camera data
    >> Light data
    >> Vertex data

> Loop through meshes
    >> extract corresponding vertex data
    >> create mesh from vertex data
    >> create mesh object
        >>> modify location, rotation
    >> add material if there is any
    >> link the object to the scene

> Loop through Camera
    >> create camera
    >> check camera type
    >> set camera properties
        >> set field of view
        >> set clip data
        >> set focal length
    >> create camera object
        >> update camera location
        >> update camera rotation

    >> create a target object
    >> link camera view to target object
    >> Link camera to scene

> Loop through Light
    >> create light by checking the type
    >> add light properties
        >> light colour
        >> light diffusion
        >> light intensity
        >> light specular factor
        >> check if shadow is on or off
            >>> assign shadow
    >> create light object
    >> link it to the scene

Code

The code for the custom Babylon importer has few boilerplate lines of code compiled in definitions. Definitions include creating a numpy matrix from the list for vertex position, indices, normals and UV maps.

The code is compiled in a way that the input file for the main definition is just json serialised data and the output is a scene with mesh, light and camera in a blender file.

The line of operation to run the code is as such below -

- Install dependencies such as blender python api ‘bpy’
- Install blender development extension in visual studio code by Jacque Lucke
- Press ‘ ctrl + shift + p’ and click start blender in order to link blender to python code
- Load babylon.js file by inputting file location
- Run the script by pressing ‘ctrl + shift + p’ and clicking run script
- The scene will be created in blender.

Challenges

During the course of the development of the importer there were multiple challenges. Some of the significant ones are listed below.

Challenge 1:
Every type of object in the json has its own data structure. Therefore it is easy to overlook certain data structures and run into errors.

- Solution: The sanctity of the data is preserved by calling the values directly from the json data instead of storing small pieces of data as lists or inside variables. This not only reduced the lines of code but also reduced confusion and complexity.

Challenge 2:
Debugging code in blender is a pain. Blender constantly crashes. This takes a lot of time to debug and test because of this.

- Solution: Linking blender to an external IDE made debugging processes smoother and faster. It was also easy to work with multiple libraries.

Challenge 3:
Verifying if the scene created is accurate to the blender.js data.

- Solution: With large datasets it is difficult to go through every line of data. The only thing that we can be sure about is the data structure. Therefore, the initial piece of code was written on a small data object such as ‘ground’ to debug and automate the code. After being sure the code was robust, applied the same logic on more data dense objects such as spheres to verify the code logic works.

Challenge 4:
Getting rid of redundancy and unnecessary repetition of data in for loops.

- Solution: Instead of looping over the whole data set to retrieve certain data and appending it into a list. I could directly seek a bigger portion of the data by calling the keyword and reconfigure its data structure using the numpy library to create arrays in whatever structure I needed for further operations. This makes the processes faster and lighter.

Further Improvements

The current code takes into account essential data to create a required scene but it is worth noting that there is tertiary data that is not considered. For example, inertia of camera, camera speed, exact material information, etc… Although these types of data are not essential they are key to add robustness and finesse to the system and algorithm. With more time and refined data set this can be integrated into the current system.

Conclusion

The code robustly handles importing of babylon.js files into blender with all the object properties integrated in them. There is room to add more complexity and improvement to accommodate varied data types that would constitute a scene.