SGI XML and Parser Issue or Improvement List

The following list is comprised of several issues or improvements to the current state of the XML parser as of 12/11/2023. It's split into various parts that, together, build a scene in THREE.js

Materials

- Non-existent shading type: In the material descriptors there's an option to define what type of shading the rendered should use. This is described as `type: "item"`, when, in reality, THREE.js uses a boolean for this property.
  - Solution suggestion: Change the shading descriptor to include a boolean type as follows `{name: "isFlatShadded", type: "boolean", required: false, default: "false"}`. Keep in mind that this property only exists for the MeshLambert, MeshMatcap, MeshNormal, MeshPhong, MeshPhysical, MeshStandard, and Shader Materials.
- Multiple textures: Remember the CG's project. There were bundles of trees with random offset positions and random textures from a given array of textures. It could be interesting to have that behavior implemented here as well.
  - Solution suggestion: When checking the `MySceneData data` variable, it can be noticed that each node has a list called `materialIDs`. This implies that each node can have multiple materials, when, in reality, only one can be given, because, if multiple `<material />` tags are present within a node, the program crashes. Making it, so this list functions like an actual list, would enable the multi-material behavior. Couple it with a selection function and the developer can choose a material selection method from a predefined set of functions like round-robbin, random, etc.
- Other material parameters are non-existent: Many attributes like the metalness and roughness of materials can't be set.
  - Solution suggestion: There's no good solution other than the one that's suggested for the next issue. Unless the professors enforce a type of material to be used by everyone, only the smallest common subset of material attributes can be used, leaving the developer to decide what to do with the other ones and which material to use, but that's outside the XML, which introduces variance between XML displays.
- No material type definition: One of the biggest gaps in the descriptors is the lack of specification when it comes to material types. The developer essentially has to stick to one material type, most likely MeshPhong, through the entirety of the project, which is far from ideal because it removes a lot of possibilities like metallic, plastic, or glass-like materials.
  - Solution suggestion: Add a `type` attribute to the material descriptors or, even better, add different types of material tags with their associated descriptors, like `<PhongMaterial>`, `<PhysicalMaterial>`, `<BasicMaterial>`, etc.

Textures

- Wrapping mode: XML-defined textures do not allow the setting of different types of wrapping modes.
  - Solution suggestion: Currently within Three.js, there are 3 different wrapping modes (`RepeatWrapping`, `ClampToEdgeWrapping`, and `MirroredRepeatWrapping`). Thus, to solve this, one could add two `item` attributes, wrapS and wrapT with the following descriptor `type: "item", required: false, choices: ["repeat", "clamp", "mirrored"], default: "clamp"`

- Texture rotation: THREE.js allows for texture rotation, which might be useful in some cases where the developer doesn't want to have multiple files for what is essentially the same texture but flipped.
  - Solution suggestion: Add a `rotation` descriptor that takes in a rotation angle as a value

Objects and primitives

- Shadow receiving and casting: Currently the XML doesn't support defining which objects should receive or cast shadows. Not the best example, but walls have no need to cast shadows, reducing the cost of the shadow calculation algorithm, even if by a tiny amount.
  - Solution suggestion: I see two ways of doing this. Either add a per primitive boolean `castshadow` and `receiveshadow` descriptor or allow for the same flow as the materials (if the parent allows for shadow casting or shadow receiving, so should the descendants unless otherwise explicitly said)
- Shadow parameter tuning: Along the shadow receiving and casting, it's sometimes (very much) needed to tune the parameters of shadows emitted by light sources. These parameters include bias, normalBias, camera near, camera far, mapSize width and mapSize height. Their definition can be the difference between having horrible shadow acne or Peter panning or having a nice smooth scene.
  - Solution suggestion: `shadowfar` and `shadowmapsize` are already descriptors of each light, so, it's only natural to add the rest of the parameters. Not that it might be important, but having only `shadowmapsize` implies that mapSize width and mapSize height have the same value. Also, since a lot of descriptors would end up being repeated, some sort of inheritance between XML elements could be beneficial, but that can and should be done by the students when creating the lights.
- Rectangle and box primitive shape definition: One of the most common points of confusion is the fact that the rectangle and box geometries were defined using diametrically opposite points. While it would make sense in some technologies, in THREE.js these geometries are defined based on their dimensions relative to the three axes and are always drawn on the center. The problem here lies with the possibility of the definition of the shapes at any point in space, which adds a level of implicit complexity because the students now have to calculate the geometric center of the shape and translate it.
  - Solution suggestion: Change the definition of the mentioned shapes to take into consideration 3 floating point values as their dimensions, and, if an object is meant to be somewhere, put that logic in the `<transforms>` block, and not in the primitive itself.
- Cylinder open ends: In THREE.js there's the possibility to have open-ended cylinders, which is taken into consideration by the cylinder primitive descriptors, but there's a problem with the very same descriptors: the meaning of the respective variable, `capsclose`, is reversed, which only introduces confusion
  - Solution suggestion: Change `capsclose` with default value `false` (meaning that the cylinder is NOT capped) to `openended` with default value `false` (meaning that the cylinder IS capped) to keep the descriptor uniform with the THREE.js default behavior.

Lights

- Directional light missing attributes: The directional light descriptors is missing the target attribute, making it always point towards the center of the world.
  - Solution suggestion: akin to the spotlight, add a `target` vector3 value.
- Ambient light intensity: The ambient light takes in no intensity parameter, which means it will always take the default value, something that might not be intended for some people.

- Solution suggestion: Akin to the other lights, add an `intensity` descriptor.

Other

- Optional Fog: in the initial demo, fog is given as an optional element, but when removed or not defined, the file reader crashes.
  - Solution suggestion: right before line 646 of the `MyFileReader` file, add `if(elem !== null && elem !== undefined)` to check for the existence of this element.
- Default values: Considering that everything in THREE.js has default values, why don't all the descriptors have them as well? It would save a lot of development time, since, for example, not all materials need to have the color, specular, emissive, and shininess parameters specified, it these match the default ones.
  - Solution suggestion: make almost all descriptors optional and give them default values in accordance with the THREE.js docs. Exceptions of this would probably include the texture `filepath` and all the cameras' descriptors
- Color RGBA: Every color is given as an RBGA value, but, in reality, only the RGB components are used. The A value stands for opacity, which is, so far, a non-existent descriptor for textures.
  - Solution suggestion: Ideally, one would want the opacity for the materials defined as an explicit descriptor, but, for this TPs sake, the consideration that the A value in the given `color` descriptor (ignoring the A value in the `specular` and `emissive` descriptors) is the opacity value of the material could be done. The problem with this consideration is that the parser doesn't do anything with that value so, without changing it, students never have access to it, making this ratchet solution unfeasible.
- Variable definition: Currently, the students have to use magical numbers in order to tune anything in the scene graph as there's no sense of parametrization. This makes reading and writing the XML code very hard, because of the overhead of tracking the dimensions of objects.
  - Solution suggestion: Add a new `<vars>` field at the top of the XML that allows for the definition of variables like `<var name="tableLength" value"3"/>`. These variables could then be used with something similar to how PHP allows for variable referencing in the middle of text: `<box xyz1="$tableHeight $tableLength $tableWidth" xyz2="$tableHeight $tableLength $tableWidth" />`
- Variable algebra: Couples with variable definition, it could be interesting to add even more parametrization with variable algebra. Having variables depend on other variables reduces, even more, the overhead of having to track values.
  - Solution suggestion: In the new suggested `<vars>` XML field, definitions like `<var name="halfTableLength" value"$tableLength/2"/>` could be allowed. Obviously, for simplicity sake, having inline variables like `<box xyz1="$tableHeight/2 $tableLength/2 $tableWidth/2" xyz2="$tableHeight/2 $tableLength/2 $tableWidth/2" />` shouldn't be allowed, as it makes the parser look even more like a compiler.
- Unsupported native functionalities: Many native THREE.js functionalities aren't supported by the XML specification. One of these functionalities is TextGeometry. TextGeometry allows for the rendering of text, without need the for a pre-made texture, as it only takes a string and a font as inputs. Without support for these types of things, many creative ideas might be shut down or cut short, which is far from ideal.
  - Solution suggestion: Not the greatest but go through the entire list of THREE.js functionalities, pick the best/most interesting ones, and add them as primitives (maybe)
- Unsupported addons: All of the addons are sadly, unsupported. Addons like the Reflector are very useful to add depth to a scene.

  - Solution suggestion: unfortunately, we couldn't find a good solution for this issue. It seems that addon integration would have to be on a case-by-case basis. This just highlights one of the flaws this type of project has, and consideration to change it in the future years should be made, as it limits the creative expression of the students, making this "just another project", instead of a fun learning experience like TP1

Compiled by

- Gustavo Costa, up20200418, SGI T03G02
- Joana Santos, up202006279, SGI T05G05
- Mafalda Costa, up202006417, SGI T05G04