# PROYECTO FINAL – FACILITACIÓN DE CREACIÓN DE CGI

ADRIÁN ALBERTO GUTIÉRREZ LEAL
JUAN JOSE MADRIGAL PALACIO

DIAMOND ENGINE

JUAN GONZALO CARCAMO ZULUAGA

UNIVERSIDAD EAFIT
ESCUELA DE CIENCIAS APLICADAS E INGENIERÍA
COMPUTACIÓN GRÁFICA
MEDELLÍN
2022

# INTRODUCCIÓN

En las últimas décadas, el CGI ha tenido un enorme impacto, en especial con sus distintas aplicaciones en numerosos sectores, como el cine, arquitectura, medicina y videojuegos. Computer Generated Imagery o, por sus siglas, CGI, significa justo eso, imágenes generadas por computadora. En otras palabras, es el uso de gráficos por computadora en escenarios como el arte u otros medios visuales. Esta, puede ser tanto en 2D como 3D, ya sean objetos o cosas totalmente animados, fondos, efectos y partículas, incluso llegando al renderizado de personajes y mundos totalmente virtuales.

Sobre su creación, puede hacerse de distintas formas, como el uso de algoritmos para generar patrones fractales complejos, editores de imágenes 2D basados en pixeles que pueden producir formas vectoriales o software de gráficos 3D generados a partir de formas simples como triángulos planos, para crear objetos complejos. De todo ello, se decidió hacer una aproximación más cercana al último.

## PROBLEMA

La creación y aplicación del CGI, aunque llamativa, es un trabajo bastante arduo y costoso. De acuerdo con el Nashville Filmical Institute, desde una perspectiva meramente del entretenimiento, el CGI involucra un proceso muy largo, que es tanto retador como altamente técnico, con equipos bastante grandes y diversos, cada uno con un rol crucial en diseñar los mejores efectos visuales para su cliente, incluyendo desde ilustradores hasta codificadores.

Aunado a ello, más allá de la creatividad necesaria para concebir una imagen, el CGI requiere de extensivo conocimiento técnico, además de enormes equipos (casi siempre, de miles de artistas). En este punto, surge una nueva dificultad: aunque el CGI avance en las películas o videojuegos, la audiencia también lo hace, siendo capaz de detectar fallas en el mismo. No obstante, este último aspecto no será de central importancia para el desarrollo del proyecto.

Por otro lado, resulta relevante discutir sus costos. Aunque usar CGI para hacer una película, clip u otro medio podría ahorrar tiempo y dinero, en especial a la hora de generar escenas difíciles o imposibles, como batallas en el espacio o dinosaurios en un parque, herramientas como el hardware, software y dispositivos como pantallas verdes o cámaras, cuestan dinero.

Desde una perspectiva proyectizada, claramente dependiendo de sus características y del tamaño del proyecto y calidad esperada, pueden estimarse los siguientes presupuestos:

- $3000 - $7000 Por minuto. Aún considerado bajo presupuesto.
- $10000 - $20000 Por minuto. Aproximadamente el mínimo de un video animado producido profesionalmente.
- $50000 - $100000+ Por minuto. A partir de este rango pueden esperarse valores de producción de calidad.

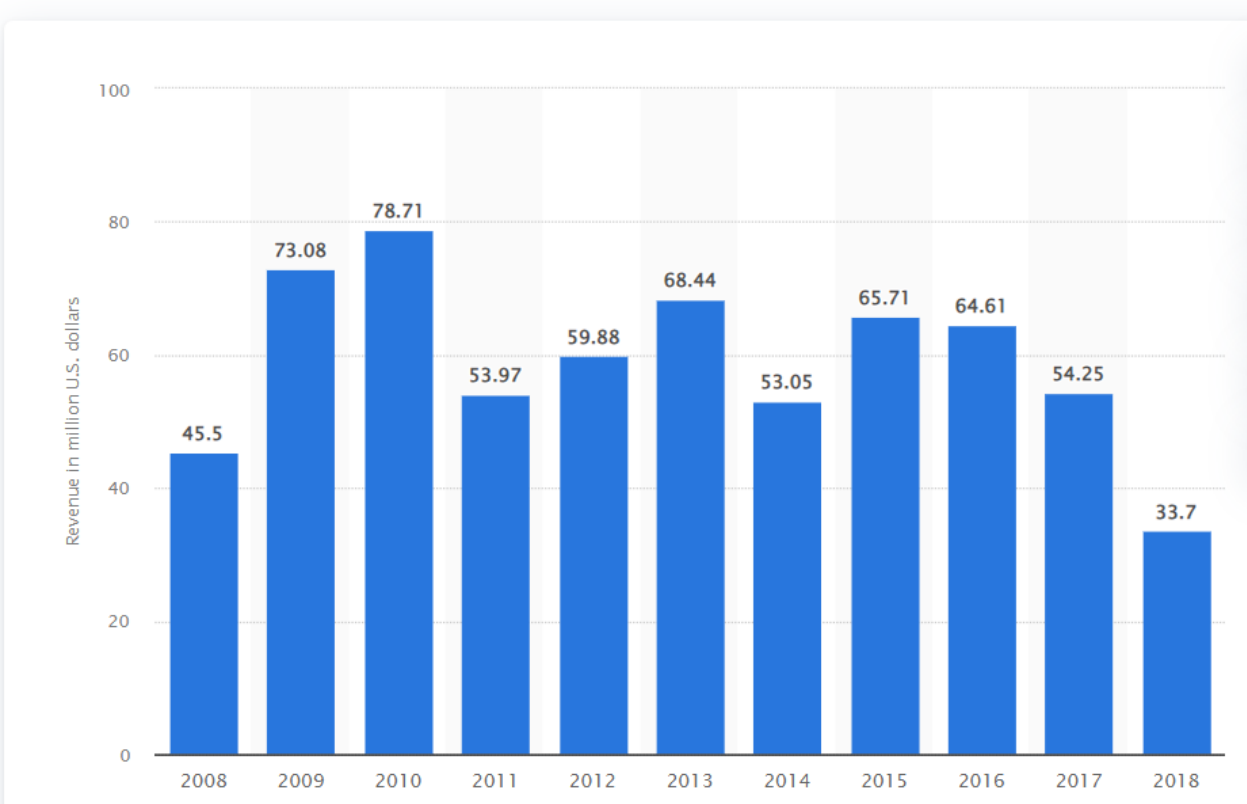
De acuerdo con Garret Parker (2016), en un artículo para MoneyInc acerca del costo promedio de un episodio de Game of Thrones, el costo el CGI es significativo por 2 razones: talento humano y recursos computacionales.

Acerca del talento humano, de acuerdo con el autor, la industria del CGI actúa como una “línea de ensamblaje”, con un tiempo promedio por equipo de al menos 4 semanas, estimando unas 1600 horas/hombre como mínimo sin tiempos extra, a unos 50USD por hora (por cada persona), se llega a un mínimo de 80000USD por minuto. Es decir, si un episodio de Game of Thrones tiene solo 10 minutos de CGI, esto le cuesta al estudio 800000USD.

Dejando el recurso humano de lado y, desde una perspectiva más computacional, la producción de CGI es extremadamente demandante. Un solo cuadro puede tardar alrededor de 12 horas en renderizar, según su complejidad. Para ponerlo en contexto, a 45fps (cuadros por segundo), se necesitarían 540 horas para completar un solo segundo. Además, el uso de computación distribuida para mejorar tiempos y ahorrar recursos (en AWS, por ejemplo) cuesta aproximadamente 0.65USD por hora, lo cual suena bastante asequible, pero, con la cantidad de horas necesarias,

suma más o menos 7000USD, que fácilmente podría alcanzar los millones de dólares dependiendo del tiempo de CGI que se requiera.

Sobre este tema, José Navarro (2018), para el portal web Statista, publicó un estudio en el que se muestra el costo promedio de CGI por película (en millones de dólares) por año.

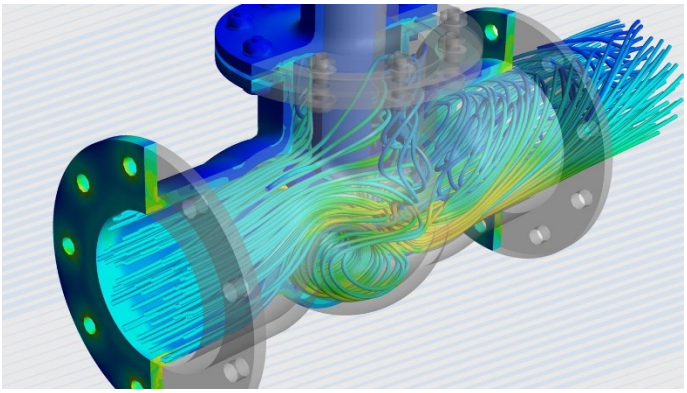



[Figura 1.1 Costos CGI Recuperado de https://www.statista.com/statistics/1020957/cgi-animated-movie-production-costs-us/]

Luego de este estudio, el autor pudo concluir que, solo en la industria del cine, se gastaban en promedio 33.7 millones de dólares para el 2018 por película. Claramente, esta inversión realizada por los grandes estudios no es posible para una persona del común que solo desea realizar animaciones como pasatiempo o tiene curiosidad al respecto. Más aún, sabiendo que se necesita de ciertas calificaciones para lograr un trabajo de calidad, llegar a conocer el software y adquirir una cantidad decente de conocimientos sobre el (véase Blender) se lleva una buena cantidad de tiempo en practicar y aprender, llevado a extremos al saber que para proyectos profesionales se tienen equipos grandes, con especialistas en distintos campos y habilidades.

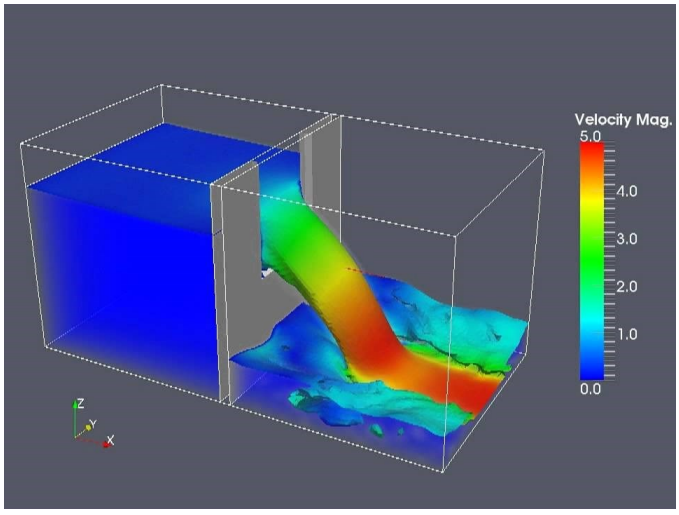
# ESTADO DEL ARTE

En el mercado, encontramos distintas soluciones a la aplicación del CGI en distintos sectores. En el área de mecánica de fluidos, de especial importancia para la ingeniería, ya que permite el movimiento de fluidos (por ejemplo, a través de tuberías), lograr una simulación no es fácil y, aunque existen distintos programas y aplicaciones que permiten realizar esta simulación, estos son bastante costosos, siendo incluso una licencia individual para acceder a un buen sistema profesional muy valiosa de adquirir.



[Figura 2.0. Mecánica de Fluidos]

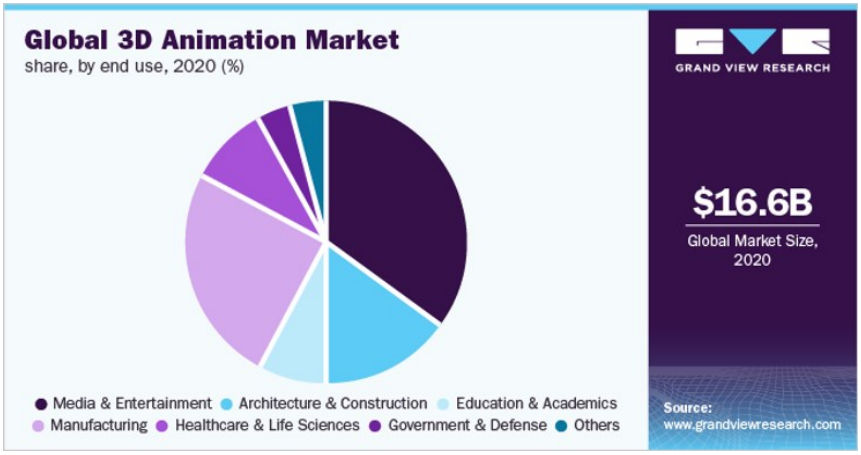
Una de las soluciones encontradas en el mercado podría ser OpenFOAM, una solución gratuita, con licencia Open Source, la cual permite realizar todos los esquemas de volumen finito. No obstante, cuenta con una curva de aprendizaje bastante pronunciada para su uso.




[Figura 2.1. OpenFOAM]

Luego, encontramos uno de los sectores más productivos en materia de CGI: el del entretenimiento. Varios estudios, como el realizado en junio de 2021 por Grand View Research,

han confirmado que al menos un tercio del uso de CGI se destina a productos mediáticos y de entretenimiento. Gráficamente:




[Figura 2.2. Uso de CGI]

En este punto, se da a conocer una necesidad latente: conseguir un realismo de la animación y los efectos CGI en las películas es muy complicado y requiere de una enorme inversión económica, debido a todo el detalle que se debe tener en cuenta. Hay mucho a lograr con tal de dar más realismo a una película, cosas tan simples como el cabello o lograr la fluidez de la animación no son tareas fáciles para los productores. En este frente, Disney y Pixar son algunos de los mejores exponentes.

Para mejorar esto Disney creó su Framework: El primero en usar métodos para estimar los parámetros para la dinámica del cabello y ajusta los parámetros al cabello capturado en movimiento, esto lo usan para volver a simular el cabello capturado y también para editar la animación cambiando el movimiento de la cabeza, las propiedades físicas del cabello, como la humedad y las fuerzas externas.

[Figura 2.3. Dinámica de Cabello]

El movimiento es otro ámbito difícil en el mismo por lo cual, en el caso de Pixar, a lo largo de su historia han creado diferentes frameworks y sistemas los cuales mejoran con cada película y les permiten crear distintos controles sobre los personajes y cómo estos pueden reaccionar o moverse según la situación o el personaje que estén animando. Han llegado a tener para un solo personaje

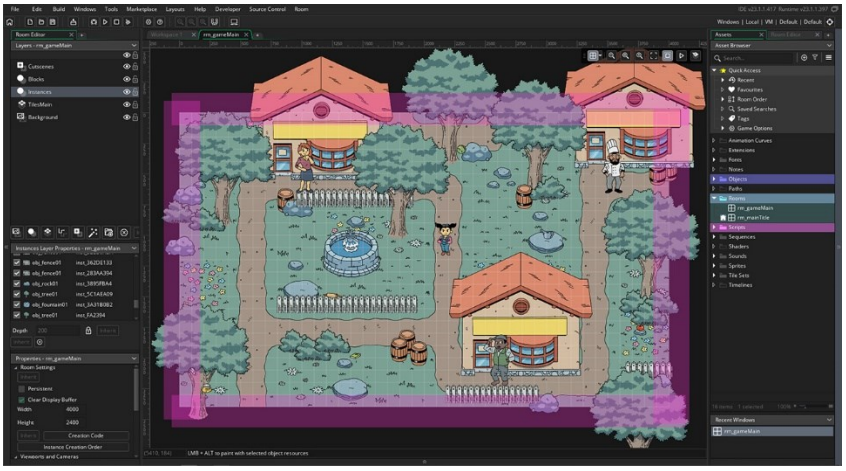
más de 1000 controles en general para lograr fluidez en sus movimientos, llegando actualmente algunos incluso a los 7000 controles.

[Figura 2.4. Fluidez de Movimiento]

En el mismo contexto del entretenimiento, además de las películas, está el CGI en videojuegos. Crear videojuegos no es una tarea sencilla, toda la programación y diseño que hay detrás de un buen videojuego no se hace en un solo día, además de que el software y los elementos que se usan en su creación muchas veces no son baratos, o aquellos creados por las compañías no fueron sencillos de crear. A pesar de esto, actualmente casi cualquiera puede hacer un videojuego con las herramientas adecuadas. Estas herramientas se conocen como “Motores de Videojuegos”.

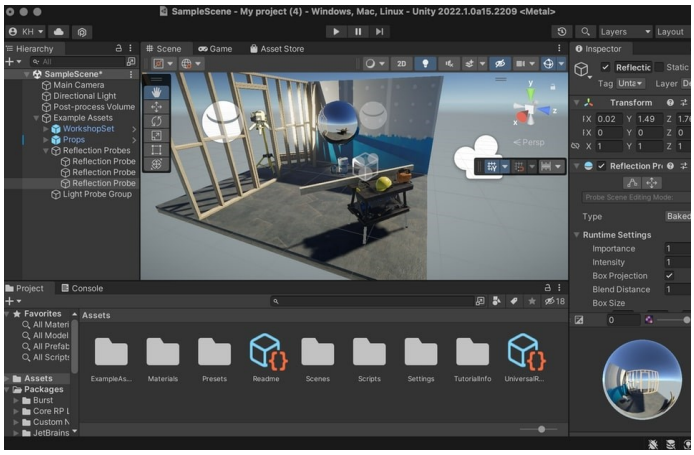
Algunos ejemplos claros de motores de videojuegos en el mercado que podemos rescatar pueden ser:

- GameMaker Studio: A diferencia de la mayoría de los otros motores de juegos se ha utilizado ampliamente porque no requiere conocimientos de programación para su uso permitiendo crear juegos mucho más fáciles y rápidos que la codificación con lenguajes nativos, aunque tiene versión gratuita para hacer los mejores juegos es recomendable su versión de pago.



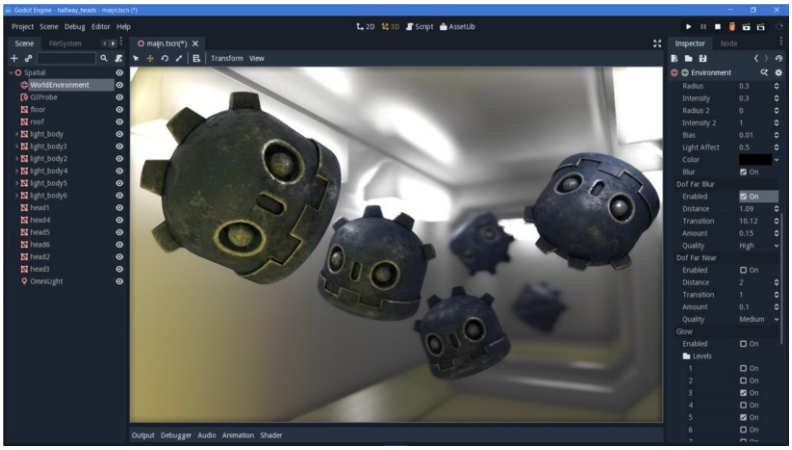
[Figura 2.5. GameMaker]

- Unity: Permite crear contenido 3D interactivo con facilidad. Una gran cantidad de desarrolladores independientes utilizan Unity por su excelente funcionalidad, contenido de alta calidad y su capacidad para prácticamente cualquier tipo de juego, teniendo una edición personal gratis y otra profesión con más opciones.




[Figura 2.6. Unity]

- -Godot: Es de uso gratuito y de código abierto a través de la licencia MIT, que permite exportar videojuegos tanto en 2D como 3D a PC, teléfonos móviles, e incluso despliegues web en formato HTML5.



[Figura 2.7. Godot]

## SOLUCIÓN PROPUESTA

Tal y como pudimos ver anteriormente, el CGI no es para nada sencillo, ni barato. Si tomamos como ejemplo cotidiano el cine, una película promedio de Marvel cuesta entre 100 y 200 millones de dólares para su producción. Con tales cifras, ninguna persona del común se atrevería a intentar implementar CGI de calidad para una obra independiente.

Además de los precios, nos vemos en la necesidad de hacer especial énfasis en la curva de aprendizaje que varias de las soluciones aquí expuestas presentan. Muchas necesitan de varias horas de estudio solo para ser utilizadas, lo cual resulta sumamente frustrante para artistas o cualquier persona que desee incursionar en el mundo del CGI sin conocimientos previos. Para ello, se plantea la creación de una interfaz amigable con el usuario, menús pequeños y con íconos que faciliten su entendimiento, para la creación de imágenes tridimensionales simples que, aunque no se comparen con las de grandes estudios con presupuestos millonarios, brindarían una introducción amigable a este mundo.

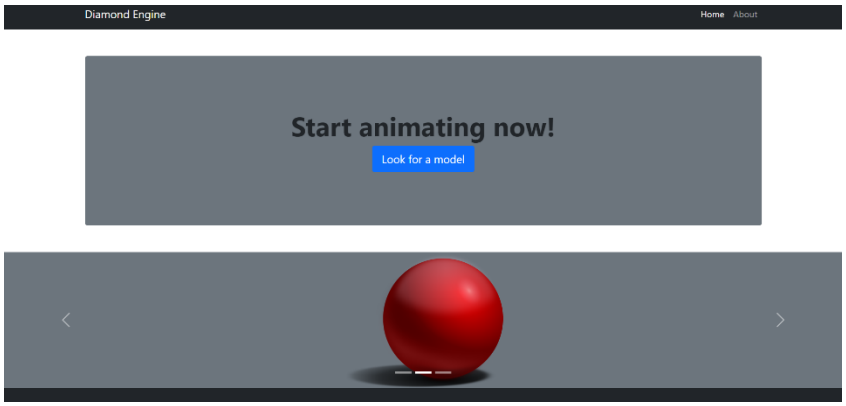
Entonces, la solución planteada por el equipo es una aplicación que permita la creación de imágenes compuestas a través de formas básicas (cubos, esferas…) sin necesidad de ningún conocimiento matemático o computacional previo, solo se necesita una idea a la que se desee dar vida. Para ello, se decidió crear un motor gráfico como los vistos en clase, que le permita al usuario, por medio de menús, insertar distintos objetos y ensamblarlos para crear una imagen que, posteriormente, pueda ser descargada.

Es sumamente necesario aclarar que, por motivos como el tiempo y el recurso humano disponible para el proyecto, no consideramos que la creación de imágenes como escenarios complejos o imágenes fotorrealistas estén dentro de nuestro alcance. Como equipo, queremos concentrarnos, al menos en lo que respecta a la materia de computación gráfica, en brindar una solución simple para la creación de objetos y personajes geométricamente sencillos.

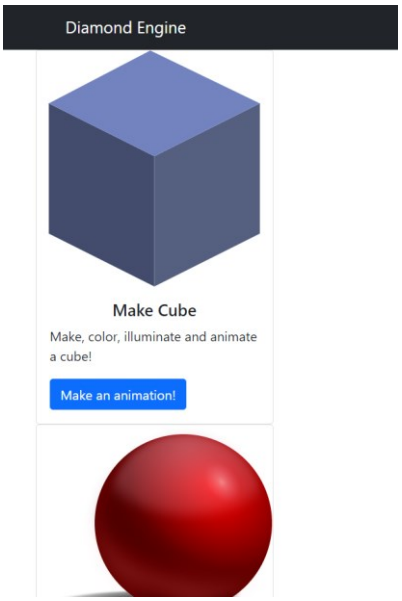
[Figura 3.1. Alcance]

Más a profundidad, se ha decidido desarrollar una aplicación de arquitectura monolítica desarrollada en JavaScript, con vistas en HTML, apoyadas por plantillas de Bootstrap 5.

La aplicación cuenta con dos vistas sencillas, en los archivos home.html y display.html.




[Figura 4.1. Home]




[Figura 4.2. Display]

Desde Display, se puede escoger una de entre 5 vistas predeterminadas, que permiten crear, animar y descargar un modelo tridimensional. Por el momento, se cuenta con 5 modelos: Esfera, Cubo, Cinta de Mobius, Flecha y Casa. De los cuales, la flecha y la casa son modelos realizados a partir de vectores a modo de pruebas y de aprendizaje del equipo.

Estas vistas resultan bastante redundantes, debido a las pobres capacidades de HTML para importar contenidos y secciones sin el uso de frameworks. La estructura de estas vistas es la siguiente:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link
      href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.c
ss"
      rel="stylesheet"
      integrity="sha384-
EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC"
      crossorigin="anonymous"
    />
    <link
      rel="stylesheet"
      href="https://stackpath.bootstrapcdn.com/font-awesome/4.7.0/css/font-
awesome.min.css"
      integrity="sha384-
wvfXpqpZZVQGK6TAh5PVlGOfQNHSoD2xbE+QkPxCAFlNEevoEH3Sl0sibVcOQVnN"
      crossorigin="anonymous"
    />
    <title>Diamond Engine</title>
  </head>
  <body>
    <!-- Responsive navbar-->
    <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
      <div class="container px-lg-5">
        <a class="navbar-brand" href="../home.html">Sphere Animation Lab</a>
        <button
          class="navbar-toggler"
          type="button"
          data-bs-toggle="collapse"
          data-bs-target="#navbarSupportedContent"
          aria-controls="navbarSupportedContent"
          aria-expanded="false"
          aria-label="Toggle navigation"
```

```
      >
        <span class="navbar-toggler-icon"></span>
      </button>
      <div class="collapse navbar-collapse" id="navbarSupportedContent">
        <ul class="navbar-nav ms-auto mb-2 mb-lg-0">
          <li class="nav-item">
            <a class="nav-link active" aria-current="page" href="../home.html"
              >Home</a
            >
          </li>
          <li class="nav-item"><a class="nav-link" href="#!">About</a></li>
        </ul>
      </div>
    </div>
  </nav>
  <div class="container">
    <div class="row align-items-center">
      <div class="col">
        <form action="">
          <label for="colorpicker">Color Picker:</label>
          <input type="color" id="colorpicker" value="#0000ff" />
        </form>
      </div>
      <div class="col">
        <button
          onclick="incr()"
          type="button"
          class="btn btn-labeled btn-danger"
        >
          <span class="btn-label"
            ><i class="fa fa-search-plus"></i>Zoom In</span
          >
        </button>
      </div>
      <div class="col">
        <button
          onclick="decr()"
          type="button"
          class="btn btn-labeled btn-danger"
        >
          <span class="btn-label"
            ><i class="fa fa-search-minus"></i>Zoom Out</span
          >
        </button>
      </div>
```

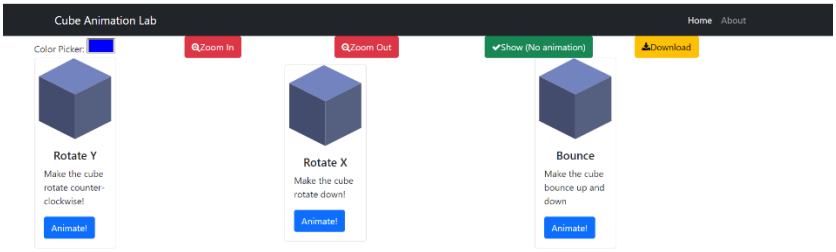
```
      <div class="col">
        <button
          id="choice"
          onclick="program()"
          type="button"
          class="btn btn-labeled btn-success"
        >
          <span class="btn-label"><i class="fa fa-check"></i></span>Show (No
          animation)
        </button>
      </div>
      <div class="col">
        <button
          id="choice"
          onclick="local_export()"
          type="button"
          class="btn btn-labeled btn-warning"
        >
          <span class="btn-label"><i class="fa fa-download"></i></span>Download
        </button>
      </div>
      </div>
      </div>
      <div class="row align-items-center">
        <div class="col">
          <div class="card" style="width: 9rem">
            <img
              src="../../img/icons/cube.svg"
              class="card-img-top"
              alt="..."
            />
            <div class="card-body">
              <h5 class="card-title text-center">Bounce</h5>
              <p class="card-text">Make the sphere bounce up and down</p>
              <a onclick="local_bounceAnimate();" class="btn btn-primary"
                >Animate!</a
              >
            </div>
          </div>
        </div>
      </div>
    </div>
    <canvas id="canvas"></canvas>
    <script src="../../js/three.js"></script>
```

```
    <script src="../../js/OrbitControls.js"></script>
    <script src="../../js/GLTFExporter.js"></script>
    <script src="../create.js"></script>
    <script src="../animation.js"></script>
    <script src="../choice.js"></script>
    <script>
      var pos = 5;
      function local_export(){
        exportAnimation()
      }
      function incr() {
        pos -= 0.1;
        program();
      }
      function decr() {
        pos += 0.1;
        program();
        justShow();
      }
      function program() {
        colour = document.getElementById("colorpicker").value;
        setBackground('/img/bg/space.jpg')
        setColor(colour);
        setPos(pos);
        choose("1");
        justShow();
      }
      function local_bounceAnimate() {
        colour = document.getElementById("colorpicker").value;
        setBackground('/img/bg/space.jpg')
        setColor(colour);
        setPos(pos);
        choose("1");
        bounceAnimate();
      }
    </script>

    <footer class="py-5 bg-dark">
      <div class="container">
        <p class="m-0 text-center text-white">
          Copyright &copy; Diamond Engine 2022
        </p>
        </div>>
      </div>
    </footer>
```

```
  </body>
</html>
```

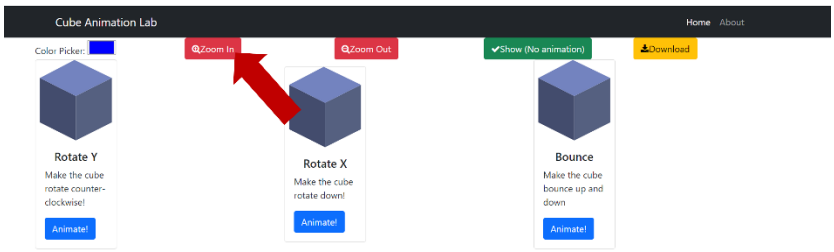
En ellas, varían el modelo geométrico a ser creado y las animaciones disponibles. Antes de crear un objeto, la vista general del motor sería la siguiente:




[Figura 4.3.Vistas Modelos]

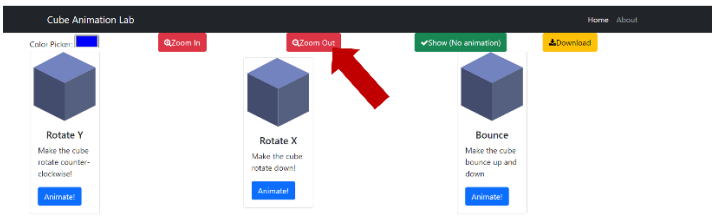
Allí, se tienen las siguientes opciones:

- Zoom In: Para acercar la imagen hacia el observador.



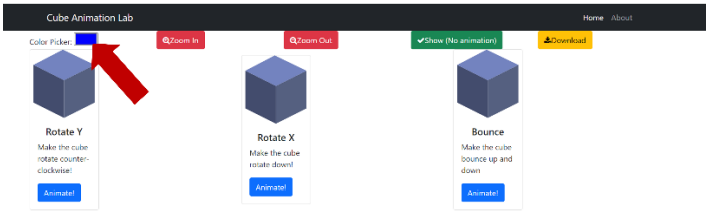

[Figura 4.4. Zoom In]

- Zoom Out: Para alejar la imagen del observador.



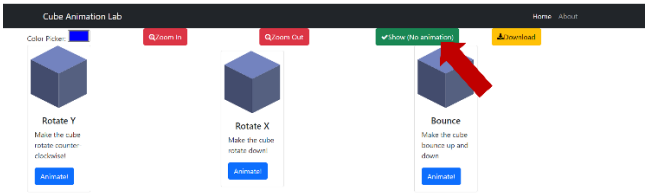

[Figura 4.5. Zoom Out]

- Color Picker: Para escoger el color predominante del modelo.



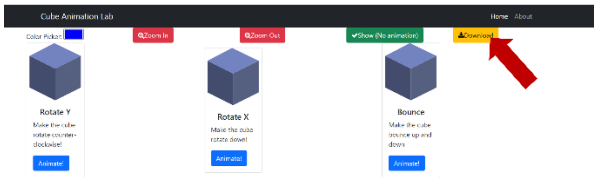

[Figura 4.6. Color]

- Show: Mostrar la imagen escogida.




[Figura 4.7. Show]

- Download: Descargar el modelo creado.




[Figura 4.8. Download]

Desde lo más interno, se tomó como base la biblioteca three.js, con las bibliotecas Open Source OrbitControls.js, para controlar el ángulo de la cámara desde el mouse y GLTFExporter.js, para permitir la descarga de escenas creadas.

Así, la elección del usuario desde las vistas acciona funciones desde choice.js, encargado de generar el objeto adecuado dependiendo de la selección del usuario. Este, cuenta con la siguiente estructura:

```
function choose(shapeChoice){
    switch (shapeChoice) {
        case '0':
            while(scene.children.length > 0){
                scene.remove(scene.children[0]);
            }
            setScene()
            object = makeCube(objectColour)
            addScene(object,objectPos)
            break;
            .
            .
            .
    }
}
```

Esto, sigue la misma lógica para cada selección posible, en la que el usuario escoge un objeto, su color y su posición relativa a la cámara. Cada vez que el usuario cambia su algún parámetro, la escena debe ser eliminada para evitar superposiciones y se añade el nuevo objeto con los parámetros establecidos a la escena a través del método addScene().

Entonces, pasamos al archivo create.js, el cual, tal y como indica su nombre, trata de crear objetos dentro de la escena, con los parámetros escogidos por el usuario. La mayoría de estos, están fuertemente basados en la documentación de three.js. Recuperada de: https://threejs.org/

Una función 'make' cualquiera cuenta con la siguiente estructura:

```
function makeCube(colour) {
    const geometry = new THREE.BoxGeometry();
    const material = new THREE.MeshToonMaterial({ color: colour });
    const cube = new THREE.Mesh(geometry, material);
    cube.position.y = 1
    makeFloor();

    return cube
}
```

Al gestionar la geometría del objeto, se crea un 'suelo' para darle perspectiva al usuario y se retorna el objeto a choice.js para continuar con el proceso de muestra en pantalla.

```
function makeFloor(){

    base=new THREE.BoxGeometry(1, 0.1, 1),
    whiteTile= new THREE.MeshPhongMaterial({ color: 0x000000, shininess: 150 }),
    blackTile=new THREE.MeshPhongMaterial({ color: 0xffffff, shininess: 150 }),

    floor = new THREE.Group()
    var cube
    for(let x = -8; x < 8; x++){
        for(let z = -8; z < 8; z++){
            if(z%2==false){
                if(x%2==false){
                    cube=new THREE.Mesh(base,whiteTile)
                }
                else {cube=new THREE.Mesh(base, blackTile);}
            }else{
                if(x%2==false){
                    cube=new THREE.Mesh(base,blackTile)
                }
                else {cube=new THREE.Mesh(base, whiteTile);}
            }
            cube.position.set(x,0,z);
            floor.add(cube)
        }
    }

    scene.add(floor);
}
```

En la función makeFloor(), se hace un suelo con el suficiente contraste como para que el usuario pueda observar correctamente la iluminación, cambio de tamaño y movimiento del objeto con respecto a la cámara. Para ello, se tienen dos cubos, negro y blanco, que formaran las 'baldosas' intercaladas de acuerdo con la función mostrada. Posteriormente, se añade el suelo a la escena.

La función addScene() fue concebida con el propósito de tener más control sobre cuándo puede un objeto unirse a la escena, para así favorecer aspectos como el control y la escalabilidad de la aplicación.

```
function addScene(object, pos) {
    scene.add(object);
    camera.position.z = pos;
    return object
}
```

Tal y como puede verse en el extracto de código anterior, esta función se encarga de recibir cada objeto y agregarlo a la escena actual, además de controlar su posición relativa.

Finalmente, se llega al apartado de animaciones, en el archivo animation.js desde donde se cambia la posición de un objeto repetidamente y se lleva a la función de three.js requestAnimationFrame(), para crear recursivamente la ilusión de movimiento. Un ejemplo bastante sencillo podría ser:

```
function rotationXAnimate() {
    requestAnimationFrame(rotationXAnimate);

    object.rotation.x += rotateX;
    renderer.render(scene, camera);

};
```

En este, se rota el objeto sobre el eje X, según un valor variable para que, en un futuro, puedan añadirse cambios en la velocidad, luego, la función es utilizada como parámetro por requestAnimationFrame().

Luego, puede exportarse el trabajo creado por el usuario, por medio de la función exportAnimation(), basada fuertemente en soluciones al mismo problema discutidas en foros de three.js. (https://discourse.threejs.org/t/exporting-an-animated-gltf-using-gltfexporter/29567)

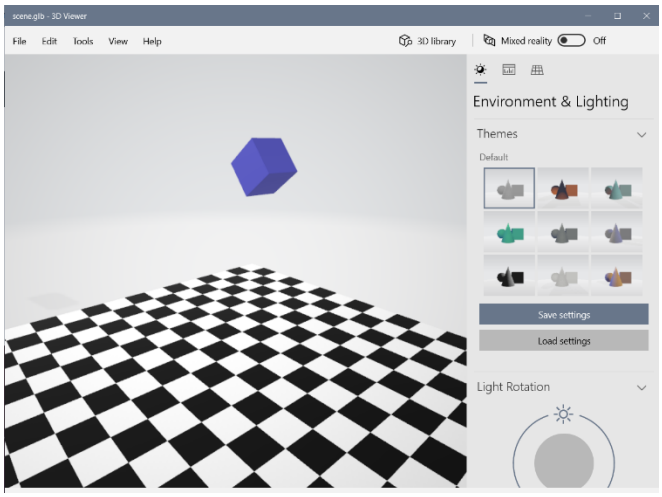
## COMPARACIÓN

El trabajo realizado por el equipo se diferencia enormemente de otras soluciones en su curva de aprendizaje y facilidad de uso. Tal y como se ha mostrado con anterioridad, nuestra interfaz es bastante amigable, fácil de utilizar y con menús cortos con botones ilustrados por íconos y colores llamativos.

Por otro lado, es una solución que no requiere de instalación ni inversión económica, que permite una introducción sencilla al mundo del CGI desde un ambiente basado en un 'market place' con distintas opciones de modelos y animaciones a escoger, sin llegar a opciones muy avanzadas que, aunque relativamente sencillas de desarrollar, podrían abrumar al usuario.

Al contrastar nuestra aplicación con otras de corte profesional, como Blender o motores de videojuegos variados, se cuenta con la especial ventaja de que requiere poco o nulo conocimiento en diseño y animación en 3D, o de codificación, puesto que todo puede hacerse desde la interfaz gráfica.

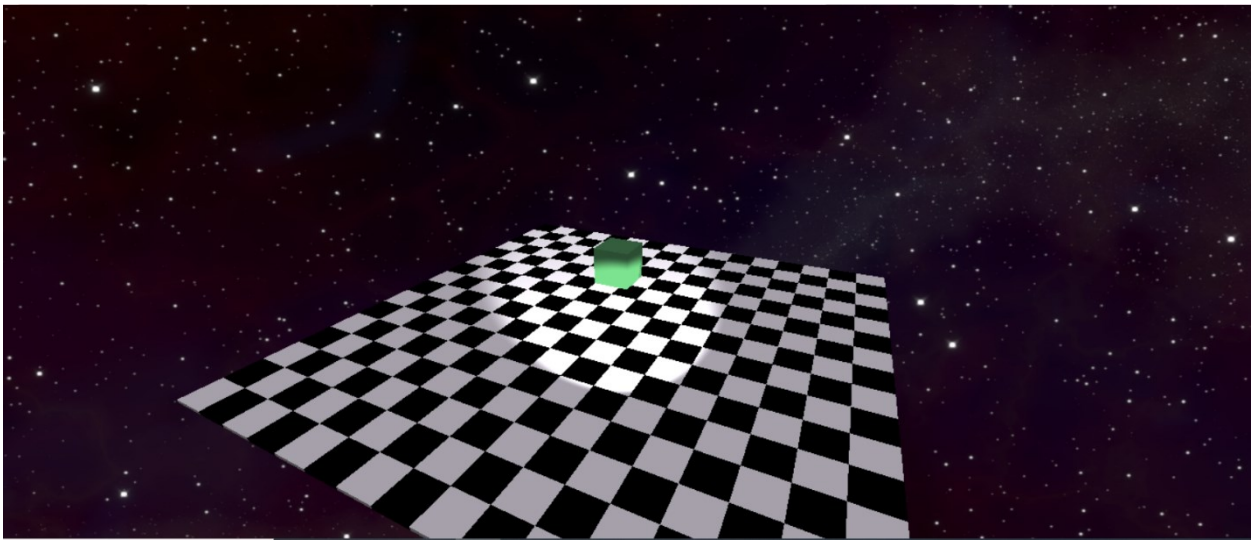
Además, permite exportar el trabajo creado y descargarlo en un formato incluso reconocido por Windows.




[figura 5.1. Modelo en Windows]

Todo esto, marca una diferencia con lo que ya se ha implementado en el mercado. Lo cual, como hemos visto anteriormente, resulta costoso, al tener que invertir no solo en licencias, sino también en calificación y enfrentar la curva de aprendizaje que, en su mayoría, es bastante pronunciada. Asimismo, cabe destacar la necesidad de instalación y el enorme uso de poder computacional, al no todo el mundo poseer los recursos tecnológicos necesarios para crear y animar imágenes tridimensionales. Este problema desaparece al tener una solución Web, pues es el navegador quien se encarga de ello.

Por ejemplo, una pantalla cualquiera de la aplicación funcionando correctamente podría ser al crear un cubo, agregarle color y darle una animación de rebote más rotación lateral.



[Figura 5.2. Ejemplo Aplicación Funcionando]

## CONCLUSIONES

Como conclusión, el equipo se permitió aprender, no solo aspectos técnicos sobre el desarrollo del CGI, sino el contexto en el que se encuentra en la industria, que era desconocido para el equipo hasta antes de iniciar el desarrollo del proyecto. Las distintas tecnologías en el mercado, entre las cuales algunas eran bastante conocidas o, incluso, se había trabajado sobre ellas (Godot, por ejemplo), permitieron basarse en ellas para la creación de un motor gráfico con mínimas capacidades que permitiera introducir al usuario casual al mundo de las imágenes generadas por computadora.

Por otro lado, desde el apartado técnico, el equipo aprendió, sobre todo, a trabajar desde lenguajes como JavaScript, que no había sido utilizado por el equipo (más que superficialmente) hasta este proyecto. El uso de las bibliotecas de three.js y de la comunidad facilitó, además, el entendimiento de conceptos matemáticos y geométricos vistos en clase a partir de la experimentación sobre código y la visualización de los resultados por pantalla. Entre estos conceptos, destacamos algunos que se entendieron menos durante las clases, como la iluminación y el comportamiento simulado de la luz frente a objetos con texturas planas, así como la necesidad de planos para perspectivas y la manera en que funciona la animación por fotogramas en threejs, en el que cada uno, representaba un movimiento mínimo con respecto a su posición anterior.

Finalmente, desde una perspectiva más personal, se dio la oportunidad de trabajar en un proyecto bastante interesante, creando un motor gráfico a partir de recursos ya existentes, pero dándole una utilidad enfocada a nuestra solución, así como un toque personal a cada una de las partes utilizadas, para apegarlas a nuestro caso particular.

## REFERENCIAS


https://www.grandviewresearch.com/industry-analysis/3d-animation-market

https://academyofanimatedart.com/breakthrough-and-expensive-cgi-scenes-in-mcu-movies/

https://icoginix.com/advantages-and-disadvantages-of-using-cgi-in-the-film-industry/

https://www.studiobinder.com/blog/what-is-cgi-meaning-definition/

https://www.creativehumans.com/blog/how-much-animation-cost

https://geekflare.com/best-cfd-analysis-software/

https://www.gamedesigning.org/career/video-game-engines/

https://www.eurekalert.org/news-releases/497585

https://www.youtube.com/watch?v=K8LFWvSqdJI

https://www.openculture.com/2021/04/how-pixars-movement-animation-became-so-realistic.html

https://discourse.threejs.org/t/exporting-an-animated-gltf-using-gltfexporter/29567

https://www.nfi.edu/what-is-cgi/

https://moneyinc.com/much-costs-make-single-episode-game-thrones/