

**TRIBHUWAN UNIVERSITY**

**INSTITUTE OF ENGINEERING**

**PULCHOWK CAMPUS**

PROJECT TITLE:

SOCIAL MEDIA ANALYSIS

SUBMITTED BY:

BIPLAB KARKI (073-BEX-412)

JAY KISHAN PANJIYAR (073 -BEX- 414)

DEPARTMENT OF COMPUTER AND ELECTRONICS ENGINEERING

PULCHOWK CAMPUS, LALITPUR

CHAITRA 26,2075

SUBMITTED TO:

Dr**.** Basanta Joshi

##### Mr. [Suresh Pokharel](https://www.facebook.com/pokharelsuresh1?__tn__=lC-R&eid=ARCPrIzMsSWoATewU6EfT6W0mnJPBVs9bvo1ZbV6Tf-iSk9t37okLHr_290GFhoJIy5oxc7g43Rc56Z9&hc_ref=ARS6YOPkoIG7KLw5lo77SIMJq17SzzGxT5T4cZHPisR1EDMV1fpIgSHy60YWqBuNVyo&__xts__%5B0%5D=68.ARBDuzAwgGuxxSyPKqgRXNi0CL0fuyUFj3MwDD4DOagnAM-0vTg1sAsw53vPBjZedb7zo9Ot904uEmSId31rVVbmJ0Y5cK1SabR-WmoRrq_kBzWyBLfBN5QcX3HR6hYwmeaOmNl6k-K0XotgmNF13u096rGloFEfrRzyJr31kFxO-tL81T-5CoSe7fbY7-YFh-EP_tMbIkePw8GascGOsxxre4x-MBt_L3lLuomf8kQciCsZysKAJXxGb6CDYEzImNvRZSOkE7MZQShb8XTy8rLgfMFHzhp2HaNl6oXCYbb-XlCyWs1uyduPEIQtSXm0B5BOZkdme2LrDtMcrbbJ8ZxThQ)

SUBMITTED BY:

BIPLAB KARKI (073-BEX-412)

JAY KISHAN PANJIYAR (073 -BEX- 414)

APPROVED BY:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Dr**.** Basanta Joshi

CLASS TEACHER

DEPARTMENT OF COMPUTER AND ELECTRONICS ENGINEERING

PULCHOWK CAMPUS, LALITPUR

# 

CHAITRA 26,2075

# ACKNOWLEDGEMENT

It is a genuine pleasure to express my deep thanks and gratitude to Dr. Basanta Joshi which granted us to make a project in this semester’s computer graphics project. Our appreciation to Mr. Suresh Pokharel, our lab teacher whose contribution in stimulating suggestions and encouragement helped us a lot for initiating this project. His dedication and keen interest to help his students has been solely responsible for this work. We are thankful for inspiring us constantly for making a project. Their scholarly advice and timely approach have helped us a great deal to begin this task. Furthermore we would also like to acknowledge with much appreciation the crucial role of the library of Pulchowk Campus, which provides us the permission to use all required equipment and the necessary materials. Pulchowk Campus family is to be thanked profusely for providing us a platform to grow and ultimately develop as an educated citizen. We are extremely thankful to our friends and family for becoming a source of constant encouragement and invaluable assistance for this process.

We would like to express our deepest appreciation to all those who provided us the possibility to complete this report.

Secondly we would also like to thank our department seniors who helped us a lot in finalizing this project within the limited time frame.

BIPLAB KARKI

(073bex412.biplab@pcampus.edu.np)

ROLL NO. 073BEX412

JAY KISHAN PANJIYAR

(073bex414.jay@pcampus.edu.np)

ROLL NO. 073BEX414

PULCHOWK CAMPUS , PULCHOWK

# ABSTRACT

This report is based on the project we made for computer graphics subject. It is web based graphics project. It uses three.js to import 3d objects created by using blender in a web page and web page displays the 3d object in interactive form .it is also used to display the audio in 3d form along with interactive objects along with 3d visualization it also display the data of social media analysis like number of likes , comments, shares of Facebook pages in very interactive way. One can login in and signup in our website and once they login they the view the stats of their Facebook pages and groups. The project is basically focused on graphics objects like lighting effect , rendering, curves generation, polygon representation ,etc. this project demonstrated importing of objects in a web page and manipulating it in various ways to create illumination, shadow effects, etc.

# TABLE OF CONTENT

Contents

[ACKNOWLEDGEMENT i](#_Toc3237192)

[ABSTRACT ii](#_Toc3237193)

[TABLE OF CONTENT iii](#_Toc3237194)

[LIST OF FIGURES v](#_Toc3237195)

[LIST OF ABBREVIATIONS v](#_Toc3237196)

[1. INTRODUCTION 1](#_Toc3237197)

[1.1. Background 1](#_Toc3237198)

[1.2. Problem Statement 1](#_Toc3237199)

[1.3. Objective 1](#_Toc3237200)

[1.4. Scope and Applications 1](#_Toc3237201)

[2. THEORETICAL BACKGROUND 2](#_Toc3237202)

[2.1. Three.js 2](#_Toc3237203)

[2.2. Blender 3](#_Toc3237204)

[2.3. HTML 3](#_Toc3237205)

[2.4. CSS 3](#_Toc3237206)

[2.5. Bootstrap 4](#_Toc3237207)

[2.6. Node.js 4](#_Toc3237208)

[2.7. Mongo DB 4](#_Toc3237209)

[2.8. GPU rendering 4](#_Toc3237210)

[2.9. Integrator 4](#_Toc3237211)

[2.10. Surface shader 5](#_Toc3237212)

[2.11. Volume shader 5](#_Toc3237213)

[2.12. Displacement shader 5](#_Toc3237214)

[2.13. Demo reels 5](#_Toc3237215)

[2.14. Rendering engines 5](#_Toc3237216)

[2.15. Block Diagram 6](#_Toc3237217)

[2.16. Working 7](#_Toc3237218)

[3. DESCRIPTION 8](#_Toc3237219)

[3.1. Code Description: 8](#_Toc3237220)

[3.1.1.Creating the scene 8](#_Toc3237221)

[3.1.2. Rendering the scene 8](#_Toc3237222)

[3.1.3. Importing the module 8](#_Toc3237223)

[3.1.4. Loading 8](#_Toc3237224)

[3.1.5Drawing lines 9](#_Toc3237225)

[3.1.6AnimationAction 9](#_Toc3237226)

[3.1.7ArrayCamera 9](#_Toc3237227)

[3.1.8Camera 9](#_Toc3237228)

[3.1.9OrthographicCamera 9](#_Toc3237229)

[3.1.10Custom Blending Equation Constants 10](#_Toc3237230)

[3.1.11Material Constants 10](#_Toc3237231)

[3.1.12WebGLRenderer Constants 12](#_Toc3237232)

[3.1.13Texture Constants 13](#_Toc3237233)

[3.1.14Magnification Filters 14](#_Toc3237234)

[3.1.15Minification Filters 14](#_Toc3237235)

[3.1.16Curve 15](#_Toc3237236)

[3.1.17Lights 15](#_Toc3237237)

[3.1.18DirectionalLight 15](#_Toc3237238)

[3.1.19HemisphereLight 16](#_Toc3237239)

[3.1.20Light 16](#_Toc3237240)

[3.1.21DirectionalLightShadow 16](#_Toc3237241)

[3.1.22LightShadow 16](#_Toc3237242)

[3.1.23SpotLightShadow 17](#_Toc3237243)

[3.1.24Loader 17](#_Toc3237244)

[3.1.25MaterialLoader 17](#_Toc3237245)

[3.1.26ObjectLoader 17](#_Toc3237246)

[3.1.26TextureLoader 17](#_Toc3237247)

[3.1.27ShadowMaterial 18](#_Toc3237248)

[3.1.28CanvasTexture 18](#_Toc3237249)

[3.1.29OBJLoader 18](#_Toc3237250)

[4. METHODOLOGY 19](#_Toc3237251)

[4.1. Interpretation with Block Diagram 19](#_Toc3237252)

[5. OUTPUT 21](#_Toc3237253)

[5.1. Output in pictorial 21](#_Toc3237254)

[**6.** CONCLUSION 26](#_Toc3237255)

[6.1. Project’s Conclusion 26](#_Toc3237256)

[6.2. Limitation 26](#_Toc3237257)

[6.3. Future Enhancement 26](#_Toc3237258)

[6.4. Recommendation 26](#_Toc3237259)

# LIST OF FIGURES

Figure 1:HEADSET

Figure 2:SKULL WITH HEADSET

Figure 3:SKULL WITH HEADSET AND GLASS

Figure 4: LOGIN SCREEN

Figure 5: HOME SCREEN

Figure 6: LOADING SCREEN

Figure 7: SELECT SCREEN

Figure 8: VISUALIZATION SCREEN

Figure 9: VISULAIZATION SCREEN 2

# LIST OF ABBREVIATIONS

js= javascript

Obj=object

# INTRODUCTION

## Background

In today world everyone is connected to various social media network like Facebook, twitter, instagram etc. although being large number of users and social media platform there are very few social media analysis tool which can analyze the data of social to produce fruitful results to be displayed in various from like bar graph, charts, box plots, etc. despite being such high demand and scope of social media analysis tool are not developed.

## Problem Statement

The existing social media analysis tools lack graphics object, it display manipulated data in traditional forms such as charts, graphs, etc. Though social media analysis tool are developed by few organizations who have developed such tools are they are very rare. The existing tools are not user friendly and show the analyzed data with poor graphics.

## Objective

* To visualize the audio in 3d
* To plot graphs in 3d from with graphical object
* To analyze the data of social media

## Scope and Applications

The social media analysis tool is required for everyone from simple users to big cooperate who uses social media for advertisement, promotions, luck draws etc. the parents can monitor their children social media data such as how much time they spent , who often they visited. The corporates can use to know about their outreach in certain area, their product sales, etc. it can help to boost their business. Also the audio visualizer can help to visualize the audio in video from

# THEORETICAL BACKGROUND

## Three.js

Three.js allows the creation of Graphical Processing Unit (GPU)-accelerated 3D animations using the JavaScript language as part of a website without relying on proprietary browser plugins. This is possible due to the advent of WebGL. High-level libraries such as Three.js or GLGE, SceneJS, PhiloGL or a number of other libraries make it possible to author complex 3D computer animations that display in the browser without the effort required for a traditional standalone application or a plugin.

The Three.js library is a single JavaScript file. It can be included within a web page by linking to a local or remote copy.

<script src="js/three.min.js"></script>

Three.js includes the following features:

* Effects: Anaglyph, cross-eyed and parallax barrier.
* Scenes: add and remove objects at run-time; fog
* Cameras: perspective and orthographic; controllers: trackball, [FPS](https://en.wikipedia.org/wiki/First-person_shooter), path and more
* Animation: armatures, [forward kinematics](https://en.wikipedia.org/wiki/Forward_kinematics), [inverse kinematics](https://en.wikipedia.org/wiki/Inverse_kinematics), [morph](https://en.wikipedia.org/wiki/Morph_target_animation) and [keyframe](https://en.wikipedia.org/wiki/Keyframe_animation)
* Lights: ambient, direction, point and spot lights; shadows: cast and receive
* Materials: [Lambert](https://en.wikipedia.org/wiki/Lambertian_reflectance), [Phong](https://en.wikipedia.org/wiki/Phong_shading), smooth shading, textures and more
* Shaders: access to full OpenGL Shading Language ([GLSL](https://en.wikipedia.org/wiki/GLSL)) capabilities: [lens flare](https://en.wikipedia.org/wiki/Lens_flare), [depth pass](https://en.wikipedia.org/wiki/Depth_pass) and extensive post-processing library
* Objects: meshes, particles, sprites, lines, ribbons, [bones](https://en.wikipedia.org/wiki/Skeletal_animation) and more - all with [Level of detail](https://en.wikipedia.org/wiki/Level_of_detail)
* Geometry: plane, cube, sphere, torus, 3D text and more; modifiers: lathe, extrude and tube
* Data loaders: binary, image, [JSON](https://en.wikipedia.org/wiki/JSON) and scene
* Utilities: full set of time and 3D math functions including [frustum](https://en.wikipedia.org/wiki/Frustum), matrix, [quaternion](https://en.wikipedia.org/wiki/Quaternion), [UVs](https://en.wikipedia.org/wiki/UV_mapping) and more
* Export and import: utilities to create Three.js-compatible JSON files from within: [Blender](https://en.wikipedia.org/wiki/Blender_(software)), [openCTM](https://en.wikipedia.org/wiki/OpenCTM), [FBX](https://en.wikipedia.org/wiki/FBX), [Max](https://en.wikipedia.org/wiki/3D_Studio_Max), and [OBJ](https://en.wikipedia.org/wiki/Wavefront_.obj_file)
* Support: API documentation is under construction, public forum and wiki in full operation
* Examples: Over 150 files of coding examples plus fonts, models, textures, sounds and other support files
* Debugging: Stats.js, WebGL Inspector, Three.js Inspector
* Virtual reality: accessing [WebVR](https://en.wikipedia.org/wiki/WebVR)

Three.js runs in all browsers supported by WebGL 1.0.

## Blender

Blender is a free and open-source 3D computer graphics software toolset used for creating animated films, visual effects, art, 3D printed models, interactive 3D applications and video games. Blender's features include 3D modeling, UV unwrapping, texturing, raster graphics editing, rigging and skinning, fluid and smoke simulation, particle simulation, soft body simulation, sculpting, animating, match moving, rendering, motion graphics, video editing and compositing.

Features

* Support for a variety of geometric primitives, including polygon meshes, fast subdivision surface modeling, Bezier curves, NURBS surfaces, metaballs, icospheres, multi-res digital sculpting (including dynamic topology, maps baking, remeshing, resymetrize, decimation), outline font, and a new n-gon modeling system called B-mesh.
* Internal render engine with scanline rendering, indirect lighting, and ambient occlusion that can export in a wide variety of formats.
* A pathtracer render engine called Cycles, which can take advantage of the GPU for rendering. Cycles supports the Open Shading Language since Blender 2.65.[39]
* Integration with a number of external render engines through plugins.
* Keyframed animation tools including inverse kinematics, armature (skeletal), hook, curve and lattice-based deformations, shape animations, non-linear animation, constraints, and vertex weighting.
* Simulation tools for soft body dynamics including mesh collision detection, LBM fluid dynamics, smoke simulation, Bullet rigid body dynamics, ocean generator with waves.
* A particle system that includes support for particle-based hair.
* Modifiers to apply non-destructive effects.

## HTML

Hypertext Markup Language (HTML) is the standard markup language for creating web pages and web applications. With Cascading Style Sheets (CSS) and JavaScript, it forms a triad of cornerstone technologies for the World Wide Web. Web browsers receive HTML documents from a web server or from local storage and render the documents into multimedia web pages. HTML describes the structure of a web page semantically and originally included cues for the appearance of the document.

HTML elements are the building blocks of HTML pages. With HTML constructs, images and other objects such as interactive forms may be embedded into the rendered page. HTML provides a means to create structured documents by denoting structural semantics for text such as headings, paragraphs, lists, links, quotes and other items. HTML elements are delineated by tags, written using angle brackets. Tags such as <img /> and <input /> directly introduce content into the page. Other tags such as <p> surround and provide information about document text and may include other tags as sub-elements. Browsers do not display the HTML tags, but use them to interpret the content of the page.

## CSS

Cascading Style Sheets (CSS) is a style sheet language used for describing the presentation of a document written in a markup language like HTML. CSS is a cornerstone technology of the World Wide Web, alongside HTML and JavaScript.

CSS is designed to enable the separation of presentation and content, including layout, colors, and fonts. This separation can improve content accessibility, provide more flexibility and control in the specification of presentation characteristics, enable multiple web pages to share formatting by specifying the relevant CSS in a separate .css file, and reduce complexity and repetition in the structural content.

Separation of formatting and content also makes it feasible to present the same markup page in different styles for different rendering methods, such as on-screen, in print, by voice (via speech-based browser or screen reader), and on Braille-based tactile devices. CSS also has rules for alternate formatting if the content is accessed on a mobile device

## Bootstrap

Bootstrap is a free and open-source front-end Web framework. It contains HTML and CSS-based design templates for typography, forms, buttons, navigation and other interface components, as well as optional JavaScript extensions. Unlike many earlier web frameworks, it concerns itself with front-end development only.

## Node.js

Node.js is an open-source, cross-platform JavaScript run-time environment that executes JavaScript code outside of a browser. JavaScript is used primarily for client-side scripting, in which scripts written in JavaScript are embedded in a webpage's HTML and run client-side by a JavaScript engine in the user's web browser. Node.js lets developers use JavaScript to write command line tools and for server-side scripting—running scripts server-side to produce dynamic web page content before the page is sent to the user's web browser. Consequently, Node.js represents a "JavaScript everywhere" paradigm,[7] unifying web application development around a single programming language, rather than different languages for server side and client side scripts.

## Mongo DB

MongoDB is a cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with schemata. MongoDB is developed by MongoDB Inc. and licensed under the Server Side Public License (SSPL)MongoDB is a cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with schemata. MongoDB is developed by MongoDB Inc. and licensed under the Server Side Public License (SSPL).

## GPU rendering

Cycles supports GPU rendering which is used to help speed up rendering times. There are two GPU rendering modes: CUDA, which is the preferred method for NVIDIA graphics cards; and OpenCL, which supports rendering on AMD graphics cards. Multiple GPUs are also supported, which can be used to create a render farm – although having multiple GPUs doesn't increase the available memory because each GPU can only access its own memory.

## Integrator

The integrator is the rendering algorithm used for lighting computations. Cycles currently supports a path tracing integrator with direct light sampling. It works well for various lighting setups, but is not as suitable for caustics and some other complex lighting situations. Rays are traced from the camera into the scene, bouncing around until they find a light source such as a lamp, an object emitting light, or the world background. To find lamps and surfaces emitting light, both indirect light sampling (letting the ray follow the surface BSDF) and direct light sampling (picking a light source and tracing a ray towards it) are used.

## Surface shader

The surface shader defines the light interaction at the surface of the mesh. One or more BSDFs can specify if incoming light is reflected back, refracted into the mesh, or absorbedSurface shader

## Volume shader

When the surface shader does not reflect or absorb light, it enters the volume. If no volume shader is specified, it will pass straight through to the other side of the mesh.

If one is defined, a volume shader describes the light interaction as it passes through the volume of the mesh. Light may be scattered, absorbed, or emitted at any point in the volume.[43]

## Displacement shader

The shape of the surface may be altered by displacement shaders. This way, textures can be used to make the mesh surface more detailed.

Depending on the settings, the displacement may be virtual, only modifying the surface normals to give the impression of displacement (also known as bump mapping) or a combination of real and virtual displacement.[43]

## Demo reels

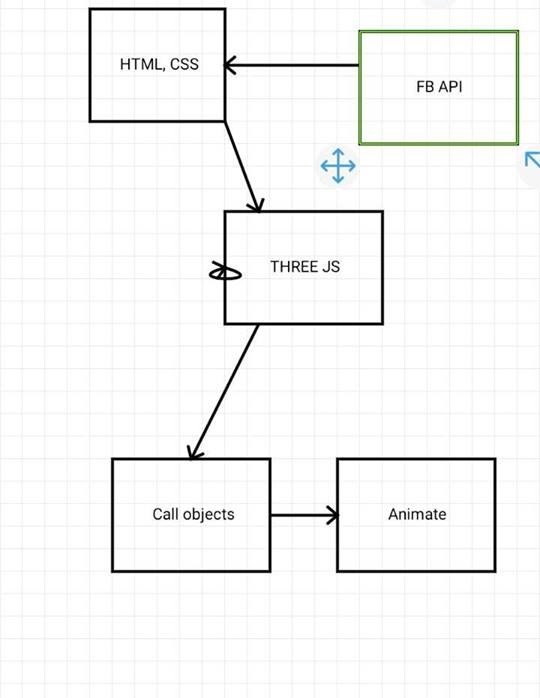
The Blender website contains several demo reels that showcase various features of Blender

## Rendering engines

Engines included in Blender:

* Blender Render (Blender Internal) — Is Blender’s non photorealistic renderer. Will be removed from Blender in version 2.8.
* Cycles — Unbiased ray tracing renderer. Included in Blender from version 2.61.
* Clay Render — Renderer overwrites materials in BI or Cycles to Render clay with choice of diffuse color. Included in Blender from version 2.79.
* EEVEE — Real-time PBR renderer. Render engine has been nicknamed Eevee, later coined backronym — Extra Easy Virtual Environment Engine.[53] Currently in development, it will be available in Blender from version 2.8

## Block Diagram



## Working

As the Facebook API sends the data from facebook to our software, the HTML and CSS file is used to create the front age . the object file is created in blender and it is loaded with texture using three.js . the three.js is used import the object file . the three.js is used to animate the object.. It uses three.js to import 3d objects created by using blender in a web page and web page displays the 3d object in interactive form .it is also used to display the audio in 3d form along with interactive objects along with 3d visualization it also display the data of social media analysis like number of likes , comments, shares of Facebook pages in very interactive way. One can login in and signup in our website and once they login they the view the stats of their Facebook pages and groups. The project is basically focused on graphics objects like lighting effect , rendering, curves generation, polygon representation ,etc. this project demonstrated importing of objects in a web page and manipulating it in various ways to create illumination, shadow effects, etc. The project has presented a simple visualizer of audio along with the statistics in a very well graphical manner. It used graphics objects to make the visualizer interactive . it uses the social media data to find a statistical data and then plotting and displaying was done through web services

# 3. DESCRIPTION

## 3.1. Code Description:

### 3.1.1.Creating the scene

To actually be able to display anything with three.js, we need three things: scene, camera and renderer, so that we can render the scene with camera.

var scene = new THREE.Scene();

var camera = new THREE.PerspectiveCamera( 75, window.innerWidth / window.innerHeight, 0.1, 1000 );

var renderer = new THREE.WebGLRenderer();

renderer.setSize( window.innerWidth, window.innerHeight );

document.body.appendChild( renderer.domElement );

### 3.1.2. Rendering the scene

function animate() {

requestAnimationFrame( animate );

renderer.render( scene, camera );

}

animate();

This will create a loop that causes the renderer to draw the scene every time the screen is refreshed.

### 3.1.3. Importing the module

Assuming that you're bundling your files with a tool such as Webpack or Browserify, which allow you to "require('modules') in the browser by bundling up all of your dependencies.

### 3.1.4. Loading

Only a few loaders (e.g. ObjectLoader) are included by default with three.js — others should be added to your page individually. Depending on your preference and comfort with build tools, choose one of the following:

// global script

<script src="GLTFLoader.js"></script>

// commonjs

var THREE = window.THREE = require('three');

require('three/examples/js/loaders/GLTFLoader');

### 3.1.5Drawing lines

Here is the code that we will use:

var renderer = new THREE.WebGLRenderer();

renderer.setSize( window.innerWidth, window.innerHeight );

document.body.appendChild( renderer.domElement );

var camera = new THREE.PerspectiveCamera( 45, window.innerWidth / window.innerHeight, 1, 500 );

camera.position.set( 0, 0, 100 );

camera.lookAt( 0, 0, 0 );

var scene = new THREE.Scene();

### 3.1.6AnimationAction

AnimationAction( mixer : AnimationMixer, clip : AnimationClip, localRoot : Object3D )

mixer - the AnimationMixer that is controlled by this action.

clip - the AnimationClip that holds the animation data for this action.

localRoot - the root object on which this action is performed.

### 3.1.7ArrayCamera

An instance of ArrayCamera always has an array of sub cameras. It's mandatory to define for each sub camera the **bounds** property which determines the part of the viewport that is rendered with this camera.

Constructor

ArrayCamera( array : Array )

An array of cameras.

### 3.1.8Camera

Constructor

Camera()

Creates a new Camera. Note that this class is not intended to be called directly; you probably want a PerspectiveCamera or OrthographicCamera instead.

### 3.1.9OrthographicCamera

In this projection mode, an object's size in the rendered image stays constant regardless of its distance from the camera.  
  
This can be useful for rendering 2D scenes and UI elements, amongst other things.

Constructor

OrthographicCamera( left : **Number**, right : **Number**, top : **Number**, bottom : **Number**, near : **Number**, far : **Number** )

left — Camera frustum left plane.  
right — Camera frustum right plane.  
top — Camera frustum top plane.  
bottom — Camera frustum bottom plane.  
near — Camera frustum near plane.  
far — Camera frustum far plane.  
  
Together these define the camera's [viewing frustum](https://en.wikipedia.org/wiki/Viewing_frustum)

3.1.10Custom Blending Equation Constants

Blending Equations

THREE.AddEquation

THREE.SubtractEquation

THREE.ReverseSubtractEquation

THREE.MinEquation

THREE.MaxEquation

3.1.11Material Constants

These constants define properties common to all material types, with the exception of Texture Combine Operations which only apply to MeshBasicMaterial, MeshLambertMaterial and MeshPhongMaterial.

Side

THREE.FrontSide

THREE.BackSide

THREE.DoubleSide

Defines which side of faces will be rendered - front, back or both. Default is FrontSide.

Colors

THREE.NoColors

THREE.FaceColors

THREE.VertexColors

NoColors is the default and applies the material's color to all faces.  
FaceColors colors faces according to each Face3 Color value.  
VertexColors colors faces according to each Face3 vertexColors value. This is an array of three Colors, one for each vertex in the face.  
See the [geometry / colors](https://threejs.org/examples/#webgl_geometry_colors) example

Blending Mode

THREE.NoBlending

THREE.NormalBlending

THREE.AdditiveBlending

THREE.SubtractiveBlending

THREE.MultiplyBlending

THREE.CustomBlending

These control the source and destination blending equations for the material's RGB and Alpha sent to the WebGLRenderer for use by WebGL.  
NormalBlending is the default.  
Note that CustomBlending must be set to use Custom Blending Equations.  
See the [materials / blending](https://threejs.org/examples/#webgl_materials_blending) example.

Depth Mode

THREE.NeverDepth

THREE.AlwaysDepth

THREE.LessDepth

THREE.LessEqualDepth

THREE.GreaterEqualDepth

THREE.GreaterDepth

THREE.NotEqualDepth

Which depth function the material uses to compare incoming pixels Z-depth against the current Z-depth buffer value. If the result of the comparison is true, the pixel will be drawn.  
NeverDepth will never return true.  
AlwaysDepth will always return true.  
LessDepth will return true if the incoming pixel Z-depth is less than the current buffer Z-depth.  
LessEqualDepth is the default and will return true if the incoming pixel Z-depth is less than or equal to the current buffer Z-depth.  
GreaterEqualDepth will return true if the incoming pixel Z-depth is greater than or equal to the current buffer Z-depth.  
GreaterDepth will return true if the incoming pixel Z-depth is greater than the current buffer Z-depth.  
NotEqualDepth will return true if the incoming pixel Z-depth is not equal to the current buffer Z-depth.

Texture Combine Operations

THREE.MultiplyOperation

THREE.MixOperation

THREE.AddOperation

These define how the result of the surface's color is combined with the environment map (if present), for MeshBasicMaterial, MeshLambertMaterial and MeshPhongMaterial.   
MultiplyOperation is the default and multiplies the environment map color with the surface color.  
MixOperation uses reflectivity to blend between the two colors.  
AddOperation adds the two colors.

### 3.1.12WebGLRenderer Constants

Cull Face Modes

THREE.CullFaceNone

THREE.CullFaceBack

THREE.CullFaceFront

THREE.CullFaceFrontBack

CullFaceNone disables face culling.  
CullFaceBack culls back faces (default).  
CullFaceFront culls front faces.  
CullFaceFrontBack culls both front and back faces.

Front Face Direction

THREE.FrontFaceDirectionCW

THREE.FrontFaceDirectionCCW

FrontFaceDirectionCW sets the winding order for polygons to clockwise.  
FrontFaceDirectionCCW sets the winding order for polygons to counter-clockwise (default)

Shadow Types

THREE.BasicShadowMap

THREE.PCFShadowMap

THREE.PCFSoftShadowMap

These define the WebGLRenderer's shadowMap.type property.  
  
BasicShadowMap gives unfiltered shadow maps - fastest, but lowest quality.  
PCFShadowMap filters shadow maps using the Percentage-Closer Filtering (PCF) algorithm (default).  
PCFSoftShadowMap filters shadow maps using the Percentage-Closer Soft Shadows (PCSS) algorithm.

Tone Mapping

THREE.NoToneMapping

THREE.LinearToneMapping

THREE.ReinhardToneMapping

THREE.Uncharted2ToneMapping

THREE.CineonToneMapping

THREE.ACESFilmicToneMapping

These define the WebGLRenderer's toneMapping property. This is used to approximate the appearance of high dynamic range (HDR) on the low dynamic range medium of a standard computer monitor or mobile device's screen.  
  
NoToneMapping disables tone mapping.  
LinearToneMapping is the default.

### 3.1.13Texture Constants

Mapping Modes

THREE.UVMapping

THREE.CubeReflectionMapping

THREE.CubeRefractionMapping

THREE.EquirectangularReflectionMapping

THREE.EquirectangularRefractionMapping

THREE.SphericalReflectionMapping

THREE.CubeUVReflectionMapping

THREE.CubeUVRefractionMapping

These define the texture's mapping mode.  
UVMapping is the default, and maps the texture using the mesh's UV coordinates.  
  
The rest define environment mapping types.  
  
CubeReflectionMapping and CubeRefractionMapping are for use with a CubeTexture, which is made up of six textures, one for each face of the cube. CubeReflectionMapping is the default for a CubeTexture.   
  
EquirectangularReflectionMapping and EquirectangularRefractionMapping are for use with an equirectangular environment map. Also called a lat-long map, an equirectangular texture represents a 360-degree view along the horizontal centerline, and a 180-degree view along the vertical axis, with the top and bottom edges of the image corresponding to the north and south poles of a mapped sphere.  
  
SphericalReflectionMapping is for use with a spherical reflection map such as may be obtained by cropping a photograph of a mirrored ball. Sphere maps will be rendered "facing" the camera, irrespective of the position of the camera relative to the cubemapped object or surface.

**Wrapping Modes**

THREE.RepeatWrapping

THREE.ClampToEdgeWrapping

THREE.MirroredRepeatWrapping

These define the texture's wrapS and wrapT properties, which define horizontal and vertical texture wrapping.  
  
With RepeatWrapping the texture will simply repeat to infinity.  
  
ClampToEdgeWrapping is the default. The last pixel of the texture stretches to the edge of the mesh.  
  
With MirroredRepeatWrapping the texture will repeats to infinity, mirroring on each repeat.

### 3.1.14Magnification Filters

THREE.NearestFilter

THREE.LinearFilter

For use with a texture's magFilter property, these define the texture magnification function to be used when the pixel being textured maps to an area less than or equal to one texture element (texel).  
  
NearestFilter returns the value of the texture element that is nearest (in Manhattan distance) to the specified texture coordinates.  
  
LinearFilter is the default and returns the weighted average of the four texture elements that are closest to the specified texture coordinates, and can include items wrapped or repeated from other parts of a texture, depending on the values of wrapS and wrapT, and on the exact mapping.

### 3.1.15Minification Filters

THREE.NearestFilter

THREE.NearestMipMapNearestFilter

THREE.NearestMipMapLinearFilter

THREE.LinearFilter

THREE.LinearMipMapNearestFilter

THREE.LinearMipMapLinearFilter

For use with a texture's minFilter property, these define the texture minifying function that is used whenever the pixel being textured maps to an area greater than one texture element (texel).  
  
In addition to NearestFilter and LinearFilter, the following four functions can be used for minification:  
  
NearestMipMapNearestFilter chooses the mipmap that most closely matches the size of the pixel being textured and uses the NearestFilter criterion (the texel nearest to the center of the pixel) to produce a texture value.  
  
NearestMipMapLinearFilter chooses the two mipmaps that most closely match the size of the pixel being textured and uses the NearestFilter criterion to produce a texture value from each mipmap. The final texture value is a weighted average of those two values.  
  
LinearMipMapNearestFilter chooses the mipmap that most closely matches the size of the pixel being textured and uses the LinearFilter criterion (a weighted average of the four texels that are closest to the center of the pixel) to produce a texture value.  
  
LinearMipMapLinearFilter is the default and chooses the two mipmaps that most closely match the size of the pixel being textured and uses the LinearFilter criterion to produce a texture value from each mipmap. The final texture value is a weighted average of those two values.

### 3.1.16Curve

An abstract base class for creating a Curve object that contains methods for interpolation. For an array of Curves see CurvePath.

### 3.1.17Lights

AmbientLight

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.

Constructor

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.  
  
Creates a new AmbientLight.

### 3.1.18DirectionalLight

A light that gets emitted in a specific direction. This light will behave as though it is infinitely far away and the rays produced from it are all parallel. The common use case for this is to simulate daylight; the sun is far enough away that its position can be considered to be infinite, and all light rays coming from it are parallel.

Constructor

DirectionalLight( color : **Integer**, intensity : **Float** )

color - (optional) hexadecimal color of the light. Default is 0xffffff (white).  
intensity - (optional) numeric value of the light's strength/intensity. Default is 1.  
  
Creates a new DirectionalLight

### 3.1.19HemisphereLight

A light source positioned directly above the scene, with color fading from the sky color to the ground color.   
  
This light cannot be used to cast shadows.

Constructor

HemisphereLight( skyColor : **Integer**, groundColor : **Integer**, intensity : **Float** )

skyColor - (optional) hexadecimal color of the sky. Default is 0xffffff.  
groundColor - (optional) hexadecimal color of the ground. Default is 0xffffff.  
intensity - (optional) numeric value of the light's strength/intensity. Default is 1.  
  
Creates a new HemisphereLight

### 3.1.20Light

Abstract base class for lights - all other light types inherit the properties and methods described here

Constructor

Light( color : **Integer**, intensity : **float** )

color - (optional) hexadecimal color of the light. Default is 0xffffff (white).  
intensity - (optional) numeric value of the light's strength/intensity. Default is 1.  
  
Creates a new Light. Note that this is not intended to be called directly (use one of derived classes instead).

### 3.1.21DirectionalLightShadow

This is used internally by DirectionalLights for calculating shadows.  
  
Unlike the other shadow classes, this uses an OrthographicCamera to calculate the shadows, rather than a PerspectiveCamera. This is because light rays from a DirectionalLight are parallel.

Constructor

DirectionalLightShadow( )

Creates a new DirectionalLightShadow. This is not intended to be called directly - it is called internally by DirectionalLight

### 3.1.22LightShadow

This is used internally by PointLights for calculating shadows, and also serves as a base class for the other shadow classes

Constructor

LightShadow( camera : Camera )

camera - the light's view of the world.  
  
Create a new LightShadow. This is not intended to be called directly - it is called internally by PointLight or used as a base class by other light shadows

### 3.1.23SpotLightShadow

This is used internally by SpotLights for calculating shadows.

Constructor

The constructor creates a PerspectiveCamera : PerspectiveCamera to manage the shadow's view of the world.

### 3.1.24Loader

Base class for implementing loaders

Constructor

Loader()

Creates a new Loader. This should be called as base class

### 3.1.25MaterialLoader

A loader for loading a Material in JSON format. This uses the FileLoader internally for loading files.

Constructor

MaterialLoader( manager : LoadingManager )

manager — The loadingManager for the loader to use. Default is THREE.DefaultLoadingManager.  
  
Creates a new MaterialLoader.

### 3.1.26ObjectLoader

A loader for loading a JSON resource in the [JSON Object/Scene format](https://github.com/mrdoob/three.js/wiki/JSON-Object-Scene-format-4).  
  
This uses the FileLoader internally for loading files

Constructor

ObjectLoader( manager : LoadingManager )

manager — The loadingManager for the loader to use. Default is THREE.DefaultLoadingManager.  
  
Creates a new ObjectLoader.

### 3.1.26TextureLoader

Class for loading a texture. This uses the ImageLoader internally for loading files.

Constructor

TextureLoader( manager : LoadingManager )

manager — The loadingManager for the loader to use. Default is THREE.DefaultLoadingManager.  
  
Creates a new TextureLoader.

### 3.1.27ShadowMaterial

This material can receive shadows, but otherwise is completely transparent.

Constructor

ShadowMaterial( parameters : **Object** )

parameters - (optional) an object with one or more properties defining the material's appearance. Any property of the material (including any property inherited from Material and ShaderMaterial) can be passed in here.

### 3.1.28CanvasTexture

Creates a texture from a canvas element.  
  
This is almost the same as the base Texture class, except that it sets needsUpdate to **true** immediately.

Constructor

CanvasTexture( canvas : HTMLElement, mapping : Constant, wrapS : Constant, wrapT : Constant, magFilter : Constant, minFilter : Constant, format : Constant, type : Constant, anisotropy : **Number** )

canvas -- The HTML canvas element from which to load the texture.   
mapping -- How the image is applied to the object. An object type of THREE.UVMapping. See mapping constants for other choices.  
wrapS -- The default is THREE.ClampToEdgeWrapping. See wrap mode constants for other choices.  
wrapT -- The default is THREE.ClampToEdgeWrapping. See wrap mode constants for other choices.  
magFilter -- How the texture is sampled when a texel covers more than one pixel. The default is THREE.LinearFilter. See magnification filter constants for other choices.  
minFilter -- How the texture is sampled when a texel covers less than one pixel. The default is THREE.LinearMipMapLinearFilter. See minification filter constants for other choices.  
format -- The format used in the texture. See format constants for other choices.  
type -- Default is THREE.UnsignedByteType. See type constants for other choices.  
anisotropy -- The number of samples taken along the axis through the pixel that has the highest density of texels. By default, this value is 1. A higher value gives a less blurry result than a basic mipmap, at the cost of more texture samples being used. Use renderer.getMaxAnisotropy() to find the maximum valid anisotropy value for the GPU; this value is usually a power of 2.

### 3.1.29OBJLoader

A loader for loading a .obj resource.  
The [OBJ file format](https://en.wikipedia.org/wiki/Wavefront_.obj_file) is a simple data-format that represents 3D geometry in a human readable format as the position of each vertex, the UV position of each texture coordinate vertex, vertex normals, and the faces that make each polygon defined as a list of vertices, and texture vertices.

Constructor

OBJLoader( manager : LoadingManager )

manager — The loadingManager for the loader to use. Default is THREE.DefaultLoadingManager.

Creates a new OBJLoader

# METHODOLOGY

## Interpretation with Block Diagram

login

click

FB API call

Threejs call object 

Start animation



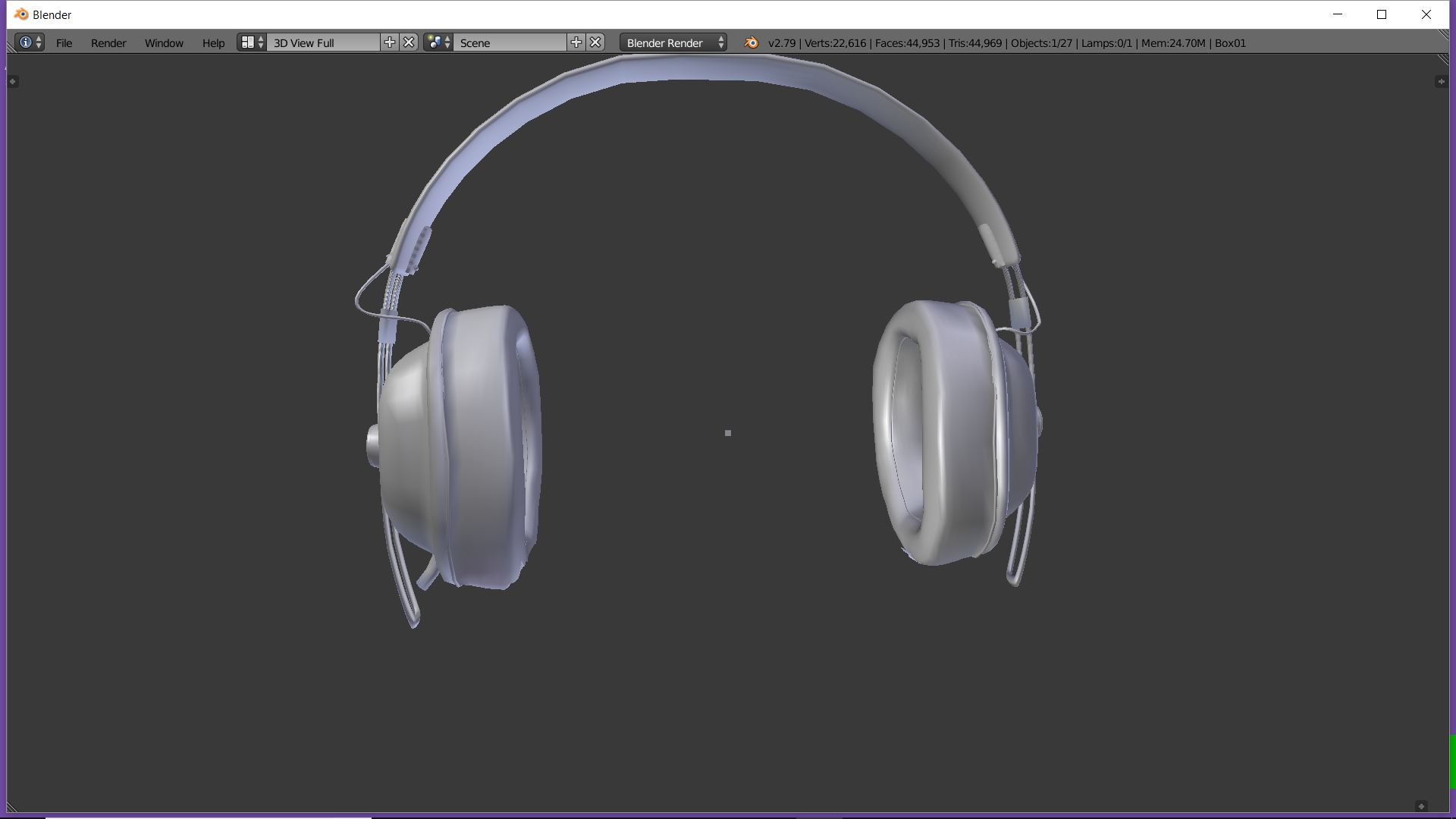


Fig 1: HEAD SET

\

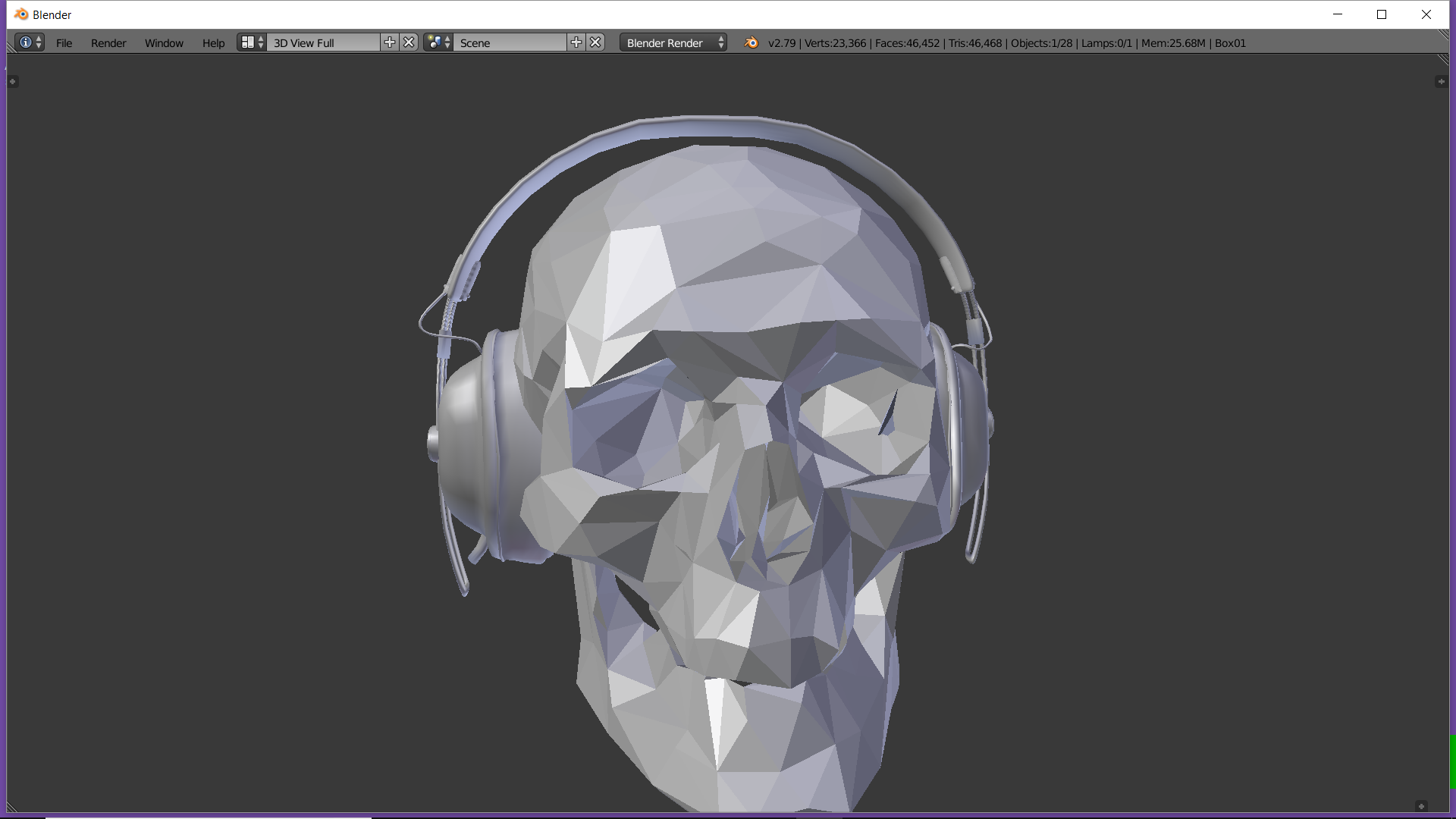


Fig 2: SKULL WITH HEAD SET

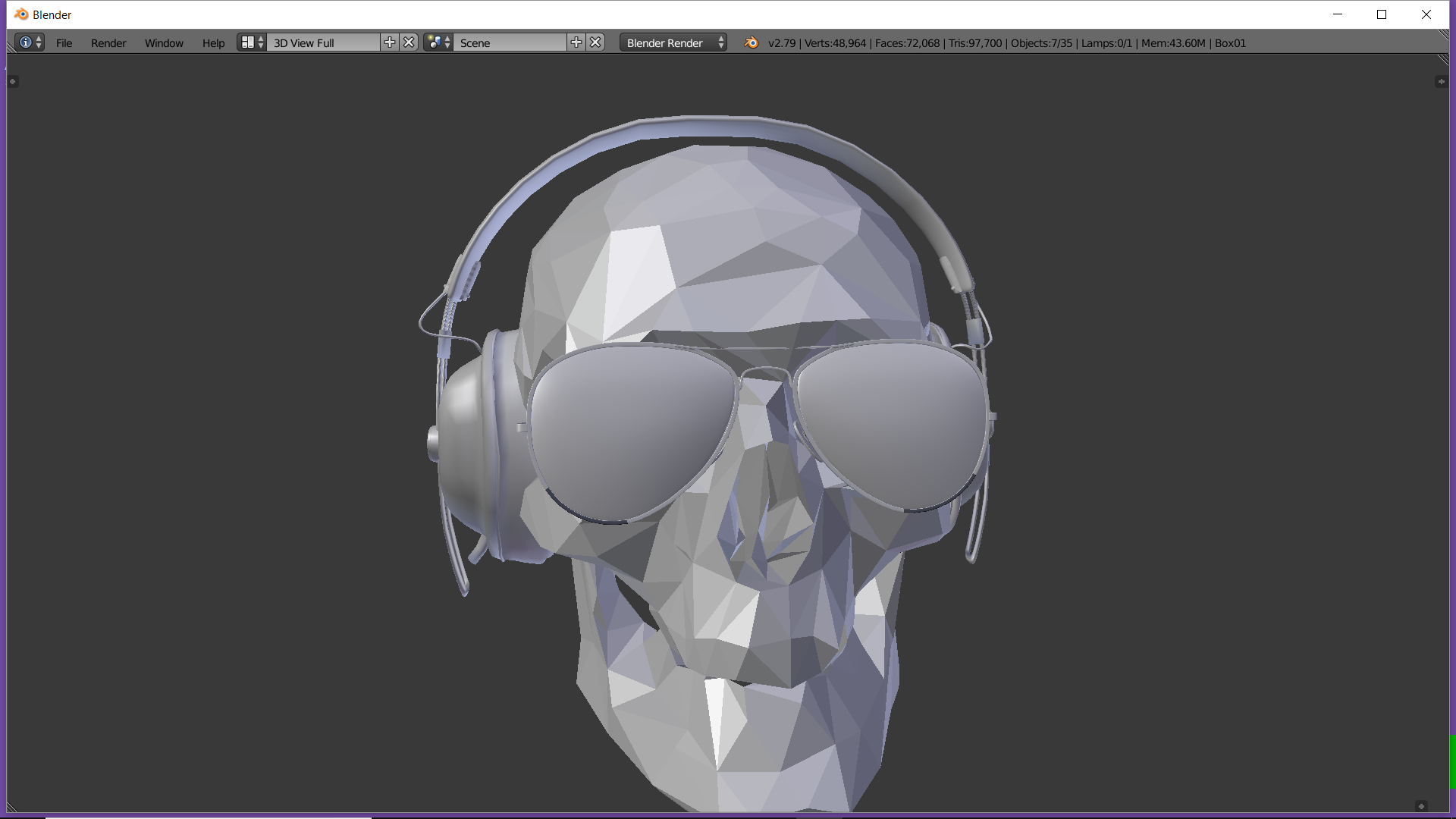


Fig 3:SKULL WITH HEADSET AND GLASS

# OUTPUT

## Output in pictorial

The output of this project can be realized by the help of following pictures.

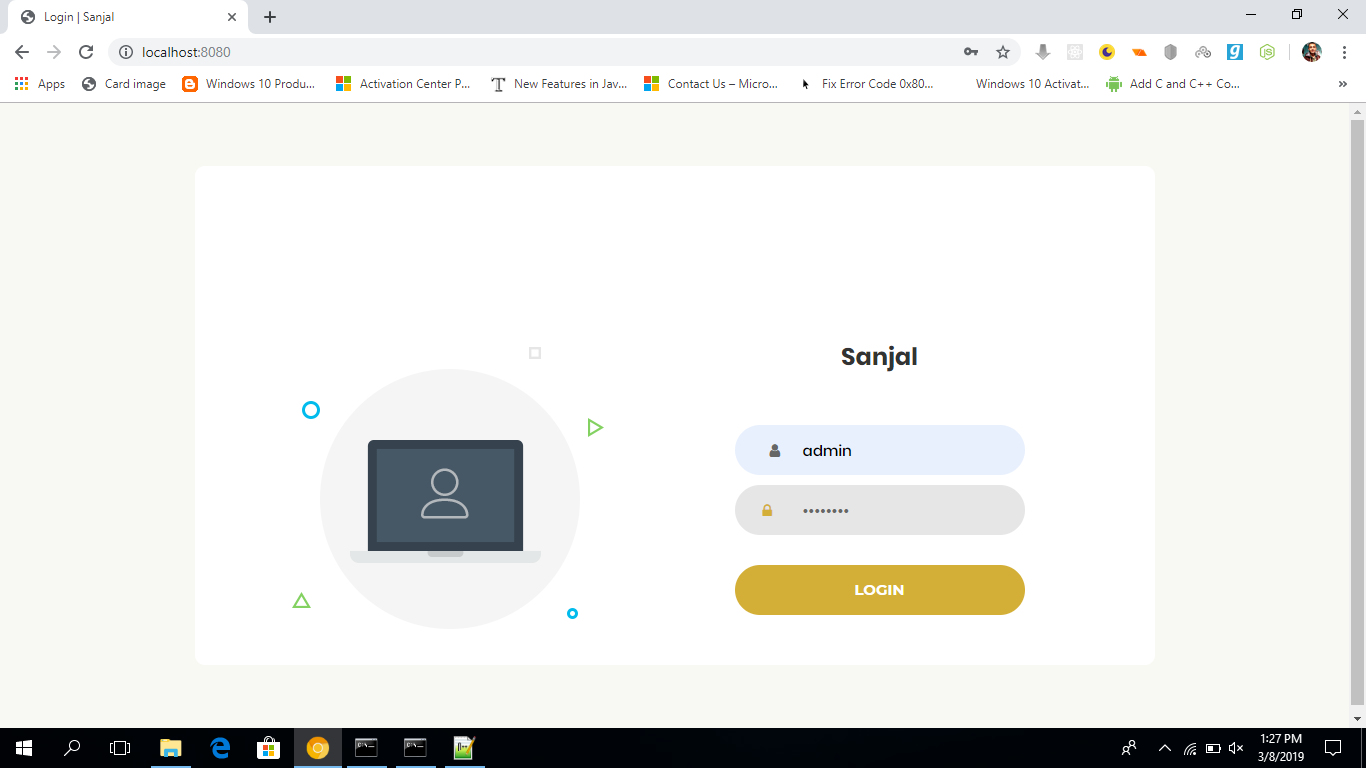
****

Fig 4: LOGIN SCREEN

When our website is visited it asks for credentials of the user . there are two types of account in our system one for admin whi controls all the users who are using the product. The next account is for users who can browse their respective pages and groups to see their analyzed reports.

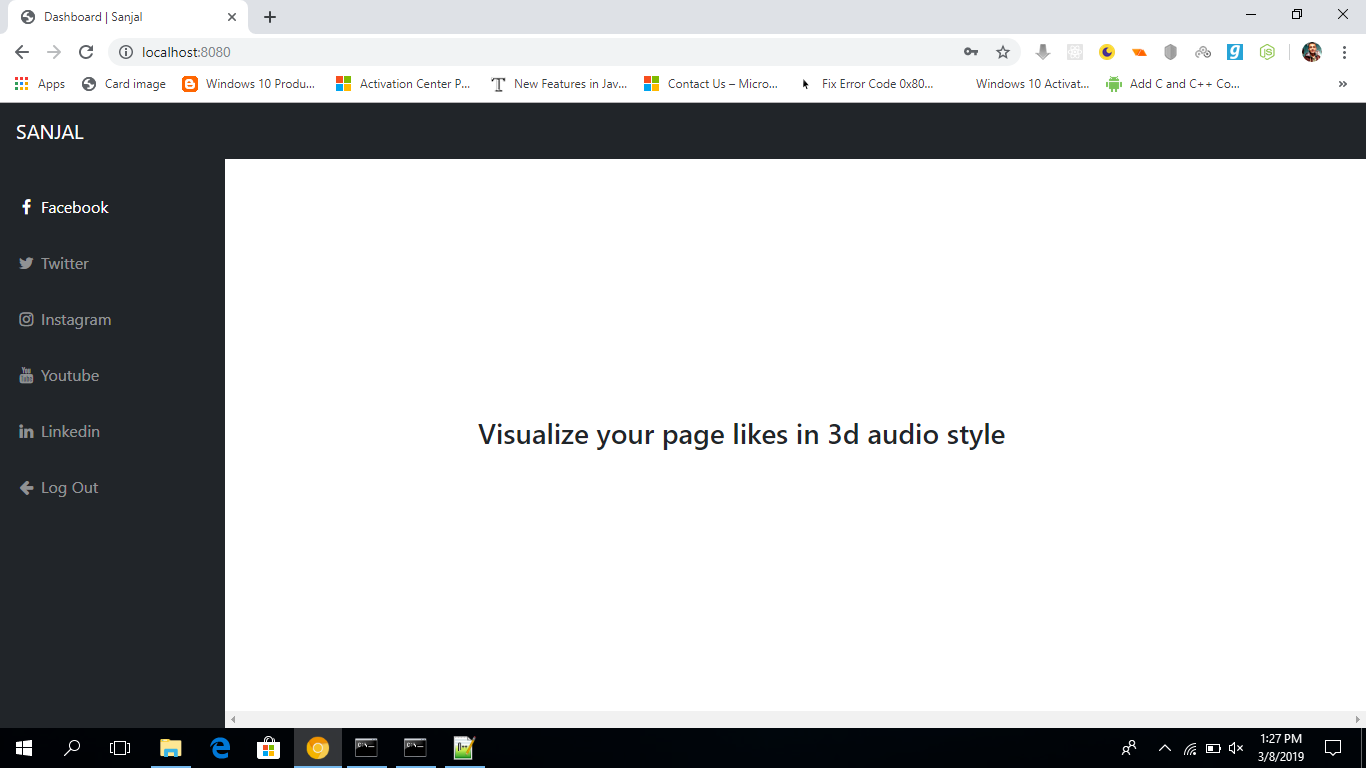


Fig 5: HOME SCREEN

When the user login to the website providing the credentials the above home screen is displayed . in this home page you can browse to facebook, twitter, instagram, or youtube to view your statistics.

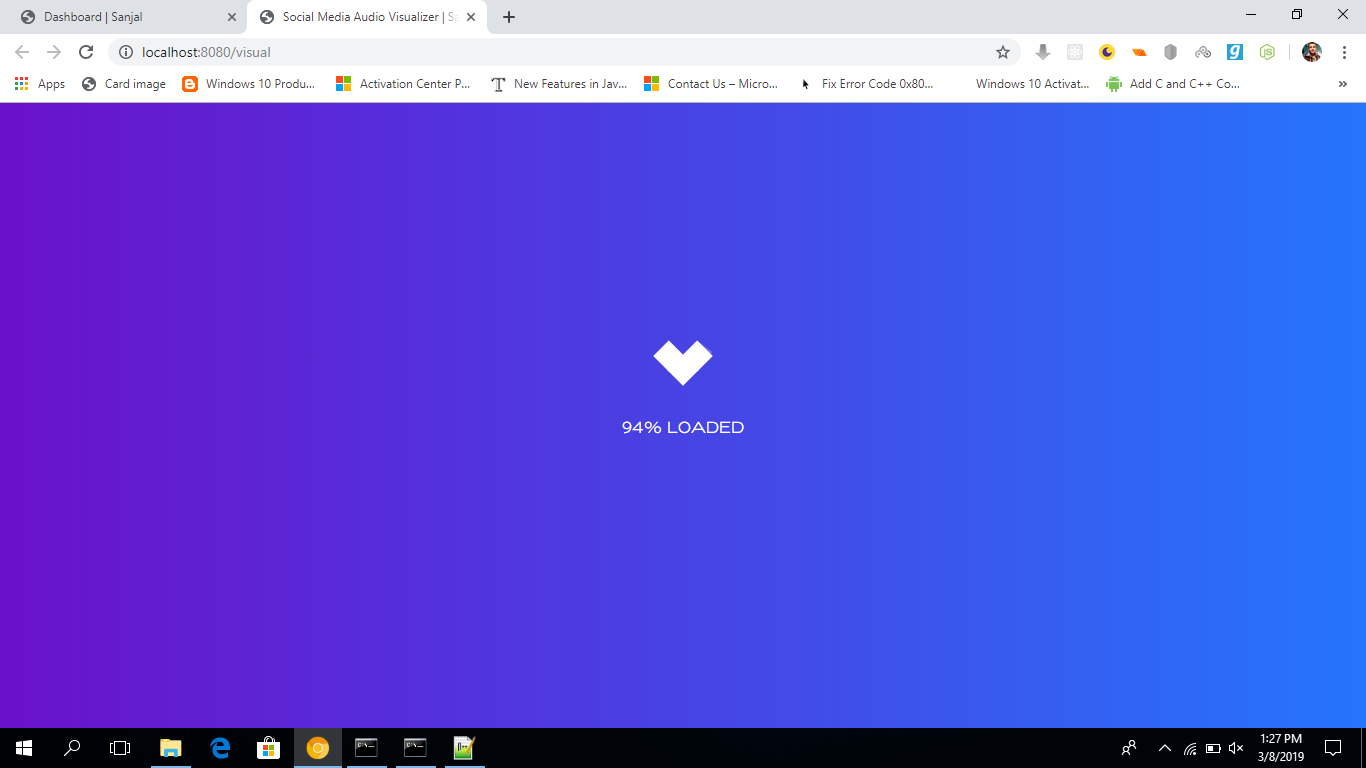


Fig 6: LOADING SCREEN

After the user click on any social network site , the screen loads to process further

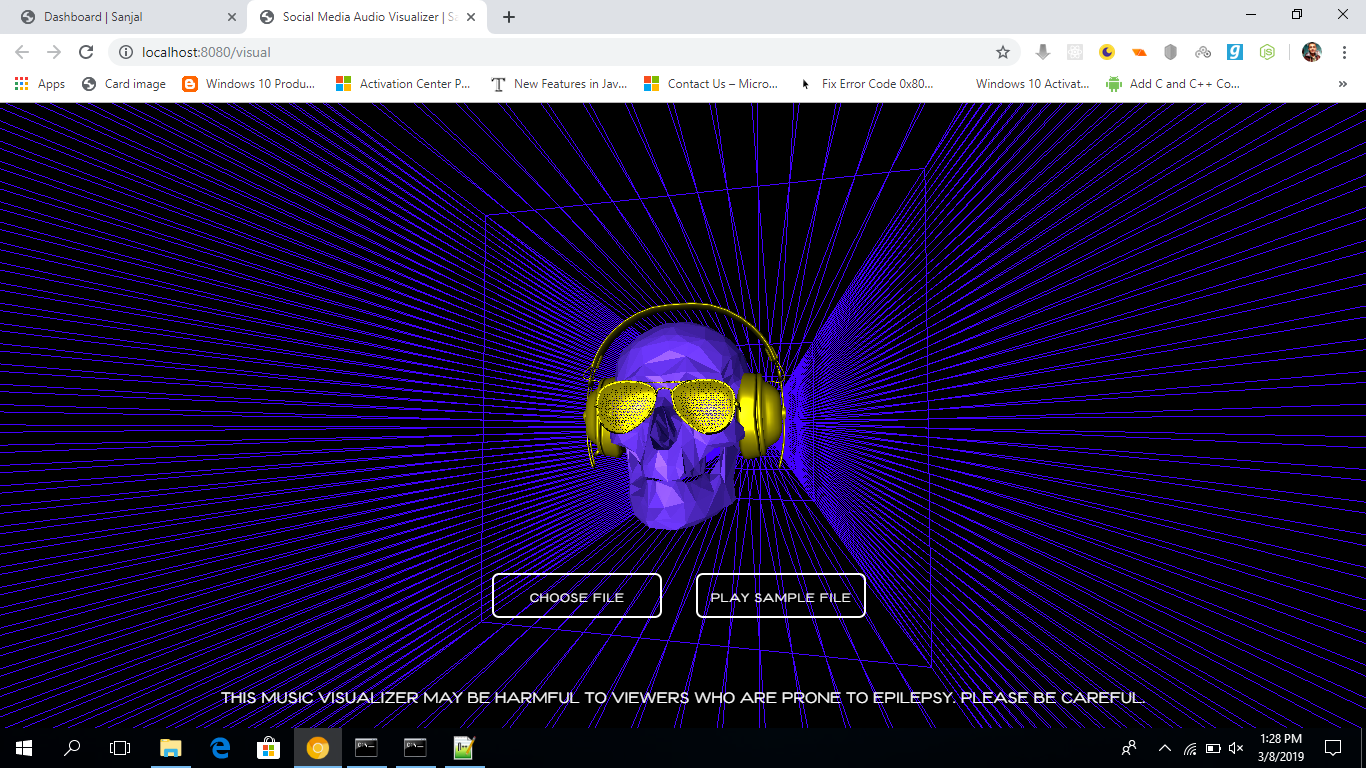


Fig 7: SELECT SCREEN

After the screen loads the page gives option to either choose a file whose visualization has to be displayed or play a sample file .

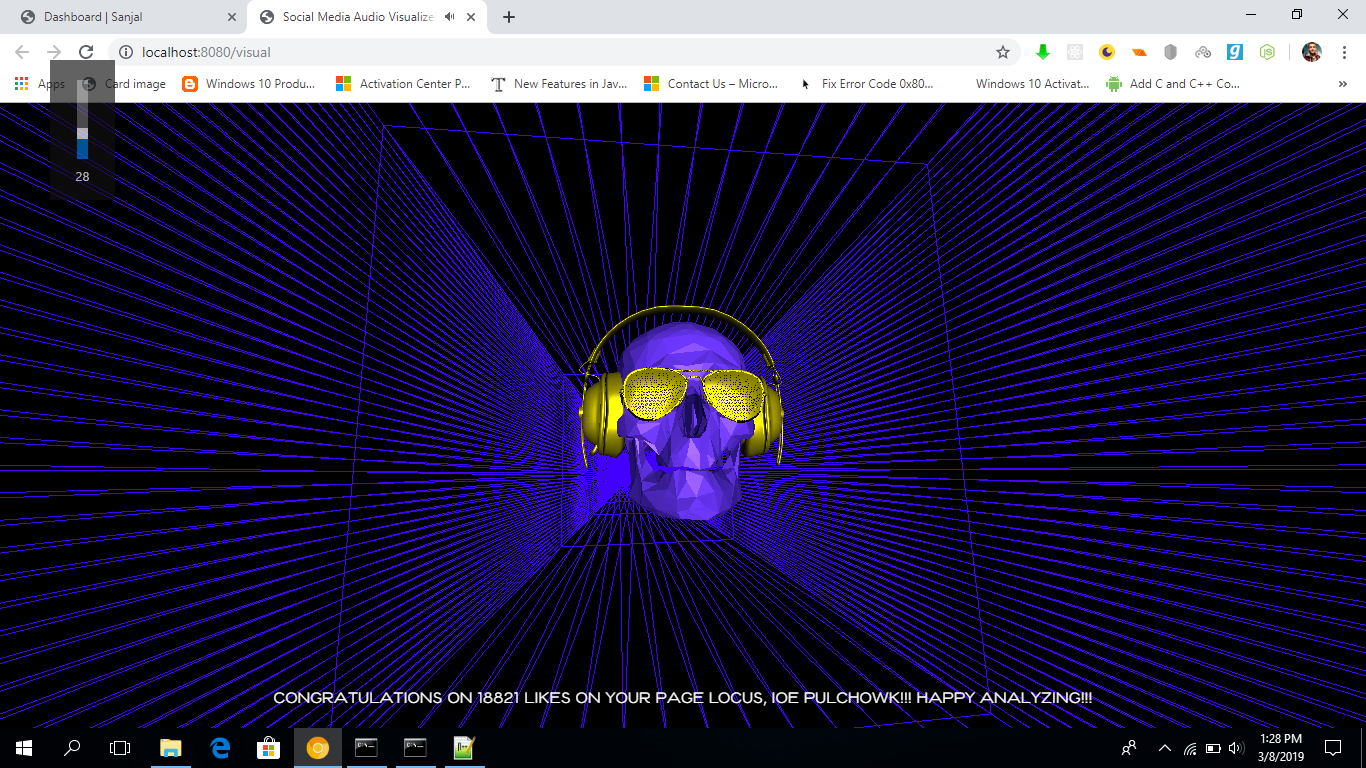


Fig 8: VISUALIZATION SCREEN

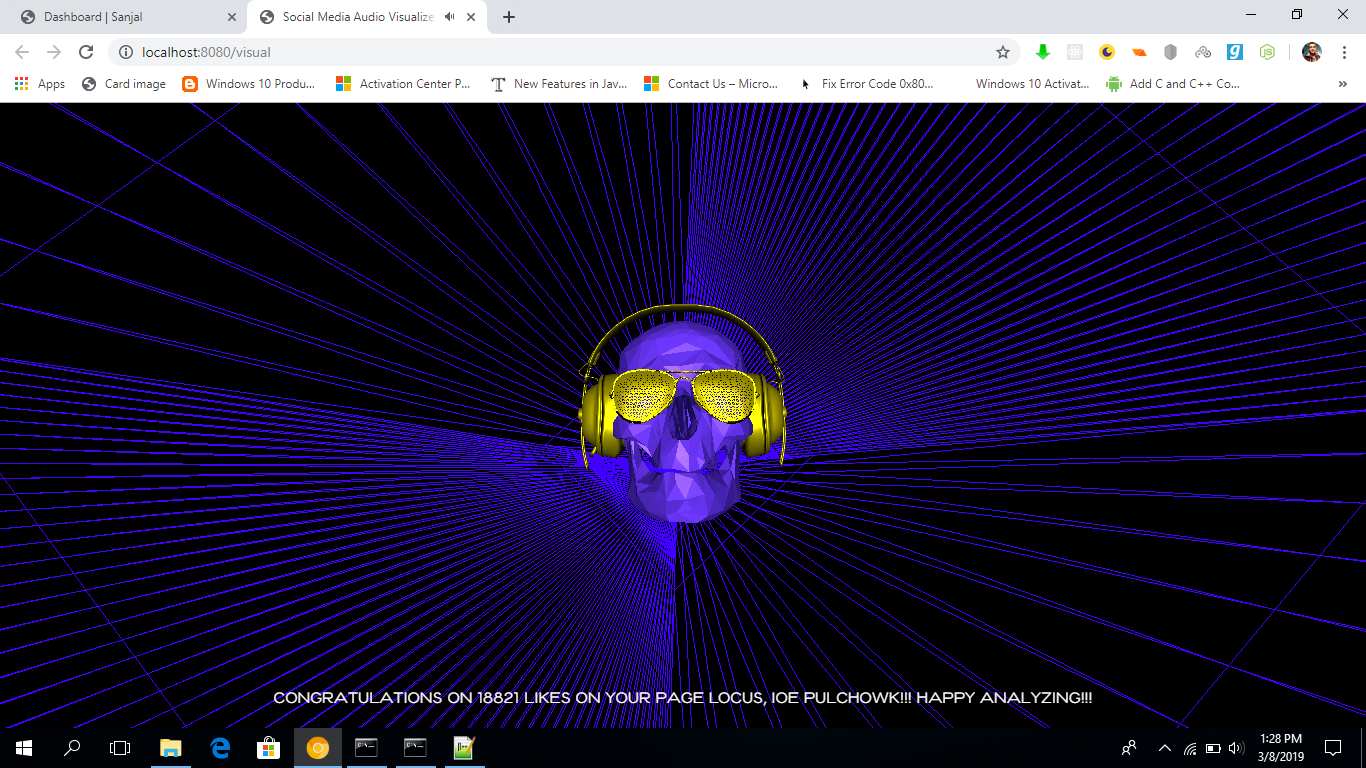


Fig 9: VISULAIZATION SCREEN 2

Once the sample or the file is choosen ,the visulalization screen start to play the audio along with the visual effects.

# CONCLUSION

## Project’s Conclusion

The project has presented a simple visualizer of audio along with the statistics in a very well graphical manner. It used graphics objects to make the visualizer interactive . it uses the social media data to find a statistical data and then plotting and displaying was done through web services.

## Limitation

The limitation of this project is that it is only able to display the statistics of facebook .as the facebook only access to its data to limited extent. Also the audio visulalizer works with bass and tremble of sound. It can be modified to sense very sensitive parts of sound to make it more effective.

## Future Enhancement

This project can be further advanced to display the audio of various social media sites like instagram,twitter,etc .the visualizer can be made more interactive with enchanced graphics . the visualizer can be used to represent the statistical values in interactive moving objects.such as a man moving on a bar graph , moving around pie chart ,etc which can be very useful for learning as well as understanding purpose. The children can be taught easily through such graph.

## Recommendation

This project is a decent enough guide to get started in computer graphics and it deals with

Creating graphical object using blender. r. It used graphics objects to make the visualizer interactive . it uses the social media data to find a statistical data and then plotting and displaying was done through web services.

**APPENDIX**

**OrbitControl.js**

/\*\*

\* @author qiao / https://github.com/qiao

\* @author mrdoob / http://mrdoob.com

\* @author alteredq / http://alteredqualia.com/

\* @author WestLangley / http://github.com/WestLangley

\* @author erich666 / http://erichaines.com

\*/

// This set of controls performs orbiting, dollying (zooming), and panning.

// Unlike TrackballControls, it maintains the "up" direction object.up (+Y by default).

//

// Orbit - left mouse / touch: one finger move

// Zoom - middle mouse, or mousewheel / touch: two finger spread or squish

// Pan - right mouse, or arrow keys / touch: three finger swipe

THREE.OrbitControls = function ( object, domElement ) {

this.object = object;

this.domElement = ( domElement !== undefined ) ? domElement : document;

// Set to false to disable this control

this.enabled = true;

// "target" sets the location of focus, where the object orbits around

this.target = new THREE.Vector3();

// How far you can dolly in and out ( PerspectiveCamera only )

this.minDistance = 0;

this.maxDistance = Infinity;

// How far you can zoom in and out ( OrthographicCamera only )

this.minZoom = 0;

this.maxZoom = Infinity;

// How far you can orbit vertically, upper and lower limits.

// Range is 0 to Math.PI radians.

this.minPolarAngle = 0; // radians

this.maxPolarAngle = Math.PI; // radians

// How far you can orbit horizontally, upper and lower limits.

// If set, must be a sub-interval of the interval [ - Math.PI, Math.PI ].

this.minAzimuthAngle = - Infinity; // radians

this.maxAzimuthAngle = Infinity; // radians

// Set to true to enable damping (inertia)

// If damping is enabled, you must call controls.update() in your animation loop

this.enableDamping = false;

this.dampingFactor = 0.25;

// This option actually enables dollying in and out; left as "zoom" for backwards compatibility.

// Set to false to disable zooming

this.enableZoom = true;

this.zoomSpeed = 1.0;

// Set to false to disable rotating

this.enableRotate = true;

this.rotateSpeed = 1.0;

// Set to false to disable panning

this.enablePan = true;

this.keyPanSpeed = 7.0; // pixels moved per arrow key push

// Set to true to automatically rotate around the target

// If auto-rotate is enabled, you must call controls.update() in your animation loop

this.autoRotate = false;

this.autoRotateSpeed = 2.0; // 30 seconds per round when fps is 60

// Set to false to disable use of the keys

this.enableKeys = true;

// The four arrow keys

this.keys = { LEFT: 37, UP: 38, RIGHT: 39, BOTTOM: 40 };

// Mouse buttons

this.mouseButtons = { ORBIT: THREE.MOUSE.LEFT, ZOOM: THREE.MOUSE.MIDDLE, PAN: THREE.MOUSE.RIGHT };

// for reset

this.target0 = this.target.clone();

this.position0 = this.object.position.clone();

this.zoom0 = this.object.zoom;

//

// public methods

//

this.getPolarAngle = function () {

return spherical.phi;

};

this.getAzimuthalAngle = function () {

return spherical.theta;

};

this.saveState = function () {

scope.target0.copy( scope.target );

scope.position0.copy( scope.object.position );

scope.zoom0 = scope.object.zoom;

};

this.reset = function () {

scope.target.copy( scope.target0 );

scope.object.position.copy( scope.position0 );

scope.object.zoom = scope.zoom0;

scope.object.updateProjectionMatrix();

scope.dispatchEvent( changeEvent );

scope.update();

state = STATE.NONE;

};

// this method is exposed, but perhaps it would be better if we can make it private...

this.update = function () {

var offset = new THREE.Vector3();

// so camera.up is the orbit axis

var quat = new THREE.Quaternion().setFromUnitVectors( object.up, new THREE.Vector3( 0, 1, 0 ) );

var quatInverse = quat.clone().inverse();

var lastPosition = new THREE.Vector3();

var lastQuaternion = new THREE.Quaternion();

return function update() {

var position = scope.object.position;

offset.copy( position ).sub( scope.target );

// rotate offset to "y-axis-is-up" space

offset.applyQuaternion( quat );

// angle from z-axis around y-axis

spherical.setFromVector3( offset );

if ( scope.autoRotate && state === STATE.NONE ) {

rotateLeft( getAutoRotationAngle() );

}

spherical.theta += sphericalDelta.theta;

spherical.phi += sphericalDelta.phi;

// restrict theta to be between desired limits

spherical.theta = Math.max( scope.minAzimuthAngle, Math.min( scope.maxAzimuthAngle, spherical.theta ) );

// restrict phi to be between desired limits

spherical.phi = Math.max( scope.minPolarAngle, Math.min( scope.maxPolarAngle, spherical.phi ) );

spherical.makeSafe();

spherical.radius \*= scale;

// restrict radius to be between desired limits

spherical.radius = Math.max( scope.minDistance, Math.min( scope.maxDistance, spherical.radius ) );

// move target to panned location

scope.target.add( panOffset );

offset.setFromSpherical( spherical );

// rotate offset back to "camera-up-vector-is-up" space

offset.applyQuaternion( quatInverse );

position.copy( scope.target ).add( offset );

scope.object.lookAt( scope.target );

if ( scope.enableDamping === true ) {

sphericalDelta.theta \*= ( 1 - scope.dampingFactor );

sphericalDelta.phi \*= ( 1 - scope.dampingFactor );

} else {

sphericalDelta.set( 0, 0, 0 );

}

scale = 1;

panOffset.set( 0, 0, 0 );

// update condition is:

// min(camera displacement, camera rotation in radians)^2 > EPS

// using small-angle approximation cos(x/2) = 1 - x^2 / 8

if ( zoomChanged ||

lastPosition.distanceToSquared( scope.object.position ) > EPS ||

8 \* ( 1 - lastQuaternion.dot( scope.object.quaternion ) ) > EPS ) {

scope.dispatchEvent( changeEvent );

lastPosition.copy( scope.object.position );

lastQuaternion.copy( scope.object.quaternion );

zoomChanged = false;

return true;

}

return false;

};

}();

this.dispose = function () {

scope.domElement.removeEventListener( 'contextmenu', onContextMenu, false );

scope.domElement.removeEventListener( 'mousedown', onMouseDown, false );

scope.domElement.removeEventListener( 'wheel', onMouseWheel, false );

scope.domElement.removeEventListener( 'touchstart', onTouchStart, false );

scope.domElement.removeEventListener( 'touchend', onTouchEnd, false );

scope.domElement.removeEventListener( 'touchmove', onTouchMove, false );

document.removeEventListener( 'mousemove', onMouseMove, false );

document.removeEventListener( 'mouseup', onMouseUp, false );

window.removeEventListener( 'keydown', onKeyDown, false );

//scope.dispatchEvent( { type: 'dispose' } ); // should this be added here?

};

//

// internals

//

var scope = this;

var changeEvent = { type: 'change' };

var startEvent = { type: 'start' };

var endEvent = { type: 'end' };

var STATE = { NONE: - 1, ROTATE: 0, DOLLY: 1, PAN: 2, TOUCH\_ROTATE: 3, TOUCH\_DOLLY: 4, TOUCH\_PAN: 5 };

var state = STATE.NONE;

var EPS = 0.000001;

// current position in spherical coordinates

var spherical = new THREE.Spherical();

var sphericalDelta = new THREE.Spherical();

var scale = 1;

var panOffset = new THREE.Vector3();

var zoomChanged = false;

var rotateStart = new THREE.Vector2();

var rotateEnd = new THREE.Vector2();

var rotateDelta = new THREE.Vector2();

var panStart = new THREE.Vector2();

var panEnd = new THREE.Vector2();

var panDelta = new THREE.Vector2();

var dollyStart = new THREE.Vector2();

var dollyEnd = new THREE.Vector2();

var dollyDelta = new THREE.Vector2();

function getAutoRotationAngle() {

return 2 \* Math.PI / 60 / 60 \* scope.autoRotateSpeed;

}

function getZoomScale() {

return Math.pow( 0.95, scope.zoomSpeed );

}

function rotateLeft( angle ) {

sphericalDelta.theta -= angle;

}

function rotateUp( angle ) {

sphericalDelta.phi -= angle;

}

var panLeft = function () {

var v = new THREE.Vector3();

return function panLeft( distance, objectMatrix ) {

v.setFromMatrixColumn( objectMatrix, 0 ); // get X column of objectMatrix

v.multiplyScalar( - distance );

panOffset.add( v );

};

}();

var panUp = function () {

var v = new THREE.Vector3();

return function panUp( distance, objectMatrix ) {

v.setFromMatrixColumn( objectMatrix, 1 ); // get Y column of objectMatrix

v.multiplyScalar( distance );

panOffset.add( v );

};

}();

// deltaX and deltaY are in pixels; right and down are positive

var pan = function () {

var offset = new THREE.Vector3();

return function pan( deltaX, deltaY ) {

var element = scope.domElement === document ? scope.domElement.body : scope.domElement;

if ( scope.object instanceof THREE.PerspectiveCamera ) {

// perspective

var position = scope.object.position;

offset.copy( position ).sub( scope.target );

var targetDistance = offset.length();

// half of the fov is center to top of screen

targetDistance \*= Math.tan( ( scope.object.fov / 2 ) \* Math.PI / 180.0 );

// we actually don't use screenWidth, since perspective camera is fixed to screen height

panLeft( 2 \* deltaX \* targetDistance / element.clientHeight, scope.object.matrix );

panUp( 2 \* deltaY \* targetDistance / element.clientHeight, scope.object.matrix );

} else if ( scope.object instanceof THREE.OrthographicCamera ) {

// orthographic

panLeft( deltaX \* ( scope.object.right - scope.object.left ) / scope.object.zoom / element.clientWidth, scope.object.matrix );

panUp( deltaY \* ( scope.object.top - scope.object.bottom ) / scope.object.zoom / element.clientHeight, scope.object.matrix );

} else {

// camera neither orthographic nor perspective

console.warn( 'WARNING: OrbitControls.js encountered an unknown camera type - pan disabled.' );

scope.enablePan = false;

}

};

}();

function dollyIn( dollyScale ) {

if ( scope.object instanceof THREE.PerspectiveCamera ) {

scale /= dollyScale;

} else if ( scope.object instanceof THREE.OrthographicCamera ) {

scope.object.zoom = Math.max( scope.minZoom, Math.min( scope.maxZoom, scope.object.zoom \* dollyScale ) );

scope.object.updateProjectionMatrix();

zoomChanged = true;

} else {

console.warn( 'WARNING: OrbitControls.js encountered an unknown camera type - dolly/zoom disabled.' );

scope.enableZoom = false;

}

}

function dollyOut( dollyScale ) {

if ( scope.object instanceof THREE.PerspectiveCamera ) {

scale \*= dollyScale;

} else if ( scope.object instanceof THREE.OrthographicCamera ) {

scope.object.zoom = Math.max( scope.minZoom, Math.min( scope.maxZoom, scope.object.zoom / dollyScale ) );

scope.object.updateProjectionMatrix();

zoomChanged = true;

} else {

console.warn( 'WARNING: OrbitControls.js encountered an unknown camera type - dolly/zoom disabled.' );

scope.enableZoom = false;

}

}

//

// event callbacks - update the object state

//

function handleMouseDownRotate( event ) {

//console.log( 'handleMouseDownRotate' );

rotateStart.set( event.clientX, event.clientY );

}

function handleMouseDownDolly( event ) {

//console.log( 'handleMouseDownDolly' );

dollyStart.set( event.clientX, event.clientY );

}

function handleMouseDownPan( event ) {

//console.log( 'handleMouseDownPan' );

panStart.set( event.clientX, event.clientY );

}

function handleMouseMoveRotate( event ) {

//console.log( 'handleMouseMoveRotate' );

rotateEnd.set( event.clientX, event.clientY );

rotateDelta.subVectors( rotateEnd, rotateStart );

var element = scope.domElement === document ? scope.domElement.body : scope.domElement;

// rotating across whole screen goes 360 degrees around

rotateLeft( 2 \* Math.PI \* rotateDelta.x / element.clientWidth \* scope.rotateSpeed );

// rotating up and down along whole screen attempts to go 360, but limited to 180

rotateUp( 2 \* Math.PI \* rotateDelta.y / element.clientHeight \* scope.rotateSpeed );

rotateStart.copy( rotateEnd );

scope.update();

}

function handleMouseMoveDolly( event ) {

//console.log( 'handleMouseMoveDolly' );

dollyEnd.set( event.clientX, event.clientY );

dollyDelta.subVectors( dollyEnd, dollyStart );

if ( dollyDelta.y > 0 ) {

dollyIn( getZoomScale() );

} else if ( dollyDelta.y < 0 ) {

dollyOut( getZoomScale() );

}

dollyStart.copy( dollyEnd );

scope.update();

}

function handleMouseMovePan( event ) {

//console.log( 'handleMouseMovePan' );

panEnd.set( event.clientX, event.clientY );

panDelta.subVectors( panEnd, panStart );

pan( panDelta.x, panDelta.y );

panStart.copy( panEnd );

scope.update();

}

function handleMouseUp( event ) {

// console.log( 'handleMouseUp' );

}

function handleMouseWheel( event ) {

// console.log( 'handleMouseWheel' );

if ( event.deltaY < 0 ) {

dollyOut( getZoomScale() );

} else if ( event.deltaY > 0 ) {

dollyIn( getZoomScale() );

}

scope.update();

}

function handleKeyDown( event ) {

//console.log( 'handleKeyDown' );

switch ( event.keyCode ) {

case scope.keys.UP:

pan( 0, scope.keyPanSpeed );

scope.update();

break;

case scope.keys.BOTTOM:

pan( 0, - scope.keyPanSpeed );

scope.update();

break;

case scope.keys.LEFT:

pan( scope.keyPanSpeed, 0 );

scope.update();

break;

case scope.keys.RIGHT:

pan( - scope.keyPanSpeed, 0 );

scope.update();

break;

}

}

function handleTouchStartRotate( event ) {

//console.log( 'handleTouchStartRotate' );

rotateStart.set( event.touches[ 0 ].pageX, event.touches[ 0 ].pageY );

}

function handleTouchStartDolly( event ) {

//console.log( 'handleTouchStartDolly' );

var dx = event.touches[ 0 ].pageX - event.touches[ 1 ].pageX;

var dy = event.touches[ 0 ].pageY - event.touches[ 1 ].pageY;

var distance = Math.sqrt( dx \* dx + dy \* dy );

dollyStart.set( 0, distance );

}

function handleTouchStartPan( event ) {

//console.log( 'handleTouchStartPan' );

panStart.set( event.touches[ 0 ].pageX, event.touches[ 0 ].pageY );

}

function handleTouchMoveRotate( event ) {

//console.log( 'handleTouchMoveRotate' );

rotateEnd.set( event.touches[ 0 ].pageX, event.touches[ 0 ].pageY );

rotateDelta.subVectors( rotateEnd, rotateStart );

var element = scope.domElement === document ? scope.domElement.body : scope.domElement;

// rotating across whole screen goes 360 degrees around

rotateLeft( 2 \* Math.PI \* rotateDelta.x / element.clientWidth \* scope.rotateSpeed );

// rotating up and down along whole screen attempts to go 360, but limited to 180

rotateUp( 2 \* Math.PI \* rotateDelta.y / element.clientHeight \* scope.rotateSpeed );

rotateStart.copy( rotateEnd );

scope.update();

}

function handleTouchMoveDolly( event ) {

//console.log( 'handleTouchMoveDolly' );

var dx = event.touches[ 0 ].pageX - event.touches[ 1 ].pageX;

var dy = event.touches[ 0 ].pageY - event.touches[ 1 ].pageY;

var distance = Math.sqrt( dx \* dx + dy \* dy );

dollyEnd.set( 0, distance );

dollyDelta.subVectors( dollyEnd, dollyStart );

if ( dollyDelta.y > 0 ) {

dollyOut( getZoomScale() );

} else if ( dollyDelta.y < 0 ) {

dollyIn( getZoomScale() );

}

dollyStart.copy( dollyEnd );

scope.update();

}

function handleTouchMovePan( event ) {

//console.log( 'handleTouchMovePan' );

panEnd.set( event.touches[ 0 ].pageX, event.touches[ 0 ].pageY );

panDelta.subVectors( panEnd, panStart );

pan( panDelta.x, panDelta.y );

panStart.copy( panEnd );

scope.update();

}

function handleTouchEnd( event ) {

//console.log( 'handleTouchEnd' );

}

//

// event handlers - FSM: listen for events and reset state

//

function onMouseDown( event ) {

if ( scope.enabled === false ) return;

event.preventDefault();

switch ( event.button ) {

case scope.mouseButtons.ORBIT:

if ( scope.enableRotate === false ) return;

handleMouseDownRotate( event );

state = STATE.ROTATE;

break;

case scope.mouseButtons.ZOOM:

if ( scope.enableZoom === false ) return;

handleMouseDownDolly( event );

state = STATE.DOLLY;

break;

case scope.mouseButtons.PAN:

if ( scope.enablePan === false ) return;

handleMouseDownPan( event );

state = STATE.PAN;

break;

}

if ( state !== STATE.NONE ) {

document.addEventListener( 'mousemove', onMouseMove, false );

document.addEventListener( 'mouseup', onMouseUp, false );

scope.dispatchEvent( startEvent );

}

}

function onMouseMove( event ) {

if ( scope.enabled === false ) return;

event.preventDefault();

switch ( state ) {

case STATE.ROTATE:

if ( scope.enableRotate === false ) return;

handleMouseMoveRotate( event );

break;

case STATE.DOLLY:

if ( scope.enableZoom === false ) return;

handleMouseMoveDolly( event );

break;

case STATE.PAN:

if ( scope.enablePan === false ) return;

handleMouseMovePan( event );

break;

}

}

function onMouseUp( event ) {

if ( scope.enabled === false ) return;

handleMouseUp( event );

document.removeEventListener( 'mousemove', onMouseMove, false );

document.removeEventListener( 'mouseup', onMouseUp, false );

scope.dispatchEvent( endEvent );

state = STATE.NONE;

}

function onMouseWheel( event ) {

if ( scope.enabled === false || scope.enableZoom === false || ( state !== STATE.NONE && state !== STATE.ROTATE ) ) return;

event.preventDefault();

event.stopPropagation();

handleMouseWheel( event );

scope.dispatchEvent( startEvent ); // not sure why these are here...

scope.dispatchEvent( endEvent );

}

function onKeyDown( event ) {

if ( scope.enabled === false || scope.enableKeys === false || scope.enablePan === false ) return;

handleKeyDown( event );

}

function onTouchStart( event ) {

if ( scope.enabled === false ) return;

switch ( event.touches.length ) {

case 1: // one-fingered touch: rotate

if ( scope.enableRotate === false ) return;

handleTouchStartRotate( event );

state = STATE.TOUCH\_ROTATE;

break;

case 2: // two-fingered touch: dolly

if ( scope.enableZoom === false ) return;

handleTouchStartDolly( event );

state = STATE.TOUCH\_DOLLY;

break;

case 3: // three-fingered touch: pan

if ( scope.enablePan === false ) return;

handleTouchStartPan( event );

state = STATE.TOUCH\_PAN;

break;

default:

state = STATE.NONE;

}

if ( state !== STATE.NONE ) {

scope.dispatchEvent( startEvent );

}

}

function onTouchMove( event ) {

if ( scope.enabled === false ) return;

event.preventDefault();

event.stopPropagation();

switch ( event.touches.length ) {

case 1: // one-fingered touch: rotate

if ( scope.enableRotate === false ) return;

if ( state !== STATE.TOUCH\_ROTATE ) return; // is this needed?...

handleTouchMoveRotate( event );

break;

case 2: // two-fingered touch: dolly

if ( scope.enableZoom === false ) return;

if ( state !== STATE.TOUCH\_DOLLY ) return; // is this needed?...

handleTouchMoveDolly( event );

break;

case 3: // three-fingered touch: pan

if ( scope.enablePan === false ) return;

if ( state !== STATE.TOUCH\_PAN ) return; // is this needed?...

handleTouchMovePan( event );

break;

default:

state = STATE.NONE;

}

}

function onTouchEnd( event ) {

if ( scope.enabled === false ) return;

handleTouchEnd( event );

scope.dispatchEvent( endEvent );

state = STATE.NONE;

}

function onContextMenu( event ) {

if ( scope.enabled === false ) return;

event.preventDefault();

}

//

scope.domElement.addEventListener( 'contextmenu', onContextMenu, false );

scope.domElement.addEventListener( 'mousedown', onMouseDown, false );

scope.domElement.addEventListener( 'wheel', onMouseWheel, false );

scope.domElement.addEventListener( 'touchstart', onTouchStart, false );

scope.domElement.addEventListener( 'touchend', onTouchEnd, false );

scope.domElement.addEventListener( 'touchmove', onTouchMove, false );

window.addEventListener( 'keydown', onKeyDown, false );

// force an update at start

this.update();

};

THREE.OrbitControls.prototype = Object.create( THREE.EventDispatcher.prototype );

THREE.OrbitControls.prototype.constructor = THREE.OrbitControls;

Object.defineProperties( THREE.OrbitControls.prototype, {

center: {

get: function () {

console.warn( 'THREE.OrbitControls: .center has been renamed to .target' );

return this.target;

}

},

// backward compatibility

noZoom: {

get: function () {

console.warn( 'THREE.OrbitControls: .noZoom has been deprecated. Use .enableZoom instead.' );

return ! this.enableZoom;

},

set: function ( value ) {

console.warn( 'THREE.OrbitControls: .noZoom has been deprecated. Use .enableZoom instead.' );

this.enableZoom = ! value;

}

},

noRotate: {

get: function () {

console.warn( 'THREE.OrbitControls: .noRotate has been deprecated. Use .enableRotate instead.' );

return ! this.enableRotate;

},

set: function ( value ) {

console.warn( 'THREE.OrbitControls: .noRotate has been deprecated. Use .enableRotate instead.' );

this.enableRotate = ! value;

}

},

noPan: {

get: function () {

console.warn( 'THREE.OrbitControls: .noPan has been deprecated. Use .enablePan instead.' );

return ! this.enablePan;

},

set: function ( value ) {

console.warn( 'THREE.OrbitControls: .noPan has been deprecated. Use .enablePan instead.' );

this.enablePan = ! value;

}

},

noKeys: {

get: function () {

console.warn( 'THREE.OrbitControls: .noKeys has been deprecated. Use .enableKeys instead.' );

return ! this.enableKeys;

},

set: function ( value ) {

console.warn( 'THREE.OrbitControls: .noKeys has been deprecated. Use .enableKeys instead.' );

this.enableKeys = ! value;

}

},

staticMoving: {

get: function () {

console.warn( 'THREE.OrbitControls: .staticMoving has been deprecated. Use .enableDamping instead.' );

return ! this.enableDamping;

},

set: function ( value ) {

console.warn( 'THREE.OrbitControls: .staticMoving has been deprecated. Use .enableDamping instead.' );

this.enableDamping = ! value;

}

},

dynamicDampingFactor: {

get: function () {

console.warn( 'THREE.OrbitControls: .dynamicDampingFactor has been renamed. Use .dampingFactor instead.' );

return this.dampingFactor;

},

set: function ( value ) {

console.warn( 'THREE.OrbitControls: .dynamicDampingFactor has been renamed. Use .dampingFactor instead.' );

this.dampingFactor = value;

}

}

} );