Character Modeling using 3js and WebGL Libraries

Nikhil Swami *CSE (Leader)*
School Of Engineering
nikhil46_soe@jnu.ac.in

Ayush Bharti *CSE*
School Of Engineering
ayush38_soe@jnu.ac.in

Ajay Verma *CSE*
School Of Engineering
ajay53_soe@jnu.ac.in

Sayantan Roy *CSE*
School Of Engineering
sayant29_soe@jnu.ac.in

Shish Pal Jatav *CSE*
School Of Engineering
shish40_soe@jnu.ac.in

***Abstract*—In this paper, we've mentioned the fundamentals of any 3D modeling character. This project was mainly carried out in 3js library to create 3D characters and apply the associated graphics to them. WebGL programs consist of control code written in JavaScript and shader code (GLSL) that is executed on a computer's Graphics Processing Unit (GPU). WebGL elements can be mixed with other HTML elements and composited with other parts of the page or page background.**

I. Introduction

Three.js is a cross-browser JavaScript library and application programming interface used to create and display animated 3D computer graphics in a web browser using WebGL. 3D models are available in hundreds of file formats, each with different purposes, assorted features, and varying complexity. Although three.js provides many loaders, choosing the right format and workflow will save time and frustration later on. Some formats are difficult to work with, inefficient for real time experiences, or simply not fully supported at this time. WebGL is a very low-level system that only draws points, lines, and triangles. To do anything useful with WebGL generally requires quite a bit of code and that is where three.js comes in. It handles stuff like scenes, lights, shadows, materials, textures, 3d math, all things that you'd have to write yourself if you were to use WebGL directly. we leverage this library to create 3D scenes with procedural generation techniques.

A. Abbreviations and Acronyms

***JS** (Java Script) , **WebGL** (Web Graphics Library), **API** (Application programming interface) ,**CPU** (central processing unit) , **GPU** (graphics processing unit).*

II. Problem Statement

To create a 3D model. Character modeling is the process of transforming a concept, essentially an idea, into a three-dimensional model. The character artist builds the model from the ground up using tools such as polygon box modeling, hard surface modeling, and digital sculpting techniques. 3D animation is the art of using motion to bring characters, vehicles, props, and more to life within TV shows, films, and games. 3D Artists are often involved in several early steps of the VFX pipeline in order to ensure they deliver an animation-ready model. It'll take lots of trial and error. You may have heard before that it takes a thousand bad drawings to get to those good drawings.

III. Challenges Faced

A. CORS error

We faced many challenges: the code never run at first attempt. we required CORS heading (cross origin resource sharing) to load resources in the browser.

B. Version Requirement

After that there was a requirement to use the correct version of the 3js library. After hours of grinding, we were finally able to make it. Our Team Introduction video was completely written from scratch, while the other code were inspired from online library.

C. Performance Enhancements

There were serious performance issues at the beginning of project, we removed junk models, models with too many vertices and faces. then used a smaller view to reduce ram requirements.

IV. Methodology

The following concepts have been used in the project. and these concepts are very important to get the final output.

A. Perspective Camera

This projection mode is designed to mimic the way the human eye sees. It is the most common projection mode used for rendering a 3D scene. The most common camera in three.js and the one we've been using up to this point is the PerspectiveCamera. It gives a 3d view where things in the distance appear smaller than things up close. Formula we used to move camera in JS : ***Math.sin(2*Math.PI*(n%timeInterval/timeInterval)).*** the camera moves in a sine wave so regions which are PerspectiveCamera defines its frustum based on 4 properties. near defines where the front of the frustum

starts. far defines where it ends. fov, the field of view, defines how tall the front and back of the frustum are by computing the correct height to get the specified field of view at near units from the camera. The aspect defines how wide the front and back of the frustum are. The width of the frustum is just the height multiplied by the aspect.

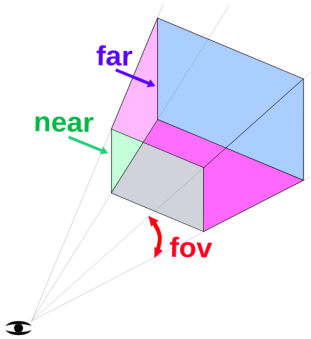
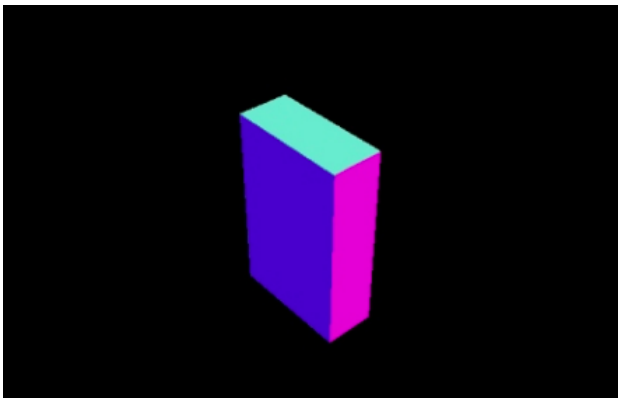


Figure 1: Working of a camera viewport

B. WebGLRenderer

The WebGL Renderer displays your beautifully crafted scenes using WebGL. its the most important aspect of the 3js machinery.When you want to create an initial Three.js project that uses WebGL for rendering, you always have to set up the same couple of variables. You need a THREE.WebGLRenderer object, a THREE.scene object, a camera, and some way to render the scene. ***Three.js now only uses WebGL Renderer to render 3D objects in our browser, and it uses the WebGL API of the browser to render objects and scenes.***

WebGL is fast because it uses the client graphics processing unit to do the work. The CPU of the client isn't doing anything and isn't part of the render. WebGL is highly performant because the CPU is spared while a dedicated GPU built for graphics rendering does



the work.

Figure 2: Example of global illumination

C. AmbientLight - three.js

This light globally illuminates all objects in the scene equally. This light cannot be used to cast shadows as it does not have a direction. The AmbientLight is the cheapest way of faking indirect lighting in three.js. This type of light adds a constant amount of light from every direction to every object in the scene. It doesn't matter where you place this light, and it doesn't matter where other objects are placed relative to the light. This is not at all similar to how light in the real world works. Nonetheless, in combination with one or more direct lights, the AmbientLight gives OKish results. below is the constructor template in 3js.

AmbientLight(color : Integer, intensity : Float)
color - (optional) Numeric value of the RGB component of the color. Default is 0xffffff.
intensity - (optional) Numeric value of the light's strength/intensity. Default is 1.

D. AnimationMixer - three.js

The AnimationMixer is a player for animations on a particular object in the scene. When multiple objects in the scene are animated independently, one Animation Mixer may be used for each object.

AnimationMixer(rootObject : Object 3D)
rootObject - the object whose animations shall be played by this mixer. The AnimationMixer itself has only very few (general) properties and methods, because it can be controlled by the Animation Actions. By configuring an AnimationAction we can determine when a certain AnimationClip shall be played, paused or stopped on one of the mixers, if and how often the clip has to be repeated, whether it shall be performed with a fade or a time scaling, See an example with the help of this project.

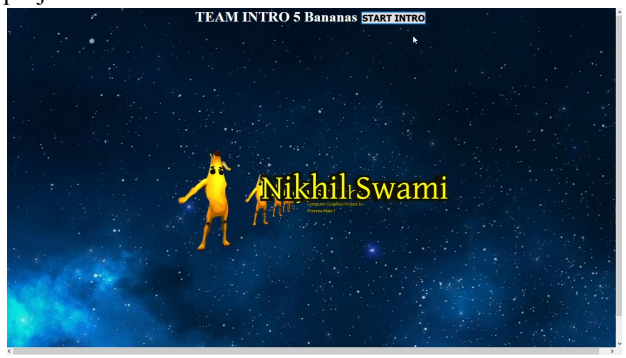


Figure 3 : 3d GLB object Banana animation and 3D text are being displayed simultaneously using 3js.

E. TextGeometry - three.js

A class for generating text as a single geometry. It is constructed by providing a string of text, and a hash of parameters consisting of a loaded Font and settings for the geometry's parent ExtrudeGeometry.

Example:
https://threejs.org/docs/scenes/geometry-browser.html#TextGeometry

Figure 4: example of text geometry in 3js

F. Texture Mapping

Texture mapping is a method of adding realism to a computer-generated graphic. An image (the texture) is added (mapped) to a simpler shape that is generated in the scene, like a decal pasted to a flat surface. This reduces the amount of computing needed to create the shapes and textures in the scene. A surface texture is created by the regular repetition of an element or pattern, called surface texel, on a surface. In computer graphics there are deterministic (regular) and statistical (irregular) textures. Deterministic texture is created by repetition of a fixed geometric shape such as a circle or square.

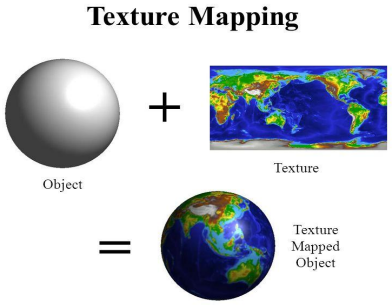


Figure 5 : the texture of earth is wrapped around a 3d sphere to make it look more realistic .

G. Phong-shader

The model that Phong used to simulate the appearance of shiny material is what we call in CG a reflection or shading model. The reason why materials look the way they do is often the result of very complex interactions between light and the microscopic structure of the material objects are made of. It would be too complicated to simulate these interactions therefore we use mathematical models to approximate them instead. The Phong model, which is very popular because of its simplicity, is only one example of such a reflection model but a wide variety of other mathematical models exist.

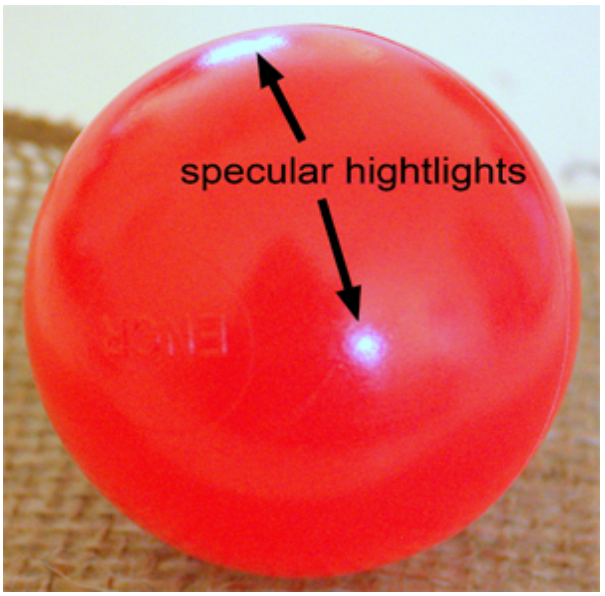


Figure 6: the specular highlights are just a reflection of the strongest sources of light in the scene surrounding the ball. The ball is both diffuse and specular (shiny).

V. Conclusion

At the end of the project we successfully render the character-model, it has illumination, camera view to change perspective, shaders, texture mapping. We get experience of the Javascript based library WebGL.

A. Future Scope

We can generalize this knowledge to various other applications including VR and AR. where objects are interacted in real time. The 3js library also has support for this functionality. And we plan to extend the project in this domain in future.

B. Areas of improvement

The final result of project is still heavy to run on smaller devices, we will improve the loading performance as we find new ways to optimize it. also the objects in model are very huge! so we tried to reduce the size of models.

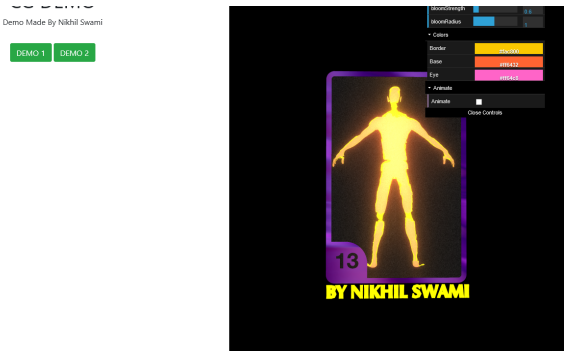


Figure 7: 3js also offers controllers to change model properties in realtime.

VI. Acknowledgement

We extend our sincerest gratitude to **Prof. Prerna Mukherjee** for their constant support and guidance for this project. We learned many new fundamentals and concepts of Computer Graphics.

VII. References

[1]
https://threejsfundamentals.org/threejs/lessons/threejs-fundamentals.html

[2]
https://threejs.org/docs/#api/en/lights/AmbientLight

[3]
https://threejs.org/docs/#api/en/animation/AnimationMixer

[4]
https://threejs.org/docs/#api/en/geometries/TextGeometry

[5]
https://threejs.org/examples/webgl_postprocessing_unreal_bloom.html

[6]
https://www.scratchapixel.com/lessons/3d-basic-rendering/phong-shader-BRDF