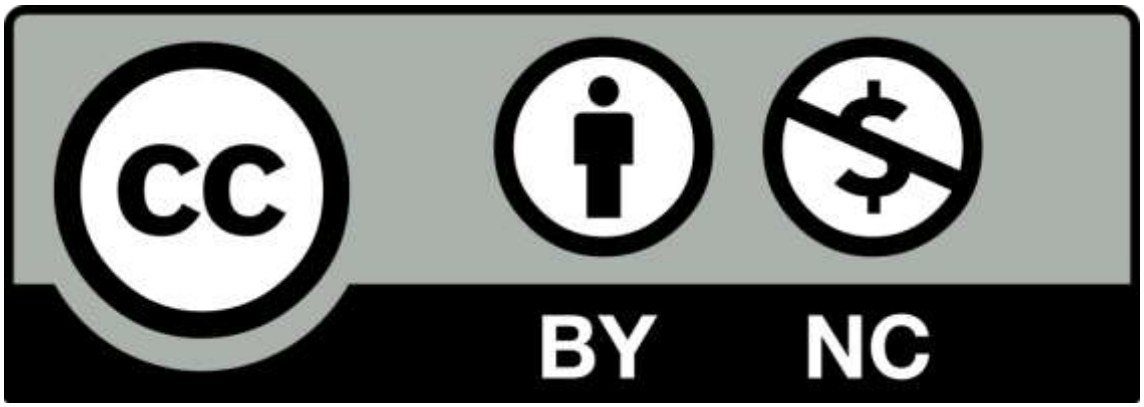
TRABAJO FINAL DE CATEDRA

Titulo
Videojuego en 3D "The Cube"

Autor/es
Ellen Margarita Coreas Pérez **Claudia Patricia Salamanca Martínez** **Fredy Ernesto Turcios Reyes** **Raúl Mauricio Portillo Muñoz**
Catedrático
Ing. Ludwin Alduvi Hernández
Facultad
Facultad Multidisciplinaria Oriental
Departamento
Depto. De Ingeniería y arquitectura
Curso Académico
Ciclo I – 2017



Email: publicaciones_UES@gmail.com

UNIVERSIDAD DE EL SALVADOR

Facultad Multidisciplinaria Oriental UES-FMO

TRABAJO FINAL DE CATEDRA

Algoritmos Gráficos

Videojuego en 3D “The Cube”

Alumnos: Claudia Patricia Salamanca Martínez

Ellen Margarita Coreas Pérez

Raúl Mauricio Portillo Muños

Fredy Ernesto Turcios Reyes

Catedrático: Ing. Ludwin Alduvi Hernández

San Miguel, 01/07/2017

Índice

Introducción

Podemos decir que la computación gráfica o gráficos por ordenador es el campo de la informática visual que se encarga de la representación de gráficas u figuras en el ordenador, donde estas utilizan computadoras tanto para generar imágenes visuales sintéticamente como integrar o cambiar la información visual y espacial probada en el mundo real.

OpenGL es una librería gráfica escrita originalmente en C que permite la manipulación de gráficos 3D a todos los niveles. Esta librería se concibió al programar en máquinas nativas Silicon Graphics bajo el nombre de GL (Graphics Library). Posteriormente se consideró la posibilidad de extenderla a cualquier tipo de plataforma y asegurar así su portabilidad y extensibilidad de uso con lo que se llegó al término Open Graphics Library, es decir lo que actualmente conocemos como OpenGL.

Así que está librería podrá utilizarse bajo todo tipo de sistemas operativos e incluso usando una gran variedad de lenguajes de programación. No obstante su uso más extenso suele ser el lenguaje C o C++.

CMake es una herramienta multiplataforma de generación o automatización de código. El nombre es una abreviatura para "cross platform make" (make multiplataforma); más allá del uso de "make" en el nombre, CMake es una suite separada y de más alto nivel que el sistema make común de Unix, siendo similar a las autotools.

De esta manera trabajaremos con C++ y por tanto nos referiremos siempre a ejemplos codificados según este lenguaje. Simplemente necesitaremos las librerías adecuadas y un editor de texto plano, es decir los estándares de este lenguaje para así lograr el cometido de diseñar y estructurar un juego en 3D con OpenGL junto a C++.

1. El problema

1.1 Situación problemática

Son muchas las necesidades que presentan los niños y las niñas en el nivel de Educación Parvularia y Educación Básica, debido a que es el primer peldaño de la educación inicial formal en la que cada uno de ellos pone de manifiesto destrezas y habilidades que por naturaleza son natas en ellos y que solo es necesaria la estimulación adecuada y oportuna, para lograr su desarrollo.

En el primer nivel educativo, es cuando el padre de familia como principal responsable de la educación de sus hijos e hijas, busca en una institución educativa el proceso de enseñanza confiando en que el maestra/o, a través de sus conocimientos pedagógicos le facilite al niño y a la niña el desarrollo del área cognitiva, socio-efectiva y psicomotora, para que estos vayan adquiriendo una formación integral.

Para poder lograr este objetivo, que está contemplado en los principios de la educación Parvularia y Básica; el docente o la docente, debe hacer uso de algunas estrategias didácticas, entre ellas, el juego siendo esta una actividad lúdica en la que el niño y la niña manifiesten su personalidad y así mismos sirva como un medio muy valioso para favorecer el desarrollo mental de los niños/as.

Además se debe tener en cuenta que durante el desarrollo de una actividad lúdica en los niños/as, intervienen todos los potenciales físicos, cognitivos, efectivos y sociales;

por lo anterior, es de suma importancia su desarrollo, porque así como ponen en práctica la psicomotricidad, la disociación y la coordinación haciendo uso del esquema corporal, el cual se hace más fácil conocerlo por medio de juegos, que utilizando metodologías tradicionales en las que hay muchos conceptos abstractos que para este nivel se vuelve más difícil su asimilación.

1.2 Titulo descriptivo del proyecto

“Videojuego “The Cube”.

Juego diseñado en OpenGL utilizando primitivas, texturas, materiales, luces y colisiones; la finalidad de esté es que “The Cube” (Nombre podría variar según país), en un plano de segunda dimensión se ira moviendo con una secuencia de imágenes que irán transcurriendo con diferentes texturas según sea la etapa/distancia que haya recorrido el jugador.

El juego consiste en evitar los obstáculos mientras transcurren las escenas, los obstáculos varían entre piedras y troncos, estos deben ser evitados por el jugador ya sea moviéndose a los lados para evitarlos. Al llegar a una distancia determinada el juego ira avanzando de nivel automáticamente aumentando la velocidad y cantidad de obstáculos.

El marcador estará en la parte superior de la pantalla la cual consiste en medidor de distancia, tiempo transcurrido y puntaje.

1.3 Planteamiento del problema

El uso de nuevas tecnologías ha modificado las capacidades y conductas de los niños, pues aunque tienen más habilidad mental limitan su desarrollo motriz, lo que provoca déficit de atención, depresión y enfermedades que antes eran exclusivas de adultos, advirtieron especialistas de la UNAM.

Como dato de medición según la UNAM (Universidad Nacional Autónoma de México) revela que la población de 0 a 14 años de edad en México es de 31.3 millones y que el país es el principal consumidor de videojuegos de América Latina con más de 3 millones de videoconsolas vendidas al año. Información de 2002 de la Secretaría de Salud menciona que el 16% de los mexicanos de entre 3 y 12 años presentan problemas de salud mental como la inquietud, irritabilidad, nerviosismo, déficit de atención, desobediencia, explosividad y conducta dependiente.

Por lo tanto a medida que se ha tomado conciencia sobre la educación en nuestro país, especialmente en los niveles de Educación Parvularia y Básica, se ha observado ciertas necesidades y algunas dificultades que presentan la población infantil, en cuanto al aprendizaje que va adquiriendo en el desarrollo de la expresión oral y corporal, por lo tanto, esto genera cierta preocupación, debido a que se espera obtener un desarrollo integral en los párvulos y sector básico.

La relevancia de realizar la presente investigación fue debido a la necesidad de incorporar una estrategia visual que favorezca a el desarrollo psicomotriz y además de

afrontar situaciones de decisiones, de tal forma que propicie en los niños y niñas un desarrollo integral.

Por lo tanto la investigación servirá para medir el nivel de aceptación por parte de los párvulos y básicos en el área de juegos de video que se implementara de forma clara y concisa dentro de las instalaciones de la Universidad, para que así aprovechen este recurso que ayudara a el desarrollo psicomotriz y además a la toma de decisiones para poder resolver problemas.

1.4 Enunciado del problema

¿En qué medida o porcentaje será aceptado el videojuego “The Cube” por los niños de hasta diez años de edad?

1.5 Justificación

Dicha investigación contribuirá a resolver el problema que la mayoría de educandos presentan a la hora de resolver problemas u tomas decisiones más que todo en los niños y niñas menores de diez años.

En el área psicomotora el niño y niña presentan ciertas dificultades al momento de ubicarse, de señalar correctamente las partes del cuerpo y conocer la función de cada una de ellas o en coordinar movimientos siguiendo una secuencia y respetando su entorno.

Por lo tanto, este trabajo proporciona las herramientas educativas visuales y psicomotoras para ayudar al desarrollo mental y corporal de los niños/as menores de diez años; tomando como herramienta didáctica el juego “The Cube”.

La finalidad del proyecto es notar la importancia y el manejo de OpenGL para la creación de videojuegos. De esta manera se espera ampliar el conocimiento y así garantizar la comprensión sobre desarrollo de videojuegos utilizando OpenGL, además del uso de infinitas herramientas para agilizar el desarrollo.

1.6 Delimitación

Delimitación temporal:

La investigación se realizó durante el periodo de mayo a junio del año 2017.

Delimitación espacial:

La investigación se ha llevado a cabo en la Universidad de El Salvador – Multidisciplinaria Oriental; en el nivel de educación de Parvularia y básica del Programa de Jóvenes Talentos.

Delimitación social:

El estudio fue delimitado y se trabajó con el nivel de Parvularia hasta básica, de niños menores a diez años, donde se realizaron diversas actividades que culminaron con la preparación e implementación de un juego de video para el desarrollo educativo.

1.6.1 Lugar o espacio

La preparación del videojuego se referirá en configuraciones regionales para el libre acceso y adaptación a las normas de uso del videojuego, por otra parte el lugar donde será implementado, será en las instalaciones de la Universidad de El Salvador en el cual estará alojado el antes mencionado videojuego “The Cube” y el espacio donde se podrá ejecutar el videojuego, será en el centro de cómputo de la Universidad de El Salvador – FMO, aunque cabe mencionar que dicho videojuego no será posible utilizarlo fuera de las instalaciones de la Universidad.

1.6.2 Tiempo

Diseño de escenas y programación:	Del 22 de mayo hasta el 17 de junio de 2017.
Diseño de figuras.	
Diseño de avatar.	
Prueba:	Del 18 de junio hasta el 24 de junio del 2017.
Validaciones y Correcciones.	
Documentación y exposición:	Del 25 de junio hasta el 30 de julio de 2017.
Elaboración de documento.	
Exposición del video-juego.	

1.6.3 Teorías

Nuestro proyecto está basado en un videojuego ya existente llamado, “Temple Run”, al comienzo planteamos lo que queríamos hacer y llegamos a la conclusión que lo más adecuado sería un videojuego de genero aventura orientado para los niños, ya que el antes mencionado videojuego “Temple Run” esta para jóvenes mayores de 17 años.

En nuestra recopilación de información nos encontramos con que ***Temple Run*** es un videojuego creado en 2011 3D sin fin en marcha; el videojuego fue desarrollado y publicado por Imangi Estudios. Se produce, diseñado y programado por el equipo de marido y esposa Keith Shepherd y Natalia Luckyanova, y con el arte de Kiril Tchangov. El juego fue lanzado inicialmente para iOS dispositivos, y más tarde portado a Android sistema y Windows Phone 8.

Para mayor información de los creadores del videojuego del cual nos basamos se puede acceder a él por medio de la siguiente dirección web: https://play.google.com/store/apps/details?id=com.imangi.templerun&hl=es

1.7 Objetivos del proyecto

1.7.1 Objetivos genérales:

Desarrollar videojuego “The Cube”, para niños menores de diez años.

1.7.2 Objetivos específicos:

- Recolectar información necesaria y precisa sobre el proyecto, para obtener lo que desea o aspira el usuario, para ir estableciendo los límites del mismo.
- Indagar sobre la viabilidad del estudio del proyecto de manera rápida, los planes, problemas, oportunidades o las normas que desencadenan y que permitirán el desarrollo de este proyecto.
- Considerar si dentro de las instalaciones de la UES - FMO existen los recursos tecnológicos necesarios para poder implementar el videojuego “The Cube”.
- Determinar si los costos de desarrollo e implantación del videojuego “The Cube” se justifican en función de los beneficios.
- Comprobar si los usuarios potenciales están en la capacidad de usar apropiadamente el videojuego, o cuánto tiempo se requerirá para enseñar a los niños/as menores de diez años, en el uso apropiado del nuevo videojuego “The Cube”.

- Establecer tiempos de desarrollo e implantación aceptables, para recuperar la inversión y satisfacer a los usuarios finales.

- Estipular mediante una investigación preliminar, las necesidades y características que deberá cubrir el nuevo videojuego.

- Diseñar el videojuego 3D “The Cube” para niños menores de diez años, cumpliendo con los requerimientos que se estipularon en la investigación preliminar.

- Desarrollar el videojuego 3D “The Cube” para niños menores de diez años.

- Implantar las herramientas, recursos, personas y el nuevo sistema de información dentro de los centros de cómputo de la UES - FMO.

- Evaluar el videojuego para identificar sus puntos débiles y fuertes, para proporcionar más información que será de ayuda para mejorar la efectividad de los esfuerzos de desarrollo de aplicaciones subsiguientes.

2. Fundamentación teórica

2.1 Marco Teórico

2.1.1 Diagnostico participativo

Los videojuegos tiene gran influencia en la vida de los niños y niñas, hay pruebas que los comportamientos impulsivos y agresivos son causados por una gran afición hacia los niños/as que con el tiempo pueden afectar su vida social limitándolo en actividades en donde pueda socializar con otros niños/as y generar sedentarismo que sin el tiempo generara enfermedades cardiovasculares, en ocasiones cuando su uso excesivo puede ser clasificado como un trastorno psicopatológico.

Actualmente los niños y niñas dejan de lado las actividades físicas por estar jugando videojuegos. El número de usuarios que utilizan los videojuegos, va creciendo, creando así una preocupación por parte de los padres de familia por las posibles consecuencias negativas que pudieran tener sobre aquellos que la utilizan con regularidad, ya que existe una considerable dedicación de tiempo. Pero por otro lado los videojuegos también permiten que interactúen con otras personas de diferentes edades que tienen sus mismas aficiones.

Los videojuegos tienen una parte positiva ya que por sus temáticas permiten que las personas aprendan a crear estrategias, a aceptar las derrotas y valorar más las batallas ganadas, los videojuegos también van formando un hábito de buscar soluciones rápidas en situaciones problemáticas.

Generalmente lo más atrayente de estos juegos es la combinación de la fantasía con el realismo que estos manejan, permitiendo así que los niños y niñas experimenten varias aventuras.

Hoy en día se considera a los “videojuegos como una droga virtual” ya que en algunas ocasiones, estas actividades pueden alejar a los niños/as de las actividades sociales y físicas, muchos padres creen que por jugar videojuegos sus hijos se tornaran violentos y caprichosos, pero estas actitudes dependen de cómo cada persona permita que los videojuegos entren en su vida.

Toda la problemática que ha surgido alrededor de los videojuegos ha hecho que se formen mitos que desprestigian a los videojuegos, algunos de ellos son que los videojuegos son solo para los niños o que los videojuegos son del diablo, gracias a esto hace un tiempo esta actividad se consideraba tabú, pero con el pasar del tiempo las personas se han dado cuenta de que no son tan malos como las demás lo quieren hacer ver.

En esta investigación, no es extraño ver que se pueda contradecir las respuestas encontradas ya que estas pueden ser relativas, debido a que hay estudios que dicen que se puede afectar la motricidad en casos especiales, o que disminuye la cantidad y calidad de interacciones sociales, pero también se dicen que los niños se hacen más inteligentes, mejoran la capacidad psicomotriz y desarrollan habilidades en el uso de computadoras.

2.1.2 Antecedentes Históricos

Se presenta a OpenGL (Open Graphics Library), como una especificación estándar que define una API multilenguaje y multiplataforma para escribir aplicaciones que produzcan gráficos 2D y 3D. La interfaz consiste en más de 250 funciones diferentes que pueden usarse para dibujar escenas tridimensionales complejas a partir de primitivas geométricas simples, tales como puntos, líneas y triángulos.

En 1982 nació en la Universidad de Stanford el concepto de "Graphics machine" y éste fue utilizado por Silicon Graphics Corporation en su propia estación Silicon IRIS para crear un renderizador. Así nació la librería IRIS GL. A raíz de esto, en 1992 muchas empresas del hardware y software se pusieron de acuerdo para desarrollar conjuntamente una librería gráfica libre: OpenGL.

Al principio de los años 1990 SGI era un grupo de referencia en gráficos 3D para estaciones de trabajo. Suya era la API IRIS GL, considerada puntera en el campo y estándar de facto, llegando a eclipsar a PHIGS, basada en estándares abiertos. IRIS GL se consideraba más fácil de usar y, lo más importante, soportaba renderizado en modo inmediato. Además, PHIGS, aparte de su mayor dificultad, fue considerada inferior a IRIS GL respecto a funcionalidad y capacidad.

La competencia de SGI (Sun Microsystems, Hewlett-Packard e IBM, entre otros) fue capaz de introducir en el mercado hardware 3D compatible con el estándar PHIGS mediante extensiones. Esto fue reduciendo la cuota de mercado de SGI conforme iban entrando diferentes proveedores en el mercado. Por todo ello, en un intento de fortalecer

su influencia en el mercado, SGI decidió convertir el estándar IRIS GL en un estándar abierto.

SGI observó que la API IRIS GL no podía ser abierta debido a conflictos de licencias y patentes; también contenía funciones no relevantes para los gráficos 3D como APIs para ventanas, teclado o ratón (en parte, porque fue desarrollada antes de la aparición del X Windows System o de los sistemas NeWS de Sun). Además, mientras iba madurando el soporte del mercado para el nuevo estándar, se pretendía mantener los antiguos clientes mediante bibliotecas añadidas como Iris Inventor o Iris Performer. El resultado de todo lo anterior fue el lanzamiento del estándar OpenGL.

Algunos de los logros que se consiguieron fueron:

- Estandarizar el acceso al hardware.
- Trasladar a los fabricantes la responsabilidad del desarrollo de las interfaces con el hardware.
- Delegar las funciones para ventanas al sistema operativo.

Con la variedad de hardware gráfico existente, lograr que todos hablasen el mismo lenguaje obtuvo un efecto importante, ofreciendo a los desarrolladores de software una plataforma de alto nivel sobre la que trabajar.

En 1992, SGI lideró la creación del OpenGL Architecture Review Board (**OpenGL ARB**), grupo de empresas que mantendría y extendería la especificación OpenGL en los años siguientes. OpenGL evolucionó desde IRIS GL, superando su problema de dependencia del hardware al ofrecer emulación software para aquellas características no soportadas por el hardware del que se dispusiese. Así, las aplicaciones podían utilizar gráficos avanzados en sistemas relativamente poco potentes.

Fue importante para el despegue definitivo de OpenGL, la elección de John Carmack para realizar el juego Quake con esta API gráfica, marcando un precedente innegable, con OpenGL era más sencillo y eficaz la realización de programas 3D (1996). Para elegir entre DirectX (3.0/3.0a) y OpenGL realizó un pequeño ejemplo de Lens Flare (brillo sobre las lentes), con OpenGL escribió unas 30 líneas de código mientras que en DirectX escribió unas 300 líneas de código, además el efecto era mejor y más rápido en OpenGL que en DirectX, por tanto su elección fue clara.

2.1.3 Definiciones

Podemos mencionar que un videojuego u programa informático, creado en principio para el entretenimiento, basado en la interacción entre una o varias personas y recrean entornos y situaciones virtuales en los cuales el jugador puede controlar a uno o varios personajes (o cualquier otro elemento de dicho entorno), para conseguir uno o varios objetivos por medio de unas reglas determinadas.

Los videojuegos pueden ser ejecutados por otros equipos electrónicos diferentes a las consolas, como computadoras y teléfonos celulares; por lo tanto podemos llegar a mencionar los primeros géneros de videojuegos:

- Aventura: Juego en los que él protagonista debe avanzar en la trama interactuando con diversos personajes y objetos.
- Deportivos: Son videojuegos basados en deportes ya sean reales o ficticios y se pueden subdividir en simuladores.
- Disparo: También conocidos como “Shooters” o “Shoot’em up” (en inglés, dispárales). En estos juegos el protagonista ha de abrirse camino a base de disparos.
- Estrategia: Se caracterizan por la necesidad de manipular a un numeroso grupo de personajes, objetos o datos para lograr los objetivos.
- Lucha: Videojuegos basados en el combate físico. Se dividen en juegos de 1 contra 1 o “versus” u juegos de avanzar y pegar.

- Plataformas: Juegos en los que el protagonista ha de avanzar a través de un mapeado con múltiples alturas en la que el protagonista debe saltar las plataformas para llegar al objetivo.

Para el presente proyecto utilizaremos el género “Aventura” para el videojuego “The Cube”, ya que nuestro videojuego consta de las siguientes escenas y tramas dentro del mismo; todo comienza con una secuencia animada que corre hacia adelante en la cual el avatar evade obstáculos saltando, agachándose o desplazándose de izquierda a derecha en la cual va recorriendo escenas diversas que cambian dependiendo la cantidad de la distancia recorrida y este termina cuando el avatar choca contra cualquiera de los obstáculos, además se muestra el puntaje por la distancia recorrida y obstáculos librados.

2.1.4 Desarrollo

2.1.4.1 Librerías añadidas

En el desarrollo de este videojuego, no sólo se trabajara con la librería OpenGL. Trabajaremos también con la GLU, GLUT, MATH y SOIL. La librería GLU contiene funciones gráficas de más alto nivel, que permiten realizar operaciones más complejas (una cosa así como el ensamblador y el C) En cambio, la librería GLUT es un paquete auxiliar para construir aplicaciones de ventanas, además de incluir algunas primitivas geométricas auxiliares. La gran ventaja de este paquete es que el mismo código nos servirá en Windows™ y en Linux™, además de simplificar mucho el código fuente del programa, como veremos en los ejemplos. También la librería MATH es un archivo de cabecera de la biblioteca estándar del lenguaje de programación C/C++ diseñada para operaciones matemáticas básicas. Muchas de sus funciones incluyen el uso de números en coma flotante. Por otra parte la librería SOIL es una biblioteca pequeña en C/C++, utilizado principalmente para la posibilidad de subir texturas en OpenGL.

2.1.4.2 Formato de las funciones

Los nombres de las funciones en estas librerías siguen la siguiente convención:

{gl, glu, glut} <Un nombre> [{d, f, u,… etc}] [v]

El prefijo gl indica que se trata de una función de la librería de OpenGL, el prefijo glu de una función de la librería GLU, y el prefijo glut es para las funciones de la GLUT. Los sufijos pueden aparecer o no, depende. Si los hay, es que la función puede estar en varias versiones, dependiendo del tipo de datos de los parámetros de ésta. Así, acabará con un sufijo d si recibe parámetros double, y con un sufijo fv si recibe parámetros de tipo *float (es decir, punteros a float). En estas prácticas, por defecto referenciaremos las funciones con sufijo f y fv, aunque haya más versiones.

Ejemplos de funciones: gluPerspective, glColor3f¸glutSwapBuffers, glMaterialf, glMaterialfv… etc.

2.1.4.3 Funciones básicas que utilizaremos para definir objetos

Primitivas geométricas básicas con las que podemos dibujar puntos, líneas y polígonos. Para definir un punto en el espacio 3D, usaremos la función: glVertex3f(x, y, z). Estos puntos se unen formando estructuras, como líneas y polígonos. La siguiente descripción muestra alguna de las opciones que tenemos:

Modo Descripción; GL_POINTS Puntos individuales aislados GL_LINES Cada par de puntos corresponde a una recta GL_LINE_STRIP Segmentos de recta conectados

GL_LINE_LOOP Segmentos de recta conectados y cerrados GL_TRIANGLES Cada tres puntos un triángulo GL_QUADS Cada cuatro puntos un cuadrado GL_POLYGON Un polígono

Primitivas de objetos predefinidos, hay algunos objetos que vamos a renderizar muy a menudo en nuestro proyecto, y que por tanto, ya vienen definidos. Así, disponemos de las siguientes funciones:

- glutSolidSphere(radius, slices, stacks)
- glutSolidCube(size)
- glutSolidCone(base, height, slices, stacks)
- glutSolidDodecahedron(void)
- glutSolidOctahedron(void)
- glutSolidTetrahedron(void)

También trabajaremos con colores dentro de nuestro proyecto de videojuego, por defecto el trazado de las líneas y puntos es blanco, pero podemos cambiarlo. Para hacer esto, usaremos la función glColor3f(r, g, b). El valor de r, g y b debe estar entre 0 y 1 (¡y no entre 0 y 255!).

2.1.4.4 Definiendo escena técnica

Hasta ahora hemos construido nuestra escena mediante la especificación directa de las coordenadas 3D, o bien utilizando las primitivas de objetos. Pero, ¿cómo podríamos

dibujar, por ejemplo, dos o más troncos? Cuando utilizamos glVertex3f, o llamamos a la función glSolidCube, estamos definiendo un objeto en coordenadas del objeto.

Por lo tanto para la primera escena se recreará el entorno o escenario que será el mismo a lo largo de todo el video juego, este tendrá diversos obstáculos los cuales consisten principalmente en troncos y charcos, si el jugador tropieza con uno de estos el juego habrá finalizado además n la pantalla se mostrara un marcador en el cual se llevara el puntaje de la partida, esto determinara la velocidad del juego ya que cada cierta distancia la velocidad ira en aumentando.

2.1.4.5 Desarrollo del proyecto escenas

```
//Funcion que dibuja el camino
void camino(void)
{
  //Definicion de texturas
  texture[0] = SOIL_load_OGL_texture // load an image file directly as a new OpenGL texture
  (
      "arena_2.jpg",
      SOIL_LOAD_AUTO,
      SOIL_CREATE_NEW_ID,
      SOIL_FLAG_MIPMAPS | SOIL_FLAG_INVERT_Y | SOIL_FLAG_NTSC_SAFE_RGB | SOIL_FLAG_COMPRESS_TO_DXT
  );
  glEnable(GL_TEXTURE_2D);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
  // allocate a texture name
  glBindTexture(GL_TEXTURE_2D, texture[0]);
  glBegin(GL_QUADS);
  glTexCoord2f(1,1000);
  glVertex3f(-1.0f,0.01f,-500.0f);
  glTexCoord2f(0,1000);
  glVertex3f(-1.0f,0.01f,500.0f);
  glTexCoord2f(1,0);
  glVertex3f( 1.0f,0.01f,500.0f);
  glTexCoord2f(0,0);
  glVertex3f( 1.0f,0.01f,-500.0f);
  glEnd();
}
```

Función que dibuja el camino definiendo textura cargando una imagen, en la cual se define textura en 2D, asignando sus respectivas coordenadas. [1]

[1] ¿Qué es una textura en OpenGL? Arrays uni, bi o tri-dimensionales respectivamente, que se le asignarán mediante una dirección de memoria.

```
//Funcion que dibuja el cesped
void cesped(void)
{
  //Definicion de texturas
  texture[0] = SOIL_load_OGL_texture // load an image file directly as a new OpenGL texture
  (
      "cesped1.jpeg",
      SOIL_LOAD_AUTO,
      SOIL_CREATE_NEW_ID,
      SOIL_FLAG_MIPMAPS | SOIL_FLAG_INVERT_Y | SOIL_FLAG_NTSC_SAFE_RGB | SOIL_FLAG_COMPRESS_TO_DXT
  );
  glEnable(GL_TEXTURE_2D);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
  // allocate a texture name
  glBindTexture(GL_TEXTURE_2D, texture[0]);
  glBegin(GL_QUADS);
  glTexCoord2f(1000,1000);
  glVertex3f(-50.0f,0.0f,-500.0f);
  glTexCoord2f(0,1000);
  glVertex3f(-50.0f,0.0f,500.0f);
  glTexCoord2f(1000,0);
  glVertex3f( 50.0f,0.0f,500.0f);
  glTexCoord2f(0,0);
  glVertex3f( 50.0f,0.0f,-500.0f);
  glEnd();
}
```

Función que dibuja el césped de la animación definiendo con textura, la cual obtiene las texturas en 2D y sus respectivas coordenadas, utilizando las primitivas de GLQuads[2]

Función que dibuja el cubo

```
void cubo(void)
{
  //Definicion de texturas
  texture[1] = SOIL_load_OGL_texture // load an image file directly as a new OpenGL texture
  (
      "lados.jpg",
      SOIL_LOAD_AUTO,
      SOIL_CREATE_NEW_ID,
      SOIL_FLAG_MIPMAPS | SOIL_FLAG_INVERT_Y | SOIL_FLAG_NTSC_SAFE_RGB | SOIL_FLAG_COMPRESS_TO_DXT
  );
// Frente
glEnable(GL_TEXTURE_2D);
glBindTexture(GL_TEXTURE_2D, texture[1]);
glBegin(GL_POLYGON);

glNormal3f( 0.0f, 0.0f, 1.0f);
glTexCoord2f(0.0f, 0.0f); glVertex3f(-0.5f, -0.5f, 0.5f);
glTexCoord2f(1.0f, 0.0f); glVertex3f( 0.5f, -0.5f, 0.5f);
glTexCoord2f(1.0f, 1.0f); glVertex3f( 0.5f, 0.5f, 0.5f);
glTexCoord2f(0.0f, 1.0f); glVertex3f(-0.5f, 0.5f, 0.5f);
glEnd();

// parte de Atras
glEnable(GL_TEXTURE_2D);
glBindTexture(GL_TEXTURE_2D, texture[1]);
glBegin(GL_POLYGON);
glNormal3f( 0.0f, 0.0f,-1.0f);
glTexCoord2f(1.0f, 0.0f); glVertex3f(-0.5f, -0.5f, -0.5f);
glTexCoord2f(1.0f, 1.0f); glVertex3f(-0.5f, 0.5f, -0.5f);
glTexCoord2f(0.0f, 1.0f); glVertex3f( 0.5f, 0.5f, -0.5f);
glTexCoord2f(0.0f, 0.0f); glVertex3f( 0.5f, -0.5f, -0.5f);
glEnd();
```

[2] ¿Qué es GLQuads? Trata cada grupo de cuatro vértices como un cuadrilátero independiente. Se dibujan 4 cuadriláteros.

```
95  // Arriba
96  glEnable(GL_TEXTURE_2D);
97  glBindTexture(GL_TEXTURE_2D, texture[1]);
98  glBegin(GL_POLYGON);
99  glNormal3f( 0.0f, 1.0f, 0.0f);
100 glTexCoord2f(0.0f, 1.0f); glVertex3f(-0.5f, 0.5f, -0.5f);
101 glTexCoord2f(0.0f, 0.0f); glVertex3f(-0.5f, 0.5f, 0.5f);
102 glTexCoord2f(1.0f, 0.0f); glVertex3f( 0.5f, 0.5f, 0.5f);
103 glTexCoord2f(1.0f, 1.0f); glVertex3f( 0.5f, 0.5f, -0.5f);
104 glEnd();
105
106 // Abajo
107 glEnable(GL_TEXTURE_2D);
108 glBindTexture(GL_TEXTURE_2D, texture[1]);
109 glBegin(GL_POLYGON);
110 glNormal3f( 0.0f,-1.0f, 0.0f);
111 glTexCoord2f(1.0f, 1.0f); glVertex3f(-0.5f, -0.5f, -0.5f);
112 glTexCoord2f(0.0f, 1.0f); glVertex3f( 0.5f, -0.5f, -0.5f);
113 glTexCoord2f(0.0f, 0.0f); glVertex3f( 0.5f, -0.5f, 0.5f);
114 glTexCoord2f(1.0f, 0.0f); glVertex3f(-0.5f, -0.5f, 0.5f);
115 glEnd();
116
117 // Lado Derecho
118 glEnable(GL_TEXTURE_2D);
119 glBindTexture(GL_TEXTURE_2D, texture[1]);
120 glBegin(GL_POLYGON);
121 glNormal3f( 1.0f, 0.0f, 0.0f);
122 glTexCoord2f(1.0f, 0.0f); glVertex3f( 0.0f, -0.5f, -0.5f);
123 glTexCoord2f(1.0f, 1.0f); glVertex3f( 0.0f, 0.5f, -0.5f);
124 glTexCoord2f(0.0f, 1.0f); glVertex3f( 0.0f, 0.5f, 0.5f);
125 glTexCoord2f(0.0f, 0.0f); glVertex3f( 0.0f, -0.5f, 0.5f);
126 glEnd();
```

```
127
128 // Lado Izquierdo
129 glEnable(GL_TEXTURE_2D);
130 glBindTexture(GL_TEXTURE_2D, texture[1]);
131 glBegin(GL_POLYGON);
132 glNormal3f(-1.0f, 0.0f, 0.0f);
133 glTexCoord2f(0.0f, 0.0f); glVertex3f(-0.5f, -0.5f, -0.5f);
134 glTexCoord2f(1.0f, 0.0f); glVertex3f(-0.5f, -0.5f, 0.5f);
135 glTexCoord2f(1.0f, 1.0f); glVertex3f(-0.5f, 0.5f, 0.5f);
136 glTexCoord2f(0.0f, 1.0f); glVertex3f(-0.5f, 0.5f, -0.5f);
137 glEnd();
138
139   glDisable(GL_TEXTURE_2D);
140   //glFlush();
141   glutSwapBuffers();
142 }
```

Definimos la función cubo como una función a la cual se le pasa textura a los lados, se le asignan texturas en 2D normalizando las esquinas con GLNormal[3], con sus respectivas coordenadas para redibujar los diferentes lados utilizando GLBeginPolygons.

[3] ¿Qué es GLNormal3f? La corriente normal se ajusta a las coordenadas dadas siempre que se emita GLNormal.

Archivo main.cpp, donde se cargaran las funciones para el videojuego.

```
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glut.h>
#include <SOIL/SOIL.h>
#include <stdlib.h>
#include <unistd.h>
#include <math.h>
#include "funciones.cpp"


// variables que representan direccion de la camara
float lx=0.0f,lz=-1.0f;
// XZ posicion de la camara
float x=0.0f,z=5.0f,zz=0;

// angulo de rotaciÃ³n
float angulo = 0.0f;
int frameNumber;

```

Incluimos las librerías esenciales para poder crear el videojuego “the cube” en las cuales se incluyen las librerías SOIL, GLGLUT para el manejo de OpenGL, Math para operaciones matemáticas e incluimos el archivo funciones.cpp donde se carga las funciones como el césped así mismo se incluyen las variables que representan las direcciones de la cámara y los ángulos de rotación.

```
void camara(int w, int h)
{
// Evitar una divisiÃ³n por cero, cuando la ventana es demasiado pequeÃ±a
// (No permitir una ventana de alto 0).
if (h == 0)
h = 1;
float proporcion = w * 1.0 / h;

// Usando Matrix proyecciÃ³n
glMatrixMode(GL_PROJECTION);

// Reseteando la Matrix
glLoadIdentity();

// configurar el viewport para una ventana completa
glViewport(0, 0, w, h);

// Configurando las perspectivas.
gluPerspective(45.0f, proporcion, 0.1f, 100.0f);

// Regresar la vista a Modelview
glMatrixMode(GL_MODELVIEW);
}
long int vaa = 100000000000000000;
```

La función cámara se le asigna valores de tipo entero w y h, en la cual se hace la proporción para el width (ancho) y height (alto) con el cual se utiliza una matriz de proyección, posteriormente se resetea esa matriz y se configura el ViewPort para una ventana completa, luego se configura las perspectivas con una misma proporción a la ventana para después regresar a la vista de modelo.

```
44  void display(void)
45  {
46    glClearColor(0.0, 0.0, 1.0, 0.5);
47    // Borrar Color and Depth Buffers
48    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
49    // Reseteando transformaciones
50    glLoadIdentity();
51    // Configurando camara
52    gluLookAt(0.0, 2.0f,500+(-zz)/100000000, lx, 1.0f, 490+lz/100000000,  0.0f, 1.0f, 0.0f);
53    cesped();
54    camino();
55    glTranslatef(x , 0.0f, 495+(-lz)/-vaa);
56    glScalef(0.6,0.9,0.6);
57    cubo();
58    //
59    glPushMatrix();
60    //glPushMatrix();
61
62  glutSwapBuffers();
63  }
64
65  void salir(unsigned char key, int x, int y)
66  {
67    if (key == 27)
68    exit(0);
69  }
```

En la función Display lo que se hará es cargar la escena que se reproducirá, se define un color (GLClear) después se borra el color con DEPTHBUFFER y se resetean las primeras transformaciones, se configura la cámara con el gluLookAt, se incluye la función césped, luego camino, posteriormente se aplica glTranslatef para verlo de una diferente perspectiva y se hace una escala para reducir. Finalmente se agrega el cubo que es el que recorrerá la animación.

```
71  void keyboard(int key, int xx, int yy)
72  {
73    float fraccion = 0.1f;
74    switch (key)
75    {
76      case GLUT_KEY_LEFT :
77        //lx = -0.5;
78        x = -0.5;
79      break;
80      case GLUT_KEY_RIGHT :
81        //lx = 0.5;
82        x = 0.5;
83      break;
84        case GLUT_KEY_UP :
85        z += lz * fraccion;
86      break;
87        case GLUT_KEY_DOWN :
88        z -= lz * fraccion;
89      break;
90      }
91  }
```

La función Keyboard se utiliza para que el cubo se desplace de izquierda a derecha, de arriba abajo, según sea la tecla que el usuario presione.

```
// --------------- Para animaciÃ³n ----------------------------------------

int animating = 0;      // 0 sin animaciÃ³n
                        // Se cambia con la llamada a las funciones startAnimation() and pauseAnimation()
void pauseAnimation()
{
// Llamamo a la funciÃ³n para detener la animaciÃ³n
   animating = 0;
}

void updateFrame()
{
// En esta funcion agregamos el codigo que hara la secuencia de cada escena como si fuera un fotograma

 //glColor3f(0.0,1.0,0.0);
 //Hacemos que la tetera gire
 for (float i=0; i<=500; i+=0.1)
 {
  zz=zz-=i;

 }
  //Verificamos el numero de frames para detener animaciÃ³n
  if(frameNumber==200)
   {
     pauseAnimation();
     frameNumber=0;
   }
 //Almacenamos el numero de frames
 frameNumber++;
//printf ("Numero de Frame: %d \n", frameNumber);
}
```

Creamos las animaciones, primero definimos una variable de tipo entera con valor cero (O sea sin animación), después proseguimos creando la función pauseAnimation, donde llamamos a la antes definida variable para detener la animación, prosiguiendo definimos la función updateFrame, en la cual en esta función se define el código que hará de secuencia de cada escena como que si fuera un fotograma, después se crea un ciclo FOR para replicar el fotograma y se hace una excepción utilizando una restricción que si el número de FRAMES = 200, este detendrá la animación y por ultimo almacenamos el número de FRAMES.

```
void timerFunction(int timerID)
{
// Invocamos la funcion para controlar el tiempo de la ejecucion de funciones
//sleep(6);
   if (animating)
   {
      updateFrame();
      glutTimerFunc(30, timerFunction, 0);
      glutPostRedisplay();
   }
}

void startAnimation()
{
    // llamamos la funciÃ³n para iniciar la animaciÃ³n
  if ( ! animating ) {
     animating = 1;
     glutTimerFunc(30, timerFunction, 1);
  }
}

```

En esta parte del código, cargaremos todas inicializaciones de glut, como posición de ventana, tamaño de ventana, nombre de la ventana, además de agregar las funciones para que se carguen en la ventana, como la DISPLAY que muestra las escenas, CAMARA que sigue la posición del objeto, SALIR para salir de la ventana, KEYBOARD para el movimiento, además de incluir las texturas, shadelModel y la inicialización de la animación.

Nuevas funciones agregadas: obstáculos y árboles.

```
//textura del arbol
void arbol(void)
{
  texture[0] = SOIL_load_OGL_texture // load an image file directly as a new OpenGL texture
  (
      "hojas1.jpg",
      SOIL_LOAD_AUTO,
      SOIL_CREATE_NEW_ID,
      SOIL_FLAG_MIPMAPS | SOIL_FLAG_INVERT_Y | SOIL_FLAG_NTSC_SAFE_RGB | SOIL_FLAG_COMPRESS_TO_DXT
  );
  glEnable(GL_TEXTURE_2D);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);

  texture[1] = SOIL_load_OGL_texture // load an image file directly as a new OpenGL texture
  (
      "arbol.jpg",
      SOIL_LOAD_AUTO,
      SOIL_CREATE_NEW_ID,
      SOIL_FLAG_MIPMAPS | SOIL_FLAG_INVERT_Y | SOIL_FLAG_NTSC_SAFE_RGB | SOIL_FLAG_COMPRESS_TO_DXT
  );
  glEnable(GL_TEXTURE_2D);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
  //Arbol#1
  float a=700.0;
  for (int i=-10; i<200; i++)
  {
    if(a>-500)
    {
    glBindTexture(GL_TEXTURE_2D, texture[0]);
    glBegin(GL_TRIANGLES);
```

```
      glBegin(GL_TRIANGLES);
        glTexCoord2f(1,1);glVertex3f( 4.0f, 1.7f, a );
        glTexCoord2f(0,1);glVertex3f( 6.0f, 1.7f, a );
        glTexCoord2f(1,0);glVertex3f( 5.0f, 4.0f, a );
      glEnd();
      glBindTexture(GL_TEXTURE_2D, texture[1]);
      glBegin(GL_QUADS);
        glTexCoord2f(1,1);glVertex3f( 4.8f, 0.0f, a );
        glTexCoord2f(0,1);glVertex3f( 4.8f, 1.0f, a );
        glTexCoord2f(1,0);glVertex3f( 5.2f, 1.0f, a );
        glTexCoord2f(0,0);glVertex3f( 5.2f, 0.0f, a );
      glEnd();
    }
    if(a>-500)
    {
      glBindTexture(GL_TEXTURE_2D, texture[0]);
      glBegin(GL_TRIANGLES);
        glTexCoord2f(1,1);glVertex3f( -4.0f, 1.0f, a );
        glTexCoord2f(0,1);glVertex3f( -6.0f, 1.0f, a );
        glTexCoord2f(1,0);glVertex3f( -5.0f, 3.0f, a );
      glEnd();
      glBegin(GL_TRIANGLES);
        glTexCoord2f(1,1);glVertex3f( -4.0f, 1.7f, a );
        glTexCoord2f(0,1);glVertex3f( -6.0f, 1.7f, a );
        glTexCoord2f(1,0);glVertex3f( -5.0f, 4.0f, a );
      glEnd();
      glBindTexture(GL_TEXTURE_2D, texture[1]);
      glBegin(GL_QUADS);
        glTexCoord2f(1,1);glVertex3f( -4.8f, 0.0f, a );
        glTexCoord2f(0,1);glVertex3f( -4.8f, 1.0f, a );
        glTexCoord2f(1,0);glVertex3f( -5.2f, 1.0f, a );
        glTexCoord2f(0,0);glVertex3f( -5.2f, 0.0f, a );
      glEnd();
    }
    a-=5.0;
    }
  }
```

```
void opstaculos(void)
{
  float a = 900.0;
  int par = 1;
  glColor3f(1, 0, 0);
  for (int i=0; i<200; i++)
  {
    if(a>-5)
    {
        glBegin(GL_POLYGON);
        glVertex3f( -1.0*par,-0.2f, a);
        glVertex3f(  0.0*par,-0.2f, a);
        glVertex3f(  0.0*par, 0.2f, a);
        glVertex3f( -1.0*par, 0.2f, a);
        glEnd();
        glBegin(GL_POLYGON);
        glVertex3f( -1.0*par, 0.2f, a+0.5);
        glVertex3f(  0.0*par, 0.2f, a+0.5);
        glVertex3f(  0.0*par, 0.2f, a);
        glVertex3f( -1.0*par, 0.2f, a);
        glEnd();
        glBegin(GL_POLYGON);
        glVertex3f( -1.0*par,-0.2f, a+0.5);
        glVertex3f(  0.0*par,-0.2f, a+0.5);
        glVertex3f(  0.0*par, 0.2f, a+0.5);
        glVertex3f( -1.0*par, 0.2f, a+0.5);
        glEnd();
    }
    par*=-1;
    a-=5.0;
  }
  glColor3f(1.0,1.0,1.0);
}
```

Función obstáculo: Esta función genera aleatoriamente obstáculos a lo largo del camino

funcion_arbol1 y 1: Esta función genera arboles a los costados, o sea dentro de la función césped, utilizando texturas para poder generar los árboles en panorama bidimensional.

Actualizaciones finales.

```
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glut.h>
#include <SOIL/SOIL.h>
#include <SDL/SDL.h>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <math.h>
#include "funciones.cpp"
#define MUS_PATH "audio.wav"
using namespace std;

//Prototipo de nuestra devoluciÃ³n de llamada de audio
// Ver la implementaciÃ³n para mÃ¡s informaciÃ³n
void my_audio_callback(void *userdata, Uint8 *stream, int len);

// Declaraciones de variables
static Uint8 *audio_pos;//Puntero global al bÃºfer de audio que se va a reproducir
static Uint64 audio_len; // Longitud restante de la muestra que tenemos que jugar
```

Incluimos nuevas librerías [4]para que el videojuego pueda desempeñarse de una mejor manera, por ejemplo librerías para insertar sonidos dentro del videojuego, además de definir el archivo a reproducir.

[4] ¿Qué es la librería SDL? SDL es una librería o biblioteca de desarrollo multiplataforma que fue diseñado para proporcionar acceso de bajo nivel a todos los dispositivos de nuestra computadora, es decir, audio, teclado, ratón, joystick, y los gráficos de hardware a través del conocido OpenGL o Direct3D.

```
214  int main(int argc, char **argv)
215  {
216
217    glutInit(&argc, argv);
218    glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGBA);
219    glutInitWindowPosition(300,100);
220    glutInitWindowSize(800,600);
221    glutCreateWindow("The Cube");
222    // sonido
223      // Inicializar SDL.
224      if (SDL_Init(SDL_INIT_AUDIO) < 0)
225            return 1;
226
227      // Variables locales
228      static Uint32 wav_length; // Longitud de nuestra muestra
229      static Uint8 *wav_buffer; // Buffer que contiene nuestro archivo de audio
230      static SDL_AudioSpec wav_spec; // Las especificaciones de nuestra pieza de mÃºsica
231
232      /* Cargar el WAV */
233      // Las especificaciones, la longitud y el bÃºfer de nuestro wav se llenan
234      if( SDL_LoadWAV(MUS_PATH, &wav_spec, &wav_buffer, &wav_length) == NULL )
235      {
236        return 1;
237      }
238      // Establecer la funciÃ³n de devoluciÃ³n de llamada
239      wav_spec.callback = my_audio_callback;
240      wav_spec.userdata = NULL;
241
242      // Establecer nuestras variables estÃ¡ticas globales
243      audio_pos = wav_buffer ; // Copia el buffer de sonido
244      audio_len = wav_length; // Copia la longitud del archivo
245
```

En la función main definimos todas las dependencias de Glut (tamaño de la ventana, posición en el pixel, nombre de la ventana) se define el sonido, se inicializa el SDL, definición de variables locales, buffers que contienen archivos de audio y especificaciones de la pieza de música.

```
246      /*Abrir el dispositivo de audio */
247      if ( SDL_OpenAudio(&wav_spec, NULL) < 0 )
248      {
249        fprintf(stderr, "No se pudo abrir el audio: %s\n", SDL_GetError());
250        exit(-1);
251      }
252
253      /* Empezar a jugar */
254      SDL_PauseAudio(0);
255
256      // Esperar hasta que no jugamos
257      while ( audio_len > 0 )
258      {
259          SDL_Delay(10);
260          // register callbacks
261    glutDisplayFunc(display);
262    glutReshapeFunc(camara);
263    glutIdleFunc(display);
264    glutKeyboardFunc(salir);
265    glutSpecialFunc(keyboard);
266    // OpenGL init
267    glEnable(GL_DEPTH_TEST);
268    //Para texturas
269    glEnable(GL_TEXTURE_2D);
270    glShadeModel(GL_SMOOTH);
271    glDepthFunc(GL_LEQUAL);
272    glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);
273    glutMainLoop();
274    return 1;
275      }
```

Se carga el wav con las especificaciones como la longitud y el buffer que se va llenando, luego retorna y se establece la función que regresara la llamada, se establece las variables estáticas y globales, copias del buffer de sonido y copia de la longitud del archivo de sonido.

```
    // Cerrar todo
    SDL_CloseAudio();
    SDL_FreeWAV(wav_buffer);
}
//FunciÃ³n de devoluciÃ³n de llamada de audio
//AquÃ- tiene que copiar los datos de su buffer de audio en el
// Solicitud de bÃºfer de audio (secuencia)
//SÃ³lo debe copiar tanto como la longitud solicitada (len)
void my_audio_callback(void *userdata, Uint8 *stream, int len)
{

    if (audio_len ==0)
        return;

    len = ( len > audio_len ? audio_len : len );
    //SDL_memcpy (stream, audio_pos, len);                  // Simplemente copie desde un buffer en el otro
    SDL_MixAudio(stream, audio_pos, len, SDL_MIX_MAXVOLUME);// Mezclar de un buffer a otro

    audio_pos += len ;
    audio_len -= len ;
}
```

Abrimos el dispositivo de audio en un ciclo en que si no se cumple arroje un mensaje que no se pudo abrir el archivo de audio, si se empieza a jugar hay una pausa, pero al iniciar el videojuego al entrar en marcha se ingresa en un ciclo en el cual el sonido empieza a reproducirse, se define las funciones donde se muestra el juego (display) la cámara con la que se sigue la actividad del cubo, la función para salir y se define la extura en dos dimensiones, se suavizan los bordes con GLsmooth.

```
//funcion para las texturas del texto
void text(float P)
{
  char text[32];
  if(P == -1.0)
    {
        sprintf(text, "Perdiste");
        frameNumber=0;
    }

  else if(P== -2.0)
    {
        sprintf(text, "Ganaste");
    }
  else
    {
      sprintf(text, "Puntuacion: %.0f", P);
    }
    //definiendo color, tipo y tamaÃ±o de letra
    glColor3f(1, 1, 1);
    glRasterPos3f( 0.8 , 2.0 , 0.1);
    for(int i = 0; text[i] != '\0'; i++)
    glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, text[i]);
    //glColor3f(1, 1, 1);
}
```

Incluimos la función Text, en esta se muestra el puntaje y además el estado del juego si se ganó o perdió, después se define el color de la letra, el tipo de letra y su respectivo tamaño.

2.1.5 Implicaciones políticas y características

2.1.5.1 Introducción

The Cube (Videojuego Desktop), respeta la privacidad de todos los usuarios de sus ordenadores y garantiza que la información personal no sea dañada o perdida y está será tratada como confidencial. En este sentido cumplimos con los requisitos de la Ley de Telecomunicaciones de los Países Latinoamericanos y la Ley de Protección de Datos Personales de El Salvador.

¡Atención! Si aún no tienes 16 años, primero debes preguntar a tus padres o representante legal; para obtener permiso para proporcionarte la herramienta tecnológica.

2.1.5.2 Modificaciones en la declaración de privacidad

Los desarrolladores del videojuego “The Cube” podrían modificar esta declaración. Por lo tanto, se recomienda leer esta declaración con regularidad, de modo que usted sea consciente de estos cambios.

2.1.5.3 Acceso y modificación del código del videojuego

Los desarrolladores tendrán el privilegio de modificar y/o eliminar datos u modificar el código base del videojuego “The Cube”.

En casos externos de querer instalar o modificar el código base, personas ajenas al grupo de desarrollo del videojuego, podrán hacer en todo momento, toda clase de preguntas sobre nuestras políticas privacidad primero; relacionadas con el acceso y modificaciones. En la política de venta del videojuego, para instituciones fuera del recinto universitario, se les solicitara ponerse en contacto con los desarrolladores u remitirse al siguiente formulario de contacto:

Desarrolladores:

- © Bach. Raúl Portillo Muñoz, correo: raul-mauricio@live.com
- © Bach. Fredy Turcios Reyes, correo: turcios095@gmail.com
- © Bach. Ellen Coreas Pérez, correo: soyellen_26@hotmail.com
- © Bach. Claudia Salamanca Martínez, correo: salamanca029cp@hotmail.com

Email: publicaciones_UES@gmail.com

3. Aspectos administrativos

3.1 Recurso humano

Las personas que participan, contribuyen o colaboran en un proyecto de investigación son el elemento activo que garantiza el éxito de los objetivos y de los resultados de la Actividad de Investigación y Desarrollo.

Debemos dedicar mucha atención a la adecuada clasificación de las personas participantes en una Actividad de Investigación y Desarrollo, lógicamente que este prerrequisito exige saber qué clases de personas se requieren en un proyecto de investigación. A la pregunta ¿Qué Recursos Humanos personales requerimos en un proceso investigativo? Podemos responder dando la siguiente clasificación y caracterización:

Diseñador, Escenas, Animador, Programador.

El Diseñador

Es la persona encargada del diseño gráfico del juego, es quien da forma a la idea llevando la representación de algo que ya se puede presentar a las personas ajenas al proyecto y que puedan comprender de lo que se trata.

Escena

Esta persona es la encargada de convertir en realidad el diseño, llevando los dibujos hechos por el diseñador al escenario mediante las herramientas antes mencionadas.

Animador

Encargado de animar las escenas creadas anteriormente para que ya exista interacción con el jugador y los objetos del juego, y que estos tengan los movimientos planeados.

El programador

Es quien se encarga de programar las condiciones del juego, como lo que se puede y no hacer a, alguna figura, límites de movimiento, colisiones, etc.

3.2 Presupuesto

Componente	Cantidad (mes)	Costo unitario	Costo total
Programador	192h	$8.50	$1632
Diseñador	120h	$10.00	$1200
Animador	192h	$10.00	$1920
Escena	120h	$8.50	$1020
Energía eléctrica	30.2kw/h	$0.20	$6.04
Viáticos			$143.52

3.3 Cronograma

Tarea	Mayo				Junio			
	1	2	3	4	1	2	3	4
Planteamiento de las ideas								
Selección de la idea								
Creación del documento								
Diseño de las Escenas								
Creación de las Escenas								
Animación de los Objetos								
Programación del juego								
Prototipo de bajo nivel								
Prototipo funcional								
Lanzamiento								

4. Referencias

Cozzi, P. (3 de agosto de 2015). *Webgl Insights.* A K Peters Ltd; Edición: Har/Psc (3 de agosto de 2015).

Haverbeke., M. (Diciembre de 2014). *Eloquent JavaScript: A Modern Introduction to Programming.* EE.UU.